Detecting Empty Wireframe Objects on Micro-Air Vehicles

Applied for Gate Detection in Autonomous Drone Racing

by

P. Duernay

in partial fulfillment of the requirements for the degree of

Master of Science

in Embedded Systems

at the Delft University of Technology,

to be defended publicly on Tuesday December 18, 2018 at 2:00 PM.

Supervisor:	Dr. D. M. J Tax	
Thesis committee:	Prof. Dr. Ir. M. J. T. Reinders,	TU Delft
	Dr. G. C. de Croon,	TU Delft

This thesis is confidential and cannot be made public until December 18, 2018.

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

TU Delft Delft University of Technology

Acknowledgements

I would first like to thank my thesis supervisors Dr. David M.J. Tax and Dr. Guido C. de Croon for the guidance throughout the project. The meetings helped me steering the thesis in the right direction while leaving enough freedom for my own choices. Any questions I could always ask.

Furthermore, I would like to thank fellow colleagues that were involved in the implementation on the Micro-Air Vehicle (MAV) and the Autonomous Drone Race. First and foremost Shuo Li for helping me with the experiments and for all the pizza we shared together. Also, Jihaio Lin, Christoph de Wagter, Simon Spronk and all the technical staff of the *MAVLab*.

Finally, I must express my very profound gratitude to my parents and friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Author

Dürnay, Philipp

Contents

Summary

Autonomous MAVs are an emerging technology that supports a wide range of applications such as medical delivery or finding survivors in disaster scenarios. As flying in such missions is difficult the robust estimation of an MAV's state within its environment is crucial to ensure safe operation.

In indoor scenarios, cameras are one of the predominant choices for state estimation sensors. This requires Computer Vision algorithms to interpret the obtained high dimensional signal. An application that allows the competitive evaluation of control and state estimation algorithms is MAV Racing such as the International Conference of Intelligent Robots (IROS) 2018 Autonomous Drone Race. Thereby a race court consisting of several race gates has to be followed. For a fast flight during such a race court the detection of the racing gates with a camera can be used in a high level control loop. As these objects consist only of small structures that are spread across large parts of the image, this gives rise to a challenging Object Detection problem.

In recent years CNNs showed promising results on various vision tasks. However, due to their computational complexity the deployment on mobile devices remains a challenge. Furthermore, CNNs typically require a vast amount of training data. Finally, the objects typically studied in Object Detection consist of solid and complex features which is not the case for racing gates. Therefore, this work defines the class of Empty Wire Frame Objectss (EWFOs) and studies their detection on MAVs with You only look once (Yolo)V3. Thereby, the training data is created with a graphical engine. We are interested in how to detect EWFOs with a CNN on a MAV, using synthetic data.

We conduct several simple experiments about EWFOs in simulation and compare their detection to more filled objects. Subsequently experiments in a more challenging environment such as an MAV race are conducted.

The experiments show how EWFOs are harder to detect than filled objects as the detector can be confused to patterns present in the empty part. Particularly for larger objects the detection performance decreases. We give several recommendations on how to generate data for the detection of EWFOs on MAVs. These include how to add variations in background as well as the camera placement. Finally, we study the incorporation of image augmentation techniques to transfer the detector to the real world. We can report that especially modelling lens distortion improves the performance on the real data. Nevertheless, a reality gap remains that can not fully be explained.

Furthermore, different architectures are studied for the detection of EWFOs. It can be seen how a relatively shallow network of 9 layers can be used for the detection of EWFOs on MAVs. A further reduction in weights leads to a gradual decrease in performance. Based on the gained insights the deployment of a detector on the example system *JeVois* is studied. A detection performance/speed trade-off is evaluated. The final detector achieves 32% ap_{60} at a frame rate of 12 Hz on a real world test set created during this work.

The gained insights can be used to deploy the detector in a control loop for MAVs. This ensures the safe flight through a racing court of an autonmous drone race. The gained insights about the detection of EWFOs can be transferred to objects with similar properties.

Glossary

AP Average Precision

CAD Computer Aided Design

CNN Convolutional Neural Network

CPU Central Processing UniT

DPM Deformable Part Model

DR Domain Randomization

DSC Depthwise Separable Convolution

EWFO Empty Wire Frame Objects

FoV Field of View

FPV First Person View

GAN Generative Adversial Network

GPS Global Positioning System

GPU Graphical Processing Unit

HOG Histogram of Oriented Gradients

HSV Hue, Saturation, Value

IR Infrared

IMU Inertial Measurement Unit

IROS International Conference of Intelligent Robots

LIDAR Light Detection And Ranging

MAV Micro-Air Vehicle

mAP Mean Average Precision

NED North-East-Down

IoU Intersection over Union

FCN Fully Convolutional Network

PCA Principal Component Analysis

RPN Region Proposal Network

SIMD Single Instruction Multiple Data

SIFT Scale Invariant Feature Transform

SSD Single Shot Multibox Detector

SVM Support Vector Machine

WRN Wide Residual Network

Yolo You only look once

Chapter 1

Introduction

Micro-Air Vehicles (MAVs) such as a Quadrotor-MAV displayed in Figure 1.1 are an emerging technology that supports society in a wide range of consumer, industrial and safety applications. For example MAVs are used to deliver medicine [52], fight fires [31] or even find survivors in disaster situations [28].

Especially in emergency scenarios the fast and safe flight of MAVs is crucial to deliver help quickly and save human lives. However, due to the complexity of such missions as well as the difficulty to control an MAV in disaster scenarios, often multiple human operators are required in order to ensure safe operation [43]. With humans in the loop a constant connection between the MAV and the operators is required which not only uses energy and requires infrastructure but also significantly increases the reaction time. Enabling MAVs to fly more autonomously could allow human operators to control more MAVs and thus to improve the support in emergency situations.

A major challenge on the way to the full autonomous flight of MAVs is the accurate estimation of the MAV's state within its environment. The system is highly dynamic so position and orientation can change rapidly. At the same time noise introduced by motor vibrations makes the position estimation with only on-board Inertial Measurement Units (IMUs) too inaccurate [42]. Light Detection And Ranging (LIDAR)-sensors can capture long and wide range 3D information but the sensors are typically heavy and require a significant amount of energy. Infrared (IR) sensors can cover distance information but are often limited in their Field of View (FoV) as well as in their range. External infrastructure like Global Positioning System (GPS) and optical tracking systems can provide accurate measurements but there is no guarantee that such systems are present in real world applications. Cameras on the other hand are cheap, lightweight and can measure long range distance information. This makes them a suitable choice as a sensor for on-board state estimation on light

Figure 1.1: An example of a Quadrotor-MAV-Platform that is used in this thesis.

MAVs [10].

However, the signal delivered by the camera is high dimensional and can not directly be interpreted as position or orientation measurements. Computer Vision algorithms are required to interpret the image and extract relevant information. This can be done by designing an algorithm manually or learning the image processing from annotated examples. In particular Deep Learning based methods aim to combine whole Computer Vision pipelines into one mapping that transforms the raw input image into a task dependent output. Experiments have shown how Deep Learning based methods outperform traditional Machine Learning approaches and manually crafted algorithms [47]. This made them the predominant choice for almost any vision task.

The hereby used Convolutional Neural Networks (CNNs) are designed in a hierarchical way, using multiple layers that are evaluated sequentially. An example architecture is displayed in Figure 1.2. The network transforms an image of size 224x224 from its input (left) to a task dependent output (right). In this case a classification network predicting 1000 class probabilities is displayed. Each layer applies a non-linear transformation for which the parameters are learned during training. By stacking more layers on top of each other (deepening) and increasing the number of nodes D per layer (widening), highly non-linear functions can be modelled.

Experiments have shown the superior performance of particularly deep/wide models [18, 20, 54, 64]. However, this model flexibility assumed to be the reason for their superior performance also leads to immense requirements in computational resources. For example a state-of-the-art Computer Vision model [20] contains 60.2 million parameters and one inference requires 11.3 billion floating point operations [58].

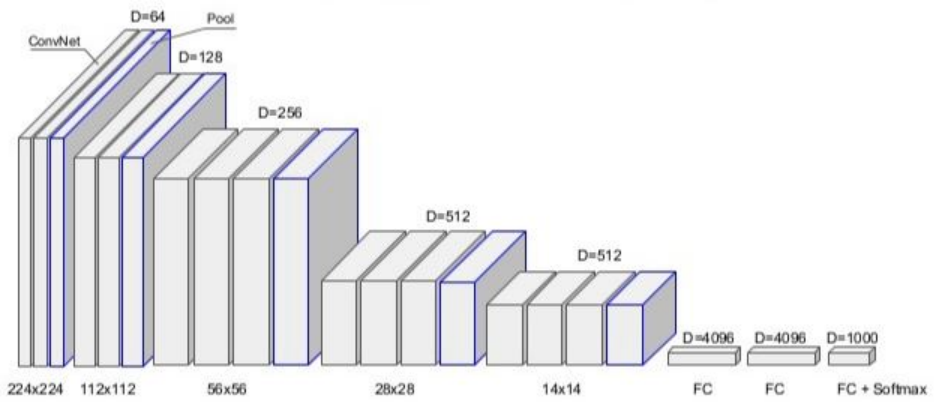


Figure 1.2: *VGGNet* [**?**] Example Architecture of a CNN.

Robotic platforms like MAVs have limited resources in terms of processing power and battery life. Hence, the use of CNNs on such devices is still an open challenge. Research has addressed to reduce the number of computations in Deep Learning models on multiple levels[14, 24, 35, 51, 64, 65]. However, the investigation of relatively shallow models with less than ten layers received only little attention by the research community.

This work investigates the deployment of a Deep Learning based Computer Vision pipeline on a MAV. The method is applied in the challenging scenario of Autonomous Drone Racing at the International Conference of Intelligent Robots (IROS) 2018. Within the race court several metal gates are placed and need to be passed one after another. Detecting the gates allows to estimate the MAV's relative position and to calculate the flying trajectory. An overview of the race court and the racing gates at the IROS 2016 Autonomous Drone Race can be seen in Figure 1.3.

The thesis builds on previous work by Li et al. [38] which uses a manually crafted image processing method to detect the racing gates. Although fast to execute the method is very sensitive to illumination changes.

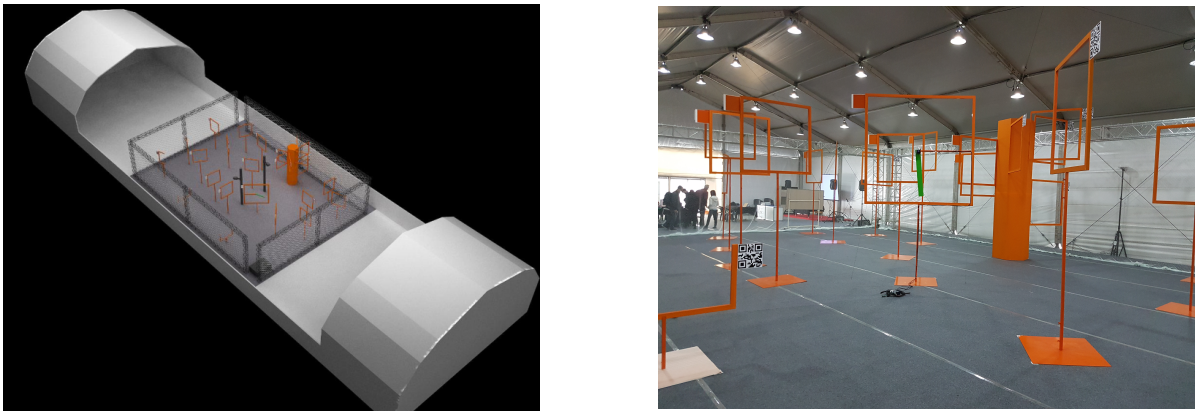


Figure 1.3: Example Images of the IROS 2016 Autonomous Drone Race

Moreover, the algorithm fails when the objects are too far away or the frame is very thin. In order to develop a more robust method, this thesis investigates a learning based approach to the detection of racing gates.

Object Detection is one of the most intensively studied topics in Computer Vision. However, the objects investigated are usually solid and contain complex shapes. For example a pedestrian consist of body parts and a face. A box that surrounds the object mostly contains parts with distinctive shape an/or texture. A Computer Vision model can use these features for detection. The racing gates in contrast are of different nature. As can be seen in Figure 1.3 a box that surrounds the object would largely contain background. Hence, this part can not be used as a hint whether an object is present. Instead it can contain other objects even other gates that might distract a detector. Additionally, the object parts themselves are of very thin structure and can be hardly visible. Thus, a detector needs to make use of fine-grain structures, while ignoring the majority of the image. This introduces a particular vision task that even humans have a hard time at solving [1] and that affects the training and design of a Computer Vision pipeline that aims to detect these kind of objects.

This thesis defines a class of objects as **Empty Wire Frame Objects (EWFO)** studies methods for their detection. The definition is given as follows:

Definition - Empty Wireframe Objects

1. **Empty.** The object parts are sparse. The bounding box around the object is largely occupied by background.
2. **Wireframe** The object does not consist of complex but only basic geometric shapes like corners, lines and edges. The object parts can be spread over large parts of the image.

The detection of EWFO is studied in the examples of the drone race gates. These can be seen can be seen in Chapter 1. Thereby the orange square on the top of the pole is considered to be the object of interest. To the best of the authors knowledge EWFO have not been particularly addressed in Computer Vision. In [12] and [36] the authors also detect racing gates, however the used objects contain more structure than the ones investigated in this thesis. Jung et al. present a framework to detect similar objects in [29] and [30] but do not study the particular effects of the object shape. This work particularly addresses the implications of the object shape in using a Deep Learning based detection system for EWFO.

A drawback of Deep Learning based vision systems is their need for vast amounts of annotated examples, which is not always available. Racing gates for example are not an object that appears often in everyday life and therefore not many example images exist. To this end no publicly available dataset can be used to train

[1] The unconvinced reader can try to count the number of gates visible in the right image of Figure 1.3

Figure 1.4: Example Image of the Empty Wire Frame Object investigated in this thesis.

a Computer Vision system for EWFO. Since a large part of the object consists of background, it is particularly crucial that the training set covers a large variety of backgrounds. Otherwise, it is likely that a model uses the background for prediction and only works in a particular domain (Overfitting).

Figure 1.5: Example of the Cyberzoo dataset. On the left an image while the MAV is hovering, on the right an image during a turn manoevre.

In Chapter 1 example images of the target domain of this work are displayed. The images are taken during a test flight at a test environment. The left image shows an example when the MAV is hovering and thus is in a very stable position. The object in this case is clearly visible as a single orange square. In contrast the right image shows a close up example during a turn manoeuvre. Here it can be seen how the used wide angle lens causes distortion and thus the lines appear as circular shape. Furthermore, large parts of the image including the horizontal bars of the object in the back appear blurred due to the circular velocity of the MAV. In addition, the light conditions of the environment significantly influence the object appearance.

While it is possible to remove lens and sensor effects in post-processing, this can lead to information loss and requires on-board resources. Instead it is computationally more efficient to perform the detection on the raw image data. However, sensor effects have been shown to significantly influence the performance of neural networks [2, 9]. Furthermore, they can lead to varying object appearance on different MAVs. This further complicates the collection of annotated examples.

Another option is the artificial generation of data. By synthetically generating samples with corresponding labels, the theoretical amount of training data is infinite. Moreover, the generation allows to incorporate domain specific properties such as motion blur or image distortion. Hence, data generation is particularly useful for the detection of MAVs on EWFOs where a large variety of backgrounds is required while samples

are difficult to obtain. Finally, as MAV are brittle vehicles and mistakes in development can lead to damage on hardware, engineers and researchers often use simulators to evaluate their systems before transferring them to the real work. Thus the basic infrastructure required to generate data is often already available.

Yet introduces the generation of data its own challenges. First and foremost because the generation process in itself is based on model assumptions. If these do not sufficiently capture the real world, a model trained in such an environment might be heavily biased and perform poorly in the real world. Secondly, because the generation of visual data is computationally intense. Despite advances in Computer Graphics can virtual environments not yet fully capture the real world. Hence, this work investigates the use of data generation in order to detect EWFOs on MAVs.

Without an accurate detection of the racing gate, the MAV is not able to determine its current position and thus to calculate its flying trajectory. On the other hand, with an algorithm that requires less computational resources a lighter MAV can be built. This allows faster and more aggressive trajectories as well as longer battery life. Moreover, the vision system is part of a greater state estimation and control system which also includes further sensor measurements. Depending on the remaining part of the system, faster and less accurate detections can be more useful than slow but accurate detections. Hence, the trade-off between accuracy and inference speed is of particular interest for this application and is addressed in this work.

1.1 Research Question

This section summarizes the research question addressed in this thesis. Furthermore it describes how the question is split in multiple subquestions that are addressed in the individual chapters.

How can we learn a CNNs to detect EWFO on MAVs using synthetic data?

RQ1 How can data be generated to train an object detector for EWFO detection on a MAVs?

RQ2 How can EWFOs be detected using a CNN on a MAVs?

1.2 Outline

The thesis is structured as follows. Chapter 2 gives an introduction to Object Detection and Synthesizing Data. Also, the target system of this work is illustrated. Chapter 3 introduces the methodology including the tool used to generate data, the object detection network as well as the test sets. Chapter 4 describes the experiments conducted in order to answer the research questions. Chapter 5 discusses the overall results, formulates a conclusion and describes possible future work.

Chapter 2

Background

This chapter describes background knowledge required to understand the remaining parts of the thesis. It introduces the target system for this work as well as datasets and metrics used for evaluation. Furthermore, it discusses related work in Object Detection and Data Generation.

2.1 Detecting Objects

On a high level Object Detection can be described by two individual goals: the description of what kind of object is seen (Classification), as well as where it is seen (Localization). Hence, an Object Detection pipeline transforms the raw image to a set of one or more areas and corresponding class labels. Images are high dimensional signals that can contain redundant and task irrelevant information. Performing detection in this space is difficult, also because the performance of machine learning models decreases when the feature space becomes too large (curse of dimensionality), Computer Vision pipelines usually apply a feature extraction stage, before the actual prediction is done. An overview is displayed in Figure 2.1.

