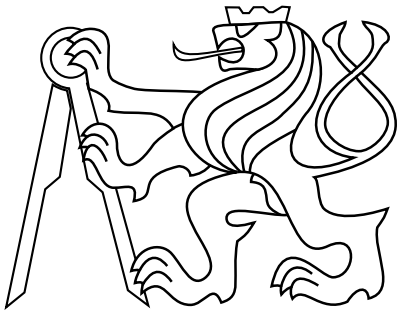
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science



Gimbal camera detection and tracking for a small autonomous UAV

Master's Thesis

Bc. Michal Hloušek

Prague, January 2023

Study programme: Open Informatics
Branch of study: Artificial Intelligence

Supervisor: Ing. Matouš Vrba

Acknowledgments

I am incredibly grateful to my supervisor for providing me with the opportunity to work on an interesting and challenging problem, and for offering valuable advice and guidance throughout the process.

I would also like to thank my family and girlfriend for their unwavering support during my studies. Their love has been a constant source of motivation and strength, and I am deeply grateful for their support.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hloušek** Jméno: **Michal** Osobní číslo: **475896**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Umělá inteligence**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Detekce a sledování cíle pomocí gimbalové kamery na palubě UAV

Název diplomové práce anglicky:

Gimbal camera detection and tracking for a small autonomous UAV

Pokyny pro vypracování:

 Develop an algorithm for detection and tracking of a target using a gimballed RGB camera placed onboard a flying UAV [1-5].
 Develop an algorithm to control the gimbal orientation and plan the trajectory of the UAV based on the predicted trajectory of the target to ensure that the target remains within the camera‘s field of view [6].
 Implement the whole solution in ROS to run online, onboard the UAV, under the MRS UAV system.
 Evaluate the precision, robustness, range and general performance of your solution in simulations as well as a real-world experiment.
 The resulting system should be suitable for deployment in the task of autonomous tracking of high-speed targets for the purpose of physical interaction or collision avoidance (see mrs.felk.cvut.cz/mbzirc2020 and mrs.felk.cvut.cz/projects/eagle-one) and in the task of autonomous monitoring of workers in high-risk environments for safety purposes (see mrs.felk.cvut.cz/projects/aerial-core).

Seznam doporučené literatury:

X. Zhou, D. Wang, and P. Krähenbühl, "Objects as Points," arXiv 1904.07850, 2019.
V. Walter, M. Vrba, M. Saska, "Automatic generation of training datasets for machine learning-based visual relative localization of micro-scale UAVs," ICRA, 2020.
M. Vrba, and M. Saska, "Marker-Less Micro Aerial Vehicle Detection and Localization Using Convolutional Neural Networks," RA-L 5(2), pp. 2459-2466, 2020.
D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," ECCV, 2016.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Matouš Vrba Multirobotické systémy FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **02.02.2022** Termín odevzdání diplomové práce: **10.01.2023**

Platnost zadání diplomové práce: **30.09.2023**

Ing. Matouš Vrba podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____ _____

Datum převzetí zadání Podpis studenta

Abstract

In this thesis, a solution to the problem of drone tracking using a gimballed RGB camera mounted on a flying UAV is proposed, implemented and tested both in simulation and real-world experiments. The proposed system uses a combination of machine learning models for visual detection and tracking to create an online estimate of the target drone's 3D position. Two large drone datasets from various environments were used to train and evaluate the models. The YOLO5 neural network architecture and the MOSSE tracking algorithm were selected for their balance of speed and accuracy. Genetic programming was used to further optimize the hyperparameters of the chosen architecture achieving satisfactory online performance in a computationally constrained environment of the on-board computer. The resulting estimates from the detection and tracking pipeline are denoised using a Kalman filter and used as an input to PID controllers to maintain an approximately constant distance and a zero altitude difference between the drones.

Keywords Unmanned Aerial Vehicles, Automatic Control, Visual Detection, Tracking, Gimbal

Abstrakt

V této práci bylo navrženo, implementováno a otestováno řešení problému sledování dronů pomocí RGB kamery umístěné na gimbalu připevněném k UAV v simulaci i experimentech v reálném světě. Navržený systém používá kombinaci modelů strojového učení pro vizuální detekci a sledování k odhadu 3D polohy sledovaného dronu. Modely jsou natrénovány a vyhodnoceny pomocí dvou velkých souborů dat pořízených za letu v různorodých prostředích. Z výsledků měření bylo zjištěno, že architektura neuronové sítě YOLO5 v kombinaci s algoritmem MOSSE nabízí optimální kompromis mezi rychlostí a přesností pro daný problém. K optimalizaci hyperparametrů zvolené architektury byla využita metoda genetického programování, čímž bylo dosaženo uspokojivé přesnosti ve výpočetně omezeném prostředí palubního počítače. U výsledných odhadů je odstraněna část šumu pomocí Kalmanova filtru a výstup je předán PID regulátorům určujícím rychlost potřebnou pro udržení konstantní vzdálenosti a relativní výšky letu dronů.

Klíčová slova Bezpilotní Prostředky, Automatické Řízení, Vizuální Detekce, Sledování, Gimbal

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Figure 1.1: Eagle.One drone hunting platform[1].

Unmanned aerial vehicles (UAVs), also known as drones, had traditionally been used almost exclusively by militaries for surveillance and combat operations [1]. However, drones are increasingly being adopted for consumer use, leading to concerns about their potential impact on security, as well as the safety of people, private property, and critical infrastructure [2]. As UAVs are capable of flying over critical facilities such as airports, stadiums, prisons, and military bases to name a few, restrictions on their use are being put in place to prevent both accidental and deliberate damage.

Anti-drone technology is catching up as there are now autonomous drones designed to search and intercept unauthorized intruders using RADAR and LIDAR sensors to guide the hunter towards its target such as the Eagle.One autonomous aerial interception system[2]. Alternative solutions include the use of high-power laser weapons such as the SkyLock system[3] or drone jamming systems that disrupt the communication between the drone and its operator.

[1] Photo courtesy of `https://www.svetchytre.cz/a/pTggy/eagleone--ceske-zarizeni-odhali-a-zpacifikuje-nebezpecny-dron`.

[2] `https://eagle.one/`

[3] `http://www.itck.co.kr/bbs/download.php`

All these methods require the target to be identified and tracked until the threat is eliminated. Drones often have RGB cameras attached to their frames, which can be employed to detect and track their targets using machine-learned models with no additional expensive hardware required. Another advantage of utilizing a color camera is that human assessment of visible-spectrum detection is more natural than from other electromagnetic sensors. Finally, for real-world deployments, it is critical to be able to differentiate between various flying objects such as a drone, a bird, a kite or a plane, something that can be done efficiently in the visible spectrum using state of the art visual detection methods.

1.2 Objective

The aim of this work is to design, develop and evaluate a fully autonomous system that enables a drone equipped with a monocular RGB camera placed on a stabilized gimbal mechanism to detect, track and follow a target UAV, although any other object whose appearance model is supplied may be tracked using the implemented system. The challenge is for the follower to maintain the target in its line of sight despite the panning and tilting of the camera caused by the highly dynamic nature of drone flight and while being limited by the on-board computer's computational resources. The system is not designed to rely on assumptions about the target drone's equipment, hence it does not consider the use of i.e. ultraviolet markers such as AprilTag [3] or UVDAR [4].

The problem can be viewed as an instance of the leader-follower problem in robotics, where the leader moves along an assigned trajectory and the follower maintains the desired relative distance and orientation.

1.3 Related Works

1.3.1 Visual Drone Detection

The first part of this thesis focuses on the detection of drones flying in diverse environments, often occupying a small portion of images taken by the on-board camera as shown later in Subsection 4.2.1. Consequently, models demonstrating good performance on small object datasets are prioritized for further experiments. The recent research described in [5] carried out an extensive empirical evaluation of several state of the art object detection architectures on a filtered PASCAL VOC 2007 dataset[4] and a small object dataset composed of data from COCO[5] and SUN[6]. YOLOv3 [6] is found to be the most precise of the one stage methods considered in the article. However, when the generally slower two-stage methods are also considered, Faster RCNN is the most accurate. For certain objects and scales, the faster and simpler one stage approach gives similar accuracy as the two stage methods while being up to over 10 times faster (see the VOC_MRA_0.20 subset in [5], where the one stage method YOLOv3 dominates). This shows that for a particular problem, multiple different models using various approaches should be tested to find an optimal performer, especially when speed is a crucial factor in the considered application.

[4] http://host.robots.ox.ac.uk/pascal/VOC/voc2007/
[5] https://cocodataset.org/
[6] https://vision.princeton.edu/projects/2010/SUN/

In a recent work of the authors of the Det-Fly dataset [7], several state of the art deep networks are compared on the task of UAV detection. Grid RCNN and Faster RCNN demonstrate good accuracy and YOLOv3 is deemed an ideal compromise between precision and inference speed, although it uses smaller input image resolution and in general performs worse than its more recent iterations (i.e. YOLO5, YOLOX) according to the results presented in [8].

YOLOv3 is also successfully utilized in paper [9], where the model approaches 90% accuracy on an unpublished dataset for visual drone detection. The authors achieve a processing speed of 30 FPS on the Jetson TX2 embedded computer, although a YOLOv3 model still achieves only 2.6 FPS on an Intel Core i76920HQ CPU [10], which has a comparable performance to the processor used on-board the drone in this thesis (see Section 3.2 for a complete description of the on-board equipment). The camera considered in this thesis is capable of producing images at 24 FPS, so at 2.6 FPS processing rate, much information would be unused. This can be addressed by combining the CNN detector with a fast visual tracking method.

1.3.2 Visual Drone Tracking

In [11], traditional visual tracking algorithms (CSRT [12], MIL [13], MOSSE [14] and KCF [15]) are compared in the context of UAV research. The benchmark uses implementations of the trackers provided by the OpenCV library [16]. The CSRT and MOSSE algorithms achieve the best efficacy of 95.5% and 75.7%, respectively. MOSSE outperforms all the other trackers in terms of average processing time, achieving 365× faster speed than its second-best competitor CSRT. This is in contrast to the results presented in [15], where KCF outperforms MOSSE (73% vs 43% mean precision) while averaging approximately half of MOSSE's FPS (292 vs 615 FPS).

A modern alternative to the traditional visual tracking algorithms is deep visual tracking, in which the appearance model is a deep neural network, for example a convolutional neural network [17] or a recurrent neural network (i.e. RTT [18]). The network can be trained prior to the tracking or online on each incoming frame, or a combination of both. These methods tend to be slower than their traditional counterparts but offer better tracking robustness and accuracy [19].

1.3.3 Autonomous Drone Pursuit

In [20], the FollowMe software is proposed, which allows a drone to follow any object selected by the user from an image taken by an on-board camera. The software uses the TLD tracking algorithm [21] and adjusts the pursuit speed with a proportional-integral-derivative (PID) controller.

In a recent work by Liu Xuancen et al. [22], a drone platform with a camera mounted on a 3-axis gimbal is presented. Similar to the previous paper [20], the target has to be manually selected and is then tracked using an improved version of the KCF algorithm, which can also adapt to scale changes and occlusions. A Jetson TX2 Module is connected to the drone to improve processing speed and allow KCF to be run online. The proportional navigation guidance law [23] is used to follow the other drone while the gimbal keeps the camera centered on the target.

The paper [24] compares the CNN and depth map based approaches in a real-world drone leader-follower experiments. A conclusion is drawn that the main source of localisation error in the CNN-based approach is imprecise distance estimate of the leader caused by inaccurately predicted size of the output bounding box by the CNN. Despite this, the method is considered suitable for autonomous pursuit and has several advantages over alternative methods, such as a larger maximal usable range.

1.4 Structure of the Thesis

In Chapter 2, several object detection and visual target tracking algorithms used in the rest of the thesis are analyzed and compared. Mathematical models of a camera and its associated lens distortions are introduced and transformations between the coordinate systems of a drone, its gimbal stabilization system and camera are derived.

The architecture of the solution, including the detection and localization packages, gimbal and flight managers and a visualisation tool is outlined in Chapter 3. The utilized hardware platform and software frameworks are also presented.

Chapter 4 explains in detail the process of training and model selection for the target detector. The chosen model is further improved using a genetic algorithm based on the principles of natural selection. The relationship between the confidence of detection and the distance of the target is explored. Finally, traditional tracking algorithms are compared to assess their performance in the low-delay setting.

Chapter 5 and Chapter 6 evaluate the proposed system in simulated experiments using the Gazebo simulator and in the real world.

Chapter 2

Theoretical Background

2.1 Object Detection in Images

Object detection is a technology related to computer vision and image processing that aims to identify and locate objects in an image or video. State-of-the-art object detectors commonly use deep convolutional neural networks (CNNs) as their feature extractors (also known as backbones) and fully connected artificial neural networks (FC ANNs) for object localization and classification [25]. The history of ANNs began with the discovery of the McCulloch-Pitts neuron [26] in 1943, the CNN was introduced in 1980 under the name of neocognitron [27] and training using the backpropagation algorithm [28] was described in 1986. However, it was not until 2012 that the performance of ANN-based methods[1] surpassed traditional detection algorithms such as Viola-Jones detector [29] based on the Adaptive Boosting meta-algorithm [30] or the Deformable Part-based model [31]. According to the original paper of the AlexNet [32], the depth of the model was essential to its success and removing even one layer decreased its performance significantly. It was also able to train on an extensive data set collected from the web, which was not possible when the CNN model was first described due to hardware constraints at that time.

2.1.1 CNN

A convolutional neural network is a specific type of artificial neural network that uses convolutional layers. A convolutional layer solves the parameter explosion of an FC ANN with a large number of features as described e.g. in [33] by exploiting the locality of pixel dependencies using a number of smaller filters convolved across the image to create feature maps used for inference. An assumption is therefore made that pixels close to each other are related and, if grouped together, are more likely to carry semantic information [34], which is reasonable in the domain of images.

Suppose the image X^l of size $n \times m$ is the input to a simple convolutional layer l. If a single $k \times k$ filter H is used (assuming k is even for the sake of notational simplicity), the output Y will have a size of $(n - k + 1) \times (m - k + 1) \times 1$. Each additional filter increases the third dimension by one. The relation between the input and output of the l-th layer of a CNN can be expressed as

$$Y^l_{i,j} = \sum_{u=-k/2}^{k/2-1} \sum_{v=-k/2}^{k/2-1} H_{u,v} X^l_{i-u,j-v}, \tag{2.1}$$

[1] See SuperVision team method description in ILSVRC2012 challenge, available here `https://www.image-net.org/challenges/LSVRC/2012/results.html`.

where $Y_{i,j}^{l}$ is the value of the feature i, j in the output of the l-th layer. As the indices $i-u, j-v$ are constrained by the input size, the output must inevitably have smaller dimension than the input due to the way the filter is being applied and this reduction of dimension is proportional to the size of the filter.

To control the dimensional shrinkage of the data and avoid losing information at the edges of the image after filter application, padding may be added to the image before it is passed through the convolutional layer. For the presented layer, there are k^2 weights to be learned, compared to $n \cdot m \cdot (n-k+1) \cdot (m-k+1)$ in case of an FC ANN. Using fewer weights (or parameters in general) can reduce the time required for a model to converge and can also prevent overfitting [35].

Another way to reduce the number of parameters in a CNN or downsample the input is to introduce a pooling layer, which provides a form of non-parametric dimensionality reduction by summarizing nearby features using e.g. the max or L2-norm function

$$Y_{i,j}^{l+1} = \max_{a,b=0}^{p-1,r-1} Y_{2i+a,2j+b}^{l}, \tag{2.2}$$

where p, r define the pooling window size.

A non-linear activation function, such as the rectified linear unit (ReLU) or a sigmoid function, can be applied to each output of the convolutional layer to introduce non-linearity to the model. This is typically done to improve the model's ability to learn complex patterns in the data [36]. Application of the non-linear activation function can be expressed as

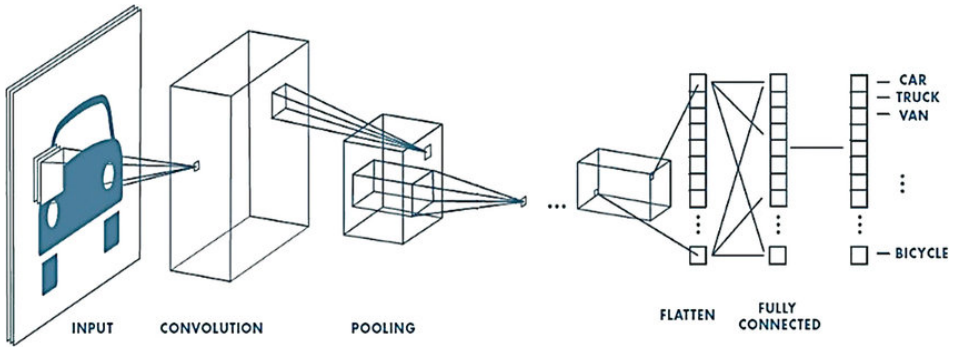
$$Y_{i,j}^{l+2} = \sigma(Y_{i,j}^{l+1}), \tag{2.3}$$

where $\sigma(\cdot)$ is the activation function. An example of an often-used activation function is the Rectified Linear Unit (ReLU):

$$\sigma_{\mathrm{ReLU}}(x) = \max(0, x). \tag{2.4}$$

In Figure 2.1, a typical example of a CNN topology is presented. The network classifies the content of the entire image, but does not implement the detection of individual objects, which is the topic of the following subsections.

