Improving DRL Of Vision-Based Navigation By Stereo Image Prediction

Thesis Report

AE5310: Thesis Control and Operations

Luc den Ridder

Delft University of Technology

TU Delft

Improving DRL Of Vision-Based Navigation By Stereo Image Prediction

Thesis Report

by

Luc den Ridder

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday June 20, 2023 at 9:30 AM.

Student number:	4543165	
Project duration:	July 12, 2022 – June 20, 2023	
Thesis committee:	Prof. dr. ir. G. C. H. E. de Croon,	TU Delft, supervisor
	Dr. Ir. C. de Wagter	TU Delft
	Dr. Ir. B. F. Lopes Dos Santos,	TU Delft
	Ir. Y. Wu,	TU Delft

Cover:	Flying Drone by Josue Bautista Garcia on Pexels (Modified)
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ACER	Actor-Critic with Experience Replay
ACKTR	Actor-Critic with Kronecker Factored Trust Region
Adam	Adaptive Moment Estimation
BN	BatchNorm
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
DCNN	Deep Convolutional Neural Network
DNN	Deep Neural Network
DPG	Deterministic Policy Gradient
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DRNN	Deep Recurrent Neural Network
ELU	Exponential Linear Unit
FID	Frechet Inception Distance
FCN	Fully Convolutional Network
GAN	Generative Adversarial Network
GELU	Gaussian Error Linear Unit
GPS	Global Positioning System
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
IMU	Inertial Measurement Unit
LIDAR	Light Detection and Ranging
LDI	Layered Depth Image
LN	LayerNorm
LPIPS	Learned Perceptual Image Patch Similarity
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MDP	Markov Decision Process
MLP	Multilayer Perceptron
MPI	Multiplane Image
MSA	Multi-head Self-Attention
MSE	Mean Squared Error
MS-SSIM	Multi-Scale SSIM
NLP	Natural Language Processing
NeRF	Neural Radiance Fields
NVS	Novel View Synthesis
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
PSRN	Peak Signal-to-Noise Ratio
RGB	Red, Green and Blue
RGB-D	Red, Green, Blue and Depth

Abbreviation	Definition
RL	Reinforcement Learning
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
ROS	Robot Operating Systems
SAC	Spft Actor-Critic
SGD	Stochastic Gradient Descent
SSIM	Structural Similarity Index Measure
Swin	Shifted Windows
SW-MSA	Multi-head Self-Attention in Shifted Windows
Tanh	Tangent Hyperbolic
TD3	Twin Delayed Deep Deterministic
TRPO	Trust Region Policy Optimization
TPU	Tensor Processing Unit
UAV	Unmanned Aerial Vehicle
VAE	Variational Autoencoder
VQ-GAN	Vector Quantized Generative Adversarial Networks
VQ-VAE	Vector Quantized Variational Autoencoder
ViT	Vision Transformer
W-MSA	Multi-Head Self-Attention in Windows

Symbols

Symbol	Definition
a	Action
$\mathcal{A}$	Action Space
$\hat{B}$	Bias Matrix
C	Number of Channels
d	Depth
d_k	Key Dimension
d_v	Value Dimension
D	Disparity
E	Embedded Patch
E	Error
$\mathbb{E}$	Expected Value
h	Interaction History
h	Height
H	Height
I	Image
J	Reward Function
K	Key
k_1	Constant
k_2	Constant
L	Dynamic Range of Pixel Values
M	Million
M^2	Number of patches in window
N	Total number of Images
o	Observation
$\mathcal{O}$	Observation Space
P^2	Resolution of Image Patches
Q	Query
Q	Action-value
r	Real-Valued Reward

Symbol	Definition
R	Expected Discounted Cumulative Reward
$\mathbb{R}$	Set of Real Numbers
t	Time
s	State
$\mathcal{S}$	State Space
$\mathcal{U}$	Uniform Distribution
v	velocity
V	Value
V	State-Value
w	Width
W	Width
x	Image
x	Coordinate on X-Axis
x_p	Image Patches
y	Target Image
y	Coordinate on Y-Axis
$y*$	Predicted Image
$\hat{y}$	Predicted Image
z	Coordinate on Z-Axis
β	Behaviour policy
δ	intensity
μ	Mean
γ	Discount Factor
σ	Standard Deviation
π	Policy
ϕ_{π_θ}	State Visitation Probability
θ	Target Policy

Preface

Welcome to my master's thesis: "Improving Deep Reinforcement Learning Of Vision-Based Navigation By Stereo Image Prediction." It's a fascinating topic, blending artificial intelligence and computer vision to make a real difference in the world of autonomous systems.

I have many people to thank for their help in bringing this work to life. First and foremost, Yilun Wu, my supervisor, who was always ready with his sharp remarks and expert guidance. His assistance has been invaluable, and this thesis wouldn't be what it is without him.

My responsible supervisor, Guido de Croon, also deserves special recognition. Though our meetings were on a biweekly basis, his insights and advice significantly influenced my work.

The computational resources for this research were indispensable. I extend my gratitude to Yilun for the use of his personal server, and Jesse Hagenaars, who managed another server that was integral to my research. I'm grateful for their support.

In addition, Hang Yu developed the AvoidBench code that was integral to my research. His help in implementing it made a big difference to my work.

My family and friends have been a steady source of support throughout this journey, and for that, I am grateful. Special thanks to my friends who shared the study room 2.56 with me. Your company made the long hours of work enjoyable and worthwhile.

In the pages that follow, I present my research on enhancing autonomous systems through stereo image prediction. I hope you find it as interesting to read as I found it to conduct.

Lastly, I want to extend my heartfelt thanks to everyone who's been a part of this journey. I've learned so much along the way, and I'm excited to see where these advancements in deep reinforcement learning lead us.

Luc den Ridder
Delft, June 2023

Abstract

Although deep reinforcement learning (DRL) is a highly promising approach to learning robotic vision-based control, it is plagued by long training times. This report introduces a DRL setup that relies on self-supervised learning for extracting depth information valuable for navigation. Specifically, a literature study is conducted to investigate the effects of learning how to synthesize one view from the other in a stereo-vision setup without relying on any preliminary knowledge of the camera extrinisics and how it can be integrated for its downstream use for an obstacle avoidance task. As such, the literature study concludes that competitive geometry-free monocular-to-stereo image view synthesis is feasible due to recent developments in computer vision. The scientific paper further develops concepts proposed in the literature study and benchmarks the proposed architectures on depth estimation benchmarks for KITTI. Competitive results are achieved for view synthesis and despite sub-optimal performance compared to state-of-the-art monocular depth estimation, an ability to encode depth and detect shapes is present and, therefore, satisfactory for the application to DRL. Additionally, the research examines the benefits of using the latent space of a view synthesis architecture compared to other feature extractor methods as an input to the PPO agent implemented as auxiliary tasks. This method achieves quicker convergence and better performance for an obstacle avoidance task in a simulated indoor environment than the autoencoding feature extractor and end-to-end DRL methods. It is only outperformed by the monocular depth estimation feature extractor method. Overall, this research provides valuable insights for developing more efficient and effective DRL methods for monocular camera-based drones. Finally, the complementary code for this research can be found: https://github.com/ldenridder/drl-obstacle-avoidance-view-synthesis.

List of Figures

List of Tables

1 Introduction

With computing power becoming cheaper and neural networks becoming ever more powerful, they get more instrumental in aiding research to improve quality of life and reduce the costs of products and services. The research presented in this report is such a tale, showing the versatility of deep reinforcement learning agents trained on stereo vision applied to monocular vision-based drones, reducing the need for expensive and heavy depth sensing technologies.

1.1. Background

LIDAR sensors use laser beams to create a three-dimensional representation of the surveyed environment. In doing so, they create a depth map, which is essential for the navigation of autonomous drones. With the costs of batteries and processors going down, these LIDAR sensors are becoming more expensive relative to other parts. As such, a priority within the computer vision industry is to reduce dependency on these sensors. Cameras are, however, reasonably cheap, and as implied by its name, the computer vision domain tries to extract as much understanding from the cameras' output. Inspired by humans and other animals, this domain succeeded in estimating depth by using the disparities of matching pixels from two images taken by cameras parallel to each other, so-called stereo vision. Humans, however, rely not solely on the geometric method but also on monocular cues. Exclusively depending on monocular vision is sufficient for humans to perceive depth (Walk & Dodge, 1962). These cues include monocular parallax and pictorial cues based on the relation of objects within the frame and their location within the three-dimensional space. The geometry and size of things, where things vertically begin and end in the frame, and the horizon are the dominant cues learned. We have observed these relations while interacting with and growing up in our environment. As these features are thought through experience, a neural network should also be able to learn this. Indeed, there has been steady progress in the ability of neural networks to estimate depth with monocular vision. For instance, a supervised learning approach that accounts for both local and global image features was proposed (Saxena et al., 2005). A couple of years later, an architecture was developed using two deep networks, one of them to estimate the global depth structure and the other to refine the details locally (Eigen et al., 2014). As stated earlier, depth is geometrically related to stereo images. Consequently, this also means that once depth and a single image are known, the other image could be obtained through geometry, except for the occluded parts in the first image that are visible in the second image. Training a network on stereo cameras to predict one of the images from the other image could be helpful. Once a network can predict the second image with decent accuracy, it should understand the geometric relation and depth, which would be encoded in the latent space of the network architecture.

1.2. Problem statement

So for which cases is this specifically helpful? Monocular camera-based drones require few components, making them cheap and lightweight. Being able to navigate efficiently and avoid obstacles consistently is of significant relevance to the use case of such autonomous drones. One method to train these drones is with deep reinforcement learning (DRL). Within DRL, an agent receives rewards

based on the results of actions taken by the agent. The agent takes an action based on available information. In the case of a drone with a monocular camera, it is a sequence of RGB images. The most traditional setup for vision-based navigation is an end-to-end approach trained by relying solely on obstacle avoidance and path planning rewards. However, currently in most state-of-the-art methods (e.g. (Ha & Schmidhuber, 2018)), the images are encoded by a deep neural network so that only essential information is presented to the agent, and a sound decision follows regarding navigation. For instance, the encoder could be trained with autoencoding. (B. Zhou et al., 2019) showed the importance of feature extraction for the navigational performance of several tasks. Specifically intrinsic surface, color, optical flow, depth, and semantic segmentation. Furthermore, (Sax et al., 2018) concluded that adding other intermediate representations, such as curvature and denoising, enhances performance. These feature extractors often have to be trained with a different model on a dataset in a supervised manner and then finetuned to the reinforcement learning task. It is also possible to train the encoder as an auxiliary task, which runs parallel to the main task of the reinforcement learning network (X. Li et al., 2015). These auxiliary tasks can be trained unsupervised (Jaderberg et al., 2016) or self-supervised (Mirowski et al., 2016). The prediction network earlier described becomes relevant for this goal. Training the agent parallel on image synthesis could improve the quality of the encoded information that the agent receives for the main task. Besides the cost advantage of a stereo-camera-based drone over a drone with depth sensors, a second advantage is that the agent of a monocular-based drone could reason about the environment outside its field of view. It should be able to imagine what occluded regions could contain and understand the shape of objects better. This leads to the following research question encompassing both experimental phases and will be answered in the succeeding thesis report.

Research Question: How can geometry-free monocular-to-stereo image view synthesis be used as an auxiliary task to improve the navigation task performance and data efficiency of the reinforcement learning agent from a monocular vision-based drone?

1.3. Report Structure

This is the thesis report with the topic: Combining Stereo Image Prediction With Deep RL On Single Camera Drones. The content and structure of this report correspond to guidelines provided for 'Master Thesis AE'[1]. The body of the report is divided into two parts.
Part I contains the scientific paper presenting the main research contributions for this project. Structured as a stand-alone document, it can be read independently of the other material presented in this report. The most relevant literature to the research is summarized within the related work. The geometry-free monocular-to-stereo view synthesis architecture is first proposed, implemented, and results are analyzed. Afterward, a method is proposed to integrate this into the DRL-based drone navigation setup. Which is compared to other setups and in different environments. Finally, the main conclusions are drawn, and future work is suggested.
Part II contains the literature study preceding the scientific paper. The research is split similarly to the method and results sections in the scientific paper and reflects the two experimental phases of the research. The first experimental phase, Chapter 2, aims to prove that end-to-end geometry-free monocular-to-stereo image view synthesis is possible and achieves good performance compared to other monocular-to-stereo image view synthesis methods, both geometric and geometry-free. Besides, proof that depth is encoded within the network is provided. In the second experimental phase, Chapter 3, deep reinforcement learning for monocular vision-based drones is trained with stereo vision. This phase aims to improve the performance and data efficiency of navigational tasks. The second objective is to demonstrate the effectiveness of training the prediction network as an auxiliary task parallel to the navigational task. Chapter 4 provides a conclusion to the literature study.

[1]https://brightspace.tudelft.nl/d2l/home/43776 [Accessed 30 May 2023]

Part I

Scientific Paper

Improving Deep Reinforcement Learning Of Vision-Based Navigation By Stereo Image Prediction

Luc den Ridder
Micro Air Vehicle Laboratory
Faculty of Aerospace Engineering
Delft University of Technology
Delft, The Netherlands

Yilun Wu
Micro Air Vehicle Laboratory
Faculty of Aerospace Engineering
Delft University of Technology
Delft, The Netherlands

Guido de Croon
Micro Air Vehicle Laboratory
Faculty of Aerospace Engineering
Delft University of Technology
Delft, The Netherlands

***Abstract*—Although deep reinforcement learning (DRL) is a highly promising approach to learning robotic vision-based control, it is plagued by long training times. In this article, we introduce a DRL setup that relies on self-supervised learning for extracting environmental features such as depth information valuable for navigation. Specifically, we investigate the effects of learning how to synthesize one view from the other in a stereo-vision setup without relying on any preliminary knowledge of the camera extrinsics and its downstream use for an obstacle avoidance task. As far as the writers are aware, this is the first paper to leverage novel view synthesis for DRL, making a significant contribution to the field. First, the study evaluates the performance of the proposed architectures on view synthesis and depth estimation benchmarks for KITTI. Competitive performance is achieved for view synthesis. Although the performance is sub-optimal compared to the state-of-the-art for monocular depth estimation, an ability to encode depth and detect shapes is present and, therefore, satisfactory for the application to DRL. Furthermore, a simple stereo-matching algorithm was used to obtain the results, and the network does not directly optimize for depth prediction. Additionally, the research examines the benefits of using the latent space of a view synthesis architecture compared to other feature extractor methods as an input to the PPO agent implemented as auxiliary tasks. This method achieves quicker convergence and better performance for an obstacle avoidance task in a simulated indoor environment than the autoencoding feature extractor and end-to-end DRL methods. It is only outperformed by the monocular depth estimation feature extractor method. Overall, this research provides valuable insights for developing more efficient and effective DRL methods for monocular camera-based drones.**

***Index Terms*—autonomous navigation, UAV, drones, deep reinforcement learning, self-supervised learning, auxiliary tasks, monocular vision, depth estimation, feature extraction, simulation**

I. Introduction

As the weight, power usage, and cost of active depth sensors remain high, drones increasingly rely on cameras for autonomous navigation. Deep Reinforcement Learning (DRL) has shown promising results in training drones to navigate complex environments. However, extracting the best environmental features from camera images remains a crucial challenge.

The most traditional DRL setup for vision-based navigation is an end-to-end approach. In this setup, an image is directly fed to the network, which outputs the control actions. This setup is appealing as it involves minimal assumptions on the part of the human designer. However, it is disadvantageous that all the weights must be learned based on low-dimensional and potentially sparse rewards. This substantially increases sample complexity and hence training time.

For this reason, most state-of-the-art methods (e.g., [1, 2]) pre-process an incoming image to reduce the dimensionality before reinforcement learning. A popular setup is to feed the image to an autoencoder, trained to reconstruct the same image while passing through a much lower-dimensional "bottleneck" hidden layer. The activities of the hidden layer are then termed the "latent state" and used by DRL. This approach is often combined with additional neural structures that map the current latent state to the next one to learn to capture the robot's dynamics. Although this effectively reduces dimensionality, the autoencoder may not capture the most relevant features to autonomous navigation tasks.

Alternatively, vision tasks such as depth estimation or optical flow could be performed, with their dense outputs fed to a deep neural network for DRL. In Zhou et al. [3], it has been shown convincingly that such generic but task-relevant inputs benefit DRL performance. A downside of this method is that it requires fully convolutional networks for generating dense flow or depth inputs, which are then encoded again. To deal with this, an alternative approach is to learn depth estimation as an "auxiliary task" for the encoding part of the network [4]. The encoding part of the network is then bound to contain depth information valuable to the visual navigation task. This approach does raise the question of how optical flow or depth perception should be learned if there is no access to ground truth depth data in the test environment.

In this article, we introduce a novel DRL setup for monocular navigation in which the robot uses self-supervised learning (SSL) to extract task-relevant depth features. Specifically, the robot trains a deep network that maps each image to the other image in a stereo-vision setup. This view synthesis can

only be successful if the network extracts depth information, as the differences between the images of a stereo-vision setup are primarily due to the displacements caused by depth differences. In contrast to most of the SSL monocular depth literature, we focus on a geometry-free approach, i.e., without knowledge of camera extrinsics. Few have attempted view synthesis with geometry-free architectures, as the performance is considered worse compared to architectures that include warping functions [5] or more extensive scene representations. However, recent advancements, such as the Transformer [6] with the ability to capture long-range dependencies, have enabled geometry-free novel view synthesis of scenery [7, 8].

This paper shows how geometry-free monocular-to-stereo image view synthesis can be leveraged as a feature extractor to improve the navigation task performance and data efficiency of an RL agent from a monocular vision-based drone. Such a feature extractor does not primarily have to be excellent at view synthesis but has to extract depth for downstream use by DRL. The ability of these geometry-free view synthesis networks to reason about scenery has yet to be extensively studied. Therefore the proposed architecture is analyzed on performance for a variety of datasets [9, 10] and transfer between the datasets, on its ability to produce depth maps, and on the importance of the camera setup, design features, and training routines. This analysis shows that such a network synthesizes the other image in the image pair and encodes the relevant scenic features.

Training a drone to navigate with RL requires a simulation environment. The quality of simulation environments is constantly developing, with environments becoming more realistic in their dynamics, scenery, and sensor handling [11–13]. In our case, the integration and development of DRL and control over the stereo setup are necessities, which the Flightmare lineage provides [10, 11].

In this paper, we propose an implementation of the Swin Transformer [14] in the view synthesis domain and, for the first time, showcase competitive results compared to geometry-induced architectures on the KITTI dataset. The superiority compared to CNNs [15], especially without skip connections, is demonstrated. The ability of geometry-free architectures to encode depth is quantified. The feasibility of these architectures as feature extraction methods are demonstrated and the importance of the feature structure for view synthesis is discussed. This leads to the three main contributions:

1) The first implementation of the Swin Transformer in the view synthesis domain leading to the first competitive performance of a geometry-free model on the KITTI dataset.
2) The first method that quantifies the ability to encode depth of geometry-free view synthesis architectures.
3) The first DRL setup that uses a self-supervised view synthesis feature extractor for monocular camera-based drone navigation, leading to quicker convergence and higher performance than vanilla end-to-end or autoencoder-based setups.

II. Related Work

A. Reinforcement Learning for Drone Navigation

In recent years, DRL has evolved into a mature field of research. OpenAI Gym [16] standardized environment development and performance benchmarks for RL agents. The state-of-the-art algorithms, such as (Proximal Policy Optimization (PPO) [17], were standardized and centralized with the introduction of OpenAI Baselines [18], structures were unified with Stable-Baselines [19], new algorithms, such as Soft Actor-Critic (SAC) [20] and Twin Delayed DDPG (TD3) [21], were added in Stable-Baselines3 [22].

At the same time, the subdomain that applies RL to Unmanned Aerial Vehicles (UAV) is diverse but lacks standardization. AlMahamid and Grolinger [23] recently created a systematic review of this field of study as an extension of their previous work, a systemic review of the RL domain [24]. They recognize four drone navigation objectives: UAV control, obstacle avoidance, path planning, and flocking. These can differ from the objectives a software framework has: energy-aware UAV navigation, path planning, flocking, and vision-based frameworks and identify several papers that use the vision-based framework for drones [25–28]. Besides an RL agent, the simulation software requires a UAV flight simulator and a 3D graphics engine. They recognize that architectures using Robot Operating Systems (ROS) [29] integrated with Gazebo [12] and Microsoft AirSim [13] using the Unreal Engine [30] are the most popular.

Recently, Flightmare [11] was introduced as a flexible quadrotor simulator with a configurable rendering engine and a flexible physics engine. It has a sizeable multi-modal sensor suite, and wrappers are available for OpenAI Gym [16] and Stable Baselines [19]. Loquercio et al. [31] showed good real-life transfer using Flightmare with the RotorS Gazebo plugin [32] and rendering engine Unity [33].

There is an ongoing discussion on how to use vision within an RL framework for navigation. Due to its small feature space, many frameworks make use of monocular depth prediction as a feature extractor [34–37], or combine it with ego-motion [38]. Others use optical flow with semantic segmentation [39] or pretrained encoders [40] on ImageNet [41] classification.

Zhou et al. [3] researches the importance of feature extraction and what type of feature extraction is most effective. They analyze this for the extraction of intrinsic surface, color, optical flow, depth, and semantic segmentation and conclude that depth and semantic segmentation are most important to navigational performance for several tasks. Sax et al. [42] showed that adding other intermediate representations, such as curvature, denoising, and occlusion edges, can enhance performance even more.

Most previous work uses pretrained encoders and relies on sparse rewards. However, within the RL vision domain, the use of auxiliary tasks has also been researched. Jaderberg et al. [43] explored the use of unsupervised auxiliary tasks, whereas

Mirowski et al. [4] showcased the utility of supervised tasks, such as depth estimation.

B. Monocular Depth Estimation and View Synthesis

Within the domain specializing in monocular depth estimation, it was already proven that a neural network could learn to estimate depth from a single image [44]. Their network was trained and tested on the KITTI dataset [9]. The Eigen et al. [44] split created for training and testing would become the leading benchmark for monocular depth estimation and is known as the Eigen Split. Their method relies on groundtruth, which is often sparse and hard to obtain. Therefore self-supervised methods were developed that use a warping function based on camera knowledge and a stereo image pair to predict depth maps [45, 46].

Up to the introduction of Transformers [6], CNNs [47] were considered the standard in computer vision. This neural network can perform high-quality per-pixel predictions by using a U-Net architecture with skip connections [48]. The performance of per-pixel predictions can be further enhanced by learning structural latent representations of the data. Architectures renowned for this are VAE [49] and GAN [50]. Using vector quantization to derive discrete latent representations has further advanced the field, as VQ-VAE [51] and VQ-GAN [52] demonstrated.

Being able to capture long-range dependencies Transformers [53] achieved state-of-the-art in Natural Language Processing. Following Vision Transformers (ViT) [6] achieves state-of-the-art performance in image classification [41] by applying the Transformers to patches of 16×16 pixels. The Swin [14] Transformer achieves state-of-the-art in semantic segmentation [54] and object detection [55] by using shifted windows and a hierarchical structure inspired by CNNs.

Dosovitskiy et al. [56] describes a method that synthesizes a novel view of different chairs, which learns their 3D representations. Tatarchenko et al. [57] changed the architecture and was able to synthesize images of more complicated objects. Zhou et al. [5] developed a network able to predict appearance flow instead of direct pixel generation. As the appearance flow network outperformed direct pixel generation, more extensive scene representations, and warping functions were proposed [58–60] and applied to view synthesis [61–63]. These view synthesis architectures benchmarked on a split on KITTI proposed by Tulsiani et al. [61].

VQ-GAN uses Transformers to do vector quantization, and for the task of novel view synthesis, it was implemented by Rombach et al. [7] on RealEstate10K [59] and ACID [64]. These datasets contain images from continuous trajectories, and VQ-GAN is trained to synthesize views from a new viewpoint on these trajectories. Their method does not require geometrically induced biases. By using linear probing [65], they proved that depth was encoded in their architecture. Following, Sajjadi et al. [8] developed an architecture integrating ViT and CNNs to predict new poses from a collection of images from the same scene and, again, does not require geometrically induced biases.

The implementation of Swin Transformers enables per-pixel predictions. Able to capture long-range dependencies, it seems more suitable than CNNs for end-to-end geometry-free image view synthesis. The network extracts high-quality geometric features if the translation to another viewpoint is done accurately.

III. Geometry-Free Monocular-to-Stereo View Synthesis

This section proposes the geometry-free monocular-to-stereo view synthesis architecture and experiments on this architecture are performed.

A. Proposed Method

To gain insight into the competitiveness and potential of the view synthesis architecture, it is designed for the KITTI [9] dataset and the LDI [61] split. This view synthesis split requires bi-directional image predictions, so information regarding the direction of translation for synthesis (left-to-right or the other way around) is provided to the network. Inspired by Zhou et al. [5], Tatarchenko et al. [57], the information is given as a token in the bottleneck. It is after the token is introduced that translation is performed. Consequently, geometric information has to be present at the bottleneck. This makes the encoder suitable for the navigation task.

Two architectures similar to U-Net [48] are proposed. The first architecture uses two subsequent convolutional layers [15] for feature extraction, pooling layers for downsampling, and a convolutional layer followed by an upsample layer for upsampling. Before the feature extraction blocks in the encoder, the skip connections concatenate the features from the encoded and upsampled layers. The second architecture uses two subsequent Swin [14] layers for feature extraction and fully connected layers for downsampling. Changed from the original Swin architecture to make the architecture more similar to a U-Net architecture, embedding is done with a patch size of two. The implementation of upsampling and skip connections follow Cao et al. [66]. Modified from their architecture is a skip connection introduced at the original image resolution to make the architecture more similar to a U-Net. The general structure of both architectures is shown in Fig. 1. The Swin and CNN architecture uses an embedding dimension of 96 and 56 channels, respectively. These architectures will also be compared to Encoder-Decoder variants, which exclude the skip connections. For the CNN architecture, this means that the number of channels for the first convolutional layer after upsampling stays constant. Whereas for the architecture including Swin Transformers, the fully connected layer after upsampling is removed.

B. Training Details

The framework is designed in PyTorch [67]. The loss function selected is the Smooth L1 Loss, covering the advantages of both the L1 and L2 Loss. The cosine learning rate scheduler [68] is used as implemented by Dosovitskiy et al. [6], Liu et al. [14]. AdamW [69] is state-of-the-art and therefore used.

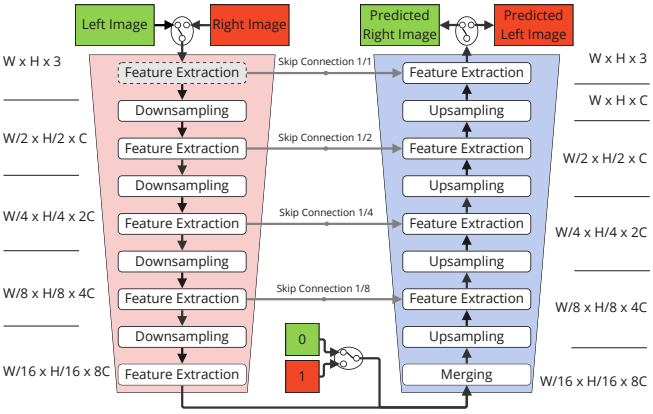


Fig. 1: U-Net architecture for geometry-free monocular-to-stereo view synthesis

For data augmentation, we predominately follow horizontal flipping and color augmentation from Godard et al. [70] and data grafting from Peng et al. [71].

C. Results and Discussion

The view synthesis architecture is benchmarked on the split proposed by Tulsiani et al. [61], as this split benchmarks bi-directional stereo view synthesis. The split contains 28 predetermined city scenes from the KITTI-raw dataset, of which 20 sequences are used for training, 4 for validation, and 4 for testing. The images are used as inputs at their original resolution (1280×384). During training, the output resolution is downsized from the original resolution by $1/3$, and 5% of the edges on the left and right sides are removed. These are removed as many artifacts were generated by their architecture in these regions. During testing, they render the output downsized by two to avoid cracks. Both images will have the same resolution for the application to a stereo-camera-based drone. Therefore we test our architecture with equal input and output resolution and only remove the edges in post-processing. Some papers also trained and tested with the output resolution at $1/3$ of the original resolution. The metrics used by Tulsiani et al. [61] for benchmarking are SSIM [72] and Peak Signal-to-Noise Ratio (PSNR). Since then, LPIPS [73] with VGG features became a popular metric for view synthesis due to its effectiveness in measuring perceptual realism. The benchmark results are shown in Table I. Training for this benchmark is computationally intensive due to the image resolution and a large number of epochs. As such, only the best-performing network, Swin (U-Net), is benchmarked.

All other methods that benchmark on the split rely on geometric models. LDI [61] has an associated depth value for each pixel, and MPI [62] uses multiple semi-transparent planes at different depths. MINE [63] and VEST [74] use NeRF [60], which learns volume density and color at each 3D location. VEST [74] uses sequential information about the relative change in viewpoint from one image pair to the next in theirs. As such, these methods have an inherent advantage to a model directly predicting pixels. Rombach et al. [7], Sajjadi et al. [8] introduce architectures that do not create implicit or explicit geometric models but are not tested on comparable datasets. On the SSIM and PSNR metrics, the Swin (U-Net) outperforms LDI [61] and MPI [62]. It is worse on all metrics compared to MINE [63] and is better on the PSNR metric compared to VEST [74], while worse than on the SSIM and LPIPS metrics. Presently, LDI and MPI are commonly used techniques to encode geometric information. Therefore, they can serve as a standard for evaluating the effectiveness of geometric encoding methods in this context. The performance of Swin (U-Net) exceeds that of LDI and MPI, therefore achieving a reasonable ability to encode geometric information.

Subsequently, we analyze whether the network captures depth information on the Eigen [44] Split for monocular depth estimation. By using StereoSGBM from OpenCV [75], the original and predicted image are matched to predict disparity, from which depth is obtained. As this method leaves a border of the image that is the size of the number of disparities (64) empty, stereo-matching is also done in the other direction. Where the two depth maps overlap, their average is taken, and on the sides, the depth map with values is used. The depth map is then resized to overlap with the ground truth, and predictions are limited to 80 meters as stated by Garg et al. [45]. The architecture is compared to Eigen et al. [44] and two self-supervised methods [45, 46]. These are not state-of-the-art, as the primary purpose of our method is not depth estimation accuracy. The self-supervised methods are of interest as they are not trained on depth directly but have an induced geometric advantage through their warping mechanism. The results are shown in Table II.

Comparing the results from Swin (U-Net) to Eigen et al. [44], it performs better on the absolute relative error and a delta criterion while worse on the squared relative error, root mean squared error, its logarithmic domain, and two delta criteria. Compared to the self-supervised methods, it performs worse on all metrics. It should be noted that the matching algorithm limits the architecture's benchmark performance. By using this algorithm on the original images to obtain depth maps, it is outperformed by Godard et al. [46] on six of the seven metrics. The earlier proposed architectures are also benchmarked on this split. CNN (U-Net) achieves slightly worse scores, while removing skip connections reduces the performance. Overall, these architectures encode depth, although at lower accuracy.

The different architectures are further analyzed by comparing them for view synthesis on the LDI [61] Split as well as on a generated dataset. Ordinarily, 5% of the edges are removed, resulting in images of 128×384. Swin uses a fixed window size, so the images must be padded up to 128×512. Able to directly scale to 128×384 without removing the edges would reduce the network size significantly. This is done for the experiments that compared the performance of the proposed architectures. The generated dataset is obtained in the outdoor environment of AvoidBench [10]. Two baselines between the cameras are used to study further the ability to predict images

TABLE I: Comparison of the view synthesis performance of the Swin (U-Net) architecture trained for 500 epochs on KITTI with the LDI split compared to existing geometrically-included view synthesis methods

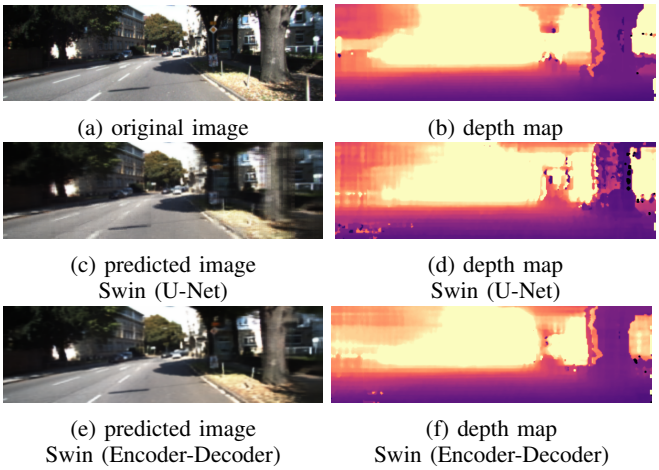
LDI [61] Split	Method	LPIPS ↓	SSIM↑	PSNR↑
train 256 × 768 test 128 × 384	LDI [61]	N/A	0.572	16.5
	MPI [62]	N/A	0.733	19.5
	MINE [63]	0.108	0.820	21.3
	VEST [74]	**0.085**	**0.825**	**21.6**
train 128 × 384 test 128 × 384	MINE [63]	0.129	0.812	**21.4**
	VEST [74]	**0.097**	**0.818**	21.1
	Ours [Swin (U-Net)]	0.161	0.787	21.3

TABLE II: Comparison of the monocular depth performance of the four proposed view synthesis architectures trained for 60 epochs on KITTI with the Eigen Split using a stereo matching algorithm compared to the original (self-)supervised methods

Eigen [44] Split	Method	Abs Rel ↓	Sq Rel ↓	RMSE ↓	RMSE log ↓	$\delta < 1.25 \uparrow$	$\delta < 1.25^2 \uparrow$	$\delta < 1.25^3 \uparrow$
80 meter 1242 × 375	Eigen [44]	0.203	1.548	6.307	0.282	0.702	0.890	0.958
	Garg [45]	0.152	1.226	5.849	0.246	0.784	0.921	0.967
	MonoDepth [45]	**0.133**	**1.142**	**5.533**	**0.230**	0.830	**0.936**	**0.970**
	Ours [Swin (U-Net)]	0.199	2.196	6.976	0.425	0.707	0.887	0.945
	Ours [CNN (U-Net)]	0.211	2.519	7.228	0.451	0.701	0.878	0.937
	Ours [Swin (Encoder-Decoder)]	0.202	2.075	7.086	0.427	0.690	0.880	0.941
	Ours [CNN (Encoder-Decoder)]	0.232	2.588	7.940	0.497	0.629	0.846	0.925
	Stereo Matching Algorithm	0.136	2.120	5.954	0.435	**0.857**	0.930	0.955

(0.5m and 1.0m). The images are generated at a resolution of 224 × 224. Both training datasets contain 20,000 images, and testing datasets contain 4,000 images. Table III shows the results for the four architectures.

