# Detecting Empty Wireframe Objects on Micro-Air Vehicles

Applied for Visual Target 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 14:00 AM.

| | | |
|---|---|---|
| Supervisor: | Assistant professor dr. ir. D. M. J Tax | |
| Thesis committee: | Associate professor dr. C. F. Guido de Croon, | TU Delft |
| | Dr. E. L. Brown, | TU Delft |
| | Ir. M. Scott, | Acme Corporation |

*This thesis is confidential and cannot be made public until December 31, 2013.*

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

TU Delft Delft University of Technology

# Preface

Preface...

*P. Duernay*
*Delft, January 2013*

# Contents

# Summary

# 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

**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

**SIFT** Scale Invariant Feature Transform

**SSD** Single Shot Multibox Detector

**SVM** Support Vector Machine

**TO** Target Object

**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 [53], fight fires [34] or even find survivors in disaster situations [31].

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 [44]. 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 [43]. 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 [14].

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 [48]. 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 [20, 22, 55, 66]. 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 [22] contains 60.2 million parameters and one inference requires 11.3 billion floating point operations [60].

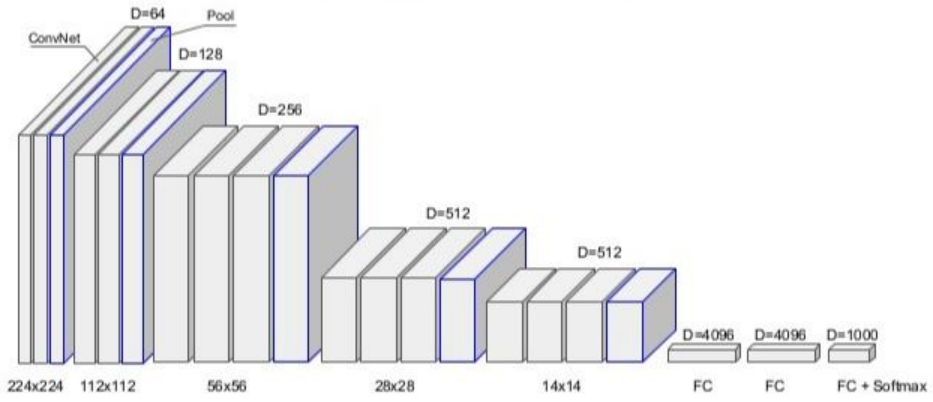


Figure 1.2: Example Architecture of a CNN.

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[17, 26, 37, 52, 66, 67]. 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.

Reference to Current Method once it is published

The thesis builds on previous work by Ozo et. al 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.

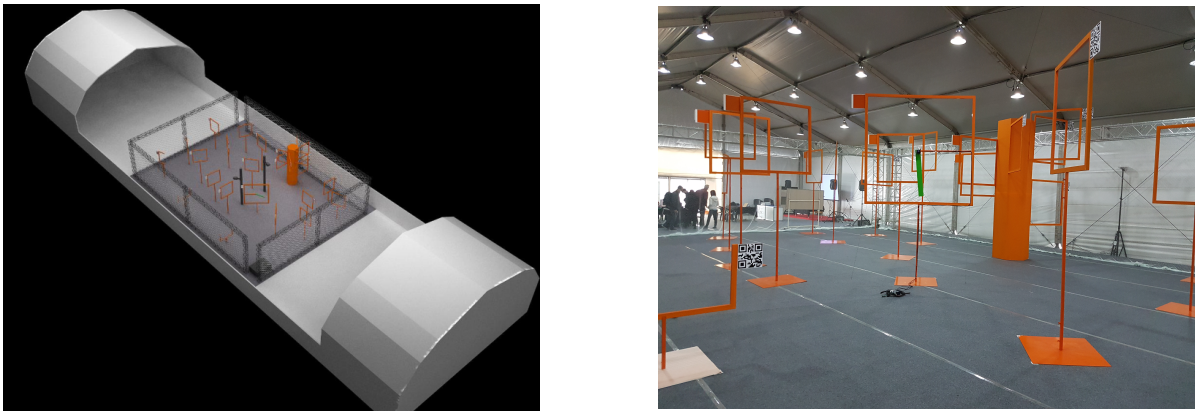

Figure 1.3: Example Images of the IROS 2016 Autonomous Drone Race

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 IROS drone race gates. These can be seen can be seen in Figure 1.4. The image shows the *Closed Gate* as well as the *Jungle Gate*. Thereby the orange part 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 [15] and [38] 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 [32] and [33] 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

[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 Images of the Empty Wire Frame Objects investigated in this thesis.

and therefore not many example images exist. To this end no publicly available dataset can be used to train 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 [6, 13]. 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 a detection model for EWFO detection on a MAVs?

**RQ2** What kind of architecture is suitable to detect EWFOs?

**RQ3** What are the trade-offs in detection performance and inference time when a detection model for EWFOs is deployed on a MAV?

**RQ4** Can the gained insights be used to build a lightweight and robust detection model for racing gates in the IROS Autonomous Drone Race?

## 1.2 Results/Contributions

Put some results at the end.

## 1.3 Outline

Refactor contributions once done

The thesis is structured as displayed in Figure 1.6. Chapter 2 describes the metrics and systems used for evaluation. **??**, **??**, **??** and **??** address the individual research questions. Each chapter contains an introduction to the topic, the methodology used in this thesis and experiments that have been carried out. **??** describes methods to generate synthetic data for machine learning. It concludes with the datasets used for the remaining parts of this thesis. **??** describes object detection and evaluates current methods in the application

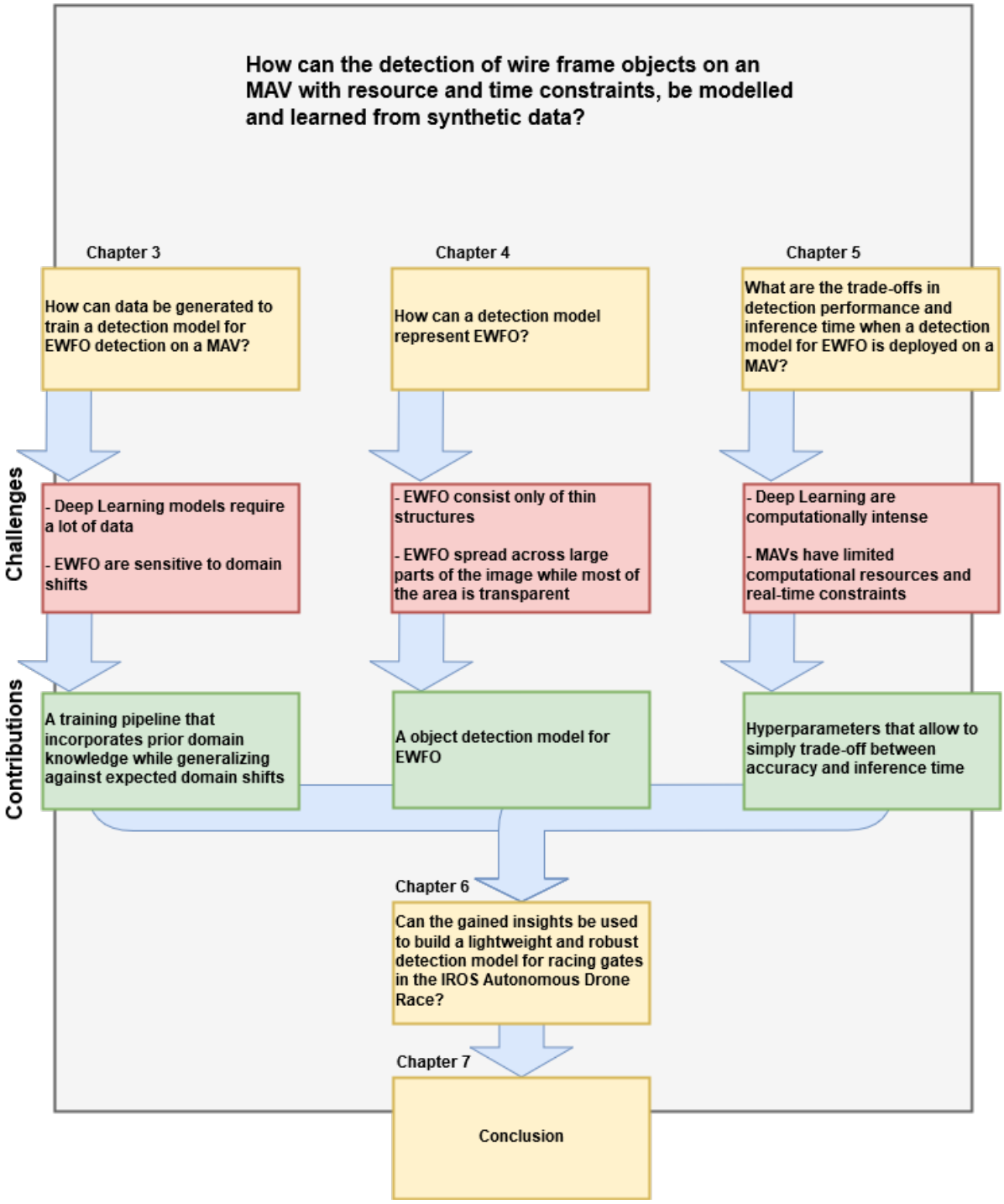


Figure 1.6: Thesis Outline

for EWFOs. **??** illustrates and evaluates measures to reduce computations and optimize an object detection system for a particular hardware. It investigates the trade-off between detection performance and inference time. **??** describes how the gained insights are used to develop a detector for racing gates at the IROS 2018 Autonomous Drone Race. It also compares the current method to a traditional image processing method in terms of speed and detection performance. **??** discusses the overall results and formulates a conclusion.