Figure 2.1: Example of a CNN architecture. Image courtesy of [37].



2.1.2 R-CNN

In 2013, Ross Girshick et al. applied a novel convolutional neural network-based approach to object detection and proposed a method called the Region-based CNN (R-CNN)

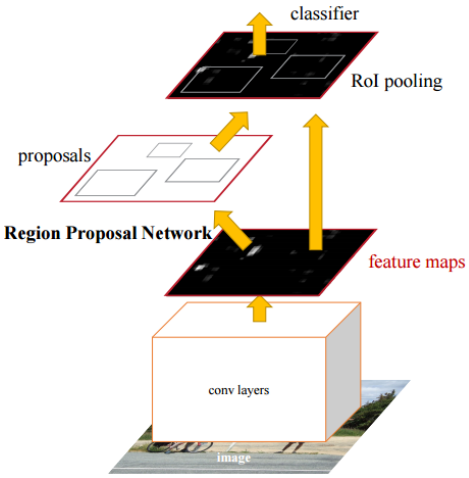
[38]. The architecture uses an optimized version of selective search [39] to generate around 2000 class-independent region proposals (RPs) using various similarity metrics (color, texture, size, etc.) to determine the quality of objectness. A 4096-dimensional feature vector is extracted from each such RP using a CNN described by Krizhevsky et al. [32] with five convolutional layers and two fully connected layers. Before the region can be passed to the CNN, it has to be resized to 227×227 pixels due to architectural constraints. The feature vectors are then utilized by kernelized Support Vector Machines trained for each individual class to perform classification by comparing the resulting score. Once an object is localized, a greedy non-maximum suppression is applied that removes overlapping bounding boxes with lower detection confidence so that only local maxima remain. Finally, the bounding boxes around them are tightened to further improve the localization performance, for which a non-linear regression model is trained. In this method, all parameters of the model cannot be trained simultaneously using a single loss function. Optimizing different model parameters using different loss functions may result in a suboptimal model when judged by the natural objective function of maximizing i.e. the accuracy of a model compared to the ground truth. Furthermore, the proposed model is slower than its later iterations as it applies the convolution operation to every region proposal separately.

Fast R-CNN [38] builds on the previous method by generating region proposals on the output of the convolutional network instead of the whole image. Classification and bounding box correction are implemented by several fully connected layers, which replace both the SVM and regression model used in the R-CNN.

Finally, Faster R-CNN [40] removes the only remaining stand-alone component of the network, the selective search, which it replaces with yet another CNN, the region proposal network (RPN). Region proposal is again carried out on the output of the feature extractor, as shown in Figure 2.2. It is implemented as three convolutional layers, one to preprocess output from the backbone and the other two to compute bounding box offsets and their corresponding confidences. The offsets move and scale the fixed anchor boxes distributed throughout the image. This object detection model is end-to-end trainable.

Recently, novel approaches are emerging, replacing the final bounding box regression with more robust methods such as feature map information fusion used in the Grid R-CNN [41].

Figure 2.2: Faster R-CNN architecture, courtesy of [40].



2.1.3 CenterNet

CenterNet [42] is a one-stage detection method as opposed to the Faster R-CNN, as it skips region proposal and instead performs bounding box inference and object classification on the feature map. In theory, this improves inference speed but comes at the cost of possibly lower accuracy and lesser robustness [43].

Anchors used in Faster R-CNN are replaced by keypoints learned from heatmaps. A disadvantage of using anchors is that a large number of them needs to be generated to ensure a high intersection over union (IoU) rate with the ground truth labels unless some assumptions like a known size of bounding boxes can be made, which is not possible in the domain of UAV detection. Furthermore, the generated anchors are often not aligned with labels [42], which decreases the ability of the anchor-based models to make informed predictions because part of every object may be cut from the input.

The CenterNet model is based on CornerNet [44], which detects objects by identifying keypoints at the top-left and bottom-right corners of the bounding box. The CenterNet model improves upon this approach by using triplets consisting of the center point in addition to the corners and by introducing a new corner pooling layer to help the network detect the corners more accurately. In the CenterNet original paper [42], its authors claim that their major improvement over the CornerNet architecture is in small object detection performance since, from a probabilistic point of view, its center keypoint is more easily located than that of a large object. It matches novel two-stage architectures in accuracy while offering solid inference speed [42].

2.1.4 YOLO

YOLO [45] is a series of one-stage object detection methods widely used for their speed and accuracy trade-off, suitable for real-time applications [8].

YOLO1 [46] divides an image into an $S \times S$ grid and predicts a vector of bounding boxes and associated confidences for each cell. These predictions are encoded in the 3-dimensional tensor output of the CNN with dimensions $(S \times S \times (5 \cdot B + C))$ where C is the number of classes. The input image resolution is limited to 224×224, which makes small object detection difficult. Furthermore, it has a lower recall than region proposal-based methods such as the Fast R-CNN [45].

YOLOv2 [45] implements the idea of anchor boxes whose sizes are determined using the K-Means clustering algorithm on the ground truth. In addition, Darknet-19 is used as the new feature extractor with resolution increased to 448×448, which alone improves performance by 4% mAP. Multi-scale training is added so that during training, every ten batches the algorithm chooses a new input image size from $320, 352, ..., 608$ (multiples of 32). This enables the network to learn to predict well at different resolutions. Overall, YOLOv2 outperforms the first iteration while retaining a fast inference speed.

In YOLOv3 [6], the Darknet-19 backbone is replaced by the Darknet-53. The number in the name refers to the number of layers in the convolutional network. In addition, residual block, skip connections, multi-scale prediction and up-sampling are introduced as part of the feature extractor, as explained in the paper [6]. According to the paper, the network has significantly improved performance on small object datasets compared to the previous iterations. This improvement is speculated to be due to the multi-scale prediction feature, which works by predicting bounding box tensors, merging feature maps at different scales and

processing the combined map by a final convolutional network that benefits from fine-grained features from early on in the network.

YOLO5 [47] is based on YOLOv3, uses a slightly modified CSP backbone [48] and PA-NET neck [49] to boost the information flow and includes mosaic data augmentation capabilities. In addition, it replaces the convolution-based stem unit in YOLOv3 with the Focus layer that aims to improve training speed and reduce information loss caused by aggressive downscaling. See the SpaceToDepth layer described in [50] for details.

YOLOX is an anchor-free method based on YOLOv3 with Spatial Pyramid Pooling [51], which also applies novel techniques for object detection such as a decoupled head that separates box coordinate regression and object classification and SimOTA for label assignment, which solves the problem of label assignment by reducing it to the Optimal Transport problem [8]. Furthermore, it incorporates strong data augmentation, such as Mosaic and MixUp, into its pipeline.

2.2 Fast Visual Tracking

Given an initial state as detection in the form of a bounding box, the task of a tracking algorithm is to locate the (potentially moving) target in subsequent images, assuming the appearance and position of the target do not change significantly in a short period of time. In general, this problem is less computationally demanding than object detection, so a fast visual tracker can suitably complement a relatively slow visual detector. However in practice, the computation duration depends on the specific algorithm and implementation. Only fast trackers that utilize linear regression models are considered here.

2.2.1 Correlation Filter

The algorithms presented in the following two sections are based on a correlation filter, meaning they model the appearance of objects as filters optimized on samples of the target. The filters are then applied to an image to produce a heatmap, where peaks represent the most likely position of the tracked object. These peaks may then be transformed into bounding boxes and given as the output of the tracking algorithm. The filter can be updated with every new frame to incorporate moderate changes in the target's appearance into its model.

Repeatedly applying the convolution operator would be computationally expensive. Hence an optimized approach is utilized, which simplifies the computation by calculating in the frequency domain of the image instead, as the convolution theorem states that

$$t(x) = \{g * h\}(x) \quad x \in \mathbb{R}, \tag{2.5}$$

$$T(y) \triangleq \mathcal{F}\{t\}(y) = G(y) \cdot H(y) \quad y \in \mathbb{R}, \tag{2.6}$$

where $t(\cdot)$ is the convolution of the functions $g(\cdot)$ and $h(\cdot)$, $\mathcal{F}$ denotes the Fourier operator and $G(\cdot)$ and $H(\cdot)$ are Fourier transforms of functions $g(\cdot)$ and $h(\cdot)$. When the inverse Fourier transform is applied to the equation, the following corollary is produced

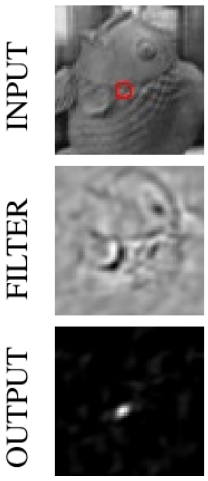
$$t(x) = \{g * h\}(x) = \mathcal{F}^{-1}\{G \cdot H\}. \tag{2.7}$$

In other words, convolution becomes element-wise multiplication in the frequency domain. The Fast Fourier Transform algorithm computes the discrete Fourier transform, or its

inverse, in the image domain in $O(N^2 \log(N))$, where N is the number of pixels in a row of an image. Meanwhile, the cost of a single convolution is $O(K^2)$, where K is the kernel's row size. Hence, for the whole image of $N \times N$ pixels, the complexity of a naive convolution algorithm is $O(N^2K^2)$. Overall, the ability to formulate the tracking task in the Fourier domain circumventing the costly convolution operation is what makes the correlation filter approach so fast [15] and thus suitable for a low-delay tracking task. The effect of the optimization is further increased when more filters are used.

2.2.2 MOSSE

Figure 2.3: MOSSE input, filter and output example, image courtesy of [14].



Minimum Output Sum of Squared Error filter-based tracker (MOSSE) [14] trains on a set of input image patches f_i and outputs g_i. In the original paper [14], g_i is generated from ground truth (i.e. from detection) such that it has a compact Gaussian-shaped peak ($\sigma = 2.0$) centered on the target in f_i, see Figure 2.3. The output is defined as the convolution of the filter h_i with the input image

$$g_i \triangleq f_i * h_i. \tag{2.8}$$

From the convolution theorem

$$H_i^* = G_i \oslash F_i, \tag{2.9}$$

where G_i and F_i are f_i and g_i decomposed by the Fourier transform, H_i is the filter, the asterisk in superscript denotes a complex conjugate and the $\oslash$ symbol represents element-wise division. The filter is found by minimizing the squared sum of errors resulting in the loss function $L_{\text{MOSSE}}(\cdot)$:

$$L_{\text{MOSSE}}(F, G) = \min_{H^*} \sum_i |F_i \cdot H^* - G_i|^2. \tag{2.10}$$

The optimization problem can be solved in closed form by setting the derivative equal to zero as

$$0 = \frac{\partial \sum_i |F_i \cdot H^* - G_i|}{\partial H^*}. \tag{2.11}$$

By solving for H^*, the following expression for the filter is obtained:

$$H^* = \frac{\sum_i G_i \cdot F_i^*}{\sum_i F_i \cdot F_i^*}. \tag{2.12}$$

Once the initial filter is found, each frame j is updated using the following set of recurrent equations:

$$H_j^* = \frac{A_j}{B_j}, \tag{2.13}$$

$$A_j = rG_j \cdot F_j^* + (1-r)A_{j-1}, \tag{2.14}$$

$$B_j = rF_j \cdot F_j^* + (1-r)B_{j-1}, \tag{2.15}$$

$$A_0 = G_0 \cdot F_0^*, \tag{2.16}$$

$$B_0 = F_0 \cdot F_0^*, \tag{2.17}$$

where r is the learning rate, which determines how quickly the effect of previous frames decays over time. See [14] for the complete derivation. The outline shows why the algorithm is so efficient as well as its implementation simplicity. It also illuminates some of its limitations and possible remedies.

The most relevant of these limitations for this work is that scale changes of the target are unaccounted for because the width and the height of the bounding box from which training samples are drawn stay constant until the next initialization from a new detection. This results in noise accumulation in the appearance model as the target ground truth becomes smaller. On the other hand, as the target grows, more local features are tracked, which tarnishes the appearance model, and the tracker may begin to drift. Exponentially decaying the learning rate r might improve the situation by giving more weight to the initial detection.

One issue that can arise when using the MOSSE algorithm for tracking is related to recovery from losing the target. When the target performs aggressive maneuvers or becomes hidden from view, for example behind a tree or a pylon, the MOSSE model cannot produce useful data. However, it is possible to use information from the target's past trajectory to continue tracking the target for a limited time, even when a line of sight is lost. This information can be extracted using a Kalman filter or a similar algorithm. By incorporating this information, the tracking algorithm can better handle occlusion and maintain its ability to locate the target.

2.2.3 KCF

Compared to MOSSE, KCF [15] assumes input data (F) to be non-linear and uses the "kernel-trick" [52] to project it to a higher dimension space using a non-linear function $\phi(\cdot)$ (e.g. a radial basis function kernel), allowing the use of an efficient linear classifier on non-linearly separable data

$$F_i \mapsto \phi(F_i). \tag{2.18}$$

Furthermore, KCF also adds a regularisation component to the optimization task

$$L_{\mathrm{KCF}}(K, G) = \min_H(|KH - G|^2 + \lambda H^T KH), \tag{2.19}$$

where H is the filter, G is the desired response taken from the ground truth, λ is the weight of the regularizer and K is the kernel consisting of products of elements from the training patches

$$K_{i,j} = \phi(F_i)^T \phi(F_j). \tag{2.20}$$

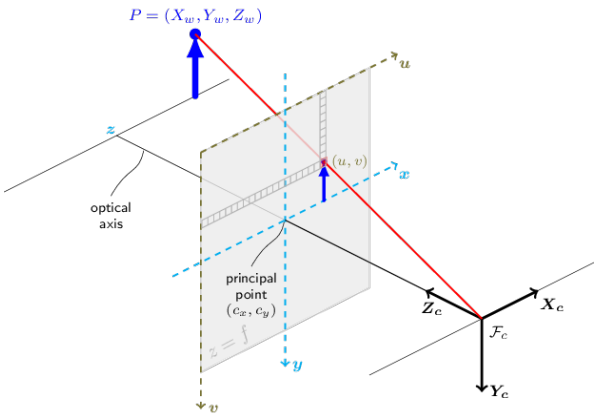
The solution of the optimization task is then

$$H = (K + \lambda I)^{-1} G. \tag{2.21}$$

The KCF is slower than MOSSE since it works with data in a higher dimensional space, but as shown in Section 1.3, it can give more precise results and its performance can still be sufficient for low-delay applications.

2.3 Coordinate Systems

Figure 2.4: Pinhole camera model[2].



The 2D position of the leader in the image that is provided by the visual detection and tracking algorithms can be projected into 3D using a calibrated projection model of the camera. If no further information is supplied (e.g. the size of the target), the projection of the 2D point corresponds to a line going through the object as illustrated in Figure 2.4. Therefore, an estimate of a 3D ray going from the camera through the object is sought instead, which is sufficient for calculating the drone's and gimbal's desired orientations but not their precise distance from the leader. To estimate the exact distance, a stereo camera, a triangulation method, or a neural network such as DisNet [53] could be used, which might introduce further inaccuracies due to noise in odometry or the auxiliary model itself (caused by training).