In general, Swin (U-Net) outperforms the other architectures on all datasets for LPIPS. For the synthetic dataset with the smaller baseline, the LPIPS score is almost identical to CNN (U-Net). Swin (U-Net) also achieves the best score on SSIM for the LDI Split. However, the Swin (Encoder-Decoder) scores higher on both synthetic datasets. SSIM measures structural similarity, so the Encoder-Decoder preserves structural patterns better but is less accurate at capturing fine-grained details and textures. For the PSNR metric, Swin (Encoder-Decoder) performs the best. It should be noted that some argue that PSNR is not a suitable metric for comparing quality and performance over different scenes [76]. Nevertheless, Swin is a better-performing layer compared to CNN. The skip connections aid in the performance significantly. However, without the skip connections, the Swin architecture maintains performance better, indicating that it relies less on the skip connections. It further shows that the image prediction performance is reduced for all architectures with an increased baseline. Most notable is that CNN (U-Net) loses more performance than Swin (U-Net), although the reduction is significant in both cases. An example of the predicted images and their corresponding depth maps through stereo matching for the Swin architectures are shown in Fig. 2. Fig. 2b is obtained through stereo matching of the original images. Visually it looks like Swin (Encoder-Decoder) produces structures such as the tree and poles more accurately. Nevertheless, analyzing the depth map, it seems that the depth map of Swin (U-Net) is overall better, and as such, the placement of objects is more accurate. This would make sense as the skip connections could help with pixel precision, but as such, ignore the shapes of these objects from the deeper layers. More examples are available in Appendix VII-E.



(a) original image (b) depth map

(c) predicted image Swin (U-Net) (d) depth map Swin (U-Net)

(e) predicted image Swin (Encoder-Decoder) (f) depth map Swin (Encoder-Decoder)

Fig. 2: Example of predictions and depth maps through stereo matching of predicted images and input images on the KITTI dataset

Ablation studies are done on augmentation techniques and the type of loss function. Details on these studies are discussed in Appendix VII-A and Appendix VII-B. The main takeaways are that color augmentation and image flipping aid performances, while data grafting does less so. Smooth L1 Loss achieves better performances compared to L1 and L2

TABLE III: Comparison of the view synthesis performance of the proposed architectures on KITTI with the LDI split and two generated datasets with AvoidBench for two baselines.

Split	Method	LPIPS ↓	SSIM↑	PSNR↑
LDI Split	Swin (U-Net)	**0.163**	**0.790**	20.8
train 128 × 384	CNN (U-Net)	0.176	0.787	20.7
test 128 × 384	Swin (Encoder-Decoder)	0.201	0.781	**21.0**
150 Epochs	CNN (Encoder-Decoder)	0.438	0.658	18.8
AvoidBench - Outdoor	Swin (U-Net)	0.197	0.752	21.2
Baseline: 0.5m	CNN (U-Net)	**0.196**	0.754	21.1
224 × 224	Swin (Encoder-Decoder)	0.232	**0.769**	**21.5**
50 Epochs	CNN (Encoder-Decoder)	0.411	0.624	19.4
AvoidBench - Outdoor	Swin (U-Net)	**0.260**	0.694	19.6
Baseline: 1.0m	CNN (U-Net)	0.304	0.681	18.4
224 × 224	Swin (Encoder-Decoder)	0.311	**0.716**	**20.3**
50 Epochs	CNN (Encoder-Decoder)	0.464	0.591	17.3

Loss.

The view synthesis analysis concludes that a geometry-free monocular-to-stereo image view synthesis achieves good view synthesis performance and can encode depth reasonably well.

IV. View Synthesis for DRL-based Drone Navigation

Able to encode depth and geometric information, the geometry-free monocular-to-stereo image view synthesis will be applied to drone navigation in this section.

A. Proposed Method

The simulation environment selected is AvoidBench [10], based on Flightmare [11]. Similar to Loquercio et al. [31], the RotorS Gazebo plugin [32] and rendering engine Unity [33] are used. To have more control over the process and system communication, the OpenAI Gym [16] and Stable Baselines [19] wrappers are not used. However, they are integrated into the existing code structure. PPO [17] is used from Stable Baselines3 [22]. Minor adjustments are made to the algorithm, such as removing a feature extraction layer and using AdamW [69] as the optimizer instead of Adam [77].

AvoidBench [10] uses a stereo camera setup aligned to the center of the drone. Although our architecture relies on stereo vision in training, the drone only uses a single camera during evaluation. Therefore in our simulator, the left camera is aligned to the center of the drone, and the right camera images are obtained by a virtual camera 0.48 meters from the center. The camera resolution is set to 64×64 and the frame rate to 10 Hz in simulation time. The policy frequency of the RL agent is set equal to the camera frame rate.

An indoor and an outdoor environment are available. For training, the indoor environment is used for all experiments, except when it is explicitly stated that the experiment is done in the outdoor environment. Each run spawns red cylinders with shape $(x, y, z) \sim \mathcal{U}((0.5\text{m}, 0.5\text{m}, 2.6\text{m})(1.0\text{m}, 1.0\text{m}, 3.8\text{m}))$ using Poisson-disk sampling. During training, the radius of the Poisson disk sampling: $r \sim \mathcal{U}(1.5\text{m}, 3\text{m})$ and during evaluation, $r = 3$. The radius was kept constant during evaluation to get more consistent test results. The drone is spawned on a line with coordinates $(-1.8\text{m}, y, 3\text{m})$, where $y \sim \mathcal{U}\{-8\text{m}, \dots, 8\text{m}\}$ and the goal is to fly to a target on a line with coordinates $(32\text{m}, y, 3\text{m})$, where $y \sim \mathcal{U}\{-6\text{m}, \dots, 6\text{m}\}$, with objects in between. Drone control is of lower interest, so the mission is completed once the drone passes $x = 32\text{m}$ and is within a 6-meter radius of the target. Fig. 3 shows the indoor environment and a sketch of the top view, including spawn points and targets in blue and green, respectively. The outdoor environment contains bushes and trees, and the experimental setup is similar. Due to the increased size of these objects during training, the radius $r \sim \mathcal{U}(3\text{m}, 5\text{m})$ and during evaluation, $r = 5$. The outdoor environment does not contain natural boundaries like the warehouse walls in the indoor environment. Therefore the drone is limited to flying within the area with coordinates spanning $y \sim \mathcal{U}(-10\text{m}, \dots, 10\text{m})$.

(a) indoor environment (b) top view

Fig. 3: Indoor Simulation Environment

The architecture proposed has an encoder trained by an auxiliary task, with an RL agent trained on an observation that includes the extracted features from the image. Asynchronous parallel training should foster feature representation and encourage the policy to use the feature representation. The mission task is path planning, therefore, the agent requires extra information besides the image features. Giving the relative position to the target in the body frame $\left(x_t^b, y_t^b, z_t^b\right)$ and the velocity and the yaw rate in the body frame $\left(v_x^b, v_y^b, v_z^b, \omega_z^b\right)$, should provide enough information for drone control and path planning. To prevent the drone from flying out of bounds in the outdoor environment, the position of the drone in the global

frame is provided to the agent (x_d^g, y_d^g, z_d^g).

In Flightmare, a PID controller manages velocity control in the global frame. When actions from the RL policy are transformed from the body frame to the global frame, the RL agent supplies target velocities and yaw rates in the body frame. At every timestep, a reward equal to the relative distance from the previous to the current timestep for the drone to the target $(\Delta\vec{r}_t^b)$ is provided to the RL agent. This is visualized in Fig. 4. Besides, there is a minor penalty for collision and a small reward for reaching the target. The reward function is concisely presented in Eq. 1. After each collision, the environment is reset, and the run ends. The architecture is shown in Fig. 5.

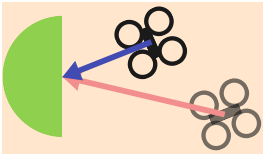


Fig. 4: The current relative displacement (blue arrow) is subtracted from the previous relative displacement (red arrow) to obtain the reward

$$Reward = \begin{cases} 20 & \text{if target reached} \\ -5 & \text{if collision} \\ \Delta\vec{r}_t^b & \text{otherwise} \end{cases} \quad (1)$$

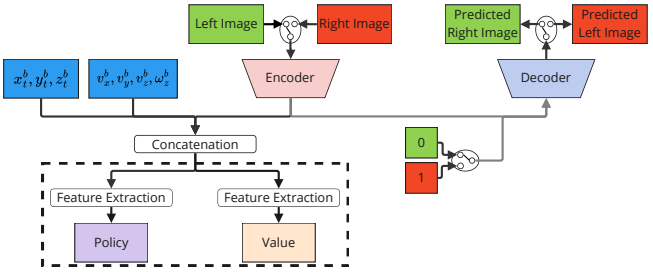


Fig. 5: Deep RL architecture for path planning with an auxiliary task

In practice, the RL agent is trained after a rollout that receives images and provides actions. After 2048 steps, back-propagation is performed on all images. The Encoder-Decoder is trained on images collected every step in batches of 8 images. The images are trained with color augmentation to make transfer better and with a learning rate scheduling policy that decays the learning rate from 10^{-3} to 10^{-6} with a constant factor.

B. Results and Discussion

The four geometry-free monocular-to-stereo image view synthesis architectures (Swin (U-Net), CNN (U-Net), Swin (Encoder-Decoder), and CNN (Encoder-Decoder)) were ranked based on view synthesis performance. This order is not guaranteed to translate to navigational performance. The RL agent receives feature spaces from the extractors. In the experiments, the feature extractors will be the same size. However, due to the skip connections in the U-Net architecture and these containing relevant features for view synthesis, they are also provided in the feature space to the RL agent, making it significantly more extensive compared to the feature space of the Encoder-Decoder. The embeddings produced by the Swin Transformer do not conform to the grid-like activation patterns typically generated by convolutional layers. This divergence could impact the complexity of training the reinforcement learning algorithm.

For this and the following experiments, the networks are trained in the indoor environment of AvoidBench. After every 100,000 steps, the architectures are validated for ten simulation runs, and the experiment is repeated four times. All feature extractors are trained as an auxiliary task, and the mission progress is measured as the percentage of the distance the agent has traversed relative to the total distance between the starting point and the target. As previously noted, hitting the target with absolute precision is not obligatory. Consequently, a mission is completed without achieving 100%. Comparing three of the four proposed view synthesis architectures yields the results in Fig. 6.

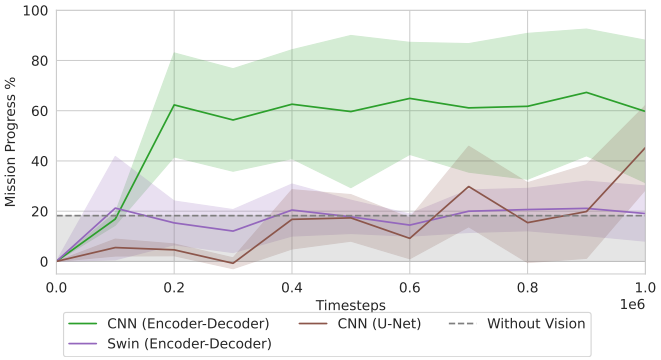
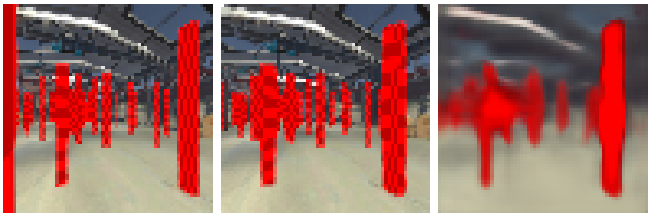


Fig. 6: Mean mission progress ($\pm$ standard deviation) for RL agent with the proposed view synthesis tasks, averaged over 4 training sessions, 10 evaluation runs per session, every 10^5 timesteps.

Fig. 6 shows that the CNN (Encoder-Decoder) setup achieves the quickest convergence and the best performance. Due to the increased feature space of the CNN (U-Net), learning the essential features for obstacle avoidance takes longer. The Swin (Encoder-Decoder) setup does not improve over time at all. Its performance is limited to flying in the direction of the target without avoiding obstacles. As the Swin (Encoder-Decoder) setup does not improve, it was decided not to show the Swin (U-Net) setup results as they will be similar. A feature ablation study is performed to determine whether the RL agent uses the features from the Swin (Encoder-Decoder). The features are split into two groups. The first group contains the features yielded from the feature extractor, and the second group contains the seven navigational features $\left(x_t^b, y_t^b, z_t^b, v_x^b, v_y^b, v_z^b, \omega_z^b\right)$. Each group of features is masked

during the run to study the impact of those features in the decision-making of the RL agent. Masking the features is done by providing randomized values within the limits defined in the observation space, and results are shown in Table IV. As expected, the CNN architecture relies entirely on the image features, whereas the change in performance of the Swin architecture is negligible, seemingly optimized to fly straight toward the target. Therefore, from this point on CNNs (Encoder-Decoder) will be used. An example of images used and produced by the network is Fig. 7 and examples of the trajectories this setup flies are sown in Fig. 8.

Furthermore, in Appendix VII-C, an ablation study on the performance of each of the seven navigational features are also done. x_t^b has the most considerable contribution to the navigational performance, whereas randomized inputs for z_t^b and ω_t^b even improved performance. This implies that outcomes could be further enhanced through more strategic feature selection or by fine-tuning the neural network to leverage these features better.



(a) left (b) right (c) predicted right

Fig. 7: Indoor environment of AvoidBench during evaluation from the POV of the drone with the predicted right image

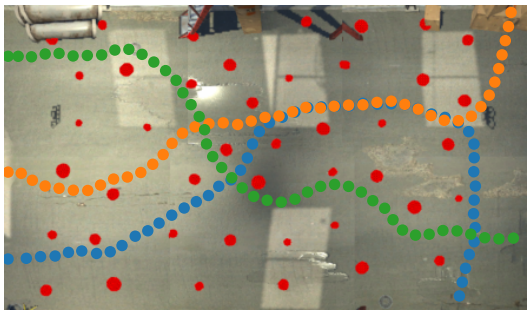


Fig. 8: Top view of the indoor environment in the first evaluation map with different trajectories from a trained CNN (Encoder-Decoder) setup

It is compared to other setups to study the navigation task performance and data efficiency of the reinforcement learning agent using monocular-to-stereo image view synthesis as an auxiliary feature extractor. The first setup is designed such that the RL agent receives the images directly without a feature extractor, which means that it is trained end-to-end by the RL agent. The second setup uses the encoder of an autoencoder as the feature extractor. A third setup uses the output of a depth prediction encoder-decoder. This setup is trained on groundtruth depth available in the simulator. A fourth setup is proposed that again is trained on depth prediction, but this time the latent space at the bottleneck is provided to the RL agent instead of the output. The second, third, and fourth architectures are identical in size to the novel view synthesis feature extractor. Again all feature extractors are trained in an auxiliary fashion, and the mission progress is measured as the percentage of the distance to the target point. For these runs, the result is Fig. 9.

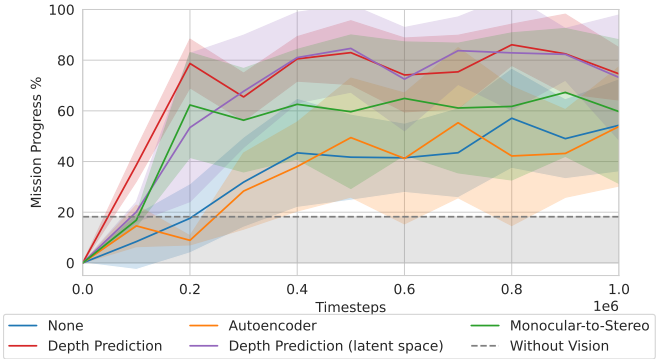


Fig. 9: Mean mission progress ($\pm$ standard deviation) for RL agent with various auxiliary feature extractors, averaged over 4 training sessions, 10 evaluation runs per session, every 10^5 timesteps in the indoor environment.

The results of the architectures, including monocular-to-stereo view synthesis and monocular depth prediction, clearly show two training phases. During the first 200,000 timesteps, there is a steep learning curve, whereas afterward, it seems stable. Architectures including an autoencoder and no feature extractor learn slower, and therefore the distinction between these two phases is less clear. Depth prediction achieves better results compared to monocular-to-stereo view synthesis. However, even after 1,000,000 timesteps, monocular-to-stereo view synthesis still performs better than autoencoding and no feature extractor. For all methods, the reward, mission progress, and success rate after training are stated in Table V. As expected, given enough time, the RL agent can determine the relevant features of an image for navigation. Autoencoding follows a similar trend. This indicates that no specific features in constructing an image are present that specifically aid directly with obstacle avoidance, and the RL agent has to find correlations between multiple features and obstacle avoidance again.

Looking at the auxiliary Loss in Fig. 10, it is clear that all tasks converge relatively quickly. Autoencoding converges the quickest, whereas depth prediction takes the longest. As monocular-to-stereo view synthesis more quickly converges and is tasked with learning both to predict depth and to generate a view, this could indicate that the potential ability of this network to encode depth is more limited. Which is likely due to a feature space that is more limited in allocating its features for depth estimation.

These setups are also trained for the outdoor environment

TABLE IV: Comparison of the path planning performance of the RL agent with the monocular-to-stereo view synthesis feature extractor with masked image features

View Synthesis Method	Reward ($\mu \pm \sigma$)	Mission Progress ($\mu \pm \sigma$)	Success Rate
Swin (Encoder-Decoder)	2.0 ± 4.0	20.6% ± 11.5%	0%
Swin (Encoder-Decoder) w. Masked Image Features	3.3 ± 4.5	24.6% ± 13.4%	0%
CNN (Encoder-Decoder)	**21.5 ± 15.2**	**59.7% ± 28.3%**	**25%**
CNN (Encoder-Decoder) w. Masked Image Features	-2.4 ± 2.7	7.6% ± 7.9%	0%

TABLE V: Comparison of the path planning performance of different RL architectures after 1,000,000 steps averaged for four runs

Environment	Feature Extraction Method	Reward ($\mu \pm \sigma$)	Mission Progress ($\mu \pm \sigma$)	Success Rate
indoor	None	14.6 ± 9.5	54.3% ± 18.4%	5%
	View Synthesis (Autoencoder)	15.1 ± 11.4	53.8% ± 23.5%	7.5%
	View Synthesis (Monocular-to-Stereo)	21.5 ± 15.2	59.7% ± 28.3%	25%
	Depth Prediction	30.3 ± 9.8	**74.6% ± 10.3%**	40%
	Depth Prediction (Bottleneck)	**30.4 ± 17.2**	73.2% ± 24.6%	**42.5%**
outdoor	None	2.6 ± 7.2	22.5% ± 21.4%	0%
	View Synthesis (Monocular-to-Stereo)	**7.8 ± 8.4**	**37.8% ± 24.9%**	0%
	Depth Prediction	2.6 ± 7.2	22.5% ± 21.4%	0%

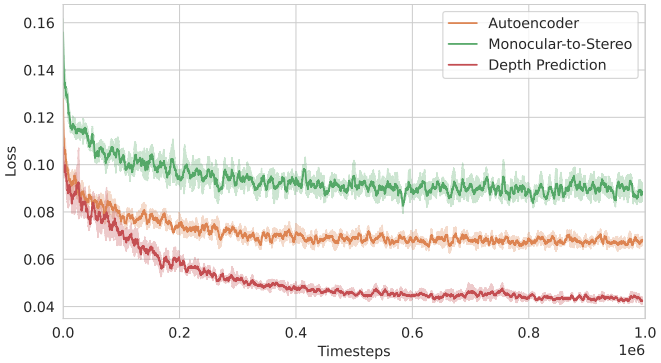


Fig. 10: Auxiliary loss for the various feature extractors (different kinds of losses) averaged over 4 training sessions.

to show generalizability and learning ability in other environments. The outdoor environment is designed similarly to the indoor environment, with a different density of objects. As stated previously, the RL agent is provided with the coordinates in the global frame. Still, learning in this environment was more difficult, as shown in Fig. 11. A reason for this could be the high complexity of the objects within this environment and an inability to use the correct features. From visual inspection, it seems that the drone detects obstacles but is unsure of the appropriate behavior to fly past them.

The stereo setup was assumed. Both image size and stereo baseline affect the path planning ability, as evidenced by the analysis in Appendix VII-D. The main takeaway is that for a symmetric stereo camera setup with a baseline of 12cm, improvements of the navigational performance compared to the autoencoder are already clearly present. Increasing the baseline to 48cm does increase the convergence speed but does not improve the performance significantly after 1,000,000 timesteps. Reducing the resolution lowers the convergence but does not significantly impact the overall performance.

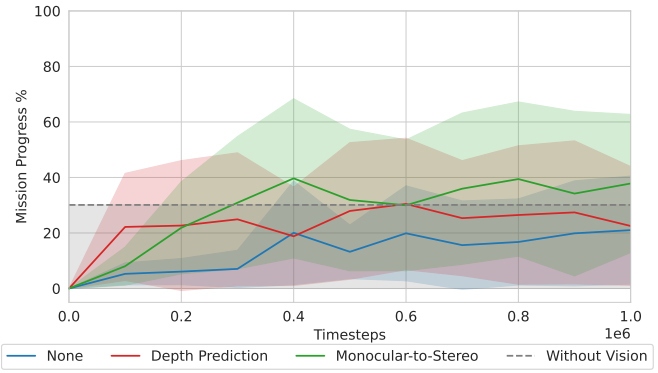


Fig. 11: Mean mission progress (± standard deviation) for RL agent with various auxiliary feature extractors, averaged over 4 training sessions, 10 evaluation runs per session, every 10^5 timesteps in the outdoor environment.

V. Conclusion

This study has presented a novel method for Deep Reinforcement Learning (DRL) to enhance the performance of vision-based obstacle avoidance. We demonstrated that utilizing the view synthesis as a feature extractor has shown that the learning speed and navigation potential can be improved. The proposed view synthesis architectures were tested against established benchmarks, where they demonstrated competitive performance in view synthesis and satisfactory results in monocular depth estimation through stereo matching.

The paper underscores the advantage of employing the latent space of view synthesis architectures as input to PPO agents over other feature extractors. While the study affirmed the advantage of depth estimation, it also showed that our proposed method offers quicker convergence and superior

performance compared to using autoencoding as a feature extractor and training end-to-end with DRL.

Moreover, the study presented the first implementation of the Swin Transformer in the view synthesis domain, demonstrating its competitive edge, especially compared to Convolutional Neural Networks (CNNs) without using skip connections. This work also quantified the ability of geometry-free architectures to encode depth and discussed the importance of feature structure for view synthesis.

Despite the promising results, this study acknowledges certain limitations. For instance, while our approach outperforms autoencoding and end-to-end DRL methods, monocular depth estimation still outperforms our method. Future work could explore the benefits of increasing the view synthesis architecture and feature space size. The architectures also make limited use of sequential information, and adding an LSTM layer could aid performance. Besides, increased image size and action frequency should eventually help performance. However, all suggestions increase the network's complexity and will likely reduce the convergence speed of the RL agent. Secondly, the performance could likely be enhanced through more strategic navigational feature selection or by fine-tuning the neural network to leverage these features better. Finally, all results in the outdoor environment are worse compared to the indoor environment and the outcomes of both experiments do not align. Therefore, transferability and navigational feature selection should be studied better.

In conclusion, the research contributes significantly to the field of DRL for monocular camera-based drones by showcasing the potential of view synthesis as a feature extractor. It opens up new possibilities for improving drone navigation and provides valuable insights for future research. Ultimately, our findings cement the vital role of innovative methods and architectures in advancing the performance and efficiency of DRL systems for monocular vision-based drones.

VI. Future Research

This research focused on mapping a single image to another and extracting geometric information in the process. If a drone has stereo, it can use the two images even better by entering both into a predictive framework. On the other hand, in a purely monocular framework, the current image can also be used to predict the next image via pose and depth prediction. These alternative learning setups will be explored in future work.

References

[1] D. Ha and J. Schmidhuber, "World models," *arXiv preprint arXiv:1803.10122*, 2018.

[2] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[3] B. Zhou, P. Krähenbühl, and V. Koltun, "Does computer vision matter for action?" *Science Robotics*, vol. 4, no. 30, p. eaaw6661, 2019. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.aaw6661

[4] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell, "Learning to navigate in complex environments," *CoRR*, vol. abs/1611.03673, 2016. [Online]. Available: http://arxiv.org/abs/1611.03673

[5] T. Zhou, S. Tulsiani, W. Sun, J. Malik, and A. A. Efros, "View synthesis by appearance flow," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 286–301.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. [Online]. Available: https://arxiv.org/abs/2010.11929

[7] R. Rombach, P. Esser, and B. Ommer, "Geometry-free view synthesis: Transformers and no 3d priors," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2021, pp. 14 356–14 366.

[8] M. Sajjadi, H. Meyer, E. Pot, U. Bergmann, K. Greff, N. Radwan, S. Vora, M. Lucic, D. Duckworth, A. Dosovitskiy, J. Uszkoreit, T. A. Funkhouser, and A. Tagliasacchi, "Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations," *CoRR*, vol. abs/2111.13152, 2021. [Online]. Available: https://arxiv.org/abs/2111.13152

[9] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[10] H. Yu, G. C. de Croon, and C. De Wagter, "Avoidbench: A high-fidelity vision-based obstacle avoidance benchmarking suite for multi-rotors," *arXiv preprint arXiv:2301.07430*, 2023.

[11] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," *CoRR*, vol. abs/2009.00563, 2020. [Online]. Available: https://arxiv.org/abs/2009.00563

[12] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[13] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and service robotics*. Springer, 2018, pp. 621–635.

[14] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision

transformer using shifted windows," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9992–10 002.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[16] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[18] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[19] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[20] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[21] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[22] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[23] F. AlMahamid and K. Grolinger, "Autonomous unmanned aerial vehicle navigation using reinforcement learning: A systematic review," *Engineering Applications of Artificial Intelligence*, vol. 115, p. 105321, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095219762200358X

[24] ——, "Reinforcement learning algorithms: An overview and classification," in *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2021, pp. 1–7.

[25] A. Singla, S. Padakandla, and S. Bhatnagar, "Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 1, pp. 107–118, 2021.

[26] L. He, N. Aouf, J. F. Whidborne, and B. Song, "Integrated moment-based lgmd and deep reinforcement learning for uav obstacle avoidance," *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7491–7497, 2020.

[27] W. Andrew, C. Greatwood, and T. Burghardt, "Deep learning for exploration and recovery of uncharted and dynamic targets from uav-like vision," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE Press, 2018, p. 1124–1131. [Online]. Available: https://doi.org/10.1109/IROS.2018.8593751

[28] M. A. Akhloufi, S. Arola, and A. Bonnet, "Drones chasing drones: Reinforcement learning and deep search area proposal," *Drones*, 2019.

[29] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[30] B. Karis and E. Games, "Real shading in unreal engine 4," *Proc. Physically Based Shading Theory Practice*, vol. 4, no. 3, p. 1, 2013.

[31] A. Loquercio, E. Kaufmann, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Learning high-speed flight in the wild," *Science Robotics*, vol. 6, no. 59, p. eabg5810, 2021. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abg5810

[32] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart, *RotorS—A Modular Gazebo MAV Simulator Framework*. Cham: Springer International Publishing, 2016, pp. 595–625. [Online]. Available: https://doi.org/10.1007/978-3-319-26054-9_23

[33] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *CoRR*, vol. abs/1809.02627, 2018. [Online]. Available: http://arxiv.org/abs/1809.02627

[34] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars, and L. Van Eycken, "Cnn-based single image obstacle avoidance on a quadrotor," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 6369–6374.

[35] L. Xie, S. Wang, A. Markham, and N. Trigoni, "Towards monocular vision based obstacle avoidance through deep reinforcement learning," *CoRR*, vol. abs/1706.09829, 2017. [Online]. Available: http://arxiv.org/abs/1706.09829

[36] Z. Xue and T. Gonsalves, "Monocular vision obstacle avoidance uav: A deep reinforcement learning method," in *2021 2nd International Conference on Innovative and Creative Information Technology (ICITech)*, 2021, pp. 1–6.

[37] M. Kim, J. Kim, M. Jung, and H. Oh, "Towards monocular vision-based autonomous flight through deep reinforcement learning," *Expert Systems with Applications*, vol. 198, p. 116742, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417422002111

[38] K. Yokoyama and K. Morioka, "Autonomous mobile robot with simple navigation system based on deep

reinforcement learning and a monocular camera," in *2020 IEEE/SICE International Symposium on System Integration (SII)*, 2020, pp. 525–530.

[39] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, "Obstacle avoidance drone by deep reinforcement learning and its racing with human pilot," *Applied Sciences*, vol. 9, no. 24, 2019. [Online]. Available: https://www.mdpi.com/2076-3417/9/24/5571

[40] F. Sadeghi and S. Levine, "Cad2rl: Real single-image flight without a single real image," in *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017.

[41] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[42] A. Sax, B. Emi, A. Zamir, L. J. Guibas, S. Savarese, and J. Malik, "Mid-level visual representations improve generalization and sample efficiency for learning active tasks," *CoRR*, vol. abs/1812.11971, 2018. [Online]. Available: http://arxiv.org/abs/1812.11971

[43] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu, "Reinforcement learning with unsupervised auxiliary tasks," *CoRR*, vol. abs/1611.05397, 2016. [Online]. Available: http://arxiv.org/abs/1611.05397

[44] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 2366–2374.

[45] R. Garg, V. K. B.G., G. Carneiro, and I. Reid, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds. Cham: Springer International Publishing, 2016, pp. 740–756.

[46] C. Godard, O. M. Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6602–6611.

[47] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, pp. 84 – 90, 2012.

[48] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[49] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013. [Online]. Available: https://arxiv.org/abs/1312.6114

[50] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[51] A. van den Oord, O. Vinyals, and K. Kavukcuoglu, "Neural discrete representation learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6309–6318.

[52] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 12 868–12 878.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[54] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 633–641.

[55] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[56] A. Dosovitskiy, J. T. Springenberg, and T. Brox, "Learning to generate chairs with convolutional neural networks," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1538–1546.

[57] M. Tatarchenko, A. Dosovitskiy, and T. Brox, "Multi-view 3d models from single images with a convolutional network," in *European Conference on Computer Vision*. Springer, 2016, pp. 322–337.

[58] J. Shade, S. Gortler, L.-w. He, and R. Szeliski, "Layered depth images," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998, pp. 231–242.

[59] T. Zhou, R. Tucker, J. Flynn, G. Fyffe, and N. Snavely, "Stereo magnification: Learning view synthesis using multiplane images," *ACM Trans. Graph.*, vol. 37, no. 4, jul 2018. [Online]. Available: https://doi.org/10.1145/3197517.3201323

[60] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *Computer Vision – ECCV 2020*, A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, Eds. Cham: Springer International Publishing, 2020, pp. 405–421.

[61] S. Tulsiani, R. Tucker, and N. Snavely, "Layer-structured 3d scene inference via view synthesis," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds. Cham: Springer International Publishing, 2018, pp. 311–327.

[62] R. Tucker and N. Snavely, "Single-view view synthesis with multiplane images," in *CVPR*, 2020. [Online]. Available: https://arxiv.org/abs/2004.11364

[63] J. Li, Z. Feng, Q. She, H. Ding, C. Wang, and

G. H. Lee, "Nemi: Unifying neural radiance fields with multiplane images for novel view synthesis," *CoRR*, vol. abs/2103.14910, 2021. [Online]. Available: https://arxiv.org/abs/2103.14910

[64] A. Liu, A. Makadia, R. Tucker, N. Snavely, V. Jampani, and A. Kanazawa, "Infinite nature: Perpetual view generation of natural scenes from a single image," in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 14 438–14 447.

[65] M. Chen, A. Radford, J. Wu, H. Jun, P. Dhariwal, D. Luan, and I. Sutskever, "Generative pretraining from pixels," in *International Conference on Machine Learning*, 2020.

[66] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian, and M. Wang, "Swin-unet: Unet-like pure transformer for medical image segmentation," *arXiv preprint arXiv:2105.05537*, 2021.

[67] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[68] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," *arXiv preprint arXiv:1608.03983*, 2016.

[69] ——, "Fixing weight decay regularization in adam," *ArXiv*, vol. abs/1711.05101, 2017.

[70] C. Godard, O. M. Aodha, M. Firman, and G. Brostow, "Digging into self-supervised monocular depth estimation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 3827–3837.

[71] R. Peng, R. Wang, Y. Lai, L. Tang, and Y. Cai, "Excavating the potential capacity of self-supervised monocular depth estimation," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15 540–15 549, 2021.

[72] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[73] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The unreasonable effectiveness of deep features as a perceptual metric," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 586–595.

[74] Y. Zhang and J. Wu, "Video extrapolation in space and time," in *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVI*. Springer, 2022, pp. 313–333.

[75] OpenCV, "StereoSGBM Class Reference," Online: https://docs.opencv.org/3.4/d2/d85/classcv_1_1StereoSGBM.html, 2021, accessed: 2023-05-02.

