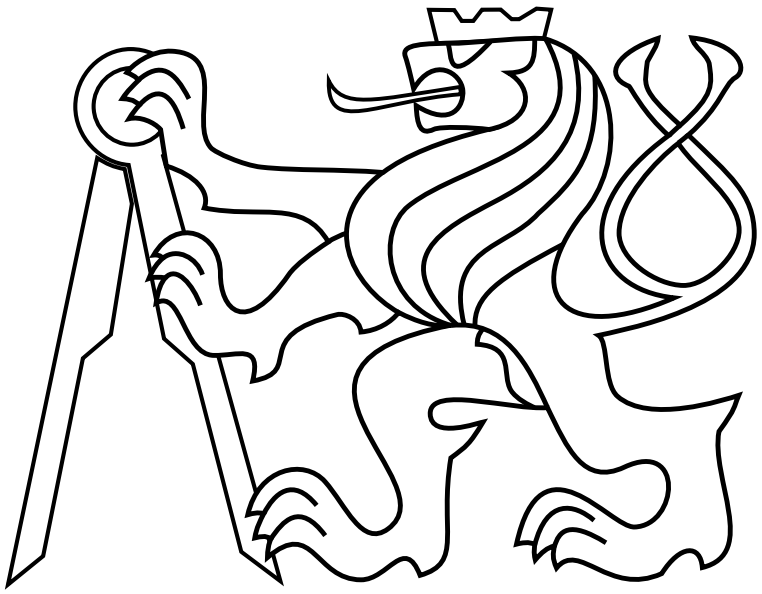
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

MASTER'S THESIS



Matouš Vrba

Relative localization of helicopters from an onboard camera image using neural networks

May 2018

Department of Cybernetics

Thesis supervisor: **Dr. Martin Saska**

Author statement for undergraduate thesis

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..... .....

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne..... .....

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Vrba Matouš** Personal ID number: **420115**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

Branch of study: **Robotics**

II. Master's thesis details

Master's thesis title in English:

Relative Localization of Helicopters from an Onboard Camera Image Using Neural Networks

Master's thesis title in Czech:

Relativní lokalizace helikoptér z obrazu palubní kamery pomocí neuronových sítí

Guidelines:

The aim of this thesis is to design a neural network to detect multiple Micro Aerial Vehicles (MAVs) in an image, and use it to develop a relative localization system for stabilization of an MAV group. The following main tasks will be solved:
1. Design, train and verify a neural network for detecting MAVs in a camera image, which is capable of running online on an MAV using onboard PC and camera.
2. Design and implement an algorithm for online relative localization of MAVs using the neural network and the onboard camera of the MAV.
3. Integrate the relative localization algorithm into the system used at MRS group for UAV control [3]
4. Verify the designed system in the Gazebo simulator and determine its precision and limitations.
5. Verify the designed system using data from real experiments using RTK-GPS as the ground truth.

Bibliography / sources:

[1] J. Redmon and A. Farhadi, YOLO9000: Better, Faster, Stronger, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017
[2] Krishneel Chaudhary, Moju Zhao, Fan Shi, Xiangyu Chen, Kei Okada, Masayuki Inaba, Robust Real-Time Visual Tracking Using Dual-Frame Deep Comparison Network Integrated with Correlation Filters Proceedings of The 2017 IEE/RSJ International Conference on Robotics and Systems, 2017
[3] T. Baca, P. Stepan and M. Saska. Autonomous Landing On A Moving Car With Unmanned Aerial Vehicle. In The European Conference on Mobile Robotics (ECMR), 2017.

Name and workplace of master's thesis supervisor:

Ing. Martin Saska, Dr. rer. nat., Multi-robot Systems, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **12.01.2018** Deadline for master's thesis submission: **25.05.2018**

Assignment valid until: **30.09.2019**

Ing. Martin Saska, Dr. rer. nat.	doc. Ing. Tomáš Svoboda, Ph.D.	prof. Ing. Pavel Ripka, CSc.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. _____ _____

Date of assignment receipt | Student's signature

Acknowledgements

I would like to thank my supervisor Martin Saska for his help and guidance throughout this thesis. I would also like to thank all guys from the lab for their help with specific problems I encountered during my work on this thesis and with the experiments, as well as for generally being nice to me. Finally, I would like to thank my parents and my family for their help, support and guidance in my life.

Abstract

The problem of camera-based relative localization and stabilization of multiple Micro Aerial Vehicles (MAVs) is tackled in this thesis. A relative localization system, which is able to work without any markers or other special equipment on the MAVs, is presented. The system utilizes a convolutional neural network for object detection in an image, which is trained to detect the MAVs. It was designed and implemented to run onboard our MAV platform in real-time in order to enable relative stabilization of several MAVs in a formation or swarm-like behavior. Performance and limitations of the system were evaluated in simulations. Furthermore, capabilities for relative stabilization were demonstrated in simulations as well as in real-world experiments. The proposed system proved to be robust and is ready for practical deployment.

Keywords: convolutional neural network, micro aerial vehicle, relative localization, computer vision, robotics, cybernetics

Abstrakt

Tato práce je zaměřena na problematiku relativní lokalizace a stabilizace bezpilotních helikoptér pomocí kamery. Je představen systém relativní lokalizace, který nevyžaduje žádné značky ani speciální vybavení na bezpilotní helikoptéře. Tento systém využívá konvoluční neuronovou síť pro detekci objektů v obraze, která je natrénována na detekování bezpilotních helikoptér. Byl navržen a implementován tak, aby fungoval na palubním počítači naší experimentální bezpilotní helikoptéry v reálném čase, za účelem použití tohoto systému pro relativní stabilizaci několika helikoptér nebo pro rojové chování. Výkony a omezení tohoto systému byly ověřeny v simulacích i v experimentech v reálném světě. Navrhovaný systém prokázal svoji robustnost a možnost nasazení v praxi.

Klíčová slova: konvoluční neuronová síť, bezpilotní helikoptéra, relativní lokalizace, počítačové vidění, robotika, kybernetika

Contents

List of Figures

1 Introduction

In this thesis, relative localization of Micro Aerial Vehicles (MAVs) from a camera image using a Convolutional Neural Network (CNN) for image processing is tackled. Robot localization is a necessary prerequisite for deployment of mobile robots and teams of mobile robots. A precise mutual localization among the robots is required in these applications, such as in swarm robotics [1, 2], for collision avoidance [3, 4, 5], or even in non-cooperative tasks like interception of intruding MAVs [6]. Absolute position of a robot in a global coordinate system is not always as important as relative position of multiple robots. Currently, robot localization is often solved by using a Global Navigation Satellite System (GNSS) [1, 3] or IR-based motion capture systems such as Vicon[1] or Optitrack[2] [7], but these systems have serious disadvantages. GNSS can provide relatively precise global localization, but it is not always available, its precision may be negatively affected by the environment, and it can be easily jammed [8, 9, 10]. Motion capture systems offer sub-millimeter precision at the cost of very expensive specialized hardware, which has to be installed and calibrated in the area of deployment as well as on the robots. These requirements limit their usability in real applications.

All these circumstances lead to a demand for a simple, mobile and ideally onboard relative localization systems. Computer vision-based systems have the advantage that theoretically they do not require extra hardware except for a camera, but most of the state of the art computer vision relative localization systems also rely on markers, installed on the robots to be localized [11, 12, 4]. This is usually not a big problem for ground robots, performing cooperative tasks, but the extra weight and volume of the markers can be a disadvantage especially for aerial robots. Carrying markers is also not an option under some circumstances, such as in situations with hostile robots, where the enemy robots cannot be expected to carry a compatible marker in order to be localized.

Thanks to the recent advancements in computer vision using Convolutional Neural Networks (CNNs) [13], new methods for fast object detection from camera image are available [14, 15, 16, 17, 18, 19, 20, 21]. These methods can potentially be used for relative localization of robots, which would have the advantage of the computer vision-based relative localization without the need for markers on the localized robots. This is because the neural network can be trained to directly detect selected robots. It may therefore enable using the relative localization also for non-cooperative tasks. On the other hand it brings new difficulties, such as the relatively high computational intensity of CNNs, which has to be overcome to enable real-time relative localization.

In this thesis, the relative localization of MAVs from camera image using CNNs is implemented. An onboard camera provides the image stream and a CNN running on an onboard PC of the MAV detects neighboring MAVs in the image. These detections are used

[1] `https://www.vicon.com/motion-capture/engineering`
[2] `http://optitrack.com/motion-capture-robotics/`

to estimate the relative position of the MAVs. A neural network structure was designed based on the YOLO [14, 15] neural network, which is a state of the art object detector with very good precision and speed of detection. The neural network was implemented to run on an MAV platform (see Figure 1), developed by the Multi-robot Systems Group from the Faculty of Electrical Engineering, Czech Technical University in Prague, for the MBZIRC competition[3]. Hardware of the MAV platform is described in detail in [22, 23, 24]. Control algorithms of the platform are described in [25, 26]. The used neural network is described in section 2. It was trained on a hand-labeled dataset, which was created specifically for this purpose, and then it was used as a part of the relative localization system, presented in section 3. The system was tested in simulations (see section 4.1), as well as in real-world experiments in two leader-follower scenarios (described in section 5).

Figure 1: Photo of the MAV platform used in the experiments.

1.1 Related work

Robot localization in the context of multi-robotic systems can be divided into two main categories: absolute localization and relative localization. Absolute localization systems relate the position of the robots to a static ground truth coordinate system (CS). This approach has its advantages, such as an inherent resistance to integrative errors because the ground truth CS is static, and in some cases very good precision. The main disadvantage

[3] http://www.mbzirc.com

of these systems is that they usually rely on a pre-installed infrastructure to provide the ground truth CS, which is the reason why they are not usable in all scenarios.

The Global Navigation Satellite Systems (GNSS, e.g. GPS, Galileo, etc.) are an example of an absolute localization system. GNSS systems have limited usability in obstructed areas (such as indoor areas, dense urban areas or woods), because they rely on the satellite signals. Another typical example are camera motion capture systems, which are constrained to laboratory conditions, because they require setup and calibration of the cameras. However these systems can still be used in laboratory experiments or in specific conditions, such as in open outdoor spaces, since they offer excellent precision, and are often used as ground truth measurements to evaluate robotic experiments [1, 3].

Another type of absolute robot localization is Simultaneous Localization and Mapping (SLAM), where a map is being constructed in which the robots are being localized. Thus it can be also used for localization of multiple robots if they can share information from the map. An example of such system is in [27], where a SLAM system for a group of cooperating Unmanned Ground Vehicles (UGVs), equipped with 3D LiDAR sensors is presented and evaluated in real-world experiments. Another example is in [28], where the authors propose a multi-camera visual SLAM system. The cameras are carried by MAVs and features from the camera images are used to create a map in which all MAVs are localized. The system performance is demonstrated in several experiments.

Knowing the absolute location is not always required in robotics, and in some scenarios only relative positions of the robots are important. In such cases, it can be more useful to use different localization techniques, which may offer increased precision and robustness or can work in areas without pre-installed infrastructure.

A typical visual relative localization system uses some kind of printed markers, which are detected in an image and their shape and size is used to calculate relative position of the markers. Numerous systems based on this method have been tested in experiments [29, 30, 31]. An example of such system is [11], which uses black and white circular markers, and offers scalable 3D relative localization with up to centimeter precision. Flood-fill segmentation and on-demand thresholding is utilized to detect the markers in the image, and their relative location is then calculated from size, shape and position of each marker in the image. A mathematical model of the method, allowing estimation of the localization precision, is also presented in the paper. This system has been used in multiple robotic experiments either to enable testing of different multi-robotic algorithms, or as a ground truth [31, 32, 33]. A similar system is presented in [12], where four spherical markers are used. The markers can be passive plastic spheres or active light-emitting spheres to improve performance under worse light conditions. Standard image processing methods are applied for detection of the markers in the image together with prediction of their position from previous data for constraining the region of interest in the image and speeding up the detection. The system achieves 6DOF tracking with centimeter precision, as is demonstrated in the paper in experiments with an MAV and ground robots.

A slightly different approach, which takes advantage of ultra-violet (UV) light emitters as active markers, is presented in [34]. A combination of filters and a UV-sensitive camera sensor is used to detect the UV markers and filter out other light sources. This approach is based on the assumption that UV light is less common in natural outdoor environments than other light wavelengths (infra-red and visible). Multiple markers placed on a single rigid body of a robot can be used to estimate its relative 3D pose, and blinking markers at different frequencies enable their unique identification. As it is demonstrated in experiments, the main source of error using this method is distance estimation, which is a common problem of monocular vision-based localization methods. This problem has been addressed in [35], where the authors use a combination of a vision sensor with a distance sensor. An infra-red (IR) camera measures bearing of the target by detecting an IR emitter, placed on the target. An ultrasound transmitter-receiver pair measures distance from the target using the Time of Arrival method. These measurements are fused by a Kalman Filter to obtain a relative 3D position of the target with precision in the order of centimeters, and maximal range of 9 m. Update rate of the system is 10 Hz for two robots, but it is smaller for higher number of robots, because Time Division Multiplier Access is used to multiplex the distance measurements.

All of the localization methods listed so far rely either on pre-installed infrastructure or on detecting some kind of markers on the robots. The method presented in this paper is able to localize robots without relying on either, using only information from a camera image. This means that it can be used in scenarios where it cannot be expected that the robot being localized will carry some kind of markers or cooperate in any other way. It also removes the need for extra hardware carried by the robots, which may be important for example in the case of MAVs, with limited carrying capacity. This is achieved by using a convolutional neural network (CNN) to directly detect the robots in the camera images instead of detecting some arbitrary markers. An example output of an object detection CNN is in Figure 2.

There are multiple methods of detecting general objects in a camera image. In [36] the authors introduced the Deformable Parts Models (DPM) method, which model an object in the image as a set of parts with deformable relative spatial constraints. These parts are detected using a dedicated classifier running in a window, which sequentially slides over the image. The detected parts are then evaluated based on their relative spatial constraints to determine class and position of the whole object. However, recently the DPM methods were outperformed in speed and precision by methods based on Convolutional Neural Networks. The R-CNN method [17] and its further improvements [18, 19] first generate class-agnostic proposals of regions in the image where objects might be detected. The proposed regions are then processed by a CNN to decide whether an object is present in each region and to classify it. The latest iteration, the Faster R-CNN, is a state of the art object detector regarding precision, but it is relatively slow when compared to other CNN-based methods, such as YOLO [14] or SSD [21].

In this thesis, the object detection has to run in real-time with limited processing resources onboard the MAVs. This is why a variation of the YOLO neural network structure [14, 15] is used, which offers the best suitable compromise between precision and speed, when compared to the other methods. The YOLO (first proposed in [14]) is an object detection CNN, which performs the region proposal and classification in one pass over the input image. It leads to a significant speed-up when compared to two-stage methods like Faster R-CNN. An update to the original YOLO method is described in [15] with some modifications to increase the detection precision and also proposing a method to improve classification by training on a classification dataset in addition to a detection dataset. During the course of work on this thesis, a third update to the YOLO CNN was released, which is described in the technical report [16]. The authors present some new techniques of detection and classification incorporated into the CNN structure as well as a new feature extractor, and report an increase in precision when detecting small objects in comparison to the older iterations of YOLO. This might be interesting for future work on the relative localization methods, presented in this thesis, because in real-world conditions the MAVs usually occupy a relatively small area of the whole image (see Figure 2).

Figure 2: Image from a real-world experiment with MAVs. Output of the object detection convolutional neural network is overlaid in red.

2 Neural network description

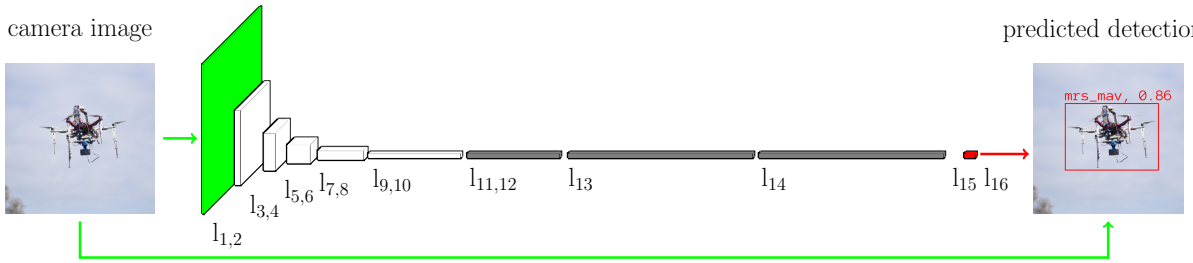


Figure 3: General principle of the neural network used in this work. Input of the neural network is a camera image and output is a set of bounding boxes of detected MAVs with their respective confidences.