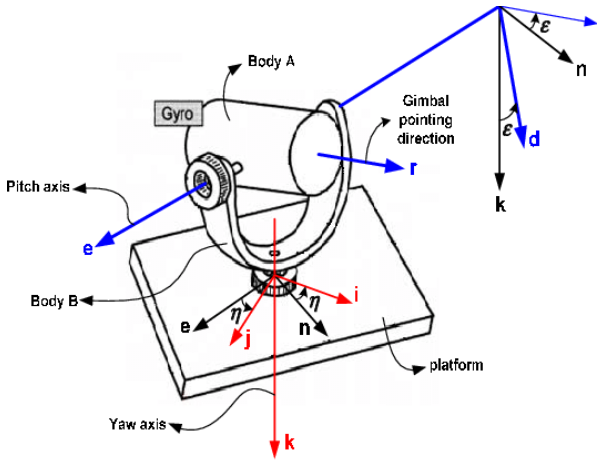
The following reference frames are used in the ensuing calculations inspired by [54]: W is the world coordinate system (GPS) and B is the frame of reference of the follower with origin at the base of the drone (the Pixhawk flight controller). G_1 is the first gimbal reference frame with origin at the joint controlling the yaw and G_2 is the second gimbal reference frame with origin at the joint controlling the pitch. The translation vector between G_1 and G_2 is zero for the gimbal used in the experiments. F_C is the frame of the camera and I is the frame of the image. The relative angles of the gimbal are yaw (η) and pitch (ε), with roll assumed to be constant (equal to zero) in this work. See Figure 2.5 for a model of the gimbal system utilized in this work. There exist isometric transformations between all the mentioned reference frames except I.

Given two arbitrary reference frames A and B, vector v in the reference frame A and assuming an isometric transformation exists between A and B, v can be transformed into B using the following equation:

$$v^B = {}^A R^B v^A + {}^A t^B, \tag{2.22}$$

[2]Taken from the OpenCV online documentation: `https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html`.

Figure 2.5: Model of a 2-axis gimbal system used in this work. Taken from [55].



where R is the rotation matrix and t is the translation vector with respect to the two reference frames.

Consequently, assuming a camera mounted on a gimbal, the transformation from I to G_1 may be represented by the following equation

$$d_{target}^{G_1} = \begin{bmatrix} x_{target}^{G_1} \\ y_{target}^{G_1} \\ z_{target}^{G_1} \end{bmatrix} = {}^{G_2}R^{G_1}\,{}^{I}R^{G_2} \begin{bmatrix} (u^I - c_x^I - {}^{Fc}t_x^I)/f_x \\ (v^I - c_y^I - {}^{Fc}t_y^I)/f_y \\ 1.0 \end{bmatrix} + \begin{bmatrix} {}^{G_2}t_x^{G_1} \\ {}^{G_2}t_y^{G_1} \\ {}^{G_2}t_z^{G_1} \end{bmatrix}, \tag{2.23}$$

where ${}^{G_2}R^{G_1}$ and ${}^{G_1}R^{G_2}$ are estimated from the gimbal's inertial measurement units (IMUs) and the translation vectors can be measured before the experiments. The scalars u and v are coordinates in an image, c is the principal point on the image plane onto which the perspective center is projected and f is the focal length. The parameters of the camera f and c are calculated during camera calibration explained later in this section. The resulting d_{target} vector represents the direction of the target in the first gimbal reference frame. Using this vector, the two desired gimbal angles η_d and ε_d are calculated

$$\eta_d = \arctan(\frac{x^{G_1}}{y^{G_1}}), \tag{2.24}$$

$$\varepsilon_d = \arctan(\frac{x^{G_1}}{z^{G_1}}), \tag{2.25}$$

which represent how much the gimbal has to turn relative to its current position in order for the mounted camera to be centered on the target.

The base rotation and translation need to be applied in order to compute the direction of the target drone in the coordinate frame of the drone

$$d_{target}^{B} = {}^{G_1}R^{B} d_{target}^{G_1} + \begin{bmatrix} {}^{G_1}t_x^B \\ {}^{G_1}t_y^B \\ {}^{G_1}t_z^B \end{bmatrix}, \tag{2.26}$$

which can then used to calculate the rotation of the drone necessary for the camera to point in the direction of the target with the gimbal in its initial configuration (non-rotated joints).

The projection model used in Equation 2.23 assumes a distortion-free pinhole camera model as shown in Figure 2.4. But real lenses may have radial and tangential distortion, so the model described above must be extended by first undistorting the coordinates in the image frame. Radial distortion causes straight lines to appear curved. In the OpenCV library used in this work, radial distortion is modelled using the following equations as described in the OpenCV documentation[3]:

$$x_{undo\_radial} = x/(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \tag{2.27}$$

$$y_{undo\_radial} = y/(1 + k_1 r^2 + k_2 r^4 + k_3 r^6), \tag{2.28}$$

where k_1,k_2,k_3 are distortion parameters, x and y are distorted coordinates in the image plane and

$$r^2 = x^2 + y^2. \tag{2.29}$$

Tangential distortion makes certain parts of the image look closer than they really are. This may be modelled as such:

$$x_{undo\_tangent} = x - [2p_1 xy + p_2(r^2 + 2x^2)], \tag{2.30}$$

$$y_{undo\_tangent} = y - [p_1(r^2 + 2y^2) + 2p_2 xy]. \tag{2.31}$$

These two systems of equations can be combined into one transformation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \{x - [2p_1 xy + p_2(r^2 + 2x^2)]\}/(1 + k_1 r_t^2 + k_2 r_t^4 + k_3 r_t^6) \\ \{y - [p_1(r^2 + 2y^2) + 2p_2 xy]\}/(1 + k_1 r_t^2 + k_2 r_t^4 + k_3 r_t^6) \end{bmatrix}, \tag{2.32}$$

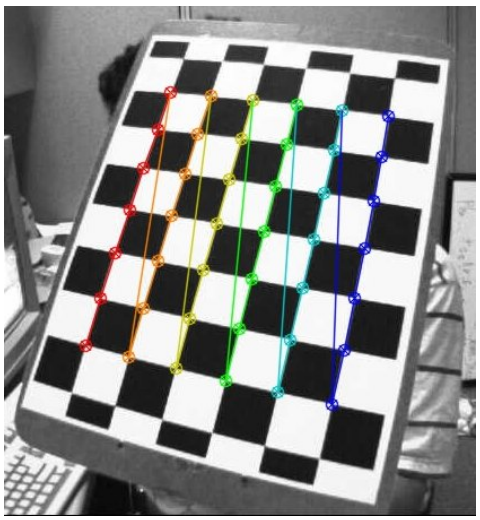
where

$$r_t^2 = x_{undo\_tangent}^2 + y_{undo\_tangent}^2. \tag{2.33}$$

The distortion coefficients $\left[k_1, k_2, k_3, p_1, p_2\right]$ are calculated during camera calibration. For camera calibration, a known pattern with a fixed size is used, for example a chess board with 30mm long squares. Points where four squares intersect are chosen for the calculations, see Figure 2.6. These points are selected as they are easily detectable. By moving the object around and thus changing its scale in the image frame, 2D and associated 3D positions are recorded. A system of equations can be created in such way that its unknowns are the instristic camera parameters and the knowns are the coordinates. Once the equation is solved, Levenberg-Marquardt global non-linear optimization algorithm [56] is run to minimize the reprojection error between the observed and the projected calibration pattern points. The result of the optimization are the calibrated parameters of the camera model.

[3] https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

Figure 2.6: Chess calibration pattern. Taken from the OpenCV on-line documentation [54].



Chapter 3

Solution Architecture

3.1 Robot Operating System

Robot Operating System (ROS) utilized in the implementation of the proposed system is a free and open-source set of frameworks that simplifies the development of robotic applications. It implements low-level device control and abstractions, message passing between individual components, coordinate frames and transformations among them, package management, and more[1]. Each individual computational component in the ROS environment is called a node, which is an executable that can interface with the aforementioned subsystems. Nodes are organized in packages, which logically group the functionality of a robotic system.

The MRS UAV System [57] is a set of packages built on top of ROS by the Multi-robot Systems Group at the Czech Technical University in Prague that implements useful UAV functionalities. To name a few utilized in this work, the UAV Manager package provides a high-level interface for takeoff and landing, the Trajectory Generation package can generate a time-parametrized trajectory from a path, the Reactive Obstacle Bumper prevents the drone from crashing into obstacles and the Sensor Fusion and Localisation provides full UAV state estimate including lateral position, velocity, acceleration and heading. It can be used with any multi-rotor vehicle equipped with a PX4-compatible flight controller.

3.2 Hardware

The solution proposed and implemented as part of this thesis is designed to run on a drone equipped with a Pixhawk 4 embedded flight stabilizer, a 2-axis gimbal with the SimpleBGC controller, an RGB camera, and an on-board computer with a Linux-based operating system capable of running ROS.

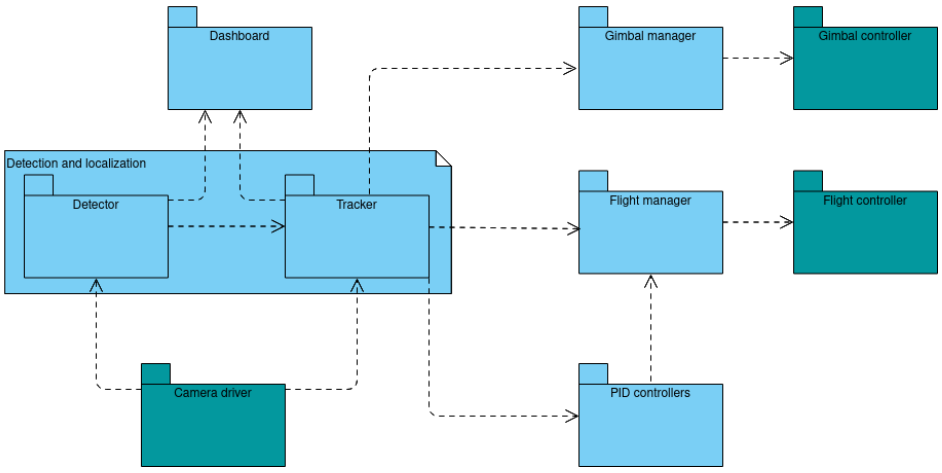
In the experimental part of this work, a quadcopter is equipped with a FLIR BlackFly BFS PGE 50S5C camera mounted inside a 2-axis gimbal controlled by a BaseCam SimpleBGC microcomputer. The camera is capable of producing images at a rate of 24 FPS having a resolution of 2448×2048 pixels[2]. The gimbal is mechanically limited to approximately 50 degrees of movement in both the yaw and pitch axes. The drone is also equipped with an Intel NUC 10i7FNK barebone computer kit running Ubuntu 20.04.4 LTS and ROS Noetic, which receives the camera's images over a GigE interface.

[1] For a more detailed overview, see `http://robotics.stanford.edu/~ang/papers/icraoss09-ROS.pdf`.

[2] The complete camera specification sheet is available on the manufacturer's website: `http://softwareservices.flir.com/BFS-PGE-50S5/latest/Model/spec.html`.

3.3 Architecture Overview

Figure 3.1: Architecture of the implemented system. The blue boxes are separate packages and the arrows represent information flow.



The solution depicted in Figure 3.1 transforms input from a monocular RGB digital camera into instructions for the gimbal driver and the Pixhawk flight controller. Data between the packages are passed as ROS messages over the TCPROS protocol which uses persistent, stateful TCP/IP connections[3].

At the beginning of the pipeline, an input image received from the camera is passed to the detection model and a hypothesis with the highest prediction confidence is selected as the current target to follow. The detection model is significantly slower than the framerate of the camera, so a fast visual tracker is used to track the position of the target in frames between the detections.

The inferred position of the target in the world frame including the timestamp of the corresponding input image is then passed to the gimbal and flight managers, which compute and update the desired drone and gimbal orientations.

Both the vertical and heading velocities of the follower are regulated by separate PID controllers whose parameters are tuned in a simulation.

A dashboard package serves as a visualization tool to display images from the camera, the current target's location estimates, and other information about the drone state.

3.4 Detection and Localization

3.4.1 Detector

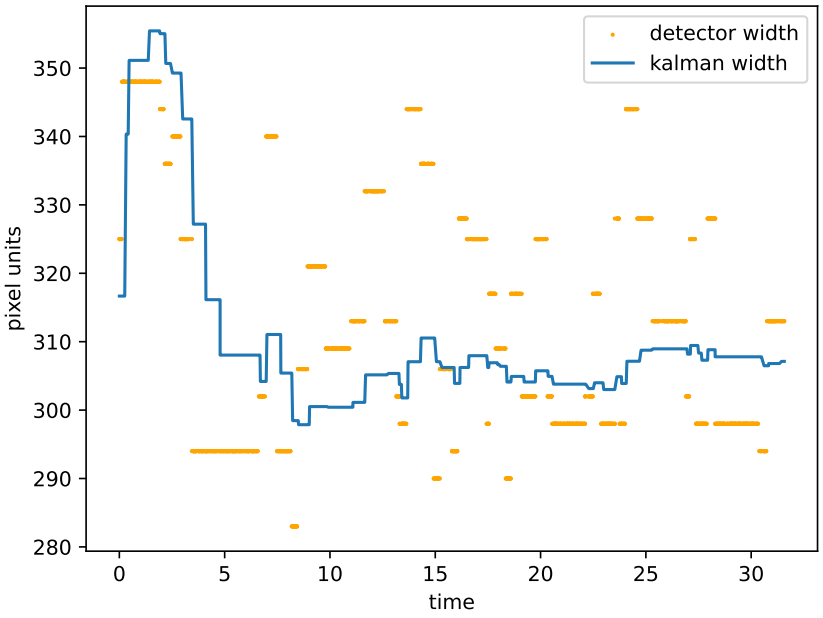
The detector module is implemented as a listener that waits in a sleep state for a message from the camera driver, which it can receive asynchronously. Once an image is received, it is resized to a lower resolution (640×640 pixels) to improve processing speed and is then passed as an input to the pre-trained model further described in Chapter 4. If no drone is

[3]See details on ROS wiki: `http://wiki.ros.org/ROS/Technical%20Overview`.

detected with confidence over an empirically inferred threshold (the value of 0.5 was used for the experiments described in this thesis), the original image is split into 4×3 subimages and each is processed by the model separately without resizing to extract maximum information.

From the resulting list of detections, the target with the highest confidence is selected and its bounding box is sent to the tracker. Its width is also supplied to the PID controllers. The output of the detector is noisy, therefore a Kalman filter [58] is applied on the output width to give a better estimate and to make the drone flight smoother, see Figure 3.2 for a test result on a static target.

Figure 3.2: Kalman filter smoothing of the width of the target's bounding box. The time on the x-axis is in seconds.



3.4.2 Tracker

The tracker node stores a queue of images with a maximum size of 24 images (1 second of footage). Once a detection is received, the corresponding image is found by the timestamp from the detector and a MOSSE tracker with the learning rate set to 0.15 is initialized using the detection's bounding box and the image.

As detections may be produced more quickly than the tracker loses its target, if detection is received and the tracker is still considered valid, the intersection over union between the two is checked at the timestamp of the detection. If it is over 0.5, the tracker is not reinitialized to save computation cycles, only the expected width of the target is updated to keep the PID controllers informed.

3.5 Gimbal Manager

The gimbal manager node controls the yaw and pitch joints of the gimbal based on the output from the tracker. The gimbal is either searching for a target if there is none currently tracked by following a constant circle trajectory or keeping an object in its field of view by keeping the camera centered on the received coordinates. If the target is lost, the gimbal remains turned in the direction of its last position for a predefined duration before it switches back to the searching state. The desired joint angles are calculated according to the equations derived in Section 2.3. The angles are passed to the gimbal controller (BaseCam SimpleBGC), which then controls the gimbal's actuators inside a feedback loop with encoders measuring the actual angles of the gimbal joints.

3.5.1 PID Controllers

The proportional-integral-derivative controller (PID controller) [59] is a control loop mechanism that automatically applies a correction to a control function based on feedback. Assuming a measured process variable $w(t)$, e.g. the width of the target bounding box at time t, and desired setpoint variable $r(t)$, e.g. the desired width of the target, the process error is calculated as follows

$$e(t) = r(t) - w(t). \tag{3.1}$$

The process error signal is what a PID controller attempts to minimize. The controller has three parameters p, i and d which can be empirically estimated. It calculates a control signal $u(t)$ as

$$u(t) = pe(t) + i \int^t e(\tau)d\tau + d\frac{\partial e}{\partial t}, \tag{3.2}$$

which is then used to e.g. control the velocity of the drone.

Two separate PID controllers are used in the system. The velocity PID controller takes as input the process variable the width of the target detection's bounding box and the width setpoint (a dynamic parameter) from the tracker. The width process variable from the tracker is first passed through a Kalman filter. It is approximately equal to the inverse of the leader-follower distance (see Figure 3.3). For a more detailed derivation of the relation, see [24]. As can be seen in Figure 3.3, once the target is lost, the setpoint is advertised to the controller to stabilize the drone. The altitude PID controller receives the desired pitch angle of the detection (obtained in Equation 2.24) as input and outputs the desired altitude velocity to control this angle to 0 so that both drones are at the same height.