[76] Q. Huynh-Thu and M. Ghanbari, "The accuracy of psnr in predicting video quality for different video scenes and frame rates," *Telecommun. Syst.*, vol. 49, no. 1, p. 35–48, jan 2012. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1007/s11235-010-9351-x

[77] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

VII. Appendix

A. Ablation: Augmentation Techniques

During the view synthesis experiments on the KITTI dataset, four augmentation techniques are applied to the images the architectures are trained on. These techniques include image flipping and color augmentation from Godard et al. [46], data grafting from Peng et al. [71], and bi-directional training. This ablation study performs training regimes that exclude each of these techniques. Instead of bi-directional training, the architecture is trained from left to right. This means that the layer adding the token at the bottleneck is also removed. The LDI split Eigen et al. [44] is used. However, a slight adjustment is made to the image size. The LDI split removes 5% of the edges and results in images of 128×384. This size is suitable for the hierarchical Swin architecture with a window size of 8. As the edges are removed after the architecture produces a result, the images must be padded to a width of 512 pixels. This increased size is not ideal for training. Therefore the images are directly scaled to 128×384 and trained on this size. This results in Table VI.

At the bottom of the table, the network's performance, including all techniques, is shown, which is the benchmark the ablation study is compared against. Looking at the results, training without color augmentation reduces the performance of the architecture the most. No image flipping follows closely after. Prediction of the right image from the left image even improves the performance. This is expected as this is a more specialized task compared to bi-directional predictions. Bi-directional predictions are preferred in the context of obstacle avoidance, as it learns to generalize information in both directions. However, the architecture trained without data grafting also performed better than the benchmark. This indicates its limited value and will not be used for drone navigation.

B. Analysis: Loss Function

During the view synthesis experiments on the KITTI dataset, the Smooth L1 Loss was used as the loss function for training. This function combines the more popular L1 loss (2), and L2 loss (3), in one function (4). The loss function is studied as well. Smooth L1 Loss is seen less in the literature compared to the more popular L1 Loss and L2 Loss. Similar to the study of the augmentation techniques, the split is trained by scaling to 128×384 without removing the edges, and the results are shown in Table VII.

$$L1\left(y, \hat{y}\right) = \sum_{i=1}^{N} \left|y_i - \hat{y}_i\right| \tag{2}$$

$$L2\left(y, \hat{y}\right) = \sum_{i=1}^{N} \left(y_i - \hat{y}_i\right)^2 \tag{3}$$

$$L1_{smooth}\left(y, \hat{y}\right) = \begin{cases} \frac{1}{2}\left(y - \hat{y}\right)^2 & \text{if} |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{4}$$

TABLE VI: Comparison of the view synthesis performance of the Swin (U-Net) architecture without certain augmentation techniques on KITTI with the LDI split and two generated datasets with AvoidBench for two baselines.

Split	Augmentation	LPIPS ↓	SSIM↑	PSNR↑
LDI split train 128 × 384 test 128 × 384 150 Epochs	No Color Augmentation	0.169	0.777	20.7
	No Image Flipping	0.164	0.782	20.7
	Only Left-to-Right	<u>0.148</u>	**0.800**	**21.0**
	No Grafting	**0.145**	<u>0.792</u>	**21.0**
	All techniques	0.163	0.790	20.8

TABLE VII: Comparison of the view synthesis performance of the Swin (U-Net) architecture with different loss functions on KITTI with the LDI split and two generated datasets with AvoidBench for two baselines.

Split	Loss	LPIPS ↓	SSIM↑	PSNR↑
LDI split train 128 × 384 test 128 × 384 150 Epochs	L1 Loss	0.198	0.778	21.1
	L2 Loss	0.190	0.782	21.0
	Smooth L1 Loss	0.163	0.790	20.8

The performance of the network trained with the Smooth L1 Loss on the LPIPS metric significantly improved compared to the network trained on L1 Loss and L2 Loss. Likewise, the performance is improved on the SSIM metric and only slightly reduced on the PSNR metric.

C. Ablation: Features for Navigation

During the navigation experiments, seven features were always provided to the RL agent as they were deemed necessary for path planning. These were the relative position to the target in the body frame $\left(x_t^b, y_t^b, z_t^b\right)$ and the velocity and the yaw rate in the body frame $\left(v_x^b, v_y^b, v_z^b, \omega_z^b\right)$. Here it is studied whether all these features were necessary for the path planning tasks. This experiment will be performed on the network with the monocular-to-stereo view synthesis feature extractor, shown in Table VIII.

Fascinating is the improved navigational ability by providing randomized values instead of z_t^b and ω_z^b. Such an improvement is not feasible if the RL agent does not know how to use these features. Instead, the drone's behavior in the evaluation should differ significantly from the behavior of the drone in training. During training, the actions the drone receives are stochastic, while in evaluation, the actions are deterministic. As the target is on the same height as the spawn position, there is little need for moving up and down, but due to the stochastic nature in training, z_t^b will fluctuate a bit around zero. Therefore, the agent does not learn a strong policy regarding this and will likely output actions for v_z^b equal to zero, which might not excite the network correctly. Besides, it should be noted that the features in the direction of x are the most critical navigational features for the performance of this model.

D. Analysis: Stereo Setup

During the path planning experiments, a particular stereo setup was decided on. This setup was a trade-off of potential performance gains and computational power. It was decided to use a resolution of 64 × 64, whereas the selected baseline was 0.48m. Shorter baselines are also trained to test the importance to the navigational performance potential and the convergence speed of such a large baseline. Firstly, a realistic real-life setup is proposed. This setup has both cameras aligned to the center of the drone and a baseline of 0.12m. A second setup is proposed that doubles this baseline but aligns the left camera to the center. Finally, a setup is proposed that reduces the image resolution to 32 × 32. Fig. 12 shows the mission progress over time, and Table IX shows the reward, mission progress, and success rate after training.

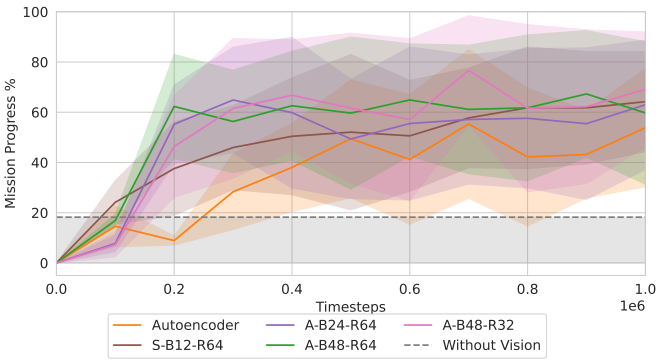


Fig. 12: Mean mission progress (± standard deviation) for RL agent with various camera setups, averaged over 4 training sessions, 10 evaluation runs per session, every 10^5 timesteps in the indoor environment. In the labels, "A" denotes an asymmetric camera, "S" denotes a symmetric camera, "B" followed by a number indicates the baseline in centimeters, and "R" followed by a number refers to the resolution (e.g., "R64" signifies a 64x64 resolution).

Fig. 12 indicates a correlation between the length of the baseline and the convergence speed. Larger baselines converge quicker compared to smaller baselines. It is, however, expected that increasing the baseline further than 0.48cm will start to reduce the performance, as the improvements are already reducing from 0.24cm to 0.48cm. Interestingly, a reduction in

TABLE VIII: Comparison of the navigational performance of an RL agent with the monocular-to-stereo view synthesis feature extractor with masked path planning features

Masked Path Planning Feature	Reward ($\mu \pm \sigma$)	Mission Progress ($\mu \pm \sigma$)	Success Rate
Feature: x_t^b	18.4 ± 15.8	56.1% ± 25.4%	17.5%
Feature: y_t^b	23.9 ± 15.0	64.9% ± 23.5%	27.5%
Feature: z_t^b	25.4 ± 15.6	65.9% ± 23.5%	**32.5%**
Feature: v_x^b	20.2 ± 14.1	61.4% ± 22.2%	17.5%
Feature: v_y^b	23.4 ± 16.1	63.7% ± 25.9%	27.5%
Feature: v_z^b	21.3 ± 17.4	59.1% ± 24.7%	25.0%
Feature: ω_z^b	**27.5 ± 14.7**	**71.9% ± 16.9%**	**32.5%**
All Path Planning Features	21.5 ± 15.2	59.7% ± 28.3%	25%

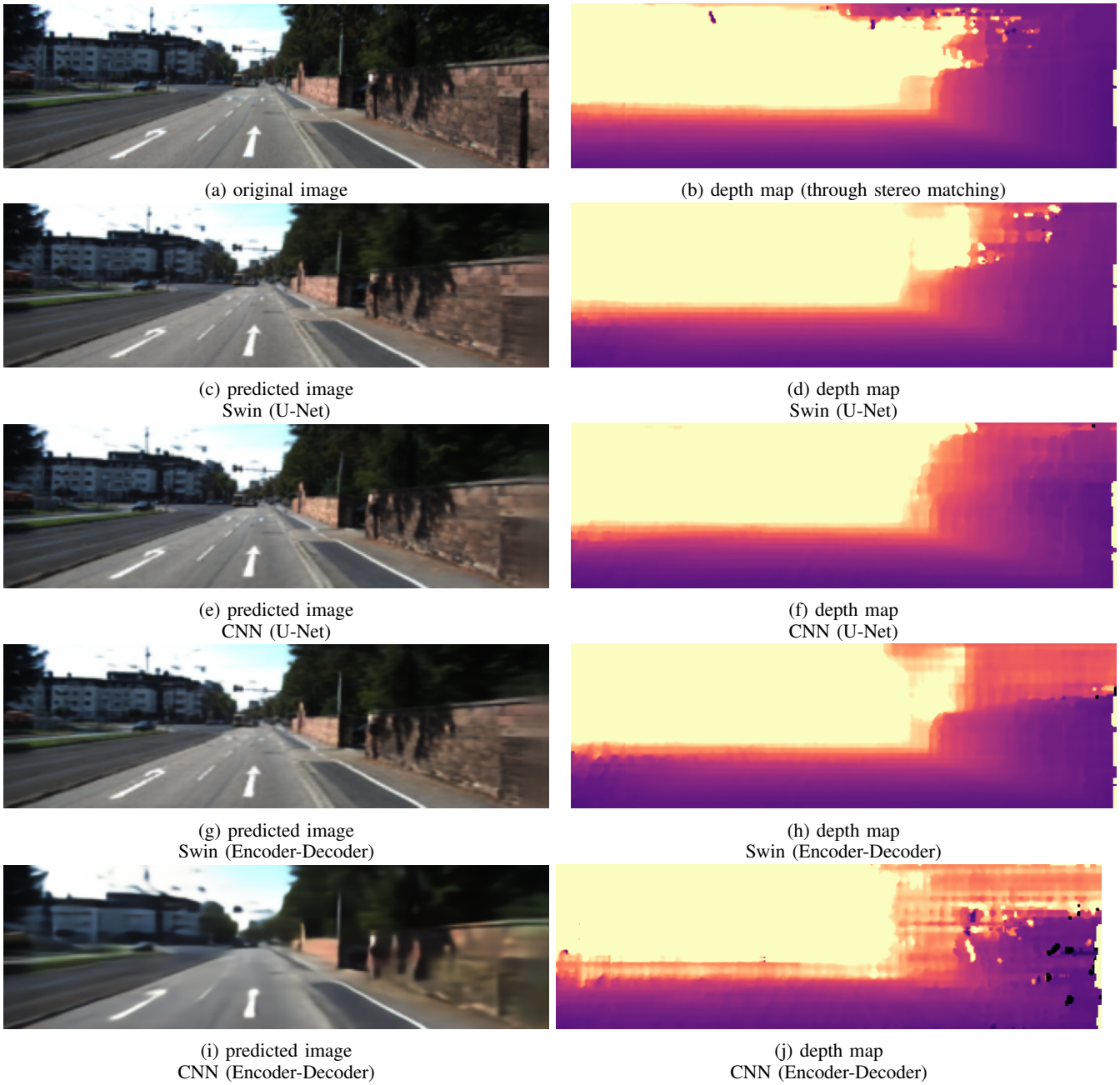
feature space due to the reduction in image size to 32×32 does not increase the convergence speed but seems to reduce it slightly.

E. Additional: Image from KITTI

Additional examples of monocular-to-stereo image view synthesis predictions and depth maps obtained through stereo matching with the original images of the KITTI dataset are shown for Swin (U-Net), CNN (U-Net), Swin (Encoder-Decoder) and CNN (Encoder-Decoder) in Figs. 13–22

TABLE IX: Comparison of the navigational performance of the RL agent with the monocular-to-stereo view synthesis feature extractor for different stereo setups

Symmetrical	Baseline	Resolution	Reward ($\mu \pm \sigma$)	Mission Progress ($\mu \pm \sigma$)	Success Rate
Monocular	-	64×64	15.1 ± 11.4	53.8% ± 23.5%	7.5%
Symmetric	0.12m	64×64	19.2 ± 11.3	64.2% ± 19.9%	10%
Asymmetric	0.24m	64×64	21.3 ± 14.2	63.0% ± 25.7%	20%
Asymmetric	0.48m	32×32	**23.4 ± 15.3**	**69.1% ± 22.8%**	20%
Asymmetric	0.48m	64×64	21.5 ± 15.2	59.7% ± 28.3%	**25%**



(a) original image

(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)

(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)

(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

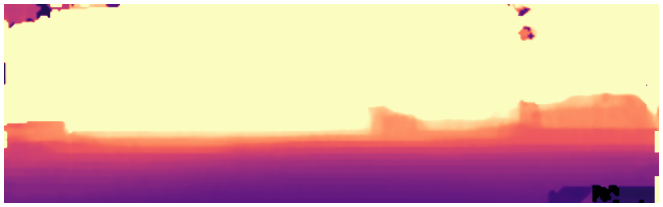
(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)

(j) depth map
CNN (Encoder-Decoder)

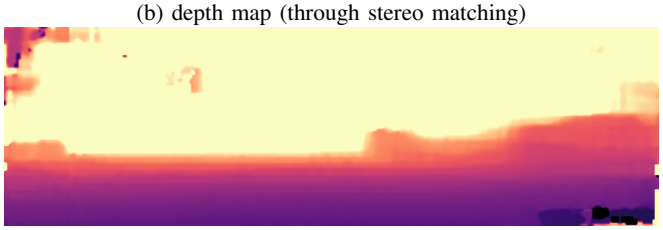
Fig. 13: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

(a) original image



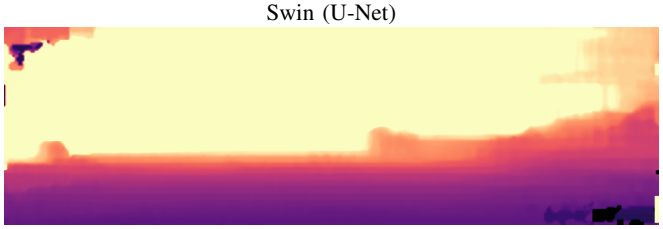
(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)



(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)



(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)

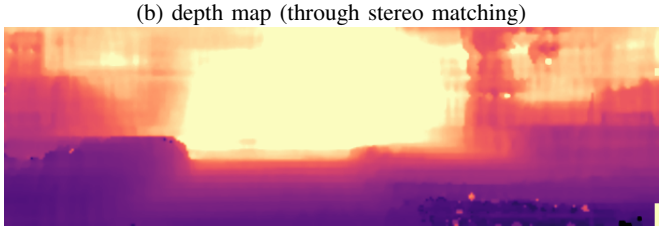
(j) depth map
CNN (Encoder-Decoder)

Fig. 14: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

(a) original image

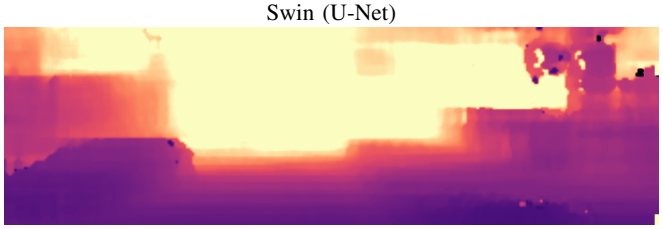
(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)



(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)



(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

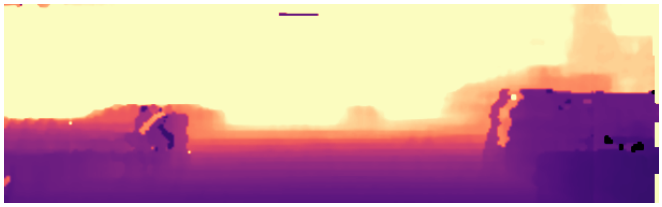
(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)

(j) depth map
CNN (Encoder-Decoder)

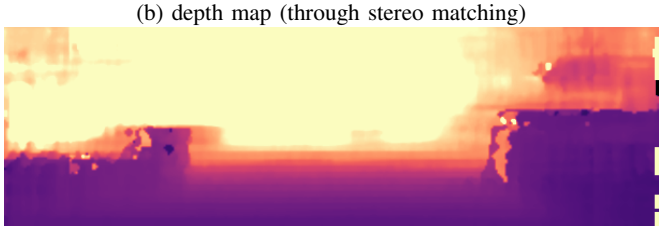
Fig. 15: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

(a) original image



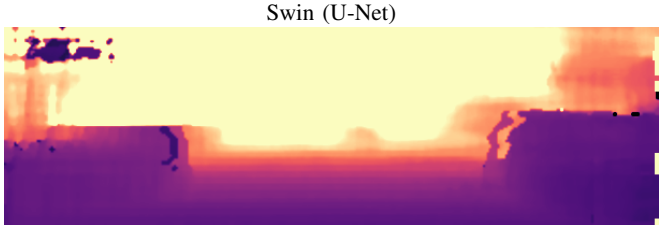
(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)



(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)



(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)

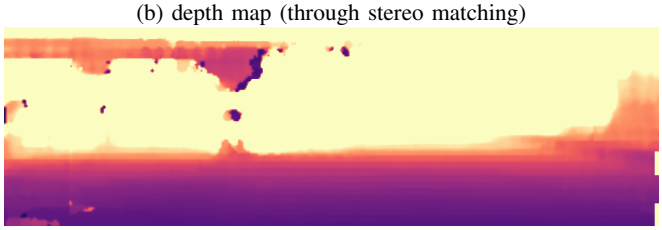
(j) depth map
CNN (Encoder-Decoder)

Fig. 16: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

(a) original image

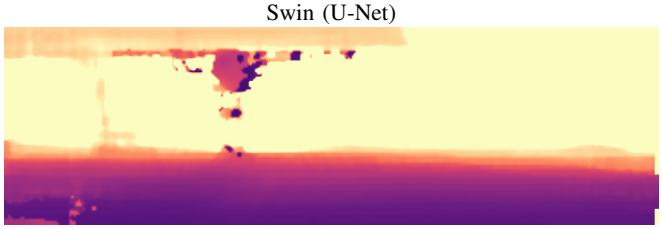
(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)



(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)



(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)

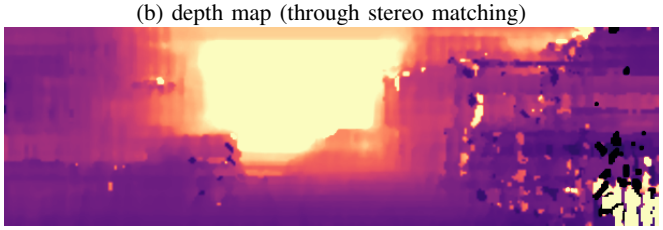
(j) depth map
CNN (Encoder-Decoder)

Fig. 17: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

(a) original image

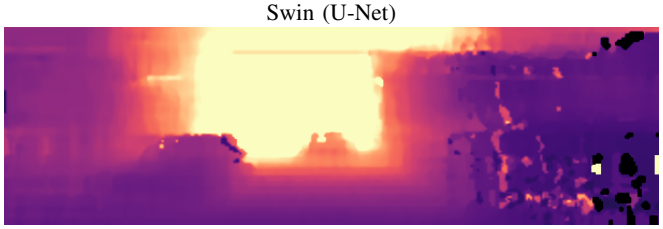
(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)



(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)



(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)

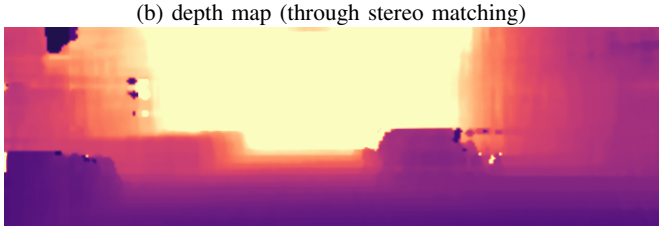
(j) depth map
CNN (Encoder-Decoder)

Fig. 18: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

(a) original image

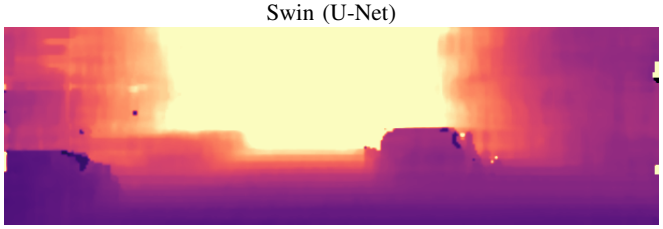
(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)



(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)



(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

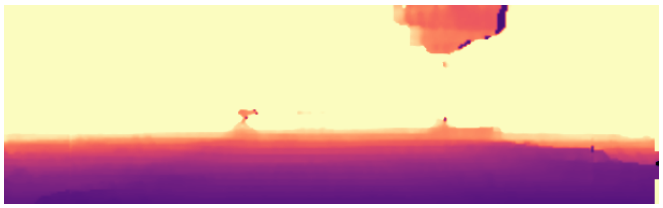
(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)

(j) depth map
CNN (Encoder-Decoder)

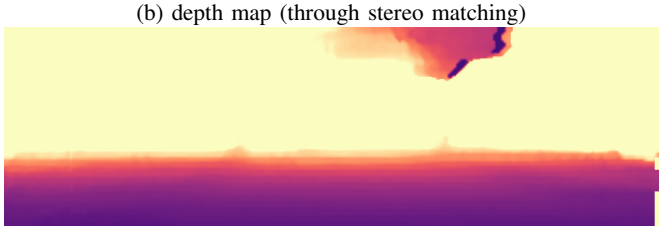
Fig. 19: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

(a) original image



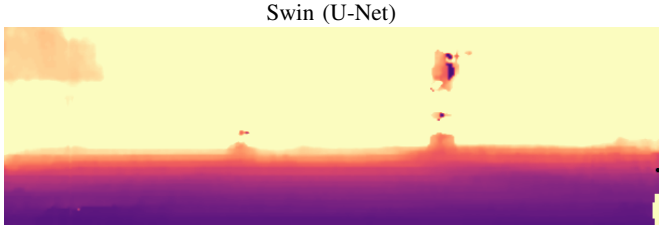
(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)



(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)



(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)

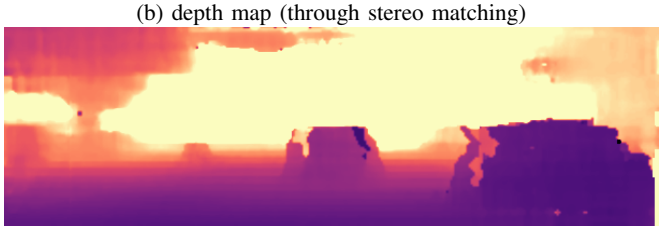
(j) depth map
CNN (Encoder-Decoder)

Fig. 20: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

(a) original image

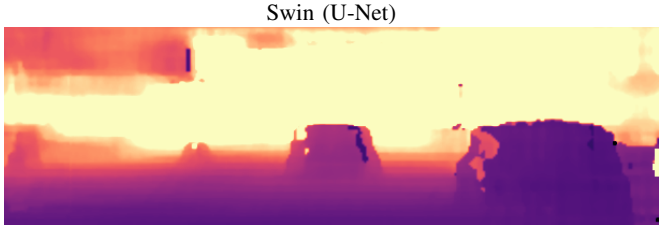
(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)



(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)



(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)

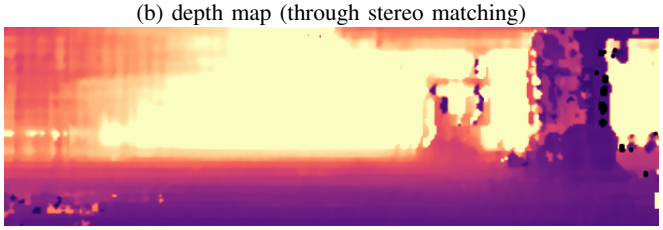
(j) depth map
CNN (Encoder-Decoder)

Fig. 21: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

(a) original image

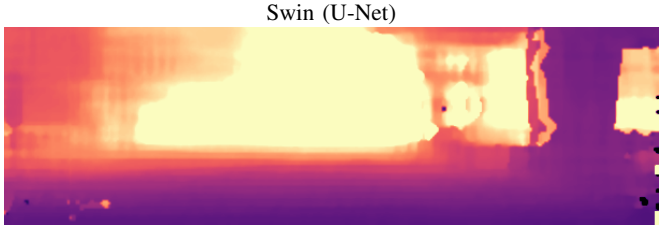
(b) depth map (through stereo matching)

(c) predicted image
Swin (U-Net)



(d) depth map
Swin (U-Net)

(e) predicted image
CNN (U-Net)

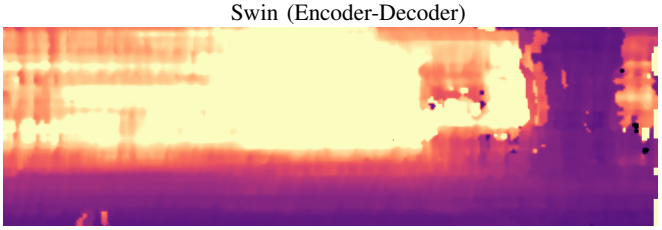


(f) depth map
CNN (U-Net)

(g) predicted image
Swin (Encoder-Decoder)

(h) depth map
Swin (Encoder-Decoder)

(i) predicted image
CNN (Encoder-Decoder)



(j) depth map
CNN (Encoder-Decoder)

Fig. 22: Examples of predicted images for the different architectures and their depth maps obtained through stereo matching with the input image on the KITTI dataset

Part II

Literature Study

2

Geometry-Free Monocular-to-Stereo Image View Synthesis

Following the extensive introduction, this chapter will discuss all aspects surrounding the geometry-free monocular-to-stereo image view synthesis. First, a literature overview is composed, which consists of three themes, followed by a discussion to bring everything together. The first theme delves into the historical context of computer vision in Subsection 2.1.1. It explains general computer vision concepts, what neural networks are and how they came onto the scene, and what kind of hardware is used in computer vision. The second theme follows the historical context by providing general developments within computer vision and artificial intelligence in Subsection 2.1.2. Current relevant architectures in computer vision, different kinds of optimisers, training schedules and loss functions are explained. The limitations and potential of the state-of-the-art networks on both the hardware and software side are discussed. The final theme is discussed in Subsection 2.1.3, where the fields of research adjacent to the research topic of geometry-free monocular-to-stereo image view synthesis are analysed. Research has shown up fairly similar to this topic in the last few years. The merging of different subdomains and the emergence of better architectures enabled a trend that converges to the creation of the subdomain of geometry-free monocular-to-stereo image view synthesis. The context of this convergence, the datasets that benchmark the performance of the algorithms, as well as state-of-the-art practices, are discussed. Subsection 2.1.4 brings these three themes together in a short discussion. After the literature overview, the research plan for this phase is proposed in Section 2.2. The research plan contains the goals, proposes a method and architecture and explains the execution of the experiments. It provides an overview of the datasets and metrics that show the performance and hypothesis about possible results. Besides, it already shows preliminary results and discusses the impact of these results on the application of the neural network to deep reinforcement learning.

2.1. Literature Overview

Significant trends and subdomains are dismantled into sections and structured chronologically to bring structure. The terms to find the research include, but are not limited to, the following: *CNN, Convolution, Deep, Depth, End-to-end, Estimation, Geometry-free, Image, Generation, Monocular, Multi, Network, Neural, Prediction, Self-Supervised, Single, Stereo, Synthesis, Supervised, Swin, Transformer, U-Net, Unsupervised, View, Vision*

2.1.1. Historical Context of Computer Vision

Computer vision is the field of study that enables computers to derive meaningful information from digital images, videos and other visual inputs and use it to take appropriate actions or make valuable recommendations, simulating and automating human capabilities. As an extension of robotics and artificial intelligence, computer vision has a rich history in pop culture dating back to the *1950s* (Asimov, 1950).

It was not until the *early 1970s* that computer vision became a topic of interest in the research field. Similar to how it was utilised in science fiction, the research field viewed computer vision as the vi-

sual perception component of a larger goal to create robots with intelligent behaviour by mimicking human intelligence. Until the *early 2000s* the algorithms were mathematical and became more and more complicated. However, from this point on machine learning techniques became essential to the computer vision domain. This change was made possible due to two developments. The first one was the continued increase in the computational power of computers. The exponential growth had been happening for decades and was already recognised by Gordon Moore, (Moore et al., 1965), in 1965. The second development was that due to the internet, large amounts of data, often partially labelled, could be shared by and was available to everybody. Machine learning techniques could be trained on this data without the use of careful human supervision.
In the *2010s*, this trend pushed through and started dominating the entire field. Large-scale, high-quality annotated datasets became available in the form of ImageNet, by (Deng et al., 2009), Microsoft COCO, by (T.-Y. Lin et al., 2014) and LVIS, by (Gupta et al., 2019). Each had sufficient labelled data so that solutions based entirely on machine learning came to exist. It also saw a deep neural network become state-of-the-art for the first time in the form of AlexNet, by (Krizhevsky et al., 2012). After which many neural networks would follow.

Neural Networks
The idea of a machine being able to emulate the human brain is ancient. (Fitch, 1944) suggested mimicking the behaviour of neurons by using binary thresholds to Boolean logic. From this, (Rosenblatt, 1958) developed Rosenblatt's perceptron, a single-layer perceptron. The architecture is shown in Figure 2.1. A vector of inputs is multiplied by a series of corresponding weights, whose sum plus a bias is compared to a threshold. If the sum exceeds the threshold, the "neuron" fires and takes the activated value; if not, the deactivated value is taken. Equation 2.1 is the formula for the first step and Equation 2.2 describes how the node is activated. Rosenblatt's perceptron consists of a single node.

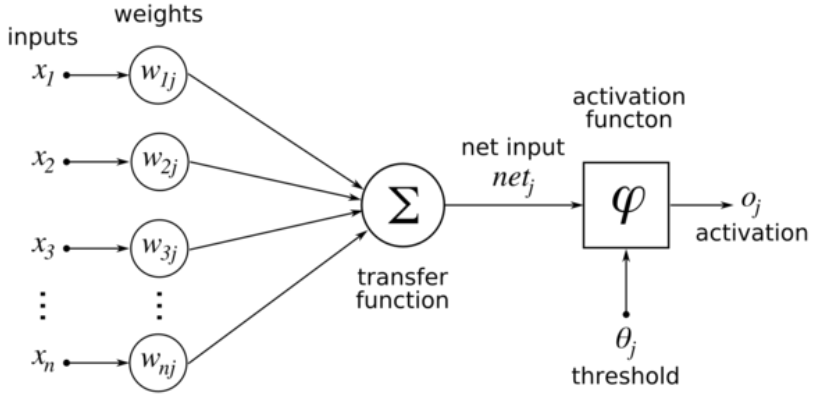


Figure 2.1: Architecture of Rosenblatt's perceptron[1]

$$\text{net input} = \sum_{i=1}^{n} w_i x_i + b = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b \tag{2.1}$$

$$\text{activation} = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i + b \geq 0 \\ 0 & \text{if } \sum_{i=1}^{n} w_i x_i + b < 0 \end{cases} \tag{2.2}$$

A multilayer perceptron (**MLP**), also known as a position-wise fully connected feed-forward network, has many nodes per layer, each activated separately. These layers can also be stacked one after the other, meaning that the first layer's output becomes the second layer's input. An example of such an architecture is Figure 2.2, where the layers between the input and output layers are called hidden layers. Historically the activation functions of neurons in neural networks are not binary but are sigmoids which can have values between 0 and 1. The outputs are either part of a classification or a regression, and whether they are accurate is evaluated by a cost or **loss function**. The model adjusts its weights and bias based on the outcome to reduce the loss.

[1]https://commons.wikimedia.org/wiki/File:Rosenblattperceptron.png [Accessed 2 November 2022]

In Figure 2.3, a loss function for a weight is shown. At a certain weight, the loss function has a global minimum. For the optimisation to this minimum, gradient descent is used. A gradient is the derivative of the loss with respect to the weight, $\frac{dL}{dw}$. To decrease the loss, the weight should move in the direction the gradient is negative. The learning rate determines by how much the weights should be moved. This process of determining the direction in which the weights have to move is often not as simple. For instance, a loss function of a neural network does not only have a global minimum but many local minima as well. Besides, neural networks do not have a single layer of nodes with weights but have multiple layers that need to be optimised. The weights have to be optimised for all nodes, so a gradient needs to be obtained for all nodes: $\frac{dL}{dW}$ for $W \in \mathbb{R}^{nm}$, where nm is the total number of nodes. The loss is calculated as a function of the outputs: $L = f\left(\mathbf{y}_{output}\right)$. The outputs follow from activations as a result of the summation of the product of the weights and outputs of the previous layers: $\mathbf{y}_{output} = a_{output}\left(\mathbf{w}_{j:output}, \mathbf{y}_j\right)$ for $\mathbf{w}_{j:output}, \mathbf{y}_j \in \mathbb{R}^n$, where j is the last hidden layer and n the number of nodes within the last hidden layer. By combining both functions, the derivative of the loss with respect to the weights of the last hidden layer can be obtained. The gradients of weights in earlier layers can be obtained similarly. This process is called backpropagation.
Batch Gradient Descent takes the average of the gradients over the *entire* dataset and uses the mean gradient to update the parameters. This is impossible for large datasets as the entire dataset has to be saved in the memory simultaneously. Stochastic Gradient Descent (**SGD**) updates the weights for each training sample and can therefore be used on large datasets. The disadvantage is that the network is trained to generalise solutions applicable to all inputs and targets. Therefore, the weights should not be tuned based on individual training examples but on several examples. This is called Mini Batch Gradient Descent. In this case, the number of examples per forward and backward pass is called the **batch size**. A pass is called an **iteration** and an **epoch** is when all training examples go through the network.

