CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS

Martin Křížek

Object Count and Density Estimation for Relative Localization in Swarms of Unmanned Aerial Vehicles

May 2023

Department of Computer Science

Thesis supervisor: **Ing. Matouš Vrba**

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení:	**Křížek**	Jméno: **Martin**	Osobní číslo: **483479**
Fakulta/ústav:	**Fakulta elektrotechnická**		
Zadávající katedra/ústav:	**Katedra počítačů**		
Studijní program:	**Otevřená informatika**		
Specializace:	**Umělá inteligence**		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Odhadování počtu a hustoty objektů pro relativní lokalizaci v rojích bezpilotních letounů

Název diplomové práce anglicky:

Object Count and Density Estimation for Relative Localization in Swarms of Unmanned Aerial Vehicles

Pokyny pro vypracování:

Proveďte rešerši rojových řídicích algoritmů, metod pro relativní lokalizaci bezpilotních letounů v rojích a vizuálních detektorů.
Navrhněte metodu biologicky inspirované bezznačkové vizuální relativní lokalizace bezpilotních letounů, vhodné pro stabilizaci roje.
Tato metoda musí být použitelná na bezpilotním letounu s omezenou nosností a výpočetní kapacitou.
Porovnejte navrženou metodu se State of the Art řešeními, jako například s detektorem využívajícím konvoluční neuronovou síť.
Implementujte a otestujte algoritmus rojového řízení využívající bezznačkovou vizuální relativní lokalizaci, otestujte ho v realistické simulaci s velkým počtem členů roje a porovnejte ho se stávajícími metodami.

Seznam doporučené literatury:

1. M. Vrba and M. Saska, "Marker-Less Micro Aerial Vehicle Detection and Localization Using Convolutional Neural Networks," Robotics and Automation Letters 5(2):2459-2466, 2020.
2. Herbert-Read, James E, "Understanding how animal groups achieve coordinated movement," Journal of Experimental Biology 219, no. 19: 2971-2983, 2016.
3. Ilyas, Naveed, Ahsan Shahzad, and Kiseon Kim, "Convolutional-neural network-based image crowd counting: Review, categorization, analysis, and performance evaluation," Sensors 20, no. 1: 43, 2019.
4. Pavel Petráček, Viktor Walter, Tomáš Báča and Martin Saska, "Bio-Inspired Compact Swarms of Unmanned Aerial Vehicles without Communication and External Localization," Bioinspiration & Biomimetics 16(2):026009, December 2020.

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: **31.01.2023**   Termín odevzdání diplomové práce: **26.05.2023**

Platnost zadání diplomové práce: **22.09.2024**

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

Author statement:

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, date............................ ....

Signature

Acknowledgements

I would like to thank my thesis supervisor Ing. Matouš Vrba for his great advice on this thesis. I am wholeheartedly grateful to my dear girlfriend Han for her regular quality consultations and on-point recommendations. Many thanks also belong to my friends and family for their ever-optimistic attitude.

Abstract

The objective of this thesis is to design and test a markerless visual method for relative localization as well as a compatible control system for unmanned aerial vehicles within large swarms. A convolutional neural network is employed for the relative localization. The control system is adapted from the boids principles to integrate with the proposed relative localization system. The performance of the relative localization system is tested on both synthetic and real data with positive results. A simulation is used to prove the capability of the proposed relative localization system to be used as the source of neighbor locations together with the proposed control system for large-scale swarm stabilization. The proposed method of relative localization is compared to and outperformed an existing alternative state-of-the-art approach that utilizes a hypothetical perfect object detector.

Keywords

UAV, drone, swarm, neural network, markerless relative localization, boids

Abstrakt

Cílem této práce je je návrh a otestování metody pro bezznačkovou vizuální relativní lokalizaci a kompatibilního řídícího systému pro bezpilotní autonomní helikoptéry v rojích. Metoda pro relativní lokalizaci využívá konvoluční neuronovou síť. Řídící systém je adaptací boidů, která je integrovatelná s navrhovanou relativní lokalizací. Navržená metoda relativní lokalizace byla otestována na syntetických i reálných datech s pozitivními výsledky. Simulace byla využita k ukázání, že navržená metoda relativní lokalizace je použitelná jako zdroj lokací sousedů v navrženém rojovém řídícím systému pro stabilizaci velkých rojů. Navržená metoda relativní lokalizace byla porovnána a překonala existující alternativu, která využívá hypotetický perfektní detektor.

Klíčová slova

UAV, dron, roj, neuronová síť, bezznačková relativní lokalizace, boids

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis focuses on swarming of Unmanned Aerial Vehicles (UAVs). A UAV is an airborne vehicle that does not have a pilot onboard and is considered to navigate fully autonomously even though in many applications, UAVs are controlled remotely by an operator. UAVs used for the purposes of this thesis are quadcopters. Use of such multicopters is widespread due to their versatility and low operating costs. Some of the possible applications include search and rescue missions [1, 2, 3], monitoring [4] or film making [5].

Figure 1.1: Large-scale swarm of UAVs generated using the approach introduced in [6].

Although the usage of single-operating UAVs is widespread, it has its limitations. Typically, a UAV has a limited battery capacity and thus a limited flight time which is a problem for large-scale monitoring tasks. A further example where a single UAV can fail is large object transportation because of its limited thrust. We can use swarms of UAVs to compensate for physical limits that arise when using a single UAV. A swarm in this context is a group of UAVs that acts autonomously and is decentralized, each member has only local sensing and the members cooperate to achieve a given task (e.g. navigate between locations). Swarms of UAVs are often bio-inspired [7]. Swarms of UAVs can address those limits by coordinating and dividing the main task into subtasks that can be solved by individual swarm members. Such a solution may include using formation control within swarms to divide an area to be monitored into smaller areas that can be surveyed by the individual swarm members or organizing more UAVs to simultaneously carry heavy loads meant for transportation). Swarms of UAVs have a variety of applications thanks to the above-mentioned advantages. They can be deployed in search and rescue missions in an inaccessible terrain to search for potential survivors [1, 2]. They can aid with the safety of people by assisting with firefighting [8, 9, 10]. Surveillance is another area where swarms of UAVs can be utilized [11, 12]. They can also contribute by measuring air pollutants within a city [13]. Swarms of drones can also help with more efficient agriculture [14, 15, 16].

To carry out the above-mentioned tasks, it is crucial for UAVs to safely navigate within the swarm. For this, each UAV needs to have some kind of estimate of relative locations of its neighbors. The goal of this thesis is to develop a relative localization approach for low to high density swarms of UAVs that will provide enough information about the neighboring UAVs so that the entire swarm is stable and navigates coherently. It is necessary for this relative localization method to be lightweight in order to run online on the UAV's onboard computer.

1.1 State of the art

There is a multitude of methods that are used for relative localization of UAVs. One of them is the UVDAR visual method [17, 18]. UVDAR employs ultraviolet LED markers and onboard cameras that are sensitive to wavelengths emitted by the LED markers. A known spatial configuration of the markers is then used to determine the relative location of the neighboring robot.

A vision-based marker-less approach introduced in [19] employs a deep neural network object detector trained on images acquired from a single camera and a tracker to locate the UAV within the image. A keypoint extractor neural network is then used to find the keypoints of the UAV detected in the previous stage. The keypoints in combination with optimization algorithms are then used to estimate the 6-DoF pose of each UAV in the image.

Another example of a marker-less approach for detection of the UAVs is presented in [20]. Where the output of the depth camera is subjected to thresholding, contour detection, filtering, and contour grouping with projection to yield the 3D position of the detected UAV. This approach is also fit for online deployment on the onboard computer.

Further marker-less method introduced in [21] is analyzing the possibility of using a combination of 3D LiDAR and RADAR for the detection of UAVs. However, these systems primary goal is long-range detection and it is achieved by employing sizable stationary ground sensors. Therefore, this solution is not suitable for our task.

One of the non-visual methods is an Ultra-Wideband-based (UWB) approach [22] which utilizes UWB ranging in addition to communication and performs consensus-based fusion to estimate the relative location of the UAVs within the swarm without any further infrastructure.

The main task of this thesis is similar to the crowd counting tasks [23, 24] as we also consider large amounts of UAVs in a single image and rather than determining the exact location of each of the neighboring UAVs, the presented method estimates their general distribution in 3D space (similarly to 2D crowd counting heatmap as shown in Figure 1.2). This approach is based on the behavior of animal groups described in [25] where it can be observed that those animals are coordinating based on the density distribution of their neighbors in their surroundings.

The crowd counting approach presented in [26] utilizes a multi-column convolutional neural network (CNN) architecture with the neural network divided into multiple parallel feature detectors that connect at the end, combining the acquired feature maps into a final density map. However, the task of this thesis does not include as large amounts of objects per image as in [26] which is a problem since it is crucial to have a precise estimate even with a low amount of close objects as this is critical for collision avoidance.

The problem of estimating both large and significantly smaller groups of objects in an image using the same network is partially addressed in [27]. The authors propose a relative-count training loss function in addition to the commonly employed density map loss. This approach performs decently even with a scene containing fewer objects.

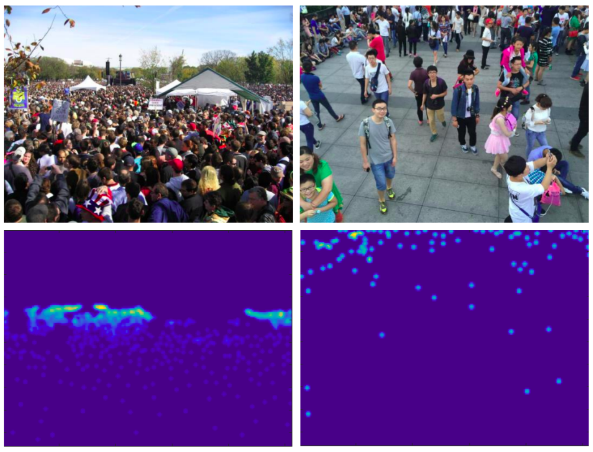


Figure 1.2: Example of an input image and the corresponding output density map of human heads taken from [27].

An approach[1] for counting sea lions and categorizing them based on their age and sex shows that even a relatively simple regression-based model can outperform its competition in an object counting and categorizing problem that is heavily dependent on a sense of the scale (rather than the shape) of the objects being counted. This task is similar to our problem of distance estimation in swarms of UAVs since the main difference between a close and a far UAV is also the size rather than the shape.

One subtask of this thesis is to estimate the distribution of the UAVs in the image over distance. There are existing techniques that solve similar distance-related problems. Approaches that produce a full depth map like [28] generate a lot of redundant information for the application introduced in this thesis because they estimate the depth information for the entire image whereas we require an estimate only for one specific type of object, the UAVs. Similarly, solutions based on various object or position detectors are introduced in [29, 30, 31, 32, 33]. However, detecting and estimating the relative pose of individual objects does not scale well to large numbers of targets [27] and are overly complex for swarm stabilization. On the other hand, we aim to estimate the overall density distribution of the swarm rather than object-wise information.

[1]https://www.kaggle.com/code/sardinipasta/kera-sea-lion-count

Furthermore, this thesis also aims to develop a bio-inspired swarm controller based on the Boids principles [34] compatible with the proposed relative localization system. One bio-inspired approach is presented in [7] where the UAVs are swarming without any explicit communication in a decentralized manner. This approach utilizes the UVDAR [18] relative localization system and is tested in real-world experiments.

The system introduced in [35] also uses Boids-like control rules. It shows that such a control mechanism is capable of stabilizing a large swarm of UAVs with large velocities in real-world conditions. Experiments were carried out with a flock of 30 UAVs and included object avoidance.

The suitability of Boids-like rules for swarm control is further demonstrated in [36] where such rules are used to navigate a decentralized swarm of UAVs without communication or any global external localization system. This approach was designed for UAVs equipped with means for visual relative localization.

The approach proposed in this thesis employs a convolutional neural network (CNN) for marker-less relative localization. The CNN is trained to estimate the distribution of the UAVs in discretized 3D space from an onboard camera image in the form of UAV density. The relative localization runs in real-time, it is independent of tilting and various perspectives of the onboard camera, and its resolution can be modified. It can also be used with different camera parameters without retraining the whole model. Furthermore, a decentralized control system based on modified Boids principles is proposed that can be integrated with the proposed relative localization system to stabilize the swarm without any communication or use of global navigation system.

1.2 Problem statement

The task is to develop a convolutional neural network that will provide information about the density of objects in the 3D space in an RGB image and to introduce a swarm controller capable of stabilizing the swarm of UAVs while using the output of the neural network as the only source of relative localization of neighboring UAVs. The presented solution is inspired by sensing and behavior of animals that navigate in herds, swarms, and flocks [25].

The input of this network is an RGB image from an onboard camera of a UAV that is a part of the swarm. The output is an estimate of the spatial distribution of other swarm members in the image and over the distance. The total count of the UAVs, as well as partial sums over the chosen depth and image regions, can be extracted from the output of the network.

We assume a swarm of UAVs where each of the UAVs is equipped with RGB cameras that are able to cover the surroundings of the given UAV. We do not consider any global navigation system or any inter-UAV communication. We suppose that all of the UAVs are of similar shape and size, however, the colors or color patterns can differ. Furthermore, we assume application both with large and small groups of UAVs. We also require the method to be fast enough for onboard deployment.

Chapter 2

Network Architecture

Firstly, we will introduce the individual components within the neural network and their respective purposes within the architecture. Further, we will describe how are those components combined in the proposed neural network architecture and how to interpret the output format of the network.

2.1 Layers of the Neural Network Architecture

This subsection describes the building blocks of the neural network. A convolutional neural network was chosen for the UAV density estimation because the estimate is inferred from an image. This section is describing the concept of convolutional neural network method and its components. A typical feedforward neural network consists of a sequence of layers. Each layer represents a specific function that processes the data within the network. Firstly, the input layer obtains the output of the network (e.g. an image in our case) and passes it over to the next layer. After the input layer follows a sequence of hidden layers. Each hidden layer gets its input from the previous layer and passes its output to the next. Each layer can represent a different operation and their goal is to extract information and complex patterns important for the task. The last layer is the output layer which takes as input the output of the last hidden layer and transforms it into the output format of the network (e.g. UAV density distribution in our case) [37, 38].

2.1.1 Convolutional Layer

The input of the convolutional layer is a $h_{in} \times w_{in} \times c_{in}$ tensor of real numbers. It typically represents an image (or a collection of feature maps) where h_{in} and w_{in} are the height and width of the image respectively and c_{in} represents the number of channels (e.g. red, green, and blue channels of an RGB image).

The main attribute of convolutional neural networks is the usage of convolutional kernels. Such a kernel is typically represented as a square $n \times n$ matrix with values representing weights. The kernel is used to perform the convolution operation on the layer's input in a sliding-window manner according to a stride parameter that specifies the amount of shift between the convolutions. Each convolutional layer of the CNN can have a different kernel size and a number of filters (distinct kernels). Other less typical variants of convolutional layers exist, e.g. dilated convolution or kernels of irregular learnable shapes [39].

The output is similar to the input where the dimensions of the output tensor h_{out} and w_{out} depend on the size of the kernel k_i and padding p_i and the number of output channels c_{out} depends on the number of filters f_i of the given layer. The values k_i, p_i, f_i of the i-th convolutional layer are non-learnable hyperparameters of the neural network that are chosen before the training process. The use of one convolution operation for $f_i = 1$ and $c_{in} = 1$ is illustrated in Figure 2.1 and expressed mathematically as

$$r_{\bar{i},\bar{j}} = \sum_{i=-m}^{m} \sum_{j=-m}^{m} \boldsymbol{v}_{\bar{i}+i,\bar{j}+j} \cdot \boldsymbol{w}_{i+m+1,j+m+1}, \tag{2.1}$$

where $r_{\bar{i},\bar{j}}$ is the result of the convolution operation centered at input coordinates $\bar{i}, \bar{j}$, kernel size $k_i = 2m + 1$, $\boldsymbol{v}$ is the matrix of input values, and $\boldsymbol{w}$ is the matrix representing the kernel weights [38].

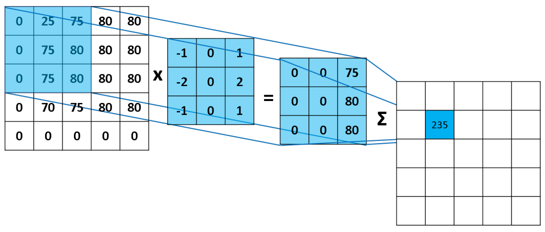


Figure 2.1: Example of the use of the convolutional layer[1].

2.1.2 Activation Function

Activation functions give the neural network the ability to fit even non-linear data. We used the Leaky Rectified Linear Unit (LReLU) as an activation function following the convolutional layers. LReLU is a version of ReLU [40] that does not cause “dead” connections. This refers to a situation when ReLU outputs zero for a given connection which basically blocks the gradient from propagating and LReLU solves this by a small slope even for the negative values instead of a zero output. It can be expressed mathematically as

$$LReLU(x) = \begin{cases} \alpha x, & x \leq 0 \\ x, & x > 0, \end{cases} \tag{2.2}$$

where x is the input and α is the coefficient that determines the slope for the negative inputs [41].

[1] https://mlnotebook.github.io/post/CNN1/

2.1.3 Pooling Layers

Two types of pooling layers are used in our approach. Firstly, max pooling layers are used in between the convolutional layers. Max pooling layer is expressed as

$$r_{\bar{i},\bar{j}} = \max_{\substack{i=-m..m \\ j=-m..m}} \boldsymbol{v}_{\bar{i}+i,\bar{j}+j}, \tag{2.3}$$

where $r_{\bar{i},\bar{j}}$ is the result of the convolution operation centered at input coordinates $\bar{i}, \bar{j}$, kernel size $k_i = 2m + 1$, and $\boldsymbol{v}$ is the matrix of input values [42].

The purpose of the max pooling layer is to downsample the feature maps while maintaining important information (by applying the max pooling operation with a given stride similarly to convolution 2.1.1). The output of each max pooling operation is the maximum value from the corresponding region of the input feature map. The downsampling allows for an increase in the number of filters of the later convolutional layers without an excessive increase in the number of computations. Another type of the pooling layer used in this thesis is the global average pooling (Figure 2.2). The difference from the max pooling is that the global average pooling considers the entire feature map without a sliding kernel and the output is not the maximum value but the average of all values in the feature map. Therefore, if the input has the dimensions of $h_{in} \times w_{in} \times c_{in}$ then the output is a vector of size c_{in}. Global average pooling is used when it is desirable to account for all of the values and not only the maximums [43, 42].