In this section, the convolutional neural network, which is used for the MAV detection from the camera image (see Figure 3), is described. The neural network outputs bounding boxes of the MAVs, detected in the input image. A bounding box is the smallest rectangle completely surrounding the part of the image containing the detected object. The bounding boxes are used for the relative localization, as is described in section 3. Structure of the neural network is based on the Tiny YOLO network [37, 14, 15], which was adapted to classify one object class, and it is shown in Figure 6. Working principle of the network is described in section 2.1. The network needs to be trained on labelled data in order to produce good outputs, which is an essential part of utilizing a neural network. The training process and the training dataset used for training the neural network in this work are described in section 2.2. Two methods to increase the relative localization performance by modifying the input data of the neural network have been designed, and are described in section 2.3.

The neural network was implemented using the Darknet neural network framework [38] and integrated with the Robot Operating System [39]. The implementation uses OpenCL, which is an alternative to the commonly used Nvidia CUDA library. OpenCL was used instead of CUDA, because it is supported by the onboard graphics chip of the used MAV platform (Intel Iris Graphics 6100). However the onboard graphics chip of the MAVs is not well suited for running a convolutional neural network in real-time, resulting in a low number of frames per second (FPS), and a long delay between obtaining an image from the camera and getting estimated bounding boxes of objects in the image. This negatively affects the relative localization, as is discussed in section 5.2. There are different platforms, which offer better performance for running neural networks, such as the Nvidia Jetson TX2 module [40]. The Jetson TX2 runs on a Pascal graphical processing unit (GPU), and is relatively small and light-weight, enabling it to be used onboard MAVs.

The neural network, presented in this section, was tested to determine its speed performance on different platforms. It was tested on the MAV onboard graphics chip, on the

Jetson TX2 GPU, and on a dedicated GPU (GeForce GTX 1080), which was also used for training of the neural network. Note that the dedicated GPU is extremely large, heavy and has high power consumption when compared to the other two, and it is not suitable for computations onboard the MAVs. This GPU is included only as a reference. Results of speed the tests are in Table 1. For comparison, benchmark results of running the neural network four times in parallel as well as results using the YOLOv2 neural network structure are included. Running the neural network four times in parallel enables full 360° coverage of the area around the robot by using four cameras, rotated by 90° each, and feeding image of each camera to a separate neural network, so it is an interesting case to investigate. Using the YOLOv2 neural network might offer more robust detection, because it is a deeper and thus potentially more precise, although slower, structure. It is also a more commonly used neural network, so it is included for reference.

Setup	**Intel Iris Graphics 6100**	**Jetson TX2**	**GeForce GTX 1080**
Tiny YOLO	250 ms	77 ms	7 ms
4x Tiny YOLO	1000 ms	167 ms	20 ms
YOLOv2	1000 ms	167 ms	13 ms

Table 1: Comparison of speed performance of different neural network setups. The network presented in this thesis is listed as Tiny YOLO.

2.1 Detection principle of the neural network

The neural network used in this work is based on the Tiny YOLO neural network structure [37, 14, 15] and consists of two basic elements: feature extractor and detector. These two parts are described in detail further in this section. The Darknet neural networks framework [38], which implements the algorithms, described in this section, was used to implement and run the neural network.

Input of the neural network is an image from an onboard camera of the MAV and the output is a set of detections (bounding boxes with their respective confidences). The input image is represented as a tensor of floating point numbers in the interval $[0; 1]$ with dimensions equal to dimensions of the input layer of the neural network, which is $416 \times 416 \times 3$. If the image has a different resolution, it has to be resized. Each triplet in the input data represents r, g and b (red, green and blue) values of one pixel in the image. The input image is processed by the first 14 (out of the total 16, see Figure 6) layers of the neural network to extract smaller resolution maps of high level features from the image. These layers of the neural network are alternating convolutional layers with max-pooling layers, and they form the feature extractor part of the network. The output of the 14-th layer is an array of 1024 feature maps with 13×13 resolution, which is evaluated by the last two layers to produce the predicted bounding boxes and their corresponding confidences. A detailed description of the functioning of the layers in the neural network follows.

Figure 4: Example of learned features to be matched by the first layer of the neural network.

2.1.1 Convolutional layer

Input of a convolutional layer is an $m_w \times m_h \times m_d$ tensor of real numbers, which is an output from the previous layer (or an RGB image in case of the first layer). This tensor can be interpreted as an array of m_d feature maps with dimensions $m_w \times m_h$. Feature map is a two dimensional matrix of real numbers, representing matches with the learned features. The unit, which finds the matches between feature maps and features is called a filter. It performs linear convolution of an area in the input feature map and the learned feature, and applies an activation function $\phi()$ to the result to obtain one value of the output map (see equation 14). The area is then shifted in the input map by k_s (stride) and the process is repeated to obtain the next value of the output. The features, matched by these filters, are learned during training by optimizing a set of parameters of the convolution to get desirable outputs, using a process, which is described in section 2.2. An example of such features is in Figure 4. The number of filters, size of the filter input area (kernel size), the stride, and the activation function are hyperparameters of the layers, and are listed in Figure 6 for the network used in this thesis. The Leaky ReLU function

$$\phi_{leaky}(x) = \begin{cases} 0.1x, & \text{if } x < 0, \\ x, & \text{if } x >= 0, \end{cases} \tag{1}$$

was used as an activation function in the neural network, which is a variation of a more classical ReLU function

$$\phi_{ReLU}(x) = \begin{cases} 0, & \text{if } x < 0, \\ x, & \text{if } x >= 0, \end{cases} \tag{2}$$

but it is more resistant to the "dying out" effect during the training process [41].

2.1.2 Max-pooling layer

Max-pooling layers are used to downsample the output feature maps of the convolutional layers. One max-pooling operation is performed by taking the maximum value from an area of the input map and outputting it as one value in the output map. The area is then shifted by the stride and the operation is repeated to obtain the second value etc. Max-pooling reduces the volume of information to be processed by the following layers, while preserving important information about matching features, and also reduces sensitivity of the network to small position perturbations of the features. Hyperparameters of the max-pooling layers are the kernel size and stride, and they are listed in Figure 6.

2.1.3 Detection layers

The detection part of the neural network consists of the last two layers (in our case the 15-th and 16-th). The 15-th layer evaluates the output of the last feature extraction layer to produce a $13 \times 13 \times 30$ tensor, representing a map of 13×13 cells. Each cell corresponds to one part of the input image and contains 5 predicted detections of the objects in that part of the image with 6 coordinates (see Figure 5). The coordinates of a detection are $t_x, t_y, t_w, t_h, t_o, t_c$, which are prediction parameters of bounding box x-offset (t_x), y-offset (t_y), width (t_w), height (t_h), confidence of the predicted bounding box (t_o) and class probability (t_c).

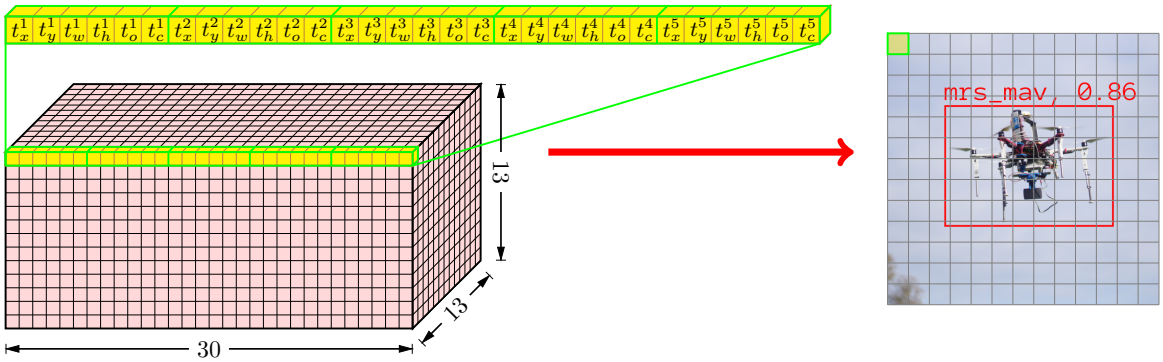


Figure 5: Detail of the feature map between layers 15 and 16 (see Figure 3). One row of the feature map contains prediction parameters for one cell of the 13×13 grid in the input image.

A predicted bounding box has center coordinates $\left[\hat{b}_x, \quad \hat{b}_y\right]^{\mathrm{T}}$, width $\hat{b}_w$, and height $\hat{b}_h$, which are all relative to dimensions of the input image (e.g. $\hat{b}_w = 1$ means a bounding box with a width equal to width of the image). $\left[q_x, \quad q_y\right]^{\mathrm{T}}$ are the top-left relative coordinates of the cell, containing the bounding box prediction. Then the prediction parameters of a detection in the cell correspond to

$$\hat{b}_x = \sigma\left(t_x\right) + q_x, \tag{3}$$

$$\hat{b}_y = \sigma\left(t_y\right) + q_y, \tag{4}$$

$$\hat{b}_w = p_w e^{t_w}, \tag{5}$$

$$\hat{b}_h = p_h e^{t_h}, \tag{6}$$

$$\hat{b}_c = \sigma\left(t_o\right), \tag{7}$$

where $\hat{b}_c$ is confidence of the prediction, and parameters p_w and p_h are prior dimensions of the bounding box, which are learned during the training process of the neural network. $\sigma\left(\right)$ is the Logistic function:

$$\sigma\left(x\right) = \frac{1}{1 + e^{-x}}. \tag{8}$$

In case that the neural network is designed to detect multiple classes, the parameter t_c would be a vector of numbers, representing confidence for the respective classes, which would then be recalculated to a probability distribution using the Softmax function (refer to [42] for more information about the Softmax function). The specific neural network presented in this work detects only one class, and thus the parameter t_c is redundant, as the probability distribution is trivial ($p(\text{class}|\text{object}) = 1$).

The last (16-th) layer calculates the predicted bounding boxes from the output tensor of the 15-th layer. Because there are $13 \times 13 = 169$ cells, each predicting 5 bounding boxes, there is a total of 845 predictions, and most of them have very low confidence. To reduce the number of predictions, the last layer filters the predictions based on their confidence and a threshold p_{thresh}. Confidence of a bounding box b can be interpreted as a predicted probability that the bounding box is correct. If it is less than the threshold, the bounding box is not considered further. Again, this is a simplified case because only one class is considered in our case. In case of multiple classes, the bounding box confidence would be multiplied by the its highest class probability, which would also determine the class, assigned to the bounding box. Finally, the remaining bounding boxes are filtered again, using the non-max suppression method, which is described in Algorithm 1, to remove overlapping bounding boxes caused by detecting the same object multiple times (the non-max suppression algorithm was introduced in [43]).

Algorithm 1 The non-max suppression algoritm

Input:
  $\mathrm{B_{in}}$ ▷ *set of input bounding boxes and probabilities of detected MAVs*
  o_{thresh} ▷ *overlap threshold of the bounding boxes*
Output:
  $\mathrm{B_{out}}$ ▷ *set of filtered bounding boxes and probabilities of detected MAVs*
$\mathrm{B_{out}} \leftarrow \emptyset$
while $\mathrm{B_{in}} <> \emptyset$ **do**
  ▷ *find the highest scoring detected bounding box and add it to the output set*
  $\mathrm{b_{best}} \leftarrow \text{find\_highest\_confidence}\,(\mathrm{B_{in}})$
  $\mathrm{B_{out}} \leftarrow \mathrm{B_{out}} \cup \mathrm{b_{best}}$
  ▷ *find all overlapping bounding boxes and remove them from* $\mathrm{B_{in}}$
  for all $\mathrm{b} \in \mathrm{B_{in}}$ **do**
    if $\text{overlap}\,(\mathrm{b_{best}}, \mathrm{b}) > o_{thresh}$ **then**
      $\mathrm{B_{in}} \leftarrow \mathrm{B_{in}} \backslash \mathrm{b}$
    end if
  end for
  ▷ *finally, remove* $\mathrm{b_{best}}$ *from* $\mathrm{B_{in}}$
  $\mathrm{B_{in}} \leftarrow \mathrm{B_{in}} \backslash \mathrm{b_{best}}$
end while

Figure 6: Detailed structure of the neural network used in this work (based on the Tiny YOLO [37, 14, 15]). Sizes of the feature maps are in format width × height × depth (number of output filters of the previous layer). The arrows represent layers of the neural network which process the previous feature map and output a new one.

2.2 Training

A vector of parameters of the neural network $\mathbf{w}$ has to be specified in order for the neural network to work. These parameters are optimized so that the neural network gives useful outputs. In context of neural networks, the optimization process is called training.

The training process utilizes the Stochastic Gradient Descent (SGD) method, which is an iterative stochastic approximation of the Gradient Descent optimization method for large datasets, and it is described in section 2.2.1. The SGD minimizes a loss function. The loss function used in this work when training the neural network is specified in section 2.2.2. SGD requires the gradient of the loss function, which is calculated using backward propagation of error (also called backpropagation), which is described in section 2.2.3.

The first 13 layers of the neural network were initialized with pretrained weights from the Pascal VOC 2007 dataset [44], which were downloaded from `https://pjreddie.com/media/files/yolov2-tiny-voc.weights`, so that the network did not have to learn extraction of basic common features from scratch. This is a common technique to speed up the training as only several last layers have to be significantly retrained [45, 14]. Since SGD is a supervised learning method, it requires a labeled dataset. Such a dataset was created specifically for this purpose, as it is described in section 2.2.4. During training, weights were saved for each 1000 training iterations (batches) for later validation. To pick the best weights for use in the experiments, a validation method, described in section 2.2.5 was used. The Darknet neural networks framework [38], which implements the algorithms, described in this section, was used for training and validation. The parameters used during training of the neural network, are listed in Table 2. They are explained in the following sections.

2.2.1 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) works similarly as the standard Gradient Descent algorithm, but uses mini-batches to approximate the gradient instead of using the whole dataset for one weight update. This approach speeds up training on large datasets, which are necessary when training deep convolutional neural networks. SGD iteratively finds an approximate minimum of a loss function $J(\mathbf{y}, \hat{\mathbf{y}})$ by optimizing parameters $\mathbf{w}$ of a function $\hat{\mathbf{y}} = f(\mathbf{x}, \mathbf{w})$ over the training dataset. The dataset consists of a set of pairs $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x}$ is an input to the function f, and $\mathbf{y}$ is the corresponding desired output (ground truth). The dataset is divided into n_b batches of n_p input/output pairs each. In this specific use-case, the inputs $\mathbf{x}$ are images, outputs $\mathbf{y}$ are manually labeled bounding boxes of the MAVs in the image, the function f is the neural network, $\mathbf{w}$ are weights of the convolutional filters in the network, and $\hat{\mathbf{y}}$ are the bounding boxes, estimated by the neural network.

The mini-batches are sequentially processed. Each image in a batch is propagated

Parameter	Explanation	Value
n_b	number of mini-batches	40
n_p	size of mini-batch	64
$n_{training}$	number of training images	2560
$n_{testing}$	number of testing images	152
μ	learning momentum	0.9
s_i	learning rate steps	$0, 10^2, 10^3, 10^4$
η_i	learning rates	$10^{-3}, 5 \cdot 10^{-4}, 10^{-4}, 5 \cdot 10^{-5}$
λ_{obj}	weight parameter of the loss function	5
λ_{noobj}	weight parameter of the loss function	1
IoU_{thresh}	IoU threshold for evaluation of recall	0.5
n_{it}	training iterations of the used weights	$5 \cdot 10^4$
r_h	image hue randomization	0.1
r_s	image saturation randomization	1.5
r_v	image lightness randomization	1.5
r_c	image crop randomization	0.2

Table 2: Parameters used for training of the neural network.

through the neural network (as it is described in section 2.1). Gradients of the loss function (which is defined in section 2.2.2) on the images from the batch are calculated using backward error propagation (see section 2.2.3). Gradient of the loss function on the whole batch R_i is defined as a sum of the gradients on the individual images:

$$\frac{\partial J_{batch}\left(\mathrm{R}_i\right)}{\partial \mathbf{w}_i} = \sum_{(\mathbf{x},\mathbf{y})\in \mathrm{R}_i} \frac{\partial J\left(\mathbf{y}, f\left(\mathbf{x}, \mathbf{w}_i\right)\right)}{\partial \mathbf{w_i}}. \tag{9}$$

Weight update ${}_{\Delta}\mathbf{w}_i$ is calculated from $\frac{\partial J_{batch}(\mathrm{R}_i)}{\partial \mathbf{w}_i}$ as

$${}_{\Delta}\mathbf{w}_i = \mu {}_{\Delta}\mathbf{w}_{i-1} - \eta_i \frac{\partial J_{batch}\left(\mathrm{R}_i\right)}{\partial \mathbf{w}_i}, \tag{10}$$

where η_i is learning rate at the i-th iteration, μ is momentum and ${}_{\Delta}\mathbf{w}_{i-1}$ is weight update in the previous iteration (or zero if it is the first iteration). The learning rate η_i is a parameter of the SGD algorithm, which sets the magnitude of weight update steps in each iteration, determining the trade-off between speed and precision of training. The momentum μ serves to reduce oscillations of the gradient by introducing what can be viewed as an analogy to a physical momentum of $\mathbf{w}_i$ [46]. Finally, the weight vector is updated as

$$\mathbf{w}_{i+1} := \mathbf{w}_i +_{\Delta} \mathbf{w}_i. \tag{11}$$

When all batches have been used, the dataset is randomly shuffled, and a new set of mini-batches is chosen. Training is finished when a preset number of batches has been processed.