The parameters of the PID controllers are autotuned using the Ziegler Nichols method by first only considering the proportional controllers and adjusting the p gains until consistent and stable oscillations are reached after which the resulting gain and oscillation period are used to calculate i and d parameters. See [60] for a detailed description of the method. The parameters are then slightly adjusted to make the drone flight less aggressive. The resulting gains are:

$$\begin{bmatrix} p_v \\ i_v \\ d_v \end{bmatrix} = \begin{bmatrix} 0.03 \\ 0.01 \\ 0.003 \end{bmatrix} \tag{3.3}$$

and

$$\begin{bmatrix} p_a \\ i_a \\ d_a \end{bmatrix} = \begin{bmatrix} 6 \\ 1 \\ 4 \end{bmatrix} \tag{3.4}$$

for the velocity and altitude PID controllers respectively.

3.6 Flight Manager

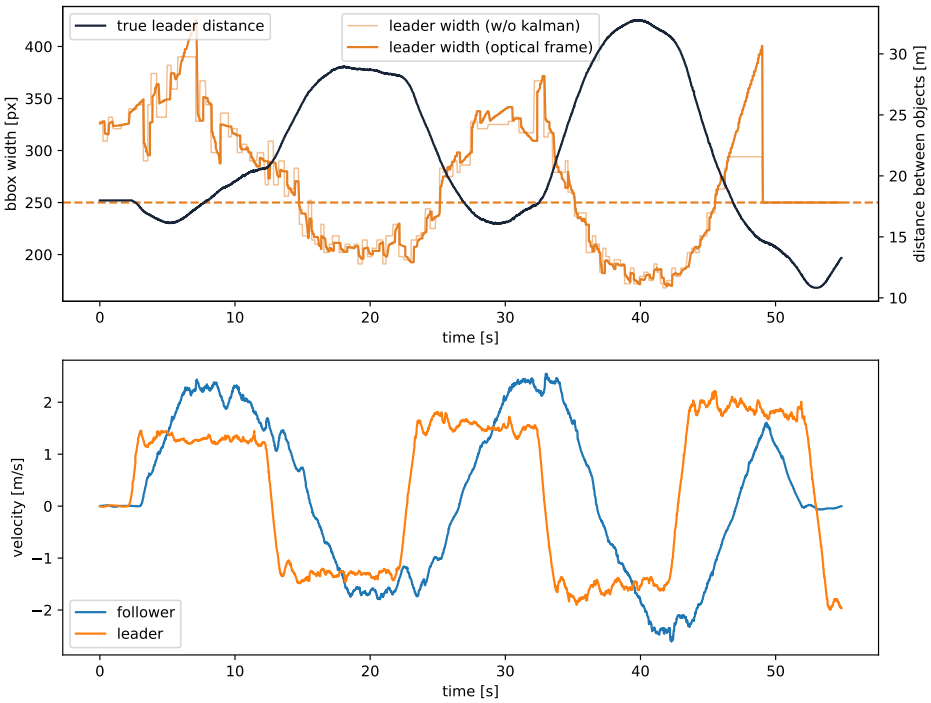
To control the flight of the follower, the two PID controllers described in the previous section are utilized to control the velocity and the altitude of the drone. The desired heading of the follower is calculated from the relative bearing of the leader drone, so the line of sight between the gimbal and the drone is not broken due to the gimbal's limited angular range.

The desired velocity vector of the follower at time t is calculated using the following equation:

$$v(t) = \begin{bmatrix} u(t)\cos(\theta(t)) \\ u(t)\sin(\theta(t)) \\ a(t) \end{bmatrix} \tag{3.5}$$

where $u(t)$ is the latest control signal from the velocity PID at time t and $a(t)$ is the latest control signal from the altitude PID at time t. $\theta(t)$ is the latest relative bearing of the leader drone at time t.

Figure 3.3: Relation between distance and the width of the target in simulation. Losing the target resets the width output of the tracker to the setpoint at $t = 49$ seconds.



3.7 Dashboard

The Dashboard is a user interface that provides an overview of the current state of the follower. This includes visualizations of the target's current position, the follower's desired velocity, altitude, and the time elapsed since the last target detection. The dashboard also includes a live feed from the follower's on-board camera, allowing the operator to see what the drone sees. The dashboard is designed to be simple and easy to use, providing key information at a glance.

Chapter 4

Model Selection

4.1 Motivation

Because the overall performance of the system is dependent on the precision and robustness of the detector and visual tracker, care was taken when selecting appropriate methods for these modules. Multiple algorithms were compared with respect to their average precision, recall, and inference time, using a merged collection of drone datasets discussed in the next section.

4.2 Detection

4.2.1 Dataset Exploration

MIDGARD [57] is an automatically annotated open image dataset of UAVs collected by an RGB camera on board a drone in different environments and under mixed conditions listed in Table 4.1. The automatic labeling is done by placing UV markers on the targets, performing detection, reprojecting estimated poses into the camera coordinate space and building an annotation that also includes target distance and its standard deviation estimates. The accuracy of the predicted annotations is affected both by the camera parameters and the distance to the target [57]. In general, the error of the sensor increases with the distance of the target.

In Figure 4.1, the results of data exploration on MIDGARD are summarized, specifically the distributions of drone positions and bounding box sizes. Positions appear to be approximately normally distributed around the center of the image. The significant cluster near the origin is partially caused by some images having near-zero or even slightly negative bounding box coordinates. This is expected behavior as explained by the MIDGARD authors in [57], nonetheless, those samples need to be cropped or ideally culled to not negatively influence training and accuracy measurements of the predictors. Bounding boxes usually take less than half of an image, with a small fraction of extremely small targets (less than 1% of an image or 8 pixels across). These extremely small targets are removed from the dataset as they are considered hard to distinguish from noise and outside the scope of this work.

Det-Fly is the second dataset utilized in this work, consisting of over 13000 images of the DJI Mavic 2 drone, again covering a wide range of environments and conditions [7]. The bounding boxes in the dataset are tighter as it is manually annotated. See the exploration in Figure 4.2. Drones in the dataset are relatively small, taking on average less than 5% of the image width. The data is supplied in high-resolution (3840×2160 pixels). Assuming the average bounding box width is approximately 192 pixels, it gets reduced to 32 pixels when the image is resized to the model input resolution of 640×640. Such a small projection might

Background	Lighting	FoV	Frames
Fields, hills	Direct sunlight	180°	780
Fields, hills	Direct sunlight	96°	554
Coniferous forest	Direct sunlight	180°	763
Coniferous forest	Direct sunlight	96°	769
Semi-urban	Direct sunlight	96°	475
Stands	Direct sunlight	96°	586
Modern architecture	Strong indirect natural light	96°	534
Historical stairwell	Low light through windows	96°	319
Church interior	Very low mixed light	96°	984
Church exterior	Overcast, late evening sky	96°	697
Warehouse interior	Low fluorescent lightbulbs	96°	564
Warehouse exit	Changes halfway	96°	272
Apartment buildings	Overcast sky	96°	300
Total			**7597**

Table 4.1: Structure of the MIDGARD dataset, courtesy of [57].

still be recognizable, but all targets that would be reduced to less than 8 pixels across are again removed.

4.2.2 Training Procedure

For all the detection architectures discussed in Chapter 2, after cropped samples are discarded, images are reduced to the size of at most 640×640 pixels and batch size training parameter is set to 32 images per iteration when possible. All the utilized backbones are first pre-trained on COCO or ImageNet to speed up convergence during early-training [61]. Aterwards, each model is trained and validated on the MIDGARD dataset.

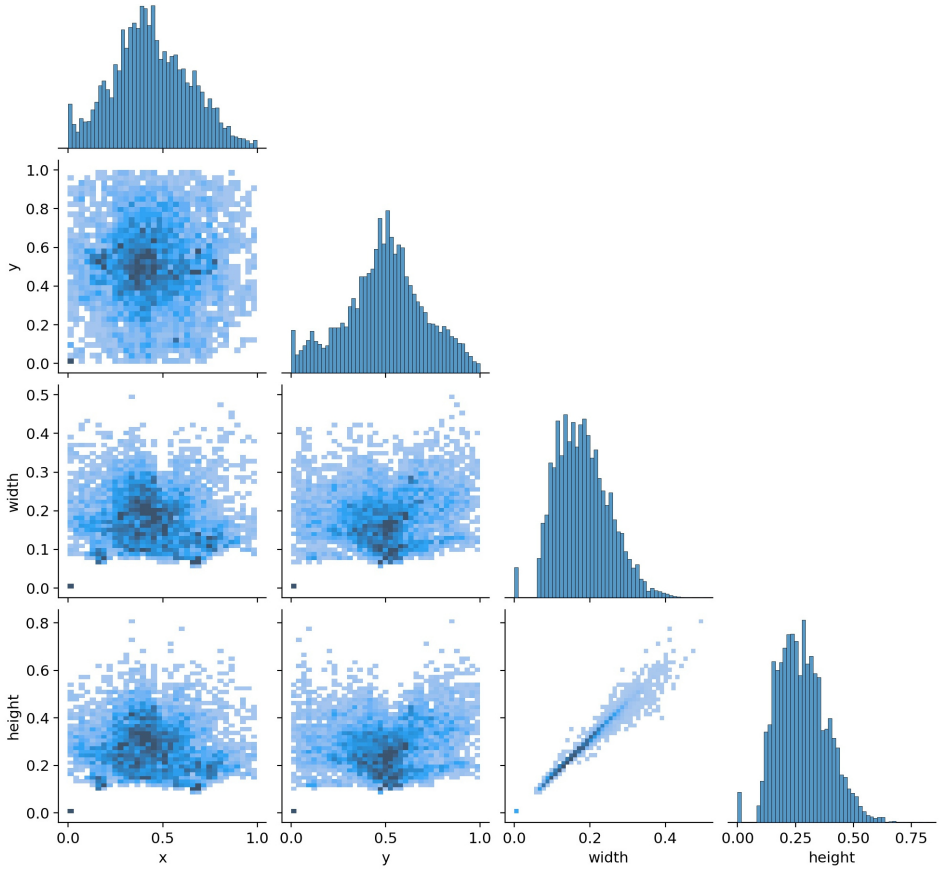
The nature of the dataset (frames extracted from multiple flights) warrants special treatment, as there exists a large correlation among consecutive frames, so the dataset cannot be simply split randomly. Each environment is kept separate and data from one flight is never used in more than one of the training, validation, and testing stages. The split ratio of the data is 0.7/0.15/0.15. The training procedure is stopped after 120 epochs (or 80 epochs when the models' hyperparameters are optimized before training) if the validation loss stops decreasing (which was the case for every model tested in this thesis). The time it took the model to reach its minimum validation loss is reported as the training time.

The algorithm that shows an ideal compromise between accuracy and inference speed on the test dataset is chosen. Finally, this model is optimized, trained, and tested on the combined dataset of MIDGARD and Det-Fly.

4.2.3 Accuracy Metrics

A set of detections D with varying confidence is produced by the model. Which detections are chosen to be valid output depends on the threshold parameter θ. The lower the parameter, the more false positives are produced

Figure 4.1: Data exploration of the MIDGARD dataset. Shows e.g. distributions of various (normalized) statistics of the dataset such as widths of the bounding boxes.



$$D^i_{out} = [D^i_{confidence} > \theta]. \tag{4.1}$$

In order to compare the resulting detections with the ground truth using common metrics, one needs to define what is still a valid detection in terms of overlap of the predicted and ground-truth bounding boxes. For this purpose, the Intersection over Union (IoU) metric is introduced. Let A and B be two convex shapes representing an area of an image. Then, the IOU of these two shapes is defined as

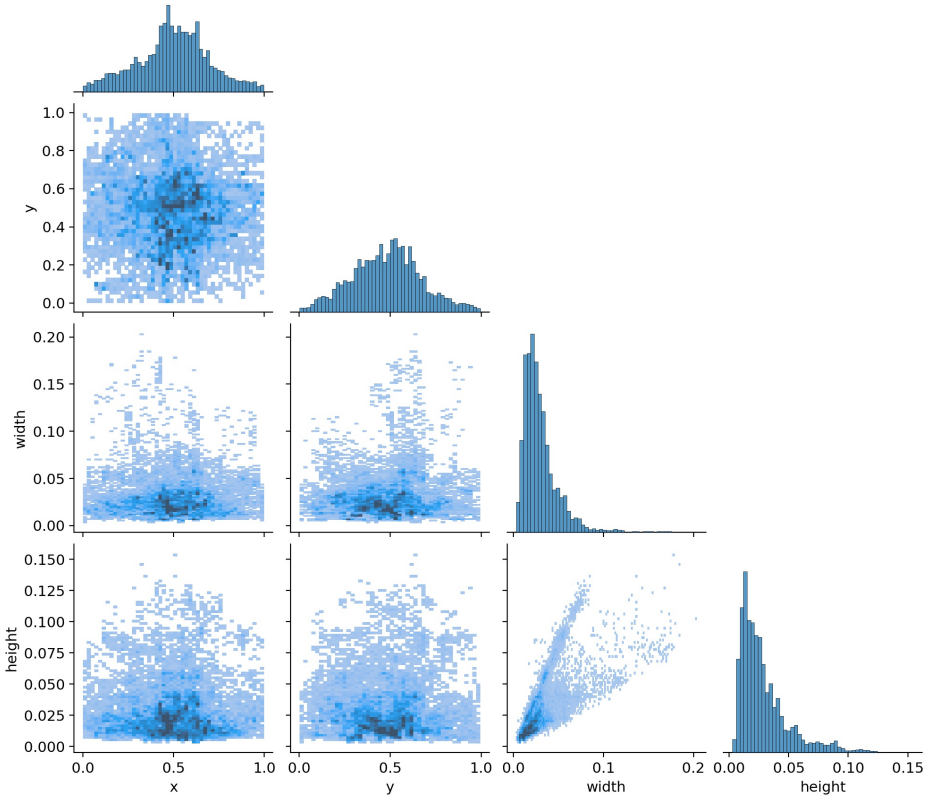
$$IoU_{A,B} = \frac{|A \cap B|}{|A \cup B|}. \tag{4.2}$$

If A is a detection (D_i) and B is the corresponding ground truth bounding box, a detection is valid if the $IoU_{A,B}$ is larger than an IoU threshold ψ. Since the dataset used is in part automatically labeled, a weaker $\psi = 0.5$ is used. Once the number of true and false positives (T_p, F_p) and negatives (T_n, F_n) is known, the precision and recall of the model may be calculated:

$$P_\theta, \psi = \frac{T_p}{T_p + F_p}, \tag{4.3}$$

$$R_\theta, \psi = \frac{T_p}{T_p + F_n} \tag{4.4}$$

Figure 4.2: Data exploration of the Det-Fly dataset.



The main evaluation metric used in this chapter is the mean average precision (mAP), which is equal to the average precision (AP) for one-class models. AP is the area under the precision-recall (PR) curve:

$$AP_\psi = \int_{\theta=0}^{1} PR_{\theta,\psi} d\theta, \tag{4.5}$$

where the PR curve expresses the precision and recall parametrized by $\theta \in [0; 1]$.

A stricter IoU of $\psi = [0.5; 0.95]$ is also used, which averages an evaluation metric (e.g. AP) over $\psi \in [0.5; 0.95]$ with a 0.05 step size.

4.2.4 Benchmark on Test Set

Refer to the Table 4.2 for the results. Inference speeds are measured on an Intel Xeon Silver 4110 CPU, which has similar performance to the Intel Core i7-10710U processor installed in the onboard computer of the drone platform used for later experiments (the Xeon being slightly slower[1]). The training time of CenterNet is higher than expected because the authors' repository does not have batch gradient descent implemented in a way that would utilize the

[1] According to a benchmark available from here `https://gadgetversus.com/processor/intel-core-i7-10710u-vs-intel-xeon-silver-4110/`.

Method	Backbone	Accuracy (AP 0.50 IoU)	Recall (0.5:0.95 IoU)	Inference (ms)	Training (h)
CenterNet [42]	dla34	0.47	0.22	957	11 (batch size 1)
Faster RCNN [40]	x101 64x4d fpn 1x	0.48	0.2	5632	20
Faster RCNN	r50 fpn 1x	0.33	0.15	3854	14
Grid RCNN [41]	r50 fpn gn-head 2x	0.38	**0.28**	4375	12
YOLOv3 [45]	CSPDarknet	0.43	0.20	350	1
YOLO5 small [47]	CSPDarknet	0.49	0.27	243	1.3
YOLOX tiny [8]	CSPDarknet	0.43	0.21	**186**	1
YOLOX small	CSPDarknet	**0.55**	**0.28**	324	1.5

Table 4.2: Comparison of results of the selected methods on the testing data using several metrics.