2.1.4 Dense Layer

Lastly, the dense layers implement linear transformations of the feature maps. They can be represented as a matrix of weights $\boldsymbol{W}$ with the dimensions of h_m, w_m that transforms an input vector $\boldsymbol{v}_{in}$ of size w_m to an output vector $\boldsymbol{v}_{out}$ of size h_m. A bias vector $\boldsymbol{b}$ is added to the output vector after the transformation as described in

$$\boldsymbol{v}_{out} = \boldsymbol{W} \cdot \boldsymbol{v}_{in} + \boldsymbol{b}, \tag{2.4}$$

where the symbol $\cdot$ represents the dot product [44].

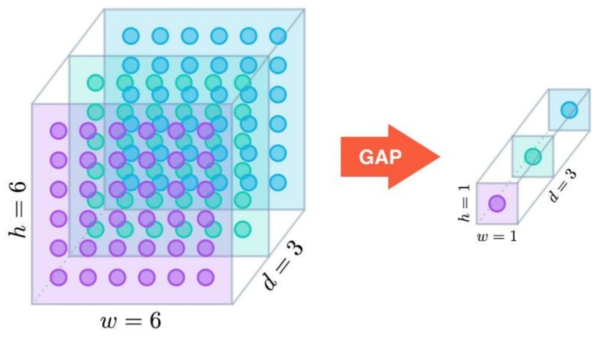


Figure 2.2: Example of a Global Average Pooling layer[2]. The labels h, w, d correspond to $h_{in/out}, w_{in/out}, c_{in/out}$ respectively.

[2] https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/

2.2 Proposed Network

2.2.1 Network Output

Most approaches for counting mentioned in Section 1.1 are designed for crowds of people and their output is a density estimate in the x, y-axes. This density estimate is in the form of a density map which is a 2D matrix of values that show the estimated crowd density at each pixel of the image. This thesis aims to apply similar principles in the z-axis for distance-density estimation. Instead of a 2D matrix, the network outputs a 1D vector $\boldsymbol{l}$ that represents a discrete distribution of the UAVs over the distance

$$l_i = |\, \{\boldsymbol{p} \mid \boldsymbol{p} \in \mathcal{M},\ ||\boldsymbol{p}|| \in \langle i \cdot {}_{\triangle}d, (i+1) \cdot {}_{\triangle}d)\} \,|, \tag{2.5}$$

where l_i is the i-th element of the vector $\boldsymbol{l}$ and $i \in \{0, 1, 2, \ldots, n_{bin} - 1\}$, ${}_{\triangle}d$ is a distance discretization step, $\mathcal{M}$ is a set of positions of all UAVs in the image, and $\boldsymbol{p}$ is a position of an observed UAV expressed in the optical frame of the camera. Each element represents one distance bin with a width ${}_{\triangle}d = 1m$ (the first bin represents the distance from 0 meters to 1 meter etc.). The number of bins was selected as $n_{bin} = 50$. The value of the element represents the estimated count of UAVs in the corresponding distance bin. Therefore, the total count of UAVs in the image can be acquired by summing all the elements of the output vector together, or for example, the number of UAVs closer than 10 meters is extracted by summing the first ten elements of the vector. Furthermore, the number of bins n_{bins} together with the distance discretization step ${}_{\triangle}d$ specify the maximum distance $d_{max} = n_{bins} \cdot {}_{\triangle}d$ (50m in our case) that is covered by the output vector $\boldsymbol{l}$. Any detected UAVs that are further than d_{max} are also counted into the last element of the vector $\boldsymbol{l}$ as

$$l_{n_{bin}-1} = |\, \{\boldsymbol{p} \mid \boldsymbol{p} \in \mathcal{M},\ ||\boldsymbol{p}|| > d_{max} - {}_{\triangle}d\} \,|. \tag{2.6}$$

The proposed network can be modified to approximate the distribution of the drones not only in the z-axis but also in the x, y-axes. The output is divided into a grid of multiple distance-density vectors of the corresponding parts of the input image. The tail of the network (the last two layers) and the training data are modified for this purpose as described in Section 2.2.2. The output of the network then represents a tensor with the dimensions of $h_{out} \times w_{out} \times n_{bin}$, where h_{out} and w_{out} are the height and width of the sub-image grid in cells (e.g. dividing the image into 9 sub-images yields $3 \times 3 \times n_{bin}$ tensor).

2.2.2 Network Architecture

The head (the first 15 layers) of the network introduced in this thesis is a feature extractor inspired by VGG [45]. It consists of a series of two convolutional layers with activation functions followed by a max pooling layer and this pattern is repeated multiple times, adding more filters to the convolutional layers with every repetition. This design is chosen because it proved to perform well on similar problem[3] (counting objects into different groups based on scale). Also, as stated in [27], the single-column architecture has the advantage of sharing low-level features resulting in fewer parameters compared to the multi-column approach as in [26]

[3]https://www.kaggle.com/code/sardinipasta/kera-sea-lion-count

which is critical for online deployment on an onboard computer with limited computational capabilities.

The feature extractor is followed by a single convolutional layer with a kernel of size 1x1 and n_{bin} filters. This 1x1 convolutional layer is also utilized by some of the above-mentioned crowd counting approaches [26, 27]. A kernel of size 1x1 removes the ability to learn relative features between nearby cells. However, 1x1 convolution is used here as a weighting procedure that chooses which feature maps are important for every distance bin in the output vector. Moreover, utilizing the 1x1 convolution instead of the typically used dense layers [4] proved to have better performance (Figure 5.17) with a significantly reduced number of parameters which is crucial for online deployment.

The 1x1 convolutional layer is connected to a global average pooling layer that produces a single vector of dimension n_{bin} which is then connected via a single dense layer to the network's output. These last layers facilitate regression of the final density values from the features acquired by the feature extractor. Diagram of the model can be seen in Figure 2.3

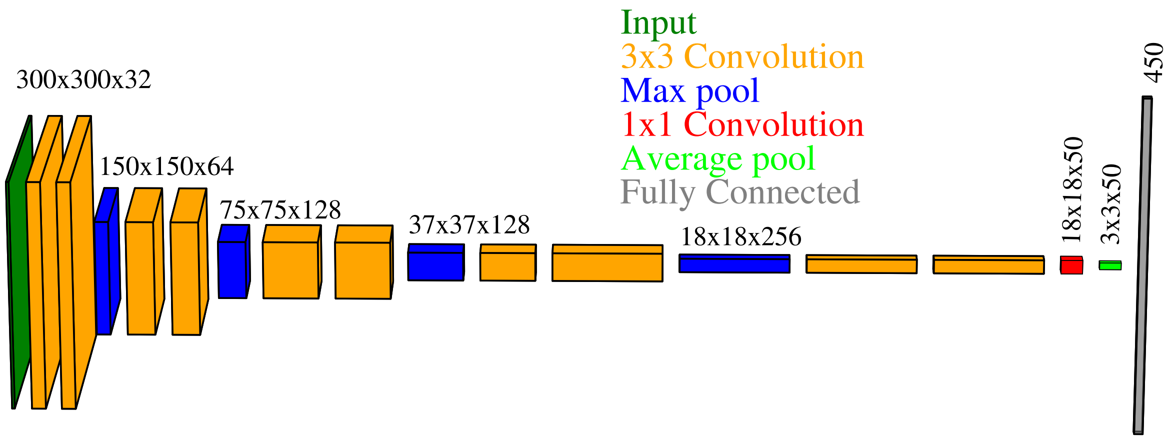


Figure 2.3: A diagram of the layers within the proposed network architecture.

To divide the image into the $h_{out} \times w_{out}$ grid and output multiple distance-density vectors, we use average pooling with a correctly chosen stride parameter instead of using the global average pooling. The average pooling is applied so that it produces a single value for each grid cell instead of pooling the entire feature map into a single value. E.g. for an 18×18 feature map, we use a pooling kernel of the size 6×6 and stride of 6 in both x and y-axes to obtain a grid division into 3×3 cells. Finally, the output vector is a vector of size $n \times n_{bin}$, where $n = h_{out} \cdot w_{out}$ is the number of grid cells.

Despite the fact that the grid division increases the size of the predicted network output by 900% (in the above-mentioned case) the number of network parameters increases significantly less (approx. 10%). The modification of the average pooling layer itself adds no new parameters. The increase in complexity is caused by the final dense layer that is used right before the output.

[4]https://www.kaggle.com/code/sardinipasta/kera-sea-lion-count

Chapter 3

Training

The neural networks considered in this work are trained using supervised learning. This means that the data are provided for the training along with the correct output labels. This allows the network to learn the mapping between the input and output [37].

The training of the neural network typically utilizes a technique called Gradient Descent (GD). A variety of optimization techniques based on GD exist. We chose the Adaptive Moment Estimation (ADAM) [46] optimizer as described in 3.1. An essential part of the training process is the loss function introduced in 3.2. The ADAM optimizer uses the gradient of the loss function to converge toward the minima leading to improved performance of the neural network. The gradient is calculated using a process called backpropagation. Backpropagation uses the chain rule to compute the gradient with respect to the trainable weights within the network so that those can be updated accordingly [37].

These methods along with other functionalities are provided within the Tesorflow framework. Tensorflow [47, 48] is an open-source library commonly used for working with neural networks. It provides tools for data processing and tools for training (Section 3) and inference of the neural network model. It implements the backpropagation as well as the weight updates. Tensorflow offers a wide variety of optimizers (Section 3.1), metrics, and loss functions (Section 3.2) but also provides the option to implement custom loss functions if desired. The layers (Section 2) and activation functions (2.1.2) needed for the construction of a neural network are included as well.

3.1 Optimizer

The goal of the optimizer is to effectively find viable values for the weights $\boldsymbol{\Theta}$ of the neural network (e.g. the weights $\boldsymbol{w}$ from eq. 2.3 or the matrix $\boldsymbol{W}$ and biases $\boldsymbol{b}$ from eq. 2.4). The ADAM optimizer introduced in [46] adapts the learning rate for each parameter of $\boldsymbol{\Theta}$ individually and it is a popular choice for a variety of machine learning problems.

Firstly, the gradient $\boldsymbol{g}_t$ of the loss function $\mathcal{L}$ (Section 3.2) at the t-th learning step with respect to the weights of the neural network $\boldsymbol{\Theta}$ is obtained using the backpropagation algorithm:

$$\boldsymbol{g}_t = \nabla_{\boldsymbol{\Theta}} \mathcal{L}. \tag{3.1}$$

Then, the first moment $\boldsymbol{m}_t$ and second moment $\boldsymbol{v}_t$ for the current step t are computed according to the equations

$$\boldsymbol{m}_t = \beta_1 \cdot \boldsymbol{m}_{t-1} + (1 - \beta_1) \cdot \boldsymbol{g}_t, \tag{3.2}$$

$$\boldsymbol{v}_t = \beta_2 \cdot \boldsymbol{v}_{t-1} + (1 - \beta_2) \cdot \boldsymbol{g}_t^2, \tag{3.3}$$

where β_1 and β_2 are tunable parameters typically set to 0.9 and 0.999 respectively, and $\boldsymbol{g}_t^2$ is an elementwise square.

The next step consists of calculating the bias-corrected first and second moments $\hat{\boldsymbol{m}}_t$ and $\hat{\boldsymbol{v}}_t$

$$\hat{\boldsymbol{m}}_t = \frac{\boldsymbol{m}_t}{1 - \beta_1^t}, \qquad \hat{\boldsymbol{v}}_t = \frac{\boldsymbol{v}_t}{1 - \beta_2^t}. \tag{3.4}$$

$$\tag{3.5}$$

This bias-correction step is needed because the moments are initiated with zero values and therefore biased towards zero.

The last step combines the two moments and uses them to update the weights of the neural network in the following way:

$$\boldsymbol{\Theta}_t = \boldsymbol{\Theta}_{t-1} - \alpha \cdot \frac{\hat{\boldsymbol{m}}_t}{\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon}, \tag{3.6}$$

where $\boldsymbol{\Theta}_t$ are the weights at the step t, α is the learning rate, and ϵ is a small constant to prevent division by zero.

3.2 Loss Function

The main purpose of the loss function [37] is to guide the optimizer (Section 3.1) toward the minima during the training process. However, it is also useful for evaluating the performance and convergence of the network during training by comparing the predicted results to the label. The loss function in this thesis is a weighted Euclidean distance between the predicted and ground truth vectors

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} ||(\boldsymbol{l}_i - \hat{\boldsymbol{l}}_i) \circ \boldsymbol{w}||, \tag{3.7}$$

where N is the number of training images, $\boldsymbol{l}_i$ and $\hat{\boldsymbol{l}}_i$ is the predicted vector (network output) and the label density vector respectively for the i-th image, $\boldsymbol{w}$ is a vector of non-learnable weights, and the symbol $\circ$ represents the element-wise multiplication between vectors.

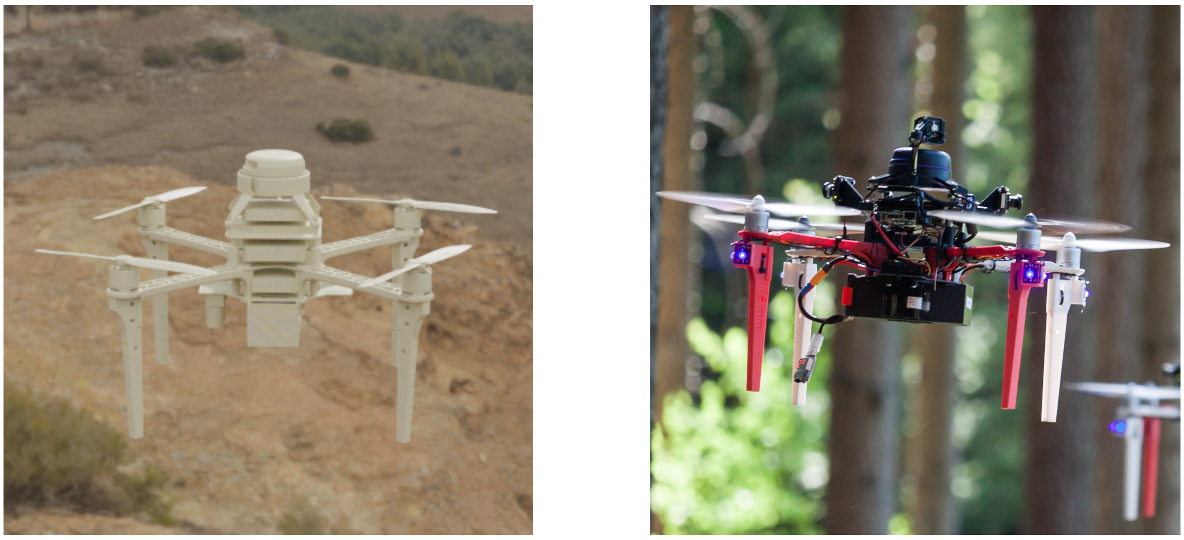
The additional vector of weights $\boldsymbol{w}$ rescales the errors for each individual distance bin. Increasing the weight for a particular bin forces the network to prioritize that given level during the training process. This modification of the loss function allows to target specific distance bins that are of higher interest e.g. near the UAV. Having reliable detection in the lower distance bins (near the UAV) is crucial for collision avoidance. However, the majority of the UAVs are typically further away from the UAV (simply because there is less space in the field of view near the camera). Therefore, the network tends to focus on the majority further away and ignore the sparse close UAVs. The weight vector $\boldsymbol{w}$ counters this bias and drastically improves the detection success rate of the nearby UAVs (Section 5).

For the case when the output is divided into a grid ($w_{out},\ h_{out} > 1$), the vectors $\boldsymbol{l}_i$, $\hat{\boldsymbol{l}}_i$, and $\boldsymbol{w}$ for all grid cells are stacked into meta-vectors $\boldsymbol{L}_i$, $\hat{\boldsymbol{L}}_i$, $\boldsymbol{W}$ and then the loss function is calculated analogically to eq. (3.7) using these meta-vectors.

3.3 Data

There are two main issues in creating a real-world dataset for the task presented in this thesis. Firstly, this task requires the distances from the camera to be accurately measured for every UAV in the image. That is complicated considering the number of UAVs in the image but still feasible. The main issue is the number of drones needed to be simultaneously in the image. This thesis works even with tens of UAVs in a single image. Unfortunately, as of writing this thesis, we do not have so many UAVs ready for deployment.

Therefore, we opted to generate the dataset using the approach introduced in [6]. The data were generated using 272 environments for background (including nature as well as urban environments). The images feature different lighting conditions and various color patterns are applied to the drones to accentuate shape-based object representation as discussed in [6]. A single UAV platform model was used in this dataset. The platform used is DJI f450, which is commonly used for swarm deployment within the Multi-robot Systems group[1]. The platform was modeled in Blender and used to generate the data. Comparison with a real-world photo can be seen in Figure 3.1.



(a) (b)

Figure 3.1: Comparison of synthetic (a) and real (b) images of the DJI f450 platform [49] used in this thesis.

[1]http://mrs.felk.cvut.cz/

The data are originally generated in Full HD resolution (1920x1080 pixels) with the projection model parameters corresponding to the chosen camera which is Intel RealSense D435[2]. However, the input resolution of the network is 300x300 pixels, so 300x300 image patches are cropped from the original images without rescaling (because the distance to be estimated depends on the size of the UAVs). The image pathces were selected so that the total counts of the UAVs in a single image are represented as equally as possible (see Figure 3.2). The total counts of UAVs within a single image range between 0 and 34 UAVs per image. We used 10000 images for training, 1000 images for validation, and 5000 images for testing.

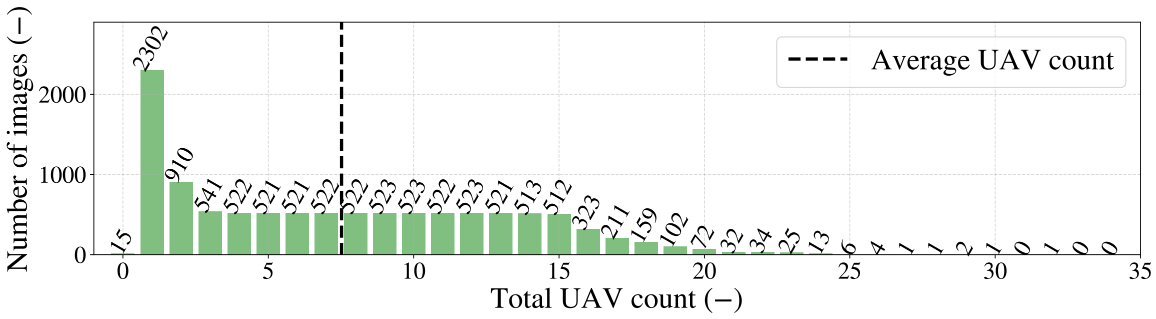


Figure 3.2: Distribution of images in the training and validation dataset based on the total number of UAVs in a given image. The exact count values are stated above the bars.