A common way to improve the trade-off between speed and precision is to set the learning rate high at the beginning and gradually decrease it during training [46, 13, 14, 15]. This helps speed up the initial convergence of the algorithm, but slow down as the SGD is approaching the minimum to increase precision. One variant of this approach is to change the learning rate in several steps, when a certain number of batches has been processed, and this method was used when training the neural network in this thesis. Parameters of the SGD algorithm described in this section, which were used during the training, are listed in Table 2. Refer to [47, 48, 49] for more information about the SGD algorithm.

2.2.2 Loss function

Output of the loss function is a quantification of the difference between the output of the neural network and the ground truth (desired output). Each input image of the neural network is divided into a grid of $S \times S$ cells, and in each of these cells the neural network predicts n_B bounding boxes (for more details, see section 2.1). One predicted bounding box $\hat{\mathrm{b}}$ per cell is picked so that it has the highest intersection over union ratio (IoU) with the corresponding ground truth bounding box b. IoU of bounding boxes $\hat{\mathrm{b}}$ and b is defined as

$$\mathrm{IoU}\left(\mathrm{b}, \hat{\mathrm{b}}\right) = \frac{|\mathrm{b} \cap \hat{\mathrm{b}}|}{|\mathrm{b} \cup \hat{\mathrm{b}}|}, \tag{12}$$

where $|\mathrm{b} \cap \hat{\mathrm{b}}|$ is size of intersection area of b and $\hat{\mathrm{b}}$, and $|\mathrm{b} \cup \hat{\mathrm{b}}|$ is size of their union area.

B is a set of the ground truth bounding boxes and $\hat{\mathrm{B}}$ is a set of the estimated bounding boxes. The loss function is then defined as

$$\begin{aligned} J\left(\mathrm{B}, \hat{\mathrm{B}}\right) = {} & \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{n_B} \mathbb{1}_{ij}^{obj} \left[\left(b_{x,i} - \hat{b}_{x,j}\right)^2 + \left(b_{y,i} - \hat{b}_{y,j}\right)^2 \right] \\ & + \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{n_B} \mathbb{1}_{ij}^{obj} \left[\left(\sqrt{b_{w,i}} - \sqrt{\hat{b}_{w,j}}\right)^2 + \left(\sqrt{b_{h,i}} - \sqrt{\hat{b}_{h,j}}\right)^2 \right] \\ & + \lambda_{obj} \sum_{i=0}^{S^2} \sum_{j=0}^{n_B} \mathbb{1}_{ij}^{obj} \left(\mathrm{IoU}\left(\mathrm{b}_i, \hat{\mathrm{b}}_j\right) - \hat{b}_{c,j}\right)^2 \\ & + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{n_B} \left(1 - \mathbb{1}_{ij}^{obj}\right) \left(\mathrm{IoU}\left(\mathrm{b}_i, \hat{\mathrm{b}}_j\right) - \hat{b}_{c,j}\right)^2, \end{aligned} \tag{13}$$

where $b_{x,i}, b_{y,i}, b_{w,i}, b_{h,i}$ are center coordinates (x and y) and dimensions (width and height) of the i-th ground truth bounding box, $\hat{b}_{x,j}, \hat{b}_{y,j}, \hat{b}_{w,j}, \hat{b}_{h,j}$ are the same parameters of the j-th estimated bounding box, and $\hat{b}_{c,j}$ is the confidence of the estimated bounding box.

The variable $\mathbb{1}_{ij}^{obj}$ is equal to one if an object appears in the i-th cell and if the j-th predicted bounding box in the cell is "responsible" for detecting the object, otherwise it is zero. One predicted bounding box is responsible for each ground truth. The responsible bounding box is from the cell, in which lies the center of the ground truth bounding box, and it is the one with the highest IoU with the ground truth. Parameters λ_{obj} and λ_{noobj} ensure that cells, which contain no ground truth object, are penalized less than cells that do. This ensures stability of the training by preventing cells with no objects to "overpower" gradient of cells containing an object [14].

This formulation of the loss function is simplified, compared to the original [14], since it predicts only one class, so the part, penalizing wrong class prediction, was dropped from the function.

2.2.3 Backward propagation of error

Backward propagation of error (or backpropagation) is a method of calculating the gradient $\frac{\partial J}{\partial \mathbf{w}}$ of the loss function J for neural networks. The gradient is a vector of the same length as the parameter vector $\mathbf{w}$, and its elements can be interpreted as contributions of the corresponding parameters from $\mathbf{w}$ to the total value of the loss function. First, an input is propagated through the network to obtain an output, which is called a forward pass. Then the output of the network is compared to the desired (ground truth) output and the difference is expressed as a cost using the loss function (defined in previous section). The cost is sequentially propagated from the last layer back to the first one to calculate contributions of the parameters of the layers to the cost. This is the backward pass.

Three types of layers are used in the neural network in this thesis: convolutional, max-pooling and the detection layer. During backpropagation the detection layer calculates the error according to the loss function, defined in the previous section. It has no parameters to learn, so it does not contribute to $\frac{\partial J}{\partial \mathbf{w}}$. The max-pooling layers backpropagate the error by assigning it to the filters of the previous layer, from which the output was taken during the forward pass (which had the maximal activation, see section 2.1.2), and setting the error of the rest of its inputs to zero. It also does not have any parameters, which are learned (only hyperparameters), so it does not contribute to $\frac{\partial J}{\partial \mathbf{w}}$. However, the convolutional layers have weights w_{kj}, which are learned parameters, and thus these layers need to be evaluated using a more complex approach.

Output o_j of a j-th convolutional filter in the neural network is defined as

$$o_j = \phi_j\left(s_j\right) = \phi_j\left(\sum_{k=1}^{n_k} w_{kj} o_k\right), \tag{14}$$

where s_j is the sum of inputs of this filter, n_k is number of its inputs, o_k is its k-th input, ϕ_j is its activation function, and w_{kj} is the weight of its k-th input. The element of the

gradient vector $\frac{\partial J}{\partial \mathbf{w}}$, corresponding to weight w_{kj} is $\frac{\partial J}{\partial w_{kj}}$, which can be expanded to

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial o_j}\frac{\partial o_j}{\partial s_j}\frac{\partial s_j}{\partial w_{kj}}. \tag{15}$$

The last partial derivative can be calculated as

$$\frac{\partial o_j}{\partial w_{kj}} = o_k, \tag{16}$$

which is an output of a filter in the previous layer, and thus it is known during the training process. The partial derivative $\frac{\partial o_j}{\partial s_j}$ can be calculated if a derivation of the activation function ϕ_j is known. In the network used in this thesis, only two types of activation function were used: the Leaky ReLU and linear activation (see section 2.1). Their derivatives are

$$\frac{d\phi_{leaky}(x)}{dx} = \begin{cases} 0.1, & \text{if } x < 0, \\ 1, & \text{if } x >= 0, \end{cases} \tag{17}$$

$$\frac{d\phi_{linear}(x)}{dx} = 1, \tag{18}$$

so this part of the expanded $\frac{\partial J}{\partial w_{kj}}$ can be calculated by substituting s_j for x, since s_j is also known. The most complicated part is evaluating $\frac{\partial J}{\partial o_j}$. This partial derivative can be further expanded by taking its total derivatives with respect to the output of all filters, which take o_j as their input, to obtain

$$\frac{\partial J}{\partial o_j} = \sum_{l=1}^{n_l}\left(\frac{\partial J}{\partial s_l}\frac{\partial s_l}{\partial o_j}\right) = \sum_{l=1}^{n_l}\left(\frac{\partial J}{\partial o_l}\frac{\partial o_l}{\partial s_l}w_{jl}\right), \tag{19}$$

where n_l is the number of filters, which take o_j as an input, s_l is input of the l-th filter, which takes o_j as an input, o_l is its output, and w_{jl} is weight of o_j in the l-th filter (similarly as in (14)). In order to evaluate this expression, knowing $\frac{\partial J}{\partial o_l}$ is required, which can be expanded in the same manner as $\frac{\partial J}{\partial o_j}$, so it leads to a recursive problem. However, if o_l is in the output layer, then $\frac{\partial J}{\partial o_l}$, can be directly calculated. This is why the gradient has to be calculated backwards from the last layer to the first, so that the values of $\frac{\partial J}{\partial o_l}$ are known when evaluating $\frac{\partial J}{\partial o_j}$. Refer to [49] for detailed information about the backward error propagation method.

2.2.4 Training dataset

Because training using the SGD algorithm is a supervised learning process, a training dataset is required. In case of the neural network used in this work, the training dataset

includes images containing the MAVs to be detected, and the corresponding ground truth detections (bounding boxes). This dataset was obtained by capturing camera images from an MAV, which was flying together with two other MAVs. The MAVs followed trajectories, designed so that the onboard camera captured the two MAVs from different angles and distances and on different backgrounds. Images from the camera were then manually labeled by marking the ground truth bounding boxes to generate a set of 2560 training images. For validation purposes, photos from other experiments with different exposures, backgrounds, in varying weather conditions, and from different cameras, were hand-picked and manually labeled as well to obtain a set of 152 testing images. Examples of images from the two datasets are in Figure 7.

During training, the dataset is augmented by randomly cropping, flipping and distorting the images to achieve higher variability in the data and thus get more general learned parameters of the neural network, leading to more robust detection [50]. If im_h, im_s, im_l are the hue, saturation and lightness values of the image in the HSL color space, then these values are distorted using parameters r_h, r_s, r_v as

$$im_{h,r} = im_h + \mathcal{U}\left(-r_h, r_h\right), \tag{20}$$

$$im_{s,r} = \mathcal{U}\left(1/r_s, r_s\right) im_s, \tag{21}$$

$$im_{l,r} = \mathcal{U}\left(1/r_l, r_l\right) im_l, \tag{22}$$

where $im_{h,r}, im_{s,r}, im_{l,r}$ are values of the distorted image and $\mathcal{U}\left(min, max\right)$ is a uniform random distribution in the interval $[min; max]$. The image is cropped from top, left, right and left by a random amount, drawn from the random distribution $\mathcal{U}\left(0, r_c\right)$, where r_c is a parameter, relative to dimensions of the image. Then the image is flipped with probability 0.5. The parameters r_h, r_s, r_l and r_c used when training the network are listed in Table 2.

2.2.5 Validation of the trained network

The training algorithm does not have a strict stopping rule. It can theoretically run indefinitely, but with the increasing number of iterations of the SGD algorithm, the risk of overfitting increases. Overfitting occurs when the trained parameters give good results on the training dataset, but low score on different data, and is caused by the parameters being fitted to the training data too precisely and thus losing generality. To avoid this, the training process has to be stopped before this situation happens, which can be determined using a separate dataset. This separate dataset is called testing or validation dataset, and it is used to evaluate performance of the trained parameters on data, which are not part of the training dataset. During training, when score of the neural network using the current parameters stops growing or starts decreasing on this testing dataset, the training begins to overfit the training data.

During the training of the neural network, the weight vector was saved every 100 batches. The training algorithm was iterated for 10^5 batches, which was estimated to be

Figure 7: Example images from the training and testing datasets. The top two images are from the training dataset and the bottom two from the testing dataset.

enough for overfitting to occur. After 10^5 training batches, the training was stopped and the saved weights were evaluated based on the recall and IoU scores on the training and testing datasets (see Figure 8). Recall is defined as

$$recall = \frac{n_r}{n_{gt}}, \tag{23}$$

where n_r is the number of predicted bounding boxes having IoU with the corresponding ground truth bounding boxes higher than a certain threshold IoU_{thresh}, and n_{gt} is the total number of ground truth bounding boxes in the dataset. It can be interpreted as the number of correctly detected bounding boxes relative to the ground truth number of bounding boxes. Value of the parameter IoU_{thresh} is listed in Table 2.

The final weight vector $\mathbf{w}$, which was used for the neural network in the experiments, was chosen from the saved weights based on the graph in Figure 8. It can be seen in the graph that altough the average IoU score on the training data continues to rise with the number of iterations of the algorithm (number of training batches used), the score on testing data is almost constant at 53% after $4 \cdot 10^4$ iterations. Similarly, the recall score on training data continues to rise, but on testing data it is almost constant at 60% after $4 \cdot 10^4$ iterations. This is why the weight vector after $5 \cdot 10^4$ iterations was chosen to be the value of the final weights $\mathbf{w}$ as a compromise between score on the training and testing datasets and the risk of overfitting.

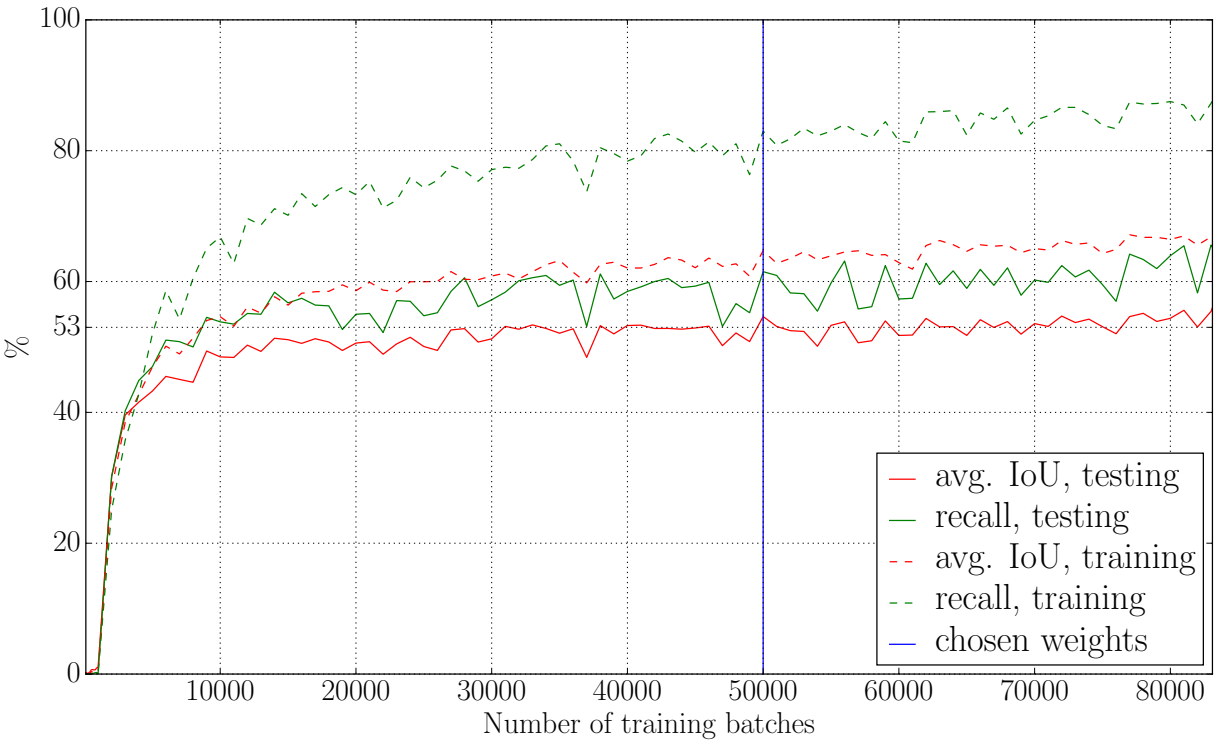


Figure 8: Graph of IoU and recall score of the neural network using weights, saved during training.

2.3 Input modifications

In order to increase precision of the relative localization, two methods of modifying the input data of the neural network have been used. The first method is aimed at better bounding box estimation by running the neural network a second time on the same input image, and is described in section 2.3.1. Target of the second method is to reduce localization error due to image reshaping and large camera distortion in the corners. It is described in section 2.3.2. The complete input modification method, combining the two, is presented in Algorithm 2.

2.3.1 Zoom-in detection

Because the relative localization algorithm relies on the bounding boxes of the detected MAVs (see section 3), its precision is limited by the ability of the neural network to precisely estimate the bounding boxes. Making the bounding box estimation of the neural network more precise is therefore desirable. The input layer of the neural network accepts a 416×416 RGB image, but the output of the onboard camera is 1280×720 pixels. This means that

the image from the camera has to be downsampled to less than 20% of pixels before it is passed to the neural network, so some of the information contained in the original image is lost (the image is resampled using linear interpolation [51]).

The zoom-in method remedies this by re-running the neural network on a zoomed area with the same dimensions as the input of the neural network around each MAV detection. Thus, the image of the MAV is presented to the neural network in the best resolution available and without distortions caused by the downsampling. The advantage of this approach is a better estimation of the bounding box, which was proved by simulation tests (see section 4.1). Another advantage is a potentially decreased sensitivity to false positives, since the false positive might not reappear in the second detection with the zoomed-in image.

A disadvantage of this approach is a slower and not constant update rate of the relative localization, since the neural network is run one more time for each detection, which is especially significant if there are more MAVs in the image or if there are a lot of false positives. However in some specific cases, such as the experiments presented in section 5.2, where there was only one MAV to be detected, this method provides significant improvements in distance estimation, at the cost of halved update rate.