# 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.

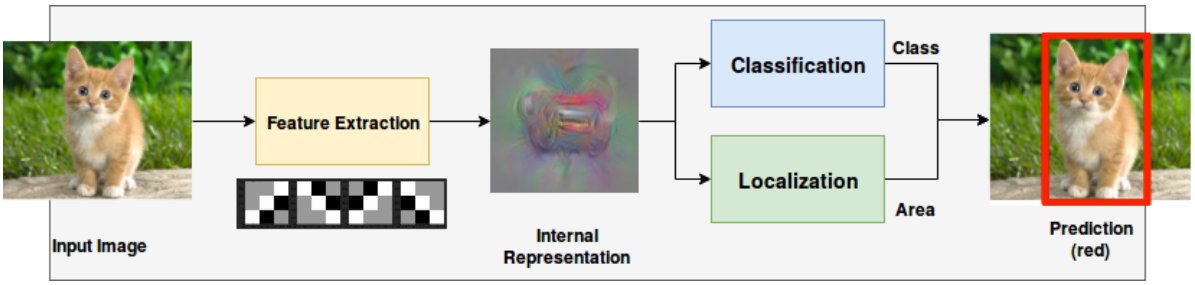


Figure 2.1: Object Detection Pipeline. An initial step extracts task relevant features of the input signal and derives an internal representation. Consecutively, classification determines what kind of object is present in the image, while localization determines where the object is located. The output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

add object model

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 Evaluation Metrics

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

**Precision**

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

**Recall**

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

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 Intersection over Union (IoU). In experiments we determine 0.6 as sufficient overlap for a detection.

The model used within this thesis associates a "confidence" value with each prediction that can trade off precision and recall. This is further explained in **??**. By accepting more detections with a lower confidence threshold, the probability increases that one of the predictions is a true positive. Hence, it increases recall. However, it also increases the probability of false positives and thus lowers precision. In order to evaluate this trade-off, precision is plotted over recall at increasing confidence values.

As the learning of CNNs is stochastic, the mean across several trainings is reported. In order to determine the average precision recall trade-off the precision is interpolated across evenly distributed recall levels between 0 and 1 using:

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

Subsequently the mean at all recall levels can be calculated. A metric that combines the precision-recall trade-off is Average Precision (AP):

$$ap = \int_r p_{interp}(r)\,dr$$

There should be a source for this

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

### 2.1.2 Baseline Algorithm

text

The above pipeline can be illustrated in the example of the baseline algorithm of this work: *SnakeGate*. Its scheme is summarized in and described here on a higher level.

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

The follow up stage searches for object parts. *SnakeGate* detects square objects and hence searches 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.

1. Filter image by colour threshold
2. Sample stochastically
3. Follow the pixels horizontally as long as they are within the colour threshold otherwise return to 2.
4. If a bar of sufficient length has been found repeat 3. vertically along one end of the line found in 3.
5. If a vertical bar is found the square is considered as gate candidate
6. Create local histogram around the corners of the gate candidate and choose the highest peak as gate corner.
7. Count the fraction of pixels within the color threshold in relation to the total number of pixels a long all edges of the gate candidate to determine the *color fitness*.
8. Gate candidates that exceed a chosen threshold are considered valid detections.

this can be described more formally

*SnakeGate* is fast to execute but lacks from several drawbacks. (1) the main feature is colour dependent and thus the method is very subtle to light changes. It must be fine tuned according to light conditions in a certain room. Strong colour variations across the object cannot be handled by the method; (2) the object is defined as the four bars, if one of the parts can not be detected e.g. due to occlusion, the object will not be found. Furthermore, the object model does not take into account context such as the object pole.

### 2.1.3 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 [7].

Later approaches focus more on appearance based features such as wavelets [45] which also applied in [63] 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 classifier assigns background to that patch. The features can be computed with simple operations and thus the detector can be executed extremely fast. However, the used Haar-wavelets cannot efficiently encode complex textures making the approach less suitable for many real world objects [7].

In contrast Histogram of Oriented Gradients (HOG) [12] and Scale Invariant Feature Transform (SIFT) [41] use the image gradient to cover shape information. In a sliding window local histograms based on the gradient orientation are calculated. Dalal and Triggs [12] 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 large variations in appearance. In order to cope with these issues Felzenszwalb et al. [16] 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 2.1.3 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.

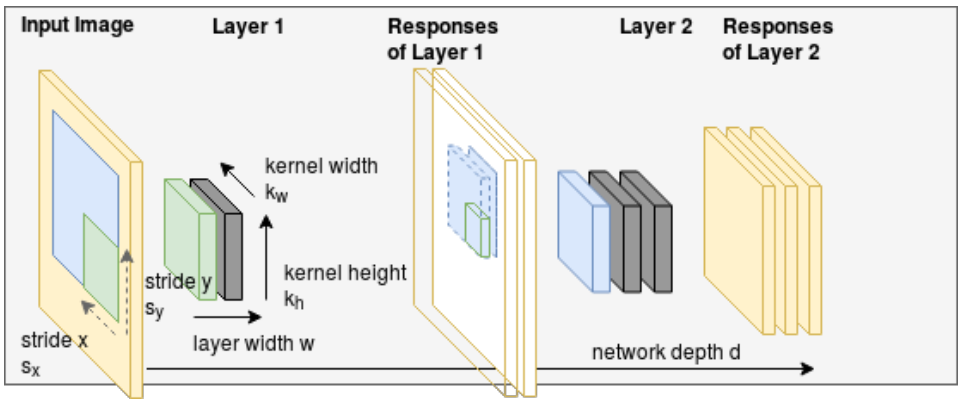


Figure 2.2: Notation in CNNs. The input image is convolved with a layer. One layer contains multiple kernels that are convolved in sliding window fashion with certain step size. The responses of the convolutions are collected and processed by the next layer. The receptive field is determined by which 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 [20]. Simonyan and Zisserman [54] 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. [55] 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. [22] 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 [66] 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* [26] and *QuickNet* [17] 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* [52] further includes linear bottlenecks to reduce the total number of operations. These are convolutions with a 1by1 kernel and linear activation.

**Channel Shuffling.** [67] addresses the computational costs of pointwise convolutions. Instead of applying a pointwise convolution on the whole input volume, group convolutions are applied on by dividing the channels in subsets. These channels are shuffled to enable cross-channel information propagation.

**Dense Connections.** *DenseNet* [27] proposes the use of dense connections in CNNs. Thereby the input of each convolutional layer does not only consist if its direct previous layer but of a concatenation of the activations of all its previous layers. By enabling feature reuse the total amount of parameters shall be reduced.

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. [18] use Selective Search [61] 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 [21] 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)[50] 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.

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)[56] 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) [40] 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 [49, 56] also included the concept of anchor boxes and prediction layers at multiple scales, making SSD and Yolo converge to a very similar solution.

In [28] 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*[36] 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 [5, 8, 29].

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.

### 2.1.4 Reducing Inference Time

Computer Vision research typically focuses on detection accuracy rather than inference speed. However, for the deployment on a MAV resource consumption is an important parameter. The following describes research topics that concern the reduction of inference time.

One pass of the state of the art CNN ResNet101 contains of 11.3 billion floating point operations. Even assuming a powerful 2 GHz processor can perform 2 billion calculations per second it would still require almost 6 seconds for one network pass. Furthermore, a Central Processing UniTs (CPUs) usually has to reload

cleanup

operands from the memory and cannot directly perform floating point multiplications. Hence, the actual forward pass would require even more time.

Therefore researchers address the reduction of inference time by reducing the number of operations. For example MobileNet and ShuffleNet aim to reduce the computations by addressing the expensive 3D convolutions. Other researchers aim to making the individual operations more efficient. This can be done by replacing floating point operations with integer operations hence quantizing the network.

In order to infer a layer in a CNNs kernels are convolved at multiple locations within one image. Thereby the same calculations are applied on various input data while the individual operations are independent of each other. Hence, theoretically the computations in one layer can be performed completely in parallel. The output of one layer can be stored similarly to the input image as a matrix. Computational platforms that exploit this fact are Graphical Processing Units (GPUs) that have been adopted for Deep Learning applications from early on. In contrast to CPUs that are optimized to perform many different operations sequentially, GPUs typically consist of more cores and are optimized to perform the same operation on different data. While each individual core is typically slower than a single core in a CPUs, GPUs are much faster for applications that can be performed in parallel.