Despite the intention of keeping the total counts of UAVs per image as equally represented as possible, it may be observed that some of the counts are represented disproportionally in Figure 3.2. There is a gradual decrease in the number of images that contain more than 15 UAVs. This is caused by a property of the original data that it is more difficult to find 300x300 pixel patches with larger amounts of UAVs than 15. There is no decrease prior to the count of 15 UAVs because the numbers of images with UAV counts lower than 15 are artificially kept close to the number of images containing 15 UAVs. Choosing the count of 15 UAVs is a tradeoff between having a uniform distribution and enough data.

Furthermore, there is a significantly larger amount of images containing one or two UAVs. This is the result of another constraint that was imposed when generating the data. We specifically wanted to include in the dataset a higher number of UAVs that are in close proximity to the focal UAV (Figure 3.3) as correctly estimating those UAVs is critical for collision avoidance. This causes larger amounts of images containing one or two UAVs because of the nature of the camera sensor. The closer the UAV is to the camera the more space in the image is occupied by the UAV and thus fewer close UAVs can fit in the image. Therefore, it is more probable to have larger groups at larger distances resulting in more UAVs at larger distances in general as presented in Figure 3.3 and smaller groups at closer distances.

Basically, the ideal dataset would be balanced in the total number of UAVs (e.g. the same number of image patches containing a total of 2 UAVs as those containing a total of 17 UAVs) and also in the number of UAVs per distance bin (same number of UAVs in each distance bin over the whole dataset). Satisfying both of those constraints at the same time is a tradeoff.

During the data preparation process, it is beneficial to crop the 300x300 image patches

[2]https://www.intelrealsense.com/depth-camera-d435/

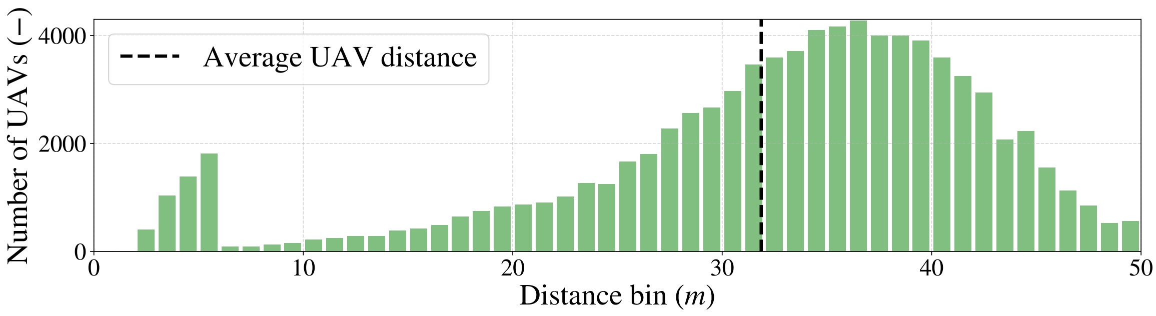


Figure 3.3: Distribution of UAVs in the training and validation dataset based on their corresponding distance bin.

in a way so that the result contains only whole UAVs. This means that no images contain UAVs partially inside the image and partially outside of the image. In the end, this leads to overall better performance even on images with only partially present UAVs. This is probably caused by the fact that the UAV is not evenly occupying the bounding box by which it is marked in the original data. For e.g. most of the UAV is in the upper part of the bounding box since it includes the body and the lower part includes only the legs, therefore, it is difficult to accurately state how significant portion of the UAV is within the image (potentially making it harder for the neural network to converge). Occlusions were included in the dataset.

During the evaluation of the trained models, we found that the large difference in the amount of UAVs in the 5th bin compared to the 6th bin is causing the error metric to spike around this distance and does not reflect the performance correctly. Therefore, we modified the distance-wise distribution of the dataset to smooth out sudden count changes as shown in Figure 3.4. However, this added more low-count groups in the dataset making the count-wise distribution even more imbalanced as described in Figure 3.5. On the other hand, smoother distance-wise distribution proved more suitable for the evaluation than a smooth count distribution.

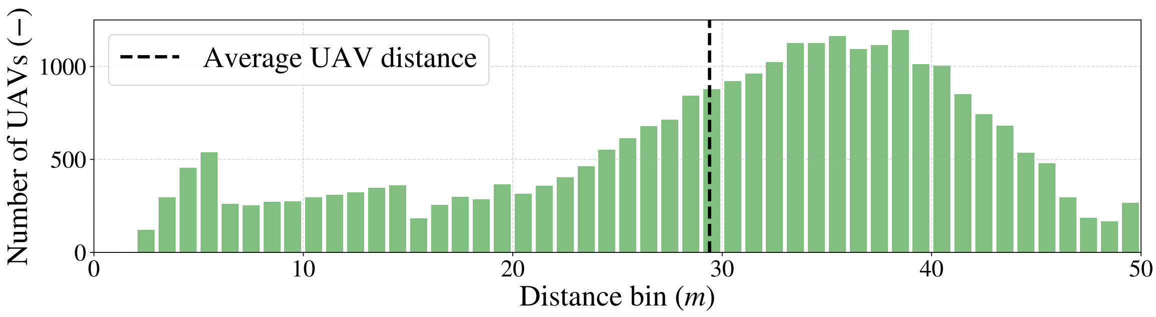


Figure 3.4: Distribution of UAVs in the testing dataset images based on their corresponding distance bin.

The label format is inspired by density maps which are often the output of the state-of-the-art crowd counting methods [26, 27, 24, 23]. Labels for the density maps are obtained by marking the heads of the people with the value 1 in the image and leaving the rest of the pixels assigned the value 0. Convolution is then applied to this matrix. The kernel of this

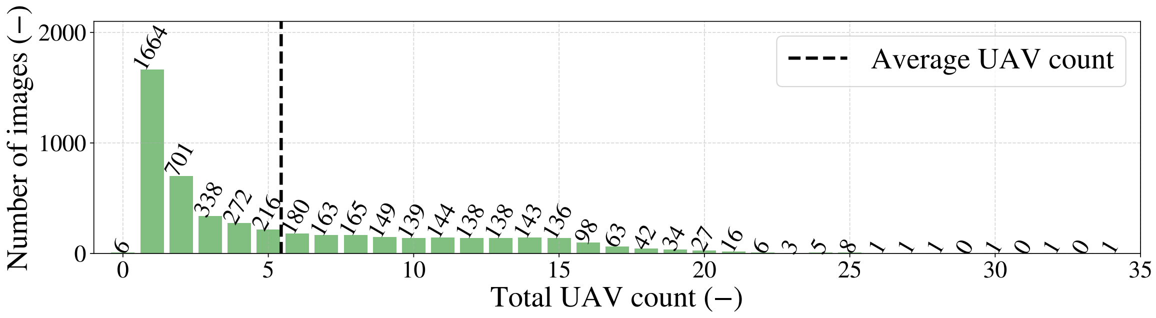


Figure 3.5: Distribution of images in the testing dataset images based on the total number of UAVs in a given image. The exact count values are stated above the bars.

convolution is typically a Gaussian kernel. Analogically, we apply a Gaussian kernel on the distance bins as

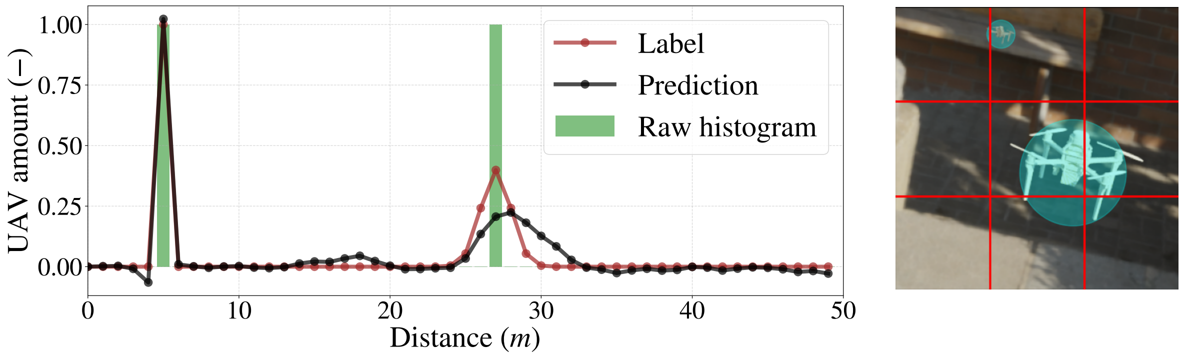
$$l_i^o = l_i * g[i], \tag{3.8}$$

$$g[i] = \begin{cases} \delta[i], & \text{for the closest } k \text{ bins,} \\ G[i], & \text{otherwise,} \end{cases} \tag{3.9}$$

where l_i^o is the i-th element of the output distance-depth vector $\boldsymbol{l}^o$, l_i is the i-th element of the original vector $\boldsymbol{l}$, $G[i]$ denotes a Gaussian kernel with the standard deviation equal to 1, $\delta[i]$ is the identity kernel, $*$ denotes the convolution, and $k = 7$ in our case. Values in the distance bins up to the k-th bin are identical to the respective original values in $\boldsymbol{l}$ (each UAV is assigned to exactly one bin). The values for the rest of the bins from the k-th to the n_{bin}-th bin are affected by the Gaussian kernel resulting in the "UAV mass" being distributed partially even into the adjacent bins (Figure 3.6). With Gaussian labels, the small inaccuracies in distance estimation (only by a couple of bins) are less penalized by the loss function than the larger ones. This is because at least a part of the "UAV mass" overlaps with the label for such a near miss compared to zero overlaps for a near miss with the raw labels. The Gaussian filter also does not change the overall sum of all values in the matrix which allows to get the total count by summing all values or partial sum for a given region by summing the corresponding values. We do not use the smoothed labels in all bins because the network is capable of estimating the distance precisely to the exact bin if the UAV is close enough and the Gaussian label forces the output to be smoothed resulting in unnecessarily reduced accuracy.

3.3.1 Real data

Despite the above-mentioned challenges of acquiring a real-world dataset, we created a small dataset of 40 images from the footage taken during various experiments of the Multi-robot systems group at the Czech technical university in Prague. However, this data lacked the correct labels. The bounding boxes were labeled manually. To label the distances of the individual UAVs, we used the same nearest-neighbor method for distance estimation as with the object detector (Section 5.2.3). The data contain between 1 to 8 UAVs per image in various environments (desert, forest, field) as shown in Figure 3.7. Some of the images include slightly different UAV types than the ones used for training.



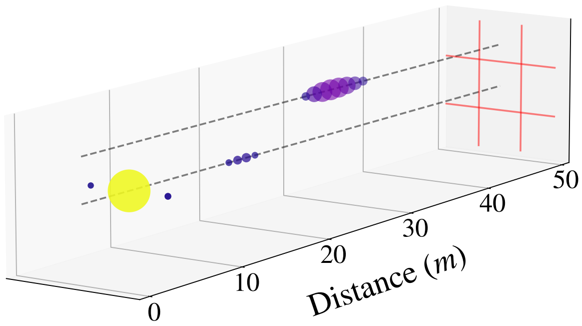


Figure 3.6: Top left: An example image from the synthetic dataset with highlighted grid division and UAVs. Top right: The raw histogram of UAV count over distance, the label, and the output of the CNN. Bottom: Output of the CNN for the version with grid division.

Figure 3.7: Examples of the real-world photos from various environments.

3.3.2 Camera Compensation

The size of the object of interest within the image is an important factor when estimating the distance of the objects. This size can differ greatly for a given object at the same distance based on the camera that is being used to capture the image. The training data are generated using a camera with given parameters and the network learns to estimate distances of the objects in images taken by such a camera (as those parameters affect the size of the objects in the image). This means that if a camera with different parameters is used to take images for the network then the output distance estimates will be incorrect.

One solution is to generate or gather data captured by the desired camera and train the network on those new data. This solution is, however, very time and resource-consuming not only because of the need to acquire large amounts of data and label them properly (position and distance of each drone) but also because of the training process itself (see Section 3).

Therefore, we designed a method to transform the output of the network based on the relation between the parameters of the training and the desired camera. Firstly, we present compensation for the different focal lengths of the two cameras. In order to transform the output properly, we assume that both cameras can be modelled using the pinhole camera projection model. The size of the object in the image according to the pinhole camera model is described as

$$h = r \cdot \mathcal{H} \cdot \frac{f}{d}, \tag{3.10}$$

where h is the size of the object in the image plane, $\mathcal{H}$ is the real size of the object, f is the focal length of the camera, and d is the depth of the object from the camera.

The first parameter influencing the size of the object is the focal length. The second parameter is the resolution of the camera because we are interested in the size of the object in pixels. The height and width dimensions of the input image to the network are constant. Therefore, if the resolution is scaled e.g. by the factor of two but the same amount of pixels is chosen then the objects appear two times larger because all of its dimensions are now represented by two times more pixels. We consider both of the resolutions to have the same height-to-width ratio. If that is not true for a given image, the same ratio can be achieved by using proper padding.

Considering the same height-to-width ratio (equal scaling of both dimensions) we can represent the resolution e.g. by the number of pixels per unit of depth in which the size of the object's projection to the image plane h is measured. The formula 3.10 for the size of the object in pixels is then

$$h_{px} = r \cdot h = r \cdot \mathcal{H} \cdot \frac{f}{d}, \tag{3.11}$$

where h_{px} is the size of the object in the image measured in pixels and r is the number of pixels per unit of depth in which the size h is measured.

We assume that the drones are all of the same real size $\mathcal{H}$. The size of the drone h_{px} is determined by the distance bin (Section 2.2.1) in which it was placed by the network. Considering models of two cameras with different focal lengths f_1, f_2 and different resolutions r_1, r_2 as shown in Figure 3.8 it holds that

$$h_{px1} = r_1 \cdot \mathcal{H} \cdot \frac{f_1}{d_1}, \qquad h_{px2} = r_2 \cdot \mathcal{H} \cdot \frac{f_2}{d_2}, \tag{3.12}$$

and assuming they produce same-sized objects with $h_{px1} = h_{px2}$ (meaning that both drones are estimated by the network to belong to the same distance bin) we obtain

$$r_1 \cdot \frac{f_1}{d_1} = r_2 \cdot \frac{f_2}{d_2}. \tag{3.13}$$

Now let us rename indices of the variables as $1 = t$ and $2 = n$, where t and n are corresponding to the camera model used for training and the new desired camera model respectively. The depth d_t from the perspective of the training camera is known (based on the distance bin estimated by the network). If the focal lengths f_t, f_n and resolutions r_t, r_n are also known, the correct distance estimate d_n for the new desired camera is obtained as

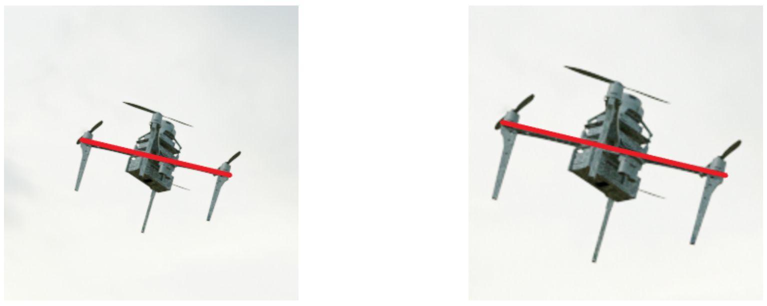


Figure 3.8: Example of the same UAV at the same distance viewed by two cameras with different parameters. The red lines depict the h_{px1} and h_{px2} respectively (which we know to be identical in reality).

$$d_n = \frac{f_n}{f_t} \cdot \frac{r_n}{r_t} \cdot d_t. \tag{3.14}$$

The bins of the output vector are each 1 meter wide (Section 2.2.1). Therefore, the borders between the bins are at 1 meter, 2 meters, etc. The equation (3.14) is applied for all bins which results in the respective transformed bins. The amount of UAVs in each bin stays the same only bins change according to the transformation. An assumption is made that the depth (used in the equations above) and distance (predicted by the network) are the same for the purposes of the camera compensation. Despite this assumption the compensation performs well as shown during the experiments (Section 5.2.8).

3.4 Training Process

The training itself consists of epochs. An epoch is concluded each time the entire training dataset is passed through the network during training. The number of epochs is a tunable hyperparameter. Both too-low and too-high numbers of epochs lead to worse results. An insufficient number of epochs causes underfitting of the model which means that the model is still too simple to handle the problem correctly. On the other hand, having too many epochs results in overfitting which occurs when the model has very good results on the training data but is unable to properly generalize to previously unseen data. We chose a version of an approach called early stopping to avoid both underfitting and overfitting. This approach also requires to have validation data (data that are not used to update network weights via backpropagation). The loss of the model on the validation data is calculated after every epoch and only the model weights with the lowest validation loss so far are saved (Figure 3.9). Therefore, the network can be trained for as long as possible without losing the generalization ability.

Batch size defines how many training samples are passed through the network between the weight updates. Having batch size equal to the number of training samples results in one weight update per epoch and can more precisely follow along the gradient, however, it is often very memory-demanding. We use mini-batches (batch size < number of training samples) during training which tends to converge faster as there are more weight updates in one epoch and it consumes less memory. In this thesis, the batch size of 32 was experimentally found to deliver the best results.

Learning rate as described in Section 3.1. The training process can be very slow if a too-low value of the learning rate is chosen. On the other hand, large learning rates can make the model unable to converge to the desired minimum. We started the training with the typically used learning rate of $1 \cdot 10^{-3}$. However, we used an additional method implemented in Tensorflow [47, 48] to change the learning rate during the training. This method lowers the value of the learning rate by a given factor (0.8 in this case) if the validation loss did not improve more than by a given minimum value in the last couple of epochs. This is beneficial as it allows for faster training at the beginning but also gradually lowers the learning rate for more precise convergence in the later epochs (Figure 3.9).