2.3.2 Subsquare detection

Along with the distortions, caused by downsampling the image (as was discussed in the previous section), the image aspect ratio also has to be changed from $16 : 9$ to $1 : 1$. This causes additional distortions of the objects in the image and possibly loss of precision when detecting these objects. One possible way to address this issue would be to modify structure of the neural network to natively accept the correct aspect ratio, but since the neural network was also used on images from different sources with different resolutions and aspect ratios, the default configuration was kept.

In case of the 2D leader-follower scenario (as described in section 5.2), the leader MAV is expected to stay in the middle of the image with only small deviations, so the left and right edges of the image are not as important as the center. Cutting these edges off from the image, so that the resulting image is square, cancels the need to deform the image, and also potentially removes false positives, since the leader MAV is unlikely to wander off to the sides of the image under the conditions of the experiment, so a detection there is likely to be a false positive. Thus, using only the center square of the image offers a good compromise between increased detection and projection precision, and decreased field of view. This method was used in the 2D leader-follower scenario.

Algorithm 2 The detection algorithm with the input modifications

Input:
  $\mathbf{I}$ ▷ *the input image from the camera stream*
  w_{cam}, h_{cam} ▷ *dimensions of the camera image*
  w_{cnn}, h_{cnn} ▷ *dimensions of the neural network input*
  $\mathbf{w}, p_{thresh}$ ▷ *parameters of the NN - weights and confidence threshold*
  o_{thresh} ▷ *parameter of the NMS algorithm - overlap threshold*
  use_subsquare ▷ *whether to use subsquare of the image*
  use_zoom ▷ *whether to use the zoomed-in redetection method*
Output:
  $\mathrm{B}_{\mathrm{out}}$ ▷ *set of bounding boxes of the detected MAVs*
▷ *this algorithm presumes that* $w_{cam} \geq h_{cam} \geq w_{cnn} = h_{cnn}$
$\mathrm{B}_{\mathrm{out}} \leftarrow \emptyset$
$w_{used} \leftarrow w_{cam}$
$h_{used} \leftarrow h_{cam}$
if use_subsquare **then**
  $w_{used} \leftarrow h_{cam}$
  ▷ *calculate offset of the sub-image so that it is centered*
  $x_{offset} \leftarrow (w_{cam} - w_{used})/2$
  ▷ *get a sub-image with dimensions* $w_{used} \times h_{used}$*, and top-left corner* $\left[x_{offset}, 0\right]^{\mathrm{T}}$
  $\mathbf{I} \leftarrow \mathrm{get\_subrectangle}\left(\mathbf{I}, w_{used}, h_{used}, x_{offset}, 0\right)$
end if
▷ *reshape and downsample the image to correct input dimensions of the neural network*
$\mathbf{I}_{\mathrm{tmp}} \leftarrow \mathrm{resize}\left(\mathbf{I}, w_{cnn}, h_{cnn}\right)$
▷ *run the neural network to get detected bounding boxes*
$\mathrm{B}_{\mathrm{det}} \leftarrow \mathrm{detect}\left(\mathbf{I}_{\mathrm{tmp}}, \mathbf{w}, p_{thresh}\right)$
▷ *filter the bounding boxes using the non-max suppression algorithm*
$\mathrm{B}_{\mathrm{det}} \leftarrow \mathrm{nonmax\_suppression}\left(\mathrm{B}_{\mathrm{det}}, o_{thresh}\right)$
if use_zoom **then**
  for all $\mathrm{b} \in \mathrm{B}_{\mathrm{det}}$ **do**
    $x_{center}, y_{center} \leftarrow \mathrm{get\_center}\left(\mathrm{b}\right)$
    ▷ *find top-left corner of the zoomed area, bounded to the image dimensions*
    $x_{left} \leftarrow \min\left(\max\left(0, x_{center} - w_{cnn}\right), w_{used} - 1 - w_{cnn}\right)$
    $y_{top} \leftarrow \min\left(\max\left(0, y_{center} - h_{cnn}\right), h_{used} - 1 - h_{cnn}\right)$
    $\mathbf{I}_{\mathrm{tmp}} \leftarrow \mathrm{get\_subrectangle}\left(\mathbf{I}, w_{cnn}, h_{cnn}, x_{left}, y_{top}\right)$
    $\mathrm{B}_{\mathrm{tmp}} \leftarrow \mathrm{detect}\left(\mathbf{I}_{\mathrm{tmp}}, \mathbf{w}, p_{thresh}\right)$
    ▷ *keep the bounding box with highest confidence, if there were multiple redetected*
    $\mathrm{b} \leftarrow \mathrm{find\_highest\_confidence}\left(\mathrm{B}_{\mathrm{tmp}}\right)$
    $\mathrm{B}_{\mathrm{out}} \leftarrow \mathrm{B}_{\mathrm{out}} \cup \mathrm{b}$
  end for
else
  $\mathrm{B}_{\mathrm{out}} \leftarrow \mathrm{B}_{\mathrm{det}}$
end if

3 Relative localization

In this section the relative localization method is presented. The method relies on the output of the neural network, which is a set of predicted bounding boxes of the detected MAVs and their respective confidences (for more details see Section 2). Direction vector of each detected MAV is calculated from the output of the neural network using the method described in section 3.2. Distance of each MAV is estimated based on the output of the neural network using the method described in section 3.3, and filtered using a sliding window filter, which averages the last n_f values to filter out outliers. An estimate of relative positions of the detected MAVs is obtained by combining the calculated directions and distances. The complete algorithm is presented in Algorithm 3. Assumptions about the system setup are discussed in section 3.1.

3.1 System setup

In the experiments, one MAV with an onboard camera and an onboard computer, capable of running the neural network in real-time, is presumed. Calibration parameters of the camera need to be known for the relative localization. One or more MAVs, located in the field of view of the onboard camera, are being localized. The MAVs are presumed to be visually similar in the sense that the neural network can learn a good generalization of them from the available training datasets. Finally, the neural network itself, properly trained to detect the MAVs on a good dataset, is required for the presented algorithm.

3.2 Direction estimation

Estimation of a direction vector to a detected MAV from a predicted bounding box of the MAV is described in this section. The center point of the predicted bounding box is first rectified using the camera distortion coefficients, obtained from calibration (the camera calibration and image rectification was done using the OpenCV library [52], which uses an algorithm, based on [53]). The rectified pixel coordinates $\begin{bmatrix} u, & v \end{bmatrix}^{\mathrm{T}}$ are then transformed to the XY plane in the camera projection coordinate system using a pinhole camera model (see Fig. 9 and reference [54]) as

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (u - c_x)/f_x \\ (v - c_y)/f_y \\ 1.0 \end{bmatrix}, \tag{24}$$

where c_x, c_y, f_x, and f_y are parameters of the camera calibration. $\begin{bmatrix} c_x, & c_y \end{bmatrix}^{\mathrm{T}}$ represent coordinates of the optical center of the camera in pixels. The constants f_x, f_y are x and y focal

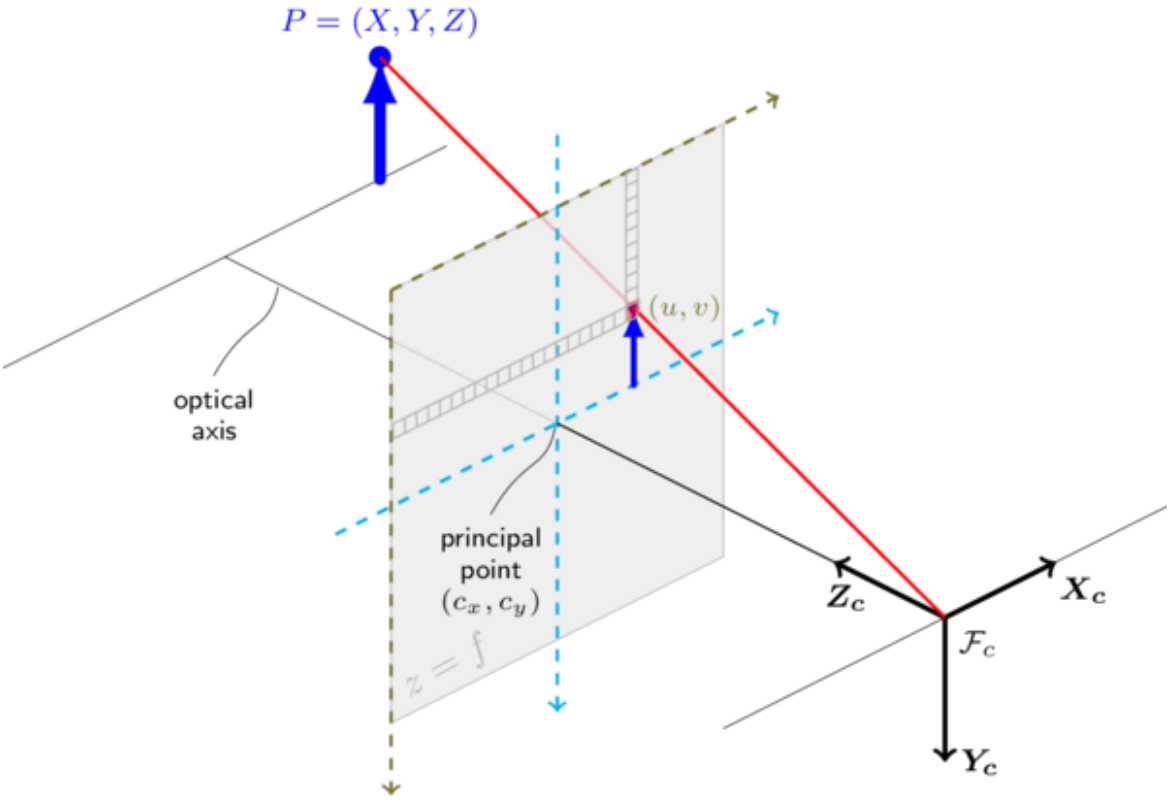


Figure 9: The pinhole camera model. The camera coordinate system is defined by its axis vectors $\boldsymbol{X_c}, \boldsymbol{Y_c}, \boldsymbol{Z_c}$, and origin $\mathcal{F}_c$. The camera projection coordinate system lies in a plane, defined by $z = f$ (where f is the camera focal length), and is translated by the vector $\begin{bmatrix} c_x, & c_y \end{bmatrix}^{\mathrm{T}}$ so that the origin of this coordinate system is in the top-left corner of the camera image. P is the point being projected onto the camera projection plane and $\begin{bmatrix} u, & v \end{bmatrix}^{\mathrm{T}}$ are coordinates of its projection (after rectification). Image source: [51].

lengths in pixels. They correspond to the camera physical focal length f (in millimeters) by the relationship

$$f_x = s_x f, \tag{25}$$
$$f_y = s_y f, \tag{26}$$

where s_x and s_y are pixel width and pixel height in pixels per millimeters.

The resulting point $\begin{bmatrix} x, & y, & z \end{bmatrix}^{\mathrm{T}}$ is then normalized to obtain a directional vector $\mathbf{a}$ of the 3D line on which the original pixel $\begin{bmatrix} u, & v \end{bmatrix}^{\mathrm{T}}$ lies. The units of the elements of the vector $\mathbf{a}$ are meters. Origin of the line is the point $\begin{bmatrix} 0, & 0, & 0 \end{bmatrix}^{\mathrm{T}}$, which is the origin of the camera coordinate system.

3.3 Distance estimation

Distance estimation of a detected MAV from the camera is based on the assumption that the 3D bounding geometrical primitive of an MAV is a sphere. This assumption is important to simplify the problem in situations when the detected MAV is tilted or viewed from below or above, although it introduces some error in these situations. A more precise assumption would be to assume a cylindrical 3D bounding primitive. However this would extremely complicate the problem as it would require a much more sophisticated model which would take into account relative position of the detecting and detected MAVs as well as tilt of the detected MAV, which would have to be estimated. Because the model described in this section, which is based on the simpler assumption of a spherical bounding primitive, proved to be precise and reliable enough in simulations and in real-world experiments, the more complex model was not pursued.

The point $\begin{bmatrix} u, & v \end{bmatrix}^{\mathrm{T}}$ is the rectified center point of the bounding box of the detected MAV, as estimated by the neural network. w_{bb} is width of the bounding box. Vectors $\mathbf{a}_l$ and $\mathbf{a}_r$ are directional vectors of 3D lines, projected through points $\begin{bmatrix} u - w_{bb}/2, & v \end{bmatrix}^{\mathrm{T}}$ and $\begin{bmatrix} u + w_{bb}/2, & v \end{bmatrix}^{\mathrm{T}}$, respectively (see previous section 3.2 for description of the projection method and Fig. 10 for an overview of the situation). These lines intersect the point $\mathcal{F}_c$ (origin of the camera coordinate system), and they are presumed to be tangent to the MAV bounding sphere $\mathrm{S}_{\mathrm{MAV}}$.

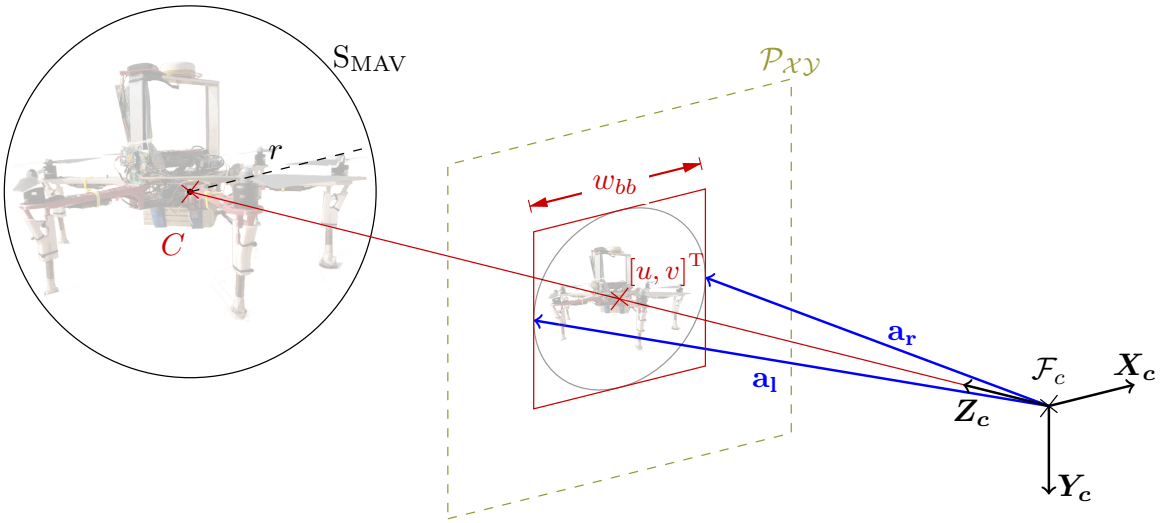


Figure 10: Perspective view of the geometrical situation of the MAV projection. $\mathrm{S}_{\mathrm{MAV}}$ is the bounding sphere of the MAV with center C and radius r. $\mathcal{P}_{XY}$ is the camera projection plane, $\mathcal{F}_c$ is the origin of the camera coordinate system and $\boldsymbol{X_c}, \boldsymbol{Y_c}, \boldsymbol{Z_c}$ are its axis vectors. $\begin{bmatrix} u, & v \end{bmatrix}^{\mathrm{T}}$ is the MAV center, projected to $\mathcal{P}_{XY}$, and w_{bb} is its width. Vectors $\mathbf{a_r}$ and $\mathbf{a_l}$ are direction vectors of lines, intersecting centers of the sides of the detected MAV bounding box.

The distance l from the origin of the camera coordinate system $\mathcal{F}_c$ to the center of the MAV C is estimated from angle α which is half of the angle between the vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ (see Fig. 11). The angle is calculated as

$$\alpha = \frac{\operatorname{acos}\left(\frac{\mathbf{a}_1 \cdot \mathbf{a}_2}{||\mathbf{a}_1|| ||\mathbf{a}_2||}\right)}{2}. \tag{27}$$

The formula to calculate the distance is derived using variables, displayed in Fig. 11. The relation

$$\tan(\alpha) = \frac{d}{l_1} \tag{28}$$

can be derived from the right-angled triangle $\triangle \mathcal{F}_c A P_r$. Similarly, the relations

$$\tan(\beta) = \frac{d}{l_2} \tag{29}$$

and

$$\sin(\beta) = \frac{d}{r} \tag{30}$$

can be derived from the right-angled triangle $\triangle C A P_r$. Using the right-angled triangle $\triangle \mathcal{F}_c C P_r$, the relation between the angles α and β is determined to be

$$\beta = \frac{\pi}{2} - \alpha. \tag{31}$$

Using the identity $l = l_1 + l_2$ and the equations (28), (29), (30), (31) the final relation between l and α can be found:

$$l = r \sin\left(\frac{\pi}{2} - \alpha\right) \left[\tan(\alpha) + \operatorname{cotan}(\alpha)\right]. \tag{32}$$

During the experiments, described in section 5, a radius $r = 0.29\,\mathrm{m}$ was used to describe the bounding sphere of the MAVs. However, due to an overlook, a simplified version of the equation (32) was used, which doesn't take into account the difference between the sphere diameter ($2r$) and length of the line segment $\overline{P_r P_l}$, which is actually being projected ($||\overline{P_r P_l}|| = 2d$). The simplified equation is

$$l = \frac{r}{\tan(\alpha)}. \tag{33}$$

The difference between $2r$ and $2d$ is only significant in very short distances. The distance error due to this mistake is less than 2% for distances larger than 3 m (see Fig. 12), which was the minimal distance between the MAVs before the collision avoidance system was activated during the experiments, so the error did not influence the results drastically.