With a large enough GPU it does not matter whether an operation is performed at an image of size 150x150 or 300x300 despite the actual computations increase by a factor of 4. However, such powerful GPUs require space and energy which is typically not available for small scale MAVs. The JeVois Smart Camera contains a MALI-400GPU with two cores and 408 Mhz each. Additionally it contains floating point registers and supports NEON operations. These are processor instructions that enable the efficient use of **SIMD!**s (**SIMD!**s) operations.

The actual inference time of a network strongly depends on the computational platform, memory access times as well as the particular low level implementation. Hence, the number of computations within a network can only give a limited insight on the actual inference time. For example the *TensorflowLite* framework allows the deployment of models created in *tensorflow* on mobile processors. However, with this framework a the application of a single kernel on a 104x104 input volume takes 27.1 ms on the JeVois. In contrast, the same operation performed with the *Darknet* framework that supports NEON operations takes only 5.07 ms.

#### Weight Quantization

The hardware of most embedded CPU support only integer operations and thus rely on software to perform floating point operations. Hence, weight quantization is another line of research to reduce the inference time of deep networks on mobile devices. By mimicking the quantization effects during training, the network learns to deal with these kind of artefacts. However, the target platform of this work supports hardware accelerated floating point multiplications. While weight quantization could still be used to accelerate the network it would require serious low level operation optimization. This work addresses the optimization on a more high level architectural basis.

put some refs where this is applied [59]

#### Knowledge Distillation

Knowledge Distillation [24] 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 [39] the approach is applied to the task of Object Detection. In [64] knowledge distillation is combined with weight quantization in order to obtain a faster and more efficient model.

The approaches have in common that a very complex function learned from a deep model is aimed to be compressed in a more efficient model. Training the smaller model directly leads to a lower performance than using the distillation approach. We assume this is successfully as the network architectures used are generally deep. Even [64] which speaks of very tiny networks uses the VGG-architecture with 23 layers. These architectures are prohibitive to be deployed on small scale MAVs. We investigate a network size of around 10 layers. Such networks are easier to train and our experiments how we can reduce the width of the model by simply training the network on the task. Therefore we do not investigate this approach further.

### 2.1.5 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. [35] use Principal Component Analysis (PCA) to incorporate colour variations. Howard [25] 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. [56] uses random scaling and translation of the input image, as well as random variations in saturation and exposure. Liu et al. [40] additionally crop and flip each image with a certain probability.

Since most methods use image augmentation and Krizhevsky et al. [35] 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.[9] 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.[47] 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.[42] 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. [23] 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 (further discussed in **??**) 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. [30] 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 [51, 57, 58] 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.

### 2.1.6 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 [10, 65].

While the aforementioned approaches require labelled samples from the target domain, Peng et al. [46] 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.

### 2.1.7 Generative Adversarial Networks

Generative Adversial Networks (GANs) [19] 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 [11]. 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.

## 2.2 System Overview

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 altitude and attitude 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.

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 extent the field of view a 120 degree wide angle lens is mounted. In Figure 2.4 the camera is shown.

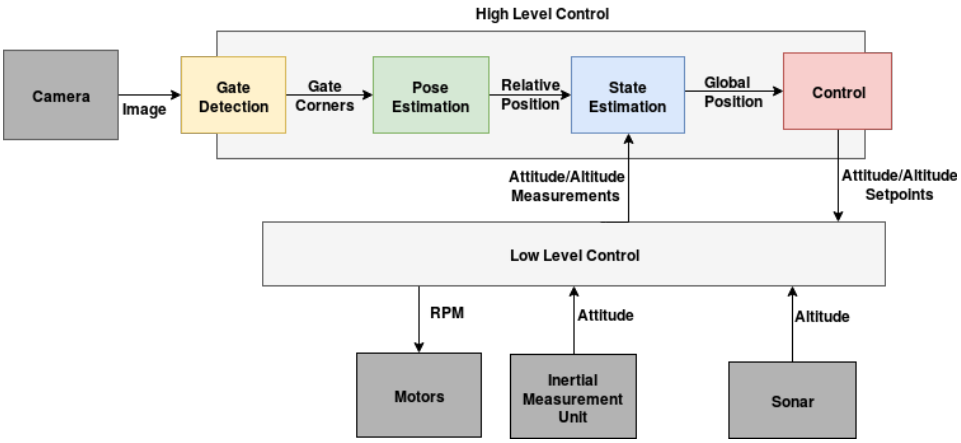


Figure 2.3: Control Loop

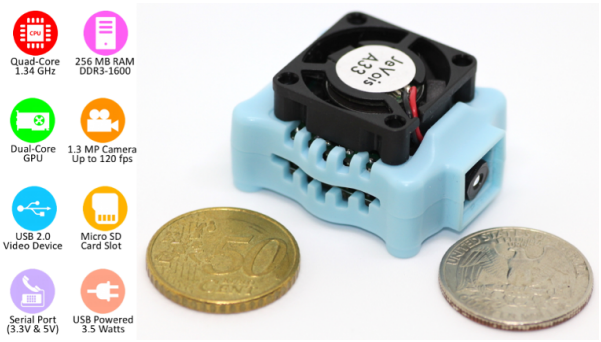


Figure 2.4: JeVois Camera

# Chapter 3

# Methodology

This chapter introduces a notation 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

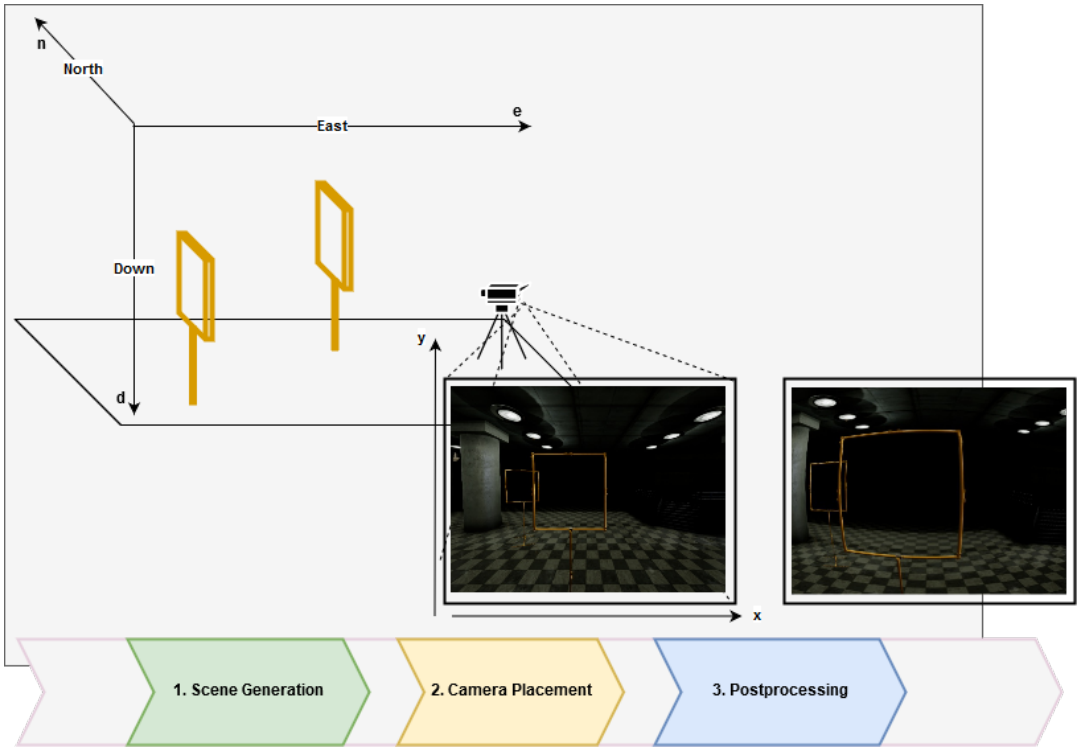


Figure 3.1: Overview of the data generation process.

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 North-East-Down (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*[4] and its *AirSim*-Plug-In[1] 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. This allows the automatic generation of ground truth labels while the camera is placed.

The *UnrealEngine* contains a profound amount of photorealistic image effects. However, their use with data generation tool would require a deep understanding of graphical programming 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 than the ones outdoors 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. Furthermore, three indoor base environments are created 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*

### 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 [9]. Variations in **HSV!** (**HSV!**) space are selected as it is a common image augmentation technique in current Object Detection pipelines. A visual overview can be seen in Equation (3.1). This section describes the mathematical models behind the applied the effects.

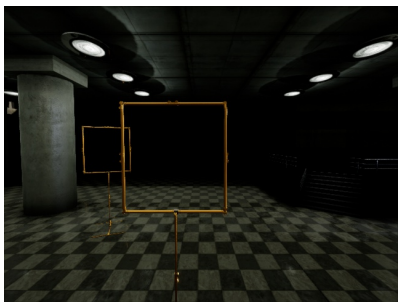

Figure 3.3: Original Image.

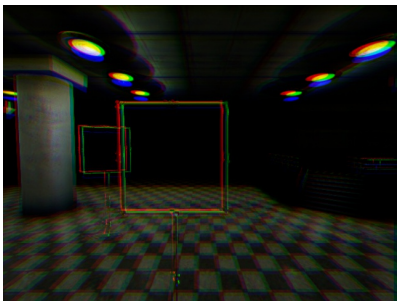

Figure 3.4: Chromatic Aberration. It can be seen how the red and green channel are shifted relative to each other. Thus two bars appear in the image.