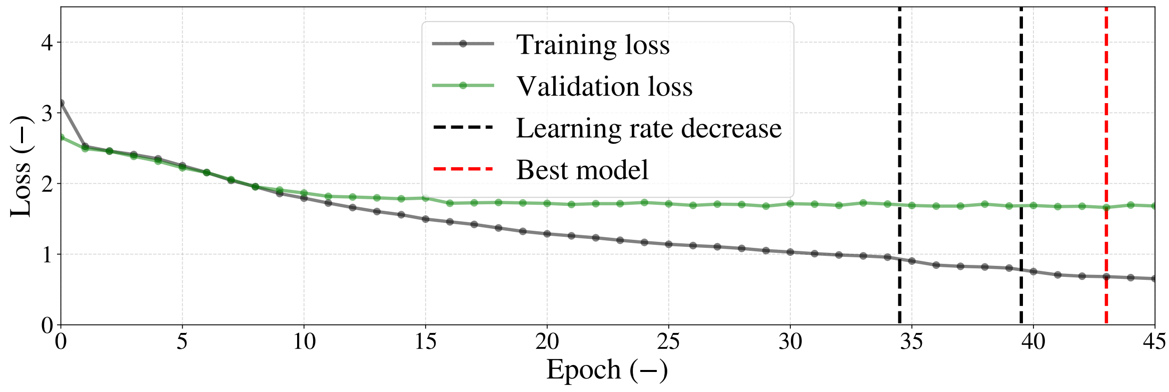


Figure 3.9: Example of training and validation loss during training together with marked changes in learning rate and best model weights.

Chapter 4

Swarming Controller

We designed a swarming controller that is an adaptation of Boids [34] to demonstrate that the method presented in this thesis can be used as the source of relative positions of neighboring UAVs for swarming purposes. The controller was combined with the proposed method in a simulation.

In the simulation, the swarm members are constrained to a 2D plane for clarity and visualization purposes. We assume that each of the UAVs within the swarm is equipped with 3 cameras and that each of the cameras covers an angle of 120°. The cameras are placed on the UAV in a way that their fields of vision do not overlap and therefore provide the UAV with 360° visual sensing as depicted in Figure 4.1. The input of each of the cameras is divided into 3 subregions (left, center, and right) as shown in Figure 4.1. This subdivision emulates the approach of dividing the image into a grid that was introduced in Section 2.2.1. As a result, each of the UAVs has information about its neighbors represented by 9 distance-density vectors (3 cameras $\times$ 3 subregions) that cover 360° around the UAV.

It is unfeasible to accurately simulate a large number of UAVs with each UAV processing the output of three cameras while also simulating the individual cameras. To solve this problem, we represent the UAVs as mass points and we do not run the neural network directly on any images but we emulate its output. For this purpose, an error is randomly generated based on the experimental results of the CNN model on testing data and this error is combined with the ground truth positions of the neighboring UAVs provided by the simulation. The error is approximated by a multivariate normal distribution (Figure 4.2) for each bin individually (using the per-bin error introduced in Section 5.1). The mean vector and covariance matrix of the sampled error values (over the testing dataset) are computed as

$$\boldsymbol{\mu} = (E[X_1], E[X_2], \dots, E[X_{n_{bin}}]), \quad \boldsymbol{\Sigma}_{i,j} = E[(X_i - \mu_i)(X_j - \mu_j)], \tag{4.1}$$

where the $\boldsymbol{\mu}$ is the vector of mean values $E[X_i]$ for each bin, $\boldsymbol{\Sigma}$ is the covariance matrix with elements $\boldsymbol{\Sigma}_{i,j}$, and n_{bin} is the number of output distance bins. The error is estimated and then randomly generated individually for each of the 50 distance-density output bins. Figure 4.2 shows the approximation of the error by the multivariate normal distribution in one of the distance bins in comparison to the real measured error for the same bin. The final

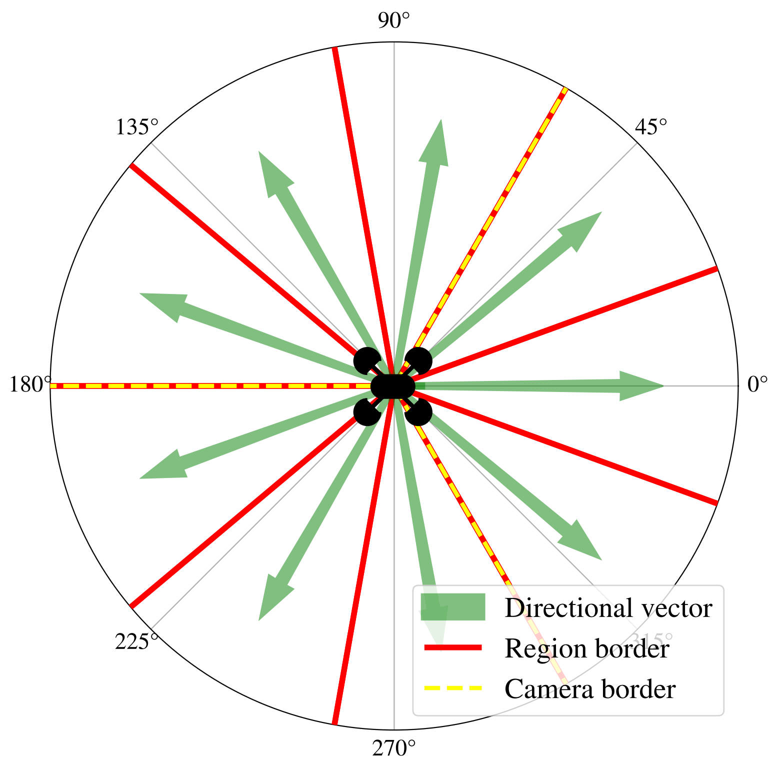


Figure 4.1: Illustration of the assumed swarming setup. The focal UAV is in the center. The dotted yellow lines mark the borders between the regions captured by individual cameras. The red lines are the borders between the individual subregions within each image. The green arrows represent the directional unit vectors.

emulated output of the neural network is then obtained as follows

$$\boldsymbol{l}^e = \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_{50} \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_{50} \end{bmatrix}, \tag{4.2}$$

where $\boldsymbol{l^e}$ is the emulated output for a given subregion, l_i is the ground truth UAV amount for the i-th bin of the output, and e_i is the error for the i-th bin of the output generated randomly from the multivariate normal distribution $\mathcal{N}_{n_{bin}}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$.

As mentioned above, the 9 distance-density vectors cover the entire 360° around the UAV. That means that there is a distance-density vector for each 40°. We assign one directional unit vector to each distance-density vector (subregion) as shown in Figure 4.1. This unit vector is parallel to the optical axis of the corresponding camera. To compute the desired direction and speed of movement of the focal UAV, the directional unit vectors are scaled based on the corresponding $\boldsymbol{l}^e$ and summed. The procedure iterates over the distance bins in all distance-density vectors simultaneously. This means that firstly the i-th distance bins in all 9 distance-density vectors are accounted for before proceeding to all of the $i+1$-th distance bins. Each of the bins is accounted for in the following way

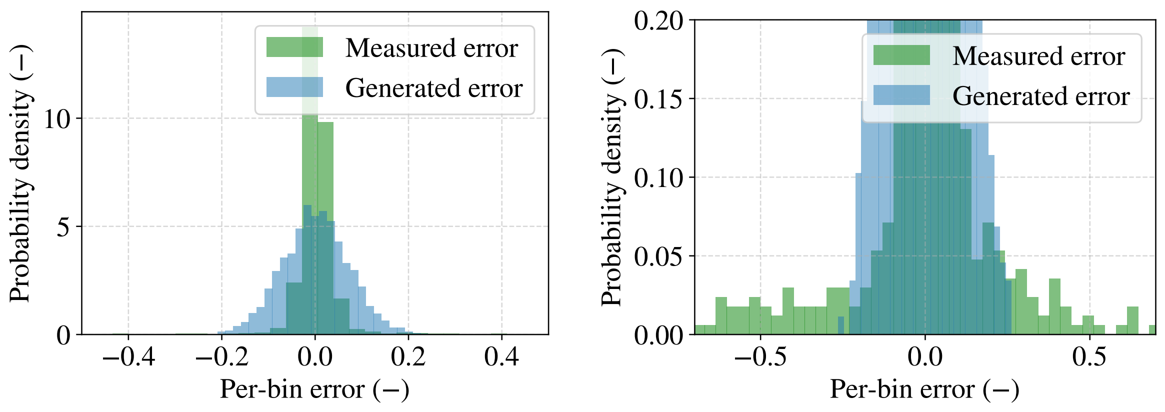


Figure 4.2: Left: Histogram of the real measured error (green) and the generated simulation error (blue) for a single distance bin. Right: Zoomed histogram on the right to show the outlying errors.

$$\operatorname{sign}(x) = \begin{cases} -1, & \text{if } x < d_{lb} \\ 0, & \text{if } d_{lb} < x < d_{ub} \\ 1, & \text{if } d_{ub} < x \end{cases}, \tag{4.3}$$

$$\boldsymbol{a}_{ji} = \operatorname{sign}(i) \cdot \frac{1}{i} \cdot \boldsymbol{l}^e_{ji} \cdot \boldsymbol{u}_j, \tag{4.4}$$

where $i \in \langle 1, ..., 50 \rangle$ corresponds to the i-th distance bin, $\boldsymbol{l}^e_{ji}$ is the predicted amount of UAVs in the i-th bin of the distance-density vector corresponding to the j-th subregion, $\boldsymbol{u}_j$ is the unit directional vector corresponding to the j-th subregion, and $\boldsymbol{a}_{ji}$ is the partial control vector produced by the UAVs predicted in the i-th bin of the j-th subregion. The d_{lb} and d_{ub} are lower and upper bound respectively. This equation can be interpreted as moving away from UAVs closer than d_{lb} meters to avoid collisions and moving towards UAVs further away than d_{ub} meters to stay within the swarm and the UAV is more reactive to closer neighbors. The partial control vectors are combined to produce the final control vector $\boldsymbol{v}_f$

$$\operatorname{pass}(d) = \begin{cases} 1, & \text{if } \sum_{i=1}^{d-1} \sum_{j=1}^{9} \boldsymbol{l}^e_{ji} < N_h \\ 0, & \text{otherwise} \end{cases}, \tag{4.5}$$

$$\boldsymbol{v}_f = \sum_{i=1}^{50} \sum_{j=1}^{9} \operatorname{pass}(i) \boldsymbol{a}_{ji} \tag{4.6}$$

The direction and size of the final control vector $\boldsymbol{v}$ represent the desired velocity of the UAV. There is one more control rule represented by the function *pass*. An amount of UAVs N_h (typically 6) is given and if the sum of the UAVs predicted up until the distance i is equal to or greater than the given amount N_h then the rest of the bins with a distance greater or equal to $i+1$ is ignored and not accounted for when calculating the final control vector. This rule can be interpreted so that the focal UAV makes decisions about its movement based on up to N closest neighboring UAVs (up to a horizon). The effect of this rule is shown in Figure 4.3 where the focal UAV (in the center of the green circle) uses only the UAVs within the green circle horizon to coordinate its movement and gradually decreases the radius of the horizon. When converged, we can see that all UAVs that are relevant for the decision-making of the focal UAV (those that are within the horizon radius) are correctly located within the desired distance limits d_{lb}, d_{ub} of the focal UAV (Figure 4.4).

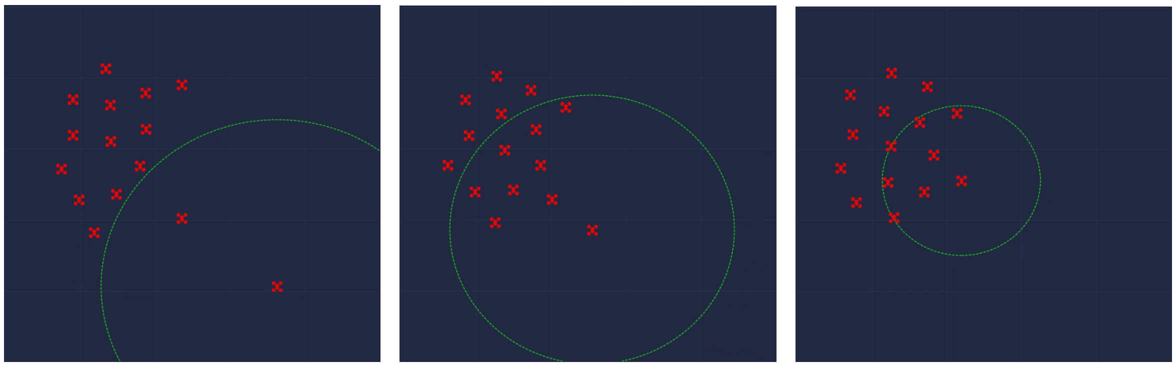

(a) t=1 (b) t=2 (c) t=3

Figure 4.3: Captured states of the swarm at three time steps while a UAV is approaching a group. The UAVs are red. The green circle marks the border of the adaptive horizon for the UAV in its center. The UAVs and the distances are not to scale.

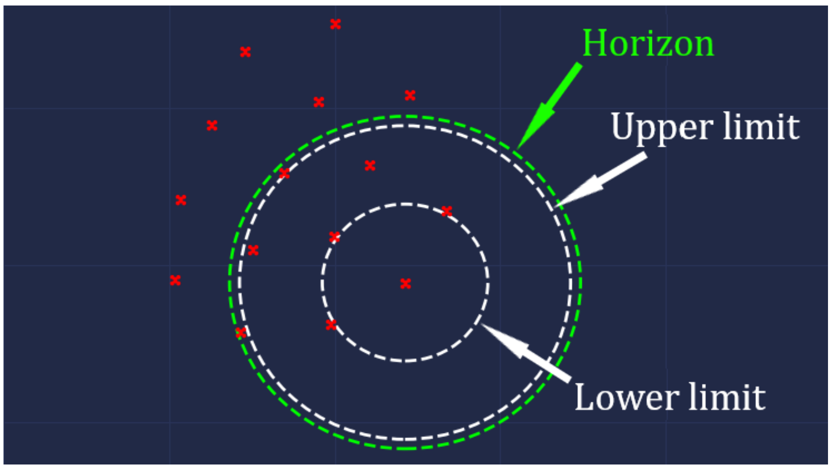


Figure 4.4: The image depicts the lower and upper limits of the controller for the situation $d_{lb} = 10, d_{ub} = 20$ and the horizon for UAVs relevant for control. The limits and horizon are shown for the UAV in the middle of the circles. The UAVs and the distances are not to scale.

There is also an option to steer the UAVs in a desired direction toward a goal location using the proposed controller. The navigation toward the goal is achieved by adding a new goal-oriented vector $\boldsymbol{v}_g$ to the previously calculated final control vector $\boldsymbol{v}_f$.

$$\boldsymbol{v}_g = \frac{\boldsymbol{g}}{||\boldsymbol{g}||}, \tag{4.7}$$

$$c = \begin{cases} c_{safe}, & \text{if } \sum_{i=1}^{d_{safe}} \sum_{j=1}^{9} \boldsymbol{l}_{ji}^{e} < 1 \\ c_{risk}, & \text{otherwise} \end{cases}, \tag{4.8}$$

$$\boldsymbol{v}_{fg} = \boldsymbol{v}_f + c \cdot \boldsymbol{v}_g, \tag{4.9}$$

where $\boldsymbol{g}$ is a vector representing the coordinates of the goal location relative to the focal UAV, c is a scaling coefficient that is used to prioritize safety (collision avoidance) over the goal following with d_{safe} as a parameter of the controller (typically $d_{safe} \leqslant d_{lb}$), and $c_{safe} > c_{risk}$. The scaling c is a rule that decreases the influence of the goal-oriented vector $\boldsymbol{v}_g$ in cases when there is a neighboring UAV within the radius of d_{safe} meters from the focal UAV which gives more relative weight to the control vector $\boldsymbol{v}_f$ to protect the UAV from collisions and return to full goal following when the way is clear.

Chapter 5

Results

Our method was tested on the synthetic dataset presented in Section 3.3 as well as on the real data discussed in Section 3.3.1. We also compared the method with a state-of-the-art alternative using an off-the-shelf object detector-based approach. Lastly, we explored the viability of the proposed method for swarm control using the swarm controller described in Section 4 developed specifically for our method of relative localization.

5.1 Metrics

We use multiple metrics to quantify the performance of the proposed method. We measure the results of the loss function introduced in Section 3.2. Which is suitable for comparing different versions of the neural network. However, it does not have a meaningful interpretation. Therefore, we use additional metrics that have more practical meaning and interpretation.

We propose measuring the error for each bin individually (the per-bin error). The error in a single bin, in this case, is the difference between the predicted amount of UAVs and the real amount of UAVs in the given bin:

$$e_i^j = p_i^j - l_i^j, \tag{5.1}$$

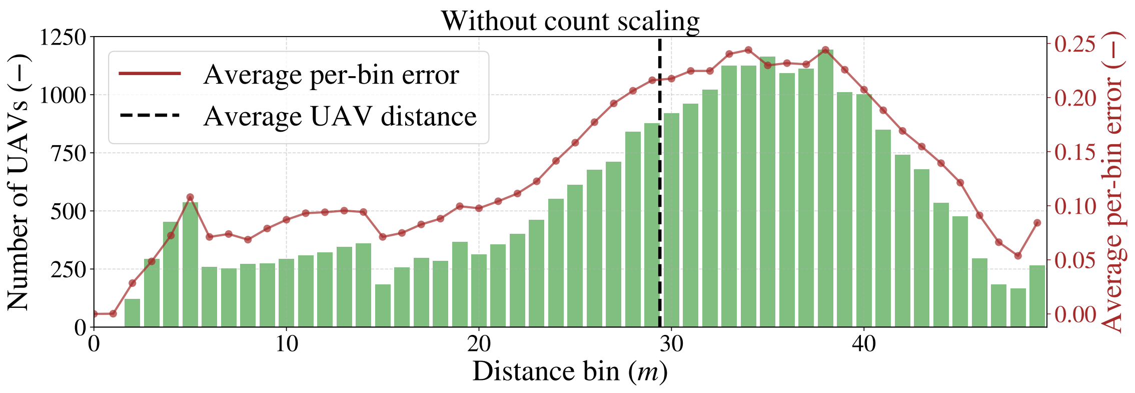
where e_i is the error in the i-th bin, p_i and l_i are the predicted and real amount of UAVs for the i-th bin, respectively. The upper index j indicates the j-th image. When computing the per-bin error on the test dataset we measure the average per-bin error $\bar{e}'_i$ for each distance bin i

$$\bar{e}'_i = \frac{\sum_{j \in \mathcal{I}} |e_i^j|}{|\mathcal{I}|}, \tag{5.2}$$

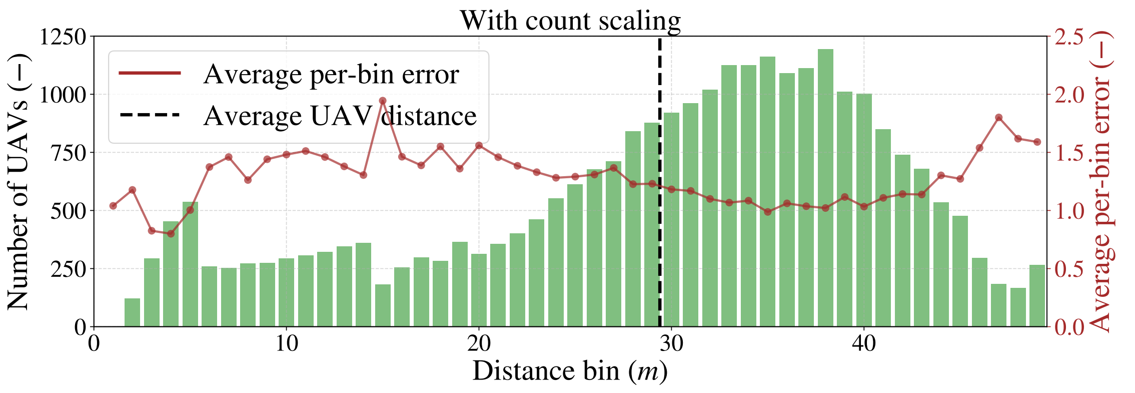
where $\mathcal{I}$ is the set of all images in the test dataset. However, it is evident from Figure 5.1a that $\bar{e}'_i$ is influenced by the amount of UAV occurrences in the given bin which makes it difficult to relatively compare the performance among individual bins. Therefore, we use a version of $\bar{e}'_i$ scaled by the number of UAV occurrences in the respective bin

$$\bar{e}_i = \frac{\sum_{j \in \mathcal{I}} |e_i^j|}{\sum_{j \in \mathcal{I}} n_i^j}, \tag{5.3}$$

where n_i^j is the number of UAVs in the i-th distance bin of the j-th image. The results of this modification are presented in Figure 5.1b.