Method	Max resolution	Accuracy (AP 0.50 IoU)	Improvement (%)
YOLO5	640	0.49	-
YOLO5	800	0.56	12
YOLO5	1000	0.58	16

Table 4.3: Effects of increasing input resolution on the accuracy. The models were trained on 640×640 images. Giving the model larger input significantly improves its performance.

massive parallelization capabilities of the NVIDIA Tesla V100 SXM2, on which the models are trained.

Notably, the novel single-stage method YOLOX gives the best average precision while also offering outstanding inference speed suitable for a low-delay application. The CenterNet also achieves competitive accuracy while being at least three times slower than the comparable anchor-less YOLOX architecture. The YOLO5 is a slightly smaller network than YOLOX with weaker data augmentation offering a good compromise on speed and precision. YOLO5 has a large number of hyperparameters that can be tuned using e.g. evolutionary algorithms to further improve its accuracy.

A surprising result is that the relatively simple YOLO architectures greatly outperform the more complex models – Faster R-CNN and Grid R-CNN. More data and or training time might be required to truly utilize the large number of parameters in these architectures. The task at hand is essentially a binary classification problem of image segments that often have tens of pixels across, hence a large number of parameters might not be necessary and in fact prove detrimental. When comparing the classifiers based on the size of the target object, Faster R-CNN demonstrates better performance than YOLOX small model on medium and large objects[2] (0.225 vs 0.195, IoU 0.50:0.95), which further supports this hypothesis.

The influence of input resolution on the testing is also measured and reported in Table 4.3. The chosen YOLO5 network is still trained on 640×640 images, but the input is scaled at inference time. While this naturally slows down the prediction, it shows that the pipeline might be enhanced by e.g. using a sliding window approach or cropping the image to the area that contained the drone in the last iteration.

Finally, YOLO5 small is trained on the MIDGARD+Det-Fly merged dataset for 120 epochs. The mAP improves by 4% compared to the model trained only on the MIDGARD dataset. The rest of the experiments are performed using this improved model. See Figure 4.4 for several examples of detections made by the model.

[2]Small objects are defined as having an area of less than 32^2 pixels.

Figure 4.3: Performance comparison of the YOLO5 trained on MIDGARD and on the combined MIDGARD+DetFly dataset.

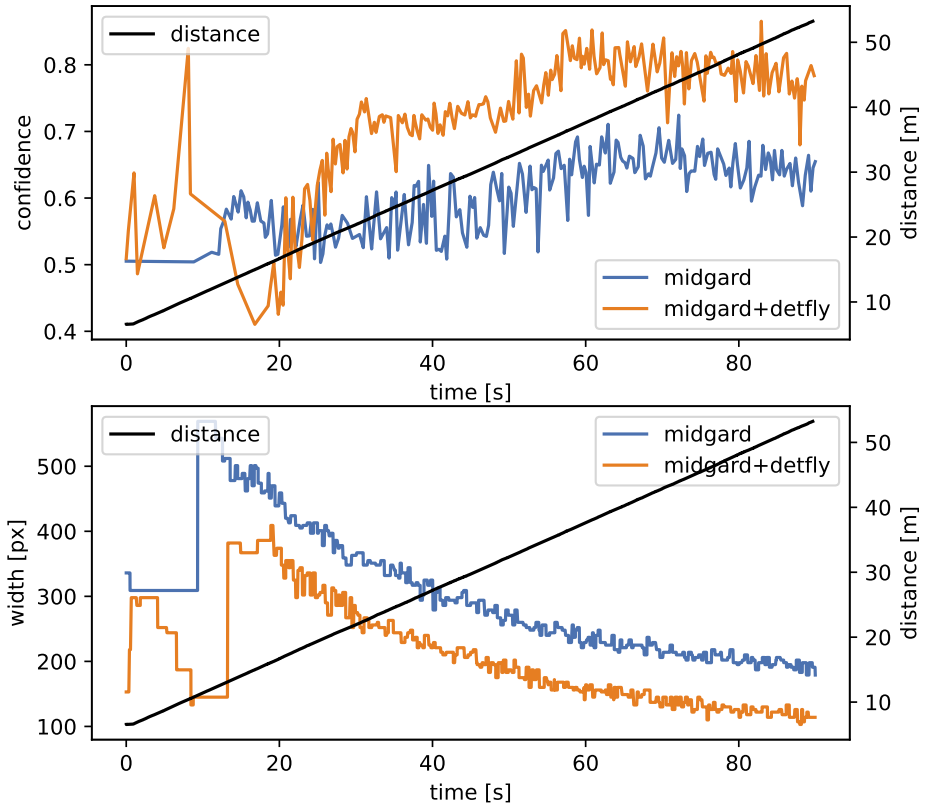
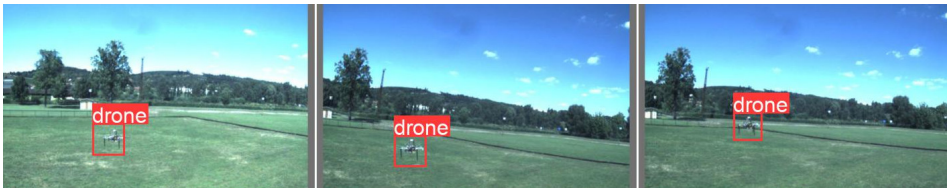


Figure 4.4: Prediction performed on a sample by the final model.



4.2.5 Evaluation of YOLO5 in Simulation

Comparison between the two models trained on different datasets is also made experimentally inside a simulation. Two Tarot 650 drones are placed beside each other inside a simulated nature reserve world. One drone remains static at origin and keeps its gimballed camera centered on the second drone, which slowly flies away along a straight trajectory.

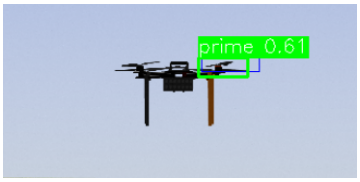
The result is shown in Figure 4.3. The model trained on MIDGARD only performs better at distances below 20 meters, while the model trained on both MIDGARD and Det-Fly is more confident yet predicts incorrect bounding box dimensions for distances below 15 meters, where it picks up local features such as propellers and classifies them as drones (see Figure 4.5). This is illustrated in the graph by the high confidence peaks below 20 meters for the Midgard+Det-Fly dataset. This might be due to both the datasets (but especially Det-Fly) being skewed towards small objects (see Figure 4.2). This might cause the follower to start accelerating rapidly when close to the leader once deployed as part of the system

developed in this thesis. This problem should be addressed in later work by adding more large images of drones to the dataset that were not necessarily taken in flight. In later experiments, the distance between the drones will be constrained by this model's limitation.

For targets 20 meters or further from the follower, the MIDGARD+Det-Fly model is both more confident and produces a tighter bounding box around the target. This might be due to the fact that the Det-Fly dataset was manually annotated, thus the model contains less noise from the drones' surroundings. The model trained on both the MIDGARD+Det-Fly combined dataset is also preferred due to its higher AP as mentioned in the previous subsection.

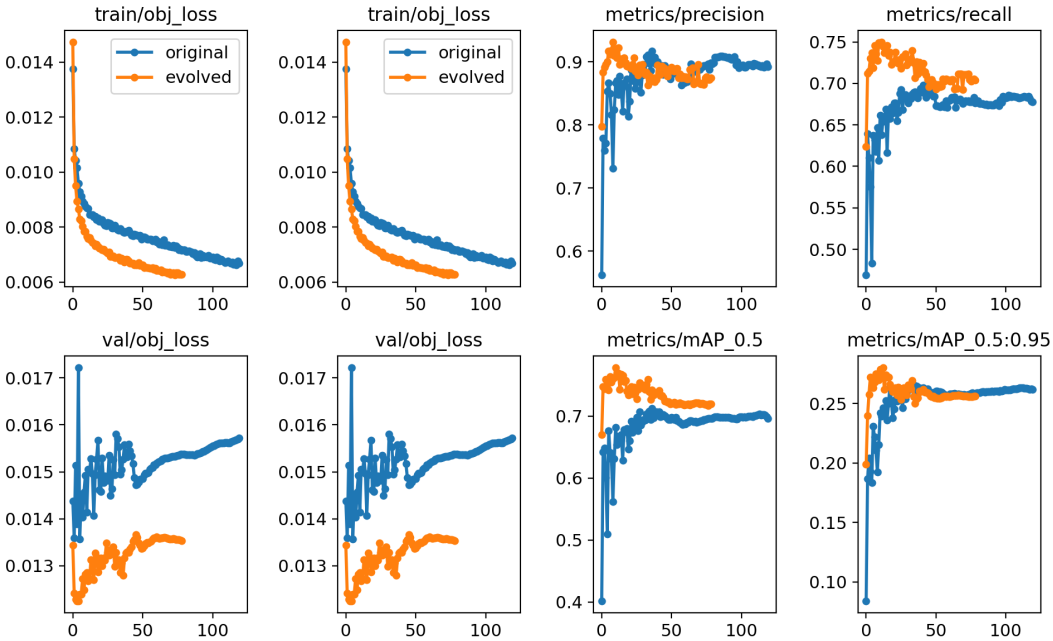
The simulation is less nuanced when compared to a real-world deployment as it contains no objects like cars, birds, and planes, which could increase the false positive rate of the classifier, but nevertheless, the result is solid and moreover confirms that the YOLO architecture can indeed run well on board a UAV and is suitable for the task of drone tracking.

Figure 4.5: At small distances of less than 15 meters, the detector sometimes picks up local features instead of the whole drone as it was trained on a dataset of mostly small drones.



4.2.6 Hyperparameter Evolution

Figure 4.6: Effect of hyperparameter evolution on training. The x-axis corresponds to the number of training epochs.



parameter	description	original	evolved
lr_0	initial learning rate (SGD=10^{-2}, Adam=10^{-3})	0.01	0.0014
lr_f	final OneCycleLR learning rate ($lr_0 * lr_f$)	0.1	0.04
momentum	SGD momentum/Adam beta1	0.937	0.919
weight_decay	optimizer weight decay	0.0005	0.00046
warmup_epochs	warmup epochs (fractions ok)	3.0	3.11
warmup_momentum	warmup initial momentum	0.8	0.95
warmup_bias_lr	warmup initial bias lr	0.1	0.08
box	box loss gain	0.05	0.047
cls	cls loss gain	0.5	0.467
cls_pw	cls BCELoss positive_weight	1.0	0.714
obj	obj loss gain (scale with pixels)	1.0	1.143
obj_pw	obj BCELoss positive_weight	1.0	0.849
mosaic	image mosaic (probability)	1.0	0.78

Table 4.4: Evolution of some of the hyperparameters.

YOLO5 has 29 parameters, which is too many to effectively optimize manually, so a genetic algorithm is utilized. The batch job ran for 3 days on two NVIDIA Tesla V100 125 TFLOPs graphic cards. The fitness being maximized is the weighted combination of the metrics mAP@0.5 and mAP@0.5:0.95. Mutation genetic operator [62] is used, with a 90% probability and 0.06 variance to create new offspring based on a combination of the best parents from the previous generations judged by their fitness. The weights are shown in the optimization criterion expressed in Equation 4.6 and are consistent with the remark regarding the label noise discussed in Subsection 4.2.1.

$$\max(0.75 \cdot \text{mAP@0.5} + 0.25 \cdot \text{mAP@0.5\_0.95}). \tag{4.6}$$

Figure 4.6 shows the considered metrics during 100 training epochs for the original and evolved values of the hyperparameters. The convergence speed increases considerably and the achieved average precision improves by 6% on the validation and 5% on the test set. The initial values as well as the results of the optimization are listed in Table 4.4. Only a subset of all the 29 parameters is considered as the task is computationally demanding. Each parameter greatly increases the search space and the network must be repeatedly retrained.

4.2.7 Results

An optimized YOLO5 model was chosen for further experiments as the best classifier for the given task out of the ones considered. It provides a good accuracy on the task of drone detection that can be further improved using a deeper architecture while maintaining a high processing speed suitable for low-delay detection and tracking. In contrast to the work presented in [7], the single-stage approach has outperformed the two-stage models, which matches the SotA benchmark referenced in the introduction [5].

4.3 Tracking

4.3.1 Experiment Set-Up

The KCF and MOSSE algorithms are compared here on a simple task inside a simulation. The follower remains static with the gimbal following the leader according to data from the tracker, while the leader flies around the follower on an arc trajectory approximately 20 meters away from the target. The tracker is initialized from the detector. The experiment is run for 2 minutes with each algorithm with the speed of the leader limited to 2 m/s. The computer used for the benchmark is the same one used for later experiments and listed in Section 3.2.

4.3.2 Results

Algorithm	Efficacy (%)	Init speed (ms)	Update speed (ms)	Average performance (FPS)
MOSSE	0.98	101	15	9.45
KCF	0.88	18	126	4.89

Table 4.5: Results of the tracker benchmark in simulation.

The results are shown in Table 4.5. MOSSE is chosen as the tracker for further work due to its superior speed which enables the drone to be more responsive to changes in the leader's trajectory.

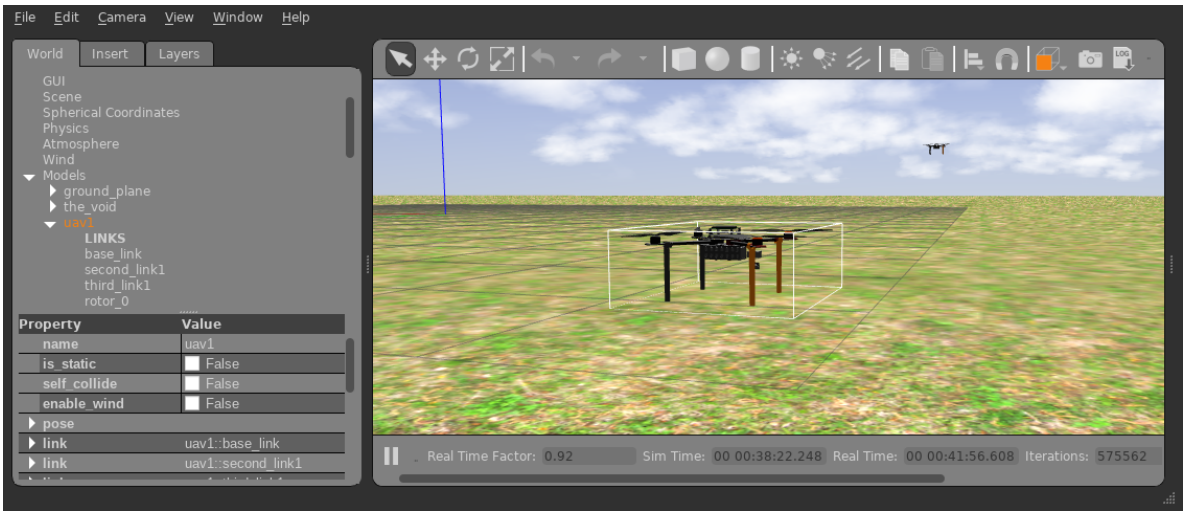
The efficacy metric shows how often the result from the tracker matches ground truth with IoU threshold set to 0.5. The reason why KCF has lower efficacy than the simpler MOSSE is partly that it is not running fast enough for the gimbal to keep up with the leader, especially when the leader suddenly starts moving in the opposite direction. Therefore, it sometimes happens that the follower flies out of the camera's line of sight.

Chapter 5

Experiments in Simulation

5.1 Gazebo Simulator

Figure 5.1: Gazebo GUI with two spawned T650 drones.



Gazebo is used for all simulated experiments in this work. It is[1] an open-source 3D robotics simulator integrating a physics engine for rigid and deformable body dynamics and collision detection (e.g. ODE, Bullet, Simbody or DART), OGRE for rendering 3D scenes and various other libraries for sensor generation, GUI, etc. The simulator is separated into two programs, the gzserver for simulating physics, rendering and sensors, and the gzclient shown in Figure 5.1, which provides a GUI to visualize and interact with the world. It can be integrated with ROS via the gazebo_ros_pkgs[2] set of ROS packages.

Simulated robots inside Gazebo are dynamic structures constructed from rigid bodies connected by joints and represented in the Unified Robot Description Format (URDF) markup language. All simulated objects may have associated mass, velocity and friction and various forces and torques may be applied to their surfaces. They are designed to accurately emulate the behavior of their physical counterparts which allows easy transfer of experiments between a simulation and the reality with little modification to code necessary thanks to the ROS interface utilized both by the physical and simulated devices.

[1] Gazebo architecture is also described in `https://gazebosim.org/docs/garden/architecture`.
[2] `http://wiki.ros.org/gazebo_ros_pkgs`

5.2 Experiment Set-Up

Figure 5.2: Experiment set-up in the Gazebo simulation.