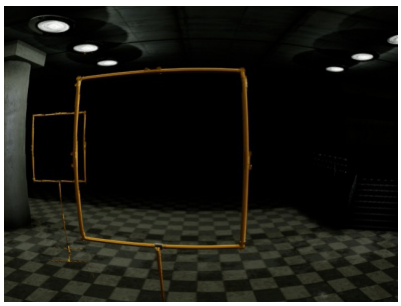

Figure 3.5: Lens Distortion. It can be seen how the previously straight lines appear as circular shape.

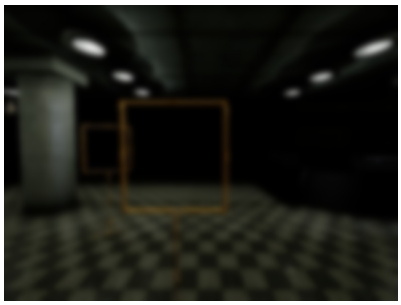

Figure 3.6: Out-of-Focus blur.

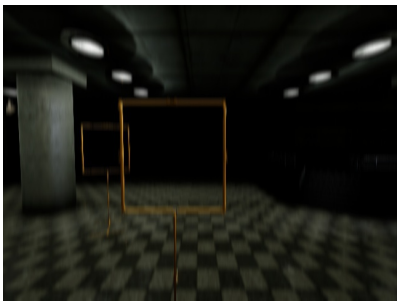

Figure 3.7: Vertical Motion Blur.

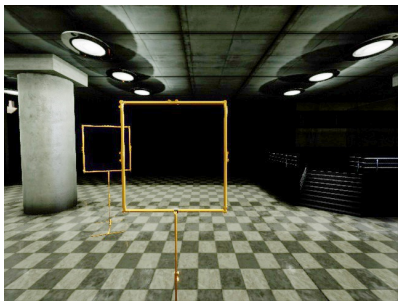

Figure 3.8: Exposure. It can be seen how lighter areas appear particularly light, while dark areas remain dark.

**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 [62]. It models the removal of lens distortion as combination of radial and non-radial part, that is approximated with a second order Taylor expansion:

double check formula is (y in first line?)

$$\begin{pmatrix} x_u \\ y_u \end{pmatrix} = f(x,y) = \begin{pmatrix} x(1+\kappa_1 x^2 + \kappa_1(1+\lambda_x)y^2 + \kappa_2(x^2+y^2)^2) \\ y(1+\kappa_1 x^2 + \kappa_1(1+\lambda_y)y^2 + \kappa_2(x^2+y^2)^2) \end{pmatrix} \tag{3.1}$$

Where:

- $x_u$ and $y_u$ are the undistorted coordinates.
- $\kappa_1$ $\kappa_2$ control the radial distortion
- $\lambda_x$ and $\lambda_y$ control the tangential distortion

Applying the lens distortion to an image is done using the inverse of Equation (3.1). However, as 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 [9] including chromatic aberration significantly improves the performance of models that are trained on fully synthesized data. Hence, we hypothesizes this will also help for our work.

Similarly to [9], chromatic aberration is applied by scaling the locations of the green channel, as well as applying translations on all channels. The model can be implemented as affine transformation of the pixel locations for each channel:

$$f(x_C, y_C) = \begin{pmatrix} S & 0 & t_x \\ 0 & S & t_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_C \\ y_C \\ 1 \end{pmatrix} \tag{3.2}$$

Where $C$ is one colour channel of the image.

**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 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 $\sigma_x$ and $\sigma_y$ are the variance in direction x and y, used to model directional (motion) blur and $\mu_x$, $\mu_y$ are at the kernel center.

**Exposure.** Exposure is the time the sensor records light in order to create an image. Over- and Underexposure are caused when this time is too short or too long, leading to too dark or too bright images.

Cameras typically have Autoexposure-functionality which adapts the exposure time depending on light conditions. However, the adoption is not instant, sudden light changes can lead to over- or underexposure. This particularly applies during a fast flight. Hence, we hypothesize incorporating the effect in the data generation will improve the performance of the trained model. The effect is modelled following [9]:

$$f(S) = \frac{255}{1 + e^{-AS}} \tag{3.4}$$

whats a

where $A$ is a constant term for contrast and $S$ the exposure. The image can be re-exposed with:

$$I' = f(S + \Delta S) \tag{3.5}$$

where $S$ is obtained from :

$$S = f^{-1}(I) \tag{3.6}$$

An example for overexposure is displayed in Figure 3.8.

**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 a scaling in HSV space. The objects in the real world dataset have a slightly different shape and different colour to the 3D-models of the data generation tool. We hypothesize including variations in HSV space will improve the performance on the real world dataset.

The variations are including with:

$$I^* = f(I) = SI \tag{3.7}$$

Where **S** is a 3D-vector where each element stems from a uniform distribution. The distributions are:

## 3.2 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 **??**. 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 fixed set of bounding 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 $S_n$. 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.

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 and to a mapping from the input image of $I_H x I_W x I_D$ to $5 * B$ nodes that encode $B = \sum_{i=0}^{G} S_i^h * S_i^w * a_i$ boxes. Where $a_i$ is the number of anchor boxes in grid $i$. 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

This mapping is implemented with a CNNs. Thereby the dimension of the network output corresponds to $S_*^w x S_*^h x 5 \cdot a_i$.

With *YoloV3* the *TinyYoloV3* network was released that is 10 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. Furthermore, the spatial resolution of 13x13 is still fine for objects that almost cover the whole image. Therefore the architecture is extended by an additional pooling and output layer.

The network architecture is referred to as *SmallYoloV3* and displayed in Figure 3.9 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 *TinyYoloV3*architecture.

vgg

update picture

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. Its architecture is displayed in Figure 3.9 and follows a similar pattern as *SmallYoloV3*. However, it contains much more parameters.

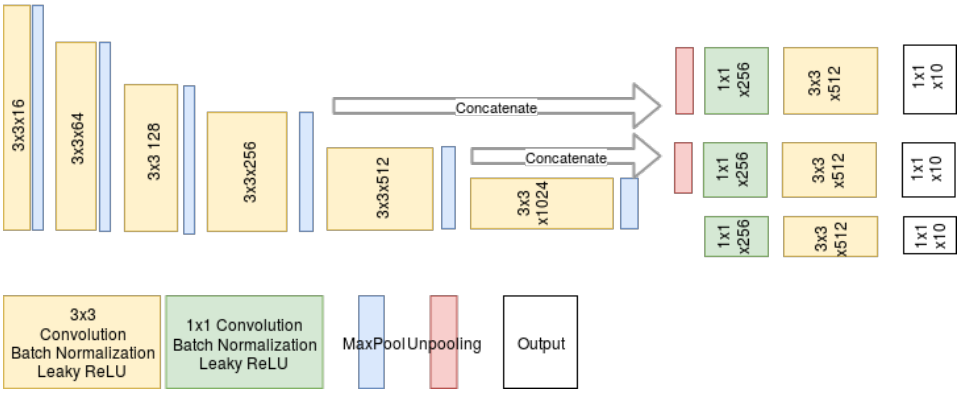


Figure 3.9: *SmallYoloV3*. In the common part the spatial resolution decreases each layer down to 26x26, while the width increases from 16 to 512. From layer 5 three branches focus on objects corresponding to different scales.

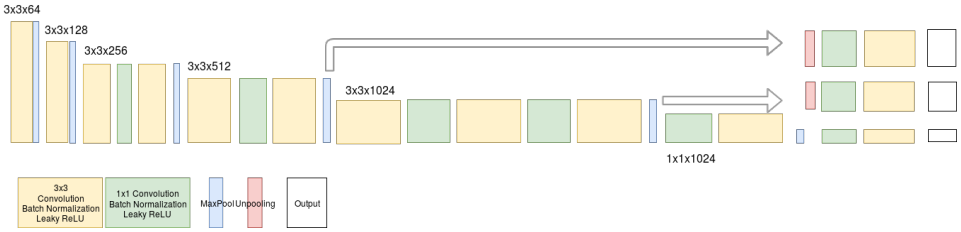


Figure 3.10: *VGGYoloV3*. A *YoloV3* architecture based on *VGG-19*. Similar to *SmallYoloV3* the width is increased as the network gets deeper and spatial resolution decreases, while three output grids focus on different object scales. The network contains additional layers in the different stages. The bottleneck layers add flexibility without adding too much computations.

### 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 derivable function and the CNN can be trained with backpropagation.

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:

$$b_x = \sigma(\hat{x}_{i,j,k}) + g^x_{i,j} \quad b_y = \sigma(\hat{y}_{i,j,k}) + g^y_{i,j} \quad b_w = e^{\hat{w}_{i,j,k}} \cdot p^w_{i,j,k} \quad b_h = e^{\hat{h}_{i,j,k}} \cdot p^h_{i,j,k} \tag{3.8}$$

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; $\sigma$ is the sigmoid-function.