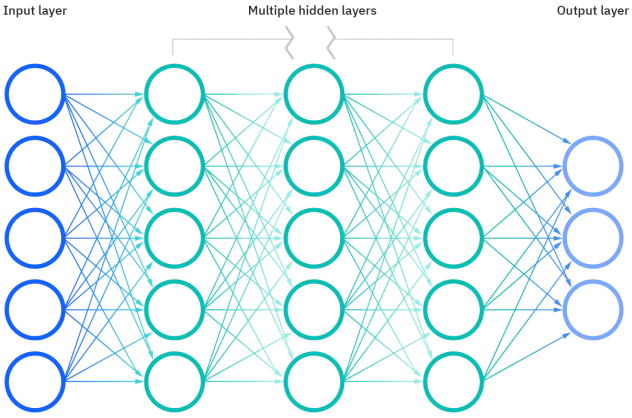


Figure 2.2: Architecture of deep neural network[2]

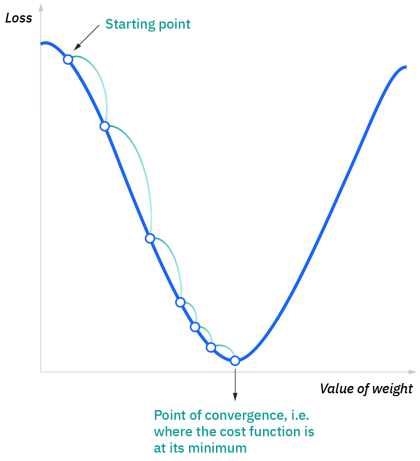


Figure 2.3: Gradient descent on a simple loss function[2]

Software

As using deep learning became more prominent, the implementation of neural networks became more standardised, and frameworks were designed to facilitate their usage. One of the most prevalent frameworks is PyTorch, (Paszke et al., 2017). This framework was designed primarily for the programming language python. Besides providing a framework for neural networks, PyTorch has a class called Tensor, which stores and operators homogeneous multidimensional rectangular arrays of numbers. These Tensors can be operated on GPUs, speeding up the learning process. It also provides an autograd module, which handles the backpropagation and computes the gradients, the most common optimisers, and standard layers and tools to develop the neural networks.

[2]https://www.ibm.com/cloud/learn/neural-networks [Accessed 2 November 2022]

Hardware
In general, more extensive networks are better able to perform complicated tasks. For more extensive networks, better hardware is required. As a result, the choice in architecture design depends on the hardware available. Not only is the application of monocular-to-stereo image view synthesis trained and run on a server relevant. The follow-up, applying reinforcement learning to a drone in real-time, has to be considered. From this, it follows that the sensors used to receive the input, the hardware to train the neural network and the hardware to use the neural network in real time are most relevant. The motivation to use stereo images instead of depth information is a result of the hardware. LIDAR, short for "laser imaging, detection, and ranging", directly measures depth. It targets an object or a surface with a laser and measures the time for the reflected light to return to the receiver. As a result, it can determine ranges. Even more budget-friendly LIDAR sensors[2], become a significant expense compared to other drone components and cameras. As the navigation task will first be performed in simulation, the network does not have to be designed with a specific drone and GPU in mind. An architecture will be developed that can be transferred to a real drone. For this thesis, the student is limited to the NVIDIA GeForce GTX 1080Ti (2x) to run experiments on.

2.1.2. General Developments within Computer Vision

Activation Functions
A couple of different activation functions are used in machine learning and computer vision. The activation function introduces linearity and has two general applications in a neural network. The first application is to activate neurons and enable the flow of information to other neurons, while the second is to activate neurons and output the prediction. For the task of regression, the target often has values within a range, and in such a case, the **sigmoid** activation function or the tangent hyperbolic (**tanh**) function is used. These functions are plotted in Figure 2.4 and scale all outputs between (0,1) and (-1,1), respectively. Both are S-shaped and push the input values to the end of their curves.
These activation curves, however, are not useful for the neurons in a deep neural network. Bounded activation functions become less effective after backpropagating through each hidden layer as the gradients vanish. Which therefore prevents changes in the values of the weights. (Nair & Hinton, 2010) introduced a more suitable activation function named Rectified Linear Unit (**ReLU**). They proposed a gated activation that lets values through when they are larger than zero. For years ReLUs were part of state-of-the-art architectures. (Clevert et al., 2016) introduced Exponential Linear Unit (**ELU**), with an activation curve similar to ReLU, but differs in that it is non-linear and can output negative values. This activation curve sometimes increases training speed. To prevent overfitting, network designers often include stochastic regularisers such as applying dropout, which can lead to better accuracy. The dropout randomly alters the activation decision through stochastic zero multiplication. The dropout acts irrespective of the activation function, which limits the potential. (Hendrycks & Gimpel, 2016) recognised this and proposed the Gaussian Error Linear Unit (**GELU**). Similar to dropout the GELU stochastically multiplies inputs by zero or one, however, it keeps dependency upon the input value. GELU is approximated with Equation 2.3 to decrease computational time. It is currently used in most state-of-the-art architectures. Figure 2.5 shows the activation function of ReLU, GELU and ELU.

$$GELU(x) \approx 0.5x \left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right]\right) \tag{2.3}$$

Normalisation
All layers try to learn a function that maps the input to an output within a deep neural network. They receive their inputs from previous layers, which are also learning to map their input to an output. As a result of training, their output and distribution of outputs change, resulting in the next layer receiving an input with a different distribution. This distribution is new to the layer, and the layer can not apply the learned behaviour. The change in distribution is called an internal covariance shift. **Batch normalisation**, proposed in (Ioffe & Szegedy, 2015), tries to solve this by standardising the distribution of inputs per neuron. Standardisation shifts data to a mean of 0 and a standard deviation

[2]https://www.garmin.com/en-US/p/557294 [Accessed 2 November 2022]

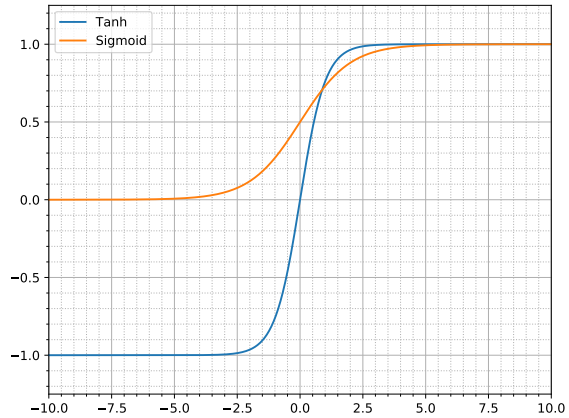


Figure 2.4: Tanh and Sigmoid activation functions

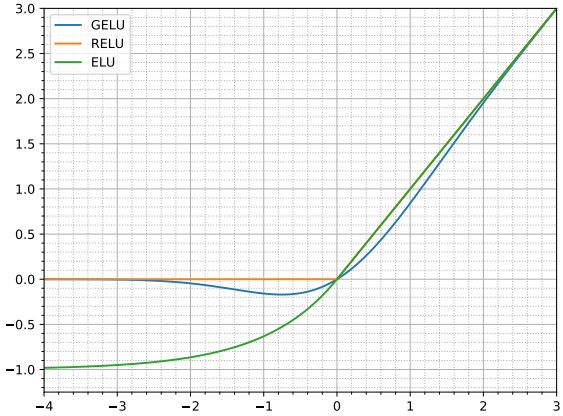


Figure 2.5: GELU, RELU and ELU activation functions

of 1. In the case of batch normalisation, the output of the previous activation layer is subtracted from the batch mean and then divided by the standard deviation of the batch. This batch-normalised layer than is transformed by two trainable parameters, a mean and a standard deviation. As such the optimum distribution of each hidden layer can be chosen. **Layer normalisation**, introduced by (Ba et al., 2016), is similar in its approach to solving this distribution problem by standardising the distribution of inputs per layer. Again the optimum distribution is chosen with trainable parameters through gradient descent.

Optimisers

Optimising the weights of a neural network is done based on gradient descent. Better algorithms to optimise the loss function have been developed compared to previously presented methods. Figure 2.3 shows the loss function. It should be noted that the loss function drops rapidly with large gradients but does not at smaller gradients. To prevent the convergence of the loss function to local minima, momentum is used. This means using the average over previous gradients to determine the impact of the learning rate. Especially useful is the exponential moving average, as this assigns greater weight to more recent values. The method RMSProp, by (Hinton et al., 2012), uses the exponentially weighted average method to the second moment of the gradients. It, however, lacks a bias-correction term, which is problematic in the case of sparse gradients. An algorithm that does work well for sparse gradients is AdaGrad (Duchi et al., 2011). (Kingma & Ba, 2015) combined the advantages of both and created Adaptive Moment Estimation (**Adam**). This achieved state-of-the-art for many years. (Loshchilov & Hutter, 2019) decoupled the optimal choice of weight decay factor from the learning rate setting for Adam. In doing so, creating the optimiser **AdamW** improved the generalisation performance of Adam. Besides using an optimiser, the learning rate is often scheduled as weights should move less significantly after longer training durations. (Loshchilov & Hutter, 2016) shows that a warm restart mechanism accelerates training, which means that after slowly decreasing the learning rate, the learning rate jumps to an increased value followed by a slow decrease in the rate. This is called cosine annealing or a cosine learning rate scheduler.

Losses and Evaluation Metrics

In the case of view synthesis, continuous values are predicted. The goal of the loss function is to regress to a set of target values of continuous nature. The loss function, therefore, should show an error from the prediction $\hat{y}$ to the target y. Two standard functions that compute this difference are the L2 loss and the L1 loss, displayed by Equation 2.4 and Equation 2.5, respectively. The L2 loss is also known as the Squared Error, and aggregating this loss value over an entire dataset gives the Mean of Squared Errors (MSE) cost function. The L1 loss is, however, known as the Absolute Error and aggregating this loss value over an entire dataset gives the Mean of Absolute Errors (MAE). In general, the L2 loss converges faster and oscillates less for slight differences than the L1 loss. However, it tends to over-smooth an image and has less steady gradients at more significant differences. For the view synthesis task, the L1 loss is often used. The features advantageous to both losses can

also be combined in Equation 2.6, known as the Smooth L1 Loss. The loss is L1 for higher values, and for lower values, the loss is L2.

$$f(y, \hat{y}) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.4}$$

$$f(y, \hat{y}) = \sum_{i=1}^{N} |y_i - \hat{y}_i| \tag{2.5}$$

$$f(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & \text{if} |y - \hat{y}| < 1 \\ |y - \hat{y}| - \frac{1}{2} & \text{otherwise} \end{cases} \tag{2.6}$$

Although these functions are excellent at converging the loss and, as a result, help generate images close to the targets, more sophisticated equations are necessary for evaluation. Noise is not beneficial for an image and the Peak Signal-to-Noise Ratio (PSNR), in Equation 2.7, represents the amount of noise compared to the highest RGB values. PSNR is not a suitable metric for comparing the quality and performance over different scenes, states (Huynh-Thu & Ghanbari, 2012). The Structural Similarity Index Measure (SSIM), by (Wang et al., 2004), measures the similarity of the predicted image to the target image by analysing their distributions, with Equation 2.8. L is the dynamic range of the pixel values, $k_1 = 0.01$, and $k_2 = 0.03$. Multi-Scale SSIM (MS-SSIM), by (Wang et al., 2003), subsamples the images to multiple stages to perform the SSIM at multiple scales. The Frechet Inception Distance (FID), by (Heusel et al., 2017), is also used to assess the image quality of generative models. This compares their multivariate normal distributions, estimated by Inception v3 features. Finally, the Learned Perceptual Image Patch Similarity (LPIPS) measures perceptual similarity between images. (R. Zhang et al., 2018) found that the similarity in activations of images through a pretrained neural network is an excellent indication of perceptual similarity for the human eye. By default, LPIPS uses a pre-trained AlexNet, by (Krizhevsky et al., 2012), or VGG, by (Simonyan & Zisserman, 2015), as the neural network for the comparison of activations between images.

$$\text{PSNR}(y, \hat{y}) = 10 \log_{10} \left(\frac{\max(y)^2}{(y - \hat{y})^2} \right) \tag{2.7}$$

$$\text{SSIM}(y, \hat{y}) = \frac{\left(2\mu_y \mu_{\hat{y}} + (k_1 L)^2\right)\left(2\sigma_{y\hat{y}} + (k_2 L)^2\right)}{\left(\mu_y^2 + \mu_{\hat{y}}^2 + (k_1 L)^2\right)\left(\sigma_y^2 + \sigma_{\hat{y}}^2 + (k_2 L)^2\right)} \tag{2.8}$$

Transformers

Attention Is All You Need
by (Vaswani et al., 2017)

Natural Language Processing is the branch of computer science that uses machine learning to understand the structure and meaning of text. Language modelling and machine translation are large subdomains within NLP. Neural sequence transduction models are used to tackle these tasks. At the time of the publication of this paper, long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) and gated recurrent units (GRU) (Chung et al., 2014) were the state-of-the-art approaches. These recurrent networks are combined with an attention mechanism to improve performance. (Vaswani et al., 2017) proposes the Transformer architecture, which uses an attention mechanism to draw global dependencies between input and output. A significant benefit is that compared to recurrent models, this allows for more parallelisation.
For transduction learning, the output is often shifted to the right, therefore learning to predict the following word. This prediction requires both the input, as well as previous outputs. Shared with most competitive neural sequence transduction models, the model this paper describes also has an encoder-decoder structure. The encoder and decoder are shown on the left and right side, respectively, in Figure 2.6. The input tokens are first embedded with learned embeddings to d_{model}-dimensional vectors and positionally encoded as the sequential order is otherwise unknown by the

model. Then they go through N identical Transformer layers, each consisting of a multi-head self-attention (MSA) mechanism and an MLP. A residual connection around both parts followed by layer normalisation is employed. The decoder requires an embedded output with positional encoding and has N identical layers. The output first goes through a masked MSA mechanism followed by a MSA mechanism which uses the outputs of the encoder as well. Finally, an MLP completes the layer. After these layers, the output is linearised, and a SoftMax is applied to output probabilities.
The attention function described here maps a query and a set of key-value pairs to an output. To do so, a compatibility function of a query with a corresponding key assigns a weight to each value. The weighted values are then added up to an output. The queries, keys, and values are simultaneously packed together into matrices $Q, K \in \mathbb{R}^{\mathbf{x} \times d_k}$ and $V \in \mathbb{R}^{\mathbf{x} \times d_v}$ respectively. d_k is the size of the dimensions of the queries and keys, d_v is the size of the dimensions of the values and $\mathbf{x}$ is the data. Equation 2.9 formulates the method, and Figure 2.8 visualises it in a block diagram. The scaling factor $\frac{1}{\sqrt{d_k}}$ is there to prevent the SoftMax function from only using extremely small gradients as a result of large d_k. In the MSA layer, the queries, keys and values are linearly projected h times in parallel. Equation 2.9 is then applied to each of the h different projected versions. Their outputs are concatenated and linearly projected again, which is visualised in Figure 2.7.

$$\text{Attention}\,(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.9}$$

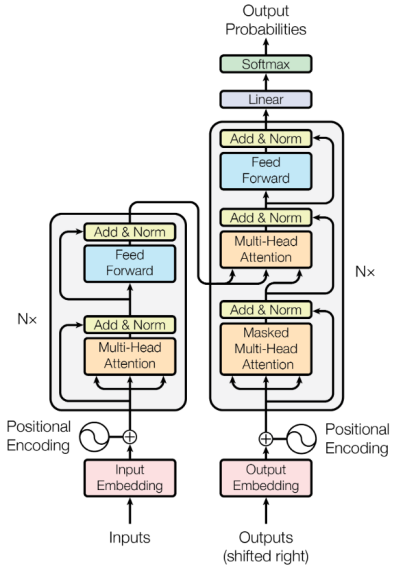


Figure 2.6: Transformer model architecture by (Vaswani et al., 2017)

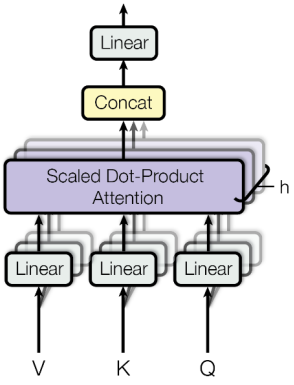


Figure 2.7: Multi-Head Attention mechanism by (Vaswani et al., 2017)

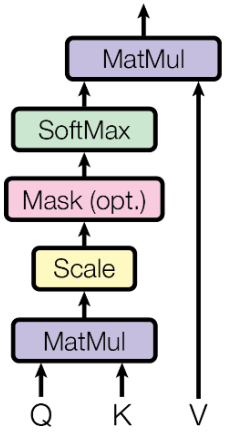


Figure 2.8: Scaled Dot-Product function by (Vaswani et al., 2017)

They achieved state-of-the-art on both WMT 2014 English-to-German and WMT 2014 English-to-French translation tasks and transformers are still state-of-the-art for both tasks[3] [4].

An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale
by (Dosovitskiy et al., 2020)

This paper presents the Vision Transformer (ViT) and follows the architecture of the Transformer used in NLP from (Vaswani et al., 2017) as closely as possible. Figure 2.9 shows the model architecture of the Vision Transformer for image classification. Pixels in images are often significantly larger than words in a text. This paper, therefore, proposes to use the attention mechanism on patches of 16×16 pixels instead of per pixel. The ViT receives images $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$, where H and W make the resolution of the original image and C, the number of channels, is three in the case of RGB. The transformer encoder requires patches to train on, so the images are reshaped to patches $\mathbf{x}_p \in \mathbb{R}^{N \times \left(P^2 \cdot C\right)}$,

[3] https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-german [Accessed 31 October 2022]
[4] https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-french [Accessed 31 October 2022]

where P^2 is the resolution of the image patches (16×16), and N is the resulting number of patches. The patches are embedded similarly to the original Transformer. Again this is combined with an encoded position, as well as a classification label of the image, formulated in Equation 2.10. The patches then go through a series of Transformer layers, identical to the one from (Vaswani et al., 2017) Finally, an MLP outputs a class-label prediction.
This method achieved state-of-the-art on ImageNet, by (Deng et al., 2009). The ILSVRC-2012 ImageNet already contains 1.3M images. The paper describes, however, that as self-attention does not use image-specific inductive biases, it needs to learn these and requires much pre-training on even large datasets. The model was pretrained on ImageNet-21k and JFT, by (Chollet, 2017; Hinton et al., 2015), with respectively 14M and 303M images. The size of the ViT-Base, ViT-Large, and ViT-Huge are also significant, with 86 million, 307 million and 632 million parameters. These models are trained on TPU v3 hardware, which the hardware available for the thesis cannot compete with.

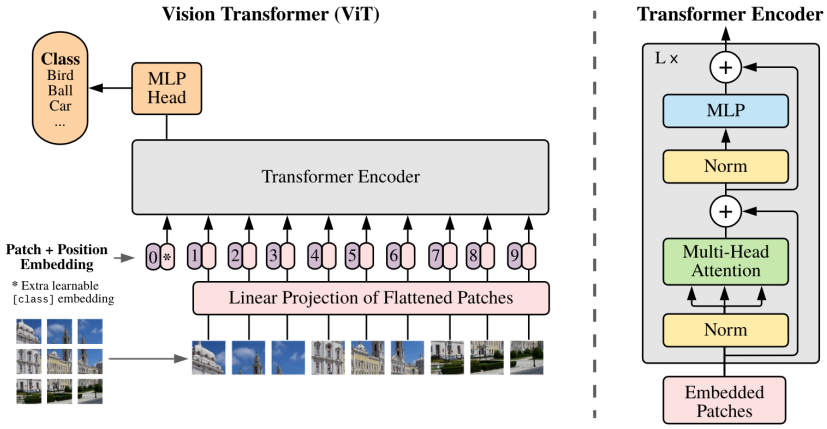


Figure 2.9: Vision Transformer Model architecture for image classification by (Dosovitskiy et al., 2020)

$$\mathbf{z}_0 = \left[\mathbf{x}_{\text{class}}; \mathbf{x}_{p,1}\mathbf{E}; \mathbf{x}_{p,2}\mathbf{E}; \cdots; \mathbf{x}_{p,N}\mathbf{E}\right] + \mathbf{E}_{\text{pos}}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D} \tag{2.10}$$

Swin Transformer: Hierarchical Vision Transformer using Shifted Windows
by (Z. Liu et al., 2021)

This is another fundamental paper showing the significant potential of transformers within computer vision. It addresses the two main problems of ViT: (1) scaling the network with increased image size becomes exponentially expensive computation-wise, which limits the applicability to datasets with high-resolution images and (2) the lack of local attention to images. As such, it proposes a hierarchical Transformer whose representation is computed with Shifted windows (Swin).
Hierarchical structures are seen in CNNs. After a convolutional layer, the input size is quartered (halved horizontally and halved vertically) by a pooling layer. Consequently, the kernels of the following convolutional layer explore portions of the picture that are four times as large, even though the kernel size is the same. By not applying all transformers to patches of the entire image but instead applying them to patches of smaller windows, local attributes can be learned. Merging neighbouring windows and increasing the size of patches lets the network learn regional attributes, and repeating this process lets the network learn global attributes. This structure is shown on the left side of Figure 2.10, while the structure of the ViT is shown on the right side. As this structure enables the network to learn local and global attributes, it can be applied as a general-purpose backbone for image classification and dense recognition tasks.
A shifted window approach is used to counteract the problem that local attributes may bridge neighbouring windows and are therefore overlooked by the network. Within the same hierarchical layer succeeding windows are shifted to form new windows with patches that have not applied attention to each other. This is visualised in Figure 2.11.
In Figure 2.12, the model architecture is shown. Like ViT, the input images are split into patches and treated like tokens. For the input layer, the patch size Swin uses is 4×4, and the RGB values

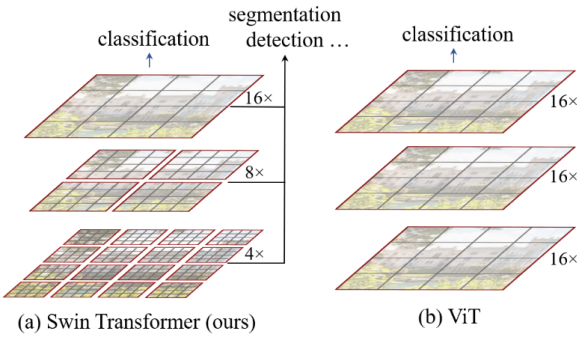


Figure 2.10: Hierarchical feature maps compared to constant feature maps by (Z. Liu et al., 2021)

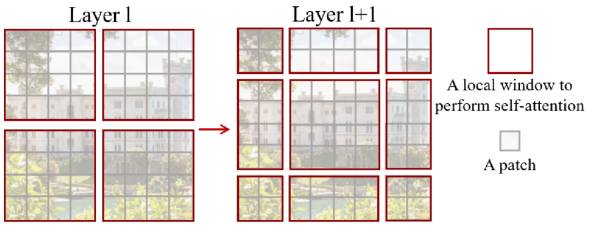


Figure 2.11: Shifted windows approach for self-attention by (Z. Liu et al., 2021)

are concatenated, resulting in a feature dimension per patch of 48. Again similar to ViT, a linear embedding layer is applied. The MSA module of the ViT has a quadratic computational complexity with increased resolution. The exact formula without accounting for SoftMax computation is given in Equation 2.11. With a consistent patch size, increases in height and width result in more patches. The self-attention increases quadratically as all new patches are attended to by all patches. In the case of a multi-head self-attention in non-overlapped windows (W-MSA) module, with a consistent window size, the number of windows will increase linearly with increases in height and width. This formula is given in Equation 2.12 with M being the window size. Besides this change, the Swin Transformer block is identical to the ViT Transformer block.

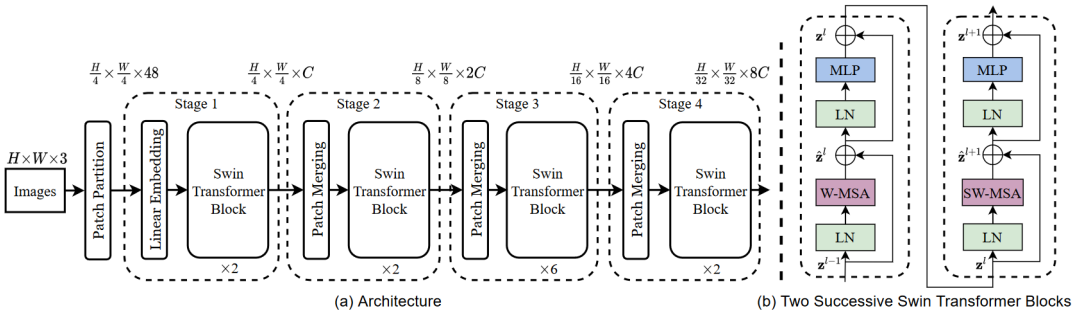


Figure 2.12: Multipurpose Swin Transformer model architecture by (Z. Liu et al., 2021)

$$\Omega\left(\text{MSA}\right) = 4hwC^2 + 2\left(hw\right)^2 C \tag{2.11}$$

$$\Omega\left(\text{W-MSA}\right) = 4hwC^2 + 2M^2hwC \tag{2.12}$$

In the next Transformer block, the W-MSA is replaced by the multi-head self-attention in the non-overlapped shifted windows (SW-MSA) module to learn local attributes that overlap neighbouring windows in the layer before. In the second stage of the hierarchical architecture, a patch merging layer is introduced. This layer merges 2×2 neighbouring patches by concatenating them, which increases the dimensionality by four. Through a linear layer, the patch merging layer halves the dimensionality, thus increasing the patch area by four and the number of channels by two. Stage 2 continues with two successive Swin Transformer blocks. Stages 3 and 4 also have a patch merging layer followed by successive Swin Transformer blocks. Stage 3 has six blocks instead of two. Instead of encoding the position bias at the patch embedding, Swin encodes it directly in the attention layer, with Equation 2.13. The query, key and value matrices are $Q, K, V \in \mathbb{R}^{M^2 \times d_k}$ with d_k being the query and key dimension and M^2 the number of patches in a window. The relative position along each axis lies in the $[-M+1, M-1]$ range. A bias matrix $\hat{B} \in \mathbb{R}^{(2M-1)\times(2M-1)}$ is parameterised where B takes values from $\hat{B}$.

$$\text{Attention}(Q, K, V) = \text{SoftMax}\left(\frac{QK^T}{\sqrt{d_k}} + B\right)V \tag{2.13}$$

For both the pre-training, training and fine-tuning, the AdamW optimiser is employed, and a cosine decay learning rate scheduler with a linear warm-up is used. Fine-tuning, however, uses significantly lower learning rates. Concluding, the Swin Transformer achieved state-of-the-art performance on COCO object detection by (T.-Y. Lin et al., 2014) and ADE20K semantic segmentation by (B. Zhou et al., 2017).

End-to-End for Pixel-wise Predictions Networks

ViT and Swin Transformer propose very detailed encoder architectures but are brief about their decoder. (Z. Liu et al., 2021), proposes an architecture capable of making per-pixel predictions with excellent performance. Improvements can be made with a more flashed-out and better decoder. Papers that propose architectures specialised for per-pixel predictions are discussed next.

Fully Convolutional Networks for Semantic Segmentation
by (Long et al., 2015)

This is the first paper to present an end-to-end trained neural network for pixel-wise predictions. They introduce the "skip" architecture to combine deep, coarse information and shallow, fine, appearance information. Their architecture is shown in Figure 2.13. The encoder consists of conventional convolutional layers with pooling layers. For the decoder, they use in-network upsampling, more specifically, deconvolution filters. The idea behind the skip connections is that the encoder encodes different kinds of information at different levels. Fine details and local attributes can be encoded in the upper levels. However, due to the reduced size of the lower-level layers, these can not be encoded here, while more global features are encoded at these lower levels as larger parts of the images are visible to the kernels. For per-pixel predictions, the details in the upper and lower levels are relevant. Therefore, the information encoded at the various encoder levels are relevant to the output. The pooled layers of the encoder are added to the upsampled layers of the decoder to connect all levels of the encoder to the decoder. Thus combining the coarse and finer details. Which is shown in Figure 2.14

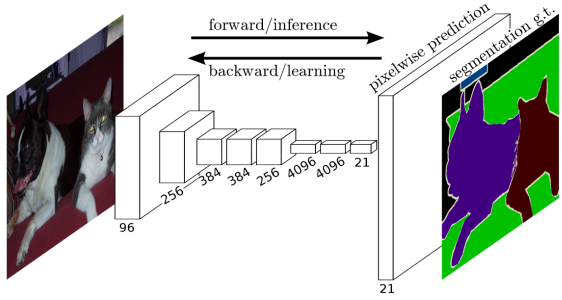


Figure 2.13: End-to-end architecture for pixel-wise predictions by (Long et al., 2015)

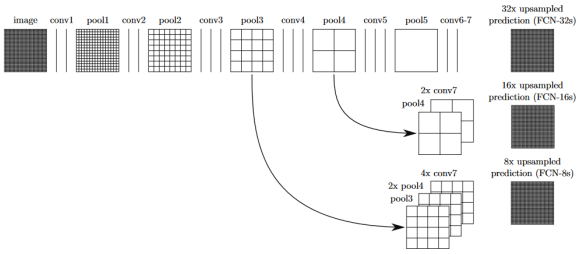


Figure 2.14: Implementation of the skip connection in (Long et al., 2015)

U-Net: Convolutional Networks for Biomedical Image Segmentation
by (Ronneberger et al., 2015)

This paper builds upon the previous paper but modifies and extends the structure. Their architecture is shown in Figure 2.15. Their implementation of the skip connection is different because they propose to concatenate instead of adding up the pooling and upsampling layer. After the concatenation, the output is linearised to the dimensions of the upsampling layer. Their upsampling layer also differs; they use a trainable upsample layer, followed by a convolutional layer with a kernel size of 2. The pooling layer of the encoder has a larger size than the upsampling layer of the decoder and is therefore cropped. They argue that this is beneficial due to the loss of border pixels in every convolution. The other convolutional layers have a kernel size of 3.

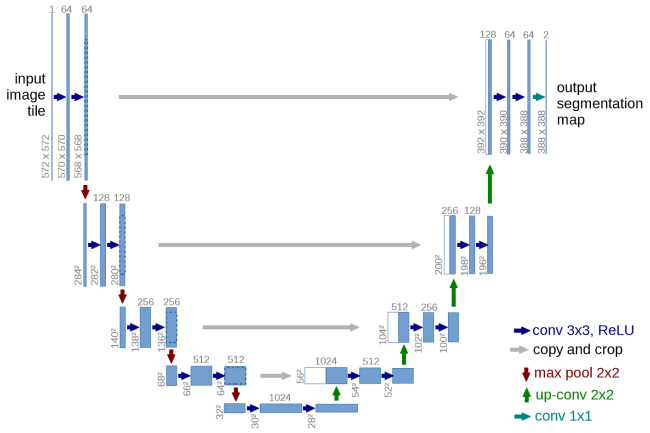


Figure 2.15: U-net architecture for pixel-wise predictions by (Ronneberger et al., 2015)

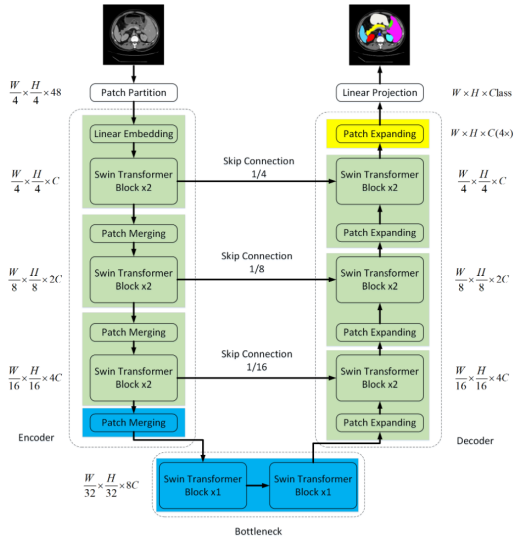


Figure 2.16: Swin-Unet architecture pixel-wise predictions by (Cao et al., 2021)

Swin-Unet: Unet-like Pure Transformer for Medical Image Segmentation
by (Cao et al., 2021)

This paper is the first to integrate the Swin Transform block into the previously discussed U-Net architecture. Figure 2.16 shows the architecture and immediately many similarities are clear between the two. Both contain an encoder, bottleneck and decoder, with skip connections directly connecting the encoder and decoder. Following (Z. Liu et al., 2021), a patch size of 4×4 is used for the patches, which are embedded to an arbitrary dimension C. Until the bottleneck, the structure of Swin Transformer blocks followed by Patch Merging blocks is identical, except that all layers contain two Swin Transformer blocks. Whereas the decoder of the U-Net uses up-convolution for upsampling from one hierarchical layer to the next, followed by convolutional layers, this paper proposes to use patch expansion to transition to the following hierarchical layer and Swin Transformer blocks instead of the convolutional layers.
As previously explained, the patch merging layer merges neighbouring patches into a large patch, which keeps the same dimensionality except for doubling the number of channels. The patch expanding layer tries to do the opposite by splitting one patch into 2×2 neighbouring patches, each with half the number of channels compared to the original patch. Finally, the patches are expanded by the patch size to scale to the original image resolution, which is 4×4. Similar to the U-Net, the skip connections concatenate the output from the encoder with the output of the decoder. However, a linear layer is used to reduce and match the dimensions to that of the patch-expanding layer. It should also be noted that the skip connections are present at $1/4$, $1/8$ and $1/16$ of the image resolution compared to skip connections being present at $1/1$, $1/2$, $1/4$ and $1/8$ of the image resolution.
A recent paper, (A. Lin et al., 2022), also integrated the Swin Transformer in a U-Net. Their architecture, DS-TransUNet, uses a dual-scale encoding mechanism: two parallel hierarchical structures with patch embedding performed with different patch sizes. A Transformer Interactive Fusion Module combines the feature representation of the different structures at the same hierarchical layer. The decoder is similar to that of the Swin-Unet. The DS-TransUNet is superior in image segmentation compared to Swin-Unet and other transformer U-net architectures.