Two simulated Tarot 650 (T650) drones are spawned inside a flat grassland Gazebo world. The follower is equipped with a 2-axis gimbal closely matching the one used in the subsequent real-world experiments in terms of its angular contraints and velocities. The camera mounted on the gimbal is a simulated Matrix-Vision Bluefox device with parameters modified to match the camera used in the real-world experiments in terms of maximal FPS, horizontal and vertical fields of view and output image resolution, see Section 3.2 for details.

Unless specified otherwise, all trajectories in the following experiments are generated from paths specified as points in the world frame. The utilized method of polynomial trajectory optimization by solving an unconstrained quadratic program implemented is part of the MRS UAV system presented in [63].

The goal of the experiments is to demonstrate and asses the ability of the solution implemented as part of this thesis to maintain distance from the leader without losing line of sight. In all the experiments in this chapter except the last one, the altitude of both the drones is kept constant to simplify analysis. Altitude control is tested specifically in Section 5.6.

5.3 Experiment I: Rectangular Trajectory

The first experiment has the leader follow a rectangular trajectory for 150 seconds. See the detailed path of both the leader and the follower in Figure 5.3. Velocity of the leader is limited to 2 m/s and kept variable as shown in Figure 5.4. Velocity of the follower is limited to 2 m/s as well. The width setpoint is set to 300 pixels in this experiment which corresponds to a distance of approximately 22 meters as shown in Figure 4.3. Once the leader arrives at its destination, the experiment runs for additional 10 seconds to confirm the position of the follower has stabilized.

5.3.1 Discussion

The follower does not lose track of the leader at any point during the pursuit and stabilizes the distance between the two UAVs at the end. The ground-truth distance between the drones is approximately inversely proportional to the target width, which again confirms that the width can be used to keep constant distance between the follower and the leader, see Figure 5.4.

Figure 5.3: Map of drones' paths including distances in the first experiment.

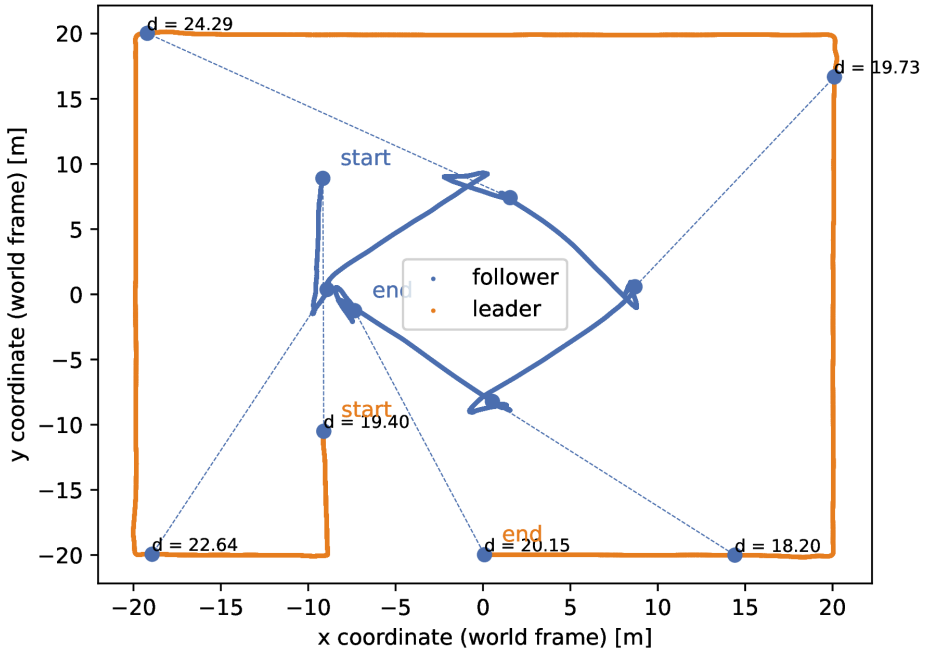
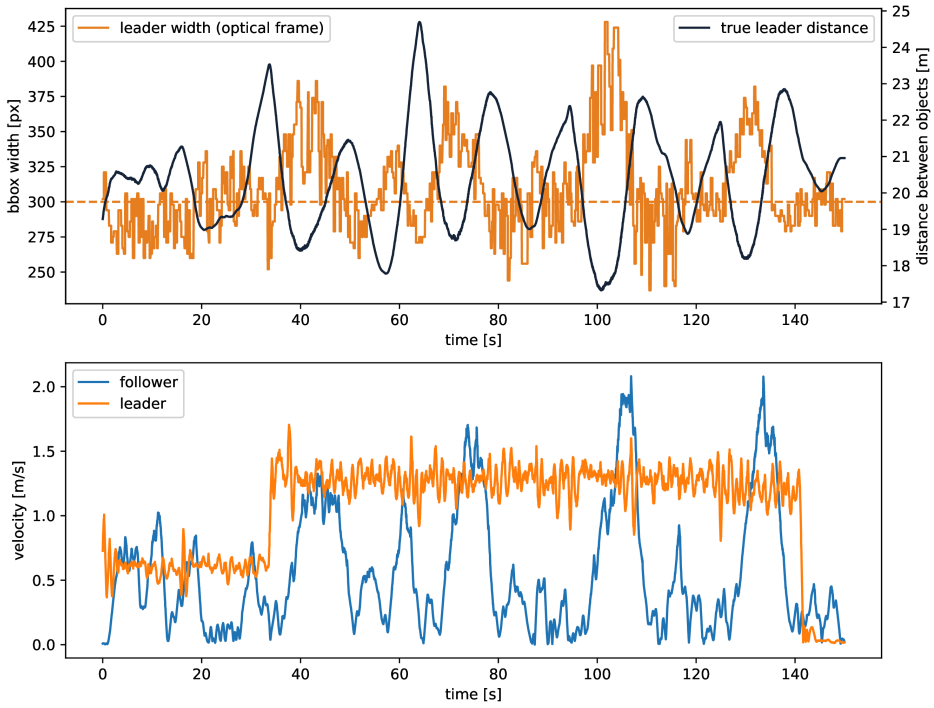


Figure 5.4: Results of the first experiment in simulation. Top: The bounding box's width (and its setpoint illustrated as a dashed line) and the distance between the leader and the follower in time. Bottom: Velocity of the leader and the follower in time.



A disadvantage of using width as the controlled variable is that the dependence of width on distance is not monotonous due to detector's noise. Thus there may be multiple

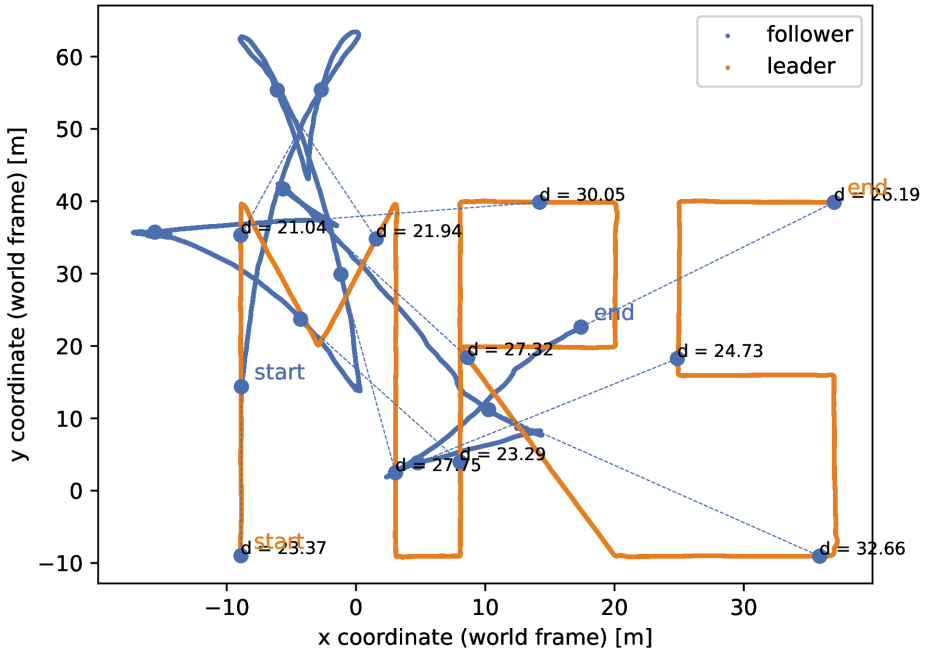
local optima among which the controller may oscillate as seen after $t = 140$ seconds, although the effect of noise is reduced by the Kalman filter.

The most challenging part of the flight is when the leader abruptly changes its direction and starts flying towards the follower since there is a delay in detection and PID controller's feedback. When the setpoint width is increased in such a situation to 400 pixels or more, which corresponds to the distance of approximately 18 meters, the drone does not have time to react and loses the target due to the appearance model's lower confidence on large targets as discussed in Section 4.2. Therefore, in the subsequent experiments, the target width is limited to fit this constraint, which will have to be addressed in further work.

The average distance between the leader and follower during the experiment is 20.41 meters and its standard deviation is 1.4 meters.

5.4 Experiment II: MRS-Shaped Trajectory

Figure 5.5: Second experiment map.



In this experiment, the follower tracks the leader flying for 300 seconds on a trajectory with various turns and twists. See map of the trajectory in Figure 5.5. The width setpoint is reduced to 250 pixels and gimbal movement is also recorded and analyzed as part of this experiment. Velocities of both the leader and the follower are limited to 3 m/s.

5.4.1 Discussion

As the drones are not exactly opposite to each other, there is a slight drift of the follower to its left at the beginning of the experiment due to the utilized navigational algorithm. Their velocities match closely as can be seen in Figure 5.7. At the end, the leader-follower distance stabilizes at slightly over 26 meters.

Figure 5.6: Gimbal movement heatmap during the second simulation experiment. The red dot represents where the gimbal is pointed before any joint rotation.

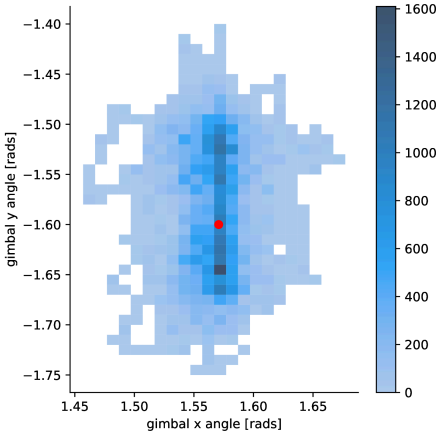
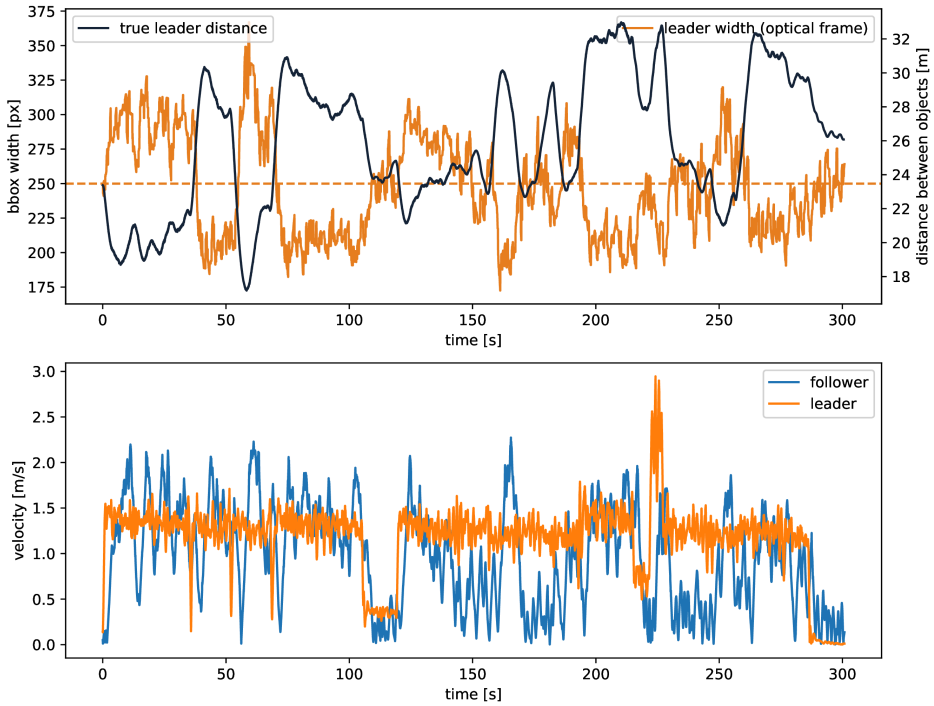


Figure 5.7: Second experiment velocity and distance.



Although the altitude of the drones is fixed for the duration of the experiment, there is significant movement of the gimbal on the pitch axis as shown in Figure 5.6. This is caused by the drone's tilt during acceleration and deceleration. When a drone accelerates, the front of the drone turns towards the ground, which causes its gimbal to turn upwards towards its target assuming it does not significantly change its position. The opposite happens during deceleration. This shows another benefit of a gimballed camera for this application, as it allows higher acceleration of the follower without losing the leader from line of sight.

The average distance between the leader and follower during the experiment is 25.85 meters and its standard deviation is 3.9 meters.

5.5 Experiment III: Random Trajectory

Figure 5.8: Third experiment map.

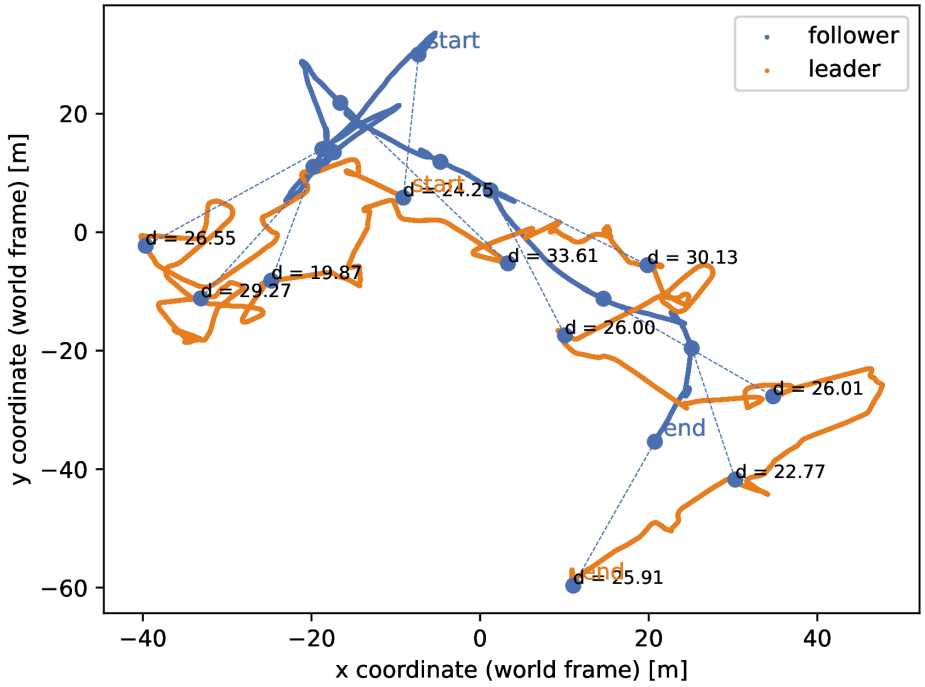
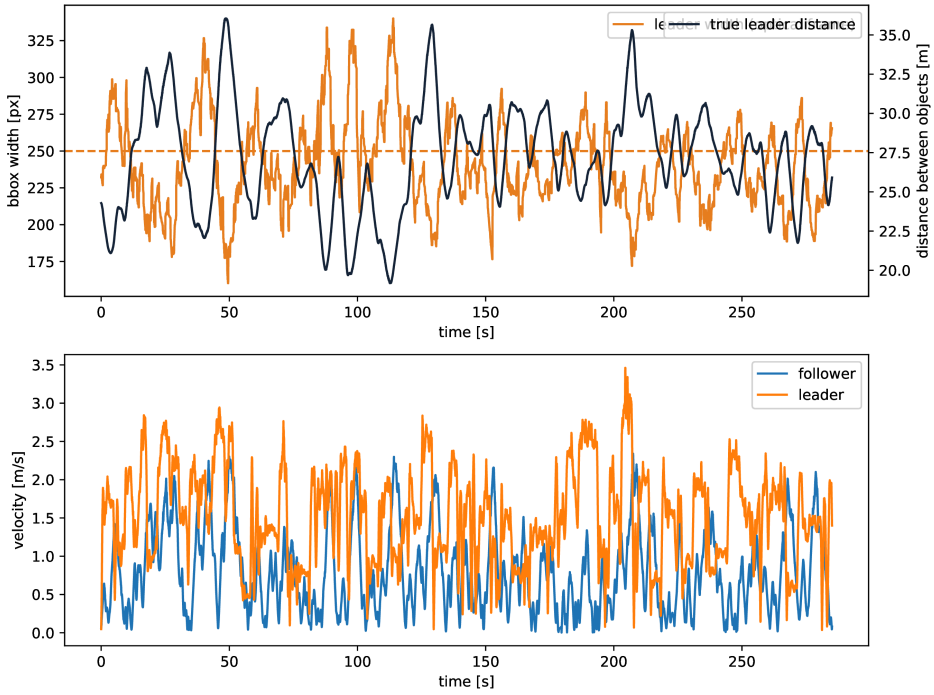


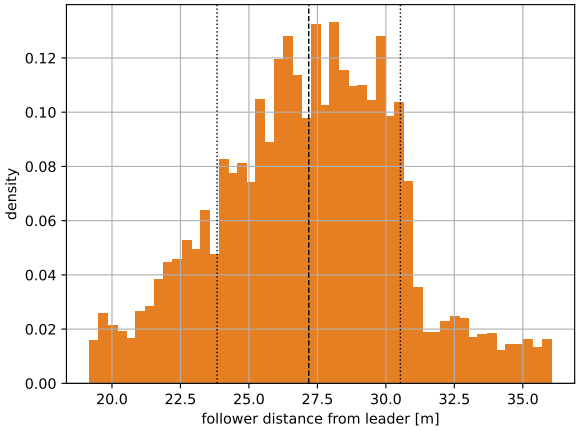
Figure 5.9: Third experiment velocity and distance.