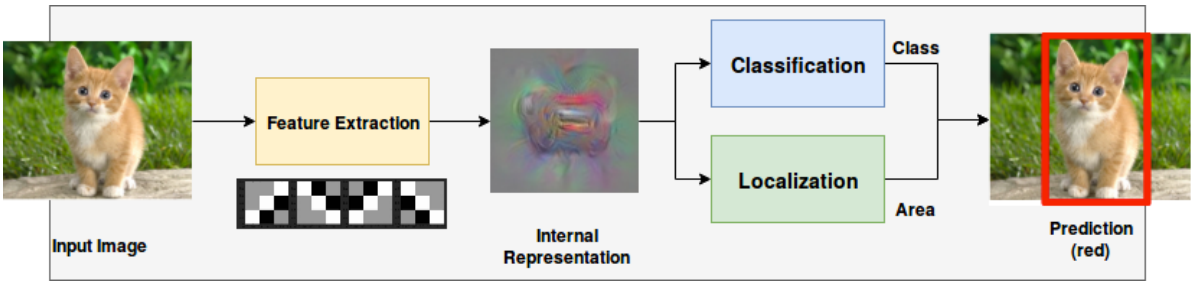


Figure 2.1: Object Detection Pipeline. An initial step extracts task relevant features of the input signal and derives an internal representation. Consecutively, classification determines what kind of object is present in the image, while localization determines where the object is located. The output consists of area(s) with class annotation.

1. The feature extraction stage extracts task relevant information from the image and infers an internal, more abstract representation of lower dimension.
2. The classification/localization stage produces the final output based on this representation.

An efficient feature extraction stage is thereby crucial for the success of an Object Detection pipeline. If the inferred representation is clearly separable, a simple classification stage can distinguish an object from the background. In contrast, even a flexible classifier cannot separate a highly overlapping feature space.

2.1.1 Baseline Algorithm: *SnakeGate*

The above pipeline can be illustrated in the example of the baseline algorithm of this work: *SnakeGate*. A high level description is given in the following. The interested reader can find more details in [38].

Initially, the image gets filtered by a higher and lower color threshold. This way ideally the largest part of pixels in the image is removed. This can be seen as the feature extraction stage.

In the resulting binary image the follow up stage searches for object parts. *SnakeGate* detects square objects and hence looks for vertical and horizontal bars that are respectively perpendicular to each other. Once this combination is found in a particular area of the image, it is counted as a valid detection. In the final step the corners of the bounding box are refined to get more accurate coordinates.

This simple method shows how crucial the feature extraction stage is. With a suitable color threshold, object parts can easily be found and the detection is accurate. However, this requires an evenly distributed colour across the object. As soon as back light or shadows fall on the object, a clear separation is difficult. Therefore, *SnakeGate* is sensitive to light changes.

Another issue of *SnakeGate* is that the object is defined as 4 bars in quadrangle shape. If one of the bars is not clearly visible the algorithm can not detect detect the object. Also, it does not exploit object context such as the object pole.

2.1.2 Related Work

The problems of *SnakeGate* are very typical for Object Detection. Since the beginning researchers investigated different methods of feature selection and the definition of object models. While initial work manually defined such features, later work replaced many of these stages with learning based methods. Today, whole state-of-the-art pipelines can be trained directly on raw images. This section investigates available literature and discusses the major milestones relevant for this work.

Traditional Methods

The early attempts to Object Detection define objects in terms of basic volumetric shapes such as cubes and cylinders. During inference these features are extracted and compared to a database. However, in practice even recognizing these basic shapes proves to be difficult [3].

Later approaches focus more on appearance based features such as wavelets [44] which also applied in [61] for human face detection. Thereby the image is processed by a cascade of classifiers using a sliding window in multiple scales. The processing of an image patch is stopped when a patch is classified as background. The features can be computed with simple operations and thus the detector can be executed extremely fast. Yet, the used Haar-wavelets cannot efficiently encode complex textures making the approach less suitable for many real world objects [3].

In contrast, Histogram of Oriented Gradients (HOG) [8] and Scale Invariant Feature Transform (SIFT) [40] use the image gradient to cover shape information. In a sliding window local histograms based on the gradient orientation are calculated. Dalal and Triggs [8] use the feature for pedestrian detection.

A general challenge in Computer Vision is the combination of local image features such as corners and edges to a more global detection of an object. Especially, when parts of the object can be occluded or deformed and thus undergo variations in appearance. In order to cope with these issues Felzenszwalb et al. [13] model pedestrians in individual parts and combine them in their proposed Deformable Part Model (DPM).

Traditional methods have in common that the individual steps of the detection pipeline are optimized separately. Furthermore, often the feature extractors and object models are designed manually. Hence, designing such a detector can result in cumbersome application dependent work. EWFO have sparse features and cameras on MAV can have a strong influence on the object appearance. Modelling this appearances manually is difficult. Also, the methods seem to have reached a limit in performance in the years of 2005-2012 where almost no improvement in performance was achieved.

CNNs-based Feature Extraction

A breakthrough in detection performance came with CNNs which emerged from Deep Learning research and subsequently became a popular feature extractor. CNNs can be seen as small neural networks that are applied locally on image patches in sliding window fashion. The outputs of the initial local operations (first layer) are further processed by higher layers until the desired output size is reached. The model parameters (weights) are trained using a loss function and the back-propagation algorithm.

The modular structure of CNNs allows to create highly non-linear models that can represent any function. However, this flexibility also introduces the challenge of choosing a suitable architecture. On a fundamental level design parameters can be summarized in depth, width and kernel size.

Section 2.1.2 displays these parameters and introduces additional terminology necessary for the remaining parts of this chapter. The *kernel size* **k** determines the spatial size of a kernel and therefore how big the patch is, the convolution is applied on. A layer usually contains multiple filters that are applied on its input. The amount of filters is also referred to as *width w*. The filters are applied in sliding window fashion which introduces the step size (*strides* **s**) as an additional parameter. The output of each convolution is concatenated and processed by the next layer. The amount of layers is also referred to as *depth*. In the image also the *receptive field* of a filter is visualized. This describes the image patch that is related to a certain feature response. The filter of the first layer (green) has a receptive field corresponding to its kernel size. The filter of the second layer (blue) combines the responses of the filters of the first layer at multiple spatial locations an thus has an increased receptive field.

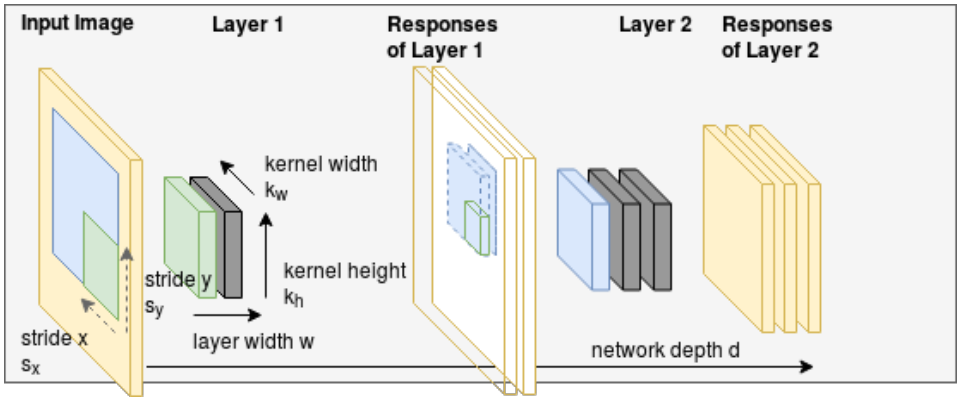


Figure 2.2: Notation in CNNs. The input image is convolved with a layer. One layer contains multiple kernels that are convolved in sliding window fashion with certain step size. The responses of the convolutions are collected and processed by the next layer. The receptive field is determined by which parts of the input image are part of the kernel response at layer x. The total amount of layers is also referred to as depth.

Among these parameters depth is considered one of the preliminary parameters to improve performance [18]. Simonyan and Zisserman [53] achieve first places in the 2014 ImageNet Classification challenge using a network that only contained filters of size 3-by-3 but up to 19 layers. Szegedy et al. [54] achieve similar performance using a network with 22 layers. The proposed network included an *Inception*-module, an architectural element that allows deeper networks at a constant computational budget.

Residual Connections. An issue that prevented training even deeper networks is the *vanishing gradient problem*. As the gradient distributes across nodes its magnitude gets smaller with increasing amount of nodes. Hence, the training becomes slow and the risk of converging in a local minima increases. This was addressed by He et al. [20] who propose the use of residual connections. Instead of propagating the gradient from the last to the first layer these connections allow the gradient to flow directly into all layers. This circumvents the vanishing gradient problem. The use of residual connections allowed to train a network 101 layers and improved on state of the art at that time.

Wide-Residual Networks. However, later work by Zagoruyko and Komodakis [64] shows how residual networks do not behave like a single deep model but more like an ensemble of more shallow networks. Moreover, the study shows that similar performance can be achieved by particularly wide networks and residual connections. Being of similar performance the proposed Wide Residual Networks (WRNs) are computationally more efficient to execute.

While wide residual networks can achieve similar performance to deep residual networks with reduced inference time the computational requirements are still large. This work addresses the detection of EWFO with very limited resources. Hence, a network in which the vanishing gradient problem would appear is likely to be already too computationally expensive to be applied on a MAV.

Fully Convolution Networks. Instead the work focuses on much smaller networks that are fast to execute. Execution time is also the motivation for Fully Convolutional Networks (FCNs). Instead of using a fully connected layer in the last stage, these networks only apply local operations. This saves many computations in the last layer and enables the application of models on various input sizes.

Dilated Convolutions. However, FCN in combination with a small amount of layers introduce a limited receptive field. If the network is too shallow the last layer cannot take into account the whole input image. A way to increase the receptive field without increasing the number of computations is the use of sparse kernels, also called Atrous/Dilated convolutions.

Depthwise Separable Convolutions. Another line of research to reduce the number of computations in CNNs address the convolution operation. *MobileNet* [24] and *QuickNet* [14] make extensive use of Depthwise Separable Convolutions (DSCs). DSCs replace the original 3D-convolution by several 2D-convolutions followed by a pointwise convolution. This reduces the total number of operations from $N = k_w \cdot k_h \cdot w_n \cdot w_{n+1}$ to $N = (k_w \cdot k_h + w_n) + w_n \cdot w_{n+1}$. *MobileNetV2* [51] further includes linear bottlenecks to reduce the total number of operations. These are convolutions with a 1by1 kernel and linear activation.

Channel Shuffling. In DSC pointwise convolutions are the bottleneck. The computation time can be reduced by performing groupwise convolutions. Thereby the channel is separated in subsets and the convolutions are applied on smaller volumes. This reduces computational cost by the number of groups but also stops the information flow between the groups. Therefore *ShuffleNet*[65] proposes a layer that shuffles the channels after the group convolution. This allows the information flow between groups while exploiting the reduced computational cost of group convolutions.

Despite a large amount of research conducted in finding suitable architectures there has not yet been a single way that always achieves a goal. It has been shown how models with a large amount of parameters combined with vast amounts of training data perform well on various vision tasks and objects. However, there is no guarantee that the found representation is also the most suitable/efficient one. The research resulted in a

collection of rules an best practices that need to be considered with the task at hand. This work investigates the design of a CNN for the detection of EWFO.

CNN-based Object Detection

After showing promising results for Classification, CNNs where also applied for Object Detection. CNNs-based Object Detectors can broadly be grouped in two categories. The categories are introduced and finally compared in relevance to this work.

Two-Stage Detectors. Girshick et al. [15] use Selective Search [59] to extract object candidates from an image and classify each region with a CNN. However, this requires to run the whole network at various scales and overlapping locations. Hence, the approach is computationally intense while many of the performed operations are redundant.

Follow up work aims to share computations for faster inference. Spatial Pyramid Pooling [19] allows to crop any region to a predefined size. Thus a CNN does not have to be run at overlapping regions anymore but the extracted features can be reused and need to be only computed once. This leads to a vast reduction in computations and leaves the region proposal algorithm as a computational bottleneck.

With the Region Proposal Network (RPN)[49] region proposals and feature extraction is combined in a single stage. The whole stage is framed as a regression problem by predicting object probability for a predefined set of bounding boxes so called anchor/prior or default boxes. Predefining the total amount of possible locations and bounding box dimensions would lead to an intractable amount of parameters and computations. Hence, the most common object appearances are defined and the network not only predicts an object probability for each box but also how to adapt location and dimensions to better fit the predicted object. Combining feature extraction and region proposals in a single network led to 213x speed up at that time.

Being currently the method with the best performance in terms of Mean Average Precision (mAP) two stage approaches would be a valid choice for the detection of EWFO. However, their two stage character makes the inference time relatively slow, which is not suitable for the application on a MAV. Also, this work investigates the detection of a single object. For this application the RPN can be used directly.

One-Stage Detectors. One stage detectors aim to further combine multiple stages of the Object Detection task into a single network. Therefore the whole pipeline is framed as a regression task. This leads to the question how to discretize the input image.

The first one stage detector You only look once (Yolo)[55] divides the input image in a fixed grid and predicts class probabilities for each grid cell. As this limits the amount of bounding box coordinates significantly the network also predicts global bounding box coordinates and a object probability for each box. As a last step a postprocessing algorithm fuses the output to the final prediction. Being a breakthrough as the first one stage detector the approach is limited to predict a single class for each grid cell. Furthermore, the approach of predicting bounding box coordinates globally proved hard for the network to learn and resulted in high localization errors.

Better results could be achieved by predicting offsets to predefined regions as in the concept of the aforementioned anchor boxes. Single Shot Multibox Detector (SSD) [39] extends the anchor box approach to perform the whole Object Detection task. Therefore the network not only predicts bounding box coordinate offsets and object probability but also a class probability for each anchor box.

An issue that arises with discretization of the object detection task is that objects can appear at different scales. For small objects it is required to have a high resolution of bounding box locations to have a sufficient

representation of the input image. For example many small objects can appear next to each other. A too coarse resolution could not capture the fact that there are multiple objects present. Furthermore, it is required to take into account fine grain features to predict such an object. In contrast, for large objects small changes in location do not make a difference. In contrary, a too fine resolution can cause the detector focus on noise and thus lead to bad predictions or unstable training. Therefore, SSD introduces the prediction of objects at multiple output grids at different layers of the network. Thus, the output grid can be defined depending on the object size and the network can extract features based on the object scale. Follow up work in [48, 55] also included the concept of anchor boxes and prediction layers at multiple scales, making SSD and Yolo converge to a very similar solution.

In [25] one and two stage detectors are compared in terms of accuracy as well as resource requirements. While two stage detectors with very deep networks tend to achieve the best performance, one stage detectors are generally faster. The experiments also show how the inference time of two stage detectors can be reduced without loosing too much accuracy when the amount of bounding box proposals is limited. However, single shot detectors are still the fastest method with the lowest memory profile. Furthermore, for the single class case the region proposal stage of a two stage detector can be used directly. Hence, in this work we investigate the the detection of a EWFO with a one stage detector. The *TinyYoloV3* architecture is 9-layer network with a suitable size to be applied on a MAV. This work uses this approach as the baseline model.

The anchor box concept allows to incorporate prior knowledge about possible objects but also introduces additional hyperparameters that need to be tuned for an application. Therefore the alternative approach of *CornerNet*[34] avoids the anchor box concept by defining objects as paired key points. Each bounding box is defined as two corners. The network predicts a heatmap with potential corner locations as well as which corners belong together. This would be an interesting approach for the detection of EWFO. However, as the publication was quite recent the approach could not be taken into account.

Attention Models

A research direction which is fundamentally different to the approaches seen so far are models based on visual attention. Instead of processing the whole input image at once, the aim is to process only relevant image patches. Typically a model processes an image crop and decides which location to evaluate next until enough information is gathered for the final prediction. Examples for the approach can be found in [1, 4, 26].

Attention models are promising as their computational complexity can be controlled independently from the number of pixels in the input image. However, to this end successes have mostly been demonstrated on digit recognition like the MNIST dataset. Scaling the approach to real world problems proves to be difficult. Furthermore, the features of an EWFOs are sparse, while most part of the image does not provide hints where to look for an object. Therefore, we assume the approach less applicable for the detection of EWFOs.

2.2 On the inference time of CNNs

For the deployment on a MAV resource consumption is an important parameter. Next to the architecture elements such as DSC and *Channel Shuffling* other ways are investigated to reduce the inference time of a CNN.

Weight Quantization Weights are quantized such that the calculations can be performed as integer operations. As Central Processing UniTs (CPUs) typically do not have hardware supported floating point units, integer operations can be executed much faster. By mimicking the quantization effects during training, the network learns to deal with these kind of artefacts.

Knowledge Distillation Knowledge Distillation [22] is way to create a smaller model based on a trained bigger model or an ensemble of models. Thereby the smaller student-network is trained to mimic the output of the final and intermediate activations of the teacher-network. *FitNet* extends the approach to create a thinner/deeper network based on a trained network thereby reducing parameters without loosing performance. In [37] the approach is applied to the task of Object Detection. In [62] knowledge distillation is combined with weight quantization in order to obtain a faster and more efficient model.

Both methods would be interesting to explore for the deployment of CNN on the target system. However, they require an efficient low level implementation. A full implementation of a CNNs on the target system would be beyond the scope of this work. Therefore the *Darknethttps://pjreddie.com/darknet/yolo/* framework is used. It supports the hardware available on the target system. Unfortunately, it does not contain the possibility to apply weight quantization or knowledge distillation. Therefore we do not further consider this option and address the inference time on a more architectural basis.

2.3 Generating Data

Related methods vary from changing low level properties of the image over using CAD models in combination with real background up to rendering full 3D-environments. Often various combinations of synthesized and real data are applied.

Low-Level Image Augmentation

A common part of current Computer Vision pipelines is to augment a given data set by transforming low level properties of the image. By artificially increasing variations in the input signal, a model that is more invariant to the augmented properties shall be obtained.

Krizhevsky et al. [33] use Principal Component Analysis (PCA) to incorporate colour variations. Howard [23] shows how several image transformations can improve the performance of a CNN-based Classification model. The proposed pipeline includes variations in the crop of the input image as well as variations in brightness, color and contrast. In CNN-based Object Detection Szegedy et al. [55] uses random scaling and translation of the input image, as well as random variations in saturation and exposure. Liu et al. [39] additionally crop and flip each image with a certain probability.

Since most methods use image augmentation and Krizhevsky et al. [33] mentions it to be the particularly reason for superior performance at ILSVRC2012 competition it can be assumed to be beneficial for Computer Vision models. Unfortunately, none of the publications measures the improvements gained by the different operations.

While the aforementioned approaches add artificial variation to the input data, Carlson et al.[5] augment the image based on a physical camera model. The proposed pipeline is applied for Object Detection and incorporates models for sensor and lens effects like chromatic aberration, blur, exposure and noise. While being of minor effect for the augmentation of real data (0.1% - 1.62% mAP70) the reported results show an improvement when training on fully synthesized datasets. Here the reported gains vary between 1.26 and 6.5 % mAP70.