2.1.3. Adjacent Fields of Research

Besides the general developments in computer vision, there are adjacent research fields with papers containing goals overlapping with ours. Specifically of interest are the choice of datasets, data treatment and processing techniques, the proposed architectures and how they are benchmarked. First, the datasets will be discussed in general, followed by monocular depth prediction and the application of self-supervision in this field. Afterwards, the view synthesis field will be reviewed, and finally, the development of geometry-free approaches in this field will be discussed.

Datasets

Datasets are fundamental to the task objective and often influence architecture designs. They are a means to an end. Middlebury, (Scharstein et al., 2001) contributed the first attempt to standardise a dataset for stereo disparity estimation, which could be used to characterise the performance of different algorithms. The dataset is small for today's standards. MPI Sintel, (Butler et al., 2012), introduced a synthetic dataset for this goal. It was sourced from an animated 3D movie and provided dense ground truth. KITTI was originally introduced in 2012, (Geiger et al., 2012), and extended in 2015, (Menze & Geiger, 2015). It contains stereo images from real road scenes, and ground truth is obtained with LIDAR but sparse. Some objects, such as cars, have point clouds from CAD models to provide denser groundtruth. Cityscapes, (Cordts et al., 2016), contains real road scenes as well. Its groundtruth is partly labelled at pixel level and partly sparse. (Mayer et al., 2016) provides three synthetic datasets. The first is FlyingThings3D, which has 25,000 stereo frames and includes randomly placed objects in a three-dimensional space. The second dataset is named Monkaa, and like MPI Sintel includes an animated 3D movie. The stereo images contain nonrigid and softly articulated motion, making algorithms challenging. Finally, they created a dataset inspired by KITTI containing synthetic road scenes. NYU Depth V2, (Silberman et al., 2012), contains indoor RGBD images. It is a good benchmark for the task of monocular depth estimation. However, it does not contain stereo images. CamVid, (Brostow et al., 2009), is similar in that it only provides RGBD images without stereo information. Their images were taken from road scenes. Two other datasets in the literature are RealEstate10K, (T. Zhou et al., 2018), which contains 10 million frames from 80,000 video clips capturing many real estates, both indoor and outdoor. The relative camera poses are known, and as such, they can be used for view synthesis. ACID, (A. Liu et al., 2021), contains coastline imagery, in a similar format as RealEstate10K. All previous discussed dataset are organised within Table 2.1

Table 2.1: Datasets used in adjacent fields of research

Dataset	Type	Size	Stereo Vision	Depth	Synthetic
Middlebury (Scharstein et al., 2001)	Indoor	Small	Stereo	Dense	Real
MPI Sintel (Butler et al., 2012)	3D Animated Movie	Medium	Stereo	Dense	Synthetic
CamVid (Brostow et al., 2009)	Road	Small	Monocular	Dense	Real
NYU Depth V2 (Silberman et al., 2012)	Indoor	Medium	Monocular	Dense	Real
KITTI (Geiger et al., 2012)	Road	Large	Stereo	Sparse (LIDAR) + Dense (3D Model Objects)	Real
Cityscapes (Cordts et al., 2016)	Road	Large	Stereo	Sparse (LIDAR) + Dense (3D Model Objects)	Real
FlyingThings3D (Mayer et al., 2016)	Random Objects in 3D	Large	Stereo	Dense	Synthetic
Monkaa (Mayer et al., 2016)	3D Animated Movie	Medium	Stereo	Dense	Synthetic
Driving (Mayer et al., 2016)	Road	Medium	Stereo	Dense	Synthetic
RealEstate10K (T. Zhou et al., 2018)	Real Estate	Very Large	Monocular	No	Real
ACID (A. Liu et al., 2021)	Coastline	Very Large	Monocular	No	Real

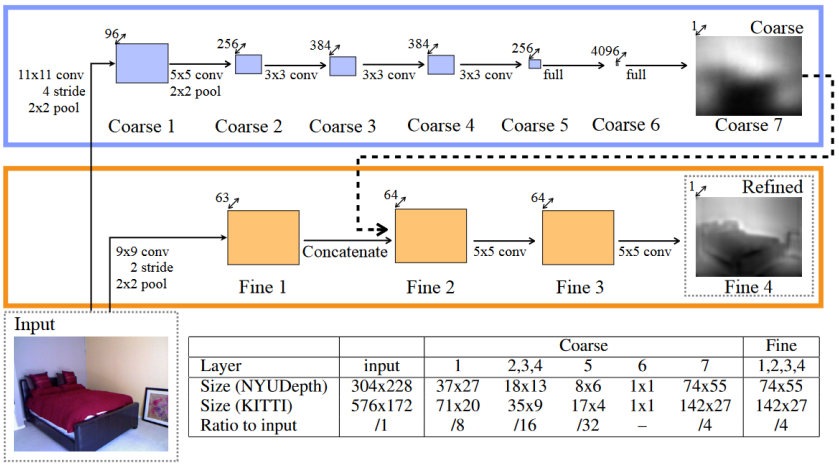
Monocular Depth Prediction

The research on monocular-to-stereo view synthesis is rather lacking. The likely reason for the lack of research in this domain is that its use case is significantly more limited than other larger subdomains. The interest of this thesis in this topic follows from the specific application of deep reinforcement learning for monocular vision-based drones. Therefore to find relevant papers, other subdomains are thoroughly examined. Monocular depth prediction is the subdomain that shares many similarities with monocular-to-stereo view synthesis. More specifically, the category of monocular depth prediction that is self-supervised by being trained on stereo images. Self-supervision is a way of

training a neural network that does not require the ground truth of the to-be-predicted information, instead training on other information that can be related to the predicted information. For monocular depth prediction, stereo images are a good substitute to train on due to the disparity between the two images inversely correlating to the depth. This section continues by dissecting several papers and judging which components of these papers should be adopted. The discussion is done per paper, and the starting point will be a well-known paper that standardises the performance analysis on the KITTI dataset.

Depth Map Prediction from a Single Image using a Multi-Scale Deep Network
by (Eigen et al., 2014)

This paper starts by expressing the practicality of monocular depth estimation over stereo depth estimation. The task of monocular depth estimation can not rely on geometry and is therefore inherently ambiguous. The paper proposes a new approach for estimating depth from a single image. The presented network consists of two component stacks. The first one is a coarse-scale network that estimates the depth of global structures, while the second one is a fine-scale network that can refine the predictions of the coarse-scale network to incorporate finer-scale details. In Figure 2.17, this architecture is visualised. Within the coars=scale network, the lower and middle layers consist of convolutional and pooling layers, while the upper layers are fully connected. Their model was pretrained on ImageNet. The local fine-scale network only has convolutional layers and a pooling layer at the first stage. The output of the course-scale network is concatenated with the output of the first fine-scale layer, which has an equal spatial size. The coarse-scale network is trained first. Afterwards, the local fine-scale network is trained, with the coarse-scale network being excluded from the backpropagation.
This paper also sets the standard for how to use the KITTI dataset. They use 56 scenes from the "city", "residential", and "road" categories of the raw data. Half of these scenes were used for training and the other half for testing. Besides, the images are downsampled by half from 1224×368 to 612×184. The depth for this dataset is captured by a rotating LIDAR scanner and is therefore sampled at irregularly spaced points. They choose to match each pixel of the image to the depth nearest to that pixel. The LIDAR scanner only provides that to the bottom part of the image, however the entire image is fed to the network for additional context. The training set has 800 images per scene. Shots in which the car is stationary are excluded to avoid duplicates. They also use both left and right RGB cameras, but the images are treated as unassociated. The Eigen Split therefore ends up with a training set of 20,000 unique images.



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

Figure 2.17: Model architecture for monocular depth estimation by (Eigen et al., 2014)

Besides presenting the first end-to-end depth prediction network and standardising the KITTI dataset, they also standardised the baselines used to determine the performance of the depth prediction. These followed from prior work and are the following:

Threshold: % of y_i s.t. $\max\left(\frac{y_i}{y_i^*}, \frac{y_i^*}{y_i}\right) = \delta < thr$	RMSE (linear): $\sqrt{\frac{1}{\lvert T\rvert}\sum_{y\in T}\lVert y_i - y_i^*\rVert^2}$
Abs Relative difference: $\frac{1}{\lvert T\rvert}\sum_{y\in T}\frac{\lvert y-y^*\rvert}{y^*}$	RMSE (log) : $\sqrt{\frac{1}{\lvert T\rvert}\sum_{y\in T}\lVert \log y_i - \log y_i^*\rVert^2}$
Squared Relative difference: $\frac{1}{\lvert T\rvert}\sum_{y\in T}\frac{\lVert y-y^*\rVert^2}{y^*}$	

Self-Supervised Monocular Depth Prediction

The following three papers came out less than a year apart from each other. They are similar in what they achieve and how they achieve it.

Deep3D: Fully Automatic 2D-to-3D Video Conversion with Deep Convolutional Neural Networks
by (J. Xie et al., 2016)

Self-supervised learning, in the context of monocular depth estimation, is learning on either stereo images or sequential images from a monocular video. In this case, the performance of the monocular depth estimation is of interest. For this thesis, however, the interest is in outputting the other image from stereo pairs. As is this paper and their method relied on a warping function and, by doing so, developed an architecture nearly identical to one required to do self-supervised learning for monocular depth estimation. They use an internal probabilistic disparity representation to render the image, similar to (Flynn et al., 2016). The loss function is a pixel-wise regression function that compares the predicted right view from the actual left view to the actual right view. This method is differentiable and, therefore, can be used in deep neural networks. This representation is similar to a disparity map but also naturally handles in-painting.
The input image is first downscaled and then goes through a series of convolutional layers. Deconvolutional layers are applied to the feature maps between each convolutional layer. This increases the feature maps to the original resolution. In a future step, a selection layer is applied that combines the output with the original image and as such creates a predicted image. Consequently, this means that the feature maps are similar to disparity maps. This paper states that it is possible to upsample the feature maps from the convolutional layers as disparity maps usually have less high-frequency content than the original colour image. A convolution is done on the sum across all up-sampled feature maps, followed by a SoftMax layer. Each feature map from this convolution represents the probability of a specific disparity across the image. The selection layer uses this accordingly and thus predicts the output right view.
This method does not require depth information. Therefore they are able to train on a large library of 3D movies. The network is based on VGG16, (Simonyan & Zisserman, 2015), a for then large convolutional network trained on ImageNet. It is initialised with VGG16 weights and two randomly initialised fully connected layers.
As far as the writer knows, this is the first algorithm to train directly on stereo pairs and not use depth maps. The fascinating part of this paper is the ablation studies it does. It compares the Mean Average Error without the selection layer to the Mean Average Error with the selection layer. The architectures had scores of 7.01 and 6.87, respectively. Although the selection layer reduces the error, the difference between the errors is promising. They state they directly regress on the novel view without internal disparity presentation and selection layer. This implies that a geometry-free monocular-to-stereo image view synthesis is possible.

Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue
by (Garg et al., 2016)

This paper, published in the same year, is another fundamental paper to the monocular depth estimation community. It is the first paper that uses self-supervision on the KITTI dataset and benchmarks its depth prediction capabilities. Their CNN encoder takes an image and predicts an inverse depth (disparity), which then is combined with the other image in an inverse warping resulting in a warped image. This image is compared to the original image with a reconstruction error. After which the weights of the network are updated. This process is shown in Figure 2.18. The three significant parts of this method are the CNN encoder, the inverse warping and the loss function. They use an L2 loss for their reconstruction error, as shown in Equation 2.14, with D_i being the disparity maps. As such, the disparity maps are parameterised to be a non-linear function of the input image and weights of

the CNN encoder. For backpropagation to be possible, the warped image needs to be linearised using Taylor expansion. The encoder contracts, expands, and has skip connections, inspired by (Long et al., 2015). As the groundtruth depth ranges between 1 and 50 meters, they fix their prediction range from 1 to 50 meters. Finally, they do not use data augmentation techniques.

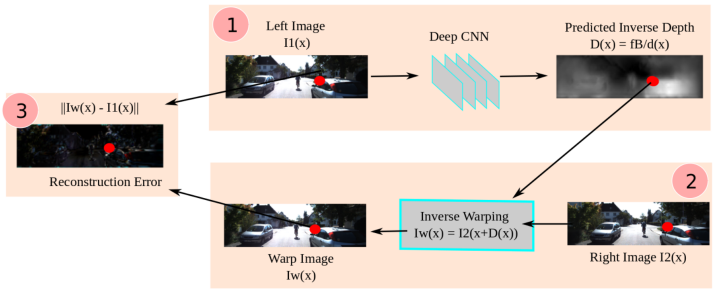


Figure 2.18: Model architecture for self-supervised monocular depth estimation by (Garg et al., 2016)

$$E^i_{recons} = \int_\Omega \| I^i_2 \left(x + D^i(x) \right) - I^i_1 (x) \|^2 dx \tag{2.14}$$

Unsupervised Monocular Depth Estimation with Left-Right Consistency
by (Godard et al., 2017)

The third paper is again similar to the previous one and almost feels like a continuation. It improves upon the flaws of the previous papers. It is also the only paper to acknowledge both previous papers and compare what they did well and what could be improved. This work uses bilinear sampling, by (Jaderberg et al., 2015), where the output pixel is the weighted sum of four input pixels, to generate fully differentiable images. The disparity maps are created at four scales, each doubling in spatial resolution. Their encoder architecture is different again and inspired by DispNet (Mayer et al., 2016). The previous paper generates a warped image from the disparity from the left image combined with the right image. This paper proposes a left-right consistency check. It predicts the left-to-right disparity map, as well as the right-to-left disparity map from one original image. With the disparity maps and the original images, warped images for both left and right are generated and included in all losses. Instead of using deconvolutions in their architecture to output disparities, they use upsampling followed by a convolution, which is the change (Ronneberger et al., 2015) proposes compared to (Long et al., 2015). They combine an L1 with a single scale SSIM, (Wang et al., 2004), for their reconstruction loss. Besides, they experiment with implementing Resnet50, (K. He et al., 2016), for their encoder.
They introduce data augmentation for self-supervised monocular depth estimation as well. They randomly flip the images horizontally and swap their left-right position. Besides, colour augmentation is done randomly with gamma, brightness and colour shifts for the individual channels.
The main datasets they evaluate their model on are KITTI (Geiger et al., 2012) and Make3D (Saxena et al., 2009). In the case of the KITTI, two test splits are used to compare to previous work. The first split is the standard KITTI split. The evaluation is done on 200 high-quality disparity images, which cover a total of 28 scenes, while training is done on 29,000 images with lower-quality disparity images as CAD models are used. The other split is the Eigen Split, proposed by (Eigen et al., 2014), which has 697 test images covering 29 scenes. For training, they use 22,600 images and 888 images for validation, similar to (Garg et al., 2016). They note that depth values are present for less than 5% of the pixels, while the rotation of the Velodyne, the vehicle's motion and motion of surrounding objects, and occlusion at object boundaries, result in errors. They achieve better results than previous papers and dedicate this to their loss function, which enforces the consistency between the predicted depth maps.

Single View Stereo Matching
by (Luo et al., 2018)

As discussed before, (J. Xie et al., 2016) does not generate disparity maps but maps similar to a disparity map. To use their architecture, this paper proposes to apply stereo matching to the synthesised

right image and the original left image. Until this point, stereo matching has not been discussed in the literature review. The goal of stereo matching is to match the individual pixels of the two stereo images, which outputs the disparity map. Integrating this after the selection module is visualised in Figure 2.19. This architecture enables self-supervised learning by first predicting a synthesised right image and afterwards creating a disparity map and a depth map. This order is reversed compared to previous self-supervised monocular depth prediction papers while achieving excellent results.

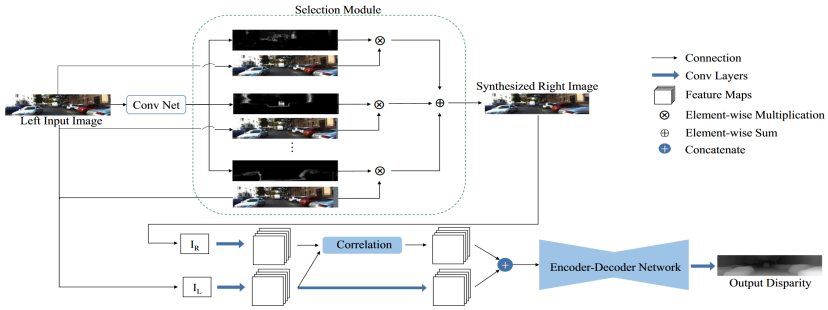


Figure 2.19: Model architecture for self-supervised monocular depth by (Luo et al., 2018)

Papers following have achieved better results by implementing GANs (Aleotti et al., 2019; Pilzer et al., 2018), cyclic learning (Pilzer et al., 2018) or the use of sequential monocular images instead of stereo images (Godard et al., 2019). Finally, last year, a paper was published that leads the monocular depth estimation on KITTI Eigen split unsupervised division of paperswithcode[5]:

Excavating the Potential Capacity of Self-Supervised Monocular Depth Estimation
by (Peng et al., 2021)

This paper does many things well. Their performance gains follow mainly from a novel data augmentation approach, self-distillation application and an efficient full-scale network. The usage of stereo images reduces the data augmentation options. For instance, translation is impossible as the direction of disparity is not horizontally aligned anymore, and scaling is impossible as the conversion from disparity to depth changes. Their data augmentation method, data grafting, applies to stereo images. As shown in Figure 2.20, new images are created by combining the top part of one image with the bottom part of another. When done for both stereo images, it does not disturb the disparity between the stereo images. The images are chosen from the same batch, and the size difference between the top and bottom is varied randomly. (Van Dijk & De Croon, 2019) proved that the essential feature learned by a neural network for monocular depth estimation is the relation between the height at which an object is located within the image and the depth of that object to the camera. Data grafting can prevent the neural network from only learning this feature by flipping the top and bottom images, which leads to significant performance gains.

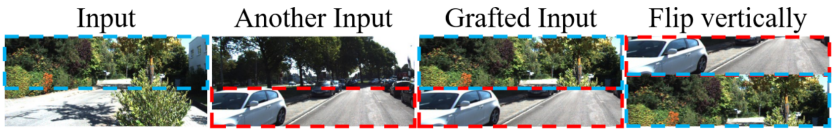


Figure 2.20: Illustration of data grafting by (Peng et al., 2021)

View Synthesis

Another field of research relevant to this literature review is that of view synthesis. Predicting both existing and novel views is central to this domain. These views are generally synthesised with explicit geometric relations within the model. Improvements in the larger computer vision domain have enabled the emergence of geometry-free approaches. The first paper discussed, actually, implicitly uses geometric relations, but afterwards, the field discovers better performance by using warping mechanisms.

[5]https://paperswithcode.com/sota/monocular-depth-estimation-on-kitti-eigen-1 [Accessed 27 October 2022]

(Tatarchenko et al., 2016) describes a method to synthesise a novel view of an object from a single image and a desired new viewpoint. Their architecture is shown in Figure 2.21. They encode an image with convolutional and pooling layers and combine this with an angle of the desired viewpoint encoded by fully connected layers. Finally, the decoder uses up-convolutional layers. This method learns a 3D representation of a single object implicitly. In a previous paper, they proved a network's ability to learn a 3D representation of different chairs in (Dosovitskiy et al., 2015). Now they changed their architecture and proved it for more complicated objects. They do perform some qualitative analysis, but no benchmarks are done. (T. Zhou et al., 2016) is inspired by this architecture and develops a network able to predict appearance flow instead of direct pixel generation. The disparity is the appearance flow in stereo vision. They use bilinear sampling from (Jaderberg et al., 2015) to obtain the synthesised view similar to (Godard et al., 2017). They also apply their model on the KITTI dataset and compare the MAE of their network to (Tatarchenko et al., 2016). Again direct pixel generation is inferior to a warp function.

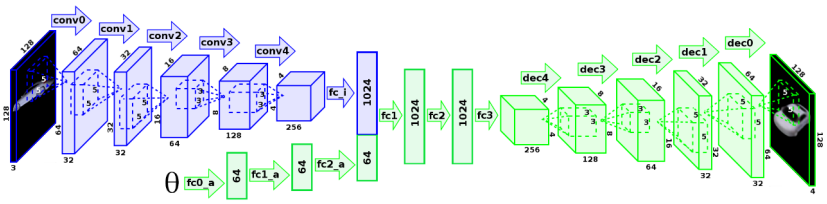


Figure 2.21: Model architecture for view synthesis by (Tatarchenko et al., 2016)

Therefore, subsequent papers in multi-view image synthesis designed architectures that took advantage of more extensive scene representations and more complicated warping functions. The scene representation LDI (Shade et al., 1998), was used in (Tulsiani et al., 2018) to predict a 3D representation. Similar to what (Luo et al., 2018) did by adding a stereo matching algorithm to (J. Xie et al., 2016) (Tulsiani et al., 2018) also added a stereo matching algorithm to their architecture and compared their results. They could not get state-of-the-art depth predictions, but their results were competitive. With the paper by (Tucker & Snavely, 2020), this new research field of monocular view synthesis saw much maturity. Their scene representation is MPI (T. Zhou et al., 2018). They started benchmarking their depth estimation on the KITTI dataset and comparing the synthesised views on image quality by using PSNR and SSIM. They evaluate the results at two different sizes: 384×128 and 1240×375. 22 images of the city category were trained on. Left or right images were randomly taken. The test sequence has 1079 image pairs. Five per cent is cropped at the edges as many artefacts are here. They also compare occluded pixels specifically. (J. Li et al., 2021) improved results again by replacing the MPI with NERF (Mildenhall et al., 2020).
Further improvements were made by not only using spacial extrapolation but extrapolation over time as well (Y. Zhang & Wu, 2022). This is possible as the sequence of images within the KITTI dataset is known over time. As far as the writer is aware this is the current state-of-the-art. Some recent architectures also included modules specifically for occlusion handling (C. Zhang et al., 2022) and loss functions at a feature level instead of at a pixel level (C. Zhang et al., 2023).

Geometry-Free View Synthesis

Geometry-Free View Synthesis: Transformers and no 3D Priors
by (Rombach et al., 2021)

The conditional synthesis task domain focuses on generating images with non-spatial information and spatial information. Controlling both enables the generation of images with specific characteristics. Non-spatial information is defined in object classes, while segmentation structures spatial information. (Esser et al., 2021) developed an architecture for this task. However, their architecture can be generalised and applied to the view synthesis domain. The architecture is inspired by VQ-VAE, (van den Oord et al., 2017). However, it uses a GAN, (Goodfellow et al., 2020), instead of a VAE, (Kingma & Welling, 2013). The architecture is shown in Figure 2.22. In (Rombach et al., 2021), they applied their previous work to novel view synthesis. They were most interested in whether a model requires 3D priors to synthesise novel views in more complicated environments compared to basic environments such as in (Tatarchenko et al., 2016). The datasets for these novel view synthesis tasks

are RealEstate10K, (T. Zhou et al., 2018) and ACID, (A. Liu et al., 2021). Their application requires a single image, meaning the novel view synthesis is extrapolated.

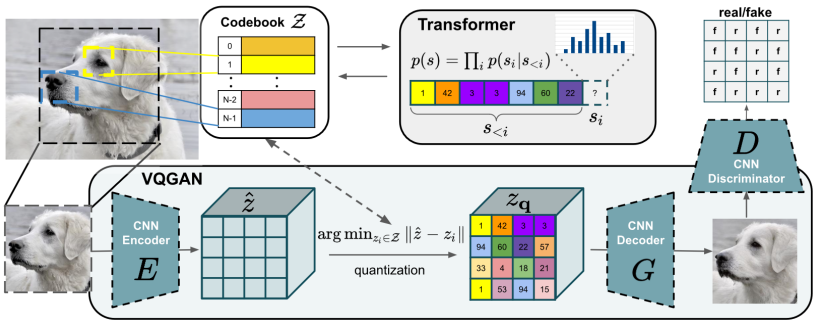


Figure 2.22: VQGAN architecture for geometry-free view synthesis by (Esser et al., 2021)

Six different levels of geometric image warping are proposed to test the need for geometric information. Three are explicit geometric transformations, and the others are implicit. For the implicit transformation, to prove that the geometric information is encoded, depth has to be able to be extracted. This is achieved by linear probing, (M. Chen et al., 2020), a technique commonly used to study the feature quality of networks trained in an unsupervised manner. The idea is that a position-wise linear classifier can be trained to predict the latent representation of a layer in a depth encoder from an equal-sized layer in the original transformer model. As a result, it can be determined with what quality depth is encoded and in what layer depth is encoded best. This paper concludes that no geometric priors are required, and their model can implicitly learn three-dimensional relations between images. The explicit transformations do not improve the performance significantly. However, for small viewpoint changes, compression artefacts dominate the error.

Scene Representation Transformer: Geometry-Free Novel View Synthesis Through Set-Latent Scene Representations
by (Sajjadi et al., 2021)

As the previous work notes, although the quality of the synthesised views is high, their method creates individually sampled frames, meaning that artefacts are not consistent over multiple frames. This paper, (Sajjadi et al., 2021), is intrigued by the model's lack of required geometric information. The goal of this paper differs from the other one in that they want to generate a series of new poses from a collection of images from the same scene. This collection is encoded with their position through a CNN. The CNN outputs embedded patches that go through an encoder consisting of multiple transformer blocks. They call the output a set-latent scene representation. This becomes the input for the key and values of the multi-head attention layer of the decoder, with the query taking a ray corresponding to the pixel to be rendered. The architecture is shown in Figure 2.23, and entire scenes can be generated from the shared set-latent space representation. This task can both be used for interpolation, for example, in the Street View dataset and can be trained directly to do semantic segmentation of novel views. Their architecture, however, has a some limitations. Firstly, the L2 loss for instance craetes blurriness for more complex datasets. Secondly, they require larger datasets due to the lack of geometric inductive biases. Finally, they note that for small translations models that have geometric inductive biases work better.

2.1.4. Discussion

The leading development seen in the broader computer vision domain is the increased use of neural networks and the size of these networks. These developments are clear when presenting the state-of-the-art within the domain and with the relevant subdomains. Techniques, such as network regulation and training scheduling, generally are more advanced in the broader vision domain. Layer normalisation and AdamW have yet to make their way to the view synthesis domain. As far as the writer is aware, the Swin Transformer has also not been applied to this field. Many techniques are available that will slightly improve the network's performance at the cost of increased model size. Some might also increase the latent space, which makes it less suitable for the navigational task of the

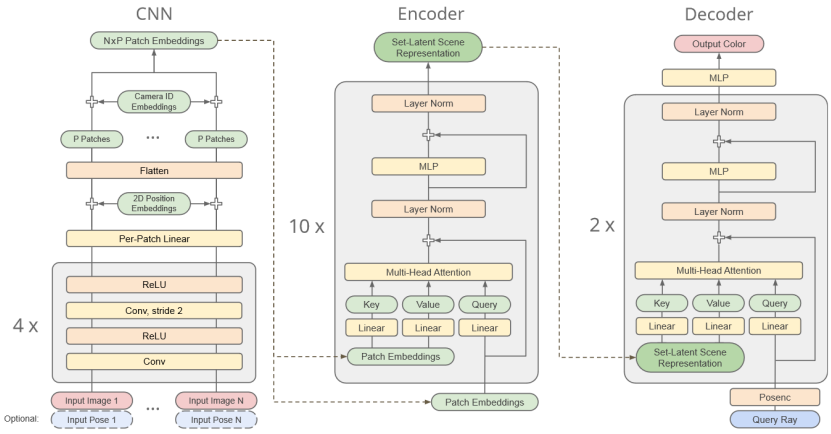


Figure 2.23: SRT architecture for geometry-free view synthesis by (Sajjadi et al., 2021)

next chapter. Therefore VAEs, GANs or self-distillation are not implemented. Performance measuring of reconstruction realism has seen significant standardisation in the last few years, which helps determine the value of our contributions.

2.2. Research Plan

Following the extensive literature review, a research plan can be decided on. The goal to perform geometry-free monocular-to-stereo image view synthesis seems feasible, and encoding depth information and environmental cues in the latent space seems realistic.

Dataset

A dataset should be defined to measure the ability of the architecture to perform the view synthesis. Firstly, to understand the value and limitations of architecture, the stereo images within the dataset should be equally as or more complicated than that of the stereo imagery obtained by a drone in a simulation. Secondly, a comparison to other architectures would be of use to judge the competitiveness of this architecture and if this path is worth further pursuing. Therefore datasets that other researchers use to test their architectures quantitatively on, with competing results are preferred. These two reasons result in selecting KITTI, (Geiger et al., 2012), as the chosen dataset. The KITTI dataset includes left images, right images and depth information.
The literature provides roughly three different splits. The KITTI split, (Geiger et al., 2012), the Eigen split, (Eigen et al., 2014) and a split we will call the Tulsiani split, (Tulsiani et al., 2018). The KITTI and Eigen split are used for monocular depth prediction, both supervised[6] and self-supervised[7]. The Tulsiani split is used for view synthesis. To compare our results, we must similarly use the splits and follow the same routines. However, the Eigen split uses 22,600 training image pairs, whilst the Tulsiani split is limited to 6000 image pairs in training. As our network is trained end-to-end without implicit geometric understanding, it is assumed that a larger dataset to train on is more beneficial to our architecture compared to our counterparts. Therefore we will consider two cases. During the first case, the model is trained on the 6000 image pairs originally available in the Tulsiani split. The other one is pre-trained on the Eigen split, modified to exclude test images from the Tulsiani split. Ideally, the training and testing are done on images with the original resolution provided by KITTI, which is $\mathbb{R}^{1280\times384}$. However, the images are often scaled down in literature for computational purposes, resulting in quicker learning processes. Besides, (Tulsiani et al., 2018) defines a resolution not seen for monocular depth estimation and that papers following their split sometimes do not adhere to. The Tulsiani split downsizes the original resolution by three whilst shortening the width by 5% on both sides. This results in images of $\mathbb{R}^{384\times128}$.

[6]https://paperswithcode.com/sota/monocular-depth-estimation-on-kitti-eigen [Accessed 20 November 2022]
[7]https://paperswithcode.com/sota/monocular-depth-estimation-on-kitti-eigen-1 [Accessed 20 November 2022]

2.2.1. Architecture

As the dataset is defined, the architecture can be specified. U-Net, (Ronneberger et al., 2015), and Swin-Unet, (Cao et al., 2021), are elegant architectures that produce pixel-wise predictions and are trained end-to-end. The U-net was originally proposed as a non-symmetrical architecture with an encoder receiving larger images than the decoder produces. Swin-Unet, however, in its presented design, is symmetrical. For the sake of computational efficiency and practicality concerning the reinforcement learning model, the architecture proposed here is also symmetrical. Therefore both ends of the network are identical in size, with input, output $\in \mathbb{R}^{W \times H \times 3}$ with W and H being the image width and height, respectively. For the simple case, from which the architecture will be further developed, the receiving image is the left image, and the predicted image is the right one from a stereo couple. The left image first goes through the encoder, followed by the bottleneck and finally through the decoder, resulting in a predicted right image. As a result, the basic architecture should look like Figure 2.24.