In the third experiment, the leader behaves stochastically. A parameter is repeatedly

and uniformly generated from the interval $[0; 5]$ that controls for how long will the leader fly given the velocity vector $\begin{bmatrix} v_x & v_y \end{bmatrix}$ where each element is also generated uniformly from another interval $[-4; 4]$. The width setpoint is again set to 250 pixels.

Figure 5.10: Distance histogram with mean and standard deviation.



5.5.1 Discussion

The pursuit finishes at distance of approximately 26 meters as in the previous experiment. The mean distance is 27.2 meters with 3.3 meters standard deviation, see the distance distribution in Figure 5.10. This also matches the results of the previous experiment and the distance expectation given the width setpoint.

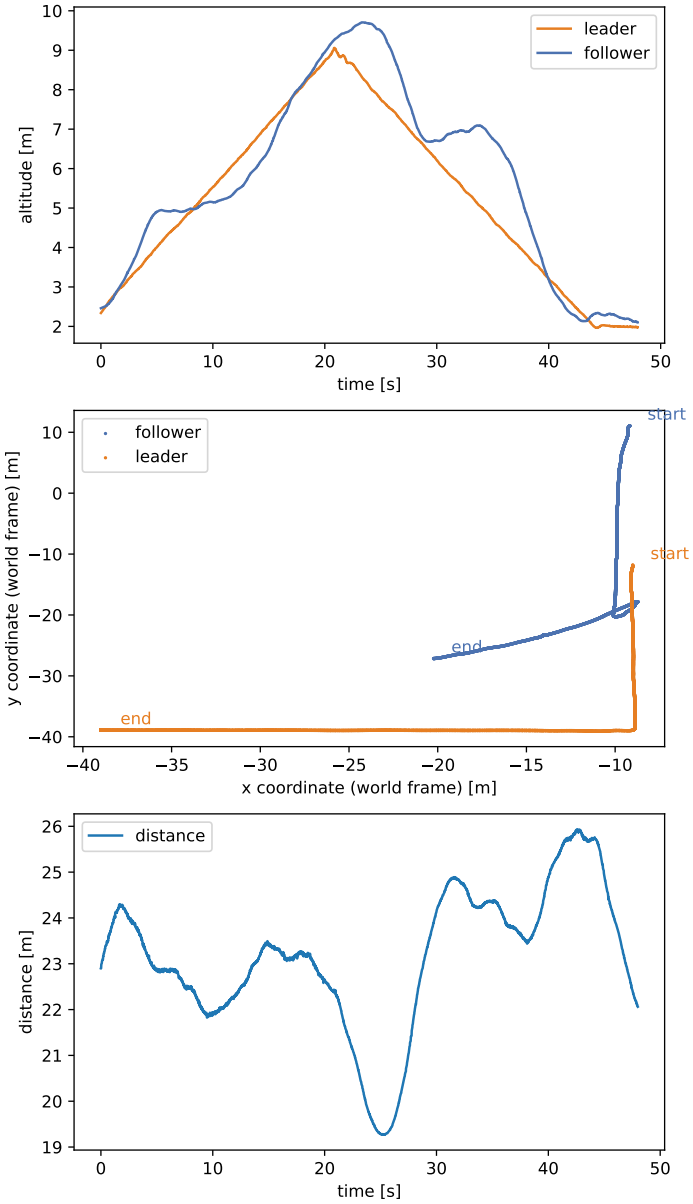
5.6 Experiment IV: Altitude Control

The system is tested on an L-shaped flight path where the leader starts at the height of 2 meters after takeoff and flies forwards and upwards to 9 meters on a slant trajectory before descending back to the initial height while changing its direction by 90 degrees. Altitude is controlled by the ε gimbal angle depicted in Figure 2.5. Both vertical and horizontal speeds are again limited to 2 m/s and the width setpoint is set to 300 pixels. See Figure 5.11 for the result of the experiment.

5.6.1 Discussion

The PID controller does not take into account the tilt of the drone, which might be the source of the disturbance at $t = 30$ seconds, which also happens shortly after the leader changes its direction of flight and causes the follower to accelerate aggresively. This should be addressed in a later version of the system. In the end, the follower stabilizes slightly above the initial height of 2 meters. Expected distance between the drones is also maintained which matches the results of the first experiment, where the same setpoint was used.

Figure 5.11: Results of the altitude control experiment. The follower maintains similar altitude and distance to the leader.



Chapter 6

Real-World Experimental Evaluation

6.1 Experiment Set-Up

Figure 6.1: A photo of the field in Temešvár, Czech Republic, where the experiments described in this section were conducted.

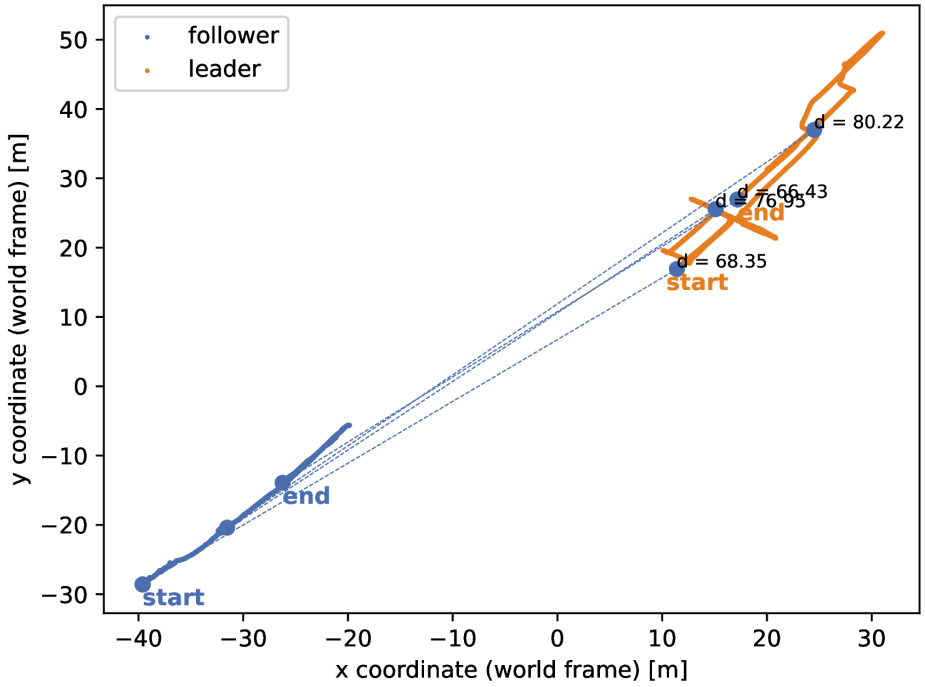
The drones used in the following experiments are described in Section 3.2. The leader is manually controlled by a pilot during the entire length of its flight. The follower's autonomy is gradually enabled for safety reasons. First, the gimbal's alignment with the target is tested by activating the gimbal manager, then the follower is allowed to turn around the vertical axis and lastly the flight manager is turned on, thus giving the drone full autonomy. Speed of the leader is initially constrained to 0.1 m/s and gradually increased up to over 6 m/s.

Before conducting the experiments, it was necessary to perform camera calibration in order to accurately estimate the camera parameters described in Section 2.3. This was done using multiple images of a checkerboard calibration pattern, which allows the algorithm to determine the intrinsic and extrinsic parameters of the camera. This process is essential for ensuring the accuracy of the results obtained from the experiments, as it allows the system to accurately interpret the visual data that the camera captures. See Section 2.3 for explanation of the estimated parameters.

After adjusting the gimbal manager's code to use the SimpleBGC API, the system works out of the box on the drone with the same code as for the simulations described in the previous section, except for the altitude control, which was not tested due to a mistake in the system discovered during the experiments. A video from the experiment is available at `https://youtu.be/yzDNcZyWY3c`.

6.2 Experiment I: Straight Pursuit

Figure 6.2: Map of the paths the leader and the follower took in the first experiment.



The first experiment was designed to evaluate the system on a short trajectory including several turns to test the gimbal manager node's function. Near the end at $t = 330$ seconds, the leader begins to fly towards the follower after changing its direction of flight abruptly, see Figure 6.3.

6.2.1 Discussion

The leader was successfully detected after takeoff ($t = 50$ seconds) and followed for over 300 seconds. There were multiple false positive detections during the pursuit which manifest as abrupt changes in the width of the bounding box as can be seen in Figure 6.3. When the background contains forests, the detector sometimes detects clusters of branches as drones with a higher confidence than the actual drone, as shown in Figure 6.4. This may be due in part to the model's tendency to have higher confidence in smaller target detections, as shown in Figure 4.3. The false positive has little effect on the flight itself in this experiment as the system quickly recovers and redetects the drone correctly. Another difference compared to the experiments in simulation is that the reaction time of the real drone is significantly slower due to the camera itself having a measured delay of 0.5 seconds compared to 0.15 seconds in the simulation before an image is presented to ROS.

Figure 6.3: Results of the first real-world experiment. Top: Bounding box width and the ground-truth leader-follower distance over time. Bottom: Velocities of the two drones. There are multiple visible false positives in the graph showing the width of the target bounding box at around 200 and 310 seconds. They do not appear to influence the PID controller significantly and the target is redetected soon afterwards.

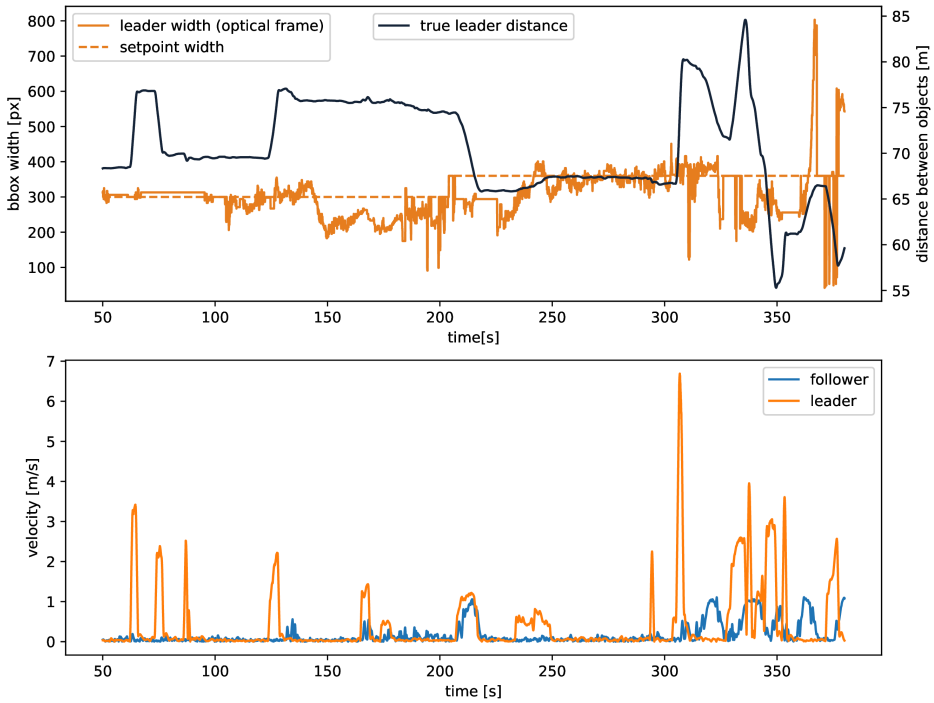
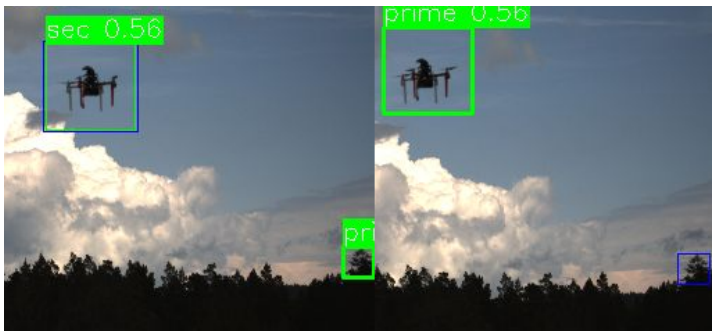
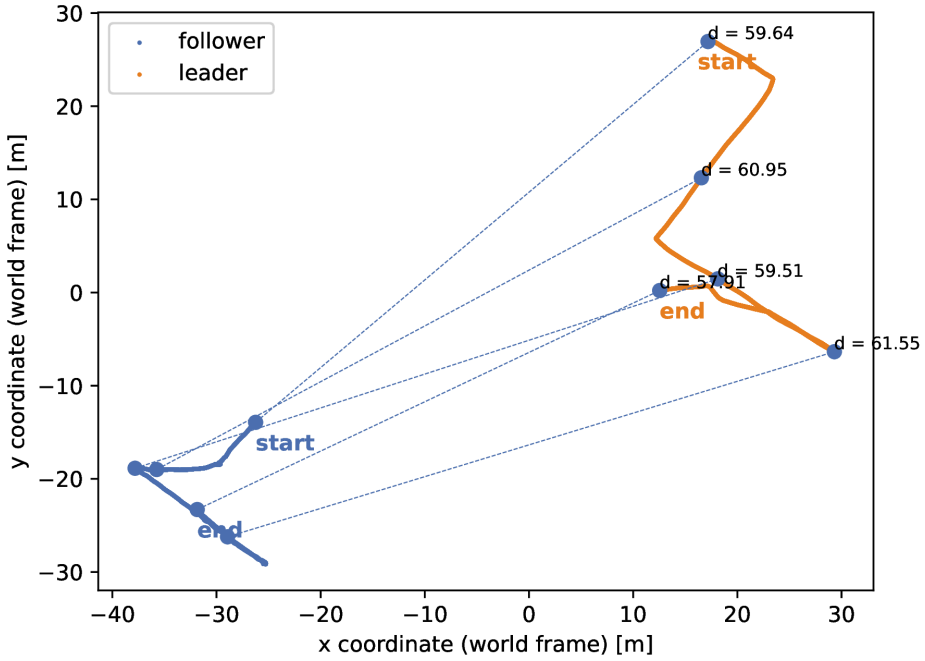


Figure 6.4: Tree branches wrongly classified as a drone. The drone is still detected correctly, but the model has lower confidence in the correct detection. In the next frame, the drone is correctly localized again.



6.3 Experiment II: Escape Manouver

Figure 6.5: The map shows the paths taken during the second experiment.