Low-level image augmentation is a comparatively cheap method to increase the variance in a dataset. However, it cannot create totally new samples or view points. Furthermore, it cannot change the scene in which an object is placed. Therefore it needs a sufficiently large base dataset that is augmented. This work addresses the case when no real training data is available. Hence, low-level image augmentation is incorporated in the training process but can not be the only method applied.

Augmenting Existing Images with CAD - Models

In order to create new view points Computer Aided Design (CAD)-models can be used. These models describe 3D-shape of an object and can be placed on existing images to augment or increase a dataset.

Peng et al.[46] study the use of CAD-models in the context of CNN-based Object Detection. The authors particularly address how image cues like texture, colour and background affects the detection performance. The experiments show how the used CNNs are relatively insensitive towards context but use shape as primary, texture and colour as secondary most important features. This enables competitive performance even when the object of interest is placed only on uniformly covered backgrounds. However, the study only covers solid objects such as birds, bicycles and airplanes. EWFO are substantially different and we hypothesize that other image cues must be relevant.

Madaan et al.[41] study the segmentation of wires based on synthetic training. As wires similarly to EWFO only consist of thin edges, the application is quite close to this work. However, the experiments focus on a single domain, namely sky images and thus the variations in background are comparatively small. We hypothesize that EWFO are particularly sensitive to such variations and address the application in multiple domains.

Hinterstoisser et al. [21] propose to use a base network that has been trained on real images and to continue training on images with CAD-models. During training the base network is frozen and only the last layers are updated. The method does not use real data but requires a suitable base network. As most available feature extractors are of a size that is computationally prohibitive for MAV the method is not really applicable for this work.

The use of CAD-models in combination with real backgrounds allows to generate totally new view points for the object of interest. Furthermore, the image background consists of real data and thus the synthetic textures only concern the rendered object. However, the geometric properties like perspective as well as the physical properties like object placement are violated and therefore create an artificial scene. Despite this fact, literature shows that such images can benefit model performance in various cases. Yet, most of the approaches still use real data and/or focus on solid objects with rich textures and complex shape. We hypothesize that since EWFO do not provide these kind of structures the results do not apply in the same way. Hence, we incorporate the method to generate data and investigate how it can be applied for the detection of EWFO.

Fully Synthesizing Environments

A more realistic placement of objects can be achieved when fully synthesizing environments. The object of interest can be placed according to physical laws, shadows fall correctly and geometric properties of an image are followed. However, if the graphical models do not fully capture the details of real world objects, the generated data might look too artificial.

Johnson-Roberson et al. [27] use a powerful graphical engine and a highly detailed environment to train an Object Detection model entirely in simulation. The results show an improvement towards data annotated by humans especially when using vast amounts of simulated data.

In order to create realistic environments intense manual work is required for the design. In contrast [50, 56, 57] use a relatively simple environment but a high degree of randomization to address the reality gap. The aim is to learn an abstract representation by strongly varying textures, light conditions and object locations. Tobin et al. introduced this technique as Domain Randomization (DR). The drawback of the approach is that a too high degree of randomization may omit pattern in the target domain that could otherwise be exploited by the model.

This work addresses the generation of data for the detection of EWFO on MAVs in GPS denied scenarios. Such scenarios cover a wide range of possible environmental conditions and the images taken from MAV cameras are peculiar. Hence the creation of a full environment is investigated in this work.

2.3.1 Transfer Learning

The field of transfer learning particularly addresses domain shifts in the modelling process. Hence, a common application is the learning from synthetic data.

A common approach in CNN-based models is the incorporation of a domain classifier in the model. By augmenting the data with domain labels, the classifier learns to distinguish the two domains. Subsequently a gradient reverse layer is applied and thus the weights are updated in such a way that a domain agnostic representation is learned. Examples of the approach can be found in [6, 63].

While the aforementioned approaches require labelled samples from the target domain, Peng et al. [45] propose to include task-irrelevant samples and a source classifier. As a result no samples of the target domain are required.

While transfer learning provides the theoretical framework as well as methods to deal with domain shifts, it does not allow to generate data. Furthermore, it often requires samples of the target domain. This work addresses the case when no real data is used for training. The field is interesting to be incorporated in the data generation pipeline investigated in this thesis but it can not be used as a start off point. Hence, the use of transfer learning in the modelling process is denoted as future work.

2.3.2 Generative Adversarial Networks

Generative Adversial Networks (GANs) [17] are a learning technique to generate new samples. The method uses two models a generator and a discriminator. While the generator generates samples the discriminator aims to distinguish the generated samples from a real dataset. Thereby the training goal of the generator is to maximize the error of the discriminator. Both models are updated with back propagation. The method shows promising results for image transformation or image synthesises but also for audio signals [7]. Yet to the authors knowledge GANs have not yet been applied to generate samples for Object Detection. Furthermore, the discriminator needs a training set for initialization. As for this work only a small amount of training samples are available, we do not investigate this approach for generating samples to detect EWFOs.

2.4 The target system

A MAV consist of multiple components of Software that are responsible for higher and lower level tasks. Figure 2.3 illustrates these components in the example of the target platform of this thesis. On the lowest level drivers read out sensors such as the camera and an IMU or communicate with a ground station. A low level control loop is responsible for controlling the local state of the MAV such as altitude and attitude. A higher level control loop controls the global state of the MAV which is the position and the flying trajectory.

The high level control loop of this work is described in further detail. A first step detects the racing gate and yields the corner coordinates. These are used to estimate the relative position of the MAV towards the gate. In the second step the visual measurements are fused with measurements of other sensors. In this case IMU and a sonar deliver attitude and altitude data. This step yields a global position estimate of the MAV. Combined with prior knowledge about the race court, the desired attitude and altitude required to fly the trajectory is calculated. The results are send as set points to the low level controller.

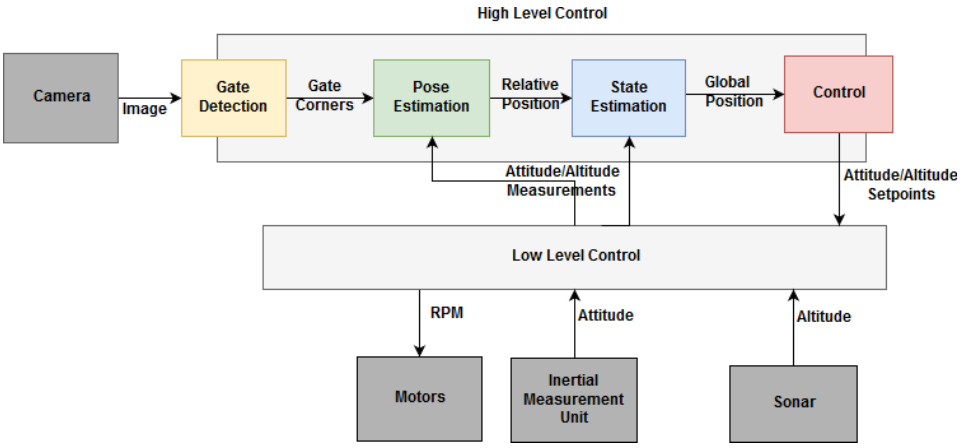


Figure 2.3: Control Loop

The hardware platform used to run the high level control loop is the *JeVois* smart camera. It contains a 1.3 MP camera with 65 degree field of view. The processing units are a quad core ARM Cortex A7 processor with 1.35 GHz and a dual core MALI-400 GPU with 233 Mhz. In order to extend the field of view a 120 degree wide angle lens is mounted. In Figure 2.4 the camera is shown.

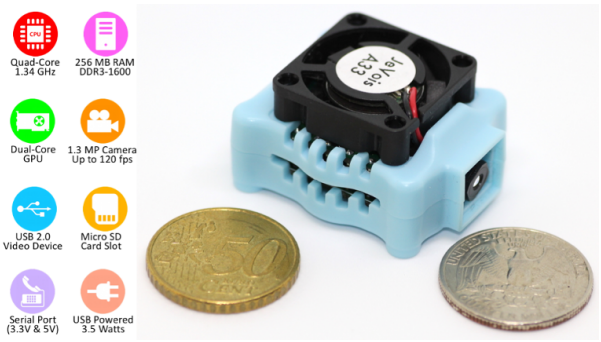


Figure 2.4: JeVois Camera

Chapter 3

Methodology

This chapter introduces notations as well as the mathematical models and software tools used in this thesis. It starts with describing the data generation pipeline that is used to generate synthetic data. Subsequently, the Object-Detection network is explained. Finally, two datasets used for evaluation are introduced.

3.1 Data Generation Pipeline

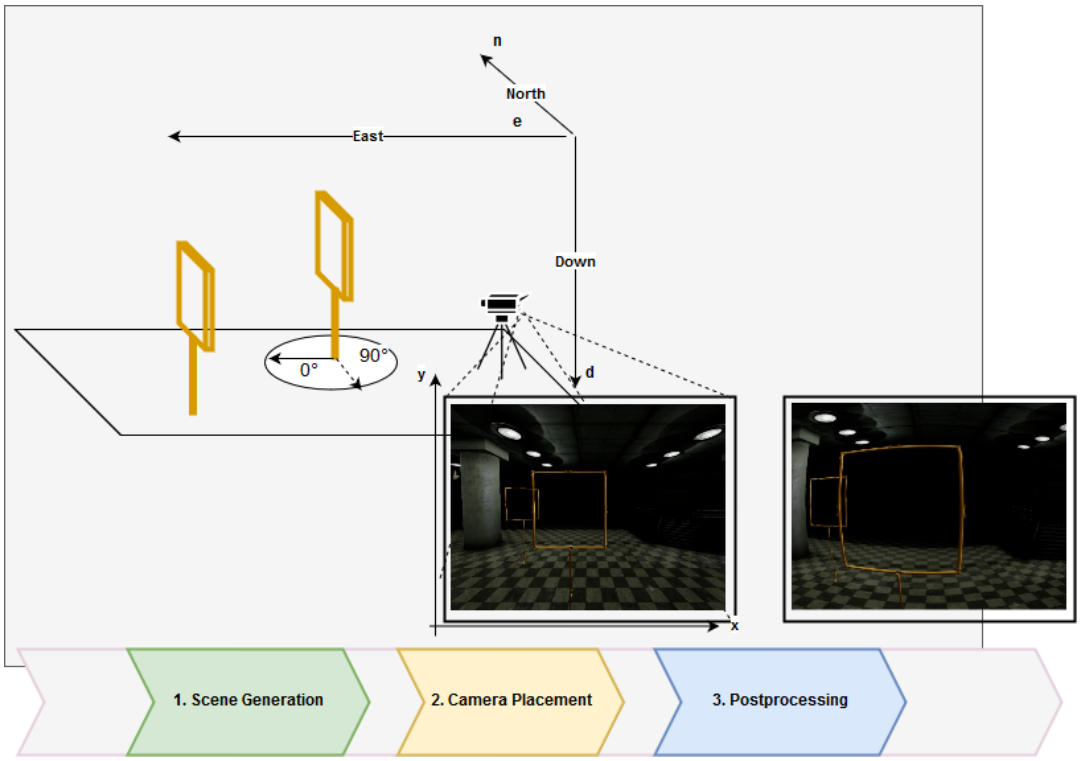


Figure 3.1: Notation and data generation process. In the first step a scene is created which defines the environment and background. The second step places the camera in the environment and creates a 2D view of the scene. The third step applies further image transformations. The coordinate system in 3D is North-East-Down (NED) originating at the initial position of the camera. In 2D, positions are defined in Cartesian coordinates originating at the bottom left. With a rotation of 0° the object appears as an empty square, with 90° as a straight line.

An overview of the data generation pipeline can be seen in Figure 3.1. In the first step a scene is created in which the objects of interest as well as the camera are placed. In 3D space position and orientation (pose) of each object are determined by its translation $\mathbf{t} = [x, y, z]$ and rotation $\mathbf{r} = \phi, \theta, \psi$. The used coordinate system is NED while the center of origin is placed at the initial position of the camera.

A view projection yields an image through the lens of the camera. The coordinates of each point in 3D space are projected on the 2D image plane. The position of each point in the image space is defined by $\mathbf{p} = [p_x, p_y]$. The origin is on the bottom left of the image.

These steps are implemented using *Epic*'s *UnrealEngine*[1] and its *AirSim*-Plug-In[2] by *Microsoft*. This allows the creation of environments using CAD models, as well as the automatic placement of the camera in C++. The tool is extended to store the location of a bounding box that surrounds an object in the environment. Hence, for any position of the camera the bounding box for all objects in view can be stored. This allows the automatic generation of ground truth labels while the camera is placed.

The *UnrealEngine* is a graphical engine that contains a profound amount of photorealistic image effects. However, their use with the data generation tool would require a deep understanding of graphical programming as well as the *UnrealEditor* and a substantial amount of work. Hence, the final post processing step is implemented using the image processing library *OpenCV* in Python. This also allows to study the incorporation of particular sensor effects or image augmentation while training the network. All source code is made publicly available at `https://github.com/phildue/datagen.git`.

3.1.1 Environments

Images can appear substantially different depending on the particular environment in which they are taken. The light conditions in an indoor scene with artificial light are substantially different from the ones outdoor in sunlight. The environment also influences the appearance of an object and therefore its detection.

In order to train and evaluate the detection of EWFOs in different conditions, three environments are created. A black environment serves as base to replace the background with existing images. Then, three indoor base environments are produced that fully simulate illumination and background. An overview can be seen in Figure 3.2. Within the environment light conditions, background textures, object locations can be changed manually. The environments are described in the following:

1. *Dark:* The environment is a room without windows, only containing artificial light sources.
2. *Daylight:* The environment is a room with windows along all walls that allow daylight to illuminate the room. The windows can lead to strong variations in the contrast between different parts of the object.
3. *IROS:* The environment resembles the room of the IROS Autonomous Drone Race 2018. The light sources stem from a window front at one side of the room, as well as artificial light sources at the ceiling. Depending on the view point, the object might appear against bright or dark background.

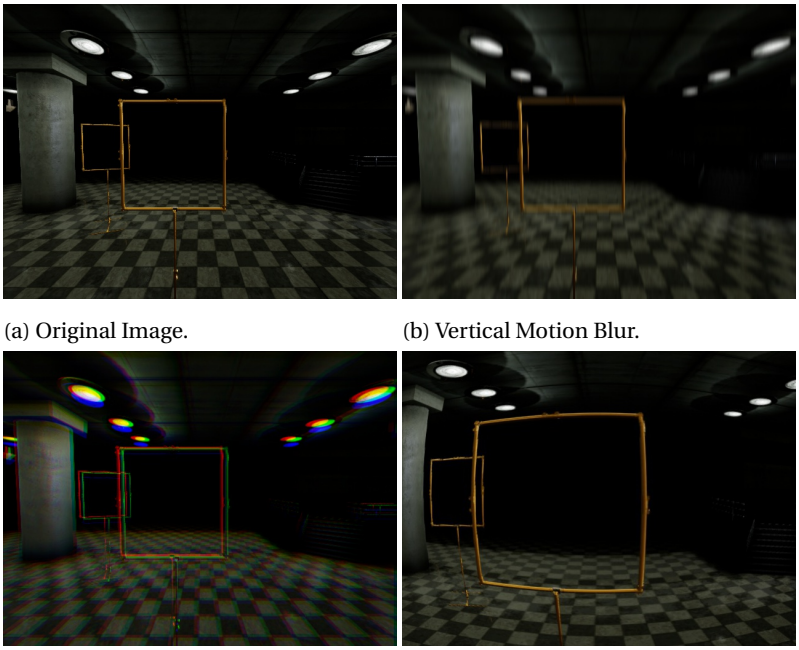
Figure 3.2: The environments from left to right *Dark*, *Daylight*, *IROS2018*

[1] `https://www.unrealengine.com/en-US/what-is-unreal-engine-4`
[2] `https://github.com/Microsoft/AirSim`

3.1.2 Post Processing

Another parameter that influences the image and object appearance are camera and lens conditions. An object detector should be able to cope with these effects. Hence, this work studies the influence of some effects when modeled in the training set.

Lens distortion and motion blur are chosen due to their presence in the real world data set. Furthermore, Chromatic Aberration is studied because it led to vast improvements in [5]. Variations in Hue, Saturation, Value (HSV) space are selected as it is a common image augmentation technique in current Object Detection pipelines. A visual overview can be seen in Figure 3.3. This section describes the mathematical models behind the applied effects.



(a) Original Image. (b) Vertical Motion Blur.

(c) Chromatic Aberration. (d) Lens Distortion.

Figure 3.3: Overview of the visual effects appearing due to camera and lens conditions

Lens Distortion Lens distortion is a form of optical aberration which causes light to not fall in a single point but a region of space. For MAVs commonly used wide-angle lenses, this leads to barrel distortion and thus to straight lines appearing as curves in the image.

The effect is applied using the model for wide-angle lenses from [60]. It models the removal of lens distortion as combination of radial and non-radial part, that is approximated with a second order Taylor expansion:

$$\begin{pmatrix} p_x^u \\ p_y^u \end{pmatrix} = f(p_x, p_y) = \begin{pmatrix} p_x(1+\kappa_1 p_x^2 + \kappa_1(1+\lambda_x)p_y^2 + \kappa_2(p_x^2+p_y^2)^2) \\ p_y(1+\kappa_1 p_x^2 + \kappa_1(1+\lambda_y)p_y^2 + \kappa_2(p_x^2+p_y^2)^2) \end{pmatrix} \tag{3.1}$$

Where:

- p_x and p_y are the original coordinates.
- p_x^u and p_y^u are the undistorted coordinates.
- κ_1 κ_2 control the radial distortion
- λ_x and λ_y control the tangential distortion

Applying the lens distortion to an image is done by inverting Equation (3.1). Since there is no closed form solution the Newton-approximation is used.

Chromatic Aberration. Chromatic Aberration is caused when different wavelengths of light do not end up in the same locations of the visual sensor. This leads to a shift in the colour channels of the image.

In [5] including chromatic aberration significantly improves the performance of models that are trained on fully synthesized data. Hence, we hypothesize this will also help for our work.

Similarly to [5], chromatic aberration is applied by scaling the locations of the green channel by s^G, as well as applying translations on all channels by t^R, t^G, t^B. Mathematically this is:

$$\begin{pmatrix} p_x^c \\ p_y^c \end{pmatrix} = f(p_x^c, p_y^c) = \begin{pmatrix} s^c p_x^c + t_x^c \\ s^c p_y^c + t_y^c \end{pmatrix} \tag{3.2}$$

Where c is a colour channel.