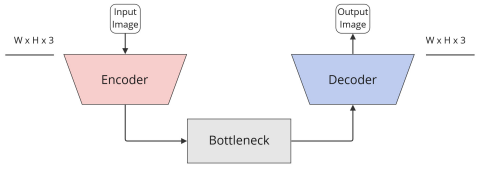


Figure 2.24: Simple end-to-end architecture for pixel-wise predictions

For the sake of clarity, the U-net and Swin-Unet architectures are redrawn in a corresponding style to better showcase their relevant components and differences. They are shown in Figure 2.25 and Figure 2.26, respectively. Although (Cao et al., 2021) claim to have based their Swin-Unet architecture on the U-net architecture, from (Ronneberger et al., 2015), there are, surprisingly, some significant differences. In their respective first layer, the U-Net directly applies a convolutional layer to the input increasing the dimensions from $\mathbb{R}^{W \times H \times 3}$ to $\mathbb{R}^{\ W \times\ H \times C}$ with C being the number of channels. For Swin-Unet, the first layer performs patch embedding. Consequently, the input resolution is directly divided by the size of the patches. In their case, they use a patch size of four, equivalent to the original Swin design, (Z. Liu et al., 2021), resulting in the output of the embedding layer being $\mathbb{R}^{\frac{W}{4} \times \frac{H}{4} \times C}$. These dimensions are unchanged after the consecutive Swin Transformer blocks. As a result of trying to follow the U-Net closely, a skip connection is present directly after the first layer. However, the skip connection of the U-Net has a resolution four times larger than Swin-Unet.
Following this first layer, the U-Net has four subsequent hierarchical layers, whereas the Swin-Unet has only three. Consequently, the Swin-Unet has only three instead of four skip connections. In addition, the resolution of the data in the bottleneck of the Swin-Unet is smaller with $\mathbb{R}^{\frac{W}{32} \times \frac{H}{32} \times 8C}$, compared to the resolution of the data in the bottleneck of the U-Net with $\mathbb{R}^{\frac{W}{16} \times \frac{H}{16} \times 16C}$. Noting these differences, the latent space of the U-Net, which includes four skip connections and the bottleneck, all have a larger resolution than their counterpart.
Therefore some changes to the architecture of Swin-Unet are proposed. Henceforth, the architecture including these changes will be called Swin-U-Net. The first change is an additional skip connection at the highest resolution. As patch embedding reduces the size, this connection takes the input image. This skip connection will have dimensions $\mathbb{R}^{W \times H \times 3}$ compared to the dimensions of the skip connection in the U-Net architecture of $\mathbb{R}^{\ W \times\ H \times C}$. The second change is to reduce the patch size from four to two, creating layers of equivalent size to U-Net. This new Swin-U-Net architecture is shown in Figure 2.27.
This architecture still needs to be tailored to monocular-to-stereo image view synthesis. Most literature for self-supervised learning does a left-to-right prediction. Most literature for view synthesis does, however, train in both directions. Besides the increased training set, our case has an extra benefit. The network has to be trained with knowledge of the direction of translation. Therefore before the network receives this information, it cannot execute a shift but can only encode the original information and a bidirectional disparity or a unitless depth. Inspired by (Tatarchenko et al., 2016) and (T. Zhou et al., 2016), the information to determine which way the image should warp to is given in the bottleneck. If this was given at the start of the encoder or was not given at all, the warp could already be performed in the first layer. Consequently, the following layers encode the warped image,

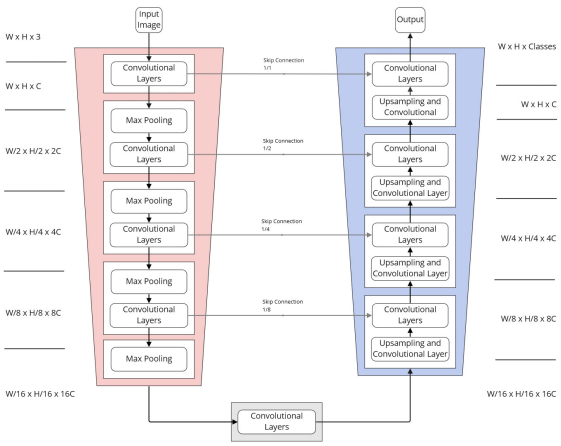


Figure 2.25: Redrawn U-net architecture for pixel-wise predictions

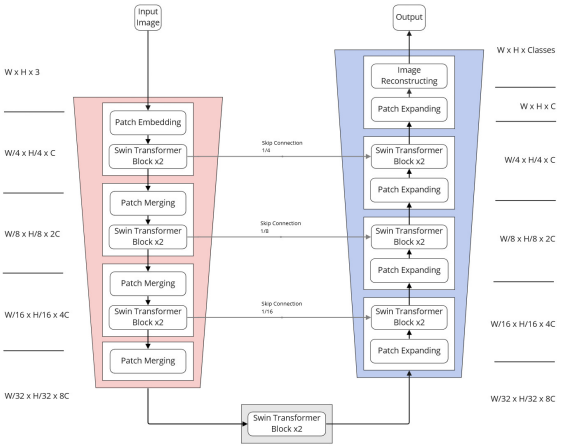


Figure 2.26: Redrawn Swin-Unet architecture for pixel-wise predictions

meaning that the depth information is already lost in the bottleneck. This limits the capability of the reinforcement learning algorithm significantly. Provided these augmentations, the architecture of the geometry-free monocular-to-stereo image view synthesis network is shown in Figure 2.28.

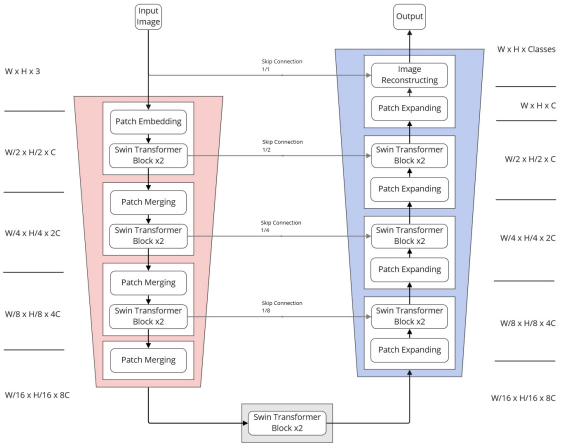


Figure 2.27: Proposed Swin-U-Net architecture for pixel-wise predictions

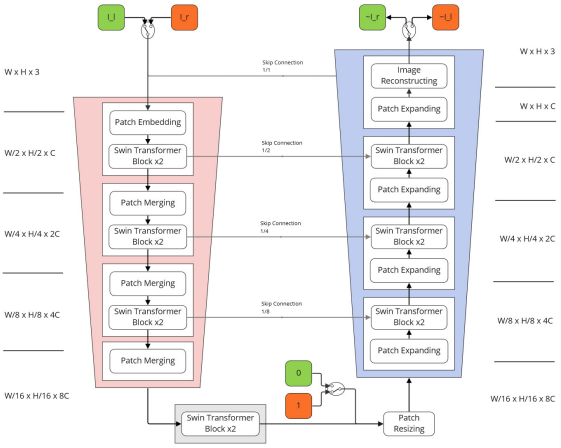


Figure 2.28: Proposed Swin-U-Net architecture for monocular-to-stereo image view synthesis

For other architectures that are tested, such as the U-Net architecture, the identity token is integrated similarly. A fully connected layer is used to reduce the dimensions of the combined skip connection and upsampled layer to the original number of channels. During the upcoming analysis, when the skip connections are removed, and the architecture is trained from the start, the layers are not combined, and the fully connected layer is removed. However, when certain skip connections are tested, the skip connection tensor is replaced by a tensor of equal size consisting of zeros only.
As one of the primary goals of the monocular-to-stereo image view synthesis is to encode depth, the encoder should be tested on its ability to predict depth. One way is to swap out the decoder of the architecture and replace it with a decoder that outputs a depth image. The weights and biases of the encoder will remain fixed. This architecture is shown in Figure 2.29. Depth could also be predicted in a self-supervised manner. The first way would be to use a warp function, similar to (Garg et al., 2016) and (Godard et al., 2017). This architecture is shown in Figure 2.30. Finally, a stereo matching network could be used, similar to (Luo et al., 2018). This architecture is visualised in Figure 2.31.

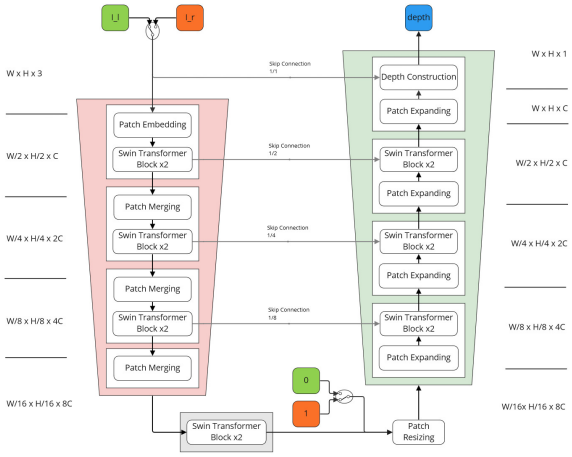


Figure 2.29: Swin-U-Net architecture for direct monocular depth estimation

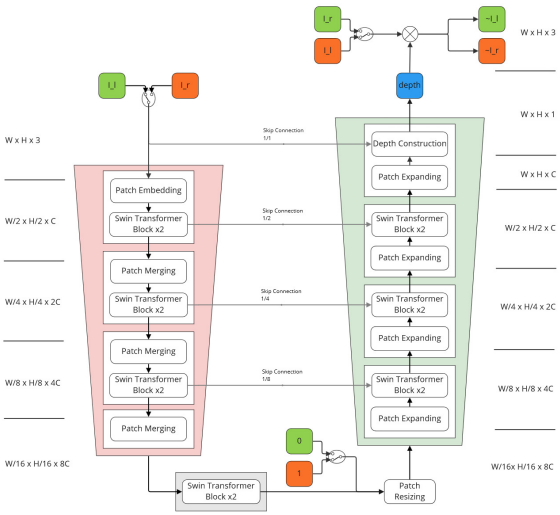


Figure 2.30: Swin-U-Net architecture with a warping mechanism for self-supervised monocular depth estimation

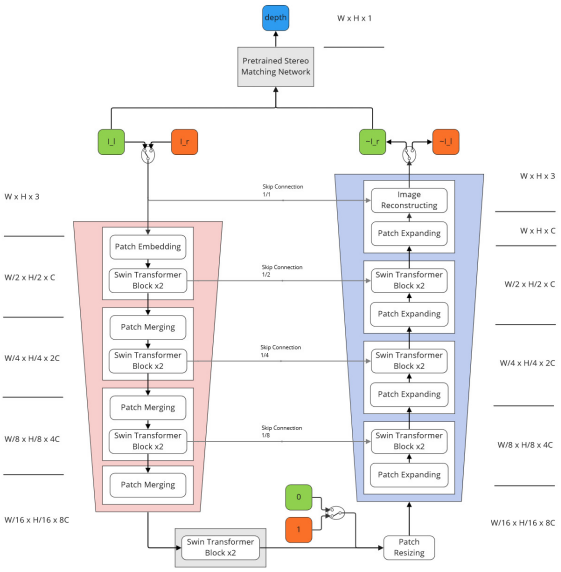


Figure 2.31: Swin-U-Net architecture with stereo matching for self-supervised monocular depth estimation

2.2.2. Implementation Details

The model is implemented with (Paszke et al., 2017). Regarding the training regime, some details have to be specified. The Smooth L1 Loss, from Equation 2.6, is used for the loss function. It seems to cover both advantages of the L1 and L2 loss. The cosine learning rate scheduler, (Loshchilov & Hutter, 2016), was implemented in ViT, (Dosovitskiy et al., 2020) and follow-up papers. Therefore it is implemented here as well. The optimizer (Z. Liu et al., 2021) uses is AdamW, (Loshchilov & Hutter, 2019), which we will continue using. All parameter values not specified in this paper can be assumed to be adopted from (Z. Liu et al., 2021). The data augmentation regime is partially taken over from (Godard et al., 2019) and (Peng et al., 2021). The code for horizontal flipping and colour augmentation, which include random changes to *brightness* $\in (0.8, 1.2)$, *contrast* $\in (0.8, 1.2)$, *saturation* $\in (0.8, 1.2)$ and *hue* $\in (-0.1, 0.1)$, was used from the former. In contrast, the code for data grafting was used from the latter. The Swin Transformer uses windows for its multi-head attention mechanism. As a consequence, the image size should be such that $\textit{image size} \,\%\, \textit{window size} = 0$. This holds for the deeper layers of the model, so $\frac{\textit{image size}}{\textit{patch size} \cdot 2^2} \,\%\, \textit{window size} = 0$. Images from the KITTI dataset (Geiger et al., 2012) at one-fourth of the original resolution are $\mathbb{R}^{320 \times 96}$. The previous formula is correct at this image height for a window size of 6. The image is padded on both sides by 32 to create an image

width of 384, which also works for the deeper layers with a window size of 6. Standard training is done on 60 epochs, of which the first 10 are for warm-up. For batch size selection, there is a trade-off between learning stability and occupation of memory on the GPU. A batch size of 8 was selected as a result.

2.2.3. Experiments

With all the ingredients listed, the model can be assembled. This will be done slowly with unit tests and general tests for bugs. When the network is developed, the dataset is in place, augmentations can be performed, training can be scheduled, and certain experiments can be performed.
The first experiment is shown in Table 2.2. It is to benchmark the proposed architectures on reconstruction realism. The first architectures tested are U-Net and Swin-Unet, as described in Subsection 2.2.1. The Swin-U-Net follows this. The first configuration has a patch size of 2, and the second configuration has a patch size of 4. The training split that will be used at first is the Eigen split, as this dataset is large and therefore reflects the network's capabilities better. The metrics to measure reconstruction realism are PSNR, SSIM, and LPIPS.

Table 2.2: Benchmark the proposed architectures for monocular-to-stereo image view synthesis on reconstruction realism

Architectures:	1. U-Net	2. Swin-Unet	3. Swin-U-Net patch size = 4	4. Swin-U-Net patch size =2
Dataset:	KITTI			
Image Resolution:	320×96			
Training Split:	Eigen Split			
Metrics:	PSNR SSIM	LPIPS		
Loss:	Smooth L1 Loss			

This experiment is relevant to the feasibility of implementing this architecture in a RL algorithm. Therefore this experiment is already performed. The developed architectures are trained on Nvidia GeForce GTX 1080 Ti. Training a single architecture for 60 epochs can take around five days. As such, not all suggested architectures and experiments were trained and tested at the time of writing. Table 2.3 shows the results of this experiment. Looking at these results, it can be concluded that Swin-U-Net achieves better reconstitution realism than U-Net and Swin-Unet. It should be noted that the difference with the original Swin-Unet is quite significant, which might be the result of the different image sizes. Reducing the patch size also seems to aid performance significantly on all metrics. Compared to U-Net, the LPIPS metric's performance is the most impressive of the Swin-U-Net, which is the most accurate reconstruction realism metric.

Table 2.3: Results of the proposed architectures for monocular-to-stereo image view synthesis on reconstruction realism

Configuration	PSNR ↑	SSIM ↑	LPIPS ↓
U-Net	21.365	0.787	0.0977
Swin-Unet^b (original)	20.396	0.730	0.1798
Swin-U-Net (updated with patch size = 4)	21.317	0.783	0.1077
Swin-U-Net (updated with patch size = 2)	21.604	**0.797**	**0.0947**

Worth inspecting is the latent space size when considering the preferred architecture for the navigational task. Table 2.4 presents figures regarding the latent space's size and the entire network's parameter count. The U-Net has by far the largest latent space size, mostly due to its highest skip connection. The Swin-Unet, in comparison, has a latent space size ten times smaller. It can also be noted that the addition of skip connections does not impact the number of trainable parameters. An architecture that does not include skip connections would also be significantly smaller and, therefore, might be more efficient for the navigational task.
The impact of the skip connections is also relevant, as these will be part of the latent space connected to the reinforcement learning model discussed in Chapter 3. First, the network will be trained without skip connections. This is followed by training an architecture with the deepest skip connection. Afterwards, the second deepest skip connection is added, and the entire architecture is trained from scratch again until all skip connections are present. This experiment is shown in Table 2.5.

Table 2.4: Latent space size of the proposed architectures

Configuration	Feature Channels in Layer 1		Space (H × W) × C	Total Size		Parameters
Bottleneck			(6 × 24) × 896	0.13M		
Skip connection 1/8			(12 × 48) × 448	0.26M		
Skip connection 1/4			(24 × 96) × 224	0.52M		
Skip connection 1/2			(48 × 192) × 112	1.03M		
Skip connection 1/1			(96 × 384) × 56	2.06M	+	
U-Net	56			***4.00M***		27.2M
Configuration	Patch Size	Embed Dimension	Space (H × W) × C	Total Size		Parameters
Bottleneck			(3 × 12) × 768	0.03M		
Skip connection 1/16			(6 × 24) × 384	0.05M		
Skip connection 1/8			(12 × 48) × 192	0.11M		
Skip connection 1/4			(24 × 96) × 96	0.22M	+	
Swin-Unet (original)	4	96		*0.41M*		27.8M
Bottleneck			(3 × 12) × 768	0.03M		
Skip connection 1/16			(6 × 24) × 384	0.05M		
Skip connection 1/8			(12 × 48) × 192	0.11M		
Skip connection 1/4			(24 × 96) × 96	0.22M		
Skip connection 1/1			(96 × 384) × 3	0.11M	+	
Swin-U-Net (updated)	4	96		*0.53M*		27.7M
Bottleneck			(6 × 24) × 768	0.11M		
Skip connection 1/8			(12 × 48) × 384	0.22M		
Skip connection 1/4			(24 × 96) × 192	0.44M		
Skip connection 1/2			(48 × 192) × 96	0.88M		
Skip connection 1/1			(96 × 384) × 3	0.11M	+	
Swin-U-Net (updated)	2	96		*1.77M*		27.6M

Table 2.5: Ablation studies on the impact of the skip connections in both training and testing for monocular-to-stereo image view synthesis on reconstruction realism

Included skip connection:	1. -	2. 1/8	3. 1/8 and 1/4	4. 1/8, 1/4 and 1/2
Architecture:	Swin-U-Net (patch size = 2)			
Dataset:	KITTI			
Image Resolution:	96 × 320			
Training Split:	Eigen Split			
Metrics:	PSNR SSIM LPIPS			
Loss:	Smooth L1 Loss			

More quantitative analyses can be done on skip connections. A completely trained network will be tested in Table 2.6 by including individual skip connections in different runs. Following certain skip connections and even the bottleneck are excluded. The goal of this study is also to support the qualitative results from the analysis of the different parts of the architecture with statistics about reconstruction realism. The qualitative results aim to show the function of each layers in the architecture.

Table 2.6: Ablation studies on the impact of the skip connections after training in testing for monocular-to-stereo image view synthesis on reconstruction realism

Included skip connection:	1. -	2. 1/8	3. 1/4	4. 1/2	5. 1/1
Excluded skip connection:	6. w/h bottleneck	7. 1/8	8. 1/4	9. 1/2	10. 1/1
Architecture:	Swin-U-Net (patch size = 2)				
Dataset:	KITTI				
Image Resolution:	96 × 320				
Training Split:	Eigen Split				
Metrics:	PSNR SSIM LPIPS				
Loss:	Smooth L1 Loss				

These experiments are also already performed although partly as training architectures from scratch is time-consuming. The most relevant configuration was excluding all skip connections and noting the difference in performance. As shown in Table 2.7, it achieves 21.446, 0.787 and 0.1198 for PSNR, SSIM and LPIPS, respectively. Although worse than including all skip connections, it achieves comparable results to U-Net in PSNR and SSIM. This indicates that it is an interesting architecture to use for the navigation task. Other things to note in this table are the seemingly little contribution of the skip connection at $1/8$ to the performance and the low performance of the architecture only using the first skip connection and not using the first skip connection at all.

Table 2.7: Results of Experiment 1-6 and Experiment 1-7

Configuration during training	Configuration during testing	PSNR ↑	SSIM ↑	LPIPS ↓
Excl. all skip connections	Excl. all skip connections	21.446	0.787	0.1198
Incl. all skip connections	Incl. bottleneck	6.994	0.038	0.7225
Incl. all skip connections	Incl. skip connections 1/8	7.249	0.037	0.6785
Incl. all skip connections	Incl. skip connections 1/4	12.842	0.292	0.4543
Incl. all skip connections	Incl. skip connections 1/2	15.888	0.440	0.3056
Incl. all skip connections	Incl. skip connection 1/1	6.594	0.024	0.7263
Incl. all skip connections	Excl. bottleneck	15.840	0.491	0.3741
Incl. all skip connections	Excl. skip connections 1/8	20.876	0.779	0.1014
Incl. all skip connections	Excl. skip connections 1/4	17.430	0.534	0.3063
Incl. all skip connections	Excl. skip connections 1/2	11.855	0.355	0.3382
Incl. all skip connections	Excl. skip connection 1/1	4.554	0.069	0.6182
Incl. all skip connections	Incl. all skip connections	**21.604**	**0.797**	**0.0947**

Looking at a test example, Figure 2.42 shows the original right image and Figure 2.43 shows the predicted right image. The results are relatively similar. Figure 2.32 shows the output of the bottleneck, and Figure 2.33 shows the output without the bottleneck. Although the colouring is done well, a lot of the detail is missing from this image, and the alignment also seems incorrect, indicating that depth is encoded here. Looking at the skip connection at $1/8$ and corresponding figures, Figure 2.35 and Figure 2.34, its contribution seems minor, as the results earlier already indicated. The skip connection at $1/4$ with corresponding figures, Figure 2.37 and Figure 2.36 seem to encode some of the depth as well.

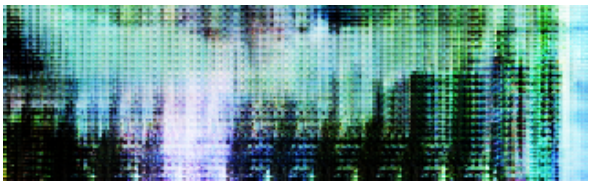


Figure 2.32: Predicted right image without skip connections

Figure 2.33: Predicted right image without the bottleneck

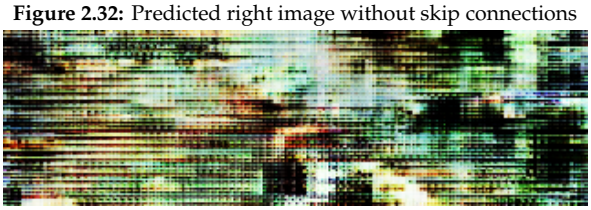


Figure 2.34: Predicted right image including the skip connection at $1/8$

Figure 2.35: Predicted right image excluding the skip connection at $1/8$

Figure 2.36: Predicted right image including the skip connection at $1/4$

Figure 2.37: Predicted right image excluding the skip connection at $1/4$

The skip connection at $1/2$ with corresponding figures, Figure 2.39 and Figure 2.38 show that the general colouring is done in this layer. The outputs that only include the skip connection show somewhat realistic colouring, while the output excluding the skip connection shows a saturated version of the correct image. Finally, the skip connection at $1/1$ with corresponding figures, Figure 2.41 and Figure 2.40 indicate that fine colouring is done in this layer.

Figure 2.38: Predicted right image including the skip connection at $1/2$

Figure 2.39: Predicted right image excluding the skip connection at $1/2$

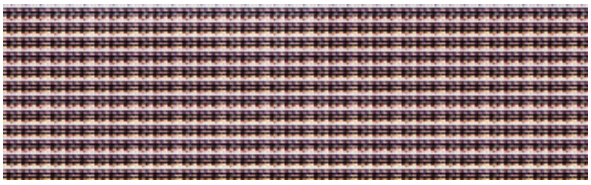


Figure 2.40: Predicted right image including the skip connection at $1/1$

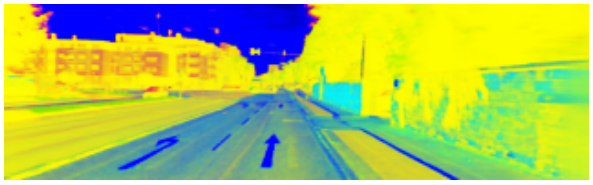


Figure 2.41: Predicted right image excluding the skip connection at $1/1$

Figure 2.42: Original right image

Figure 2.43: Predicted right image

Following those experiments, now experiments will be stated that have not been performed yet, but will be necessary to draw strong conclusion from this research. The first experiment that will be done is to benchmark against state-of-the-art architectures. The experiment is stated in Table 2.8. The Swin-U-Net architecture with a patch size of 2 is expected to be the best-performing architecture from the first experiment and will be used for all succeeding experiments. It will be trained on the original Tulsiani split at first, and afterwards, the architecture will be pre-trained on the Eigen split that excludes images that are part of the test set of the Tulsiani split. The image resolution is changed to match that of other architectures, and PSRN, SSIM, and LPIPS will be used, following the other papers.

Table 2.8: Benchmark the proposed architectures for monocular-to-stereo image view synthesis on reconstruction realism

Training Split:	1. Tulsiani Split	2. Tulsiani Split + pre-training
Architecture:	Swin-U-Net (patch size = 2)	
Dataset:	KITTI	
Image Resolution:	384×128	
Training Split:	Tulsiani Split	
Metrics:	PSNR SSIM LPIPS	
Loss:	Smooth L1 Loss	

After benchmarking, ablation studies can be performed to measure the impact of certain design choices. As proposed in Table 2.9, the first ablation study measures the impact of the applied data augmentation techniques. These include colour augmentation and image flipping. However, bi-directional training is specifically of interest instead of training to predict from left to right. Besides, due to the novelty of data grafting, the impact of this augmentation technique is also studied.
The second ablation study is about the impact of the loss function, shown in Table 2.10. The model will be trained on an L1 Loss, an L2 Loss and a Smooth L1 Loss. Besides, it is noted that other litera-

Table 2.9: Ablation studies on the impact of the applied data augmentation techniques to the KITTI dataset for monocular-to-stereo image view synthesis on reconstruction realism

Augmentation Techniques:	1. Colour augmentation	2. Image flipping	3. Bi-directional training	4. Data grafting
Architecture:	Swin-U-Net (patch size = 2)			
Dataset:	KITTI			
Image Resolution:	320×96			
Training Split:	Eigen Split			
Metrics:	PSNR SSIM LPIPS			
Loss:	Smooth L1 Loss			

ture sometimes combines one of these losses with a perceptual loss. As we identify LPIPS as the best measure of reconstruction realism, this is the perceptual loss that we will use.

Table 2.10: Ablation studies on the impact of the loss function for monocular-to-stereo image view synthesis on reconstruction realism

Loss:	1. L1 Loss	2. L2 Loss	3. Smooth L1 Loss	4. Smooth L1 Loss + LPIPS
Architecture:	Swin-U-Net (patch size = 2)			
Dataset:	KITTI			
Image Resolution:	320×96			
Training Split:	Eigen Split			
Metrics:	PSNR SSIM LPIPS			

The following ablation study is about the importance of choosing the right dataset. The Swin-U-Net architectures are likely better for larger disparities than U-Net because of its global attention mechanism. The dataset could therefore influence the performance significantly. The KITTI dataset with the Eigen Split will therefore be compared to another dataset. It was decided not to use Cityscapes due to its similarities to KITTI. MPI Sintel could be an interesting dataset. However, as this dataset is synthetic, we could develop a new synthetic dataset with images from a stereo vision camera on a drone in a simulation environment defined in Chapter 3. The experiment is shown in Table 2.11.

Table 2.11: Ablation studies on the impact of the dataset for monocular-to-stereo image view synthesis on reconstruction realism

Dataset:	1. KITTI	2. Not yet defined, could be self-generated
Training Split:	Eigen Split	
Architectures:	1. U-Net	2. Swin-U-Net patch size =2
Image Resolution:	320×96	
Metrics:	PSNR SSIM LPIPS	
Loss:	Smooth L1 Loss	

Finally, as described earlier, it is interesting to understand the network's ability to encode depth and therefore predict depth. In the following experiment, the performance is benchmarked in Table 2.12. Although it is expected that training directly on depth information might lead to better results than stereo matching, stereo matching will be used to measure the performance. This is because the qualitative results they provide are of higher quality, and the quantitative results following stereo matching are, therefore, more relevant to the combined discussion. The stereo matching network used is (Yamaguchi et al., 2014). This is the same algorithm as (Tucker & Snavely, 2020) uses. The metrics that will measure depth estimation performance are the standard Eigen split metrics, from (Eigen et al., 2014). Analysis of the impact of the skip connections is of interest again. Therefore these experiments will be done again, but while measuring depth performance. The experiments are shown in Table 2.13 and Table 2.14.

Table 2.12: Benchmark the proposed architecture for monocular-to-stereo image view synthesis extended with stereo matching on depth estimation

Architecture:	Swin-U-Net (patch size = 2)						
Stereo Matching:	SPS-Stereo						
Dataset:	KITTI						
Image Resolution:	192×640						
Training Split:	Eigen Split						
Metrics	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Loss:	Smooth L1 Loss						

Table 2.13: Ablation studies on the impact of the skip connections in both training and testing for monocular-to-stereo image view synthesis extended with stereo matching on depth estimation

Included skip connection:	1. -	2. 1/8	3. 1/8 and 1/4	4. 1/8, 1/4 and 1/2
Architecture:	Swin-U-Net (patch size = 2)			
Dataset:	KITTI			
Image Resolution:	96×320			
Training Split:	Eigen Split			
Metrics	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Loss:	Smooth L1 Loss						

Table 2.14: Ablation studies on the impact of the skip connections after training in testing for monocular-to-stereo image view synthesis extended with stereo matching on depth estimation

Included skip connection:	1. -	2. 1/8	3. 1/4	4. 1/2	5. 1/1
Excluded skip connection:	6. w/h bottleneck	7. 1/8	8. 1/4	9. 1/2	10. 1/1
Architecture:	Swin-U-Net (patch size = 2)				
Dataset:	KITTI				
Image Resolution:	96×320				
Training Split:	Eigen Split				
Metrics	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Loss:	Smooth L1 Loss						

2.3. Conclusion

In conclusion, this chapter extensively discusses the topic of geometry-free monocular-to-stereo image view synthesis, which involves predicting a stereo pair from a monocular view using neural networks without 3D priors. The chapter starts with a literature overview that covers three themes, namely the historical context of computer vision, general developments in computer vision and artificial intelligence, and research adjacent to the topic. The chapter brings these themes together in a discussion, and a research plan is proposed that outlines the goals, methods, experiments, and datasets involved in the study. The preliminary results are also presented and discussed, highlighting the potential impact of the research on the application of neural networks to deep reinforcement learning. Overall, this chapter provides valuable insights into the current state of research on geometry-free monocular-to-stereo image view synthesis and sets the stage for further studies in this area.

3

Deep Reinforcement Learning for Monocular Vision-Based Drones trained with Stereo Vision

The previous chapter, Chapter 2, concluded that geometry-free monocular-to-stereo image view synthesis is feasible and performs well. Different architectures were proposed that could be trained end-to-end, and depth and environmental cues are encoded in the latent space of these architectures. This chapter is the stepping stone to integrating the neural networks in a reinforcement learning algorithm for monocular vision-based drones. First, a literature overview is presented, followed by a research plan. The literature overview is composed of three themes. The first theme presents the current application of deep reinforcement learning in Subsection 3.1.1. It demonstrates the versatility of deep reinforcement learning and the fundamental strength of the components. The goal of this theme is not to update the reader on techniques relevant to this thesis but to get into the mindset of how to view deep reinforcement learning. The second theme is the use of reinforcement learning for drone navigation, described in Subsection 3.1.2. An overview of the driving objectives, the primary simulation environments and popular reinforcement learning agents will be discussed. Finally, the more general theme of vision-based deep reinforcement learning approaches is discussed in Subsection 3.1.3. This section contains an overview of vision-based simulation environments and discusses different design elements of deep reinforcement learning architectures. Following the literature review, the research plan is presented in Section 3.2.

3.1. Literature Overview

3.1.1. Deep Reinforcement Learning in Today's World

Reinforcement learning (RL) is the field of making decisions over time through consequences. An agent tries to maximise the expected cumulative reward, $R = \mathbb{E}[\sum_t r_t]$, in an environment by learning an optimal policy, $\pi(h_t)$, to take actions, $a_t \in \mathcal{A}$, from observations, $o_t \in \mathcal{O}$, with real-valued reward, r_t, and the interaction history, h_t for discrete steps t = 1, 2, ..., n. It often includes a discount factor, $\gamma \in (0, 1)$, within the expected cumulative reward. In the case of a Markov decision process (MDP), the states, $s_t \in \mathcal{S}$, follow directly from observations. This process, where the agent performs actions impacting the environment, but decided based on states and rewards from the environment, is schematically shown in Figure 3.1. Often a problem is non-Markovian such as the partially observable Markov decision process (POMDP). In such a case, the observations only provide partial information about an unobserved state. Domain knowledge is required for such problems to define the set of hidden states and observation probabilities. Deep learning can represent and track hidden states without much domain knowledge. Deep Reinforcement Learning (DRL) uses neural networks to compress extensive state representations to a manageable representation for the reinforcement learning algorithm.

DRL has proven to be a powerful tool in creating artificial intelligence. Its application is societal-wide,

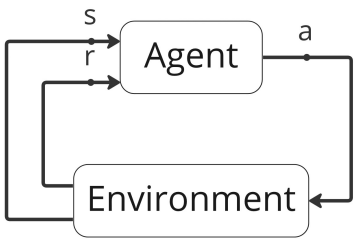


Figure 3.1: Information flow of reinforcement learning algorithms

seen in computer applications, but also in the automation and optimisation of jobs, systems and institutions. At the forefront of development, DeepMind has paved the way for creative and inspiring applications. First, by learning artificial intelligence to play Atari video games, followed by more complex games. Their DRL algorithms received global attention by beating the best professionals in chess[1] and Go[2]. Back then, both games were perceived as too complicated and creative for artificial intelligence to ever be competitive in. Humans observe their environment through their sensors. These include their eyes, their ears, and their noses. This information is processed, and together with memory, the human continuously performs actions and interacts with the environment. To a certain degree, deep reinforcement learning operates similarly. It can convert an extensive environment into concrete actions and evaluate the quality of such actions. The earlier breakthroughs showed that creativity could be thought through incentivising exploration while optimising for performance, making DRL perfect for creative problem-solving.
(Mnih et al., 2013) is one of the most influential papers regarding DRL. They showed the capabilities of these algorithms with improved computational power. The simulation environments they chose to showcase their algorithms on were Atari 2600 games from the Arcade Learning Environment. These games provide highly-dimensional sensory inputs and clearly defined action spaces. Due to the recognised potential of deep reinforcement learning following this paper, attempts were made to standardise protocols and have benchmarks to measure the performance of different algorithms. OpenAI Gym, (Brockman et al., 2016), is the most successful attempt to standardise the practices. They only provide the environments, but not the learning agents. The benchmarks measure the performance through sample complexity and final performance. Their tool is for research, not for competition, so researchers have to describe how they achieved their scores. To standardise and centralise reinforcement learning algorithms, OpenAI Baselines, (Dhariwal et al., 2017), was introduced. This was further improved with Stable-Baselines, (Hill et al., 2018), which includes a unified structure for all algorithms, more complete documentation and state-of-the-art algorithms such as SAC, (Haarnoja et al., 2018), and TD3, (Fujimoto et al., 2018). The same developers further improved the backend in (Raffin et al., 2021), which they called Stable-Baselines3. They extensively benchmarked all their algorithms on Atari games and continuous control PyBullet envs.

3.1.2. Reinforcement Learning for Drone Navigation

By now, reinforcement learning is a mature field of study. There are many attempts to standardise environments, benchmarks and learning agents. However, unexpectedly no such attempts were made in the autonomous navigation of the Unmanned Aerial Vehicle subdomain. No benchmarks are present, and new architectures and research do not slightly alter existing architectures but are often fundamentally novel. With such diversity of research, structure concerning the presentation of these papers is lacking. However, recently, (AlMahamid & Grolinger, 2022) provided a systematic review of this field of study, providing a place or manual from which essential papers can be found, and new papers can build.