(a) Error without compensated counts.



(b) Error with compensated counts.

Figure 5.1: Example of compensating for the number of UAVs in an image to better compare results of individual bins.

Intuitively, it may be expected for the per-bin error to rise with the distance (at higher distances, there is less difference in the dimensions of the projection, fewer details and features are distinguishable, etc.). However, it can be observed in Figure 5.1b that the characteristic of the per-bin error is not rising but rather almost constant with increasing distance. This is caused by the nature of the data and the fact that our detection is not object-specific. If there is a close UAV in an image then it is likely that it does not have many neighbors depth-wise (e.g. it is physically impossible to fit 7 UAVs 3 meters away from the camera in the limited FoV of the assumed camera without collisions). On the other hand, our data often contain distant UAVs grouped together. If we assume a hypothetical scenario with a single UAV in some close bin and the CNN miss-estimates its distance by one bin, then there is an error equal to one UAV (neglecting the label smoothing) in the bin that the UAV was assigned to (because there should be none) and an error equal to one UAV in the bin that should contain the UAV but does not. The error is equal to two UAVs in total for the first scenario.

Now, let us assume a second scenario where there is a group of multiple UAVs in multiple neighboring distance bins further away. If again the estimate missed by one bin for all of the UAVs and estimated each of them to be one bin closer to the camera then there would be an error equal to 1 UAV for the closest UAV that is assigned to an empty bin and an error equal to 1 UAV for the furthest UAV because it is missing in its bin. The rest of the shifted estimates does not produce any error because they are in a place where an estimate should be (and it does not matter which UAV is the source since it is not object-specific). The second scenario produces again an error equal to 2 UAVs.

However, the error $\bar{e}'_i$ in the first scenario is divided by a lower number (since there are on average fewer UAVs in the close bins) and the error from the second scenario is divided by a larger number (since there are on average more UAVs in the further bins). Therefore, the error in the more distant bins can decrease.

To test this explanation, we created a dataset where each image contained exactly 1 UAV. We can observe that in this case, the effect described above is mitigated as expected, and the error does rise with distance as shown in Figure 5.2. However, testing only on the data only containing a single UAV in an image provides no information about the counting and scaling potential of the proposed method. Therefore, we use the original dataset from Figures 5.1a, 5.1b for further testing.

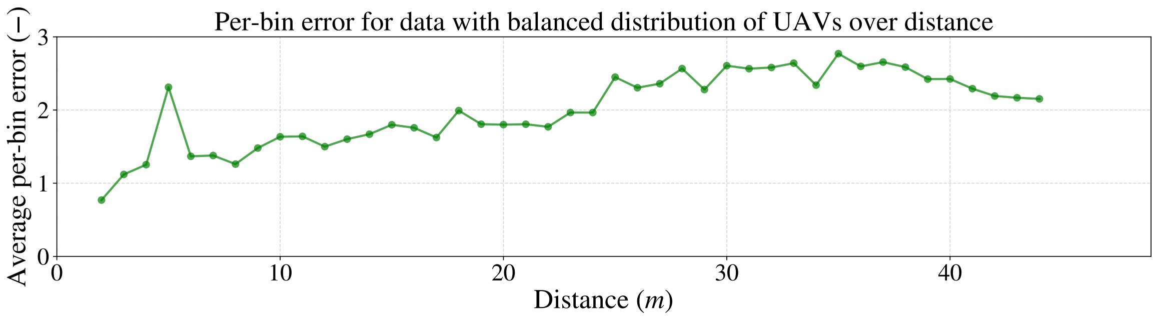


Figure 5.2: Per-bin error for a dataset with dataset balanced UAV counts over distance.

The per-bin error defined in eq. 5.1 shows which distances are more susceptible to inaccuracies but this error has two fundamental sources. The first cause is an object that is not a UAV counted as a UAV and vice versa (when a UAV is not recognized by the network). The second source is an incorrect estimation of the distance of a UAV. These two causes represent different problems and yet cannot be distinguished solely based on the per-bin error. Therefore, we measure the total amount of UAVs in the whole image in addition to the per-bin error. This approach allows us to estimate which of the above-mentioned two causes is the main source of error for a given image. We also use the relative version of this error which is scaled according to the amount of UAVs within the image because the absolute value of the error and its practical significance depends on the total amount of UAVs (similar to the scaling issue mentioned with the per-bin error). The total amount error for a single image is defined as

$$T^j = \left| \sum_{i=1}^{n_{bin}} p_i^j - \sum_{i=1}^{n_{bin}} l_i^j \right|, \qquad T_r^j = \frac{T^j}{\sum_{i=1}^{n_{bin}} l_i^j}, \tag{5.4}$$

where T^j is the absolute total amount error and T_r^j is the relative total amount error, p_i^j and l_i^j are the predicted and real amounts of UAVs for the i-th bin respectively. If T_r^j is low, we can assume that most of the per-bin error is caused by inaccuracy in the distance estimation since the image contains approximately the correct amount of UAVs but distributed differently within the distance bins of the image. Large T_r^j suggests that the neural network has more significant issues with recognition of the UAVs themselves rather than distance estimation issues.

5.2 Experiments

We used a synthetic dataset as described in Section 3.3 consisting of 5000 images. The per-bin errors defined in eq. 5.1 are plotted in Figure 5.3. It may be observed that the error is mostly symmetric with means close to zero.

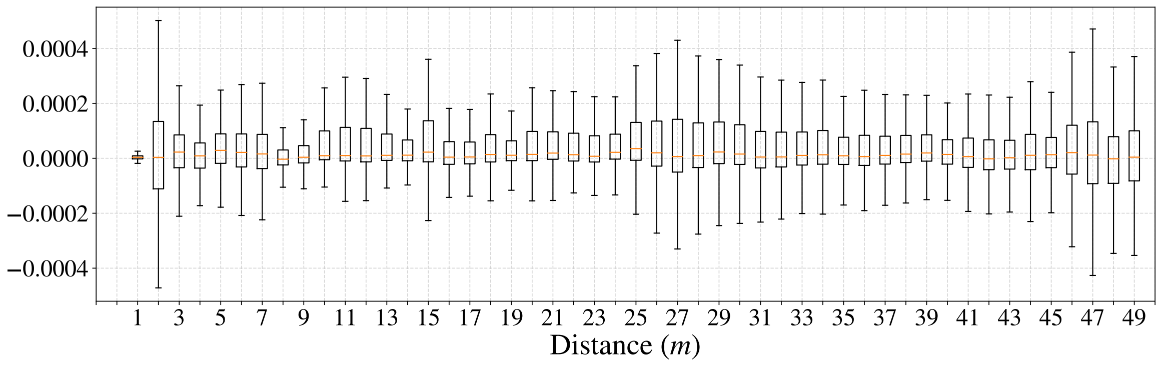
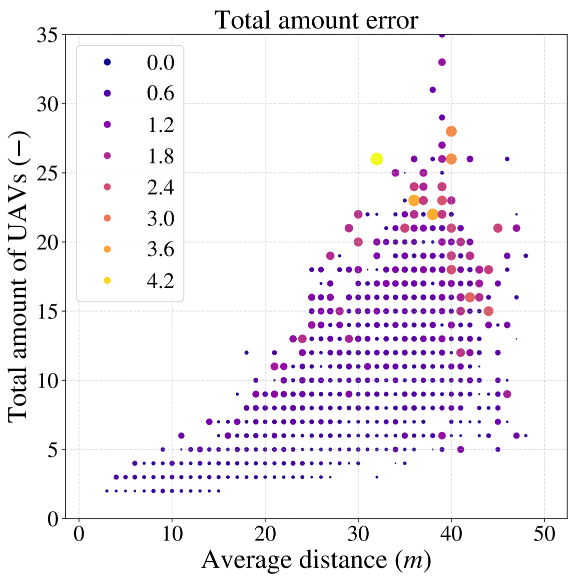


Figure 5.3: Boxplots of the per-bin error (eq. 5.1) on the testing dataset. Values of the error are scaled by the UAV counts (for better relative comparison between the bins) individually and therefore reach very low values and should not be interpreted as the actual per-bin error.

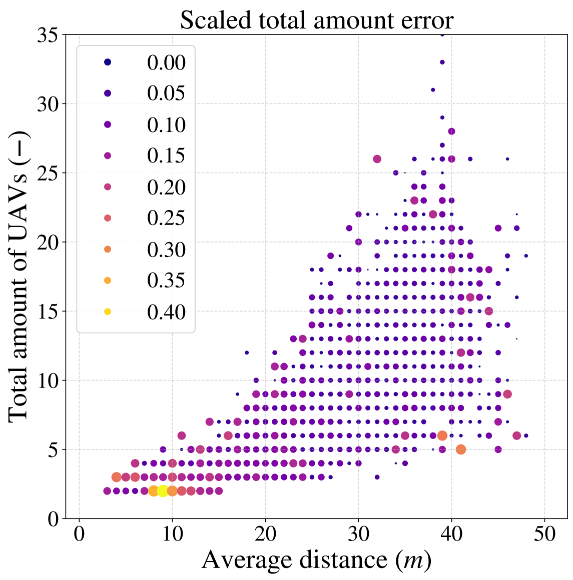
Furthermore, the performance of the total and relative amount errors (eq. 5.4) are analyzed. The results are divided into categories based on the total amount and the average distance (rounded into the correct bin) of the UAVs within the given image (from small sparse groups to large dense swarms). The average distance is calculated as

$$\bar{d} = \frac{1}{\sum_{j=1}^{n_{bin}} l_j} \sum_{i=1}^{n_{bin}} l_i \cdot i, \tag{5.5}$$

where $\bar{d}$ is the average distance and l_i is the real amount of UAVs in the i-th bin. The average total amount error is computed for each of the categories as shown in Figure 5.4a. The graph shows that the absolute error rises with the total amount of UAVs present in the image as well as with the distance. By comparing the y-axis of the relative total error (Figure 5.4b) it may be concluded that the total error is proportional to the number of UAVs in the image. It can be observed that the relative total error is largely similar. The average relative total amount error per UAV was calculated as 0.142. Therefore, it can be concluded that the main source of the per-bin error is not a misrecognition of the UAVs but rather inaccuracies in distance estimates and noise.



(a) The absolute total amount error (the number of missing or redundant UAVs).



(b) The relative total amount error (the number of missing or redundant UAVs per a unit UAV).

Figure 5.4: Comparison between the total amount error and the relative total amount error (eq. 5.4). The value of the error is represented by the size and color of the corresponding point.

5.2.1 3D Output

So far, it was assumed that $w_{out} = h_{out} = 1$. Next, we test the influence of modifying the architecture by increasing $w_{out} = h_{out}$. Each drone is assigned not only to a particular distance but also to one of the nine subregions in the image. For this purpose, we used different labels for the same dataset. The new label for an image contains not one but nine distance-density vectors, one for each of the subregions. The subregions are squares of 100x100 pixels forming a 3x3 matrix over the image.

We performed measurements to show that the modification from 1D to 3D output does not significantly impact the performance of the network. The per-bin errors for all distance bins are shown in Figure 5.5). To obtain the graph of the 3×3 version, all of the nine distance-density vectors (one for each subregion) were averaged per-bin. The errors are overall similar amongst the 1×1 and 3×3 outputs except for the first bin because of a lack of data (only 1 UAV). The average relative total amount error per UAV is 0.174.

The absolute value of the per-bin error as shown in eq. 5.3 is used for clearer visualization and comparison in Figure 5.6. It can be observed that the values of the error are virtually identical between the two versions except for the first bin.

It may be concluded that increasing the output dimensions does not decrease the performance on the original task (distance distribution estimation). Furthermore, despite the new functionality and the significant increase in the size of the output, the number of learnable network parameters is larger by only 10% and the inference time is only $2.5\times$ longer for the 3×3 version as shown in Table 5.1.

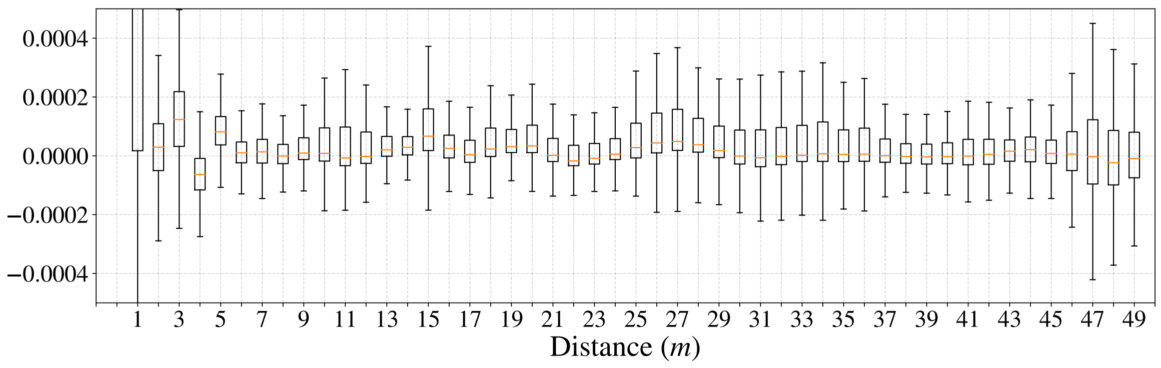


Figure 5.5: Boxplots of the per-bin error (eq. 5.1) for the 3×3 version of the network. Values of the error are scaled by the UAV counts (for better relative comparison between the bins) individually and therefore reach very low values and should not be interpreted as the actual per-bin error.

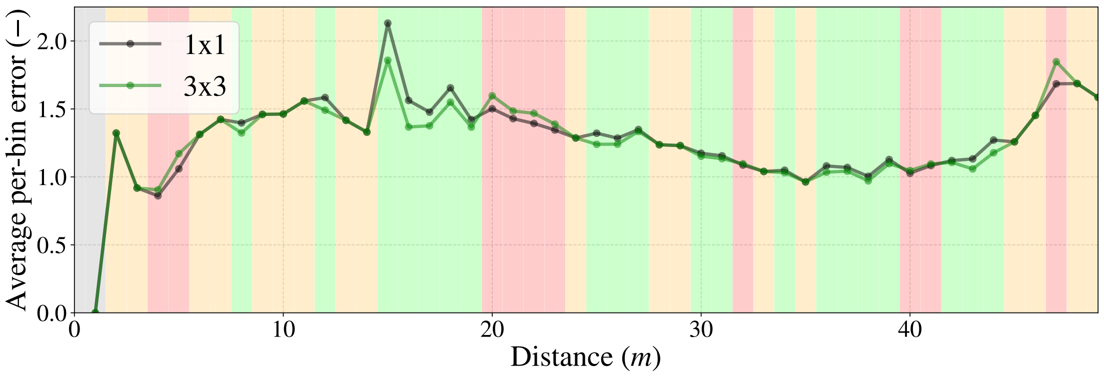
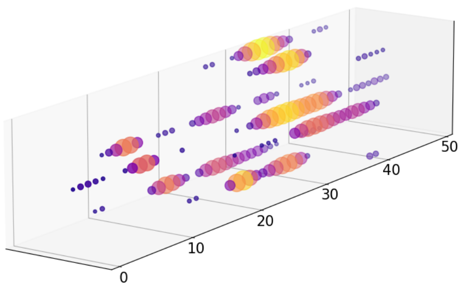


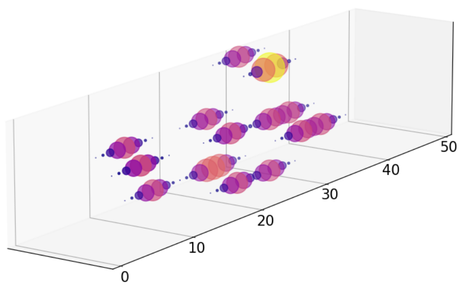
Figure 5.6: The average absolute per-bin error comparison for each distance bin. The background colors indicate whether the 3×3 version outperforms the 1×1 version in the given bin (green) or not (red) or they equal (orange).

A visualization of the prediction results for the 3×3 case is presented in Figures 5.7 and 5.8. The prediction results are filtered for visualization purposes (values very close to zero are replaced by 0). It is clearly recognizable where are the UAVs located within the subregions and distance bins. Both figures also demonstrate the robustness of our proposed method towards overlapping UAVs that partially obscure each other. All 16 and all 3 UAVs respectively were counted in both images including their correct placement within the 3D space.