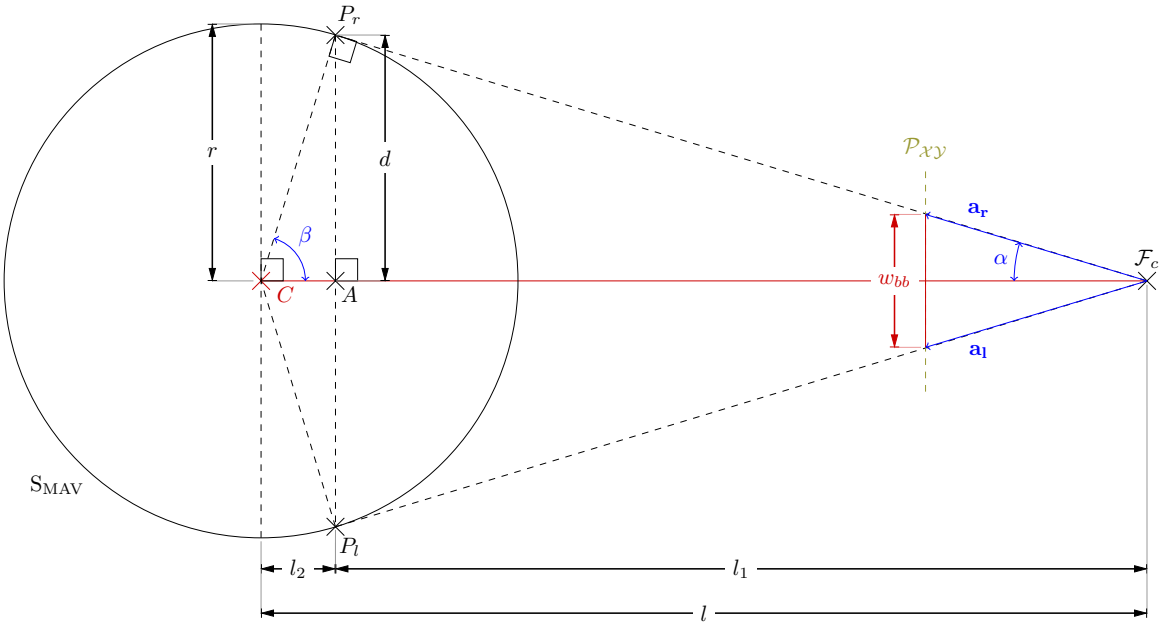


Figure 11: Top view of the geometrical situation of the MAV projection. S_{MAV} is the bounding sphere of the MAV with center C and radius r. $\mathcal{P}_{XY}$ is the camera projection plane and $\mathcal{F}_c$ is the origin of the camera coordinate system. Vectors $\mathbf{a_r}$ and $\mathbf{a_l}$ are direction vectors of the lines, intersecting the centers of the sides of the detected MAV bounding box (see Fig. 10).

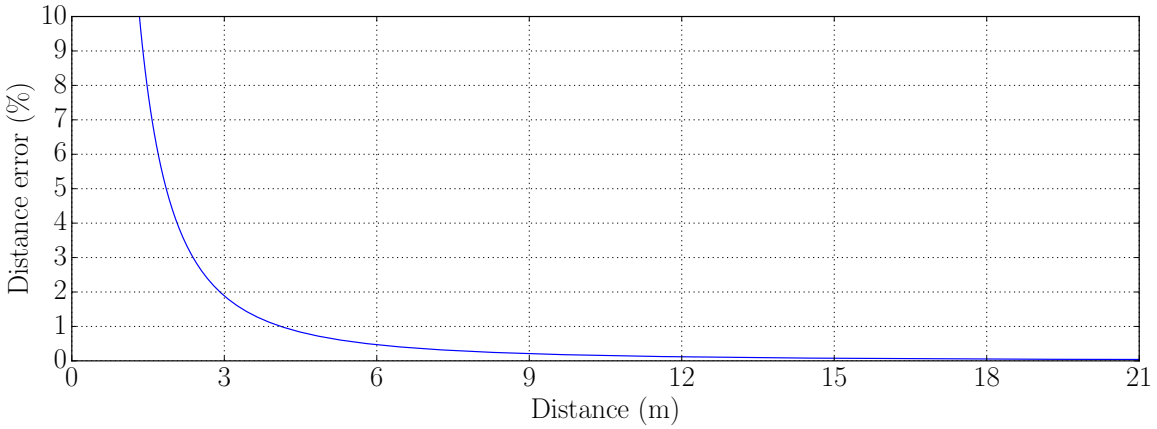


Figure 12: Error in distance estimation when using the simplified equation (32) over the full equation (33).

Algorithm 3 The relative localization algorithm

Input:
  B ▷ *set of bounding boxes of detected MAVs*
  camera_calibration ▷ *calibration parameters of the camera*
  $\mathbf{T}_c^w$ ▷ *transformation from camera to world coordinate system*
  n_f ▷ *order of the sliding window distance filter*
Output:
  M ▷ *set of estimated positions of detected MAVs*
$\mathrm{M} \leftarrow \emptyset$
filter $\leftarrow$ initialize_filter (n_f)
for all $\mathrm{b} \in \mathrm{B}$ **do**
  ▷ *estimate distance of the MAV as described in section 3.3*
  $l \leftarrow$ estimate_distance (b, camera_calibration)
  ▷ *filter the distance using a sliding window averaging filter*
  $l_f \leftarrow$ sliding_window_filter $(l,$ filter$)$
  if distance_valid (l_f) **then**
    ▷ *get the projection unit vector as described in section 3.2*
    $\mathbf{a} \leftarrow$ project_line (b, camera_calibration)
    ▷ *multiply the unit vector by the distance to get the point*
    $\mathbf{x}_c \leftarrow l_f \cdot \mathbf{a}$
    ▷ *transform the point from camera to world coordinate system*
    $\mathbf{x}_w \leftarrow$ transform $(\mathbf{x}_c, \mathbf{T}_c^w)$
    ▷ *add the point to the set of estimated MAV positions*
    $\mathrm{M} \leftarrow \mathrm{M} \cup \mathbf{x}_w$
  end if
end for

4 Evaluation of presented methods

The system was studied in realistic simulations using the Gazebo simulator[4] to determine whether it is feasible to use the neural network for relative localization. Parameters for the relative localization method, described in the previous sections, have been chosen based on these simulations. From the simulations, two leader-follower scenarios have been designed to be tested in a real-world experiment.

The simulation set-ups are described in section 4.1. In section 4.2 is described fine-tuning of the methods on datasets from previous real-world experiments, with focus on the distance estimation, to ensure robustness of the system under real conditions. After proving that the system is working robustly in simulations as well as on the datasets, it was tested in several real-world experiments, where an MAV was controlled based on the relative localization system.

4.1 Simulations

To confirm the hypothesis that the zoom-in redetection method, described in section 2.3.1, improves estimation of the bounding box, a simulated experiment was designed. In this experiment, one MAV was holding position at coordinates $\begin{bmatrix}0, & 0, & 5\end{bmatrix}^{\mathrm{T}}$ of the global coordinate system. Another MAV with a simulated camera sensor, pointing in the direction of the stationary MAV, was running the CNN detector (described in section 2) on images from the simulated camera, and logging the detected bounding boxes. After 100 measurements have been logged at a certain position, the camera MAV changed its distance from the stationary MAV by 0.25 m. This was repeated from a starting mutual distance 0.5 m to the maximal distance at which the stationary MAV was detected by the neural network, which was around 13.75 m.

The experiment was done with a standard single detection per image, and with the zoom-in redetection method. Both methods were then compared to the ground truth bounding box for each distance. The ground truth was obtained by projecting the edges of the stationary MAV using known calibration parameters of the simulated camera, similarly as in section 3.2. Because the width of the bounding box is used for the relative localization (see section 3.3), it was used as a comparison metric.

The zoom-in redetection (or double detection) method proved to provide more accurate estimates of bounding box width, as was expected. Average absolute difference between the ground truth and single detection bounding box widths (which are relative to width of the camera image) was 0.026, whereas for the double detection it was 0.007, which is a 73% error decrease. A comparison between the two methods and the ground truth bounding box

[4] `http://gazebosim.org/`

width over distance is in Figure 13, where the difference can clearly be seen. The bounding box height estimation was unsurprisingly also improved from an average absolute error of 0.012 to 0.007, which is a 42% error decrease, although this is irrelevant for the purposes of relative localization in this work.

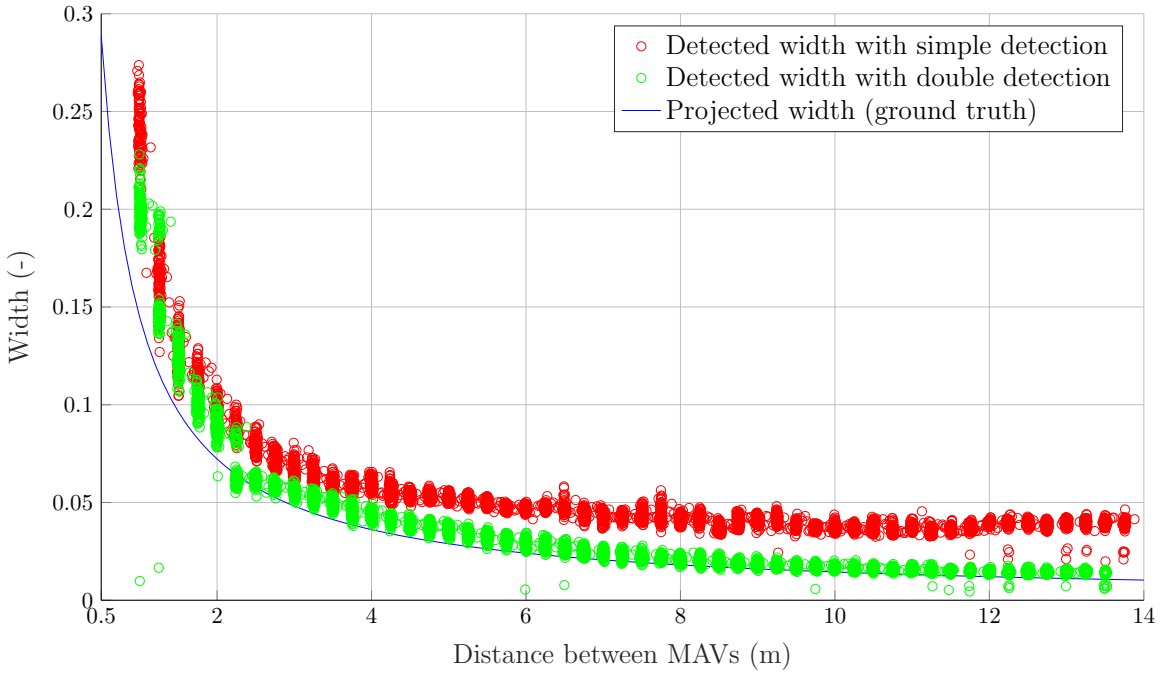


Figure 13: Comparison of bounding box width over distance for simple and double detection.

4.2 Dataset experiments

To determine usability of the proposed methods on real-world data, experiments on a dataset from real-world experiments have been concluded. The dataset was collected during an unrelated experiment using the same experimental MAV platform, as was later used for real-world experiments in this thesis (see section 5 for a description of the platform and the real-world experiments). It consists of video data from an onboard camera and corresponding ground truth positions of the MAV, carrying the camera, as well as of another MAV, which appears in most of the images in the video. The dataset was used for evaluating the vision-based relative localization, presented in this thesis. The ground truth positions were acquired using the same RTK-GPS system, as is described in section 5.

The relative localization method was applied to the dataset to generate estimates of the relative position of the MAV in the image. Similarly as in the simulations, described in the previous section, both the single detection and zoom-in redetection methods have been

Detection method	Average absolute error	Maximal absolute error
Single detection	1.59 m	19.77 m
Zoom-in redetection	1.49 m	11.18 m
Zoom-in redetection, filtered	1.43 m	11.19 m

Table 3: Results of dataset experiments using different methods.

used in order to compare them. Because errors in distance estimation were expected to be the bottleneck of the relative localization (as was confirmed in the real-world experiments, see section 5), focus was put on fine-tuning the distance estimation method. Outliers in the distance estimation can lead to unstable and potentially dangerous behavior in case that position of the MAVs is controlled based on the relative localization, as it is in the leader-follower experiments presented in section 5.2. It is presumed that in that case the MAVs would move relatively slowly, unlike in the dataset used in this section, where the MAVs were moving fast (indicated by the sharp changes in relative distance of the MAVs in Figure 14). Because of this a sliding window filter, which takes an average of the last n_f values, was applied to the estimated distance to suppress the outliers. Length of the sliding window was empirically chosen as $n_f = 5$.

Graph of the estimated distances during the course of the experiment is in Figure 14. Graph of the estimated distances with respect to the ground truth distance is in Figure 15. The average absolute error as well as the maximal absolute error in the distance estimation during the whole experiment is in Table 3. It can be seen that the double detection improves precision significantly, and also suppresses outliers, although some outliers still remain. The filtering does not have such a large positive effect. Although some outliers are removed, new ones are introduced due to fast changes in the MAV distance and slower response of the filter when compared to the unfiltered estimation (see the graph in Figure 14 at times 280 s, 395 s, etc.). However since it is assumed that in case the maximal speed of the MAVs is limited when they are controlled based on the relative localization, the negative effects of the filter will be mitigated. This is why the filtering was used in the real-world experiments.

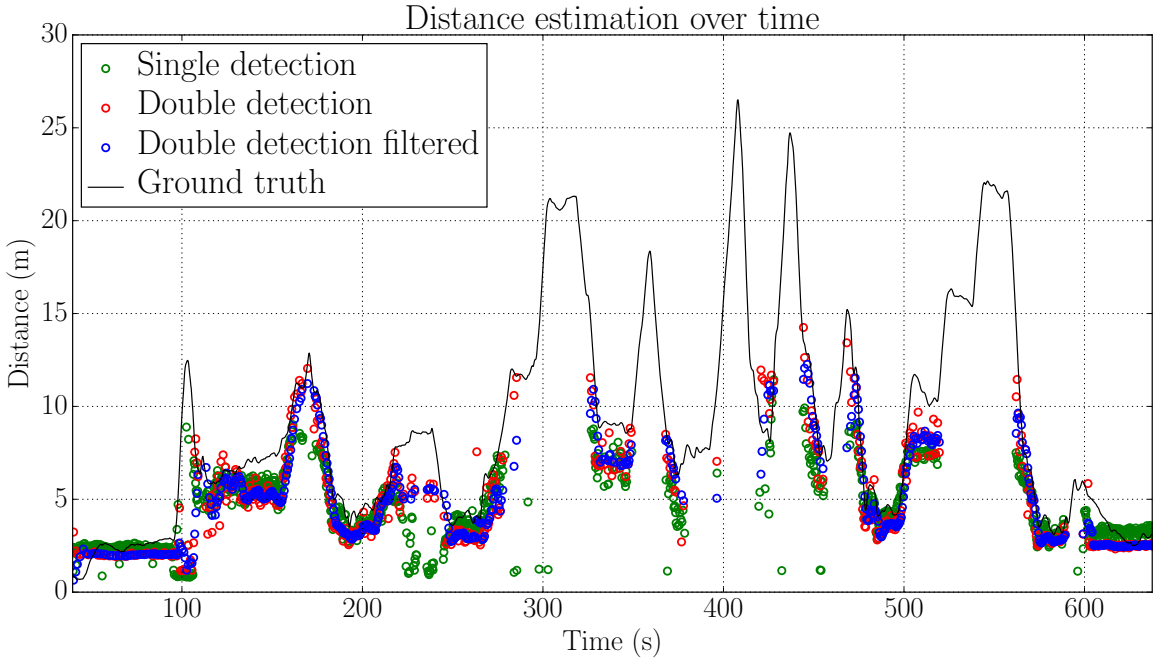


Figure 14: Comparison of estimated relative distance of the MAVs over time using the different methods.