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 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 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 label the training goal can be formulated. The loss needs to capture the localization and the classification goal. 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 $\lambda_{obj}$ and $\lambda_{noobj}$ compensate for this class imbalance. Furthermore, $\lambda_{loc}$ trades-off the localization goal and the classification goal. The abstract training loss is summarized in:

$$\mathscr{L} = \lambda_{loc}\mathscr{L}_{loc} + \lambda_{obj}\mathscr{L}_{obj} + \lambda_{noobj}\mathscr{L}_{noobj} + \lambda_{class}\mathscr{L}_{class} \tag{3.9}$$

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 is used by *YoloV3* is the logarithmic loss which can be formulated as follows:

$$\mathscr{L}_{log} = -(o_{ij}\log(\hat{o}_{ijk}) + (1 - o_{ij})\log(1 - \hat{o}_{ijk})) \tag{3.10}$$

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:

$$\mathscr{L}_{obj} = \sum_{i=0}^{G}\sum_{j=0}^{S_i^2}\sum_{k=0}^{B_i} \mathbb{1}^{obj}_{ijk}(-(c_{ijk}\log(\hat{c}_{ijk}) + (1 - c_{ijk})\log(1 - \hat{c}_{ijk}))) \tag{3.11}$$

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:

$$\mathscr{L}_{loc} = \sum_{i=0}^{G} \sum_{j=0}^{S_i^2} \sum_{k=0}^{B_i} \mathbb{1}_{ijk}^{obj} [(x_{ijk} - \hat{x}_{ijk})^2 + (y_{ijk} - \hat{y}_{ijk})^2 + (\sqrt{w_{ijk}} - \sqrt{\hat{w}_{ijk}})^2 + (\sqrt{h_{ijk}} - \sqrt{\hat{h}_{ijk}})^2] \tag{3.12}$$

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:

$$\min_{w} \mathscr{L}(f(I, w), y) \tag{3.13}$$

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*[**?** ], 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. The three moments are weighted with $\alpha$,$\beta_1$ and $\beta_2$, where $\alpha$ is also referred to as learning rate. This work uses the parameters recommended in the initial public publication of *Adam*[**?** ]:

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

An optimization process such as the training of a CNNs depends on the initialization of its parameters. **?** [**?** ] 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* [2] with *tensorflow* [3] 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 ground. The samples stem

Figure 3.11: Examples of the three test domains. From left to right: *Basement*, *Cyberzoo* and *Hallway*

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 3.11.

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 farer 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. External light sources are covered 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. *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 Synthetic Test Set

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 a synthetic test set is created by simulating a flight through the *IROS*-environment. 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 arrangement of racing gates is based on the race court of the Autonomous Drone Race at IROS 2018. This test set contains 550 images with a total of 1361objects.

recount with filtered

# Chapter 4

# Experiments

This chapter introduces the line of experiments taken out in this work and their intermediate conclusions. Initially, basic experiments about the detection of EWFOs are conducted. In further steps the 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

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

As training a neural network is a computationally intense process common practice is to use 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. Therefore 0.1 % of the training samples are used as validation set. The training is stopped when the validation error converges; that is when it does not decrease for more than $1e^{-08}$ in 3 epochs.

The detector is tested by inferring the network on a given test set and calculate the metrics described in Chapter 2. While this gives a good estimation about the overall performance of a network it can be required to investigate the results in greater detail. It can be important to now how the detector deals with certain view points for example. In order to perform this evaluation predictions and true labels are assigned on bins based on certain conditions e.g. the bounding box size. Subsequently the performance is evaluated only 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 prediction is of size 10 a predicted label of 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 training algorithm 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 Empty 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 EWFOs do depend more on 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. This is compared to a more complex object such as the cat image.

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

### 4.2.1 Training Set

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. On these training sets the two architectures illustrated in **??** *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.


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*).

| Trained/Tested | EWFO | Sign | Cats |
|---|---|---|---|
| EWFO | 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 |
| EWFO 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. 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.

### 4.2.2 Test Set

For each object a test set of 200 samples is created within the *Dark*-Environment, as well as the *IROS*-Environment. Hence, in total there are 6 test sets with 150 samples each. Similar to the training set the view points are limited to frontally facing the object at various distances.

### 4.2.3 Results

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 almost to zero. A similar observation can be made for the *Sign*-object. In both cases the performance can not really be improved by using a deeper network.

For detecting the *Gate*-object, the lowest performance 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.

Table 4.2 shows the change in performance when the trained detectors are applied 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.

| 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. The most severe drop can be seen at the *Cats*-object. The drop for EWFOs is comparable to the *Sign*-object

### 4.2.4 Discussion

It can be seen how the detector exploits the added structure in the filled objects. 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!**s (**Gate!**s)-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.

### 4.2.5 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 the detector can use. This hypothesis can be confirmed as when adding structure inside the empty part the performance gest significantly better.

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.

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.

## 4.3 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 placing the object in front of backgrounds of different images. 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. 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. The detectors are evaluated on the test sets created in the previous section.

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 to not fall correctly. With the simulated environment the general scene looks more realistic although the background is synthetic.

### 4.3.1 Training Set

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

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

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 .

VOC

### 4.3.2 Results

Figure 4.3 display the results. It can be seen how using uniformly coloured backgrounds achieve almost no detection. Only on the simulated data a few objects could be detected.

In contrast using fully synthesized data achieves the best performance on the simulated data for low and high quality detections. It can be seen that almost all detections have an IoU of more than 60% and are thus good

| Trained/Tested | Gate | Sign | Cats |
|---|---|---|---|
| Single Background | 0.24 +- 0.06 | 0.02 +- 0.02 | 0.23 +- 0.06 |
| Simulated Background | 0.74 +- 0.05 | 0.53 +- 0.21 | 0.71 +- 0.07 |
| VOC Background | 0.30 +- 0.22 | 0.45 +- 0.19 | 0.33 +- 0.21 |

Table 4.3: Performance in the *IROS* environment when adding more variance in the background.

quality. On the real data the performance in low quality detections is comparable to the performance on the simulated data. However, in hiqh quality detections the performance drops significantly to an equal level of using real backgrounds.

Using real backgrounds performs poorer than using a fully synthesized environment in almost every case. However, for higher quality detections the performance is competitive to using a fully synthesized environment. Combining both methods does achieve at most the performance of using fully synthesized data. However, in most cases the performance is worse.

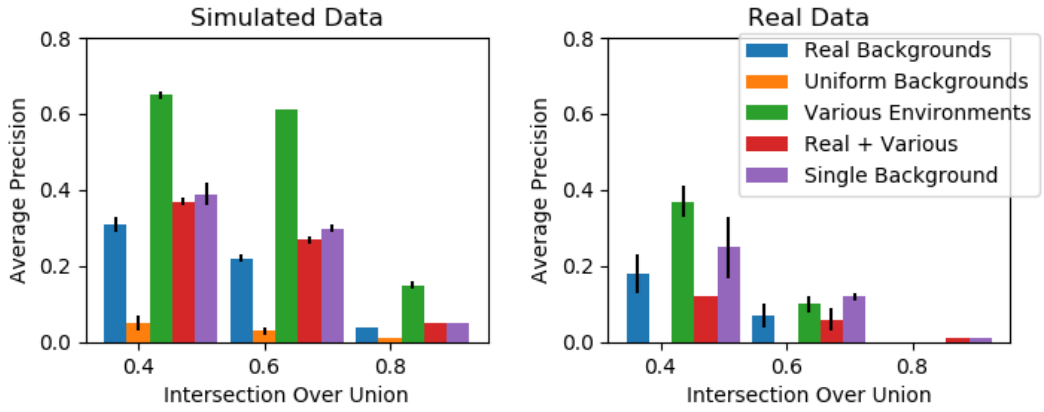


Figure 4.3: Results of different ways to generate context. Synthesizing more parts of the environment improves performance on both test sets. However, this only holds for weakly localized predictions.

### 4.3.3 Discussion

The results meet our hypothesis that using only uniformly coloured backgrounds is not enough to train an Object Detector for EWFO. Providing more variance in the background is crucial to improve performance.

Despite having not seen a single real background, the model trained on fully synthesized data achieves the best performance on the real data. Even a model trained in a single virtual environment achieves competitive performance. Hence, it seems that synthesizing correct geometric/physical properties is more important than providing a large variance in background. However, the margin gets small for higher quality detections. Thus, the simulation of geometric/physical properties does not help for good localization.

### 4.3.4 Conclusion

We investigated the role context plays for the detection of EWFOs on MAVs. We can conclude that providing some background is crucial for the detection. Otherwise the detector performs poorly in simulation as well as the real world. Furthermore, synthesizing the whole environment benefits the detection on real images. However, most of the additional detections have not very accurate bounding boxes.

## 4.4 Transferring the detector to an MAV race

Until now the conducted experiments where 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 synthetic test set described in Section 3.3.

Due to the challenges in this dataset we hypothesize that the detector trained so far will perform poorly on this dataset. In order to be able to handle overlapping objects in difficult angles, such situations should be included in the training set. In order to obtain such views different methods are compared.