5.2.2 Real-world data

The data described in Section 3.3.1 were used for this experiment. We used 30 images for fine-tuning the network that was pretrained using the synthetic dataset. The other 10 images were used as both validation and testing data because of the limited amount of images. The results on the real-world data improved after fine-tuning and the results on the synthetic


(a) Prediction


(b) Ground truth

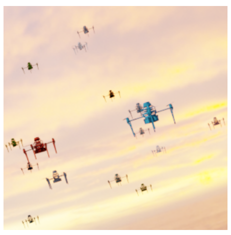
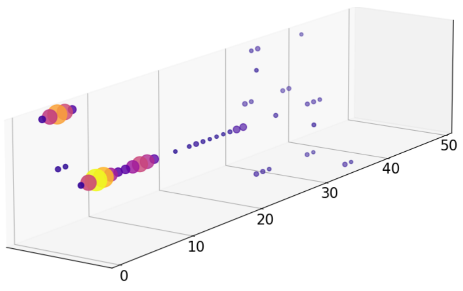
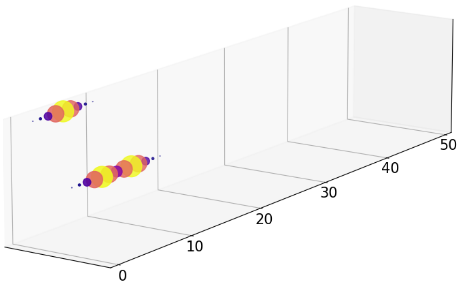

(c) Input

Figure 5.7: Graph of the predicted and ground-truth UAV distributions over distance in each subregion based on the image on the right. The longest axis describes the distance in meters. There are nine distance-distribution vectors in a 3×3 grid arrangement. The amount of UAVs for a given distance is represented by the distinct color and size. All 16 UAVs were detected.


(a) Prediction


(b) Ground truth

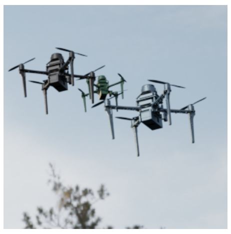
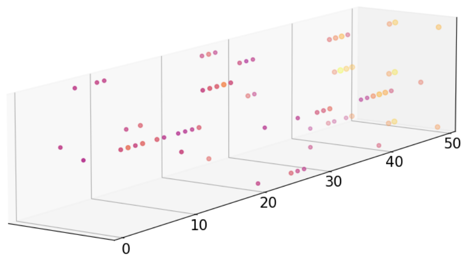

(c) Input

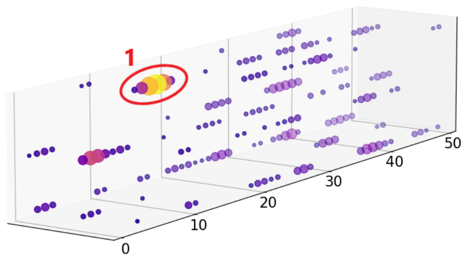
Figure 5.8: Graph of the predicted and ground-truth UAV distributions over distance in each subregion based on the image on the right. The longest axis describes the distance in meters. There are nine distance-distribution vectors in a 3×3 grid arrangement. The amount of UAVs for a given distance is represented by the distinct color and size. All 3 UAVs were detected.

data got worse but that is expected given the small amount of real-world data and their imperfections. There is too few testing data to provide any useful statistical results, however, we used the 3×3 version of the network to better visualize the performance on the real-world data. Figure 5.9 shows that the neural network trained on the synthetic data and further fine-tuned with as little as 30 real images is superior in improving performance on the real-world data to both purely synthetic trained and purely real-world data trained (with a limited amount of data) model.

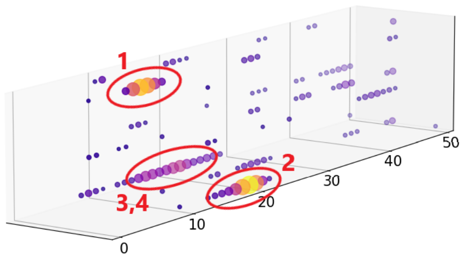
Figure 5.9 shows an example with 4 UAVs in the input image, the amount corresponding to 4 UAVs was estimated in the purely synthetic trained and fine-tuned cases. The amount corresponding to only 2 UAVs was estimated in the purely real-trained case. This image is more challenging due to the forest background which makes it difficult to detect the UAVs in the front. It is clear that the best results come from the model that was trained synthetically and fine-tuned on real-world data. The UAVs were highlighted and enumerated as well as their corresponding clusters in the estimate if correctly predicted.



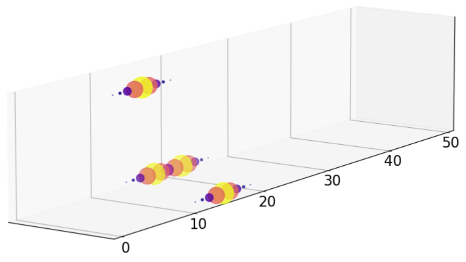
(a) Purely real-trained



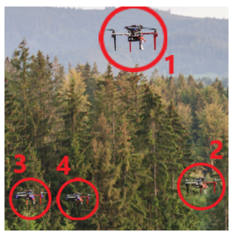
(b) Purely synthetic-trained



(c) Synthetic-trained and real-tuned



(d) Ground truth



(e) Input

Figure 5.9: Predictions of various models. The longest axis describes the distance in meters. The amount of UAVs for a given distance is represented by the distinct color and size. The UAVs are highlighted and enumerated together with their correct estimates.

5.2.3 Object detection comparison

We show that the proposed approach performs better than an off-the-shelf state-of-the-art solution by comparing our method with an approach based on object detection. We assume a hypothetical perfect object detector with an output identical to the ground-truth label. The output of the object detector is in the form of bounding boxes, which is a rectangle that is as small as possible and still includes the entire object of interest. In our case, the output is one bounding box for each UAV.

The first step of the off-the-shelf state-of-the-art solution is to extract the bounding box for each UAV. The second step is to estimate the distance based on the dimensions of the bounding box. In order to estimate the distance, the width and height of the bounding box are measured and the nearest neighbor method is used to find the most fitting distance. Average widths and heights are measured for each distance bin in the training dataset and then used for the nearest neighbor method. The dimensions of each extracted bounding box are compared to the average dimensions for each distance bin and the average dimensions closest to the extracted dimensions determine the estimated distance bin.

We show the absolute value of the per-bin error (eq. 5.3) in Figure 5.10. The data show that our method outperforms the off-the-shelf state-of-the-art solution in the majority of the distance bins. However, the perfect object detector performs better in the closer bins because the difference between the dimension of the bounding boxes is relatively large at low distances as presented in Figure 5.11 (closer distances are clearly more discernable compared to the further ones).

Despite assuming the perfect object detector (no misdetection, no noise, perfectly fitted bounding boxes) the method has a disadvantage compared to our approach. The width and

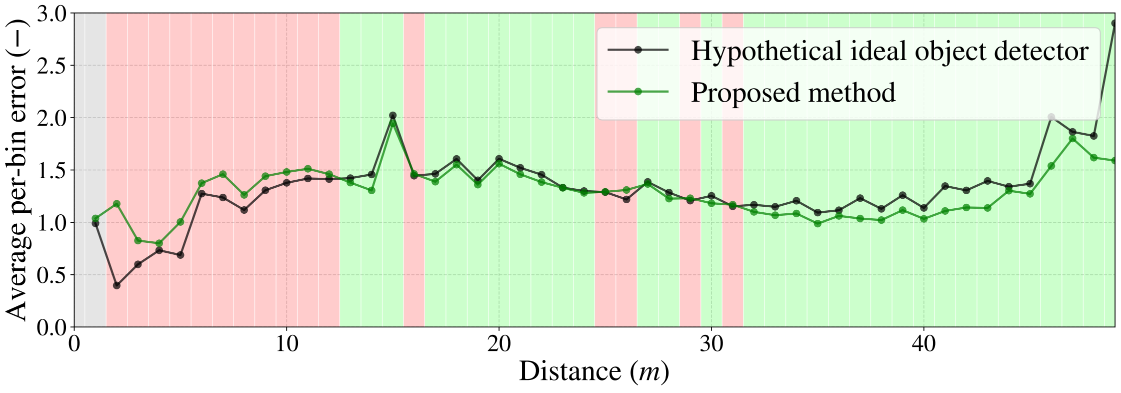


Figure 5.10: The average absolute per-bin error comparison for each distance bin. The background colors indicate whether the proposed method outperforms the object detector in the given bin (green) or not (red).

height of the bounding box do not always measure the same dimensions of the UAV. This is caused e.g. by the different yaw, tilt, or perspective of the UAV in the image which causes the bounding box dimensions to correspond to a different part of the UAV (not only the width and height of the UAV). The detector-based method then estimates a wrong distance because the dimensions gradually become very similar with a rising distance as can be seen in Figure 5.11.

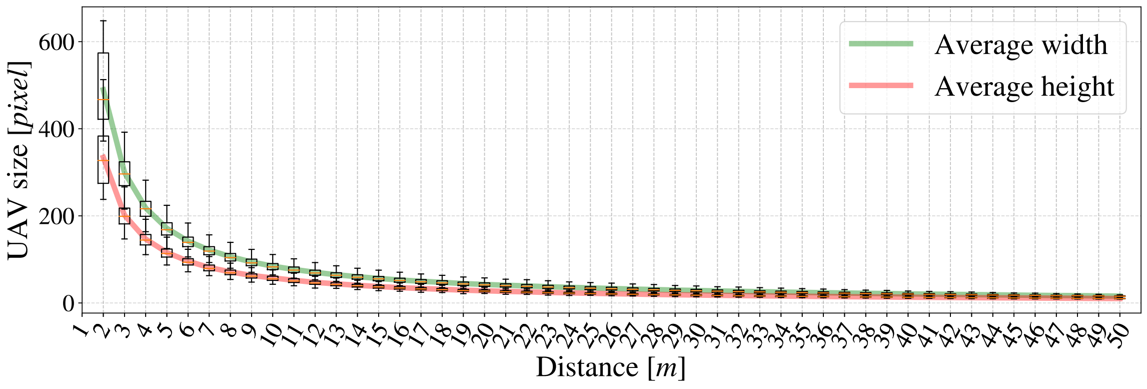


Figure 5.11: The measured average width and height of the bounding boxes of UAVs at every distance bin together with boxplots to visualize the overlaps between nearby distance bins.

Our method does not suffer from this issue. Because it can learn to detect features that are independent of rotation or by learning different combinations of features for different perspectives. A representative example is shown in Figures 5.12, 5.13 where the UAV with the camera is tilted which makes the rest of the swarm appear tilted relative to the focal UAV. The shift needed for the estimate of our method to fit the ground truth the best is 1 in Figure 5.12, whereas the shift for the object detection approach is 3 meters. The shift of our method in Figure 5.13 is 1 meter and 5 meters for the object detector. There are 5 UAVs in Figure 5.12 and 14 UAVs in Figure 5.13. Our method detected 5.3 and 14.7 UAVs respectively.

In addition to the perfect object detector, we trained a real object detector (EfficientDet

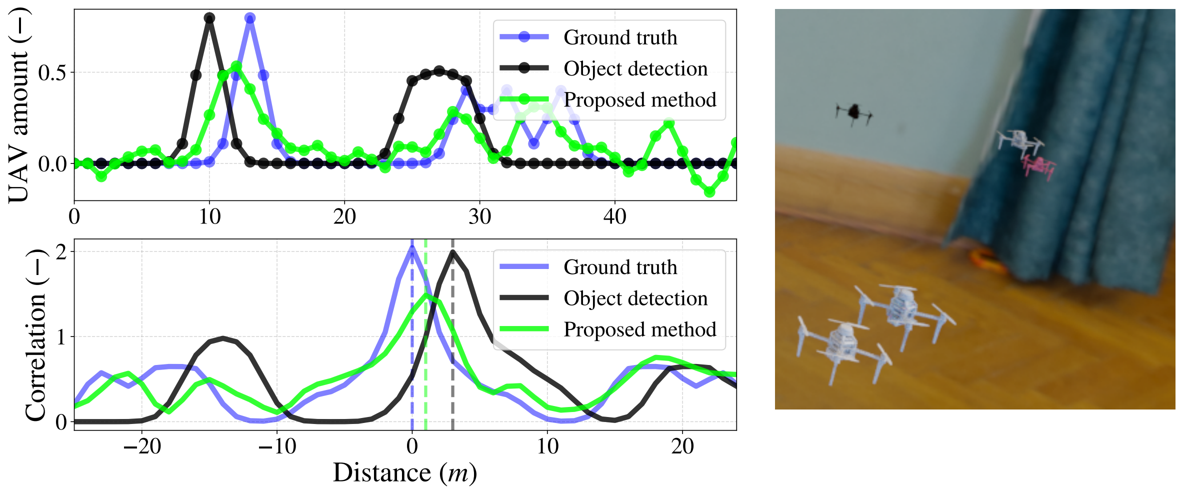


Figure 5.12: Top left: The UAV distributions over the distance bins. Bottom left: The correlation results of each of the curves from the upper graph with the ground truth and the distance shift with the highest correlation value (needed for the best overlap of the curves) is marked with a vertical line. Right: The input image.

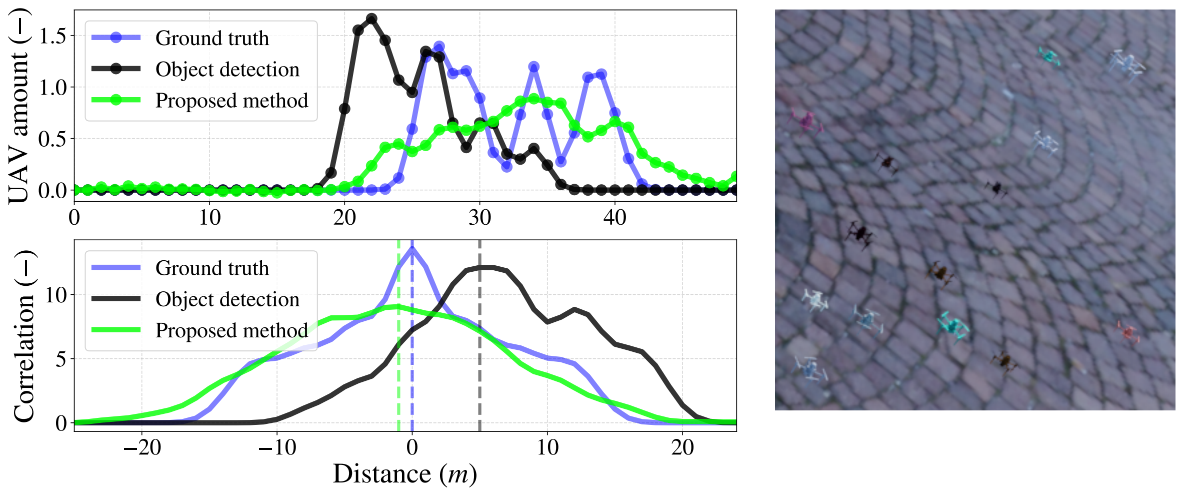


Figure 5.13: Top left: The UAV distributions over the distance bins. Bottom left: The correlation results of each of the curves from the upper graph with the ground truth and the distance shift with the highest correlation value needed for the best overlap of the curves is marked with a vertical line. Right: The input image.

[50]) and tested it on the same data. The real object detector is outperformed by our approach even in the close bins as shown in Figure 5.14.

5.2.4 Ablation study - Weighted loss

To show the effect of the weighted loss function as introduced in Section 3.2, we train one network using the weighted loss and a second network without the weights in the loss

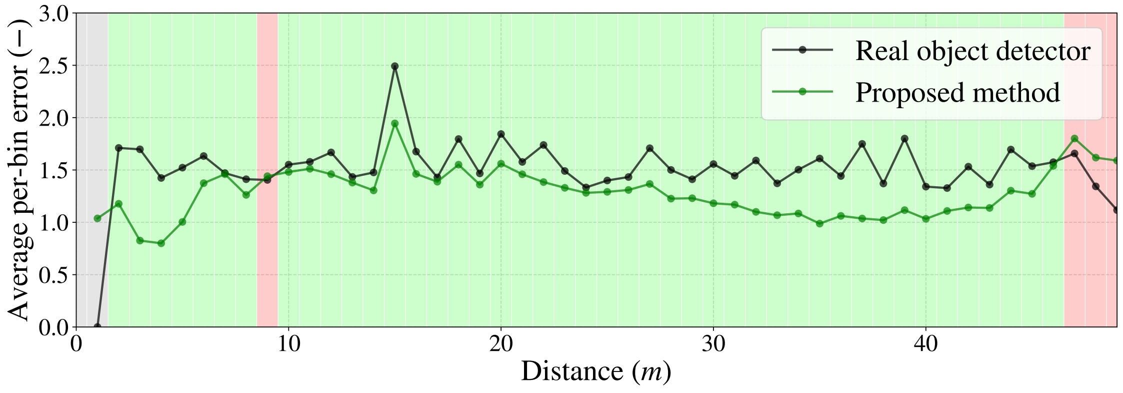


Figure 5.14: The average absolute per-bin error comparison for each distance bin. The background colors indicate whether the proposed method outperforms the object detector in the given bin (green) or not (red).

function (all the weights were set to 1). The vector of weights employed in the weighted version has the following values

$$w_i = \begin{cases} 2, & \text{if } i \in \langle 0, 8), \\ 4, & \text{if } i \in \langle 8, 25), \\ 1, & \text{if } i \in \langle 25, 50), \end{cases} , \tag{5.6}$$

where w_i is the weight corresponding to the i-th distance bin. The weights were determined empirically. The requirement for the weights is to improve the prediction results in the bins closer to the focal UAV that require a faster reaction time (e.g. for collision avoidance) than the further distance bins without significantly decreasing the performance on the bins further away (partial decrease is expected since the weights only shift the attention to the closer bins). The first 8 bins have lower weights than the rest of the weights up to the 25th bin because the first 8 bins have different type of labels than the rest as described in Section 3.3 which leads to higher errors.

The results are summarized in Figure 5.15. It shows a clear improvement within the lower-distance bins while keeping a similar performance for the more distant bins.