Autonomous Unmanned Aerial Vehicle navigation using Reinforcement Learning: A systematic review
by (AlMahamid & Grolinger, 2022)

This paper aims to help researchers select the right algorithms for their problem and categorise pa-

[1]https://www.bbc.com/news/technology-42251535 [Accessed 22 November 2022
[2]https://www.bbc.com/news/technology-35785875 [Accessed 22 November 2022]

pers in the different navigational problems. 159 relevant papers were found. One of the things they do in this paper is to explain reinforcement learning and different reinforcement learning algorithms. Previously they did a general analysis on all reinforcement learning algorithms in (AlMahamid & Grolinger, 2021).
The architectures of navigation tasks for drones that use a reinforcement learning agent are designed as shown in Figure 3.2. Drones can access devices that obtain scenery images, depth map images, localisation information and more. This data is used to determine reward information and calculate the reward. Besides, it provides a state that is also given to the agent. This provides an action, which can, for example, be a movement command or location information. This paper divides this navigation task into four. There is an objective, a resulting framework, the simulation software and an algorithm. The four drone navigation objectives that use DRL are UAV control, obstacle avoidance, path planning and flocking. These are further divided into sub-objectives. Obstacle avoidance and path planning are of most interest to this thesis, as the monocular-to-stereo image view synthesis likely adds the most benefit in these scenarios. Path planning has two main types: global path planning and local path planning. Global path planning plans from start to destination, whereas local path planning plans from waypoint to waypoint while keeping track of the destination. Besides, a division can be made in algorithms that use map-based navigation and algorithms that are mapless but require GPS or IMU.
From this, the objective of the software framework is established. These objectives are the following: energy-aware UAV navigation, path planning, flocking, vision-based frameworks and transfer learning. Specifically, the vision-based frameworks are of interest, as this thesis also intends to use the cameras as a primary means of navigation. They discuss the need for a recurrent neural network to capture relations over time. Papers that used vision-based frameworks are, (Akhloufi et al., 2019; Andrew et al., 2018; L. He et al., 2020; Singla et al., 2021).
The simulation software requires a reinforcement learning agent, a UAV flight simulator and a 3D graphics engine. As a flight simulator, a Robot Operating Systems (ROS) (Quigley et al., 2009) integrated with Gazebo (Koenig & Howard, 2004), and Microsoft AirSim (Shah et al., 2018). are primarily used; see Figure 3.3. The 3D Graphics Engine often depends on the flight simulator, and the primary graphics engines for AirSim is the Unreal Engine (Karis & Games, 2013). There is simulation software that combines both, such as MATLAB. AirSim is an open-source photo-realistic simulator with a tightly coupled physics and rendering engine. This results in limited simulation speeds, which limits the application of reinforcement learning. ROS has a high-fidelity physics engine, and Gazebo is used in many competitions for its simulated environments. However, it is limited in its rendering capabilities, especially by today's standards.

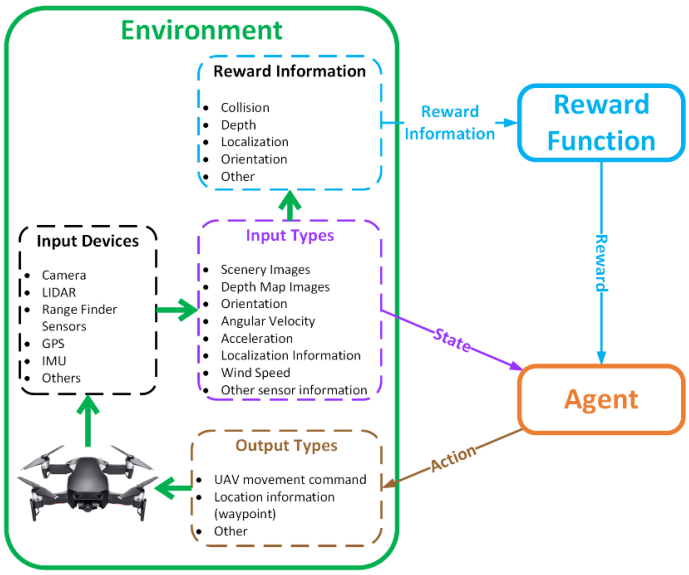


Figure 3.2: Information flow of reinforcement learning algorithms for UAVs by (AlMahamid & Grolinger, 2022)

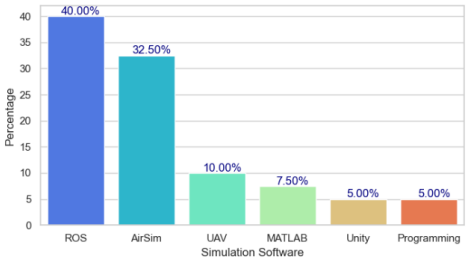


Figure 3.3: Distribution of simulation software used by papers that use reinforcement learning for UAVs by (AlMahamid & Grolinger, 2022)

The reinforcement learning algorithms can be divided into three classes, as (AlMahamid & Grolinger, 2021) did. The first class is limited in the number of states and has a discrete action space. The second class is not limited in the number of states but still has a discrete action space. The final class is not limited in the number of states and has a continuous action space. A discrete action space has a limited number of actions which can be selected. In contrast, a continuous action space provides a range from which the algorithm can select any number of actions.
The first class consists of two algorithms: Q-Learning, (Watkins & Dayan, 1992) and SARSA, (Rummery & Niranjan, 1994). Within Q-Learning, a Q-table has values for every possible combination of an action and state, which is updated by using the Bellman equation to obtain the action with the highest reward after every play. SARSA has a similar Q-table but uses an already-known next action to update the table.
The second class consists primarily of deep neural network variants of Q-Learning and SARSA. In its simplest form, the states are reduced by, for example, fully connected layers to an output of equal size to the action space. The output then corresponds to the Q-values for each action, (Mnih et al., 2013). More advanced algorithms compensate for overestimating the importance of the highest Q-value by DQN. Double DQN, (Van Hasselt et al., 2016), for instance, introduces policy and target networks. These kinds of algorithms cannot be used in the case of a continuous action space.
For the third class, a parameterised policy has to be learned to maximise an expected summation of discounted rewards. This is considered a maximisation problem, so similar to supervised learning, a gradient descent method can be used. The reward function looks like Equation 3.1, with state visitation probability ϕ_{π_θ}, state-value V^π, and action-value function Q^π, when following stochastic policy π_θ and is called the policy gradient theorem. The agents learn two policies: a target policy $\theta(a|s)$ and a behaviour policy $\beta(a|s)$ for learning the value function and choosing the action, respectively. An algorithm is on-policy when the target policy collects the training sample and calculates the expected reward. In the off-policy case, the behaviour policy obtains the training sample, while the target policy calculates the expected reward. Algorithms that solely focus on improving gradient descent performance are called policy-based algorithms. Trust Region Policy Optimization (TRPO), (Schulman et al., 2015), and Proximal Policy Optimization (PPO), (Schulman et al., 2017) are such algorithms that have seen use in the drone navigation domain as well. Both are on-policy, and the gradient descent is formalised like Equation 3.2. The off-policy case is formalised like Equation 3.3 and is called the off-policy gradient theorem. Actor-Critic algorithms make use of off-policy, where the Actor is used to find the optimal policy π_θ and the Critic to estimate the value function Q^{π_θ}. Deterministic Policy Gradients (DPG), (Silver et al., 2014), Deep Deterministic Policy Gradient (DDPG), (Lillicrap et al., 2015), Twin Delayed Deep Deterministic (TD3), (Fujimoto et al., 2018), change the stochastic policy to a deterministic policy. This changes the reward function to Equation 3.4 and the gradient to Equation 3.5. Soft Actor-Critic (SAC), (Haarnoja et al., 2018) adds the entropy to the maximisation. Actor-Critic with Experience Replay (ACER), (Wang et al., 2016) as the name indicates, uses experience replay. Finally, Asynchronous Advantage Actor-Critic (A3C), (Mnih et al., 2016), Advantage Actor-Critic, (Mnih et al., 2016) (A2C), Actor-Critic with Kronecker Factored Trust Region (ACKTR), (Wu et al., 2017) differentiate themselves as they are trained in a distributed manner with multiple agents working in parallel.

$$J(\pi_\theta) = \sum_{s\in S} \rho_{\pi_\theta}(s)\, V^{\pi_\theta}(s) = \sum_{s\in S} \rho_{\pi_\theta}(s) \sum_{a\in A} Q^{\pi_\theta}(s,a)\, \pi_\theta(a|s) \tag{3.1}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_\theta}, a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a)\, \nabla_\theta \ln \pi_\theta(a_t|s_t)\right] \tag{3.2}$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho^{\beta}, a\sim\beta}\left[\frac{\pi_\theta(a|s)}{\beta_\theta(a|s)} Q^{\pi_\theta}(s,a)\, \nabla_\theta \ln \pi_\theta(a_t|s_t)\right] \tag{3.3}$$

$$J_\beta(\mu_\theta) = \int_S \rho^\beta(s)\, V^\mu(s)\mathrm{d}s = \int_S \rho^\beta(s)\, Q^\mu(s, \mu_\theta(s))\,\mathrm{d}s \tag{3.4}$$

$$\nabla_\theta J_\beta(\mu_\theta) = \mathbb{E}_{s\sim\rho^\beta}\left[\nabla_\theta \mu_\theta(s)\, \nabla_a\, Q^\mu(s,a)|_{a=\mu_\theta(s)}\right] \tag{3.5}$$

3.1.3. Vision-Based Deep Reinforcement Learning

The navigation framework most relevant to us, as defined by (AlMahamid & Grolinger, 2022), is the vision-based framework. Therefore the general vision-based DRL domain is explored for additional content. First other simulation environments are discussed, followed by a sequence of papers, on the one hand explaining the usage of auxiliary tasks in this domain and the other hand further investigating DRL in relation to Vision-Based drone navigation.

Simulation Environments

As the previous decade crossed its halfway point, DRL was tested in more complicated environments for more complicated tasks. The navigational task received interest at the time, and DeepMind Lab, (Beattie et al., 2016), also known as Labyrinth, was developed. This three-dimensional environment was designed for artificial intelligence and machine learning systems and inspired by the game Quake III Arena. Reinforcement learning is natively integrated into DeepMind Lab, and the rendering uses OpenGL, (Shreiner, Group, et al., 2009). It is a first-person game with textures that are often dynamic and supports continuous motion. The learning agents learn to navigate through randomly generated mazes. These mazes consist of rooms and corridors, and an agent receives a reward upon finding apples and portals. Besides the rewards, the agent receives velocity, images and depth information, which is given in an RGB-D format with a resolution of $\mathbb{R}^{84\times84}$ at each timestep. The action space of this agent consists of 17 discrete actions and includes control movements, looking and tagging. Four types of levels can be generated. The first one consists of static maps for fruit gathering. The second types are static maps as well. They are used to train for navigation. The starting point is always randomised within these maps, and the location of the goal can be fixed or randomised. The third variant contains maps generated at the start of each episode. These maps test the ability of the agent to strategise and explore new environments. The final type contains laser-tag levels, in which bots are tagged and are supposed to imitate Quake III Arena gameplay.
The previous environment is for navigation purposes. However, the MuJoCo physics engine, (Todorov et al., 2012), is not. Its relevancy is its continuous action space, which architectures can take advantage of. The two outputs are two real number vectors: mean vector μ and scalar variance σ^2, creating a multidimensional normal distribution with a spherical covariance. A linear layer provides the mean vector, and a SoftPlus layer provides the scalar variance.
(AlMahamid & Grolinger, 2022) discussed both AirSim (Shah et al., 2018), and ROS, (Quigley et al., 2009). It, however, did not discuss Flightmare, (Song et al., 2020). Likely, due to the novelty and its limited use in literature, for now. Flightmare is a flexible quadrotor simulator. Its two main components are a configurable rendering engine and a flexible physics engine for dynamics simulation. It has a large multi-modal sensor suite and an API for reinforcement learning. This paper gives a demonstration by using deep reinforcement learning and collision-free path planning. Flightmare offers the ability to control photo-realism from low fidelity to high fidelity and can simulate sensor noise. On the physics side, the user controls the complexity of dynamics, from basic quadrotor models to advanced rigid-body dynamics. Besides, simulation can be performed in parallel, which enables fast data collection and training. This is important, especially for deep reinforcement learning. OpenAI Gym, (Brockman et al., 2016), wrappers are provided by the simulator. They state that OpenAI baselines, (Dhariwal et al., 2017), are integrated for reinforcement learning algorithms, although they refer to Stable-Baselines, (Hill et al., 2018) in their paper. This would be beneficial as this contains SAC and TD3. Besides, Flightmare is quite new, so patch updates including integration with Stable-Baselines3, (Raffin et al., 2021) are expected.
The novelty of this simulation environment means that the full capabilities of this framework have yet to be explored. (Loquercio et al., 2021a) showed some of the capabilities and came out of the same research group. Their objective is to fly autonomously through a complex environment by purely relying on vision and onboard computing. Although their objective is similar to this research, their approach significantly differs by using stereo matching and imitation learning from a privileged expert. This means that while training, the privileged expert requires full terrain knowledge and, consequently, has to have a zero-shot transfer from simulation to reality. For this, they require a series of simulation environments where the images obtained by the onboard cameras are as diverse as and similar to reality. They tested their algorithm in dense forests, snow-covered terrains, derailed trains and collapsed buildings and achieved good performance. The Flightmare simulator was used with the RotorS Gazebo plugin, (Furrer et al., 2016), and the rendering engine Unity, (Juliani et al.,

2018). Their environments were developed with off-the-shelf obstacles from the Unity environment [3]. These obstacles include simulated trees and a set of convex shapes such as ellipsoids, cuboids and cylinders. Both were randomised with a continuous uniform random distribution with $x \in \mathcal{U}(0.5, 4$, $y \in \mathcal{U}(0.5, 4)$, and $z \in \mathcal{U}(0.5, 8)$. Training environments spawn in either the trees or the convex shapes according to a homogeneous Poisson point process with intensity $\delta \in \mathcal{U}(4, 7)$. A total of 850 environments were created. Multiple examples of their environments are shown in Figure 3.4. Figure 3.5 shows an image and corresponding depth for one example. Their code is provided by (Loquercio et al., 2021b).

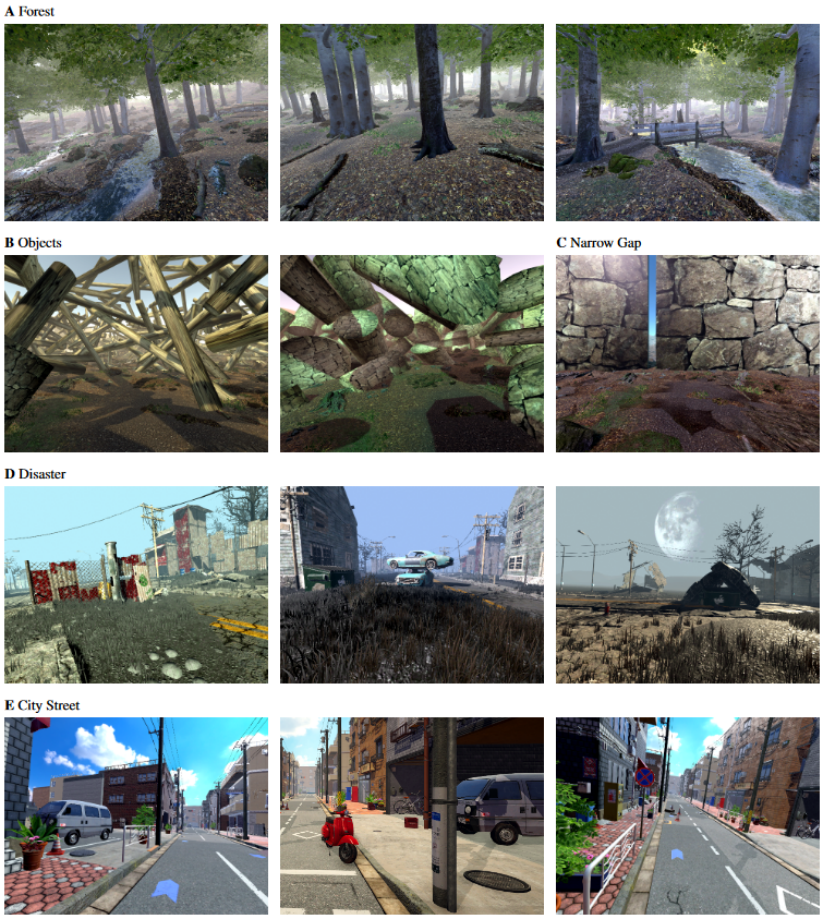


Figure 3.4: Simulation environments created in Flightmare by (Loquercio et al., 2021a)

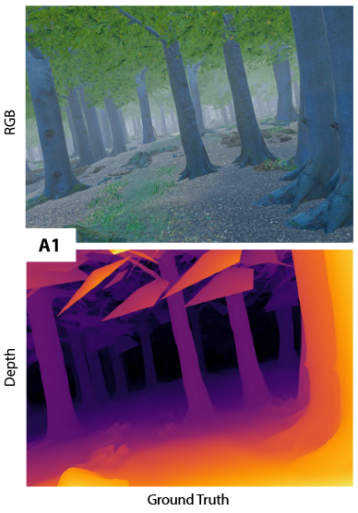


Figure 3.5: Depth and RGB images of a simulation environment in Flightmare by (Loquercio et al., 2021a)

Auxiliary Tasks and other Architecture Designs

An auxiliary task is, by definition, a second objective next to the main objective. The idea of this is that an auxiliary task can improve the main objective of a reinforcement learning agent.

Recurrent Reinforcement Learning: A Hybrid Approach
by (X. Li et al., 2015)

This paper proposes using supervised learning on an auxiliary task adjacent to reinforcement learning, used for the main objective. The term auxiliary task was not coined at this point, so the paper describes this as a hybrid approach instead. The objective of this paper is to take optimal actions to maximise total profits regarding customer relationship management. To reduce the problem of partial observability, they deploy a recurrent neural network optimised to predict observations and immediate rewards. This recurrent neural network can be trained by supervised learning and swapped with an LSTM model. A DQN uses the hidden-state representation resulting from this recurrent

[3] https://assetstore.unity.com/packages/3d/vegetation/forest-environment-dynamic-nature-150668

neural network to maximise long-term rewards. The DQN architecture shares many similarities with (Mnih et al., 2015). Their approach to training the RNN as an auxiliary task concurrent to reinforcement learning outperformed previous architectures significantly.

Reinforcement Learning with Unsupervised Auxiliary Tasks
by (Jaderberg et al., 2016)

(Jaderberg et al., 2016) realised the potential of auxiliary tasks to vision-based reinforcement learning. They opt for the simulation environment Labyrinth, by (Beattie et al., 2016). They call their algorithm UNREAL, trained with the A3C agent (Mnih et al., 2016) and has a CNN-LSTM architecture. They use a replay buffer for their auxiliary task, and the motivation to use auxiliary tasks is two-fold. The first part is similar to the previous paper in that they want to organise intermediate representations so that important context is more prevalent to the reinforcement learning agent. The second part has to do with the sparsity of their reward function. Their model can be adjusted consistently during training by introducing an auxiliary task. To better organise the intermediate representations, the objective of the different layers of a neural network has to be clear. For instance, the function of a convolutional layer is often related to image context, whereas the function of an LSTM is to capture the sequential relation between images. The LSTM layer is implemented after the convolutional layer to compare the sequential relation of image context. With the functionality in mind, different network parts can be trained in parallel. (Jaderberg et al., 2016) introduces reward prediction as an auxiliary task. For this task, it was decided that only the convolutional part of the network was relevant to prediction. As such, the optimisation due to this auxiliary task was purely on the convolutional part. They define pixel control as a second objective, which tries to maximise the change in pixel intensity. They think the sequential relation between these changes correlates to preferred navigation behaviour. Therefore both the CNN and LSTM are trained on this objective. Finally, a value function replay uses the CNN and LSTM to train the value function to promote faster value iterations. Their architecture is sketched in Figure 3.6. What is interesting about their auxiliary tasks is that they are completely unsupervised and are also trained with reinforcement learning.

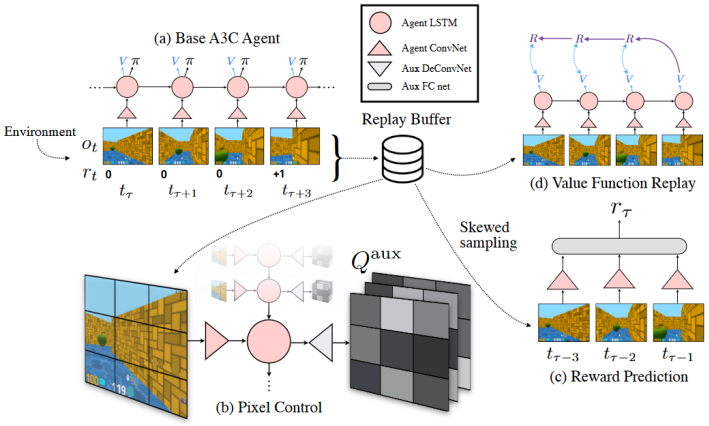


Figure 3.6: Architecture of a reinforcement learning algorithm using an A3C agent with pixel control, reward prediction and value function replay as auxiliary tasks by (Jaderberg et al., 2016)

Learning to Navigate in Complex Environments
by (Mirowski et al., 2016)

This paper and the previous paper share many similarities. For instance, both use the same simulation environment with Labyrinth, (Beattie et al., 2016), and the same algorithm with A3C, (Mnih et al., 2016). Auxiliary tasks are also implemented concurrently with the main objective. However, the tasks they choose are fairly different, as well as how they implement them. (Jaderberg et al., 2016) preferred tasks that are unsupervised. As a result, they are limited in how effectively they can organise their intermediate representations. This paper uses supervised auxiliary tasks but does not use any form of replay.

The algorithm receives a colour image as an input, which is encoded and goes through an LSTM for memory purposes. A second task, auxiliary to the main task, is to minimise the loss of inferring the depth map from the colour image, sharing the same encoder. This task is similar in spirit to monocular-to-stereo image synthesis, as both try to encode depth in the latent space, aiming to improve the algorithm's performance. A third task, again auxiliary to the main task, is to detect loop closures, which encourages implicit velocity integration. A weighted sum of the gradients from the A3C, depth predictions and loop closure is used to train the agent. The architecture is shown in Figure 3.7. Their observation s_t includes an image $\mathbf{x}_t \in \mathbb{R}^{3\times W\times H}$ with W and H being the image resolution, the previous reward $r_{t-1} \in \mathbb{R}$, the agent-relative lateral and rotational velocity $\mathbf{v}_t \in \mathbb{R}^6$, and the previous action $\mathbf{a}_{t-1} \in \mathbb{R}^{N_A}$. The previous reward and the encoder output are introduced in the first LSTM layer. In contrast, the agent-relative velocity and the previous action are introduced in the second LSTM layer together with the encoder output and the output of the first LSTM layer. It should be noted that depth is predicted twice in $g_d(\mathbf{f}_t)$ and in $g_d'(\mathbf{h}_t)$. Finally, $g_l(\mathbf{h}_t)$ predicts the loop closure. They note that they are specifically interested in data efficiency and therefore reduce the depth map to a resolution of 4×16. They also debate about defining depth as a regression or classification task. Finally, they conclude that supervised learning on depth information as an auxiliary task significantly improves the algorithm.

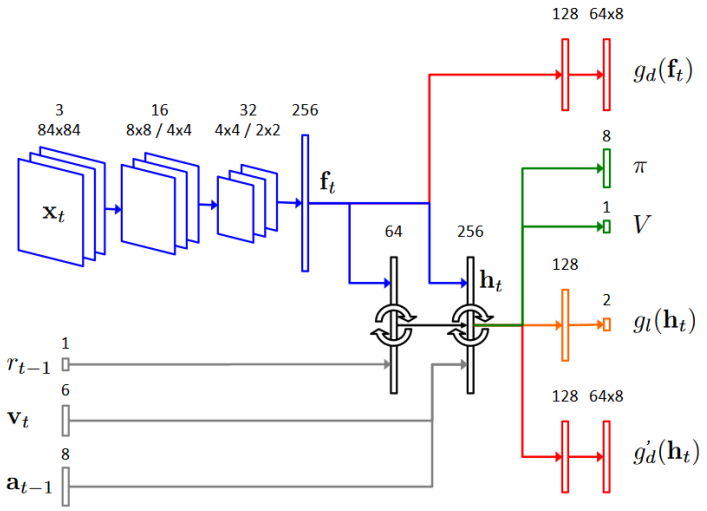


Figure 3.7: Architecture of a reinforcement learning algorithm with two depth predictions and loop detection as auxiliary tasks by (Mirowski et al., 2016)

(Shelhamer et al., 2016) discusses the different ways in which self-supervision can be implemented subsequent to reinforcement learning and the benefits of those methods. They note that self-supervised pre-training is advantageous. However, training concurrent to reinforcement learning does improve data efficiency. They note as well that the auxiliary task does not necessarily have to be accounted for in the reward function of the reinforcement learning agent.
(Bojarski et al., 2016) does not use reinforcement learning. Instead, it uses an end-to-end approach to create steering commands from input images. This paper is relevant as it shows other methods to achieve autonomous driving besides reinforcement learning. It also shows why reinforcement learning is such an effective tool for this problem. Their input consists of three simultaneous images from a left, centre, and right-orientated camera. These go through a normalisation layer, followed by convolutional layers to learn the image features and fully connected layers, which output a control value. The idea behind this structure is that the convolutional layers learn features while the fully connected layers act as a controller for steering. This end-to-end approach requires supervised learning, which in this case, means a training set of recorded steering commands. There are two problems. First, obtaining this data is time-consuming, but the second problem is that the trained network is fragile. This has to do with the controller copying the human behaviour, but if a slightly different choice is made, the consequence is that the camera inputs can be completely new. Therefore the network is unprepared for such situations and cannot take appropriate action. They do implement augmentation by adding artificial shifts and rotations to teach the network how to recover from poor position or

orientation, they say. This however still is very limiting, whereas a reinforcement learning agent can interact freely in an environment and will receive rewards based on behaviour and therefore is more suited for the control task.
(Y. Chen et al., 2019) follows up on (Bojarski et al., 2016) by introducing an auxiliary task to their architecture. Their auxiliary task is similar to monocular depth prediction and monocular-to-stereo image view synthesis in that the network has to learn image context to perform their respective tasks well. This paper implements image segmentation concurrent with the control task. It can be noted that an auxiliary task effectively means an extended loss function the network is optimised for. Therefore it can be concluded that many papers that were discussed in Chapter 2 do use auxiliary tasks. (Garg et al., 2016) and (Godard et al., 2017) use a disparity smoothness loss as an auxiliary task, while the codebook from (Esser et al., 2021) could be considered as such as well.
Instead of using an auxiliary task, layers from a deep reinforcement learning model could also be pre-trained on another task. (Chakravarty et al., 2017) is an example where the CNN is pre-trained on depth prediction. Afterwards, the weights and biases of the neurons within this network are copied to the reinforcement learning model, which is then trained on obstacle avoidance. (Yokoyama & Morioka, 2020) pre-trains their navigation network to predict depth and egomotion with a monocular camera in a self-supervised manner by using monocular videos. This method was proposed by (Casser et al., 2019).
All previous papers that do obstacle avoidance use a discrete action space, meaning the agent can choose between specific actions. (Shin et al., 2019) shows that a continuous action space improves performance significantly. The continuous action space provides a spectrum from which an agent selects a value. This agent has to be Actor-Critic, as other algorithms require discrete actions. In the case of the discrete action space, they use RGB and depth maps to two CNNs, while a U-Net segmentation model is used in the continuous action space. Their control command in the continuous action space are linear velocities in X, Y and Z-direction.
Previously organising the intermediate representation by using auxiliary tasks has been discussed. Other methods are used to organise the latent space, facilitating improved convergence. The Variational Autoencoder, (Kingma & Welling, 2013) is one of these mechanisms. (Xue & Gonsalves, 2021) implements this algorithm with a TD3, (Fujimoto et al., 2018), RL agent and a continuous action space. A convolutional variational autoencoder is used to predict depth from RGB images. Once trained, the encoder of this network outputs 32 variables and updates the actor-, target actor-, critic- and target critic-networks. Their actor networks contain four fully connected layers that output two values within a spectrum from -1 to 1, representing the velocity in the y-direction and z-direction. At the same time, the critic networks contain three fully connected layers, which estimate the Q value twice after two parallel fully connected layers. The Q value is obtained from the selected action and state input. Their reward function values collisions, reaching the destination, obstacle avoidance and distancing from obstacles. A replay buffer is used to save previous observations, and the simulation environment is Airsim, (Shah et al., 2018).
Although (Mirowski et al., 2016) already showed the benefits of LSTM, (Hochreiter & Schmidhuber, 1997), layers in their neural network architectures, it had not been used for vision-based drone navigation. (Singla et al., 2021) does use an LSTM layer combined with temporal attention, (Pei et al., 2017), to better process sequential information. Temporal attention values the importance of previous observations and results in improved training speed and better generalisability. A replay buffer is used as well for the stability of the algorithm. The main argument for these features is the partial observability that results from the monocular vision in a three-dimensional space. In comparison, they state that prior work such as (L. Xie et al., 2017) and (Sadeghi & Levine, 2017) assume that monocular vision, however, gives the complete picture. Like (Xue & Gonsalves, 2021), they implement a latent space regulator. However, this paper uses conditional GAN, (Isola et al., 2017), which uses a generator and a discriminator. The network is first trained on depth, like most previous papers. However, they first train on simulation images, then fine-tune the network with frozen lower layers on NYU2K, (Silberman et al., 2012). The depth images contain noise for more realistic training. As a simulation environment, they use Gazebo, (Koenig & Howard, 2004), and 22 different simulated indoor environments inspired by (Chakravarty et al., 2017). A Kinect sensor obtains RGB-D images in the simulation. Their reinforcement learning agent is DQN, and their action space is discrete, which includes going straight, turning right and turning left. Their reinforcement learning model is first trained in environments with lower complexity. The complexity is gradually increased by narrowing

down pathways and enclosing free space. Finally, they reward higher minimal distances and going straight. Another paper that uses the Gazebo environment is (Kim et al., 2022). They demonstrate the transfer of their learned policy to real flight experiments. D3QN is used as a reinforcement learning agent, and its reward function consists of a velocity component, a depth component and a collision component. Their action space has linear and angular velocities, with three and five discrete options, respectively.

Does computer vision matter for action?
by (B. Zhou et al., 2019)

B. Zhou et al. questioned the relevance of computer vision in conjunction with images to the task of sensorimotor control. Their assessment was performed in three environments: urban driving, off-road traversal and battle. The intermediate representations that were analysed were intrinsic surface colour, optical flow, depth and semantic segmentation. The results showed significantly improved performance with the inclusion of either depth or semantic segmentation to the observation space, whilst the inclusion of intrinsic surface colour added little to the performance. The impact of optical flow was also not that significant. (Sax et al., 2018) showed that adding other intermediate representations, such as curvature, denoising and occlusion edges can aid the performance even more.

3.2. Research Plan

This research aims to be a part of the overarching goal of developing an autonomous drone with a monocular camera able to navigate reliably through complex environments in real life. The method this research is developing is for a drone to achieve autonomous flight through reinforcement learning combined with supervised view synthesis on stereo cameras. The application further down the line is in real life. However, training a drone to fly in real life with reinforcement learning is tedious and expensive. Making sure that all components communicate with each other is often time-consuming already. In the case of a drone that has to learn how to fly and therefore crashes often, this problem becomes infinitely more time-consuming. Thus, the algorithm is first trained in a simulated environment. Besides, to understand the benefit and effectiveness of this training regime, tests are performed at rising levels of fidelity. In this case, the fidelity of two major components is considered: the dynamic model and the environmental realism. The dynamic model deals with the realism of flight. Advanced rigid body dynamics, such as friction and rotor drag or hardware properties, result in different fidelity levels. For navigation, the decision-making process is of the most interest. Realistic dynamics might make analysing this process more difficult in certain cases. Therefore a lower level of fidelity might be preferred in the early stages of development. On the contrary, a more complex, photo-realistic environment might aid the neural network in understanding the environment. At this point, it is also assumed that the advantage of training on stereo vision grows larger in more photo-realistic environments compared to training on LIDAR data or a single camera. The reward function will be focused on obstacle avoidance and therefore be sparse.

3.2.1. Simulation Environment

Most previously examined literature uses Gazebo, (Koenig & Howard, 2004) to develop their environments. However, these environments are often not at a photo-realistic level. Therefore the Flightmare quadrotor simulator, (Song et al., 2020), is chosen as the preferred simulator. As a fast, physically accurate and photo-realistic simulator, this fulfils the needs of the experiments. (Loquercio et al., 2021a) showed that the transfer to real life was excellent for their algorithm, which used Flightmare. This is due to realistic sensor functioning, such as noise, out-of-focus objects and dynamic lighting. Training on monocular vision, stereo vision and depth is possible. At the same time, it provides integration with OpenAI Gym, (Brockman et al., 2016), and OpenAI baselines, (Dhariwal et al., 2017), making the development of deep reinforcement learning applications with good RL algorithms possible. A minor concern is that it is not compatible with Stable-Baslines3, (Raffin et al., 2021), which means that the SAC, (Haarnoja et al., 2018), and TD3, (Fujimoto et al., 2018) algorithms are not built-in, making it more challenging to use them. Although it is preferred to have a state-of-the-art reinforcement learning agent, a slightly outdated agent does not interfere with the research objective. The assumed need for a continuous action space does, however, matter. Therefore the reinforcement learning agent has to be of the Actor-Critic class. Besides, decisions on the exact action space are also not made yet.

An action space consisting of three velocity vectors is seen in the literature. However, vectors that account for throttle and attitude might be more stable. To evaluate the performance of obstacle avoidance for Flightmare, AvoidBench[4] can be used. As this is developed within the TU Delft MAVlab group, in-house expertise is available.