Blur Fast movement and sensor noise can lead to blurry images. This is particularly present in the domain of MAV/Autonomous Drone Racing. Hence, we hypothesize that including this effect will improve the detection in the real world.

The effect is modelled using a Gaussian-filter. The image is convolved with a 2D-kernel build from:

$$k(x, y) = \frac{1}{2\sigma_x \sigma_y \pi} e^{-\frac{1}{2}\left(\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2}\right)} \tag{3.3}$$

Where σ_x and σ_y are the variance in direction x and y, used to model directional (motion) blur and μ_x, μ_y are the coordinates of the kernel center.

Variations in HSV The 3D-models and textures used in the simulator are limited and creating a large variation in environments or objects requires manual effort. An alternative method to increase the variation in colour and illumination is directly varying the colour. The HSV colour space groups colours that are visually close to each other. Hence, for augmentation the image can be varied in this space. Therefore we draw a ΔH, ΔS and ΔV uniform distributions and it to the respective pixel intensities in HSV space.

3.2 You only look once V3 (YoloV3) - Object Detector

This work investigates the detection of EWFOs with *YoloV3* a typical one stage detector with anchor boxes. The fundamental concept of one-stage detectors with anchor boxes is explained in Section 2.1. This section describes the detailed implementation of *YoloV3* and its training goal. It further introduces the baseline network architectures of this work.

3.2.1 Concept

On a high level basis *YoloV3* maps the input image to a predefined set of anchor boxes. For each box the network predicts an object probability $\hat{o}$ that classifies the class as object (1) or background (0). The original version of *YoloV3* further distinguishes between object classes, however this work considers the single class case. There we remove this output node from the prediction.

The anchor boxes have a predefined width p_w, height p_h as well as a center location c_x,c_y and are arranged in G 2D-Grids with individual spatial resolution S_n ($S^h \times S^w$), with S^h, S^w respectively the height and width resolution of the grid. This allows to define different output resolutions depending on the object size. Smaller objects need a more fine grain resolution as more of them can appear close to each other. The same fine grain resolution can be too high for larger objects and thus lead to confusion of the detector during training. The bounding box dimensions p_w and p_h can be chosen manually or determined by k-means clustering on label width and height of the training set. In our experiments a k-means clustering is performed before each training such that the p_w and p_h are tuned to the training set.

The predefined anchors only cover a subset of possible areas that can contain an object. Hence, *YoloV3* also predicts how to adapt the anchor box to better fit the predicted object. These are the bounding box center b_x, b_y as well as its width b_w and height b_h.

In total this leads to 5 predicted parameters for each bounding box $(\hat{o}, b_x, b_y, b_w, b_h)$ and to a mapping from the input image of $I_H \times I_W \times I_D$ to $5B$ nodes that encode $B = \sum_{i=0}^{G} S_i^h \times S_i^w \times a_i$ boxes, where a_i is the number of anchor boxes in grid i, and I_H, I_W, I_D respectively the image height, width and depth. In a last step boxes that contain the same class prediction and have high overlap are filtered such that only the boxes with the highest confidence remain.

3.2.2 Architectures

The above-mentioned mapping is implemented with a CNN. Thereby the dimension of the network output corresponds to $S_*^w \times S_*^h \times 5a_i$, where $*$ depends on the output grid i.

With *YoloV3* the *TinyYoloV3* network was released, it is a 9 layer CNNs and thus a suitable baseline for this work. However, due to the small amount of layers, the receptive field of the network is only 223 pixels. Hence, the final layer does not see the all pixels of the image. This is crucial for EWFOs where the features are spread over large parts of th image and the centre is empty. Furthermore, the spatial resolution of 13x13 is still coarse for objects that almost cover the whole image. Therefore the architecture is extended by an additional pooling and output layer. This results in a final layer with $S^w = S^h = 7$ that sees the whole image and is responsible to detect large scale objects.

The network architecture is referred to as *SmallYoloV3* and displayed in Figure 3.4 and explained in detail in the following. The input image with a resolution of 416x416x3 is processed by 5 layers that stepwise decrease the spatial resolution (max pooling) while increasing the width, leading to a intermediate volume of 26x26x512. This part can be seen as a common part that extracts features for objects at all scales. The architecture is a typical example of current CNNs. In the early layers the receptive field of the filters is small. Hence, the patterns that can be represented are not very complex and only a small amount of filters is used. As the network gets deeper more complex patterns can be present and more weights are required to encode these features. Hence, the width is increased. Research has shown that fixing kernels to a size of 3x3 and stacking them in deep layers is particularly efficient[**?**]. This can also be seen in the *SmallYoloV3* architecture.

Convolving the wide volume of deeper layers such as the 26x26x512 output of layer 5 with a 3x3 kernel requires many computations. Therefore a common technique is to first compress the volume by applying a 1x1 kernel intermediately. Such *bottleneck* layers can be seen in layer 6-1 and 7-2.

From layer 5 the network splits to two branches responsible for smaller and larger objects. The lower branch extracts features for larger objects leading to a final grid of 13x13. The higher branch extracts features for smaller objects leading to a grid of 26x26.

The created network serves as a baseline since it is relatively shallow and suits to the computational requirement of an MAV. In order to study the influence of network complexity on the detection of EWFOs a second

much deeper architecture is created. Therefore the baseline architecture in Figure 1.2 is used. In order to use the network for object detection the last three layers are replaced by a similar structure as the last three layers in *SmallYoloV3*.

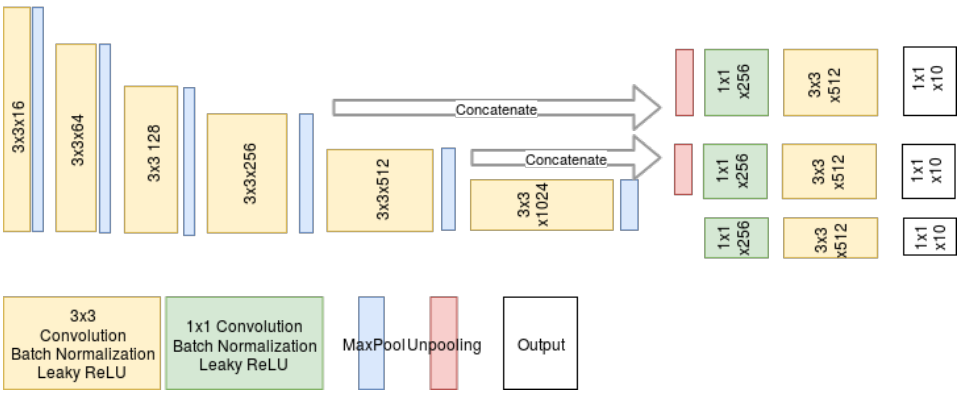


Figure 3.4: *SmallYoloV3*. In the common part the spatial resolution decreases each layer down to 26x26, while the width increases from 16 to 512. From layer 5 three branches focus on objects corresponding to different scales.

3.2.3 Training Goal

In order to train a CNN to predict the desired properties a ground truth has to be defined for each output node. Subsequently the loss is formulated as a derivable function and the CNN can be trained with back-propagation.

Thereby it is desirable that a network output of zero corresponds to no filter activation and henceforth to keep all predicted bounding boxes in the default shape. Therefore, *YoloV3* encodes the ground true coordinates as follows:

$$b_x = \sigma(\hat{x}_{i,j,k}) + g^x_{i,j} \quad b_y = \sigma(\hat{y}_{i,j,k}) + g^y_{i,j} \quad b_w = e^{\hat{w}_{i,j,k}} \cdot p^w_{i,j,k} \quad b_h = e^{\hat{h}_{i,j,k}} \cdot p^h_{i,j,k} \tag{3.4}$$

where $\hat{x}, \hat{y}, \hat{w}_{i,j,k}$ and $\hat{h}_{i,j,k}$ correspond to output nodes of anchor box at grid i, cell j, anchor k; $g^x_{i,j,k}$, $g^y_{i,j}$ is the top left coordinate of the respective grid cell; σ is the sigmoid-function; p_w and p_h are the pre set parameters of the anchor box.

The question remains to which grid cell and anchor box a label is assigned to. During training the loss is only propagated through the nodes that correspond to the "responsible" anchor box. This responsibility is determined by the center location of the ground truth box and its width and height. A label is assigned to the grid in which its center falls into and within that cell to the anchor that has the highest Intersection over Union (IoU). This can be a very strict assignment when an object has a high overlap with multiple anchor boxes. While other one stage detectors allow multiple anchors to be responsible to predict a box, *YoloV3* only assigns one anchor, but ignores the predictions of an output node that has an IoU of more than 0.5.

With true and predicted labels the training goal can be formulated. The loss needs to capture the localization and the classification goals. In a typical ground truth image only a small subset of anchors is assigned responsible to predict an object. All the other anchors see only background. Hence, there is a class imbalance between the "object" class and the "background" class. Treating both losses equally would lead the model to simply assign "background" for all anchors. The weight terms λ_{obj} and λ_{noobj} compensate for this class imbalance. Furthermore, λ_{loc} trades-off the localization goal and the classification goal. The abstract training loss is summarized in:

$$\mathscr{L} = \lambda_{loc}\mathscr{L}_{loc} + \lambda_{obj}\mathscr{L}_{obj} + \lambda_{noobj}\mathscr{L}_{noobj} + \lambda_{class}\mathscr{L}_{class} \tag{3.5}$$

where $\mathscr{L}_{loc}$ is the loss for bounding box dimensions, $\mathscr{L}_{obj}$ the loss where a object is present, $\mathscr{L}_{noobj}$ the loss where there is no object. The weights are kept to the default value of $\lambda_{loc} = 5$,$\lambda_{obj} = 5$ and $\lambda_{noobj} = 0.5$.

The object loss quantifies a binary classification loss. Hence, it is the difference between a predicted probability $\hat{o}$ and an actual class label c, where $o \in \{0,1\}$ and $\hat{o} \in (0,1)$. In order to learn such a goal it is desirable that the weights of the network get updated significantly when the difference between truth and prediction are high. However, when prediction and truth are already close to each other, the updates to the weights should be smaller otherwise the training might miss the optimal solution. A loss function that contains the desired properties and that is used by *YoloV3* is the logarithmic loss which can be formulated as follows:

$$\mathscr{L}_{log} = -(o_{ij}\log(\hat{o}_{ijk}) + (1 - o_{ij})\log(1 - \hat{o}_{ijk})) \tag{3.6}$$

where $\hat{o}_{ij}$ is an output node with sigmoid activation assigned to anchor box i,j,k and o_{ij} the ground truth label assigned to that box. The logarithmic loss is calculated for each output grid G_i, for each grid cell S_j and each anchor box B_k. However, only the loss of the responsible anchor boxes are summed in the total loss calculation:

$$\mathscr{L}_{obj} = \sum_{i=0}^{G}\sum_{j=0}^{S_i^2}\sum_{k=0}^{B_i} \mathbb{1}_{ijk}^{obj}(-(c_{ijk}\log(\hat{c}_{ijk}) + (1 - c_{ijk})\log(1 - \hat{c}_{ijk}))) \tag{3.7}$$

Thereby the binary variable $\mathbb{1}_{ijk}^{obj}$ is 1 if an the anchor box at i,j,k is assigned responsible to predict the respective object. $\mathscr{L}_{obj}$ is defined vice versa but controlled by the $\mathbb{1}_{ijk}^{noobj}$ binary variable.

For the localization loss, similar properties are desirable, although the loss should be invariant to direction. A loss that contains these properties is the squared distance between each bounding box parameter. The localization loss is summarized in:

$$\mathscr{L}_{loc} = \sum_{i=0}^{G}\sum_{j=0}^{S_i^2}\sum_{k=0}^{B_i} \mathbb{1}_{ijk}^{obj}[(x_{ijk} - \hat{x}_{ijk})^2 + (y_{ijk} - \hat{y}_{ijk})^2 + (w_{ijk} - \hat{w}_{ijk})^2 + (h_{ijk} - \hat{h}_{ijk})^2] \tag{3.8}$$

where x_{ijk},y_{ijk} are the ground truth center coordinates of anchor box i,j,k and w_{ijk},h_{ijk} the corresponding width and height. $\hat{x}_{ijk}$,$\hat{y}_{ijk}$, $\hat{w}_{ijk}$,$\hat{h}_{ijk}$ are the predicted bounding box coordinates.

3.2.4 Training Procedure

With a quantified loss the training can be formalized as minimization problem:

$$\min_{w} \mathscr{L}(f(I,w),y) \tag{3.9}$$

where I is an input image, w are the network weights and y is the true label.

The optimization can be performed using gradient descent. However, with large amounts of data and high amount of parameters this algorithm has a slow update rate. Hence, *Adam*[32], a version of stochastic gradient descent is used. Thereby the gradient is estimated based on a subsample of the training set. While this leads to faster convergence, estimation errors can cause updates in the wrong direction. *Adam* compensates for this by not only taking the mean gradient into account but also its first and second statistical moment.



Figure 3.5: Examples of the three test domains. From left to right: *Basement*, *Cyberzoo* and *Hallway*

The three moments are weighted with α,β_1 and β_2, where α is also referred to as learning rate. This work uses the parameters recommended in the initial public publication of *Adam*[32]:

$$\alpha = 0.001 \quad \beta_1 = 0.9 \quad \beta_2 = 0.999$$

An optimization process such as the training of a CNNs depends on the initialization of its parameters. Glorot and Bengio [16] propose to initialize weights based on a uniform distribution centered around zero, where the borders depend on the input size of the layer. This scheme is a standard in Deep Learning frameworks and also used in this work.

YoloV3 and its training is implemented using *keras*[3] with *tensorflow*[4] backend.

3.3 Datasets

For the objects investigated in this work no public dataset is available. Hence, two datasets are created in order to compare the performance of different detectors/architectures.

3.3.1 Real-World Dataset

A dataset has been recorded to serve as a benchmark for the developed methods. The dataset consists of 300 images recorded with the *JeVois* camera during flight and while remaining on the ground. The samples stem from three different rooms with varying light conditions. The rooms are referred to as *Basement*, *Cyberzoo* and *Hallway*. Example images for each room can be seen in Figure 3.5.

All environments are indoor scenes which are a typical examples for GPS-denied areas, where vision based state estimation is required. The scenes contain two gates that are arranged in varying order. Hence up to two objects are visible and can overlap which means the gate farther away can be seen through the closer gate. Each of the rooms has different environmental conditions:

1. *Basement* is a bright environment illuminated by artificial light sources. The corridor in which the objects of interest are placed are narrow while also objects and persons are visible on the samples. The dataset contains 163 samples with 312 objects in total.
2. *Cyberzoo* is taken from a test environment for MAV flights. It is surrounded by black curtains such that an even illumination and dark background is created. Only in a small subset of images distractors like other objects or persons are visible. In total 88 samples stem from this room while 71 objects are present.

[3] `https://keras.io/`
[4] `https://www.tensorflow.org/`

3. *Hallway* is a bright environment illuminated by a combination of artificial light sources as well as daylight that shines through the windows. The samples are taken with the windows as background. This leads to a very bright background such that the thin structure of the objects are hardly visible. The dataset contains 49 samples with a total of 86 objects.

3.3.2 Sets created in simulation

The data that can be created with the data generation tool is limited by the graphical engine, the used textures and the created environments. It is possible that this reality gap limits the performance of the detection on real data. Hence, only measuring the performance on the real world dataset would limit the insights gained from the experiments. That is why synthetic test sets are created. This also allows to evaluate the performance of the detection network on race courts with more than two gates and at view angles that are not present in the real world dataset. The different test sets are described in the following. Furthermore, the training sets used in various experiments are introduced here. An overview is given in Table 3.1 a detailed explanation is given in the experiments section.

Table 3.1: An overview of the datasets created in simulation

Name	Environment	View Angles	Object Class	Nb. Images
Testing				
Simulated MAV Race	IROS	Generated by Simulated Flight	Gate	550
Test Simple Basement Gate	Basement	Frontal View	Gate	200
Test Simple Basement Sign	Basement	Frontal View	Sign	200
Test Simple Basement Cat	Basement	Frontal View	Cat	200
Test Simple IROS Gate	IROS	Frontal View	Gate	200
Test Simple IROS Sign	IROS	Frontal View	Sign	200
Test Simple IROS Cat	IROS	Frontal View	Cat	200
Test Simple Basement Gate	Basement	Frontal View	Gate	200
Test Simple Basement Sign	Basement	Frontal View	Sign	200
Test Simple Basement Cat	Basement	Frontal View	Cat	200
Test Simple IROS Gate	IROS	Frontal View	Gate	200
Test Simple IROS Sign	IROS	Frontal View	Sign	200
Test Simple IROS Cat	IROS	Frontal View	Cat	200
Training				
Train Simple Basement Gate	Basement	Frontal View	Gate	200
Train Simple Basement Sign	Basement	Frontal View	Sign	200
Train Simple Basement Cat	Basement	Frontal View	Cat	200
Train Simple IROS Gate	IROS	Frontal View	Gate	200
Train Simple IROS Sign	IROS	Frontal View	Sign	200
Train Simple IROS Cat	IROS	Frontal View	Cat	200
VOC Background	Backgrounds from Pascal VOC Dataset	Frontal View	Gate	20000
Simulated Frontal	Basement, Daylight	Frontal View	Gate	20000
Race Courts	IROS, Dark, Daylight	Generated by Simulated Flight	Gate	20000
Random Placement	IROS, Dark, Daylight	Generated by Random Placement	Gate	20000

Chapter 4

Experiments

This chapter introduces the line of experiments taken out in this work and their intermediate conclusions. Initially, experiments about the detection of EWFOs are conducted on basic simulated environments. In further steps the acquired insights are applied when transferring to the more challenging environment of autonomous drone races. Finally, a detector for EWFOs is deployed on an example MAV.

4.1 Experimental Setup

This section gives an overview of the hardware used for training the detector as well as details on this training process. Furthermore a description of particular plots used for evaluation is given.

As training a neural network is a computationally intense process, the common practice is to use Graphical Processing Units (GPUs) for faster execution. In this work all trainings are carried out on a Nvidia Pascal GTX 1080 Ti GPU with 3584 cores and 11GB RAM.

The training is stopped when the performance on unseen examples does not further improve, that is when the validation error converges. Specifically, the training is stopped when the error does not decrease for more than $1e^{-08}$ in 3 epochs. 0.1 % of the training samples are used as validation set for this step.