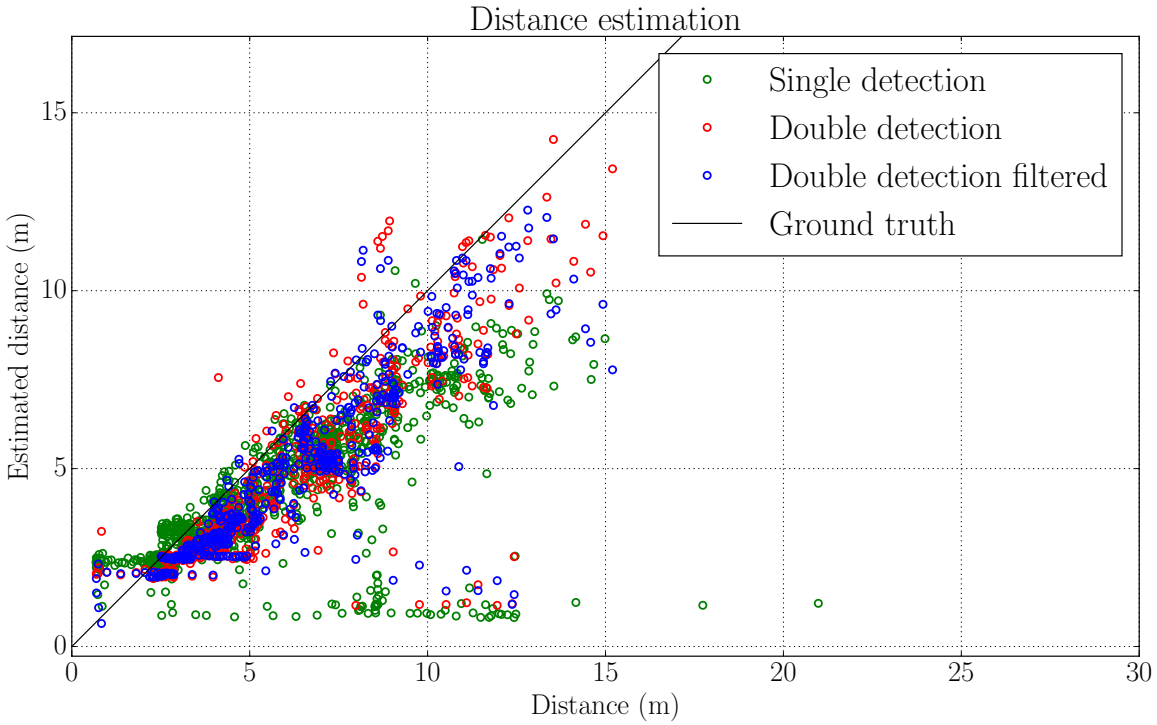


Figure 15: Comparison of estimated relative distance of the MAVs with respect to ground truth distance using the different methods.

5 Real-world experiments

Two sets of real-world experiments were conducted to test precision and robustness of the system under real conditions. First, the ability of the system to detect an MAV was successfully tested in a one-dimensional leader-follower scenario. This experiment is described in section 5.1. After the first experiment demonstrated reliability of the system in the simplified scenario, the second set of experiments was conducted. In the second experiment the leader-follower formation moved in two dimensions in the XY plane of the global coordinate system. Target of the second scenario was to test the relative position estimation method (described in section 3), which was successful, as is described in section 5.2.

An MAV platform (see Figure 1), developed by the Multi-robot Systems Group from the Faculty of Electrical Engineering, Czech Technical University in Prague, for the MBZIRC competition[5], was used to carry out all the experiments. It is a hexacopter MAV with an onboard computer, capable of running the neural network in real-time. The hexacopter is stabilized using the Pixhawk Px4 flight controller [55], and controlled using a Model Predictive Controller (MPC). It carries an RTK-GPS sensor (model Tersus Precis-BX305), which provides 1 cm horizontal localization precision under good conditions [56]. This sensor was used as a ground truth for the horizontal position of the MAVs during the experiments. The vertical position (height) of the MAVs was obtained using fusion of two downward-pointing distance sensors, namely Garmin Lidar Lite v3 with a typical accuracy of ± 0.1 m [57], and Teraranger One with a typical accuracy of ± 0.02 m [58]. If an MAV had the role of a follower in an experiment, it was equipped with a Mobius ActionCam camera This camera was capturing images with 1280×720 pixel resolution at 30 Hz, which served as inputs of the neural network. For a detailed description of hardware of the MAV platform, see [22, 23, 24], and for description of the algorithms, used for controlling the MAV platform, see [25, 26].

During all experiments, a collision avoidance system for the MAVs was used for security reasons with an activation radius $r_{collision} = 3$ m. If two MAVs approached each other closer than $r_{collision}$, the collision system was activated and the MAV with a lower (predefined) priority evaded the other one by flying into a higher altitude. Ground truth positions of the MAVs, based on the RTK-GPS sensor, were shared for needs of the collision system, but they were not used in any other way. The collision avoidance system is described in detail in [3].

5.1 One-dimensional leader-follower scenario

One experiment using this scenario was conducted with the aim to demonstrate that the implemented neural network system is capable of detecting the leading MAV reliably

[5] http://www.mbzirc.com

and in real-time under real-world conditions. A follower MAV was following a leader MAV along one axis based on outputs of the neural network, while the other two coordinates were constant. Distance between the MAVs was neglected for this experiment and it was not estimated. Specific setup of this experiment is described in section 5.1.1, and the results are discussed in 5.1.2. A video from the experiment is available at `https://youtu.be/RK3kTX56yFo` and at the attached CD.

5.1.1 Setup of the experiments

The leader MAV was flying randomly along the X-axis of the global coordinate system (which is given by the onboard RTK-GPS sensor) in an interval $[-20\,\mathrm{m}; 20\,\mathrm{m}]$ with a maximal speed $0.5\,\mathrm{m\,s^{-1}}$ (see Fig. 16). The Y position of the leader was set to 8 m and its height to 5 m. The follower MAV carried a camera, pointed in the direction of the Y axis, providing images for the neural network, which was detecting the leader MAV. The Y position of the follower was set to 0 and the height to 5 m. Its speed along the X axis was controlled based on a horizontal deviation of center of the detected MAV bounding box from center of the image. This deviation was an input to a PID controller, which was stabilizing it at zero by setting the speed setpoint for the MPC controller of the follower MAV in the X coordinate of the global coordinate system.

The PID controller is formulated as

$$e\left[k\right] = u\left[k\right] - 0.5, \tag{34}$$
$$e_D\left[k\right] = {}_{\Delta}t\left[k\right]\left(e\left[k\right] - e\left[k-1\right]\right), \tag{35}$$
$$e_I\left[k\right] = \max\left(\min\left(e_I\left[k-1\right] + {}_{\Delta}t\left[k\right]e\left[k\right], e_{I,max}\right), -e_{I,max}\right), \tag{36}$$
$$v_x\left[k\right] = K_P e\left[k\right] + K_I e_I\left[k\right] + K_D e_D\left[k\right], \tag{37}$$

where $u\left[k\right]$ is the horizontal coordinate of the center of the bounding box at time step k (relative to width of the camera image), ${}_{\Delta}t\left[k\right]$ is the time, elapsed from the last update of the controller, at time step k (in seconds), and $v_x\left[k\right]$ is a setpoint for the MAV speed in the direction of the X-axis at time step k (in meters per second). The integrated deviation $e_I\left[k\right]$ is bounded to the interval $[-e_{I,max}; e_{I,max}]$.

Parameters of the controller were tuned in simulations before the experiment and their values were

$$K_P = 26,$$
$$K_I = 2,$$
$$K_D = 2,$$
$$e_{I,max} = 0.1.$$

Detection parameters of the neural network and the non-max suppression algorithm (for

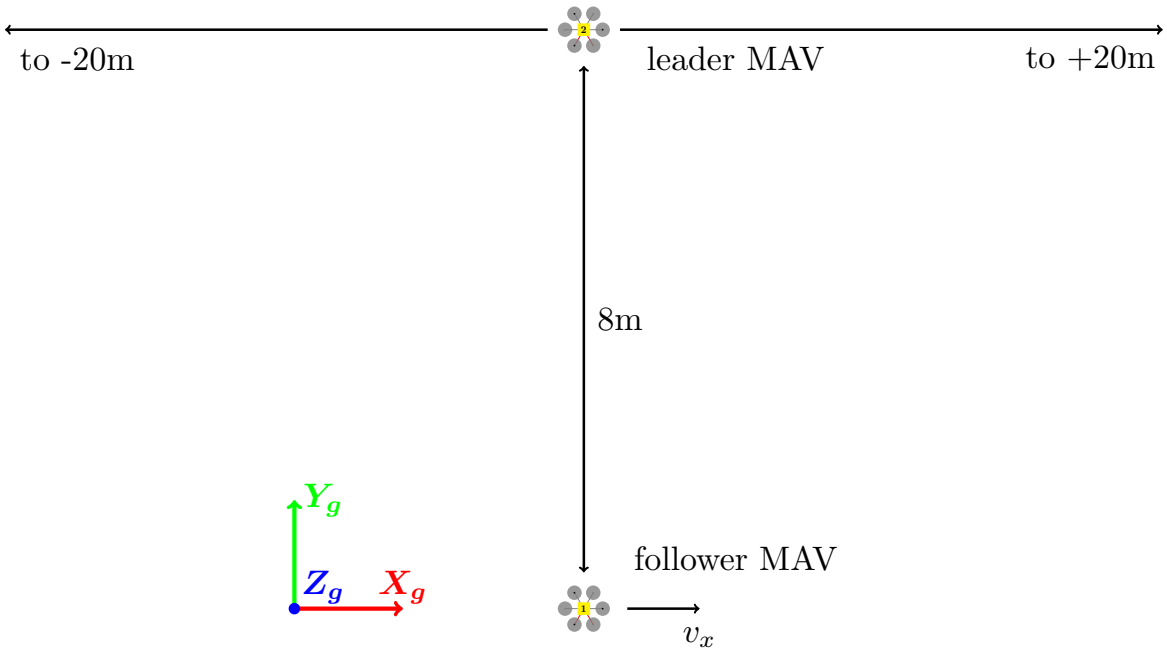


Figure 16: Setup of the one-dimensional leader-follower experiment. Axes of the global coordinate system are marked with the red, green and blue vectors $\boldsymbol{X_g}, \boldsymbol{Y_g}, \boldsymbol{Z_g}$. v_x is speed of the follower MAV in the direction along the X-axis.

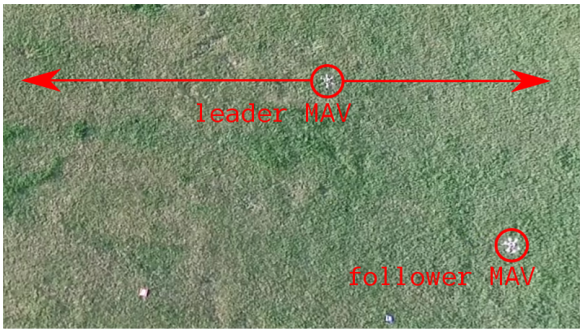
their description, see section 2.1) were

$$p_{thresh} = 0.2,$$
$$o_{thresh} = 0.2,$$

and weights $\mathbf{w}$ of the neural network, which were trained as described in section 2.2. Neither the zoom-in nor subsquare detection methods, described in sections 2.3.1 and 2.3.2, were used in this experiment, since precise relative localization was not necessary in this setup as the follower MAV was only controlled based on the horizontal position of the detected bounding box in the image.

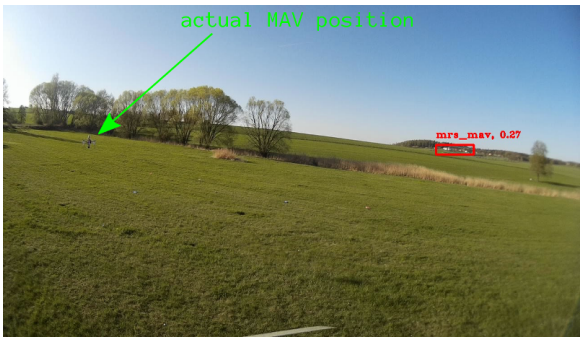
A single detection during an update was used directly as an input of the PID controller. In case that there were multiple detections in one image (which was possible due to false positives), the one with the closest coordinates (based on a euclidean distance metric) to the previously used bounding box was employed for controlling the MAV speed. If there was no previously used bounding box (i.e. at the beginning of the algorithm), no control update was done until there was an image with only one detection.

(a) Top view at time $t = 150s$



(b) Top view at time $t = 160s$

(c) Detections at time $t = 150s$



(d) Detections at time $t = 160s$

Figure 17: Photos from the end of the 1D leader-follower experiment. Top view of the experiment is in Figures 17a and 17b. Images from the onboard camera of the follower MAV with highlighted detections are in Figures 17c and 17d.

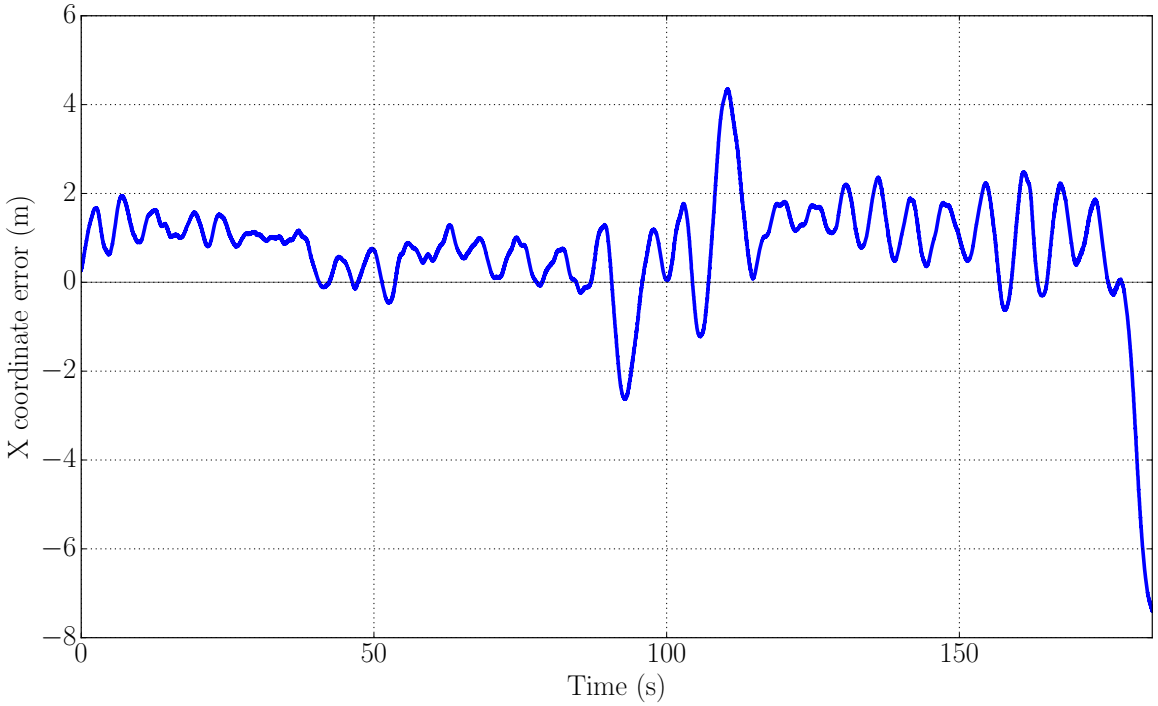


Figure 18: Error of the follower X coordinate during the one dimensional leader-follower experiment.

5.1.2 Results

The experiment lasted for 180 s, until the follower lost visual contact with the leader. This was due to a few frames where the leader MAV was not detected by the neural network and instead one false positive in the background was detected (see Figure 18). In the next frame the follower started following the false positive while ignoring the leader because of the rule how to deal with multiple detections, used during this experiment (described in the previous section). As a result, the leader eventually flew out of the camera frame and the follower got lost. For the next experiments, the strategy of dealing with unexpected multiple detections was changed to always using the bounding box with highest confidence, and this problem was not repeated (see section 5.2).

The average absolute error of the follower (difference between X coordinates of the leader and follower) was 1.12 m, and the maximal absolute error before the follower lost contact was 4.34 m. The error is caused mainly by lag of the camera (0.13 s) and the neural network (0.27 s) and by a control lag of the PID controller itself. A graph of the error over time during the experiment is in Figure 18. The spikes in the error around the 100 s mark were caused by false positives. Oscillations of the error are caused partially by a too aggressive PID tuning, and also by tilting of the follower MAV when it is changing position, which shifts the detection in the image, causing a small instability in the system.

Although the experiment ended with the follower eventually losing the leader MAV, the results are deemed satisfactory for proving reliability of using the neural network for MAV detection in a real-world experiment. With a more robust and more sophisticated approach, such as using a Kalman filter to estimate the X coordinate and speed of the leader, a better result could be achieved, but that was not the aim of this thesis. This experiment proved that the used neural network structure and weights are reliable and focus was put on the experiments, described in the next section, which used the full relative localization method.

5.2 Two-dimensional leader-follower scenario

Similarly as in the previous scenario, a follower MAV was following a leader MAV in a static formation, but this time in two dimensions, and using the relative localization method, described in section 3. The experiment was repeated three times with the same setup. Setup of the experiments is described in the next section. Results are discussed in section 5.2.2. A video from the experiments is available at `https://youtu.be/8nvaTJoOrIg` and at the attached CD.