3.2.2. Architecture

The main objective of this research is to study the effectiveness of implementing geometry-free monocular-to-stereo image view synthesis as an auxiliary task to the main navigation task. Therefore different architectures are tested that make a comparison feasible. The original architecture for the geometry-free monocular-to-stereo image view synthesis is shown in Figure 2.28. The latent space of this architecture consists of the bottleneck and four skip connections. These are concatenated to connect this to the second part of the model. This is then connected to an LSTM layer. The LSTM layer is a recurrent neural network. It is implemented as a one-to-one architecture, meaning that a single observation is provided to the LSTM and the temporal information it provides follows from the memory of the LSTM. Afterwards, this outputs the policy and value or whatever is required for the reinforcement learning agent. This deep reinforcement learning architecture is shown in Figure 3.8.

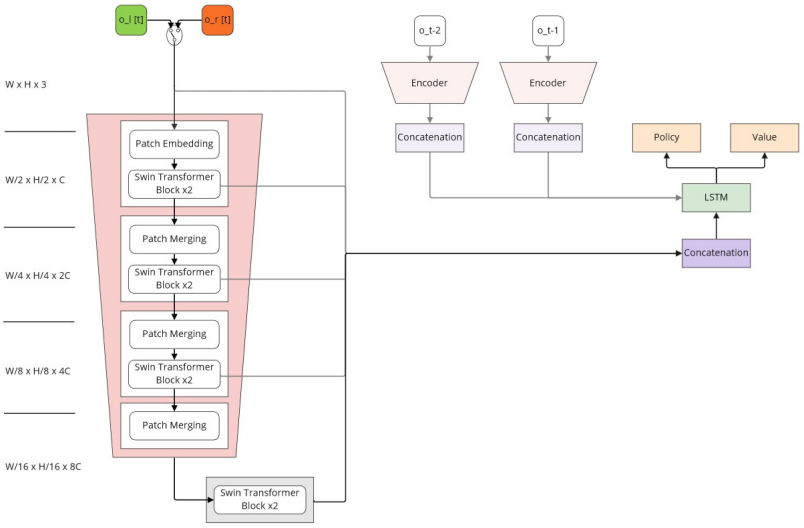


Figure 3.8: Deep Reinforcement Learning architecture for obstacle avoidance

A second configuration would be to pre-train the encoder in red and grey on the view synthesis task and take the weights and biases of the neurons and use these in this architecture. The view synthesis is incorporated as an auxiliary task by sharing the encoder and bottleneck, as shown in Figure 3.9. The impact of the skip connections on the convergence speed would be of interest, therefore removing the skip connections and concatenation layer would result in another architecture.
The general architecture can also be used for other auxiliary tasks. Monocular depth estimation could be trained on parallel to obstacle avoidance, as shown in Figure 3.10. A warping mechanism could be implemented as well for the monocular-to-stereo image view synthesis task, as shown in Figure 3.11.

Experiments

The architecture must be implemented in Flightmare, and experiments can be run afterwards. Gaining insight into the impact of the environmental complexity would be relevant to the subsequent experiments and will therefore be tested first. Therefore the first experiment will be an ablation study on the impact of environmental complexity as proposed in Table 3.1. From this, it is decided at what level the upcoming experiments will be performed.
After the first experiment, the main experiment can be performed by studying the impact of pre-training and implementing the auxiliary task to the deep reinforcement learning model. As proposed in Table 3.2, the experiment compares four architectures. The first architecture is just the deep reinforcement learning model. The second architecture uses the weights of a pre-trained encoder for the

[4]https://github.com/tudelft/AvoidBench [Accessed 28th November 2022]

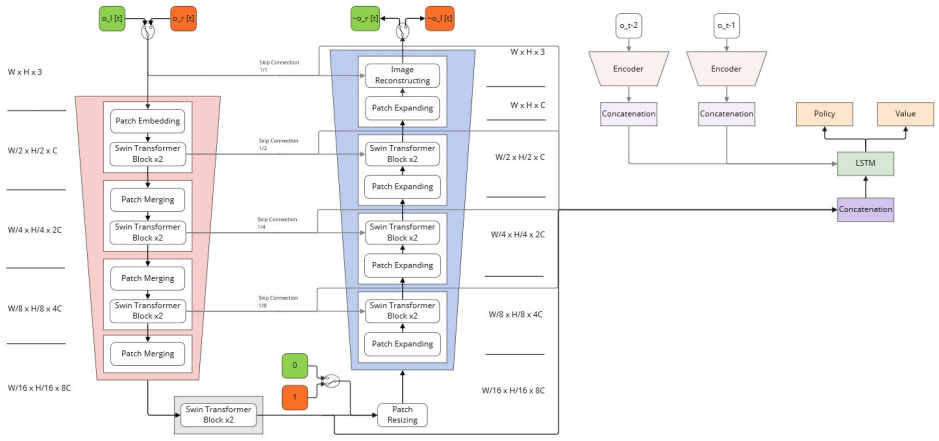


Figure 3.9: Deep Reinforcement Learning architecture for obstacle avoidance with an auxiliary task for monocular-to-stereo image view synthesis

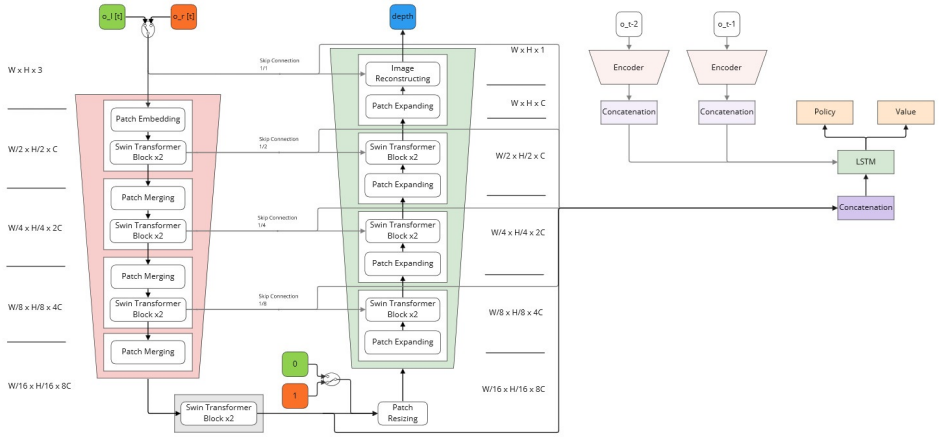


Figure 3.10: Deep Reinforcement Learning architecture for obstacle avoidance with an auxiliary task for monocular depth estimation

Table 3.1: Ablation studies on the impact of the environmental complexity

Environmental Complexity:	1. Low	2. Medium	3. High
RL Architecture:	pre-trained encoder and auxiliary task		
Architecture:	Swin-U-Net (patch size = 2)		
Simulation:	Flightmare		
Optimizer:	AdamW		
Loss:	Smooth L1 Loss		
Algorithm:	SAC/PPO		

deep reinforcement learning model. The pre-training is done on the supervised monocular-to-stereo image view synthesis task. Instead of pre-training, the third architecture extends the model with monocular-to-stereo image view synthesis as an auxiliary task. Finally, a model with a pre-trained encoder and the auxiliary task is trained. They are compared on performance and the speed of convergence.

The dynamic fidelity is also important. A higher fidelity is preferred as this reduces the gap to reality. Training will likely take longer as the reinforcement model has more difficulty controlling the drone. The expectation is that the auxiliary task will be most useful at high fidelity as this provides the most context for the drone to react to. The levels of fidelity are defined as low, medium and high, but exact definitions will be decided later. The experiment is shown in Table 3.3

It is relevant to know how the performance of the monocular-to-stereo image view synthesis as an auxiliary task compares to the tasks of monocular depth estimation and view synthesis by warping

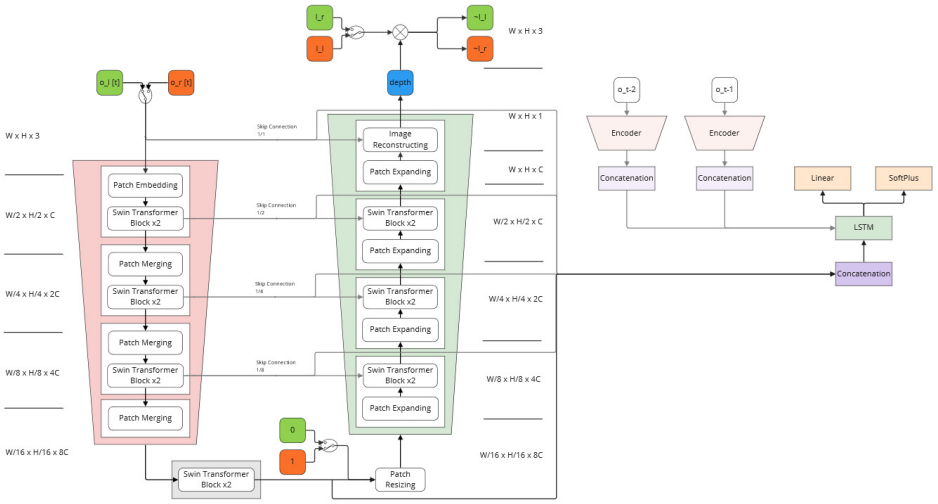


Figure 3.11: Deep Reinforcement Learning architecture for obstacle avoidance with an auxiliary task for monocular-to-stereo image view synthesis with a warping mechanism

Table 3.2: Ablation studies on the impact of pre-training and the auxiliary task

RL Architecture:	1. bare RL	2. pre-trained encoder	3. auxiliary task	4. pre-trained encoder and auxiliary task
Architecture:	Swin-U-Net (patch size = 2)			
Simulation:	Flightmare			
Optimizer:	AdamW			
Loss:	Smooth L1 Loss			
Algorithm:	SAC/PPO			

Table 3.3: Ablation studies on the impact of the dynamic fidelity

Fidelity:	1. Low	2. Medium	3. High
RL Architecture:	pre-trained encoder and auxiliary task		
Architecture:	Swin-U-Net (patch size = 2)		
Simulation:	Flightmare		
Optimizer:	AdamW		
Loss:	Smooth L1 Loss		
Algorithm:	SAC/PPO		

the original image to a target image. This experiment is shown in Table 3.4.

Table 3.4: Ablation studies on the impact of the type of auxiliary task

Decoder Prediction:	1. Stereo Image	2. Depth	3. Stereo Image by Warp
RL Architecture:	pre-trained encoder and auxiliary task		
Architecture:	Swin-U-Net (patch size = 2)		
Simulation:	Flightmare		
Optimizer:	AdamW		
Loss:	Smooth L1 Loss		
Algorithm:	SAC/PPO		

Finally, the impact of the skip connections is studied. The performance of the view synthesis task improves with the inclusion of skip connections. The latent space, however, increases, meaning that the size of the LSTM layer increases as well. This could reduce the efficiency of the network. Therefore an analysis is done with Table 3.5.

Table 3.5: Ablation studies on the impact of the skip connections

Skip connections:	1. All	2. None
RL Architecture:	pre-trained encoder and auxiliary task	
Architecture:	Swin-U-Net (patch size = 2)	
Simulation:	Flightmare	
Optimizer:	AdamW	
Loss:	Smooth L1 Loss	
Algorithm:	SAC/PPO	

4

Conclusion

The literature study serves as a guide for the thesis to fall back on and to provide the basis for decisions and goals. Within the introduction, in Chapter 1, the research question is developed utilizing a problem statement and general context of the research domain. After defining the question, the different aspects are dismantled and discussed. With more detailed domain knowledge, a general plan is set out to achieve the goal of the thesis. The subsequent chapters, Chapter 2 and Chapter 3 delve into the two main stages required to answer the thesis question. The literature overview covers the research and relevant aspects of those fields. By presenting the historical context and connecting this to the currently relevant research, the reader is given the perspective required to understand the motivation and decision-making process in this literature review. Both the applicability and limitations of the presented literature are discussed, as well as the coherence and difference between the different subdomains.

Context is provided to motivate the research plan for the geometry-free monocular-to-stereo image view synthesis. First, the historical context of computer vision, neural networks and hardware are discussed. This is followed by a presentation of the current state-of-the-art in computer vision. The architectures used, their limitations and their potential are also discussed. An in-depth analysis of the relevant fields for monocular-to-stereo image view synthesis is performed. The clearly defined subdomains adjacent to this topic are discussed as well as the emergence of the new research domain.

Preliminary results following the research plan for the geometry-free monocular-to-stereo image view synthesis are shown and discussed. Benchmarked on the KITTI dataset, the performance on reconstruction realism is good for architectures including CNNs and architectures including Swin Transformers. Skip connections improve the performance of the network. However, results without skip connections are still fairly good. Through stereo matching, it can be assumed that depth is encoded due to the similarity of the disparity maps for prediction and target images. It is concluded that the architecture is viable to be integrated as an auxiliary task to navigation.

Deep reinforcement learning is used in many applications in today's world. It is discussed why the possibilities are endless. The use of reinforcement learning within drone navigation and more general vision-based problems are explored. The papers provide a base to motivate the choice for the simulation environment, type of reinforcement learning agent and proposed architectures. It is discussed how the monocular-to-stereo image view synthesis task can be implemented to facilitate quicker convergence of the navigation objective. Finally, clear objectives and experiments are defined in the research plan.

To conclude, the literature studies define the research question for the subsequent thesis: How can geometry-free monocular-to-stereo image view synthesis be used as an auxiliary task to improve the navigation task performance and data efficiency of the reinforcement learning agent from a monocular vision-based drone? Within the literature studies, a method is developed and built on arguments provided through knowledge from many papers in the research domains of computer vision and UAVs. The following experiments are set out to test all critical aspects of the method on quality and functionality. The literature review finishes by providing a plan containing target deadlines to finish the main objectives.

References

Akhloufi, M. A., Arola, S., & Bonnet, A. (2019). Drones chasing drones: Reinforcement learning and deep search area proposal. *Drones*.

Aleotti, F., Tosi, F., Poggi, M., & Mattoccia, S. (2019). Generative adversarial networks for unsupervised monocular depth prediction. In L. Leal-Taixé & S. Roth (Eds.), *Computer vision – eccv 2018 workshops* (pp. 337–354). Springer International Publishing.

AlMahamid, F., & Grolinger, K. (2021). Reinforcement learning algorithms: An overview and classification. *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 1–7. https://doi.org/10.1109/CCECE53047.2021.9569056

AlMahamid, F., & Grolinger, K. (2022). Autonomous unmanned aerial vehicle navigation using reinforcement learning: A systematic review. *Engineering Applications of Artificial Intelligence*, *115*, 105321. https://doi.org/https://doi.org/10.1016/j.engappai.2022.105321

Andrew, W., Greatwood, C., & Burghardt, T. (2018). Deep learning for exploration and recovery of uncharted and dynamic targets from uav-like vision. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1124–1131. https://doi.org/10.1109/IROS.2018.8593751

Asimov, I. (1950). *I, robot*. Gnome Press.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. https://doi.org/10.48550/ARXIV.1607.06450

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D., . . . Petersen, S. (2016). Deepmind lab. *CoRR*, *abs/1612.03801*. http://arxiv.org/abs/1612.03801

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). End to end learning for self-driving cars. *CoRR*, *abs/1604.07316*. http://arxiv.org/abs/1604.07316

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *CoRR*, *abs/1606.01540*. http://arxiv.org/abs/1606.01540

Brostow, G. J., Fauqueur, J., & Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database [Video-based Object and Event Analysis]. *Pattern Recognition Letters*, *30*(2), 88–97. https://doi.org/https://doi.org/10.1016/j.patrec.2008.04.005

Butler, D. J., Wulff, J., Stanley, G. B., & Black, M. J. (2012). A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, & C. Schmid (Eds.), *Computer vision – eccv 2012* (pp. 611–625). Springer Berlin Heidelberg.

Cao, H., Wang, Y., Chen, J., Jiang, D., Zhang, X., Tian, Q., & Wang, M. (2021). Swin-unet: Unet-like pure transformer for medical image segmentation. *arXiv preprint arXiv:2105.05537*.

Casser, V., Pirk, S., Mahjourian, R., & Angelova, A. (2019). Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. https://doi.org/10.1609/aaai.v33i01.33018001

Chakravarty, P., Kelchtermans, K., Roussel, T., Wellens, S., Tuytelaars, T., & Van Eycken, L. (2017). Cnn-based single image obstacle avoidance on a quadrotor. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 6369–6374. https://doi.org/10.1109/ICRA.2017.7989752

Chen, M., Radford, A., Wu, J., Jun, H., Dhariwal, P., Luan, D., & Sutskever, I. (2020). Generative pretraining from pixels. *International Conference on Machine Learning*.

Chen, Y., Praveen, P., Priyantha, M., Muelling, K., & Dolan, J. (2019). Learning on-road visual control for self-driving vehicles with auxiliary tasks. *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 331–338. https://doi.org/10.1109/WACV.2019.00041

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1251–1258.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). *arXiv: Learning*.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3213–3223. https://doi.org/10.1109/CVPR.2016.350

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*, 248–255.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y., & Zhokhov, P. (2017). Openai baselines.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, *abs/2010.11929*. https://arxiv.org/abs/2010.11929

Dosovitskiy, A., Springenberg, J. T., & Brox, T. (2015). Learning to generate chairs with convolutional neural networks. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1538–1546. https://doi.org/10.1109/CVPR.2015.7298761

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, *12*(null), 2121–2159.

Eigen, D., Puhrsch, C., & Fergus, R. (2014). Depth map prediction from a single image using a multi-scale deep network. *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, 2366–2374.

Esser, P., Rombach, R., & Ommer, B. (2021). Taming transformers for high-resolution image synthesis. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 12868–12878. https://doi.org/10.1109/CVPR46437.2021.01268

Fitch, F. B. (1944). Warren s. mcculloch and walter pitts. a logical calculus of the ideas immanent in nervous activity. bulletin of mathematical biophysics, vol. 5 (1943), pp. 115–133. *Journal of Symbolic Logic*, *9*(2), 49–50. https://doi.org/10.2307/2268029

Flynn, J., Neulander, I., Philbin, J., & Snavely, N. (2016). Deep stereo: Learning to predict new views from the world's imagery. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5515–5524. https://doi.org/10.1109/CVPR.2016.595

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing function approximation error in actor-critic methods. *International conference on machine learning*, 1587–1596.

Furrer, F., Burri, M., Achtelik, M., & Siegwart, R. (2016). Rotors—a modular gazebo mav simulator framework. In A. Koubaa (Ed.), *Robot operating system (ros): The complete reference (volume 1)* (pp. 595–625). Springer International Publishing. https://doi.org/10.1007/978-3-319-26054-9_23

Garg, R., B.G., V. K., Carneiro, G., & Reid, I. (2016). Unsupervised cnn for single view depth estimation: Geometry to the rescue. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer vision – eccv 2016* (pp. 740–756). Springer International Publishing.

Geiger, A., Lenz, P., & Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 3354–3361. https://doi.org/10.1109/CVPR.2012.6248074

Godard, C., Aodha, O. M., Firman, M., & Brostow, G. (2019). Digging into self-supervised monocular depth estimation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 3827–3837. https://doi.org/10.1109/ICCV.2019.00393

Godard, C., Aodha, O. M., & Brostow, G. J. (2017). Unsupervised monocular depth estimation with left-right consistency. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6602–6611. https://doi.org/10.1109/CVPR.2017.699

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144.

Gupta, A., Dollar, P., & Girshick, R. (2019). Lvis: A dataset for large vocabulary instance segmentation. *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 5356–5364.

Ha, D., & Schmidhuber, J. (2018). World models. *arXiv preprint arXiv:1803.10122*.

Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., et al. (2018). Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778. https://doi.org/10.1109/CVPR.2016.90

He, L., Aouf, N., Whidborne, J. F., & Song, B. (2020). Integrated moment-based lgmd and deep reinforcement learning for uav obstacle avoidance. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 7491–7497.

Hendrycks, D., & Gimpel, K. (2016). Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, *abs/1606.08415*. http://arxiv.org/abs/1606.08415

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Klambauer, G., & Hochreiter, S. (2017). Gans trained by a two time-scale update rule converge to a nash equilibrium. *CoRR*, *abs/1706.08500*. http://arxiv.org/abs/1706.08500

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., & Wu, Y. (2018). Stable baselines.

Hinton, G., Srivastava, N., & Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, *14*(8), 2.

Hinton, G., Vinyals, O., Dean, J., et al. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, *2*(7).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Huynh-Thu, Q., & Ghanbari, M. (2012). The accuracy of psnr in predicting video quality for different video scenes and frame rates. *Telecommun. Syst.*, *49*(1), 35–48. https://doi.org/10.1007/s11235-010-9351-x

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, 448–456.

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5967–5976. https://doi.org/10.1109/CVPR.2017.632

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., & Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *CoRR*, *abs/1611.05397*. http://arxiv.org/abs/1611.05397

Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2015). Spatial transformer networks. *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, 2017–2025.

Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2018). Unity: A general platform for intelligent agents. *CoRR*, *abs/1809.02627*. http://arxiv.org/abs/1809.02627

Karis, B., & Games, E. (2013). Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, *4*(3), 1.

Kim, M., Kim, J., Jung, M., & Oh, H. (2022). Towards monocular vision-based autonomous flight through deep reinforcement learning. *Expert Systems with Applications*, *198*, 116742. https://doi.org/https://doi.org/10.1016/j.eswa.2022.116742

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *CoRR*, *abs/1412.6980*.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. https://doi.org/10.48550/ARXIV.1312.6114

Koenig, N., & Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, *3*, 2149–2154 vol.3. https://doi.org/10.1109/IROS.2004.1389727

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*, 84–90.

Li, J., Feng, Z., She, Q., Ding, H., Wang, C., & Lee, G. H. (2021). Nemi: Unifying neural radiance fields with multiplane images for novel view synthesis. *CoRR*, *abs/2103.14910*. https://arxiv.org/abs/2103.14910

Li, X., Li, L., Gao, J., He, X., Chen, J., Deng, L., & He, J. (2015). *Recurrent reinforcement learning: A hybrid approach* (ArXiv, tech. rep. MSR-TR-2015-98) [Proceedings of COLING-94, Kyoto, Japan].

https://www.microsoft.com/en-us/research/publication/recurrent-reinforcement-learning-a-joint-training-approach/

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Lin, A., Chen, B., Xu, J., Zhang, Z., Lu, G., & Zhang, D. (2022). Ds-transunet: Dual swin transformer u-net for medical image segmentation. *IEEE Transactions on Instrumentation and Measurement*, *71*, 1–15. https://doi.org/10.1109/TIM.2022.3178991

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. *European conference on computer vision*, 740–755.

Liu, A., Makadia, A., Tucker, R., Snavely, N., Jampani, V., & Kanazawa, A. (2021). Infinite nature: Perpetual view generation of natural scenes from a single image. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 14438–14447. https://doi.org/10.1109/ICCV48922.2021.01419

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 9992–10002. https://doi.org/10.1109/ICCV48922.2021.00986

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3431–3440. https://doi.org/10.1109/CVPR.2015.7298965

Loquercio, A., Kaufmann, E., Ranftl, R., Müller, M., Koltun, V., & Scaramuzza, D. (2021a). Learning high-speed flight in the wild. *Science Robotics*, *6*(59), eabg5810. https://doi.org/10.1126/scirobotics.abg5810

Loquercio, A., Kaufmann, E., Ranftl, R., Müller, M., Koltun, V., & Scaramuzza, D. (2021b). Learning high-speed flight in the wild. *Science Robotics*.

Loshchilov, I., & Hutter, F. (2016). SGDR: stochastic gradient descent with restarts. *CoRR*, *abs/1608.03983*. http://arxiv.org/abs/1608.03983

Loshchilov, I., & Hutter, F. (2019). Decoupled weight decay regularization. *ICLR*.

Luo, Y., Ren, J., Lin, M., Pang, J., Sun, W., Li, H., & Lin, L. (2018). Single view stereo matching. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 155–163. https://doi.org/10.1109/CVPR.2018.00024

Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., & Brox, T. (2016). A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4040–4048. https://doi.org/10.1109/CVPR.2016.438

Menze, M., & Geiger, A. (2015). Object scene flow for autonomous vehicles. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3061–3070.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. In A. Vedaldi, H. Bischof, T. Brox, & J.-M. Frahm (Eds.), *Computer vision – eccv 2020* (pp. 405–421). Springer International Publishing.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., & Hadsell, R. (2016). Learning to navigate in complex environments. *CoRR*, *abs/1611.03673*. http://arxiv.org/abs/1611.03673

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. A. (2013). Playing atari with deep reinforcement learning. *CoRR*, *abs/1312.5602*. http://arxiv.org/abs/1312.5602

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Moore, G. E., et al. (1965). Cramming more components onto integrated circuits.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 807–814.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch.

Pei, W., Baltrušaitis, T., Tax, D. M. J., & Morency, L.-P. (2017). Temporal attention-gated model for robust sequence classification. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 820–829. https://doi.org/10.1109/CVPR.2017.94

Peng, R., Wang, R., Lai, Y., Tang, L., & Cai, Y. (2021). Excavating the potential capacity of self-supervised monocular depth estimation. *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 15540–15549.

Pilzer, A., Xu, D., Puscas, M., Ricci, E., & Sebe, N. (2018). Unsupervised adversarial depth estimation using cycled generative networks. *2018 International Conference on 3D Vision (3DV)*, 587–595. https://doi.org/10.1109/3DV.2018.00073

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. (2009). Ros: An open-source robot operating system. *ICRA workshop on open source software*, *3*(3.2), 5.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268), 1–8. http://jmlr.org/papers/v22/20-1364.html

Rombach, R., Esser, P., & Ommer, B. (2021). Geometry-free view synthesis: Transformers and no 3d priors. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 14356–14366.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.), *Medical image computing and computer-assisted intervention – miccai 2015* (pp. 234–241). Springer International Publishing.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, *65 6*, 386–408.

Rummery, G. A., & Niranjan, M. (1994). *On-line q-learning using connectionist systems* (Vol. 37). University of Cambridge, Department of Engineering Cambridge, UK.

Sadeghi, F., & Levine, S. (2017). Cad2rl: Real single-image flight without a single real image. *Proceedings of Robotics: Science and Systems*. https://doi.org/10.15607/RSS.2017.XIII.034

Sajjadi, M. S. M., Meyer, H., Pot, E., Bergmann, U., Greff, K., Radwan, N., Vora, S., Lucic, M., Duckworth, D., Dosovitskiy, A., Uszkoreit, J., Funkhouser, T. A., & Tagliasacchi, A. (2021). Scene representation transformer: Geometry-free novel view synthesis through set-latent scene representations. *CoRR*, *abs/2111.13152*. https://arxiv.org/abs/2111.13152

Sax, A., Emi, B., Zamir, A., Guibas, L. J., Savarese, S., & Malik, J. (2018). Mid-level visual representations improve generalization and sample efficiency for learning active tasks. *CoRR*, *abs/1812.11971*. http://arxiv.org/abs/1812.11971

Saxena, A., Chung, S., & Ng, A. (2005). Learning depth from single monocular images. In Y. Weiss, B. Schölkopf, & J. Platt (Eds.), *Advances in neural information processing systems*. MIT Press. https://proceedings.neurips.cc/paper/2005/file/17d8da815fa21c57af9829fb0a869602-Paper.pdf

Saxena, A., Sun, M., & Ng, A. Y. (2009). Make3d: Learning 3d scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*(5), 824–840. https://doi.org/10.1109/TPAMI.2008.132

Scharstein, D., Szeliski, R., & Zabih, R. (2001). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, 131–140. https://doi.org/10.1109/SMBV.2001.988771

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. *International conference on machine learning*, 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shade, J., Gortler, S., He, L.-w., & Szeliski, R. (1998). Layered depth images. *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 231–242.

Shah, S., Dey, D., Lovett, C., & Kapoor, A. (2018). Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *Field and service robotics*, 621–635.

Shelhamer, E., Mahmoudieh, P., Argus, M., & Darrell, T. (2016). Loss is its own reward: Self-supervision for reinforcement learning. *CoRR*, *abs/1612.07307*. http://arxiv.org/abs/1612.07307

Shin, S.-Y., Kang, Y.-W., & Kim, Y.-G. (2019). Obstacle avoidance drone by deep reinforcement learning and its racing with human pilot. *Applied Sciences*, *9*(24). https://doi.org/10.3390/app9245571

Shreiner, D., Group, B. T. K. O. A. W., et al. (2009). *Opengl programming guide: The official guide to learning opengl, versions 3.0 and 3.1*. Pearson Education.

Silberman, N., Hoiem, D., Kohli, P., & Fergus, R. (2012). Indoor segmentation and support inference from rgbd images. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, & C. Schmid (Eds.), *Computer vision – eccv 2012* (pp. 746–760). Springer Berlin Heidelberg.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). Deterministic policy gradient algorithms. *International conference on machine learning*, 387–395.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *CoRR*, *abs/1409.1556*.

Singla, A., Padakandla, S., & Bhatnagar, S. (2021). Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge. *IEEE Transactions on Intelligent Transportation Systems*, *22*(1), 107–118. https://doi.org/10.1109/TITS.2019.2954952

Song, Y., Naji, S., Kaufmann, E., Loquercio, A., & Scaramuzza, D. (2020). Flightmare: A flexible quadrotor simulator. *CoRR*, *abs/2009.00563*. https://arxiv.org/abs/2009.00563

Tatarchenko, M., Dosovitskiy, A., & Brox, T. (2016). Multi-view 3d models from single images with a convolutional network. *European Conference on Computer Vision*, 322–337.

Todorov, E., Erez, T., & Tassa, Y. (2012). Mujoco: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

Tucker, R., & Snavely, N. (2020). Single-view view synthesis with multiplane images. *CVPR*. https://arxiv.org/abs/2004.11364

Tulsiani, S., Tucker, R., & Snavely, N. (2018). Layer-structured 3d scene inference via view synthesis. In V. Ferrari, M. Hebert, C. Sminchisescu, & Y. Weiss (Eds.), *Computer vision – eccv 2018* (pp. 311–327). Springer International Publishing.

van den Oord, A., Vinyals, O., & Kavukcuoglu, K. (2017). Neural discrete representation learning. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 6309–6318.

Van Dijk, T., & De Croon, G. (2019). How do neural networks see depth in single images? *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2183–2191. https://doi.org/10.1109/ICCV.2019.00227

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the AAAI conference on artificial intelligence*, *30*(1).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. *CoRR*, *abs/1706.03762*. http://arxiv.org/abs/1706.03762

Walk, R. D., & Dodge, S. H. (1962). Visual depth perception of a 10-month-old monocular human infant. *Science*, *137*(3529), 529–530. https://doi.org/10.1126/science.137.3529.529

Wang, Z., Simoncelli, E., & Bovik, A. (2003). Multiscale structural similarity for image quality assessment. *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, *2*, 1398–1402 Vol.2. https://doi.org/10.1109/ACSSC.2003.1292216

Wang, Z., Bovik, A., Sheikh, H., & Simoncelli, E. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, *13*(4), 600–612. https://doi.org/10.1109/TIP.2003.819861

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., & de Freitas, N. (2016). Sample efficient actor-critic with experience replay. *CoRR*, *abs/1611.01224*. http://arxiv.org/abs/1611.01224

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine learning*, *8*(3), 279–292.

Wu, Y., Mansimov, E., Liao, S., Grosse, R., & Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 5285–5294.

Xie, J., Girshick, R., & Farhadi, A. (2016). Deep3d: Fully automatic 2d-to-3d video conversion with deep convolutional neural networks. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer vision – eccv 2016* (pp. 842–857). Springer International Publishing.

Xie, L., Wang, S., Markham, A., & Trigoni, N. (2017). Towards monocular vision based obstacle avoidance through deep reinforcement learning. *CoRR*, *abs/1706.09829*. http://arxiv.org/abs/1706.09829

Xue, Z., & Gonsalves, T. (2021). Monocular vision obstacle avoidance uav: A deep reinforcement learning method. *2021 2nd International Conference on Innovative and Creative Information Technology (ICITech)*, 1–6. https://doi.org/10.1109/ICITech50181.2021.9590178

Yamaguchi, K., McAllester, D., & Urtasun, R. (2014). Efficient joint segmentation, occlusion labeling, stereo and flow estimation. *ECCV*.

Yokoyama, K., & Morioka, K. (2020). Autonomous mobile robot with simple navigation system based on deep reinforcement learning and a monocular camera. *2020 IEEE/SICE International Symposium on System Integration (SII)*, 525–530. https://doi.org/10.1109/SII46433.2020.9025987

Zhang, C., Lin, C., Liao, K., Nie, L., & Zhao, Y. (2022). Sivsformer: Parallax-aware transformers for single-image-based view synthesis. *2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 47–56. https://doi.org/10.1109/VR51125.2022.00022

Zhang, C., Lin, C., Liao, K., Nie, L., & Zhao, Y. (2023). As-deformable-as-possible single-image-based view synthesis without depth prior. *IEEE Transactions on Circuits and Systems for Video Technology*.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., & Wang, O. (2018). The unreasonable effectiveness of deep features as a perceptual metric. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 586–595. https://doi.org/10.1109/CVPR.2018.00068

Zhang, Y., & Wu, J. (2022). Video extrapolation in space and time. *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XVI*, 313–333.

Zhou, B., Zhao, H., Puig, X., Fidler, S., Barriuso, A., & Torralba, A. (2017). Scene parsing through ade20k dataset. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 633–641.

Zhou, B., Krähenbühl, P., & Koltun, V. (2019). Does computer vision matter for action? *Science Robotics*, *4*(30), eaaw6661. https://doi.org/10.1126/scirobotics.aaw6661

Zhou, T., Tucker, R., Flynn, J., Fyffe, G., & Snavely, N. (2018). Stereo magnification: Learning view synthesis using multiplane images. *ACM Trans. Graph.*, *37*(4). https://doi.org/10.1145/3197517.3201323

Zhou, T., Tulsiani, S., Sun, W., Malik, J., & Efros, A. A. (2016). View synthesis by appearance flow. In B. Leibe, J. Matas, N. Sebe, & M. Welling (Eds.), *Computer vision – eccv 2016* (pp. 286–301). Springer International Publishing.