The detector is tested by applying the network on a given test set and calculating the metrics described in **??**. While this gives a good estimation about the overall performance of a network, it can be required to investigate the results in greater details, to know how the detector deals with certain view points for example. In order to perform this evaluation, predictions and true labels are assigned to bins based on certain conditions e.g. the bounding box size. Subsequently the performance is evaluated for each bin individually.

In special cases it can happen that a bin border falls right between a true and a predicted label. For example a true label has a bounding box of size 10, a predicted label has size 12 and the border is at size 11. Even if the detection is correct, this separation would lead to counting a missing detection as well as a false positive in each of the bins. Hence, the performance for the individual bins is typically a bit lower than when calculating a metric for the whole dataset.

The algorithm training as well as the network initialization are random processes. Hence, the network weights after training, as well as its performance are not deterministic. This condition has to be taken into account when interpreting the results. Ideally, each training is performed repeatedly and mean and standard deviation are used for evaluation. However, the training of CNNs takes a considerable amount of time which is why not all experiments can be taken out with many repetitions. Instead, trainings are performed at least two times

and only further repeated if the two results have a high deviation. In plots an error bar displays the standard deviation between the different repetitions.

4.2 Evaluation Metrics

The detection performance is evaluated in terms of precision and recall. These metrics are defined as:

Precision

$$p = \frac{\text{nb. of true positives}}{\text{nb. of true positives} + \text{nb. of false positives}}$$

Recall

$$r = \frac{\text{nb. of true positives}}{\text{nb. of true positives} + \text{nb. of false negatives}}$$

Where true positives are objects that are detected, false positives are detections although there is no object and false negatives are objects which have not been detected.

Hence, recall expresses how many of all objects are detected and therefore how complete the result is. Precision measures how many of the predicted objects are actually correct detections.

A correct detection is determined based on its overlap with a ground truth box. This is measured by the relation of IoU. In experiments we determine 0.6 as sufficient overlap for a detection as it allows an accurate estimation of the relative pose on the MAV.

YoloV3 predicts an object probability for each bounding box. By accepting detections with lower probability the number of potential true positives increases. Therefore the recall gets higher. However, it also increases the amount of potential false positives and thus lowers precision. In order to evaluate this trade-off, precision and recall can be compared at different confidence thresholds.

A metric that combines precision and recall in a single metric is average precision ap introduced by the Pascal Visual Object Challenge (Pascal VOC)[11]. It takes the average interpolated precision p_{interp} across evenly spaced recall levels:

$$ap = \frac{1}{11} \sum_{r \in 0, 0.1, \ldots, 1.0} p_{interp}(r)$$

The interpolation reduces the amount of zig-zags in precision-recall plots and simplifies the comparison between outputs of different detectors. It is defined as:

$$p_{\text{interp}}(r) = \max_{r' \geq r} p(r')$$

We denote precision, recall and average precision at a certain IoU threshold such as 60% as p_{60},r_{60} and ap_{60}.

4.3 Threats to Validity

The experiments are conducted within a certain environment which can have influence on the validity of the results. This section summarizes these threats to validity.

All experiments are conducted with the frameworks described in Chapter 3. Potential software faults in these frameworks can have an influence on the obtained results.

The experiments are conducted in the example of a single object and a recorded test set. An other object that falls in the category of EWFOs might produce different results.

The test set contain a limited range of backgrounds and environments. In other environments other results can be obtained.

For these experiments a standard metric for Object Detection is chosen: ap_{60}. An alternative way of assessing detection performance could lead to different results.

4.4 Empty Objects

In this first sections basis experiments in simulation are conducted. The detection of EWFOs is compared to other objects.

CNNs combine simple local features to more complex patterns layer by layer. Thereby pooling removes task irrelevant information and reduces the spatial dimension. In the deeper layers features of larger areas in the image are combined and encoded in an increasing amount of filters. In the final layer each location in the volume encodes the patterns that are present in the respective field of the preceding filters and an object prediction is performed.

The power of deep CNNs arises from their capability to learn very complex patterns. However, these are not present in EWFOs. Instead most of the object area consists of background and should be ignored by the detector. We hypothesize that this emptiness makes the detection more difficult than the detection of other objects as the detector can not exploit complex patterns. Instead any object can be present within the frame and thus distract the detector.

The combination of emptiness and simple features can have further implications on the training of an Object Detector for EWFOs. If not sufficient variations in background is provided in the training set, a detector is likely to overfit to the background of the training set. This condition can be amplified for more complex architectures with more parameters.

To summarize our hypotheses are:

1. Compared to a simple filled object, the detection of EWFOs is harder, as the object does not provide complex patterns and a detector can be confused by patterns that are present in the empty part.
2. Compared to a complex filled object, the detection of EWFOs can not be improved by using a deeper network.
3. Compared to other objects, a detector trained to detect EWFOs is more likely to overfit to background. If the environment in the training set is different to the test set, the performance drop for EWFOs is higher than for other objects.

In order to evaluate these hypotheses the detection of an EWFO is compared to a comparable object where the empty part is filled with a certain pattern. The created objects *Cats* and *Sign* are visualized in Figure 4.1. Thereby a simple object is chosen such as the stop sign which is clearly distinguishable from the background (hypothesis 1). This is compared to a more complex object such as the cat image (hypothesis 2).

The created objects allow to study how a detector that is trained on a filled object performs. Also, it allows to study how the detector for EWFOs reacts when another pattern is present in the empty part during testing.

Figure 4.1: Examples of the three objects that are compared. The EWFOs object (*Gate*) left is compared to a simple solid object (*Sign*) in the center and a complex solid object on the right (*Cats*).

For each object a dataset with 20 000 samples is created within the *Dark* environment. As this experiment focuses on the influence of the empty part of the object, the view points are limited to frontally facing the object in various distances. These are the datasets *Train Dark Gate*, *Train Dark Cats*, *Train Dark IROS* as shown in Section 3.3.2.

On these training sets the two architectures illustrated in Section 3.2 *SmallYoloV3* and *VGGYoloV3* are trained. As 20 000 samples is a comparatively small amount of samples for a network such as the *VGGYoloV3*, the network is initialized with the weights of the *VGG-19* pretrained on ImageNet.

For each object a test set of 200 samples is created within the *Dark*-Environment, as well as the *IROS*-Environment (testing of hypothesis 3). Hence, in total there are 6 test sets with 200 samples each. Similar to the training set the view points are limited to frontally facing the object at various distances. These are the datasets *Test Dark Gate*, *Test Dark Cats*, *Test Dark IROS* as shown in Section 3.3.2.

4.4.1 Results

Trained/Tested	Gate [ap_{60}]	Sign [ap_{60}]	Cats [ap_{60}]
Gate	0.55 ± 0.04	0.01 ± 0.00	0.37 ± 0.06
Sign	0.02 ± 0.01	0.74 ± 0.06	0.05 ± 0.04
Cats	0.06 ± 0.03	0.03 ± 0.00	0.78 ± 0.01
Gate Deep	0.54 ± 0.00	0.00 ± 0.00	0.53 ± 0.00
Sign Deep	0.00 ± 0.00	0.70 ± 0.00	0.08 ± 0.00
Cats Deep	0.01 ± 0.00	0.13 ± 0.00	0.76 ± 0.00

Table 4.1: Performance of two architectures when the test environment is similar to the training environment (datasets *Test Dark Gate*, *Test Dark Sign*, *Test Dark Cats*). Each trained network (row) is evaluated on each test set (column). It can be seen how the detectors exploit the structure that is placed in the object. In contrary, the detector of EWFOs only gets confused when the structure inside the object is very different from the training set.

Table 4.1 shows the results in the *Dark*-environment. It can be seen how the best results are obtained for the *Cats*-object. Yet when the structure is removed the performance drops to 0.13% - 0.02 %. A similar observation can be made for the *Sign*-object. In both cases the performance does not improve when using a deeper network.

For detecting the *Gate*-object, the lowest performance of 0.55% is achieved. When the same detector is applied on an object where the empty part is filled, the performance drops. This happens particularly for the

Sign-structure. For the *Cat*-structure the performance drop is lower. In fact for the deep network there is not really a performance drop measurable.

Trained/Tested	Gate	Sign	Cats
Gate	-0.31 ± 0.10	0.01 ± 0.02	-0.14 ± 0.01
Sign	-0.02 ± 0.01	-0.30 ± 0.05	-0.03 ± 0.02
Cats	-0.00 ± 0.03	-0.02 ± 0.01	-0.74 ± 0.02
Gate Deep	-0.31 ± 0.00	0.01 ± 0.00	-0.26 ± 0.00
Sign Deep	-0.00 ± 0.00	-0.32 ± 0.00	-0.08 ± 0.00
Cats Deep	-0.01 ± 0.00	-0.13 ± 0.00	-0.73 ± 0.00

Table 4.2: Change in performance when the detectors are tested in another environment than their training environment (datasets *Test IROS Gate, Test IROS Sign, Test IROS Cats*). The most severe drop can be seen at the *Cats*-object. The drop for *Gate* is comparable to the *Sign*-object

Table 4.2 shows the change in performance when the trained detectors are evaluated in a different environment. All detectors are subject to a significant drop, however the strongest effect can be seen for the *Cats*-object. While the *Sign*-object can still be detected best, its relative performance drop is comparable to the *Gate*-object. The network size does not have a significant effect when changing the test environment.

4.4.2 Discussion

From the previous observations, it can be seen how the detector exploits the added structure in the filled objects, for which the performance is much better than for the *Gate* object. Also, when the detectors trained with added structure are applied on the empty object the performance drops. Thereby the network size is of minor effect. No performance boost is achieved even for the more complex *Cats* object.

When the test environment changes, the most complex object can almost not be detected anymore. It seems that the detector particularly overfitted to lightning conditions and background. This is surprising as the background in the *IROS*-environment is lighter and thus the object is better visible than in the *Dark*-environment.

The simple but solid object is subject to a smaller performance drop when the environment changes. In the new environment it can still be detected best. This is likely because the surface mainly consists of a white and red area which gives distinctive shape and colour.

The detector for the *Gate*-object is less subjective to changes in the empty part as expected. When applying the detector on objects with the *Cats* structure some performance can still be reached. The deep architecture does not suffer any performance drop in this case. This is likely because the *Cat* structure is of similar shape and colour as the background. When adding a very different structure such as the *Sign* object, the performance drops almost to zero.

The background dependency is also lower as expected. When moving to a new environment the performance drop is not higher than for the other objects. Potentially, all detector overfitted to the environmental conditions of the training room.

4.4.3 Conclusion

In this section we compared the EWFO investigated in this work to objects of similar shape that contain a structure inside the empty part. We hypothesized that a EWFO is harder to detect as it provides less features that the detector can use. This hypothesis can be confirmed as when adding structure inside the empty part the performance get significantly better.

Figure 4.2: Examples of samples with more background. On the left a sample augmented with an image from the Pascal VOC 2012 dataset. On the right a sample generated in the *Daylight* Environment. Although on the left the background contains realistic data the scene does not align with the objects. Also the shadows do not fall correctly. With the simulated environment the general scene looks more realistic although the background is synthetic.

Furthermore, we hypothesized that due to the empty part a detector for EWFOs is more dependent on the training environment than for other objects. However, this hypothesis could not be confirmed. The performance drop for other objects is at least equally high. Its possible that all networks overfitted to the environmental conditions in the training set. However, this hypothesis would need to be confirmed in further experiments.

Also, we hypothesized that in contrast to a more complex object the detection of EWFOs can not be improved by using a deeper network. While this could be confirmed for the EWFO, in the experiments there is neither an improvement for the other objects. Yet it can be seen how the deeper network is less confused when adding the *Cat* structure.

4.5 Providing Background

In the previous experiments it could be seen how the performance of a detector drops significantly when applying it in an environment that is different to the training environment. In this section it is investigated how to make the the detector less dependent on such domain shifts.

A simple method is to create more data by directly placing the object in front of backgrounds of different images (Figure 4.2 left). This way the detector can learn to be background invariant. However, with this placement the object is not aligned with its context anymore. The light conditions do not fit to the remaining image and also perspective properties are violated that otherwise could be exploited by the detector. Another method is to create new environments in simulation and change light conditions and background there (Figure 4.2 right). This requires more manual work but leads to geometrically aligned images. Yet, the images consist only of synthetic elements.

We hypothesize that the creation of samples with a simulator leads to better results than when simply replacing the background. Therefore a detector is trained with both methods and evaluated on the test sets created in the previous section.

With both methods a dataset with 20 000 samples is created. The view points are limited to frontal views.

For the dataset with random backgrounds, 2000 view points are created in a black environment. Subsequently the black image part is replaced with a randomly selected image from the Pascal VOC dataset [11].

The simulated dataset is created using *Daylight* and *Dark* environment which have different lightning conditions. Additionally, the backgrounds in both environments are varied such that a higher variance in back-

ground textures is achieved. Thereby the background texture that is present in the *IROS* environment is not used.

4.5.1 Results

Trained/Tested	gate [ap_{60}]	sign [ap_{60}]	cats [ap_{60}]
Single Background	0.26 ± 0.06	0.03 ± 0.02	0.36 ± 0.03
Simulated Background	0.75 ± 0.03	0.50 ± 0.21	0.69 ± 0.06
VOC Background	0.35 ± 0.07	0.47 ± 0.16	0.36 ± 0.17

Table 4.3: Performance of *SmallYoloV3* in the *IROS* environment when adding more variance in the background and in the random-background environment. It can be seen how including more backgrounds improves the results especially when the environment is fully simulated. Furthermore, the detector has learned to be more invariant structures that are present inside the image.

Table 4.3 shows the results. Selecting the backgrounds randomly only led to a minor improvement. In contrast, simulating more environments and backgrounds improved the performance by 50%.

For both methods the detectors became less subjective to patterns present in the empty part. The detector trained on simulated images achieves equal performance when there is a *Cat* structure present. Also, the performance drop for the *Sign* structure is less severe. Interestingly, the detector trained with random backgrounds even detects more gates when there is a *Sign* structure present in the empty part.

Another observation is the higher variance in the results. Especially, when trained with random backgrounds the standard deviation is quite high.

4.5.2 Discussion

Providing more samples from different environments was crucial for better performance. The results are even better than the ones achieved when the detector was trained and tested in the *Dark* environment (Table 4.1). By supplying more backgrounds the detector could learn an overall better representation. However, it seems similarly crucial to provide realistic environments. The detector trained on random backgrounds only achieved minor improvement.

Providing more variation in background also helped the detector to ignore the background. The performance drop when placing patterns inside the object is smaller.

4.5.3 Conclusion

In this section we investigated how to make the detector more invariant against changes in background and the environment. By increasing the training set with samples in front of different backgrounds this was achieved. This method even improved the results beyond its previous highest score. Thereby it seems important to provide a realistic alignment of object and scene. When simply pasting the object on random images only minor improvements could be achieved.

4.6 Transferring the detector to an MAV race

Until now the conducted experiments were limited to relatively simple environments. In a real world application such as an MAVs race, much more objects are in sight. These can not only appear frontally but also

in more difficult view angle. Furthermore, the objects can appear behind each other, such that in the empty part, another object is visible. This section studies whether the results obtained so far also apply in a more challenging environment. Therefore different architectures and training methods are evaluated on the data set *Simulated MAV Race* (Section 3.3.2).

Due to the challenges in this dataset we hypothesize that the detectors trained so far will perform poorly. In order to handle overlapping objects in difficult angles, such situations should be included in the training set. Yet, the amount of possible views/overlaps is large and manually constructing such examples is cumbersome especially considering the amount of samples required to train a CNNs. A simpler way is creating a scene with several objects and placing the camera randomly (within some margin) in order to cover a large variation of views on the scene. With this *Random Placement* the detector can learn a general object representation and detect unseen objects from different view points. However, CNNs cannot inherently handle object rotations and variations in scale. Including too many view angles in the training set could also confuse the detector. For example from a 90° angle the investigated object appears as a straight line and is hardly detectable even for a human. If too many view angles confuse the detector, the samples should be created by only using a limited set of view points.

The question remains how to choose those view points. In the application of this work the MAV follows a predefined trajectory which is based on the a priori known race court. It needs the detections to correct its positions but the rough object position is known in advance. Hence, the MAV can point the camera in such a way that the object is roughly faced frontally. Subsequently, it adapts its position to fly through the gate. This behaviour can be mimicked when creating the samples. Although such a *Simulated Flight* requires to create a race court and a corresponding trajectory, the obtained samples should better resemble what the detector can expect in the real world.

We investigate the creation of samples with the two methods *Random Placement* and *Simulated Flight*. For *Random Placement* 600 samples are created by choosing the following distributions based on the expected flight behaviour of an MAV:

$$x = \mathcal{U}(-30,30), \quad y = \mathcal{U}(-20,20), \quad z = \mathcal{N}(-4.5,0.5)), \quad \phi = \mathcal{U}(0,0.1\pi), \quad \theta = \mathcal{U}(0,0.1\pi), \quad \psi = \mathcal{U}(-\pi,\pi) \tag{4.1}$$

Where $\mathcal{U}(a,b)$ is a uniform distribution with borders a and b; $\mathcal{N}(\mu,\sigma)$ a Gaussian distribution with mean μ and variance σ.

For *Simulated Flight* the dynamic model of a quadcopter is used. Several race courts are created and the camera follows a predefined trajectory through these race courts.

Figure 4.3 shows the label distribution when created with the two methods. Thereby each pixel value corresponds to the number of labels that cover this particular pixel. It can be seen how, when following the race track, most of the objects are centred and distributed across the horizon, as camera focuses the next object frontally most of the time. In contrast,*Random Placement* leads to more evenly distributed object locations.

Figure 4.4 shows b_h and b_w of 600 samples created with the two methods. Although the actual view angle can not be inferred from these dimensions, the plot gives an idea on how the labels look like. When the object is faced frontally the aspect ratio is 1.1/1. It can be seen *Random Placement* covers a broader range of view angles but does not create many samples with large objects. The reason is that the field of view gets smaller as objects get closer. Hence, the camera ending up exactly in front of the object is relatively low. On the other hand *Simulated Flight* does not cover many view points where $b_h >> b_w$. In the created race courts, these view angles do not seem to be present.