A straightforward way is placing the camera randomly (within some margin) in order to cover a large variation of views on the object. That way the network can learn a general object representation and hopefully detect unseen objects from different view points. However, random placement might not resemble the real world sufficiently. An MAV does not appear at random places within a scene, especially not when it follows a racing track. We examine this by simulating a flight through a race court using AirSim's MAV model and analyzing the relative object poses. We compare these to the relative poses obtained when placing the camera randomly using the distributions from **??**.

Figure 4.4 shows the distribution of bounding boxes when created with random camera placement and when following a racing track. It can be seen how, when following the race track most of the objects are centered 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. This can also be seen in Figure 4.5 where a 2D histogram of the yaw angle and distance with respect to the camera is displayed. Thereby 0 corresponds to facing the object frontally, 180 degrees facing the object from the back. It is apparent how the random placement covers a much larger range of relative angles, while in the racing track certain angles do not appear at all. Even more importantly the largest bins of the racing track is an angle of 0 and a distance between 0m and 4m. These bins are almost not present when placing the camera randomly. This is because close to the camera the field of view is small, while the area of the object faced frontally is big. Hence, the probability of an object ending up at this specific location is relatively low. Furthermore, when placing the camera randomly there are no samples further away than 8m. This is because in the race track the camera traverses the room from one end - where it can see almost all gates - to another. The probability that the randomly placed camera ends up in a similar position is relatively low.

The filters of CNNs are translation invariant by design but cannot inherently handle variation in rotation and scale. We hypothesize that the generation of samples with only one of the two methods can miss important object appearances. Random placement does not cover appearances that are typical for a autonomous drone race. On the other hand, only training on racing tracks might lead to a bias towards the created courses.

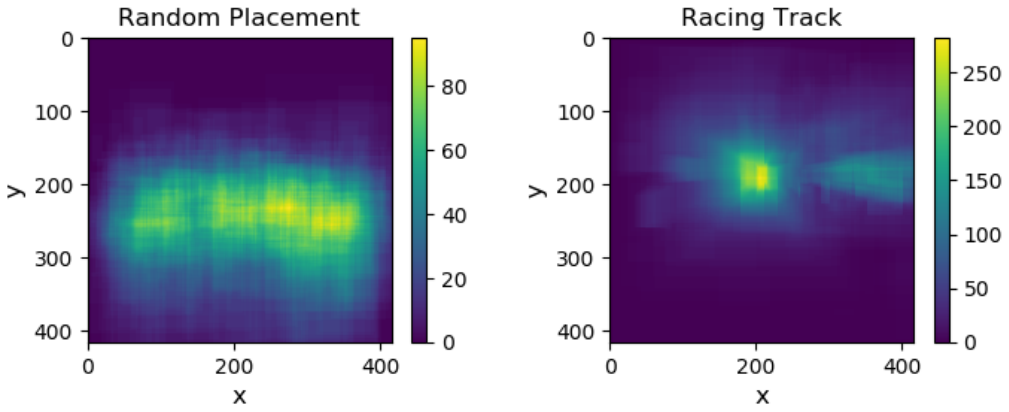


Figure 4.4: Object appearances when generating samples with random poses (left) and during a MAV flight. During the flight the object appears mostly centered on the horizontal line.

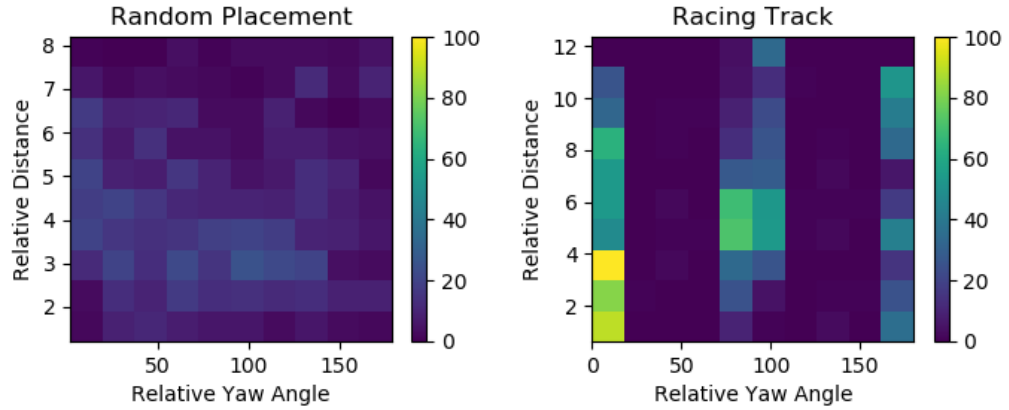


Figure 4.5: Histogram of object occurrences in yaw angle and distance relative to the camera. The random placement does rarely cover facing the object closely and frontally.

### 4.4.1 Training Set

### 4.4.2 Results

### 4.4.3 Discussion

### 4.4.4 Conclusion

## 4.5 Transferring the detector to the real world

In literature [25, 35, 40, 56] the application of image augmentation is a common tool to improve the detection performance. The experiments in [9] show how the incorporation of sensor effects particularly improves the performance of models learned on fully synthesized data. In the MAV domain sensor and lens effects have a significant influence on the obtained sample. Hence, we hypothesize that the incorporation of these effects will improve the performance of the trained models. A total of five image augmentation techniques are investigated.

### 4.5.1 Results

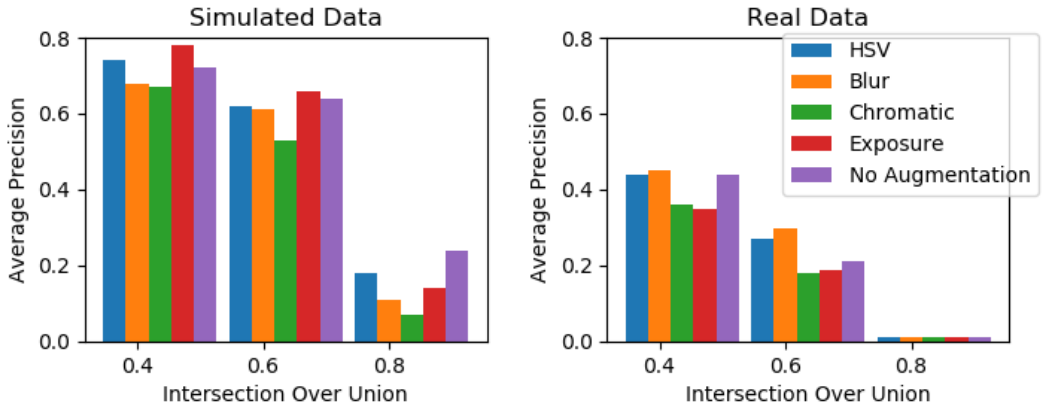


Figure 4.6: Performance in terms of Average Precision for different methods of image augmentation.

Figure 4.6 shows the results in terms of average precision in the training domain. On the simulated data variations in exposure improve the performance of low quality predictions slightly compared to using no augmentation. However, using no data augmentation achieves the best performance for high quality detections. The other effects have only minor influence.

On the real data variations in HSV space as well as blurring improves the results compared to not using data augmentation. Incorporating chromatic aberration and variations in exposure lead to a deterioration in performance.

### 4.5.2 Discussion

In simulation the data augmentation has only a minor effect on low quality predictions, while the performance in high quality predictions decreases. As most effects are not really present in the test set this confirms

that the models can learn a robust representation. Despite having more noise in the training data, the performance on the test set stays the same. However, the added noise leads to a lower quality in the predicted bounding boxes.

On the real data there is a stronger effect measurable. Variation in HSV and image blurring increase the performance compare to not using data augmentation. This meets our hypothesis that variations in HSV help to achieve a model that is more robust against changes in colour. Blur is one effect that can clearly be seen in the real world test set. Including this effect in the training process helped the model to perform better in the real world. Chromatic Aberration led to a significant improvement in [9] however, we cannot confirm these results. It is possible that our camera suffers only little from chromatic aberration and thus including the effect in the training does not further help the prediction. The same holds for variations in exposure.

### 4.5.3 Conclusion

We investigated whether including sensor effects present in the target domain in the data generation process can improve the detection. Therefore we modelled several effects that were observed when working with the camera or that improved the detection in experiments conducted in literature. Finally, image blurring improved the detection. Other effects led to a deterioration in performance.

We also investigated image augmentation by adding variations in HSV-space to evaluated whether this improves the robustness of the model, improving the performance on the real world dataset. We can confirm this hypothesis and conclude that this image augmentation should be part of the training process.

## 4.6 Deploying the detector on a MAV

In the application of the detection of EWFOs on a MAVs detection accuracy is only one important metric. 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 ot 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 performance-accuracy trade-off is studied and an experiment in a real world flight is conducted.

As explained in **??** 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 4.7. 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.

*TinyYoloV3* is optimized to detect solid feature rich objects of multiple classes with a low computational budget. In order to sufficiently represent and distinguish such objects many weights are required. Hence, the network contains 9 layers and a final width of 1024. In contrast, the features of EWFOs are relatively simple hence less weights should be required. Therefore we hypothesize a thinner network should be able to learn the task equally well while being computationally more efficient.