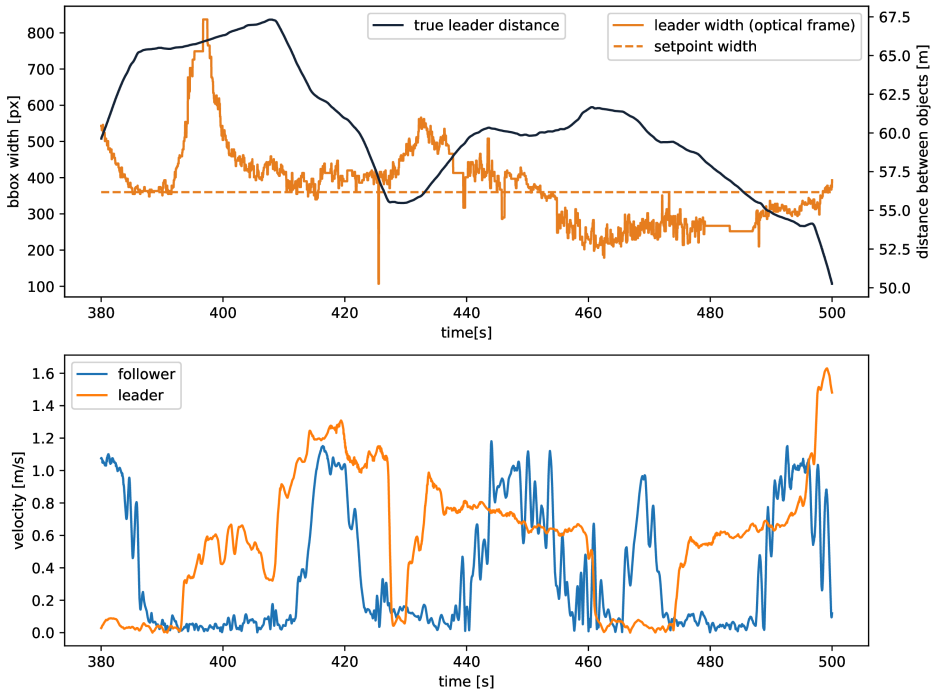
The second experiment continues after the first one stopped. The leader continues flying towards the follower and then quickly turns to the left. This maneuver is repeated twice. See the map of the experiment in Figure 6.5.

6.3.1 Discussion

The follower keeps up with the leader and the gimbal tracks him successfully to the landing spot.

In the second plot of Figure 6.6, it is shown that first the follower accelerates backwards to reduce the distance between the two drones. This maneuver is successful, as the desired width of the bounding box is reached. At approximately $t = 395$ seconds, there is a false detection of a tree in the background, causing the follower to start moving backwards and further increase the distance between the drones. The next detection is correct again. At $t = 410$ seconds, the leader starts moving towards the follower, which responds by accelerating to correct the distance 2 seconds later. At $t = 423$ seconds, there is the second false detection, which is again quickly corrected.

Figure 6.6: The width of the target's bounding box is depicted in the first plot, and the leader's and follower's velocities are displayed in the second.



Chapter 7

Conclusion

A system for drone tracking using a gimballed RGB camera placed onboard a flying UAV was proposed, implemented, and tested both in Gazebo simulations and real-world experiments. Various models for object detection and tracking were combined to reach an accurate feedback system capable of running online with the limited onboard computational capabilities of a UAV. Two large drone datasets MIDGARD and Det-Fly were fused and used for model training and evaluation, as a result of which the YOLO5 neural network architecture was determined to be an optimal compromise between speed and accuracy, with a MOSSE visual tracking algorithm updating the target location in-between the object predictions. The method of genetic programming was utilized to optimize the hyperparameters of the chosen best-performing architecture to further improve its accuracy. The proposed object detection and tracking method achieved sufficient online prediction performance on both a simulated and real drone.

To keep the gimbal centered on the target, the method was proposed where a 3D ray is calculated from the camera's optical center to the target's position in the image plane. The gimbal is then rotated by the angle difference between the current gimbal orientation and the estimated 3D ray.

To map the target location hypothesis to a movement that maintains the distance between the two drones, a simple strategy was implemented. The drone is turned towards the target based on stored odometry data from the time the image corresponding to the current detection was taken. Two PID controllers are used to correct the forward velocity and altitude of the drone, based on the predicted width of the target in pixels and the corresponding gimbal orientation. It was demonstrated in real-world experiments that, despite some noise in the bounding box regression and camera sensor that can cause the width of an object to not be directly inversely proportional to its distance, it is nevertheless suitable input for controlling the mutual distance. To further reduce noise, a Kalman filter is applied to the width hypotheses before they are passed to the controllers.

The main limitations of the implemented system are related to the detection model, specifically its lower confidence on larger and or closer drones and occasional false positive detection when there are objects such as tree branches in the background. This means human supervision during the autonomous flight is still necessary to correct the system's errors by e.g. manually tweaking the confidence threshold for the given environment and confirming the plans of the control system. Nevertheless, as part of this thesis, the suitability of the proposed method for various UAV tracking applications was demonstrated by an approximately 500 seconds long uninterrupted autonomous flight.

7.1 Further Work

The detector's low detection accuracy on drone images having a width larger than 400 pixels is an issue that could be tackled by adding static close-up photos of drones to the dataset. This could help improve the performance of the larger models by providing them with more complex training data.

To further improve close target tracking, a LIDAR system can be added to the drone and its output can be fused with the visual detector. This combination can provide a more comprehensive tracking system that is more accurate and robust under different lighting and weather conditions. The LIDAR data can provide a high-resolution 3D mapping of the environment, which can be used to track the positions of objects more precisely. While adding a LIDAR system to the drone and fusing its output with the visual detector can improve the tracking performance, it also increases the complexity and cost of the solution. The LIDAR system itself is likely to be more expensive than the visual detector, and integrating the two systems may require additional hardware and software development. Additionally, the increased computational demands of combining the two systems may require a more powerful processor, which can also add to the overall cost of the solution.

Different guidance strategies might also be compared such as proportional navigation which uses the velocity vector of the target in addition to its relative position to control the follower drone's motion with the goal of keeping it on the path for interception.

Chapter 8

References

[1] P. Fahlstrom and T. Gleason, *Introduction to UAV Systems* (Aerospace Series). Wiley, 2012, ISBN: 9781118396810. [Online]. Available: https://books.google.cz/books?id=uLsNtm99IWYC.

[2] M. A. Siddiqi, C. Iwendi, K. Jaroslava, and N. Anumbe, "Analysis on security-related concerns of unmanned aerial vehicle: Attacks, limitations, and recommendations," *Mathematical Biosciences and Engineering*, vol. 19, no. 3, pp. 2641–2670, 2022, ISSN: 1551-0018. DOI: 10.3934/mbe.2022121. [Online]. Available: https://www.aimspress.com/article/doi/10.3934/mbe.2022121.

[3] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407. DOI: 10.1109/ICRA.2011.5979561.

[4] V. Walter, N. Staub, A. Franchi, and M. Saska, "UVDAR system for visual relative localization with application to leader–follower formations of multirotor UAVs," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2637–2644, 2019. DOI: 10.1109/LRA.2019.2901683.

[5] N.-D. Nguyen, T. Do, T. Duc, and D.-D. Le, "An evaluation of deep learning methods for small object detection," *Journal of Electrical and Computer Engineering*, vol. 2020, pp. 1–18, Apr. 2020. DOI: 10.1155/2020/3189691.

[6] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," Apr. 2018, arXiv:1804.02767.

[7] Y. Zheng, Z. Chen, D. Lv, Z. Li, Z. Lan, and S. Zhao, "Air-to-air visual detection of micro-UAVs: An experimental evaluation of deep learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1020–1027, 2021.

[8] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO series in 2021," 2021. arXiv: 2107.08430 [cs.CV].

[9] D. Xun, Y. Lim, and S. Srigrarom, "Drone detection using YOLOv3 with transfer learning on NVIDIA jetson TX2," Jan. 2021, pp. 1–6. DOI: 10.1109/ICA-SYMP50206.2021.9358449.

[10] T. Li, Y. Ma, and T. Endoh, "A systematic study of tiny YOLO3 inference: Toward compact brainware processor with less memory and logic gate," *IEEE Access*, vol. 8, pp. 142 931–142 955, 2020.

[11] S. Hożyń and M. Wierszyło, "Tracking of unmanned aerial vehicles using computer vision methods: A comparative analysis," *Scientific Journal of Polish Naval Academy*, vol. 223, pp. 39–51, Dec. 2020. DOI: 10.2478/sjpna-2020-0014.

[12] A. Lukežič, T. Vojíř, L. Čehovin Zajc, J. Matas, and M. Kristan, "Discriminative correlation filter with channel and spatial reliability," *International Journal of Computer Vision*, vol. 126, Jul. 2018. DOI: 10.1007/s11263-017-1061-3.

[13] B. Babenko, M.-H. Yang, and S. Belongie, "Visual tracking with online multiple instance learning," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 983–990. DOI: 10.1109/CVPR.2009.5206737.

[14] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, "Visual object tracking using adaptive correlation filters," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2544–2550. DOI: 10.1109/CVPR.2010.5539960.

[15] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015. DOI: 10.1109/TPAMI.2014.2345390.

[16] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek, "A brief introduction to OpenCV," in *2012 Proceedings of the 35th International Convention MIPRO*, 2012, pp. 1725–1730.

[17] J. Fan, W. Xu, Y. Wu, and Y. Gong, "Human tracking using convolutional neural networks," *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1610–1623, 2010. DOI: 10.1109/TNN.2010.2066286.

[18] Z. Cui, S. Xiao, J. Feng, and S. Yan, "Recurrently target-attending tracking," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1449–1458. DOI: 10.1109/CVPR.2016.161.

[19] P. Li, D. Wang, L. Wang, and H. Lu, "Deep visual tracking: Review and experimental comparison," *Pattern Recognition*, vol. 76, pp. 323–338, 2018, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2017.11.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320317304612.

[20] R. Barták and A. Vykovský, "Any object tracking and following by a flying drone," in *2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*, 2015, pp. 35–41. DOI: 10.1109/MICAI.2015.12.

[21] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-learning-detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1409–1422, 2012. DOI: 10.1109/TPAMI.2011.239.

[22] X. Liu, Y. Yang, C. Ma, J. Li, and S. Zhang, "Real-time visual tracking of moving targets using a low-cost unmanned aerial vehicle with a 3-axis stabilized gimbal system," *Applied Sciences*, vol. 10, no. 15, 2020, ISSN: 2076-3417. DOI: 10.3390/app10155064. [Online]. Available: https://www.mdpi.com/2076-3417/10/15/5064.

[23] S. A. Murtaugh and H. E. Criel, "Fundamentals of proportional navigation," *IEEE Spectrum*, vol. 3, no. 12, pp. 75–85, 1966. DOI: 10.1109/MSPEC.1966.5217080.

[24] M. Vrba and M. Saska, "Marker-less micro aerial vehicle detection and localization using convolutional neural networks," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2459–2466, 2020. DOI: 10.1109/LRA.2020.2972819.

[25] X. Zou, "A review of object detection techniques," in *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, 2019, pp. 251–254. DOI: 10.1109/ICSGEA.2019.00065.

[26] W. Mcculloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[27] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, pp. 533–536, 1986. DOI: 10.1038/323533a0. [Online]. Available: http://www.nature.com/articles/323533a0.

[29] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[30] R. E. Schapire, "A brief introduction to boosting," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'99, Stockholm, Sweden: Morgan Kaufmann Publishers Inc., 1999, 1401–1406.

[31] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010. DOI: 10.1109/TPAMI.2009.167.

[32] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[33] R. Maisano, V. Tomaselli, A. Capra, F. Longo, and A. Puliafito, "Reducing complexity of 3D indoor object detection," Dec. 2018. DOI: 10.1109/RTSI.2018.8548514.

[34] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013, cite arxiv:1206.5538. [Online]. Available: http://arxiv.org/abs/1206.5538.

[35] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra, "Reducing overfitting in deep networks by decorrelating representations," Nov. 2015.

[36] G. Lewicki and G. Marino, "Approximation by superpositions of a sigmoidal function," *Appl. Math. Lett.*, vol. 17, pp. 1147–1152, Dec. 2004. DOI: 10.1016/j.aml.2003.11.006.

[37] M. O. Khairandish, M. Sharma, V. Jain, J. Chatterjee, and N. Zaman, "A hybrid CNN-SVM threshold segmentation approach for tumor detection and classification of MRI brain images," *IRBM*, vol. 43, Jun. 2021. DOI: 10.1016/j.irbm.2021.06.003.

[38] R. Girshick, "Fast R-CNN," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448. DOI: 10.1109/ICCV.2015.169.

[39] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, 2013. DOI: 10.1007/s11263-013-0620-5. [Online]. Available: http://www.huppelen.nl/publications/selectiveSearchDraft.pdf.

[40] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, Jun. 2015. DOI: 10.1109/TPAMI.2016.2577031.

[41] X. Lu, B. Li, Y. Yue, Q. Li, and J. Yan, "Grid R-CNN," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 7355–7364. DOI: 10.1109/CVPR.2019.00754.

[42] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "CenterNet: Keypoint triplets for object detection," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 6568–6577. DOI: 10.1109/ICCV.2019.00667.

[43] M. Carranza-García, J. Torres-Mateo, P. Lara-Benítez, and J. García-Gutiérrez, "On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data," *Remote Sensing*, vol. 13, p. 89, Dec. 2020. DOI: 10.3390/rs13010089.

[44] H. Law and J. Deng, "CornerNet: Detecting objects as paired keypoints," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[45] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[46] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[47] G. Jocher, *ultralytics/yolov5: v3.1 - Bug Fixes and Performance Improvements*, https://github.com/ultralytics/yolov5, version v3.1, Oct. 2020. DOI: 10.5281/zenodo.4154370. [Online]. Available: https://doi.org/10.5281/zenodo.4154370.

[48] C.-Y. Wang, H.-Y. Mark Liao, Y.-H. Wu, P.-Y. Chen, J.-W. Hsieh, and I.-H. Yeh, "CSPNet: A new backbone that can enhance learning capability of CNN," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2020, pp. 1571–1580. DOI: 10.1109/CVPRW50498.2020.00203.

[49] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, *Path aggregation network for instance segmentation*, 2018. arXiv: 1803.01534 [cs.CV].

[50] T. Ridnik, H. Lawen, A. Noy, and I. Friedman, "Tresnet: High performance gpu-dedicated architecture," *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1399–1408, 2020.

[51] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *Lecture Notes in Computer Science*, 346–361, 2014, ISSN: 1611-3349. DOI: 10.1007/978-3-319-10578-9_23. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-10578-9_23.

[52] S. Theodoridis and K. Koutroumbas, *Pattern Recognition, Fourth Edition*. Academic Press, 2009, ISBN: 9781597492720.

[53] M. A. Haseeb, "DisNet: A novel method for distance estimation from monocular camera," 2018.

[54] *OpenCV: Camera calibration and 3D reconstruction*, https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html, Accessed: 2020-09-30.

[55] M. Abdo, A. Reza, A. Toloei, and M. Arvan, "Research on the cross-coupling of a two axes gimbal system with dynamic unbalance," *International Journal of Advanced Robotic Systems*, vol. 10, p. 1, Oct. 2013. DOI: 10.5772/56963.

[56] H. P. Gavin, "The levenberg-marquardt method for nonlinear least squares curve-fitting problems c ©," 2013.

[57] V. Walter, M. Vrba, and M. Saska, "On training datasets for machine learning-based visual relative localization of micro-scale uavs," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 10 674–10 680. DOI: 10.1109/ICRA40945.2020.9196947.

[58] G. Welch and G. Bishop, "An introduction to the Kalman filter," University of North Carolina at Chapel Hill, Chapel Hill, NC, USA, Tech. Rep. 95-041, 1995. [Online]. Available: http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html.

[59] M. Johnson, M. Moradi, J. Crowe, *et al.*, *PID control: New identification and design methods*. Jan. 2005, pp. 1–543. DOI: 10.1007/1-84628-148-2.

[60] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *Journal of Dynamic Systems Measurement and Control-transactions of The Asme*, vol. 115, pp. 220–222, 1942.

[61] K. He, R. Girshick, and P. Dollar, "Rethinking ImageNet pre-training," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 4917–4926. DOI: 10.1109/ICCV.2019.00502.

[62] T. Gwiazda, *Genetic Algorithms Reference Volume 2 Mutation Operator for Numerical Optimization Problems* (Genetic Algorithms Reference 2. c.). Simon & Schuster, 2007, ISBN: 9788392395843. [Online]. Available: https://books.google.com.tr/books?id=O1FVGQAACAAJ.

[63] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*, Springer, 2016, pp. 649–666.

Chapter A

Appendix A: Attachment Contents

Names of root directories on the attached CD storage are listed in Table A.1.

File name	Description
thesis.pdf	Master thesis in PDF format.
thesis_sources	Latex code of the thesis.
code	Source code of the implemented system.

Table A.1: Contents of the attached CD storage.