We investigate to what extent the detector can generalize across view points. We hypothesize that a detector trained with *Simulated Flight* performs better in a simulated MAV race than a detector trained with *Random*

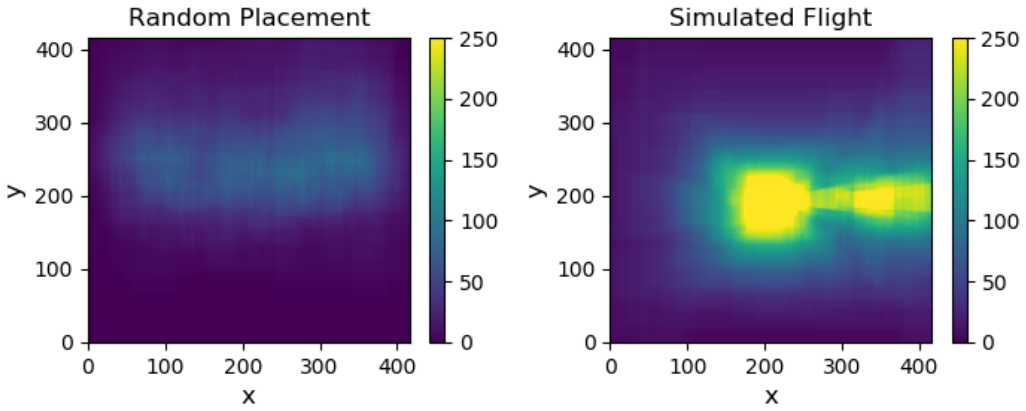


Figure 4.3: Object appearances in 2D when generating 600 samples with *Random Placement* and *Simulated Flight*. Each pixel value corresponds to the number of labels that cover this particular pixel. In the simulated flight objects appear mostly centred on the horizontal line.

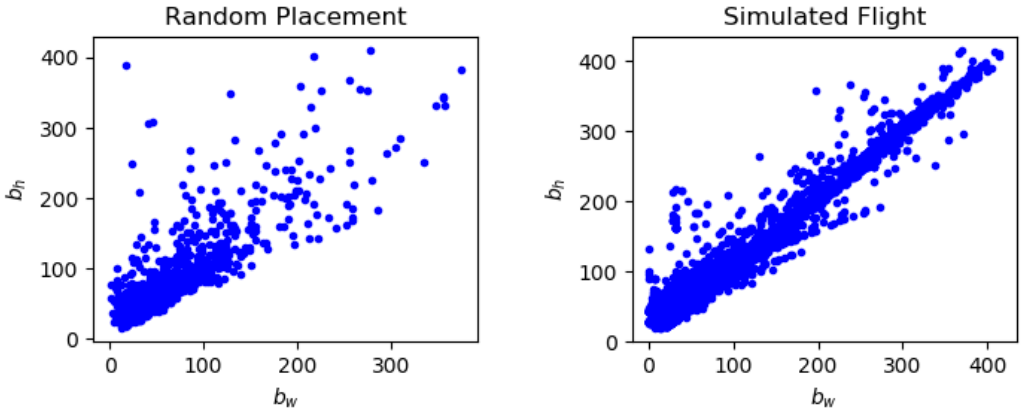


Figure 4.4: b_h and b_w of 600 samples created with *Random Placement* and *Simulated Flight*. When the object is faced frontally the aspect ratio is 1.1/1. It can be seen how with *Simulated Flight* much more close up samples are created. On the other hand *Random Placement* covers a broader range of view angles.

Placement.

Training sets with 20 000 samples each are created using *Random Placement* and *Simulated Flight*. The scenes are created with the environments introduced in Section 3.1.1. Additionally, the background textures are varied during data generation. For *Random Placement* the gates are placed throughout the room, subsequently the camera is placed according to Equation (4.1). This results in the dataset *RandomPlacement* described in Section 3.3.2. For *Simulated Flight* 3 race courts with corresponding trajectories are created. This results in the dataset *Racing Courts* described in Section 3.3.2. It should be noted that the race court present in the test set is not part in either of the training sets.

During data generation it can happen that the camera ends up at a view point where the object is only visible as a dot or a straight line. Including such samples in the training confused the detector in initial experiments. Therefore, we set a limit to aspect ratio and bounding box size and remove the corresponding labels during training. The aspect ratio is limited to a minimum of $\frac{1}{3}$ and a maximum of 3.0. For the object size a minimum of 1 % of the image size is chosen.

4.6.1 Results

The results are presented in Figure 4.5. Predicted and true labels are assigned to bins based on their covered area $A_O = b_w * b_h$. Subsequently ap_{60} is evaluated for each bin.

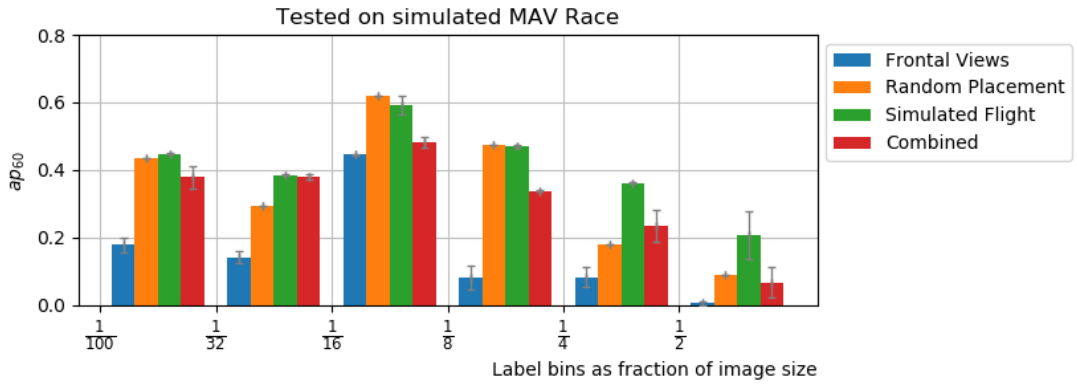


Figure 4.5: Results of different methods to include more samples in the training set. The results are clustered based on the size of the true/predicted bounding box. It can be seen how the network that contained only frontal views performs poorly when applied in the simulated MAV race track. The network trained with images obtained with *Simulated Flight* outperforms the network trained on samples obtained with *Random Placement* for larger object sizes.

Frontal Views is the network trained in the previous section. Its training set contained only frontal views and no overlap. It can be seen how it performs poorly when simulating a whole MAV race. *Random Placement* achieves competitive performance for smaller object sizes until 25% of the input image. However, the performance drops below 20% for larger objects. *Simulated Flight* achieves competitive performance on all bins. Only for objects of a size between 6.2% - 12.5% of the input image the random placement performs slightly better. In most bins combining both methods led to worse results than each of the methods individually.

Overall a significant drop in performance can be seen compared to the test sets of the previous sections. Only on the bin for objects of a size between 6.2% - 12.5% of the input image the networks achieve a comparable performance of 69 %. This bin seems to be the optimal distance for all networks. Especially for larger objects the performance decreases.

Figure 4.6 shows precision and recall at a confidence of 0.5 for the different bins and networks. Precision is the lighter coloured bar. It can be seen how *Frontal Views* achieves little recall in most bins. For larger

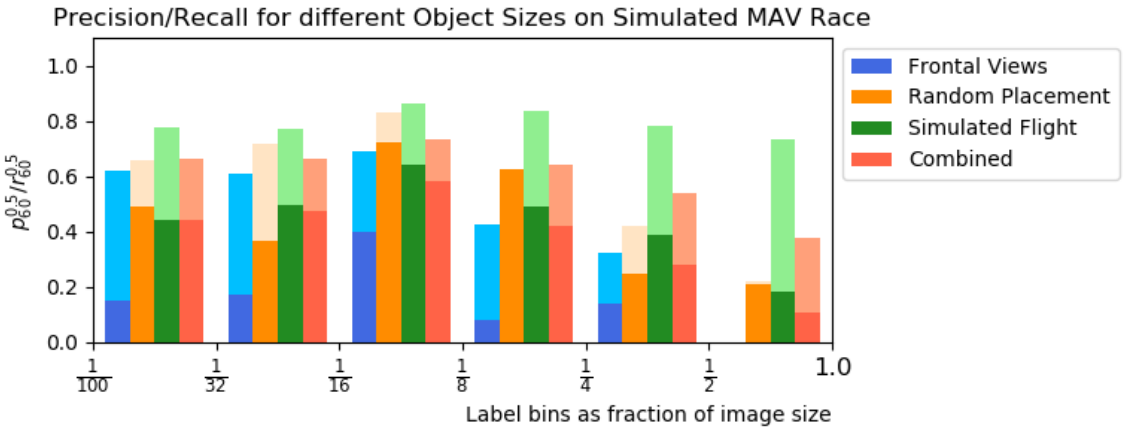


Figure 4.6: Precision and Recall at a confidence level of 0.5 of different methods to generate data. The results are clustered based on the size of the true/predicted bounding box. For each method and each bin precision and recall are shown. Precision is the lighter coloured bar. It can be seen how for *Simulated Flight* only the recall drops for larger objects. With *Random Placement* included, the precision drops similarly.

object size precision and recall drop to 0. The networks trained with *RandomPlacement* achieve a higher recall and precision but similarly the performance drops for larger objects. In contrast the network trained with *Simulated* flights, looses only recall for the bins with larger objects.

4.6.2 Discussion

In the results it can be seen how the detectors struggle in the more challenging environment of a MAV race. Compared to the results on the simple dataset of the previous section, a drop in performance can be seen for all detectors. Including more view points in the training set helped counteracting against this drop. The networks trained with more view points perform significantly better than the network which was only trained on frontal views.

Yet including too many view points seems to confuse the detector. The network trained with *Random Placement* performs poor for larger objects. Combining *Random Placement* and *Simulated Flight* leads overall to a deterioration in performance. In contrast, the network trained with *Simulated Flight* only achieves a higher performance. This is mainly achieved because the precision stays high. The detectors with *Random Placement* produce more false positives compared to the network trained with *Simulated Flight*.

Random Placement and *Simulated Flight* are limited in a way that they do not give control of which exact view points are present in the training set. A more sophisticated method could be the controlled creation of particular view points. This way a certain distribution of scales and angles could be created. On the other hand this creates other questions such as in what way to include overlap, etc. Furthermore, enough variation in background would be required for each view point. Controlling each of these parameters manually would require a lot of engineering work or a further extension of the created data generation tool. Nevertheless, it would be worth investigating and could be addressed in future work.

Overall a performance drop for larger objects can be seen. This is somewhat surprising as the object should be better visible when it is larger. However, the closer the camera gets, the less context is visible and the more likely it is that a part of the object is out view. This could be the reason for the drop in performance for larger objects. An experiment that would show whether this is true could be done by training the network without the pole and testing the model again.

4.6.3 Conclusion

In this section we evaluated how the detector performs when applied in a simulated MAV race. We hypothesized that the performance will decrease as more view points and overlapping angles are visible. The results confirm this hypothesis.

Furthermore, we investigated whether the detector can learn to detect the object from many view points. Although this seems possible, it comes at cost of precision for larger object sizes. We can conclude that restricting the network to a subset of view points is crucial to keep a better performance for larger objects.

Finally, the experiments show the limitations of the detection network. For larger object sizes the performance drops. This has to be taken into account when using the detector in a control loop.

4.7 Optimizing the Architecture

The baseline network architecture is optimized to detect solid feature -rich objects of multiple classes. In order to sufficiently represent and distinguish such complex objects many weights are required. This leads to a high computational complexity and longer inference time. Also, more weights require typically more data to train on and are more prune to overfit to a training set. The features of EWFOs are relatively simple hence less weights should be required. This section investigates whether equal performance can be reached when using a more shallow/thinner architecture.

The width in a CNNs determines how many features can be extracted in one layer. Wider networks can extract and retain more information than thinner layers. As the information in EWFOs is limited to basic geometric shapes and colour, we hypothesize a thinner network should be able to learn the detection task equally well. Only when reducing the width too strong, the performance should drop significantly as a minimum of weights is required. We examine this by training networks with thinner architectures. Therefore the number of filters in each layer of the *SmallYoloV3* network is reduced to $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$ and $\frac{1}{16}$ of its original number. The network is trained on *Race Courts*. All architectures are tested on the simulated MAV race introduced in Section 3.3.

The depth of a network has multiple effects. Deeper networks contain more non-linear elements and can thus represent more complex functions/features. EWFOs have simple features and thus only a few layers should be required. In fact the features that are relevant to detect an EWFOs are typically lines on the border of its location. Hence, a detector would only need to detect those edges in order to determine whether an object is present. This should be possible with a few amount of layers. On the other hand, in an MAVs race the objects appear overlapping and in difficult angles. In such cases the above is not valid anymore. For overlapping objects a detector would need to distinguish which line/edge belongs to which object. Such complex relations can be better represented with a deeper network. We investigate these hypotheses in an experiment. Therefore the baseline *SmallYoloV3* architecture is changed. The convolutional and pooling layers 5 to 3 are removed in stepwise experiments. An overview can be seen in Figure 4.7.

These networks and *VGGYoloV3* is trained on the *Race Courts*. All architectures are tested on the *Simulated MAV Flight*.

It should be noted that throughout the thesis many more architectures have been evaluated. For example using larger kernels, dilated convolutions, varying pooling operations at different layers. However, in the end none of these architectures led consistently to a higher performing network. While under certain conditions improvements could be observed, these usually disappeared when applying the detector on a different test set. Overall reduction in weights led to a similar decrease in performance as can be seen in the results of this experiment. Therefore, we keep the basic structure of the baseline architecture as it follows a pattern that is

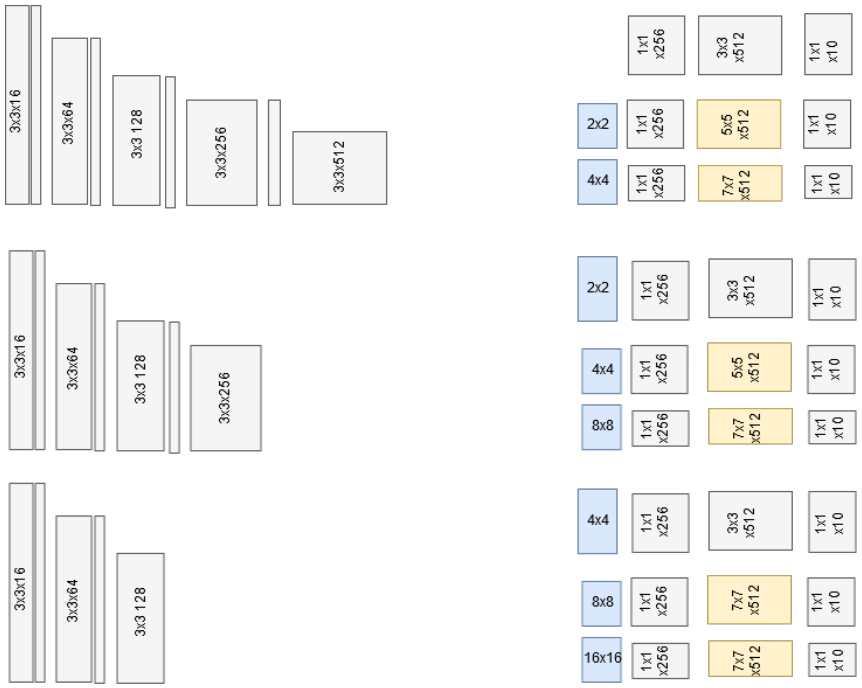


Figure 4.7: Overview over the changes in architecture to remove layers. From the baseline architecture of Figure 3.4 layers are removed step wise. In order to keep the receptive field constant and the output resolution the same, the pooling is increased.

intuitive to understand. We show how the results change when varying the parameters of the architecture on a high level basis (depth/width).

4.7.1 Results

Table 4.4 shows the results of networks with varying depth. The best network on simulated MAV race has 9 layers. Further increasing depth does not lead to an improvement in performance. In contrast reducing the number of layers decreases the performance gradually.

Layers	Simulated MAV Race ap_{60}
15	0.47 ± 0.00
9	0.51 ± 0.04
8	0.46 ± 0.01
7	0.43 ± 0.01
6	0.35 ± 0.02

Table 4.4: Performance of networks with varying depth on the simulated MAV race. It can be seen how on the more complex test set depth only improves the performance until 9 layers.

Table 4.5 shows the results of networks with varying width. The best architecture is the baseline with 51%. Reducing the width leads to a decrease in performance, particularly when only half the amount of filters is used. A further reduction in weights leads to a more gradual decrease in performance in steps of 1-2%. The lowest result is obtained when using a width of $\frac{1}{16}$.

Width	Simulated MAV Race [ap_{60}]	Weights	Reduction in Weights [%]
1	0.51 ± 0.04	12133646	100.0
$\frac{1}{2}$	0.43 ± 0.03	3039638	25.1
$\frac{1}{4}$	0.41 ± 0.05	763034	6.3
$\frac{1}{8}$	0.40 ± 0.02	192332	1.6
$\frac{1}{16}$	0.38 ± 0.00	48881	0.4

Table 4.5: Performance of networks with varying depth on the simulated MAV race.

4.7.2 Discussion

Similarly to the results in Section 4.4, the very deep network does not improve the detection performance. Hence, depth does not seem to help when detecting objects with more overlap and in difficult view angles. However, a certain amount of layers seems to be crucial. When decreasing the number of layers further than 9 the performance gradually decreases. A possible reason is that with less layers the pooling has to be increased in order to keep the receptive field large enough. Yet aggressive pooling removes spatial information quickly. Potentially this causes the performance to deteriorate.

For lower width, we hypothesized that the performance will only be affected for a very small amount of filters. However, the results show how the performance decreases already with half the amount of filters by 8%. Further reduction leads to smaller decreases until a performance of 38% is reached. That is despite using only 0.4% of the weights of the original architecture the performance drops only by 13%. This is a useful insight for the deployment of the detector on a MAV.

4.7.3 Conclusion

We investigated how depth affects the detection performance. Our hypothesis was that a deeper network could perform better at detecting objects in difficult angles or higher overlap. However, we could not confirm this hypothesis. Yet decreasing the depth to less than 9 layers also hurts the performance.

We investigated how width affects the performance for the detection of EWFOs. We hypothesized that due to the low variance in the investigated object and the simple features, less filters are required than in the baseline architecture. We can not really confirm or reject this hypothesis. Although a performance drop can be observed it is relatively low compared to the amount of weights that are removed. Nevertheless, this is a useful insight for the deployement of a detector on a MAV.

4.8 Transferring the detector to the real world

With the insights gained in simulation, experiments on real data can be performed. This section investigates the reality gap and several methods to reduce it.