5.2.1 Setup of the experiments

The leader MAV flew through a predefined trajectory with a constant speed $v_{leader} = 0.5\,\mathrm{m\,s^{-1}}$ and a constant height $z_{leader} = 11\,\mathrm{m}$ (see Fig. 19). The 2D position of the leader was estimated from its detected bounding box using the Algorithm 3 (as described in section 3) and the follower MAV was keeping a static formation in relation with this position (a constant offset of 6 m in the Y-axis).

The Z coordinate of the leader in the global coordinate system was not estimated, and the follower was flying in a constant height above ground, although it would require minimal modification of the algorithm to follow the leader MAV in the Z coordinate as well. This was due to safety reasons, so that the follower MAV would not try to descend into the ground or fly to extreme heights in case of a persistent false positive, because the relative localization algorithm was not tested in a real-world experiment beforehand. A true 3D leader-follower scenario was not tested after this experiment due to limited availability of the experimental platforms, but will be a part of future work.

Parameters of the neural network and the non-max suppression algorithm (for their description, see section 2.1) were

$$p_{thresh} = 0.2,$$
$$o_{thresh} = 0.2,$$

and weights $\mathbf{w}$ of the neural network, which were trained as described in section 2.2. The zoom-in and subsquare detection methods, described in sections 2.3.1 and 2.3.2, were both

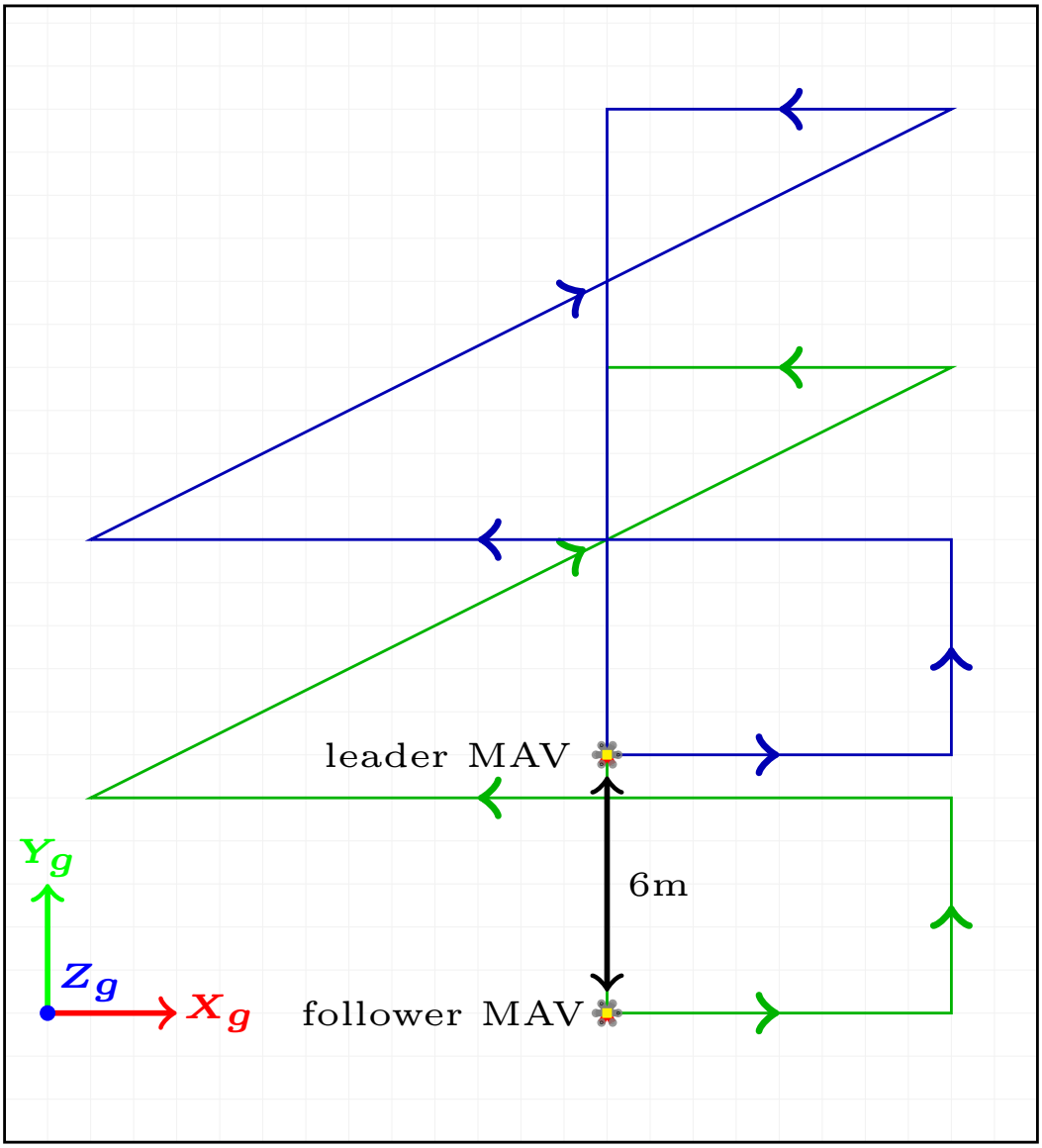


Figure 19: Setup of the two-dimensional leader-follower experiment. The global coordinate system axes are marked with the red, green and blue vectors $\boldsymbol{X_g}, \boldsymbol{Y_g}, \boldsymbol{Z_g}$. The trajectory setpoint of the leader MAV is marked with blue lines. Ideal trajectory of the follower MAV in case of perfect following is marked with green lines. Grid size is one meter.

used during this experiment to make the relative localization more reliable and precise. If there were multiple detected MAVs in an image (due to false positives), the one with the highest confidence (estimated probability) was used.

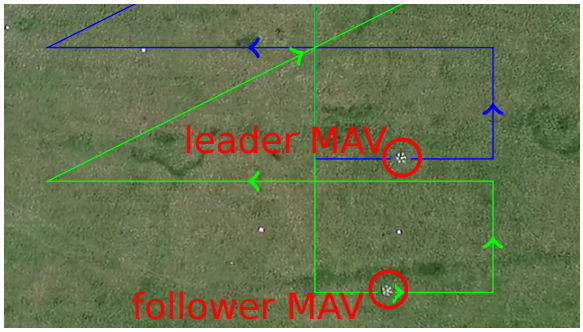
Trajectory of the leader was designed to test the relative localization method under different circumstances. Namely, the following interesting sections of the trajectory have been included:

- the leader is moving parallel to the X axis of the camera coordinate system (horizontally in the image, see section 3.2),
- the leader is moving in the positive direction of the Z axis of the camera c.s. (in the direction in which the camera is pointing),
- the leader is stationary for a short moment,
- the leader is moving in the X and Z axis of the camera c.s. at the same time,
- the leader is moving in the negative direction of the Z axis of the camera c.s.

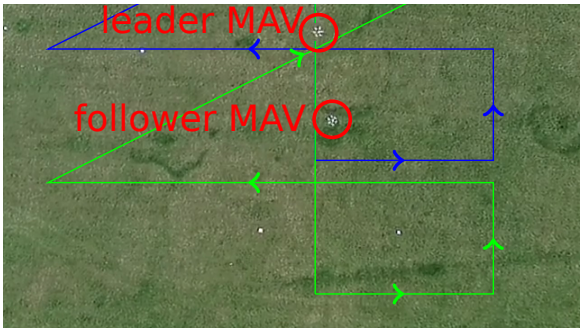
These have been put together into a single trajectory so that the parts where the relative localization was expected to fail with highest probability (due to losing visual contact or due to collision avoidance) are at the end. The final trajectory is displayed in Figure 19. This trajectory has been used in the first two experiments. A modified, shorter version of this trajectory has been used in the third experiment.

5.2.2 Results

In all three experiments, the follower successfully followed the leader in the preset formation during most parts of the trajectory. The first two experiments ended in the last part of the trajectory, where the leader is moving in the negative direction of the camera coordinate system (towards the follower), which proved to be the most tricky movement, as was expected. The follower lost visual contact with the leader when the two MAVs got too close to each other and the collision avoidance was activated. The collision avoidance sent the follower to a height of 15 m above ground, which made the leader (which stayed at 11 m height) go out of the camera image (see Fig. 20). This was because of the inaccuracy of the relative localization added up with the lag, caused by the camera (0.13 s) and the neural network (0.56 s in case of one detection), together with the non-zero reaction time of the follower MAV to a change of a position setpoint. In the third experiment the formation was kept through the whole shorter trajectory, which demonstrates that the follower can follow the leader without losing visual contact if the trajectory is suitable.



(a) Top view at time $t = 10s$



(b) Top view at time $t = 140s$

(c) Detections at time $t = 10s$

(d) Detections at time $t = 140s$

Figure 20: Photos from the first 2D leader-follower experiment. A top view of the experiment is in Figures 20a and 20b. The trajectory setpoint of the leader MAV is marked with blue lines, and the ideal trajectory of the follower MAV in case of perfect following is marked with green lines. Images from the onboard camera of the follower MAV with highlighted detections (in red) are in Figures 20c and 20d. Figures 20b, 20d are from the end of the first experiment, when the collision avoidance was just activated.

Experiment	Duration	Avg. RMS error	Max. RMS error
1	140 s	3.88 m	5.66 m
2	165 s	2.02 m	3.96 m
3	85 s	2.80 m	6.53 m
Experiment	**Avg. X error**	**Avg. Y error**	**Avg. Z error**
1	1.90 m	2.61 m	1.14 m
2	1.06 m	1.24 m	0.74 m
3	1.11 m	2.22 m	0.96 m

Table 4: Details about the three 2D leader-follower experiments.

Average and maximal RMS errors of the relative localization (difference between the predicted and ground truth position of the leader), average absolute errors in separate X, Y and Z coordinates, as well as durations of the experiments are listed in Table 4. Ground truth trajectories of the MAVs, estimated trajectory of the leader based on the relative localization, and RMS errors during the experiments are shown in Figure 21.

From the graph of the RMS errors it can be seen that the largest errors correlate with the leader moving in the direction towards or away from the follower. It can be concluded from this correlation that the relative localization method is better at estimating the direction of the detected MAV than its distance In this specific setup this translates to a better estimation of the X and Z coordinates of the leader than its Y coordinate (in the global coordinate system). This is supported by values of the errors in separate coordinates (see Table 4 and Figure 22), where the error in the Y coordinate is consistently larger than in the X or Z coordinate.

Another source of error during the localization is the time lag between taking an image and getting a new position estimation, because the MAV may have moved since the image was taken. The total lag of the system, caused by the camera and the neural network, was approximately 0.7 s, and since the leader was moving at a maximal speed of $0.5\,\mathrm{m\,s^{-1}}$, the maximal RMS error, caused by the lag, is approximately 0.35 m.

The relative localization system estimated the Z coordinate of the leader with a more or less constant error of -1 m (see Figure 22). This was partially caused by an error in the transformation from the camera coordinate system to the global coordinate system. The camera is offset in the Z coordinate by 0.24 m relatively to the MAV position, but the translation part of the transformation had the opposite sign of the corresponding element due to an oversight. Thus the camera coordinate system was shifted in the opposite direction, causing a error of -0.48 m in the Z coordinate of the global coordinate system.

The remaining part of the Z coordinate error may be caused by a combination of effects: tilt of the camera, because of imperfect mounting, and slope of the field, where the experiments have been conducted. The camera correctly with care, but a mounting error

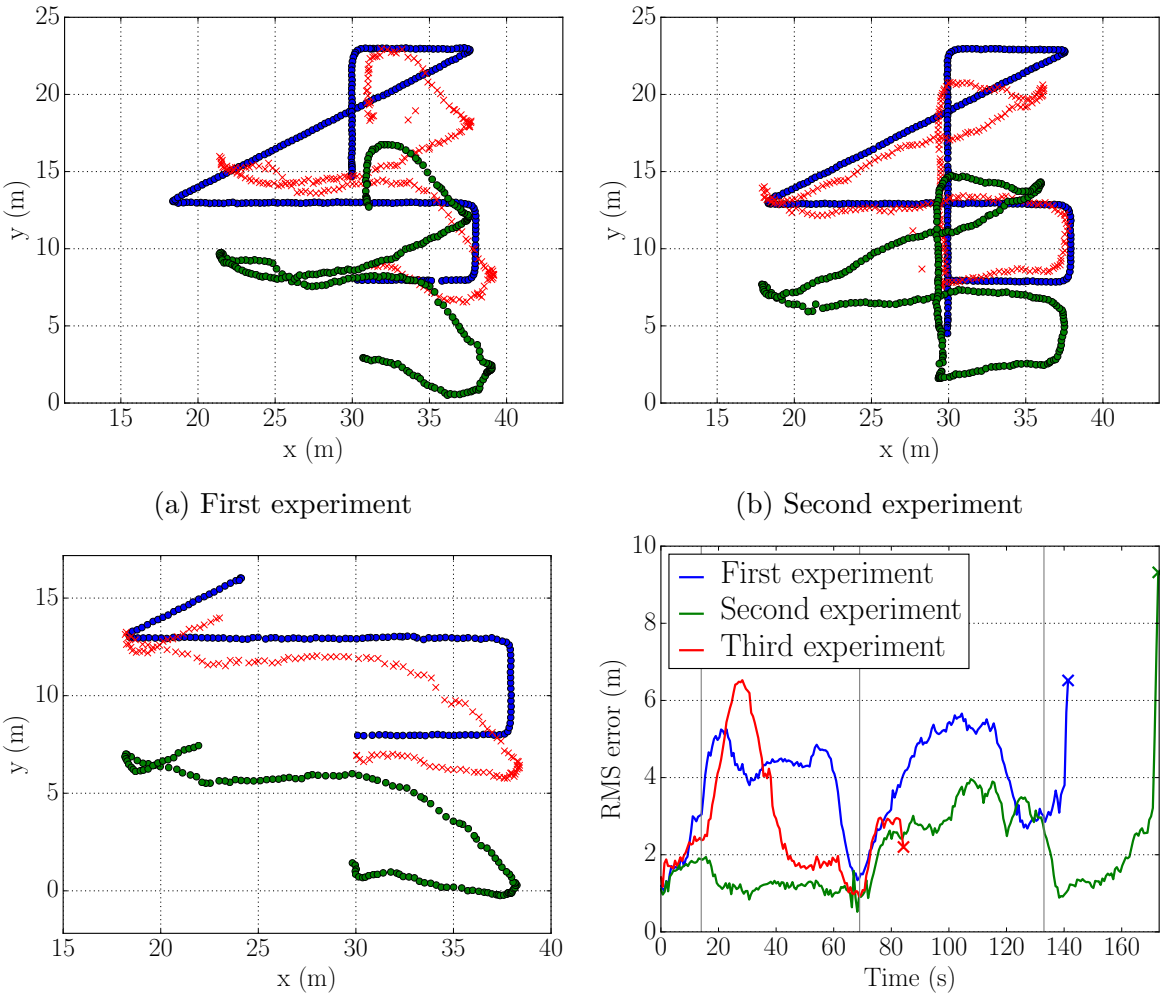
of one degree is hard to avoid with the used rudimentary mounting system. One degree mounting error leads to 0.1 m localization error at 6 m distance to the target. Slope of the ground can cause an error in a similar fashion, because ground truth height of an MAV (the Z coordinate of their position) is determined as its distance to ground, which is measured by two distance sensors. Therefore the ground truth relative height might be different than the actual relative height of the MAVs during the experiments. Both of these errors are proportional to distance between the MAVs, and since the MAVs were keeping the distance approximately the same throughout the experiments, these two causes might explain the almost constant error in the Z coordinate.

5.3 Experiments summary

Viability of the presented relative localization method was demonstrated in the real-world experiments with an average RMS localization error 2.86 m and maximal RMS error 6.53 m. The experiments show that the relative localization performs well for suitable trajectories, although the error may increase if the MAVs being localized are moving away or towards the camera. This is because estimation of the mutual distance is less reliable than estimation of the horizontal and vertical position of the localized object in the camera image.

Several sources, contributing to the total error, have been identified, including two systematic errors, which are trivial to correct in future utilization of this method. Namely, the systematic errors are using the wrong distance estimation equation (33) instead of (32), and using a wrong transformation between the MAV and camera coordinate systems, and correcting them can improve the localization error by up to 0.5 m.

Precision of the relative localization can also be improved by reducing lag of the neural network output and by using an estimator (e.g. a Kalman filter) to predict position of the localized MAV. Both these solutions are theoretically and technically viable and will be addressed in future work. Reducing the total lag of the localization (by speeding up the estimation or by using prediction) is especially important if the MAVs are moving fast, because the localization error caused by the lag is proportional to speed of the localized object. In case of the experiments, presented in the previous sections, where the leader MAV was moving at $0.5\,\mathrm{m\,s^{-1}}$, the error could be reduced by up to 0.35 m by eliminating the lag.