5.2.5 Ablation study - Smoothed labels

To support our choice of partially smoothed labels using the Gaussian kernel as described in Section 3.3, we present a comparison with two alternative labeling methods. Three models with identical architectures and hyperparameter values were trained for 30 epochs each. The best model weights were chosen from each training session as described in Section 3. One model used the training and validation data with the labels fully smoothed corresponding to $k = 0$ in eq. 3.9. The second model used the training and validation labels in the raw format that counts each UAV into exactly one distance bin as presented in Figure 3.6. The third model used the combined labels proposed in Section 3.3. We chose the raw labels for testing because it represents the true distribution of the UAVs. It can be observed in Figure 5.16 that the model trained on the raw labels performs decently at close distances but gives

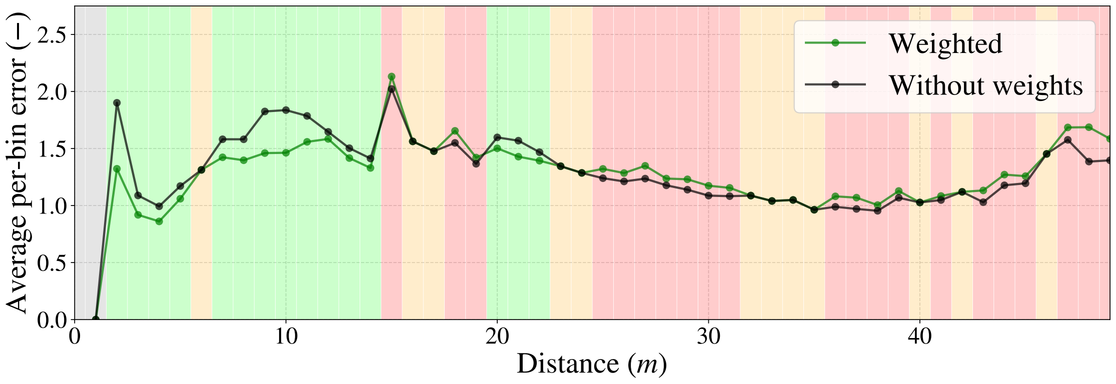


Figure 5.15: The average absolute per-bin error comparison for each distance bin in the ablation experiment comparing different loss function weighting. The background color marks where the network with the weighted loss function is not outperformed by the unweighted version (green), performs equally well (orange), or worse (red).

worse results further away The opposite is true for the model trained on the fully smoothed labels. The results of the third model show that the best performance can be achieved by using a combination of both approaches. The combination allows the network to use its full potential at close distances where the bin can be estimated more precisely while still retaining a better performance at larger distances.

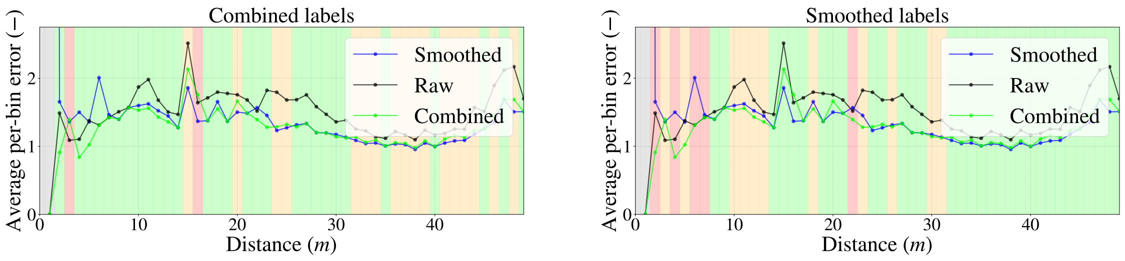


Figure 5.16: The average absolute per-bin error comparison for each distance bin in the ablation experiment comparing different labels. The background colors in the left Figure mark whether the network trained on combined labels is not outperformed by the other (green) or outperformed by one of the alternatives (orange) or by both (red). The right Figure shows the colors for the smooth labels respectively.

5.2.6 Ablation study - 1x1 vs FC

To substantiate the choice of the architecture of the neural network. We compared the effect of employing the 1x1 convolution layer as described in Section 2.2.2. With a more conventional choice of a fully connected layer consisting of 50 nodes as an analogy to the 50 filters of the 1x1 convolution. The average pooling layer was omitted due to the replacement. We trained each of these versions for 30 epochs. Adding the fully connected layer increased the number of parameters to 285% compared to the proposed 3×3 architecture. However, the performance declined despite the significant increase in learnable parameters as shown in

Figure 5.17.

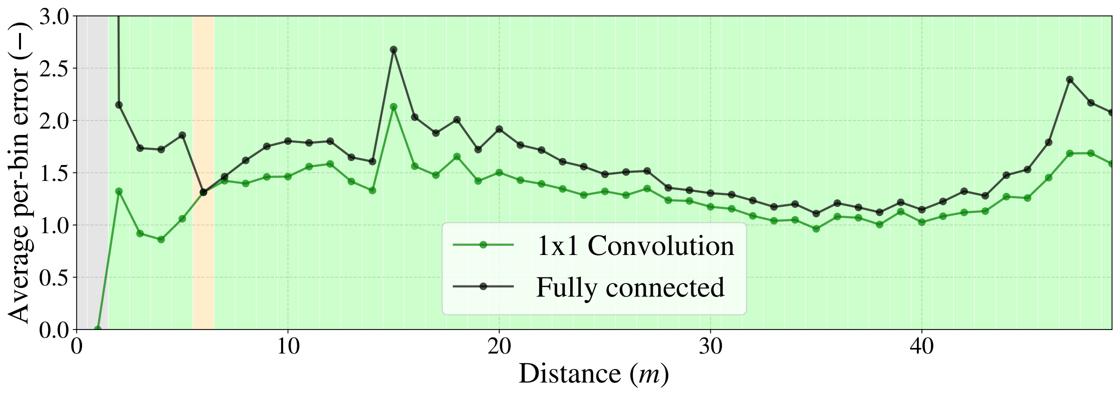


Figure 5.17: The average absolute per-bin error comparison over distance in the ablation experiment comparing different network architectures. The background color marks whether the 1×1 network is not outperformed by the fully connected version (green) or performs equally well (orange).

5.2.7 Neural network experiments summary

Table 5.1 compares all of the mentioned approaches. We compare the methods using the average per-bin error over all distances $\bar{E}$ and the average per-bin error over distances up to the 11-th bin $\bar{E}'$, which represents the performance of the method in the close bins critical for collision avoidance. Both errors are mathematically expressed as

$$\bar{E} = \frac{1}{n_{bin}} \sum_{i=1}^{n_{bin}} \bar{e}_i, \qquad \bar{E}' = \frac{1}{11} \sum_{i=1}^{11} \bar{e}_i, \tag{5.7}$$

where $\bar{e}_i$ is the error from eq. 5.3. Furthermore, we compare the total amount relative error T_r (eq. 5.4), the number of extra parameters of the model, and the execution time of the model t_{exec}. All of the errors $\bar{E}$, $\bar{E}'$, and T_r are averaged over all testing samples.

5.2.8 Camera compensation

The ability of our method to correctly estimate the distance independently of the tilt, yaw, and perspective (as shown in Section 5.2.3) is crucial for the camera compensation approach (Section 3.3.2) to work well. That is because the projected size of the object (denoted as h in eq. 3.10) must be measured independently of the relative pose of the given object (UAV in this case).

To test the camera compensation, we emulate a camera with different projection parameters by selecting different crops of the original images. In the cases shown in Figures 5.18 and 5.20 we crop a region of the image (including the UAV) of size 300×300 pixels and then another image of size 600×600 pixels. After downsampling the second image from 600×600 to 300×300 pixels it has the same appearance as an image taken by a camera with

CNN variant	T_r	$\bar{E}$	$\bar{E}'$	params	t_{exec}
Ours (3×3)	0.174	**1.30**	1.27	100%	18 ms
Ours (1×1)	**0.142**	**1.31**	**1.25**	91%	7 ms
F.C. layer (3×3)	0.261	1.60	1.71	285%	8 ms
raw labels (3×3)	0.410	1.53	1.42	100%	18 ms
smooth labels (3×3)	**0.119**	1.34	1.55	100%	18 ms
SotA detector [50]	0.963	1.55	1.54	168%	N/A
ideal detector	0 (N/A)	1.34	**0.96**	N/A	N/A

Table 5.1: Comparison of all considered variations of the relative localization approach using various metrics. The two best results are highlighted in bold. The ideal detector has no parameters (since the labels are considered its output). The execution time t_{exec} is left undefined for the detectors because of unexpected technical circumstances that prevented an accurate measurement.

parameters

$$f_n \cdot r_n = \frac{f_t \cdot r_t}{2}, \tag{5.8}$$

where f_n,r_n,f_t, and r_t are the focal length and resolution of the new simulated camera and the training camera, respectively, because the image was downsized by a factor of 2.

For Figure 5.19 we cropped regions of size 300×300 and 200×200 pixels and then upsampled the 200×200 region to 300×300 pixels. This simulates a camera satisfying

$$f_n \cdot r_n = \frac{3}{2} \cdot f_t \cdot r_t, \tag{5.9}$$

where f_n,r_n,f_t, and r_t are the focal length and resolution of the new simulated camera and the training camera respectively because the image was upsampled by a factor of $\frac{3}{2}$.

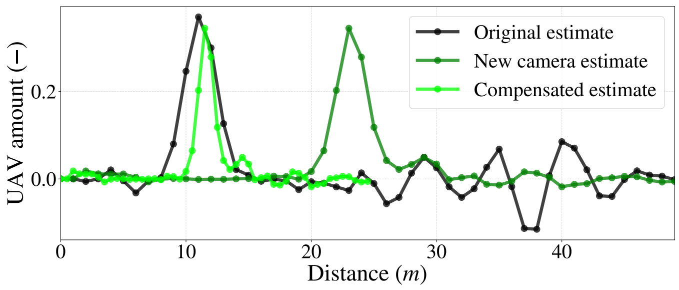
Figures 5.18, 5.19, 5.20 show the above-mentioned cases. The original estimate is the estimate of the neural network for the non-modified 300×300 pixel versions of the images (the images labeled as "Original"). The new camera estimate is the estimate of the network for the resampled version of the original image (images labeled as "New camera"). Finally, the compensated estimate is the result of the compensation transformation (eq. 3.14). All of the used images contain one UAV for clarity and all of the estimates correctly counted one UAV (despite the differences in estimated amounts in Figure 5.18 because the original estimate returned 1.32 UAV amount and the new camera estimate was 0.98 UAV).

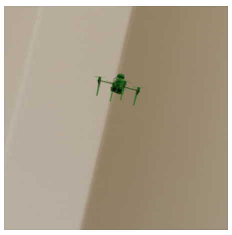
5.3 Swarming controller simulation

The swarming controller introduced in Section 4 was tested in a simulation. The values for the parameters used for the simulations are stated in Table 5.2.

Parameter	d_{lb}	d_{ub}	d_{safe}	c_{safe}	c_{risk}	N_{h}
Value	10 m	20 m	8 m	1	0.3	6

Table 5.2: Parameter values used for the simulated swarming experiments.




(a) Original

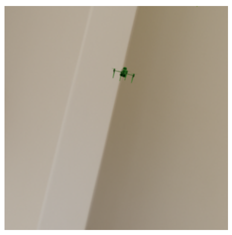
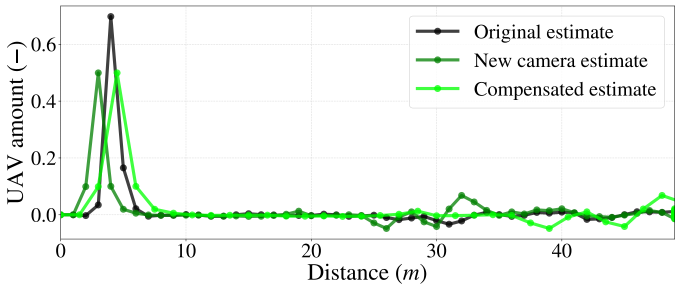
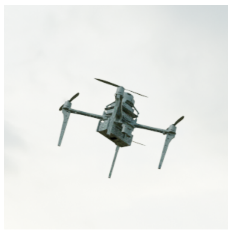

(b) New camera

Figure 5.18: The estimated UAV distributions over distance based on the images on the right and the compensated estimate.




(a) Original

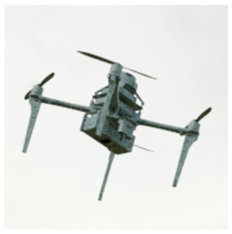
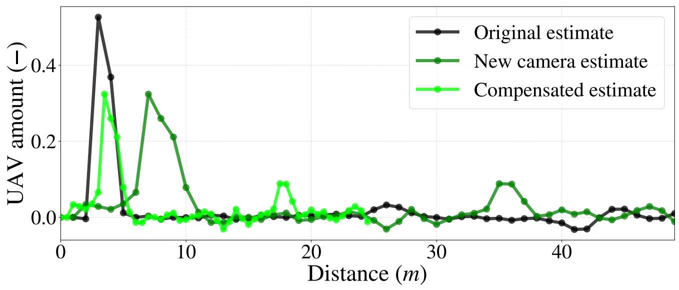
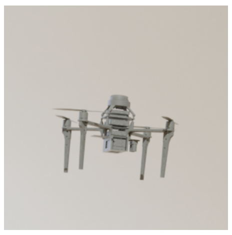

(b) New camera

Figure 5.19: The estimated UAV distributions over distance based on the images on the right and the compensated estimate.




(a) Original

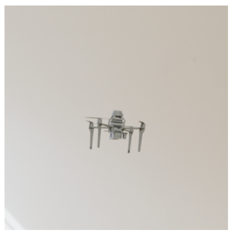

(b) New camera

Figure 5.20: The estimated UAV distributions over distance based on the images on the right and the compensated estimate.

We use two metrics for evaluation of the performance of the swarm controller similar to those presented in [51]. The two metrics are the minimum and the maximum inter-agent distance

$$d_i^{min} = \min_{\substack{j \in \langle 1, N \rangle \\ i \neq j}} ||\boldsymbol{r}_{ij}||, \qquad d_i^{max} = \max_{\substack{j \in \langle 1, N \rangle \\ i \neq j}} ||\boldsymbol{r}_{ij}|| \tag{5.10}$$

where d_i^{min}, d_i^{max} are the corresponding distances with respect to the i-th UAV, N is the number of UAVs, and $\boldsymbol{r}_{ij}$ is the vector connecting the i-th and j-th UAV. These metrics are chosen because the minimum distance d_i^{min} describes the ability of the controller to avoid collisions. The maximum distance, on the other hand, shows whether the swarm stays coherently together or if the UAVs disperse into the surroundings.

The covariance matrix used to generate the error for the simulation as explained in Section 4 is visualized in Figure 5.21. We interpret the covariance matrix as an indicator that shows the increased spread of the error over the distance bins in the more distant bins. We consider this to be the result of the fact that it is difficult to maintain the same precision with a rising distance from the camera.

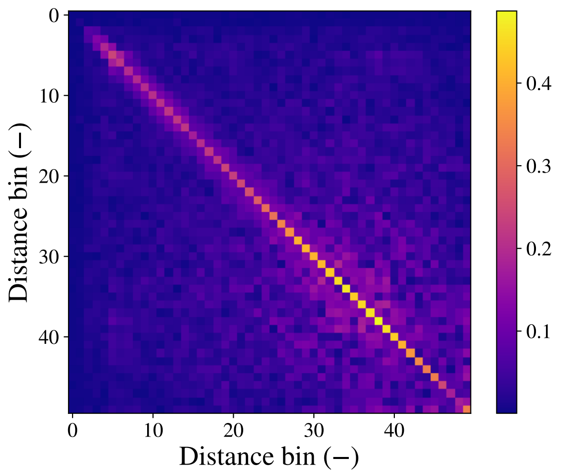


Figure 5.21: Visualization of the covariance matrix of the per bin errors on testing data (in absolute value).

5.3.1 Experiments

Firstly, a scenario was tested using 50 UAVs. The UAVs had no particular goal and followed the rules of the swarming controller. The UAVs were spawned in a grid-like formation with a distance of 10 meters between the neighboring UAVs. The UAVs hovered until they settled and hovered without any significant changes. We measure the minimum and maximum distances between any two UAVs. The minimum distance between any two UAVs was 9.5 meters and the maximum distance was 92.2 meters as presented in Figure 5.22. These results show that the swarm was coherent and without any collisions. It can be noticed that the minimum inter-UAV distances slightly increase and stabilize above the 10-meter distance since $lb = 10$ (eq. 4.3).

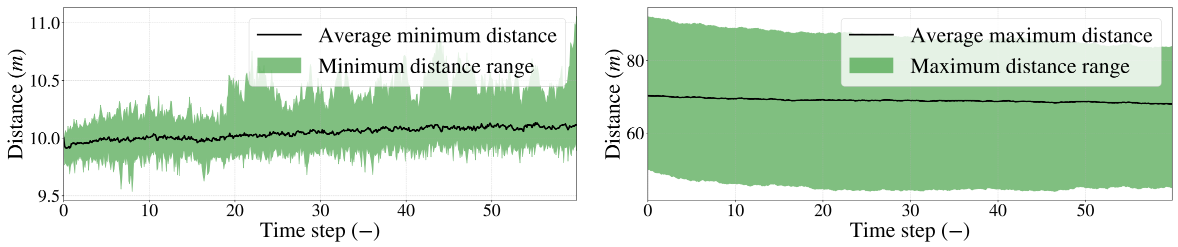
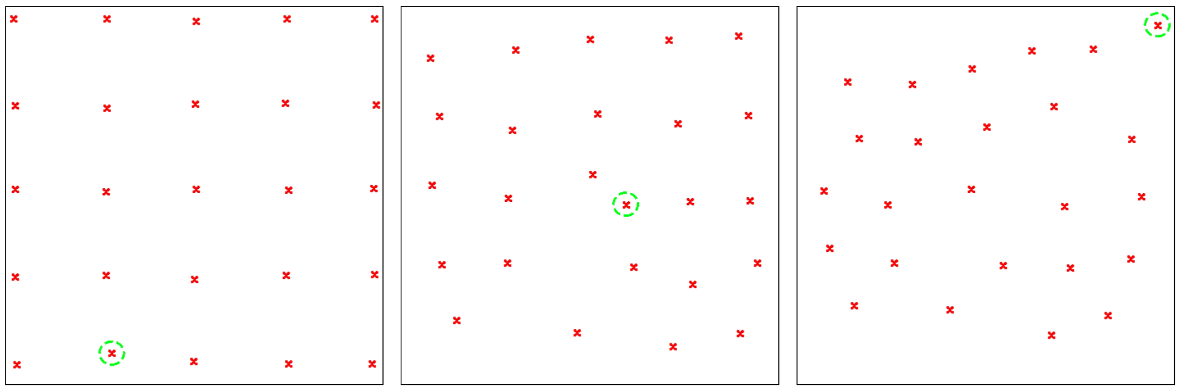


Figure 5.22: Overview of minimum and maximum distances d^{min}, d^{max} over time for the first scenario with 50 hovering UAVs. The green interval is the smallest interval that encapsulates the distances with respect to all the UAVs in that time frame.

The second scenario tested the reaction of the swarm when intercepted by a UAV that does not use our relative localization method or the swarm controller illustrated in Figure 5.23. This applies to situations where for example the cameras of one UAV stop working. The setup for this scenario was similar to the first. The UAVs started in a grid-like formation with 15-meter distances between neighboring UAVs. One of the UAVs on one side of the swarm had the swarm controller disabled and a goal was set for this UAV that was located on the other side of the swarm. The UAV followed directly through the swarm with a constant speed towards the goal. We measured again the minimum distance between any two UAVs which was 6.9 meters and the maximum distance which was 84.9 meters (Figure 5.24). We can observe the sudden decreases in the minimum inter-UAV distance which is caused by the uncontrolled behavior of the goal-following UAV. Despite the uncontrolled UAV, there were no collisions and the swarm stayed together.