As a test set the real world datasets introduced in Section 3.3 are used. The examples in Figure 3.5 show several properties that are different to the samples created in simulation. The objects used in this dataset consist of square bars and a black pole. In contrast, the CAD models used for synthesizing data consist of round bars and are uniformly coloured. Also, the aspect ratio between both objects is different. In contrast to the synthetic objects, the objects in the real world set are more wide than high. Finally, the colour is not exactly the same. A network only trained on the available CAD models is likely to overfit to the colour and

object shape. In order to evaluate this hypothesis a new 3D Model is created and samples are included in the training set. The new model can be seen in Figure 4.8.

Figure 4.8: In order to increase the variance in object appearance a new 3D-Model is created using *AirSim*. It contains a black pole and square bars.

Furthermore, it can be seen how motion blur and lens distortion change object appearance. We hypothesize that including such effects in the training data can improve the performance on the real data. The experiments in [5] show how the incorporation of sensor effects particularly improves the performance of models learned on fully synthesized data. We study this incorporation by applying image augmentation with the models introduced in Section 3.1.2.

We train the detector on *Race Courts* and add 3000 samples of the new 3D model. During training one or several image augmentation methods are used. The parameters are chosen visually and as the following:

- Distortion: $k_1 = k_2 = 0.7$.
- Blur: σ is drawn from a uniform distribution: $\mathcal{U}(0.1, 1.5)$.
- Chromatic: $t_x^R, t_y^R, s^G, t_x^B, t_y^B$ are drawn from $\mathcal{U}^R(-2,2)$, $\mathcal{U}^G(0.99, 1.01)$,$\mathcal{U}^B(-2,2)$.
- ΔH,ΔS,ΔV are drawn from $\mathcal{U}^H(.9, 1.1)$, $\mathcal{U}^S(0.8, 1.2)$, $\mathcal{U}^V(0.8, 1.2)$

Initial experiments show relatively poor results on the *Hallway* dataset. In this data set the objects are placed in front of a window which causes the objects to appear quite dark. Also the background is light and contains many details. We hypothesize that the lack of colour and the background confuse the detector. In order to examine these hypotheses, two additional networks are trained:

- *Grey* is trained by transforming 50% of the images to grayscale images during training. This should force the network to learn more colour invariant features and thus improve the results on the *Hallway* dataset.
- *VOC Background* is trained by including the images created with backgrounds chosen from the VOC dataset (Section 4.5). We hypothesize that by providing more variance in the background, the network will be more robust against details in the background. This should improve the results on the *Hallway* dataset.

Next to evaluate the network on real data a comparison to the baseline is given. *SnakeGate* is evaluated on each of the test sets. The colour filter is tuned for each environment. As the initial sampling of *SnakeGate* is stochastic, mean and standard deviation of 5 repeated runs are presented.

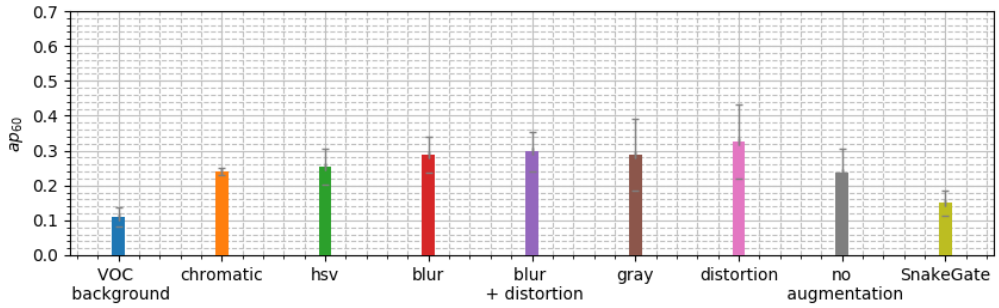


Figure 4.9: An overview of performance over all applied augmentation methods. The results are presented as the mean ap_{60} over the three data sets. The results of 4 runs are reported. It can be seen how *blur* and *distortion* as well as *grey* lead to an improvement in performance compared to not using augmentation.

4.8.1 Results

Figure 4.9 shows the mean performance of networks trained with different augmentation methods. Variations in HSV, chromatic aberration as well as including backgrounds from the Pascal VOC datasets lead to a deterioration in performance compared to not using any augmentation. In contrast, including blur, distortion and gray images leads to an improvement. The overall best results can be achieved by including distortion and gray images in the training.

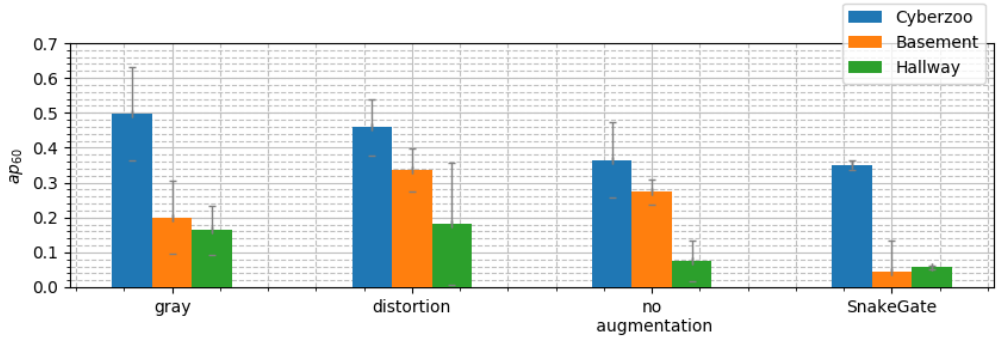


Figure 4.10: The results of Figure 4.9 in more details for the different data sets. It can be seen how *grey* improves the performance mainly on *Cyberzoo* and *Hallway* while the performance on *Basement* gets worse. In contrast *distortion* leads to a more overall improvement. Particularly for *Cyberzoo* and *Hallway* the variance in the results is high.

Figure 4.10 shows the results for the best augmentation methods in more detail. It can be seen how *grey* improves the performance mainly on *Cyberzoo* and *Hallway*. In contrast, on *Basement* the performance drops compared to not using any augmentation. *Distortion* leads to a more even improvement for all data sets.

Particularly for *Cyberzoo* and *Hallway* the variance in the results is high. This is less the case for the *Basement* dataset.

Figures 4.11 to 4.13 show example predictions of the network trained with distortion on the various data sets. On *Cyberzoo* and *Basement* several very accurate predictions can be seen, even under heavy occlusion, while on *Hallway* this is rather an exception. Yet on all data sets, several images where the object is clearly visible are wrongly or not at all detected.

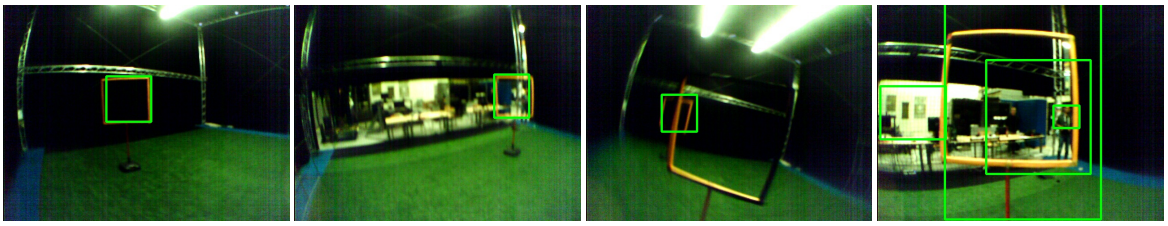


Figure 4.11: Predictions in the *Cyberzoo* environment of the network trained with distortion. Several gates are detected with good localization, even when occluded by another gate (3.). Yet the gate which is in front is not detected. Also, the perfect visible gate (4.) is not well detected. Instead many false positives are predicted.

Figure 4.12: Predictions in the *Basement* environment of the network trained with distortion. The network produces very nice detections in cases of heavy occlusion (1 and 2). However, in other cases where the gate is visible at least equally well (3 and 4) the gate is not detected or with bad localization.

Figure 4.13: Predictions in the *Hallway* environment of the network trained with distortion. In some images (1 and 2) the network predicts a half way correct bounding box. Yet in the majority of images such as 3 and 4 the gate is not detected, although clearly visible.

4.8.2 Discussion

In this experiment the detector is evaluated on real data. Only on *Cyberzoo* results comparable to simulation can be achieved. On the more difficult data sets *Hallway* and *Basement* the performance drop is quite significant. On *Hallway* almost no detection can be achieved. Hence, there seems to still be a big reality gap.

Where exactly the gap lays in is difficult to identify. Among image augmentation that addresses the colour feature such as *HSV* and *Grey*, only *Grey* leads to an improvement. Thereby the main improvement is obtained on the *Cyberzoo* dataset where illumination/colour is of smaller influence.

Lens distortion and blurring lead to an overall improvement on the datasets. However, the improvement cannot be particularly appointed to certain samples that suffer from heavy distortion or motion blur. Also, a high variance in the results can be observed which makes it difficult to conclude something. This holds especially for *Basement* and *Hallway* which are comparable small data sets. Potentially a larger real world test sets could give better insights.

For *Hallway* the results are particularly low. We hypothesized that the reason is the noise in the background and that including backgrounds from the Pascal VOC dataset can lead to an improvement on this dataset. However, including these samples seem to further confuse the detector. The results deteriorate. On *Hallway* the images are also taken against light background which makes the gate appear dark. We hypothesized that including grey images in the training improves the results on this dataset. We can observe an improvement from 8% to 16% by applying this technique. Hence, it seems that the technique helped a bit however it is not the predominant problem on this dataset.

Compared to *SnakeGate* the network achieves better results. Especially, when including distortion the results improve by 16%. Hence, the learning based detector seems to have learned a more robust detection of the racing gates.

In several examples in Figures 4.11 to 4.13, it can be seen how clearly visible objects are not detected. In simulation this is less of a problem. Hence, it seems that there is still a gap between the simulated data and the one present in the real world.

4.8.3 Conclusion

We investigated how the detector performs in the real world and if image augmentation can bridge the reality gap. We can conclude that image augmentation can improve the results on the real world data. In particular the modelling of lens distortion and the incorporation of grey images improve the results. Nevertheless, these results are preliminary as the variance in the results does not allow a final conclusion. Potentially, a more sophisticated test set could give more insights.

In more difficult environments the performance drops significantly. Also, several clearly visible objects can not be detected. As this typically not happens in simulation, we can conclude that there is still a large reality gap.

What exactly the reality gap is, is difficult to identify. The most apparent differences such as colour and deformation by distortion or general noise are addressed. It is possible that the data generated with the simulator is limited by the graphical engine. Even though the *UnrealEngine* creates high quality images, it cannot fully capture the complexity of real world data. Potentially, the results can be improved when including real world data in the training set. Such experiments should be conducted in future work.

Overall an improvement of the learning based detector compared to the baseline can be reported. The results in simulation show potential further improvement with better training data.

4.9 Deploying the detector on an MAV

In the application of the detection of EWFOs on a MAVs, detection accuracy is only one of the important metrics. Equally relevant are inference speed and energy requirements. The example control loop in which the detector of this work is integrated contains a filtering stage which fuses measurements of different sensors over time to infer a global state. This stage can deal with outliers and henceforth it can be more important to have more bad detections in high frequency than only slow but good ones. This section studies the deployment of a detector for EWFOs on a MAV in the example of the target system of this work. Therefore the speed-accuracy trade-off is studied and an experiment in a real world flight is conducted.

As explained in Chapter 2, the execution time strongly depends on the used hardware as well as its low level implementation. Therefore the inference time of several layers is measured on the JeVois using the *Darknet* framework. The results are displayed in Figure 4.14. Each sample corresponds to the number of computations and their computational time in one layer. Dashed lines connect samples at the same resolution. It is apparent how the same amount of computations at a spatial resolution of 20x15 is more than 4 times faster than at a resolution of 160x120. We assume this is because parallelism is better exploited at the lower scale.

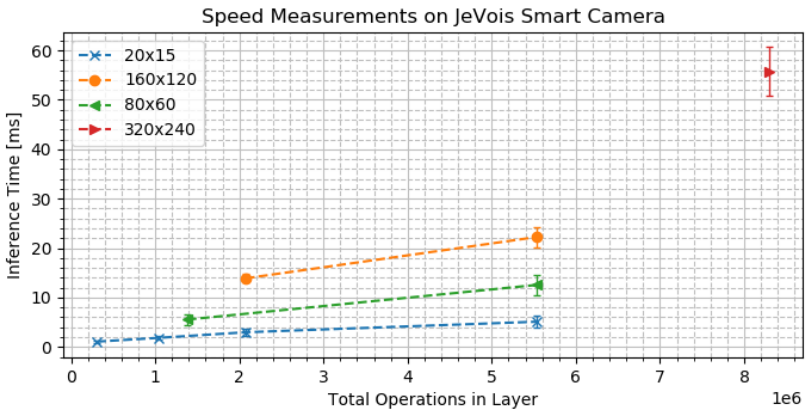


Figure 4.14: Inference Time of different layers on the JeVois. Each sample corresponds to the inference in a single layer. Dashed lines connect samples at the same resolution. It can be seen how an operation at a higher spatial resolution is significantly slower.

This means most speed can be gained when reducing the number of computations in the early layers where the spatial resolution is high. Section 4.7 it could be seen that already a small amount of filters in the early layers is enough to detect EWFOs. However, even evaluating 4 kernels at a resolution of 320x240 already takes 55 ms (Figure 4.14 red triangle). A total network of that size would require more than 100 ms and is thus too slow to be deployed in the control loop.

Current research mostly aims at reducing the computations when the convolved volumes are deep or the operations are performed on CPUs that do not support floating point operations. However, the bottleneck on the JeVois happens at shallow volumes and the hardware can perform floating point operations. Furthermore, EWFOs consist of thin elements that are spread over large parts of the image. Hence, we hypothesize that simply reducing the image resolution will lead to large drops in performance. An alternative is to increase the stride parameter in the early layers of the network. This reduces the number of locations at which the kernel is evaluated. EWFOs are sparse and spread over large parts of the image, while most of the image does not contain useful information. Hence, we hypothesize that increasing the stride parameter in the early layers will perform better than reducing the image resolution.

In order to evaluate our hypotheses, the network is trained with different configurations. Based on the results in Section 4.7 the architecture with $\frac{1}{4}$ of the original width is chosen. Stepwise the first pooling layers are

removed and instead the convolutions are applied with larger strides. Subsequently, performance on the real data and inference time on the *JeVois* are measured. Therefore the detectors are evaluated for 1 minute and the mean inference time is reported.

The *JeVois* supports aspect ratios of 4:3 and resolutions of 160x120, 320x240 and 640x480. Although, the detector does not depend on the image size, the object appearance can change at lower image resolution. Hence, during training the images are scaled to 160x160 or 320x320 respectively. The anchor boxes are scaled in similar fashion. Finally, as the input image resolution decreases, the output grid size decreases by the same factor. This is not desirable as the output resolution should stay the same. Hence, when decreasing the input image pooling layers are removed such that the output grid stays at 20,20 or 10,10 respectively.

4.10 Results

Figure 4.15 shows the results of the conducted experiments. Next to the different architectures, time and performance of *SnakeGate* are given. *SnakeGate* has high variance in time as its execution time depends on the colour filter stage. If no object is visible and everything is filtered by the colour filter, *SnakeGate* has an execution time of less than 1ms. For detection however some minimum execution time is required.

The plot shows how the different networks are generally slower than *SnakeGate*. Only the network applied at 160x120 gets with 55 ms/frame near the maximum execution time of *SnakeGate*. In terms of average precision, the networks show a better performance. The best results of 32% can be obtained by the network with one stride layer. Replacing more pooling layers with larger strides leads to a drop in performance but an increase of inference speed.

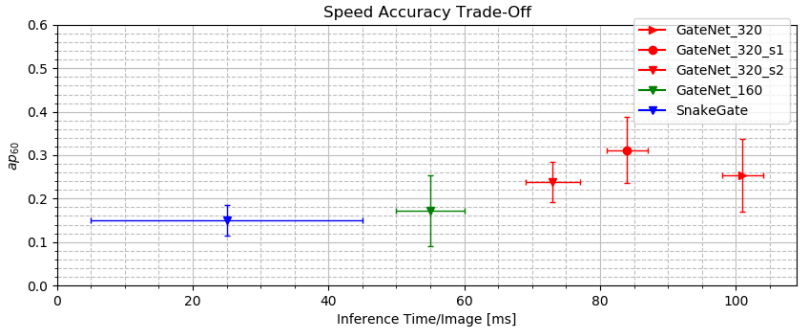


Figure 4.15: Inference Time of different layers on the JeVois. Each sample corresponds to a single layer. On the x-axis the total number of multiplications in that layer is displayed. It can be seen how an operation at a higher spatial resolution is significantly slower.

4.10.1 Discussion

As expected the performance is better when keeping a higher resolution but increasing the stride parameter. At a resolution of 160×120, the performance of the detector drops almost to the level of *SnakeGate*. In contrast the network with one stride layer achieves the best results.

Replacing the first pooling layers with larger strides gives a speed up of 17 ms/frame without loosing any performance. Further replacing the pooling layers leads to a stepwise decrease in performance. Hence, replacing the pooling layers with larger strides can be used to trade-off between performance and inference speed.

4.10.2 Conclusion

In this experiment we evaluated the trade-off in terms of speed and accuracy on the *JeVois* and compared it to the baseline. It could be seen how replacing pooling layers with larger strides can be used to trade-off inference speed and detection performance.

Overall the same inference speed as *SnakeGate* can not be achieved by the network. Hence, we can conclude that the learning based detector improves detection performance at cost of inference speed.

Chapter 5

Conclusion & Future Work

This work investigated the detection of EWFOs on MAV using a CNN. In this section a final discussion is given and a conclusion is derived. Furthermore, the research questions are answered and possible future work is discussed.

The research was motivated by the promising results of Deep Learning based Object Detectors and several drawbacks of manually crafted algorithms for the detection of racing gates in MAV races. As the manually tuned features prove to be sensitive to light changes as well as to object occlusion, the aim was to investigate a more robust method.

As no real training data were available, images and corresponding object labels were created with a simulator in order to train the CNNs based Object Detector *YoloV3*. We hypothesized that due to the simple shape of EWFOs a small network should be able to learn the detection task. This led to the two research questions of this work which are now answered based on the conducted experiments.

RQ1 How can data be generated to train an object detector for EWFO detection on a MAVs?

It could be seen how the detector is sensitive to overfit to environmental conditions present in the training set. When testing a detector in a different simulated environment than the training set, the performance deteriorates between 30% and 70%. It was further investigated how to make the detector more invariant against such environmental changes. The results show how the variance in background is less important than the creation of realistic light conditions in the training set.