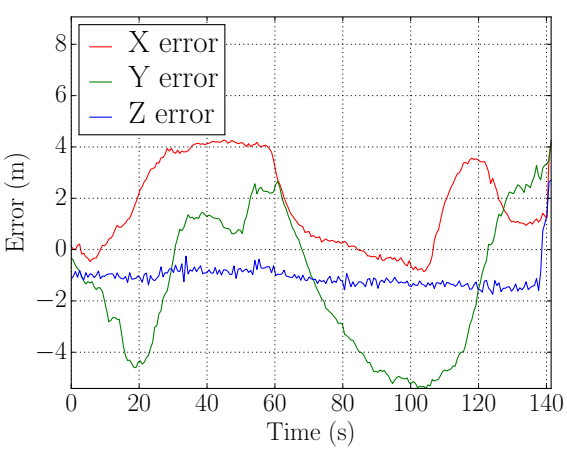
(a) First experiment

(b) Second experiment

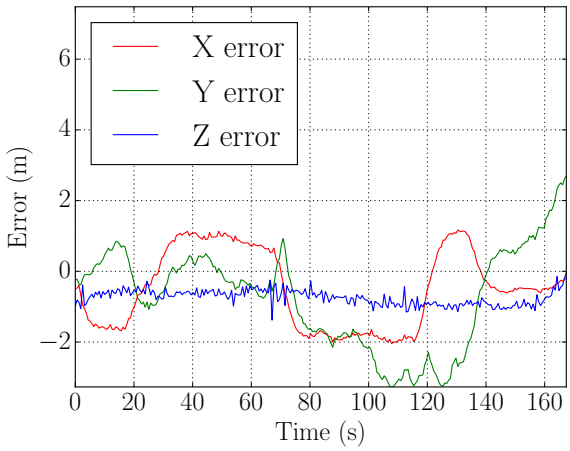
(c) Third experiment

(d) Root mean squared pos. estimation error

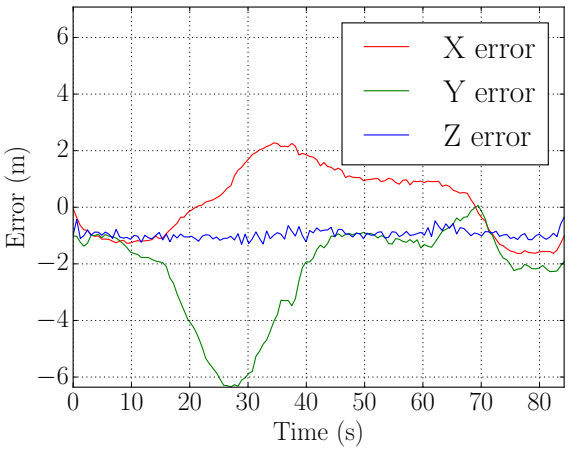
Figure 21: Results of position estimation of the leader MAV in the three 2D leader-follower experiments. A top view (in the global coordinate system) of the experiments is presented in Figures 21a, 21b and 21c. The blue dots represent ground truth positions of the leader MAV, red crosses its estimated positions, and green dots represent positions of the follower MAV. The last Figure 21d is a graph of the position estimation error over time during the three experiments. The first gray vertical line represents time at which the leader MAV changed its heading away from the follower, the second when it started following the inclined part of the trajectory, and the third when it started the last part of the trajectory, heading towards the follower.



(a) First experiment



(b) Second experiment



(c) Third experiment

Figure 22: Estimation errors in separate coordinates during the 2D leader-follower experiment. The coordinates are in the global coordinate system.

6 Conclusion and future work

A vision-based relative localization method for robots was presented in this thesis. It utilizes a convolutional neural network to detect the objects to be localized in a camera image. The main advantages of this approach are that it requires no extra onboard hardware except for the camera, and it works without any markers, unlike most of the other state of the art visual localization methods. No need for extra hardware, such as specialized distance sensors or wireless communication equipment, is especially important when the system is used for relative localization on flying vehicles, because of their limited carrying capability. Since the proposed relative localization system does not require any markers placed on the robots being localized, it can be used also in situations where the localized robots cannot be expected to carry markers, such as in search and destroy missions or in other non-cooperative scenarios.

The proposed method was evaluated in simulations and on datasets from previous experiments, as it is described in section 4.1. The simulations confirmed that using zoom-in redetection, which is introduced in section 2.3, reduces average error of bounding box dimensions of the detected objects, enabling better relative localization, and thus it was used in the real-world experiments. The dataset experiments were used to tune parameters of the relative localization method, and showed that using the method in real-world conditions is viable.

Real-world experiments with Micro Aerial Vehicles (MAVs), which are described in section 5, demonstrated practical usability of the method in two leader-follower scenarios. In the first scenario a leader MAV was flying along a line, and a second MAV was following it in one dimension based on object detection of the neural network from an onboard camera. The experiment lasted 180 s and the average position error of the follower was 1.12 m. This experiment has shown that the object detection can run onboard the MAV in real-time, and that it is sufficiently robust to be used to control an MAV position in a real-world experiment. In the second scenario the leader was flying through a 2D trajectory, and the other MAV was following it in a static formation using the relative localization method, described in section 3. This scenario was tested in three experiments and the average RMS localization error was 2.86 m.

The experiments demonstrated that the system performs well under real-world conditions, and highlighted several areas of possible improvements. One of the sources of localization error during the experiments was the time delay between taking an image with the camera and processing it to receive an estimate of the relative position. This is mostly caused by a transport delay of the image from the camera sensor to the neural network, and by the processing time of the neural network itself. The image transport delay can be decreased by using a different camera sensor with a shorter delay. Processing time of the neural network can be decreased by optimizing the implementation or by using a more suitable computing hardware. One option would be the Jetson TX2 [40], which is a platform,

designed for running convolutional neural networks on embedded hardware, such as MAVs. Preliminary tests confirmed that the neural network used in this thesis has significantly shorter processing time on the Jetson TX2 (see Table 1).

Another way to address the problem of the delay is to use a predictor, such as a Kalman filter, to predict positions of the robots being localized. This would also have the advantage of inherently filtering outliers in the estimated relative position if the predictor was set up accordingly. The predictor may estimate relative positions of the localized robots even when there is no detection from the camera image, generally increasing robustness of the relative localization. On the other hand, a way of associating a detection in an image with the correct predictor, corresponding to the detected robot, would have to be devised if there were multiple robots being localized. This is not trivial since the relative positions, estimated by the method presented in this paper, are anonymous. False positives would have to be addressed in a more sophisticated manner to avoid instantiating predictors for nonexistent robots.

The convolutional neural network (CNN) used in this work is a variation of a general object detection neural network, which was slightly modified to suit the needs of this specific application. Its output is a set of bounding boxes and their confidences, where each bounding box is defined by its center coordinates and its dimensions. The relative locations of the detected robots are calculated from the bounding boxes using camera projection techniques. A specialized CNN, designed from scratch for the relative localization task, might offer superior performance regarding speed and precision. This CNN could be trained to estimate the distance of the target instead of the dimensions of the bounding box. Such approach would simplify the neural network, since it would not estimate unused variables (such as the height of the bounding box or the class probability distribution), and the training process would be more direct, and thus potentially more effective. Incorporating some of the changes introduced in the latest iteration of the YOLO neural network [16] into the CNN structure could improve precision of the object detection. Topics for research in improving the relative localization method also include using an RGB-D camera such as the Intel Realsense [59] to get better distance estimation by utilizing the depth information from the camera, or using multiple cameras facing in different directions to expand the effective field of view.

The aim of this thesis was to design and implement a real-time onboard visual relative localization method for robots, based on neural networks, and to verify it in experiments. This assignment was satisfied, and the developed method has shown good performance, as was demonstrated in the experiments. The method also has potential for further improvement of precision and robustness, as was discussed in this thesis.

References

[1] G. Vásárhelyi, C. Virágh, G. Somorjai, N. Tarcai, T. Szorényi, T. Nepusz, and T. Vicsek, "Outdoor flocking and formation flight with autonomous aerial robots," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2014, pp. 3866–3873.

[2] 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)*, June 2015, pp. 894–903.

[3] T. Báča, D. Heřt, G. Loianno, M. Saska, and V. Kumar, "Model predictive trajectory tracking and collision avoidance for reliable outdoor deployment of unmanned aerial vehicles," *Conditionally accepted in Robotics and Automation Letters (RA-L)*, 2018.

[4] N. Moshtagh, N. Michael, A. Jadbabaie, and K. Daniilidis, "Vision-based, distributed control laws for motion coordination of nonholonomic robots," *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 851–860, Aug 2009.

[5] A. Franchi, P. Stegagno, and G. Oriolo, "Decentralized multi-robot encirclement of a 3d target with guaranteed collision avoidance," *Autonomous Robots*, vol. 40, no. 2, pp. 245–265, Feb 2016. [Online]. Available: https://doi.org/10.1007/s10514-015-9450-3

[6] D. Heřt, "Autonomous predictive interception of a flying target by an unmanned aerial vehicle," Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, 2018.

[7] V. K. Alex Kushleyev, Daniel Mellinger, "Towards a swarm of agile micro quadrotors," *Autonomous Robots*, 2013.

[8] D. Borio, C. Gioia, F. Dimc, M. Bažec, J. Fortuny, G. Baldini, and M. Basso, "An experimental evaluation of the gnss jamming threat," in *24th International Electrotechnical and Computer Science Conference, IEEE ERK*, Sept 2015.

[9] R. Mitch, M. Psiaki, and T. Ertan, "Chirp-style gnss jamming signal tracking and geolocation," *Navigation*, vol. 63, no. 1, pp. 15–37, 2016. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/navi.128

[10] F. Dimc, M. Bažec, D. Borio, C. Gioia, G. Baldini, and M. Basso, "An experimental evaluation of low-cost gnss jamming sensors," *Navigation*, vol. 64, no. 1, pp. 93–109, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/navi.184

[11] T. Krajník, M. Nitsche, J. Faigl, P. Vaněk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail, "A practical multirobot localization system," *Journal of Intelligent & Robotic Systems*, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10846-014-0041-x

[12] A. Breitenmoser, L. Kneip, and R. Siegwart, "A monocular vision-based system for 6d relative robot localization," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2011, pp. 79–85.

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

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *arXiv preprint arXiv:1506.02640*, 2015.

[15] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[16] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[17] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 2014, pp. 580–587.

[18] R. Girshick, "Fast r-cnn," in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015, pp. 1440–1448.

[19] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, June 2017.

[20] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017. [Online]. Available: http://arxiv.org/abs/1703.06870

[21] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[22] G. Loianno, V. Spurny, T. Baca, J. Thomas, D. Thakur, T. Krajnik, A. Zhou, A. Cho, M. Saska, and V. Kumar, "Localization, grasping, and transportation of magnetic objects by a team of mavs in challenging desert like environments," *IEEE Robotics and Automation Letters*, 2018.

[23] V. Spurny, T. Baca, M. Saska, R. Penicka, T. Krajnik, G. Loianno, J. Thomas, D. Thakur, and V. Kumar, "Cooperative autonomous search, grasping and delivering in a treasure hunt scenario by a team of uavs," *Conditionally accepted in Journal of Field Robotics*, 2018.

[24] P. Štěpán, T. Krajník, M. Petrlík, and M. Saska, "Vision techniques for on-board detection, following and mapping of moving targets," *Conditionally accepted in Journal of Field Robotics*, 2017.

[25] T. Báča, P. Štěpán, and M. Saska, "Autonomous landing on a moving car with unmanned aerial vehicle," in *ECMR*, 2017.

[26] T. Baca, G. Loianno, and M. Saska, "Embedded model predictive control of unmanned micro aerial vehicles," in *21st International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2016.

[27] R. Dubé, A. Gawel, H. Sommer, J. Nieto, R. Siegwart, and C. Cadena, "An online multi-robot slam system for 3d lidars," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 1004–1011.

[28] S. Yang, S. A. Scherer, X. Yi, and A. Zell, "Multi-camera visual slam for autonomous navigation of micro aerial vehicles," *Robotics and Autonomous Systems*, vol. 93, pp. 116 – 134, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0921889015302177

[29] K. Boudjit and C. Larbes, "Detection and target tracking with a quadrotor using fuzzy logic," in *2016 8th International Conference on Modelling, Identification and Control (ICMIC)*, Nov 2016, pp. 127–132.

[30] V. Dhiman, J. Ryde, and J. J. Corso, "Mutual localization: Two camera relative 6-dof pose estimation from reciprocal fiducial observation," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013, pp. 1347–1354.

[31] M. Saska, V. Vonasek, T. Krajnik, and L. Preucil, "Coordination and Navigation of Heterogeneous MAV-UGV Formations Localized by a 'hawk-eye'-like Approach Under a Model Predictive Control Scheme," *International Journal of Robotics Research*, vol. 33, no. 10, pp. 1393–1412, 2014.

[32] M. Saska, T. Baca, J. Thomas, J. Chudoba, L. Preucil, T. Krajnik, J. Faigl, G. Loianno, and V. Kumar, "System for deployment of groups of unmanned micro aerial vehicles in GPS-denied environments using onboard visual relative localization," *Autonomous Robots*, vol. 41, no. 4, pp. 919–944, 2017.

[33] M. Saska, "MAV-swarms: unmanned aerial vehicles stabilized along a given path using onboard relative localization," in *International Conference on Unmanned Aircraft Systems (ICUAS)*, 2015.

[34] V. Walter, M. Saska, and A. Franchi, "Fast mutual relative localization of uavs using ultraviolet led markers," in *Accepted to 2018 International Conference of Unmanned Aircraft System (ICUAS)*, 2018.

[35] O. D. Silva, G. K. I. Mann, and R. G. Gosine, "An ultrasonic and vision-based relative positioning sensor for multirobot localization," *IEEE Sensors Journal*, vol. 15, no. 3, pp. 1716–1726, March 2015.

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

[37] J. Redmon, "Yolov2 reference webpage," https://pjreddie.com/darknet/yolov2/, 2013–2016.

[38] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[39] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[40] *Jetson TX2 Developer Kit*, Nvidia, 2017, rev. DA_07976-001.

[41] A. Maas, A. Hannun, and A. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the International Conference on Machine Learning*, Atlanta, Georgia, 2013.

[42] J. H. Francoise Fogelman Soulie, *Neurocomputing*. Springer-Verlag Berlin Heidelberg, 1990, pp. 227–236.

[43] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, Nov 1986.

[44] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results," http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[45] B. Kieffer, M. Babaie, S. Kalra, and H. R. Tizhoosh, "Convolutional neural networks for histopathology image classification: Training vs. using pre-trained networks," in *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Nov 2017, pp. 1–6.

[46] L. Bottou, *Stochastic Gradient Descent Tricks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 421–436. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_25

[47] T. Zhang, "Solving large scale linear prediction problems using stochastic gradient descent algorithms," in *ICML 2004: proceedings of the twenty-first international conference on machine learning. omnipress*, 2004, pp. 919–926.

[48] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, Y. Lechevallier and G. Saporta, Eds. Heidelberg: Physica-Verlag HD, 2010, pp. 177–186.

[49] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[50] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams, "Scalable bayesian optimization using deep neural networks," in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, F. Bach and D. Blei, Eds., vol. 37. Lille, France: PMLR, 07–09 Jul 2015, pp. 2171–2180. [Online]. Available: http://proceedings.mlr.press/v37/snoek15.html

[51] *The OpenCV Reference Manual*, 3rd ed., OpenCV, June 2014.

[52] OpenCV, "Open source computer vision library," https://github.com/opencv/opencv, 2015.

[53] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, Nov 2000.

[54] A. K. Gary Bradski, *Learning OpenCV*. O'Reilly, 2008.

[55] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, no. 1, pp. 21–39, Aug 2012.

[56] *Precis-BX305 GNSS RTK Board datasheet*, Tersus, 2017, rev. V2.0-170324.

[57] *Lidar Lite v3 Operation Manual and Technical Specifications*, Garmin, 09 2016, rev. 0A.

[58] M. Ruffo, M. D. Castro, L. Molinari, R. Losito, A. Masi, J. Kovermann, and L. Rodrigues, "New infrared time of-flight measurement sensor for robotic platforms," 2014.

[59] *Intel RealSense Camera R200 Product Datasheet*, Intel, 2016, rev. 001.

Appendix A CD Content

Names of all root directories on the attached CD are listed in Table 5.

File name	Description
Matous_Vrba_thesis.pdf	Master's thesis in pdf format.
thesis	latex source codes
code	scripts, code and other files of the implementation
videos	videos from the experiments

Table 5: CD Content

Appendix B List of abbreviations

Abbreviations used in this thesis are listed in Table 6.

Abbreviation	Meaning
CNN	convolutional neural network
CS	coordinate system
DOF	degree of freedom
DPM	deformable parts model
MAV	micro aerial vehicle
NN	neural network
NMS	non-max suppression
HSL	hue saturation lightness
GNSS	global navigation satellite system
GPS	global positioning system
GPU	graphical processing unit
IoU	intersection over union
PID	proportional integral derivative
SGD	stochastic gradient descent
SLAM	simultaneous localization and mapping
ReLU	rectified linear unit
RGB	red green blue
RMS	root mean squared
UGV	unmanned ground vehicle
UV	ultra-violet

Table 6: Lists of abbreviations