The receptive field of *TinyYoloV3* is 223 pixels which does not cover the full input image. For solid objects this is of minor impact as even if the network only sees an object part it can still learn to recognize it. However, this does not apply for EWFOs which are empty and do not contain any information in the object centre. Instead

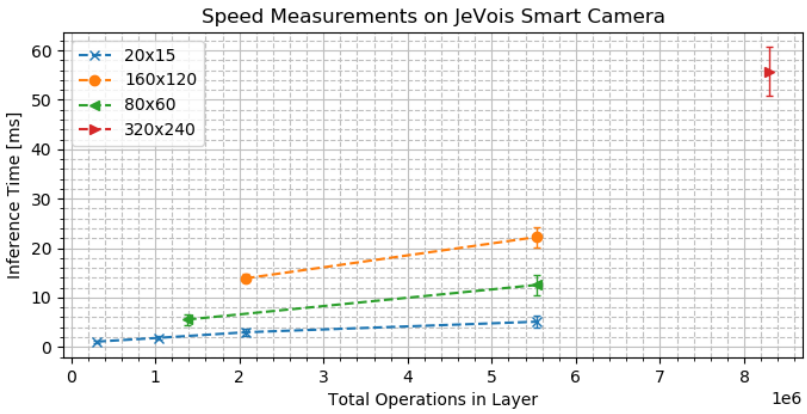


Figure 4.7: Inference Time of different layers on the JeVois. Each sample corresponds to the inference in a single layer. Dashed lines connect samples at the same resolution. It can be seen how an operation at a higher spatial resolution is significantly slower.

it can confuse an output layer that is assigned responsible to detect such an object. Therefore we hypothesize that more layers should improve the performance for larger objects. However, the c

Yet, the complexity of large objects does not increase. Hence, we hypothesize that if the receptive field is large enough, performance for larger objects.

Another parameter are network width and depth. The same condition holds for the number of layers. However, more layers also increase the receptive field. Hence, we expect the performance to increase as long as the receptive field does not fully capture the image.

In order to evaluate our hypothesis we perform an architecture evaluation by varying the number of layers filters and the receptive field. Before training the networks we tune the anchor box dimensions by performing a k-means clustering on t

Initially an architecture evaluation is performed. Therefore different architectures are trained and evaluated in terms of **ap60!** (**ap60!**). As an architecture contains many parameters we can not simply vary each of them independently. Hence, three experiments are performed iteratively. We first describe them on a high level basis before explaining the exact changes. In a first step the width is decreased until a drop in performance is noticeable. In a second step the architecture with the lowest width but without performance drop is chosen and the depth is varied. Based on the results of this step we create a final model that combines the gained insights and tune the anchor boxes.

The scheme in which the architecture is changed is visualized in **??**. The width of the baseline model is decreased stepwise by a factor of two. When decreasing depth only convolutional layers are removed while the pooling layers are kept such that the spatial resolution stays the same. When increasing depth 2 layers are inserted stepwise at the end of both branches. The results show how depth is mainly relevant to detect objects of larger scale. Hence, the final model consist of 5 common layers, 15 layers on the branch to detect larger objects and 2 layers on the branch to detect small objects.

This means most speed can be gained when reducing the number of computations in the early layers where the spatial resolution is high. In earlier experiments it could be seen that already a small amount of filters is enough to detect EWFOs. However, even evaluating 4 kernels at a resolution of 320x240 already takes 55 ms (Figure 4.7 red triangle). A total network of that size would require more than 200 ms and is thus too slow to be deployed in the control loop.

Current research mostly addresses to reduce 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.

## 4.7 Experiments

In order to answer this question we measure the inference time of different model architectures. This enables us to investigate the bottlenecks and thus optimize the model architecture. The JeVois Smart Camera uses an aspect ratio of 4:3. Therefore we change the network resolution accordingly. The JeVois Smart Camera has only limited memory available. Using a network architecture that goes beyond the memory simply results in a system crash. For example our baseline network runs performs one network evaluation in 700 ms. This is at a resolution of 160x120.

In order to evaluate our hypotheses the network is trained with different architectures. Subsequently, performance and inference time on the JeVois are measured.

The JeVois supports aspect ratios of 4:3 and resolutions of 160x120, 320x240 and 640x480. Although, FCN do not depend on the image resolution, 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 to 160x160 we remove the last pooling layer such that the output grid stays at 20,20 or 10,10 respectively.

## 4.8 Results

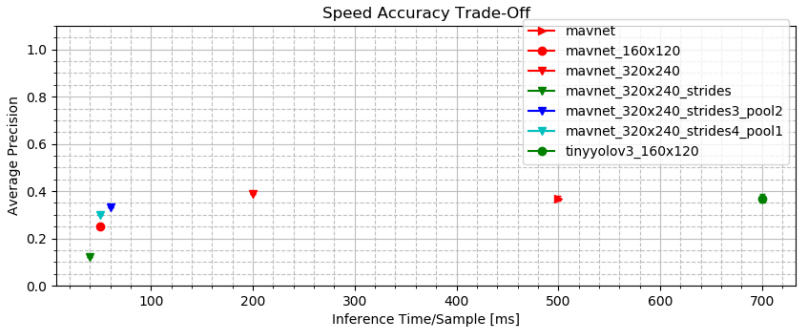


Figure 4.8: Inference Time of different layers on the JeVois. Each sample corresponds to a single layer. On the x-axis the total number of multiplications in that layer is displayed. It can be seen how an operation at a higher spatial resolution is significantly slower.

Due to their computational complexity deploying a CNNs on mobile devices is a challenging task

Figure 4.9 shows the performance for thinner and wider networks. It is apparent how the performance only undergoes slight variations despite reducing the total number of weights by a factor 1000.

retrain one in the middle to get variance, it should be more linear

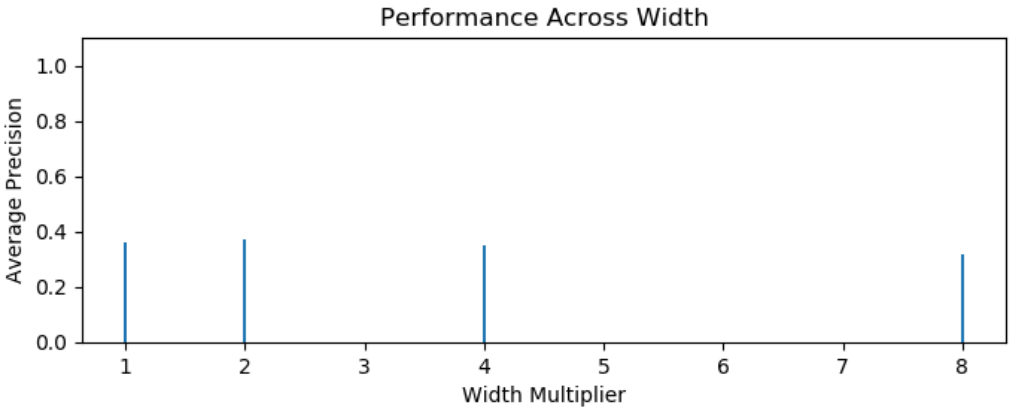


Figure 4.9: Performance in simulation when varying the amount of filters per layer. Starting from the baseline architecture with approximately 9 Mio. weights, the amount of filters per layer are decreased stepwise by a factor of 2. Only minor effects on performance can be seen, despite reducing the flexibility of the model.

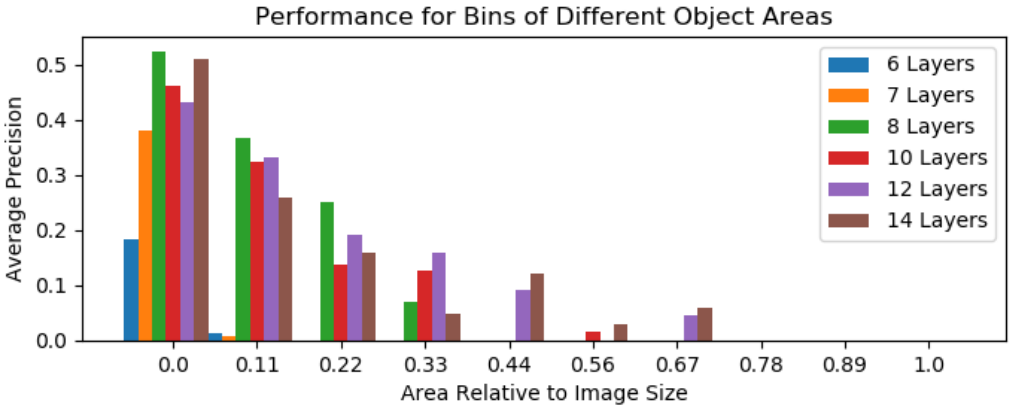


Figure 4.10: Performance in simulation of models with varying depth and for bins of different size. It can be seen that the performance for larger objects increases with the amount of layers.