When a wide range of view angles is introduced in training and test sets, the performance drops particularly for larger objects. It seems the detector has difficulties learning the detection from many angles. Better performance can be achieved by reducing the number of view angles in the training set. As this raised the question of how to create realistic view points, we proposed to simulate a flight through a race court. This way the created samples resemble the real world better. Even on unseen race courts, the detector achieves a precision of 70% compared to the 20% achieved by the network trained without simulating a flight.

Simulating flights as well as randomly placing the camera lack of control about the actual samples present in the training set. It is hard to think about all the view points that are required to train the network for a MAVs race. However, more control about the view points could give more insights about how the detector performs with different view angles in training and test set. Hence, future work could extend the data generation tool with more control about the view points and to conduct further experiments in that respect.

In order to transfer the detector to the real world, it was found that image augmentation is crucial.

Particularly modelling distortion improved the results on the real data. Yet there remains a gap between the results obtained in simulation and on real data. It cannot fully be resolved whether this is because of the complexity of the real data, or whether there are certain properties missing in the data generation process. Future work could address this issue by including real data in the training process. If this improves the results significantly, the problem is likely because of a reality gap. Otherwise, there might be a more fundamental problem in the chosen detector/complexity of the test set.

In summary, we propose to fully synthesize environments when creating training data for the detection of EWFOs on MAVs. Furthermore, the precision can be improved by training the detector based on view angles it will see in the real world, possibly by simulating the flight behaviour. In order to transfer the detector to the real world, we recommend to use image augmentation. Particularly augmenting the images by modelling lens distortion improves the performance on the investigated dataset.

RQ2 How can EWFOs be detected using a CNN on a MAVs?

The *YoloV3* can be adapted to the detection of EWFOs. The experiments showed how the detector can be confused by structures that are present within the empty part of the object. This was resolved by providing samples from different backgrounds and light conditions. We hypothesized that many backgrounds are required to achieve background invariance. However, the experiments showed that a small number of different environments is already enough to make the detector more robust against such confusion.

A general drop of recall was observed for closer objects. It can be assumed that this is because it is more likely that object parts are out of view when the object comes closer. For the detection of racing gates on MAVs this result can be taken into account in later stages of the control loop. For example by using a dedicated detector for closer gates and combining the information. For the general detection of EWFOs, this result is interesting. Despite being clearly visible for a human, the detector struggles in some examples. Further experiments could include investigating how the performance changes when the detector is trained without any context such as the object pole.

As EWFOs consist of relatively simple features, we hypothesized that a small network should be able to learn the task. The results showed that a shallow network of 9 layers performs equally well than a network with 15 layers. However, further reducing depth gradually reduces performance from 51% to 35%. When reducing the width of a network with 9 layers it can be seen how the performance slowly deteriorates from 51% at its original size to 38% to a fraction $\frac{1}{16}$ of its original size. Hence, we conclude that a minimum of 9 layers is required if detection performance is the most important metric.

For the detection of EWFOs on MAVs computational resources are more important. Our results show that a network with only one filter in the first layer can still achieve 38% average precision. Hence, by reducing the network size to 0.4% of its original size the performance drops only by 13% average precision.

The reduction of network size was taken out by retraining the network with a thinner architecture. An alternative way of reducing the network size is to apply *Knowledge Distillation* (Chapter 2). This method showed promising results on deeper network. In future work it could similarly be applied for the detection of EWFOs.

The trained detector was deployed on an example MAV. The results show how by reducing the resolution the inference time can be increased by 45 ms/frame. However, the costs in terms of average precision are large as only 18$ average precision can be achieved. Instead we propose to remove pooling layers and to use convolutions with larger strides. This parameter allows to trade-off between average precision and inference speed. An overview of the trade-off is given in Figure 4.15.

An alternative way to increase the network speed is *weight quantization* (Chapter 2). As the target sys-

tem of this work supports floating point multiplications, we did not further consider it. However, with an adequate low level implementation this could still lead to further speed up and should be investigated in future work.

In a nutshell, it could be seen that a small network is able to learn the task. The inference time of the detector can be further increased without loosing too much performance. Finally, replacing pooling layers by convolutions with larger strides allows to trade-off between detection performance and inference speed.

Based on the experiments we can give recommendations about the generation of data for the detection of EWFOs. Furthermore, we have insights of the limitations of the detector for example a drop of recall for larger object sizes. These insights can be taken into account when using the detector in a control loop. Finally, a detector could be developed and compared against the baseline. The experiments showed an improvement of performance compared to *SnakeGate* of up to 16% average precision, leading to a total of 32 % ap_{60}. This improvement is mainly obtained in cases of difficult light conditions or occlusion. Hence, it can be concluded that indeed the CNNs can work better in such situations.

A network of similar complexity achieves 41% ap_{60} on a simulated data set which contains more objects and more difficult view angles. Therefore it can be seen that the potential of the learning based detector is much higher. Yet transferring the detector to the real world proves to be difficult, despite the relatively simple features of EWFOs. In some cases the object is clearly visible but not detected by the CNNs. These results show the general drawback of Deep Learning based approaches. It is not transparent why the objects are not detected and what exactly was learned by the network.

A frequent argument for Deep Learning is that it does not require cumbersome Feature Engineering. In this work, this step was technically replaced by Data Engineering and yet the remaining reality gap is high and some results are hard to understand. We have to ask ourselves if the lack of transparency, the amount of required data as well as the computational requirements are really worth the results gained in detection performance.

Nevertheless, this work serves as a baseline for future work. The initial experiments show how a small amount of environments is already enough to make the detector relatively invariant against background. These results should apply similarly to the real world. Hence, it is not too much work to create a real world training set, and the data could be augmented with the data created in this work.

Furthermore, the experiments show trade-offs in speed and detection performance. The results can be used to design a detector for EWFOs based on available hardware and project requirements.

Bibliography

[1] Artsiom Ablavatski, Shijian Lu, and Jianfei Cai. Enriched Deep Recurrent Visual Attention Model for Multiple Object Recognition. jun 2017. doi: 10.1109/WACV.2017.113. URL http://arxiv.org/abs/1706.03581http://dx.doi.org/10.1109/WACV.2017.113.

[2] A. Andreopoulos and J. K. Tsotsos. On Sensor Bias in Experimental Methods for Comparing Interest-Point, Saliency, and Recognition Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):110–126, jan 2012. ISSN 0162-8828. doi: 10.1109/TPAMI.2011.91. URL http://ieeexplore.ieee.org/document/5765998/.

[3] Alexander Andreopoulos and John K. Tsotsos. 50 Years of object recognition: Directions forward. *Computer Vision and Image Understanding*, 117(8):827–891, aug 2013. ISSN 10773142. doi: 10.1016/j.cviu.2013.04.005. URL http://linkinghub.elsevier.com/retrieve/pii/S107731421300091X.

[4] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple Object Recognition with Visual Attention. dec 2014. URL https://arxiv.org/abs/1412.7755.

[5] Alexandra Carlson, Katherine A. Skinner, Ram Vasudevan, and Matthew Johnson-Roberson. Modeling Camera Effects to Improve Deep Vision for Real and Synthetic Data. mar 2018. URL http://arxiv.org/abs/1803.07721.

[6] Yuhua Chen, Wen Li, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Domain Adaptive Faster R-CNN for Object Detection in the Wild. mar 2018. URL http://arxiv.org/abs/1803.03243.

[7] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative Adversarial Networks: An Overview. oct 2017. doi: 10.1109/msp.2017.2765202. URL https://arxiv.org/abs/1710.07035.

[8] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE. ISBN 0-7695-2372-2. doi: 10.1109/CVPR.2005.177. URL http://ieeexplore.ieee.org/document/1467360/.

[9] Samuel Dodge and Lina Karam. Understanding How Image Quality Affects Deep Neural Networks. apr 2016. URL http://arxiv.org/abs/1604.04004.

[10] M. Elbanhawi, A. Mohamed, R. Clothier, J.L. Palmer, M. Simic, and S. Watkins. Enabling technologies for autonomous MAV operations. *Progress in Aerospace Sciences*, 91:27–52, may 2017. ISSN 0376-0421. doi: 10.1016/J.PAEROSCI.2017.03.002. URL https://www.sciencedirect.com/science/article/pii/S0376042116300367.

[11] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338,

jun 2010. ISSN 0920-5691. doi: 10.1007/s11263-009-0275-4. URL http://link.springer.com/10.1007/s11263-009-0275-4.

[12] Davide Falanga, Elias Mueggler, Matthias Faessler, and Davide Scaramuzza. Aggressive Quadrotor Flight through Narrow Gaps with Onboard Sensing and Computing using Active Vision. URL http://rpg.ifi.uzh.ch/aggressive{_}flight.html.

[13] Pedro Felzenszwalb, David Mcallester, and Deva Ramanan. A Discriminatively Trained, Multiscale, Deformable Part Model. URL http://people.cs.uchicago.edu/{~}pff/papers/latent.pdf.

[14] Tapabrata Ghosh. QuickNet: Maximizing Efficiency and Efficacy in Deep Architectures. jan 2017. URL http://arxiv.org/abs/1701.02291.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. nov 2013. URL http://arxiv.org/abs/1311.2524.

[16] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks, mar 2010. ISSN 1938-7228. URL http://proceedings.mlr.press/v9/glorot10a.html.

[17] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. jun 2014. URL https://arxiv.org/abs/1406.2661.

[18] Kaiming He and Jian Sun. Convolutional Neural Networks at Constrained Time Cost. URL https://arxiv.org/pdf/1412.1710.pdf.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. jun 2014. doi: 10.1007/978-3-319-10578-9_23. URL http://arxiv.org/abs/1406.4729http://dx.doi.org/10.1007/978-3-319-10578-9{_}23.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. dec 2015. URL http://arxiv.org/abs/1512.03385.

[21] Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart, and Kurt Konolige. On Pre-Trained Image Features and Synthetic Images for Deep Learning. oct 2017. URL http://arxiv.org/abs/1710.10710.

[22] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, 18(7):1527–1554, jul 2006. ISSN 0899-7667. doi: 10.1162/neco.2006.18.7.1527. URL http://www.ncbi.nlm.nih.gov/pubmed/16764513http://www.mitpressjournals.org/doi/10.1162/neco.2006.18.7.1527.

[23] Andrew G. Howard. Some Improvements on Deep Convolutional Neural Network Based Image Classification. dec 2013. URL http://arxiv.org/abs/1312.5402.

[24] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. apr 2017. URL http://arxiv.org/abs/1704.04861.

[25] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, Kevin Murphy, and Google Research. Speed/accuracy trade-offs for modern convolutional object detectors. URL https://arxiv.org/pdf/1611.10012.pdf.

[26] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1254–1259, 1998. ISSN 01628828. doi: 10.1109/34.730558. URL http://ieeexplore.ieee.org/document/730558/.

[27] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks? oct 2016. URL http://arxiv.org/abs/1610.01983.

[28] Joshua Bateman. China Uses Drones for Earthquake Search and Rescue Missions | WIRED, 2017. URL https://www.wired.com/2017/01/chinas-launching-drones-fight-back-earthquakes/.

[29] Sunggoo Jung, Hanseob Lee, and David Hyunchul Shim. Real Time Embedded System Framework for Autonomous Drone Racing using Deep Learning Techniques. doi: 10.2514/6.2018-2138.

[30] Sunggoo Jung, Sungwook Cho, Dasol Lee, Hanseob Lee, and David Hyunchul Shim. A direct visual servoing-based framework for the 2016 IROS Autonomous Drone Racing Challenge. *Journal of Field Robotics*, 35(1):146–166, jan 2018. ISSN 15564959. doi: 10.1002/rob.21743. URL http://doi.wiley.com/10.1002/rob.21743.

[31] Kate Baggaley. Drones are fighting wildfires in some very surprising ways, 2017. URL https://www.nbcnews.com/mach/science/drones-are-fighting-wildfires-some-very-surprising-ways-ncna820966.

[32] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. dec 2014. URL https://arxiv.org/abs/1412.6980.

[33] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks, 2012. URL https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[34] Hei Law and Jia Deng. CornerNet: Detecting Objects as Paired Keypoints. aug 2018. URL http://arxiv.org/abs/1808.01244.

[35] Youngwan Lee, Huien Kim, Eunsoo Park, Xuenan Cui, Hakil Kim, and Communication Engineering. Wide-Residual-Inception Networks for Real-time Object Detection Youngwan. pages 2–8, feb 2016. URL https://arxiv.org/pdf/1702.01243.pdfhttp://arxiv.org/abs/1702.01243.

[36] Guohao Li, Matthias Mueller, Vincent Casser, Neil Smith, Dominik L Michels, and Bernard Ghanem. Teaching UAVs to Race With Observational Imitation Learning. mar 2018. URL https://arxiv.org/pdf/1803.01129.pdfhttp://arxiv.org/abs/1803.01129.

[37] Quanquan Li, Shengying Jin, and Junjie Yan. Mimicking Very Efficient Network for Object Detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7341–7349. IEEE, jul 2017. ISBN 978-1-5386-0457-1. doi: 10.1109/CVPR.2017.776. URL http://ieeexplore.ieee.org/document/8100259/.

[38] S. Li, M. M. O. I. Ozo, C. De Wagter, and G. C. H. E. de Croon. Autonomous drone race: A computationally efficient vision-based navigation and control strategy. sep 2018. URL https://arxiv.org/abs/1809.05958.

[39] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single Shot MultiBox Detector. URL https://www.cs.unc.edu/{~}wliu/papers/ssd.pdf.

[40] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, nov 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL http://link.springer.com/10.1023/B:VISI.0000029664.99615.94.

[41] Ratnesh Madaan, Daniel Maturana, and Sebastian Scherer. Wire detection using synthetic data and dilated convolutional networks for unmanned aerial vehicles. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3487–3494. IEEE, sep 2017. ISBN 978-1-5386-2682-5. doi: 10.1109/IROS.2017.8206190. URL http://ieeexplore.ieee.org/document/8206190/.

[42] Abdulghani Mohamed, Kevin Massey, Simon Watkins, and Reece Clothier. The attitude control of fixed-wing MAVS in turbulent environments. *Progress in Aerospace Sciences*, 66:37–48, apr 2014. ISSN 0376-0421. doi: 10.1016/J.PAEROSCI.2013.12.003. URL https://www.sciencedirect.com/science/article/pii/S0376042113000912.

[43] Robin R. Murphy, Satoshi Tadokoro, and Alexander Kleiner. Disaster Robotics. In *Springer Handbook of Robotics*, pages 1577–1604. Springer International Publishing, Cham, 2016. doi: 10.1007/978-3-319-32552-1_60. URL http://link.springer.com/10.1007/978-3-319-32552-1{_}60.

[44] C.P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 555–562. Narosa Publishing House. ISBN 81-7319-221-9. doi: 10.1109/ICCV.1998.710772. URL http://ieeexplore.ieee.org/document/710772/.

[45] Kuan-Chuan Peng, Ziyan Wu, and Jan Ernst. Zero-Shot Deep Domain Adaptation. jul 2017. URL http://arxiv.org/abs/1707.01922.

[46] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. Learning Deep Object Detectors from 3D Models. URL http://www.karimali.org/publications/PSAS{_}ICCV15.pdf.

[47] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN Features Off-the-Shelf: An Astounding Baseline for Recognition. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 512–519. IEEE, jun 2014. ISBN 978-1-4799-4308-1. doi: 10.1109/CVPRW.2014.131. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6910029.

[48] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. URL https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[49] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. jun 2015. URL http://arxiv.org/abs/1506.01497.

[50] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real Single-Image Flight without a Single Real Image. nov 2016. URL http://arxiv.org/abs/1611.04201.

[51] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. jan 2018. URL http://arxiv.org/abs/1801.04381.

[52] Stephen Shankland. Watch out, Amazon. Zipline's new medical delivery drones go farther, faster - CNET, 2018. URL https://www.cnet.com/news/zipline-new-delivery-drones-fly-medical-supplies-faster-farther/.

[53] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. sep 2014. URL http://arxiv.org/abs/1409.1556.

[54] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. sep 2014. URL http://arxiv.org/abs/1409.4842.

[55] Christian Szegedy, Scott Reed, Pierre Sermanet, Vincent Vanhoucke, Andrew Rabinovich, Marcel Simon, Erik Rodner, Joachim Denzler, Joseph Redmon, Ali Farhadi, Sergey Ioffe, Christian Szegedy, Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-yang Fu, Alexander C Berg, Sergey Ioffe, Vincent Vanhoucke, Alex Alemi, Scott Reed, Pierre Sermanet, Vincent Vanhoucke, Andrew Rabinovich, Jonathon Shlens, Zbigniew Wojna, Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, Kurt Keutzer, Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Tianqi Chen, and Carlos Guestrin. YOLO9000: Better, Faster, Stronger. *Data Mining with Decision Trees*, 7(3): 352350, 2016. ISSN 0146-4833. doi: 10.1142/9789812771728_0012. URL https://arxiv.org/abs/1612.08242.

[56] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. mar 2017. URL http://arxiv.org/abs/1703.06907.

[57] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. apr 2018. URL https://arxiv.org/abs/1804.06516.

[58] Michael Tschannen, Aran Khanna, and Anima Anandkumar. StrassenNets: Deep Learning with a Multiplication Budget. dec 2017. URL http://arxiv.org/abs/1712.03942.

[59] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171, sep 2013. ISSN 0920-5691. doi: 10.1007/s11263-013-0620-5. URL http://link.springer.com/10.1007/s11263-013-0620-5.

[60] Gergely Vass and Tamás Perlaki. Applying and removing lens distortion in post production. URL http://www.vassg.hu/pdf/vass{_}gg{_}2003{_}lo.pdf.

[61] P ; Viola and Jones. Rapid Object Detection Using a Boosted Cascade of Simple Features. 2004. URL http://www.merl.com.

[62] Yi Wei, Xinyu Pan, Hongwei Qin, Wanli Ouyang, and Junjie Yan. Quantization Mimic: Towards Very Tiny CNN for Object Detection. may 2018. URL http://arxiv.org/abs/1805.02152.

[63] Gao Xu, Yongming Zhang, Qixing Zhang, Gaohua Lin, and Jinjun Wang. Domain Adaptation from Synthesis to Reality in Single-model Detector for Video Smoke Detection. sep 2017. URL http://arxiv.org/abs/1709.08142.

[64] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. may 2016. URL http://arxiv.org/abs/1605.07146.

[65] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. jul 2017. URL http://arxiv.org/abs/1707.01083.