(a) Initial state. (b) State at $t = 40s$. (c) State at $t = 70s$.

Figure 5.23: States of the swarm during the second simulated experiment. The UAVs are marked with red crosses (not to scale) and the goal-oriented UAV is highlighted with a green circle.

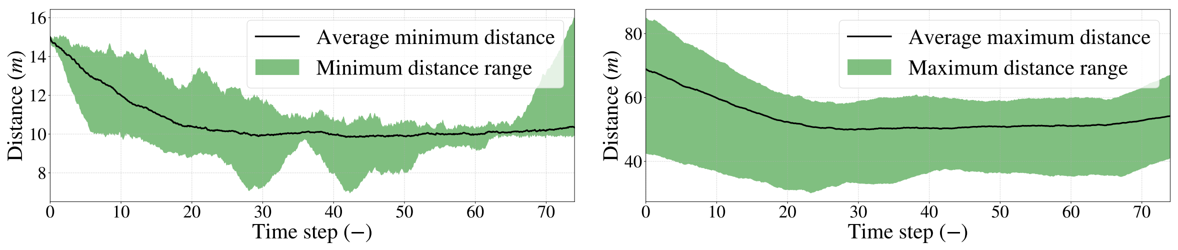


Figure 5.24: Overview of minimum and maximum distances over time for the second scenario with 25 UAVs out of which one was following a goal without using the proposed controller. The green interval is the smallest interval that encapsulates the distances with respect to all the UAVs in that time frame.

We also tested the ability of the swarm to navigate to a common goal. The starting setup consisted again of 25 UAVs in a grid-like formation with 15-meter inter-UAV distances. Each of the UAVs had a goal assigned that was

$$x^i_{goal} = x^i_{start} + 100, \qquad y^i_{goal} = y^i_{start} + 20, \tag{5.11}$$

where x^i_{goal}, y^i_{goal} are the coordinates of the goal of the i-th UAV (in meters), and x^i_{start}, y^i_{start} are the starting coordinates of the i-th UAV. The experiment ended when the centroid ("center of mass") of the swarm reached the proximity (a 10-meter radius) of the centroid of the goal positions of the swarm members. The swarm successfully reached its goal destination. This scenario reached virtually the same results in the distance measurements as the first hovering scenario (Figure 5.22) because the UAVs followed the goal in a very similar formation to the starting formation.

Therefore, we decided to conduct an experiment with a different scenario that includes the goal following using the proposed controller. The starting setup was the same as in the second scenario (Figure 5.23) with the difference that the UAV following the goal was also using the proposed controller. This scenario basically presents a UAV crossing the swarm to its assigned goal. The measured minimum and maximum distances were 9.1 meters and 84.9 meters respectively as shown in Figure 5.25. We can see that this scenario produced more stable minimum distances in comparison to the uncontrolled version of this scenario (Figure 5.24), therefore, making the flight safer.

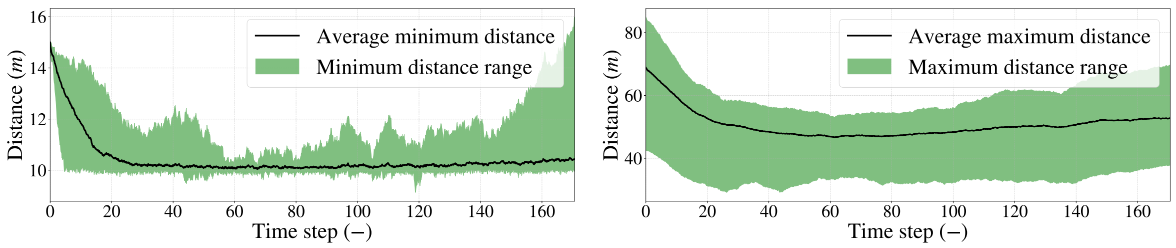


Figure 5.25: Overview of minimum and maximum distances over time for the last scenario with 25 UAVs out of which one was following a goal while also using the proposed controller. The green interval is the smallest interval that encapsulates the distances with respect to all the UAVs in that time frame.

We compared the approach to a similar alternative introduced in [51]. We include the results of one of the experiments presented in [51] in Figure 5.26 taken from [51]. We can observe that both the minimum and maximum distances are smaller than our results which is caused by the fact that the experiment in [51] includes only 9 UAVs, whereas, our controller is intended for and tested on larger swarms. Also, the difference in distances is strongly dependent on the parameters of the controller presented in 5.2 and, therefore, is subject to fine-tuning.

The primary goal of our designed swarming controller is to prove that the proposed relative localization approach is viable for swarm stabilization. In other words, the goal was to find any swarm controller (not necessarily particularly the best-performing) that successfully controls the swarm with the proposed relative localization method as the source of neighbor locations. After comparing Figures **??** with Figure 5.26, it may be concluded that both our approach and the approach introduced in [51] behave similarly in the way that both keep a safe inter-UAV distances while also keeping the swarm coherent (the exact distances are subject to tuning). This shows that the proposed method is suitable as the main source of relative localization within UAV swarms.

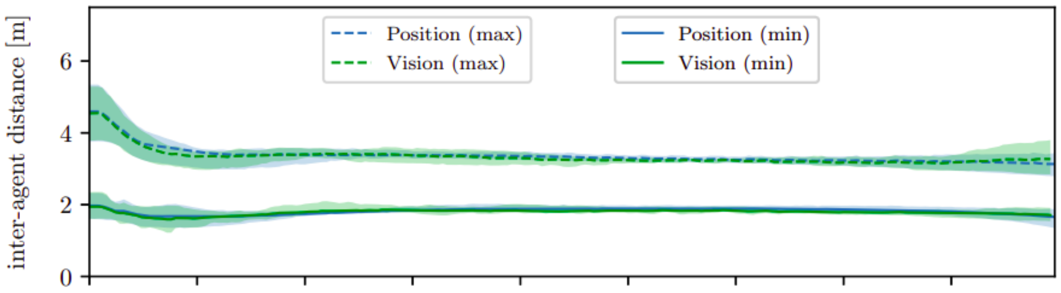


Figure 5.26: Overview of the minimum and maximum distances for one of the experiments presented in [51] using a vision-based and position-based controller.

Chapter 6

Conclusion and future work

A novel technique for large-scale visual relative localization of agents within a swarm of UAVs was introduced in this thesis. We utilize a convolutional neural network to regress the UAV density distribution in the discretized 3D space from an RGB image. This solution is capable of running in real-time onboard the UAV. The only sensor requirement is the onboard camera. The approach is marker-less and does not include any inter-UAV communication or use of a global navigation system and therefore is fully decentralized.

The proposed method was developed and tested on a custom synthetic dataset. It also successfully translated to real-world data. A comparison was carried out where our method outperformed an alternative approach based on a state-of-the-art off-the-shelf object detector (even when considering a hypothetical ideal object detector). Furthermore, we showed how to compensate the output of the relative localization when it is used with different camera parameters without retraining the model and tested it successfully. The choices in the neural network architecture, data generation, and training were substantiated by multiple ablation studies that show the advantages of the individual components.

Furthermore, we introduced modifications to the boids swarming algorithm to utilize the proposed relative localization method efficiently. Simulated swarming experiments were carried out using the proposed controller, to show that the proposed relative localization approach can be used as the sole source of relative neighbor positions for controlling the swarm. The experiments consisted of several scenarios that included large numbers of UAVs (hovering, member UAV malfunction on a collision course, goal following). We showed that the swarming was without collisions and the UAVs kept safe distances while also staying coherently together.

In the future, we would like to improve our datasets. Specifically, we would like to create a more balanced synthetic dataset according to the constraints mentioned in Section 3.3. Work is in progress on a challenging real-world dataset with up to 20 UAVs per image. Furthermore, we are just, as of writing this thesis, starting to test our approach on synthetic data that include up to 150 UAVs in a single image to show the scalability potential of our approach as well as the robustness to extreme overlaps. Furthermore, we want to try estimating the depth of the UAVs instead of the distance. Despite the two values being very similar in most cases, we think that it could potentially improve the performance of the method.

We believe this work will help with the transition from relatively small-scale swarm deployments in controlled environments to large, decentralized, and infrastructure-independent swarms.

Bibliography

[1] G. Bevacqua, J. Cacace, A. Finzi, and V. Lippiello, “Mixed-initiative planning and execution for multiple drones in search and rescue missions,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, 2015, pp. 315–323.

[2] J. Cacace, A. Finzi, and V. Lippiello, “Multimodal interaction with multiple co-located drones in search and rescue missions,” *arXiv preprint arXiv:1605.07316*, 2016.

[3] S. Mayer, L. Lischke, and P. W. Woźniak, “Drones for search and rescue,” in *1st International Workshop on Human-Drone Interaction*, 2019.

[4] H. Yao, R. Qin, and X. Chen, “Unmanned aerial vehicle for remote sensing applications—a review,” *Remote Sensing*, vol. 11, no. 12, p. 1443, 2019.

[5] I. Mademlis, V. Mygdalis, N. Nikolaidis, and I. Pitas, “Challenges in autonomous uav cinematography: An overview,” in *2018 IEEE international conference on multimedia and expo (ICME)*. IEEE, 2018, pp. 1–6.

[6] A. Barisic, F. Petric, and S. Bogdan, “Sim2air-synthetic aerial dataset for uav monitoring,” *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3757–3764, 2022.

[7] P. Petráček, V. Walter, T. Báča, and M. Saska, “Bio-inspired compact swarms of unmanned aerial vehicles without communication and external localization,” *Bioinspiration & Biomimetics*, vol. 16, no. 2, p. 026009, 2020.

[8] M. S. Innocente and P. Grasso, “Self-organising swarms of firefighting drones: Harnessing the power of collective intelligence in decentralised multi-robot systems,” *Journal of Computational Science*, vol. 34, pp. 80–101, 2019.

[9] E. Ausonio, P. Bagnerini, and M. Ghio, “Drone swarms in fire suppression activities: a conceptual framework,” *Drones*, vol. 5, no. 1, p. 17, 2021.

[10] B. Aydin, E. Selvi, J. Tao, and M. J. Starek, “Use of fire-extinguishing balls for a conceptual system of drone-assisted wildfire fighting,” *Drones*, vol. 3, no. 1, p. 17, 2019.

[11] M. Saska, V. Vonásek, J. Chudoba, J. Thomas, G. Loianno, and V. Kumar, “Swarm Distribution and Deployment for Cooperative Surveillance by Micro-Aerial Vehicles,” *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1, pp. 469–492, Dec. 2016. [Online]. Available: https://doi.org/10.1007/s10846-016-0338-z

[12] F. Flammini, C. Pragliola, and G. Smarra, “Railway infrastructure monitoring by drones,” in *2016 International Conference on Electrical Systems for Aircraft, Railway, Ship Propulsion and Road Vehicles & International Transportation Electrification Conference (ESARS-ITEC).* IEEE, 2016, pp. 1–6.

[13] G. M. Bolla, M. Casagrande, A. Comazzetto, R. Dal Moro, M. Destro, E. Fantin, G. Colombatti, A. Aboudan, and E. C. Lorenzini, “Aria: Air pollutants monitoring using uavs,” in *2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace).* IEEE, 2018, pp. 225–229.

[14] K. Spanaki, E. Karafili, U. Sivarajah, S. Despoudi, and Z. Irani, “Artificial intelligence and food security: swarm intelligence of agritech drones for smart agrifood operations,” *Production Planning & Control*, pp. 1–19, 2021.

[15] P. Tripicchio, M. Satler, G. Dabisias, E. Ruffaldi, and C. A. Avizzano, “Towards smart farming and sustainable agriculture with drones,” in *2015 international conference on intelligent environments.* IEEE, 2015, pp. 140–143.

[16] V. Puri, A. Nayyar, and L. Raja, “Agriculture drones: A modern breakthrough in precision agriculture,” *Journal of Statistics and Management Systems*, vol. 20, no. 4, pp. 507–518, 2017.

[17] V. Walter, M. Saska, and A. Franchi, “Fast mutual relative localization of uavs using ultraviolet led markers,” in *2018 International Conference on Unmanned Aircraft Systems (ICUAS).* IEEE, 2018, pp. 1217–1226.

[18] 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.

[19] M. Pavliv, F. Schiano, C. Reardon, D. Floreano, and G. Loianno, “Tracking and relative localization of drone swarms with a vision-based headset,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1455–1462, 2021.

[20] M. Vrba, D. Heřt, and M. Saska, “Onboard marker-less detection and localization of non-cooperating drones for their safe interception by an autonomous aerial system,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3402–3409, 2019.

[21] B. H. Kim, D. Khan, C. Bohak, W. Choi, H. J. Lee, and M. Y. Kim, “V-rbnn based small drone detection in augmented datasets for 3d ladar system,” *Sensors*, vol. 18, no. 11, p. 3825, 2018.

[22] K. Guo, X. Li, and L. Xie, “Ultra-wideband and odometry-based cooperative relative localization with application to multi-uav formation control,” *IEEE transactions on cybernetics*, vol. 50, no. 6, pp. 2590–2603, 2019.

[23] V. A. Sindagi and V. M. Patel, “A survey of recent advances in cnn-based single image crowd counting and density estimation,” *Pattern Recognition Letters*, vol. 107, pp. 3–16, 2018.

[24] G. Gao, J. Gao, Q. Liu, Q. Wang, and Y. Wang, “Cnn-based density estimation and crowd counting: A survey,” *arXiv preprint arXiv:2003.12783*, 2020.

[25] J. E. Herbert-Read, “Understanding how animal groups achieve coordinated movement,” *Journal of Experimental Biology*, vol. 219, no. 19, pp. 2971–2983, 2016.

[26] Y. Zhang, D. Zhou, S. Chen, S. Gao, and Y. Ma, “Single-image crowd counting via multi-column convolutional neural network,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 589–597.

[27] L. Zhang, M. Shi, and Q. Chen, “Crowd counting via scale-adaptive convolutional neural network,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV).* IEEE, 2018, pp. 1113–1121.

[28] F. Liu, C. Shen, and G. Lin, “Deep convolutional neural fields for depth estimation from a single image,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5162–5170.

[29] Y. Zhang, L. Ding, Y. Li, W. Lin, M. Zhao, X. Yu, and Y. Zhan, “A regional distance regression network for monocular object distance estimation,” *Journal of Visual Communication and Image Representation*, vol. 79, p. 103224, 2021.

[30] J. Zhu and Y. Fang, “Learning object-specific distance from a monocular image,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 3839–3848.

[31] S. Saleh, S. A. Khwandah, A. Heller, A. Mumtaz, and W. Hardt, “Traffic signs recognition and distance estimation using a monocular camera,” in *6th International Conference Actual Problems of System and Software Engineering.[online] Moscow: IEEE*, 2019, pp. 407–418.

[32] L. Bertoni, S. Kreiss, and A. Alahi, “Monoloco: Monocular 3d pedestrian localization and uncertainty estimation,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 6861–6871.

[33] Y. Cai, B. Li, Z. Jiao, H. Li, X. Zeng, and X. Wang, “Monocular 3d object detection with decoupled structured polygon estimation and height-guided depth estimation,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020, pp. 10 478–10 485.

[34] C. W. Reynolds, “Flocks, herds and schools: A distributed behavioral model,” in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 25–34.

[35] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, and T. Vicsek, “Optimized flocking of autonomous drones in confined environments,” *Science Robotics*, vol. 3, no. 20, p. eaat3536, 2018.

[36] M. Saska, “Mav-swarms: unmanned aerial vehicles stabilized along a given path using onboard relative localization,” in *2015 International Conference on Unmanned Aircraft Systems (ICUAS).* IEEE, 2015, pp. 894–903.

[37] S. J. Russell, *Artificial intelligence a modern approach Third edition.* Pearson Education, Inc., 2010.

[38] S. J. Russell and P. Norvig, “Artificial intelligence: a modern approach fourth edition,” 2021.

[39] J. Ma, W. Wang, and L. Wang, “Irregular convolutional neural networks,” in *2017 4th IAPR Asian Conference on Pattern Recognition (ACPR)*. IEEE, 2017, pp. 268–273.

[40] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[41] A. L. Maas, A. Y. Hannun, A. Y. Ng *et al.*, “Rectifier nonlinearities improve neural network acoustic models,” in *Proc. icml*, vol. 30, no. 1. Atlanta, Georgia, USA, 2013, p. 3.

[42] D. Scherer, A. Müller, and S. Behnke, “Evaluation of pooling operations in convolutional architectures for object recognition,” in *Artificial Neural Networks–ICANN 2010: 20th International Conference, Thessaloniki, Greece, September 15-18, 2010, Proceedings, Part III 20*. Springer, 2010, pp. 92–101.

[43] D. Yu, H. Wang, P. Chen, and Z. Wei, “Mixed pooling for convolutional neural networks,” in *Rough Sets and Knowledge Technology: 9th International Conference, RSKT 2014, Shanghai, China, October 24-26, 2014, Proceedings 9*. Springer, 2014, pp. 364–375.

[44] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 international conference on engineering and technology (ICET)*. Ieee, 2017, pp. 1–6.

[45] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

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

[47] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, “Tensorflow: a system for large-scale machine learning.” in *Osdi*, vol. 16, no. 2016. Savannah, GA, USA, 2016, pp. 265–283.

[48] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *arXiv preprint arXiv:1603.04467*, 2016.

[49] D. Hert, T. Baca, P. Petracek, V. Kratky, V. Spurny, M. Petrlik, M. Vrba, D. Zaitlik, P. Stoudek, V. Walter *et al.*, “Mrs modular uav hardware platforms for supporting research in real-world outdoor and indoor environments,” in *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2022, pp. 1264–1273.

[50] M. Tan, R. Pang, and Q. V. Le, “Efficientdet: Scalable and efficient object detection,” in *CVPR*, 2020, pp. 10 781–10 790.

[51] F. Schilling, J. Lecoeur, F. Schiano, and D. Floreano, “Learning vision-based cohesive flight in drone swarms,” *arXiv preprint arXiv:1809.00543*, 2018.

Appendices

List of abbreviations

In Table 1 are listed abbreviations used in this thesis.

Abbreviation	**Meaning**
UAV	Unmanned Aerial Vehicle
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
LReLU	Leaky Rectified Linear Unit
GD	Gradient Descent
RGB	Red Green Blue
FoV	Field of View
UVDAR	UltraViolet Direction and Ranging
UWB	Ultra-Wideband

Table 1: Lists of abbreviations