Figure 4.12 shows the distribution of object sizes in the bins used for evaluation. It can be seen that most objects in the test set are further away.

put some examples to show what each size actuall means

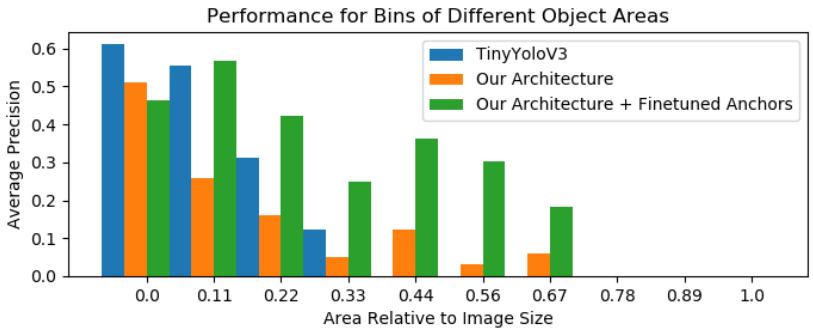


Figure 4.11: Architectural Changes when varying the depth. The upper graph shows in which order layers are removed. The lower graph shows how layers are added. Depth is increased by inserting two layers on each branch (green).

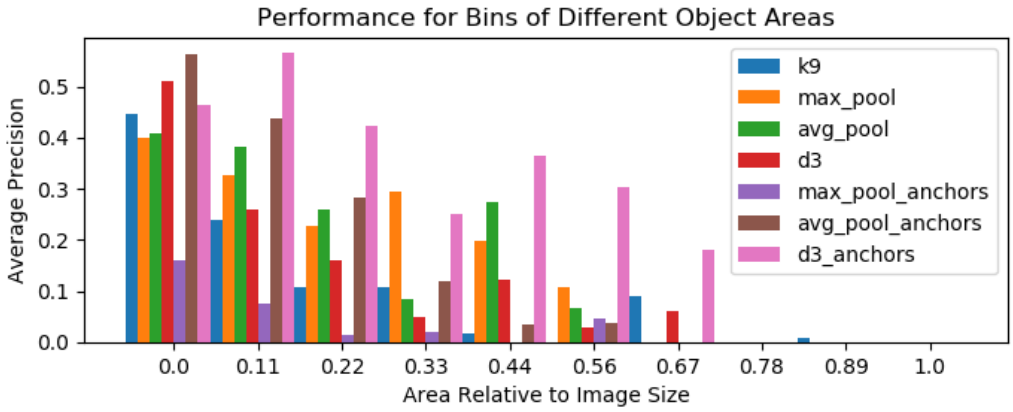


Figure 4.12: Label distribution in bins of different object size. As the field of view is larger for objects further away, the proportion of small objects is higher.

### 4.8.1 Discussion

The overall performance is only slightly affected when reducing the number of weights in terms of width and height. We can assume that this is because the objects we investigate are of quite simple structure. Intuitively the features to be considered are color, lines and edges in certain combinations. Hence, it seems logical that only a few filters are necessary to represent these shapes.

The performance in terms of different object sizes varies significantly for models with varying depth. Only deeper networks are able to detect larger objects. However, the complexity of the object does not increase for closer objects. In contrary the closer the objects are, the less context is visible. A very close object consists "only" of an orange square. Hence, it is unlikely that increased flexibility is the reason for the increase in performance.

Instead it is more likely that the increased receptive field is the reason for the improved performance. In fact only the network with 14 layers has a receptive field of 414 an thus covers the whole image. Yet even this network cannot detect the largest objects. Thus the receptive field can not be the only reason for the lower performance on larger objects.

The current structure combines features distributed across space in a pyramid fashion. So 3-by-3 convolutions are performed layer by layer such until the whole image is covered. For EWFOs many of these steps are unnecessary as the objects are empty and the network should learn to ignore the empty part in the centre. It is possible that this structure gets confused by the parts that are present in the image centre.

### 4.8.2 Conclusion

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 confirm this hypothesis as we see that the width can be decreased by a factor of 10 without loosing performance. This leads to a reduction of weights by a factor of 1000.

Furthermore, we investigated how depth affects the performance of the model. We hypothesized that a shallow network should be able to detect the object as it only consists of relatively simple features. We can see how depth is required in order to cover the whole input image. Hence, we cannot fully conclude whether depth is required for the increased flexibility or simply due to the receptive field.

# Chapter 5

# Discussion

In this work we investigated a learning based approach for the detection of EWFOs on MAVs. The research was motivated by several drawbacks of the manual crafted detection method: SnakeGate. The aim was to investigate whether a learning based Object Detector is more robust against, occlusion, out of view and changes in illumination. Our results show how the deep learning based detector outperforms SnakeGate especially in the cases of occlusion and out of view. The deep learning based detector can detect objects even when only 30% of the object is visible or when backlight leads to strong changes in colour appearance.

Yet the deep learning based detector introduces its own drawbacks. For example the bounding boxes are generally less accurate than of SnakeGate. We assume that the reason is the general structure of Yolo. CNNs collect "shallow" low level image features and combine them to a more deep representation of the image. Pooling layers reduce the spatial resolution to keep the number of computations tracktable, however thereby also local information about the features gets lost. In the final layers information is present which features are present in that part of the image, however it is not clear where they come from. Future work should address how to reflect back from the deep representation to the low level features.

A main argument for deep learning detectors is the fact that they can be trained on any kind of object and thus make feature engineering unnecessary. Furthermore, the learned representation are often much better than the features crafted manually. A drawback is the computational requirements of CNNs. For this work no training data was available and the computational requirements were limited. By implementing a data generation pipeline and refining the architecture we could show how the deep learning based method outperforms the manually crafted one. However, this process was a vast amount of work and yet we have a severe drop in peformance between the simulated and the real data. Hence, we have to wonder whether it would have not been more fruitful to invest time in feature engineering based on real data. It is questionable whether this is method would still perform better than the deep learning method. And yes for a new object it would required to be changed. But it would lead to a more transparent detector.

deep detector can be better extended but we loose track of whats going on, deep learning makes sense with the appropriate hardware and the appropriate data

# Chapter 6

# Conclusion

# Appendix A

# Appendix

## A.1 Data Generation

This section describes how the ground truth labels are obtained when generating data.

### A.1.1 Camera Model

The camera itself is modelled with the pinhole camera model that contains six parameters:

1. Focal length $f_x, f_y$
2. Central point $c_x, c_y$
3. Sensor skew $s_x, s_y$

The model can be summarized in the intrinsic camera matrix $C$:

$$C = \begin{bmatrix} \frac{fx}{s_x} & 0 & cx \\ 0 & \frac{f_y}{s_y} & cy \\ 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

The model projects 3D coordinates $X$ to the image plane following:

$$X' = CX \tag{A.2}$$

Where $X$ are points described in homogeneous coordinates originating from the cameras position.

For data generation several tools are used. 3D Models for the Target Object (TO) are taken from ... OpenGl is used to render these objects and replace the background with a particular image. The Unreal Engine and AirSim are used to render a full scene.

Within the graphic engines, the objects can be placed in 3D space. From the known object shape the surrounding bounding box can be defined in 3D coordinates. Using the pinhole camera model described in Equation (A.1) the corresponding 2D coordinates on the image plane can be obtained with the following:

The camera position is described by its rotation matrix $R$ and its translation vector $t$. Where $R$ is obtained from the Euler angles with:

$$R =$$

The 3D coordinates of the objects relative to the camera can be obtained by applying the inverse transformation $T$ of $R$ and $t$ with:

$$t' = R \times t$$

$$T = R^{-1} | -t'$$

$$X_{Cam} = T \times X$$

The full projection can then be expressed by the matrix multiplication:

$$X' = C \times T \times X$$

Where $C$ is the intrinsic camera matrix defined in Equation (A.1).

# Bibliography


[1] Microsoft/AirSim: Open source simulator for autonomous vehicles built on Unreal Engine / Unity, from Microsoft AI & Research. URL https://github.com/Microsoft/AirSim.

[2] Keras Documentation. URL https://keras.io/.

[3] TensorFlow. URL https://www.tensorflow.org/.

[4] What is Unreal Engine 4. URL https://www.unrealengine.com/en-US/what-is-unreal-engine-4.

[5] 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.

[6] 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/.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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/.

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

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

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

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. aug 2016. URL http://arxiv.org/abs/1608.06993.

[28] 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.

[29] 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/.

[30] 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.

[31] 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/.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

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

[37] 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.

[38] 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.

[39] 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/.

[40] 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.

[41] 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.

[42] 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/.

[43] 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.

[44] 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.

[45] 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/.

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

[47] 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.

[48] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson Cvap. CNN Features off-the-shelf: an Astounding Baseline for Recognition. URL https://arxiv.org/pdf/1403.6382.pdf.

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

[50] 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.

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

[52] 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.

[53] 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/.

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

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] Subarna Tripathi, Gokce Dane, Byeongkeun Kang, Vasudev Bhaskaran, and Truong Nguyen. LCDet: Low-Complexity Fully-Convolutional Neural Networks for Object Detection in Embedded Systems. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, volume 2017-July, pages 411–420, 2017. ISBN 9781538607336. doi: 10.1109/CVPRW.2017.56. URL https://vision.cornell.edu/se3/wp-content/uploads/2017/07/LCDet{_}CVPRW.pdf.

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

[61] 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.

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

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

[64] 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.

[65] 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.

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

[67] 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.