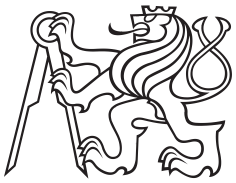
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Detection of Drones Using Neural Networks from Combined RGB Camera and LIDAR Data

Adam Škuta

Supervisor: Ing. Matouš Vrba
Field of study: Cybernetics and Robotics
January 2022

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Škuta Adam** Personal ID number: **474374**

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

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Detection of Drones Using Neural Networks from Combined RGB Camera and LIDAR Data

Bachelor's thesis title in Czech:

Detekce dronů pomocí neuronových sítí z kombinovaných dat RGB kamery a LIDARu

Guidelines:

Use a neural network for detection of drones using an RGB camera in combination with a 3D LiDAR sensor, placed onboard a flying UAV. It is possible to utilize a high-fidelity simulation or automatic dataset annotation using the UVDAR system for training of the neural network (see http://mrs.felk.cvut.cz/projects/midgard). Evaluate precision, speed, detection range and general performance of the algorithm. Compare the results with state-of-the-art solutions utilizing only RGB images as input for the detection. The task is motivated by the problem of autonomous aerial interception (see http://mrs.felk.cvut.cz/projects/eagle-one).

Bibliography / sources:

[1] Ch. R. Qi, O. Litany, K. He and L. J. Guibas, "Deep Hough Voting for 3D Object Detection in Point Clouds," arXiv 1904.09664, 2019.
[2] V. Walter, M. Vrba and M. Saska, "Automatic generation of training datasets for machine learning-based visual relative localization of micro-scale UAVs," RA-L, 2020.
[3] F. Ma and S Karaman, "Sparse-to-dense: Depth prediction from sparse depth samples and a single image," ICRA, 2018.
[4] J. Redmon and F. Ali, "YOLOv3: An incremental improvement," arXiv 1804.02767, 2018.

Name and workplace of bachelor's thesis supervisor:

Ing. Matouš Vrba, Multi-robot Systems, FEE

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

Date of bachelor's thesis assignment: **11.10.2021** Deadline for bachelor thesis submission: **04.01.2022**

Assignment valid until: **30.09.2023**

Ing. Matouš Vrba	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor'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 bachelor'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 express my utmost gratitude to my supervisor Ing. Matouš Vrba for his guidance and great support during my time writing this thesis.

I would also like to thank my parents and friends for their huge support and help.

Declaration

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.

In Prague, 4. January 2022

Abstract

Detection of drones using neural networks from combined RGB and Light Detecting And Ranging (LiDAR) data is tackled in this thesis. Multiple approaches for RGB and LiDAR data fusion into Red Green Blue Depth (RGBD) data are presented. The detection of the drone is realized via a convolutional neural network. All methods are trained and tested and the results are compared with a detection approach utilizing only RGB images. The dataset used in this thesis was generated using multiple simulated environments. The results show that there is a difference between RGBD and RGB approaches and that in some specific scenarios, RGBD approaches provide an advantage over using RGB only in drone detection. Further work on these methods is therefore a worthwhile study.

Keywords: Drone, Convolutional neural network, Detection, LiDAR, Sparse depthmap

Supervisor: Ing. Matouš Vrba

Abstrakt

Detekcia dron pomocou neurónových sietí z kombinovaných dát RGB kamery a Light Detecting And Ranging (LiDARu) je popísaná v tejto práci. Viacero spôsobov kombinácie RGB kamery a LiDARu do Red Green Blue Depth (RGBD) obrázkov je prezentovaných. Samotná detekcia dron je realizovaná pomocou konvolučnej neurónovej siete. Všetky metódy sú natrénované a otestované a následné výsledky sú porovnané s čistou RGB kamerou. Dáta používané v tejto práci boli vygenerované pomocou viacerých virtuálnych prostredí. Výsledky dokazujú, že v niektorých prípadoch RGBD metódy produkujú lepšie výsledky oproti RGB kamere v kontexte detekcie dronov. Budúca práca preto predstavuje ďalšiu zaujímavú štúdiu.

Klíčová slova: Dron, Konvolučná neurónová sieť, Detekcia, LiDAR, Riedka hĺbková mapa

Překlad názvu: Detekce dronů pomocí neuronových sítí z kombinovaných dat RGB kamery a LIDARu

Contents

Figures

Tables

Chapter 1

Introduction

In this thesis, drone detection using neural networks and combined data from an RGB camera and a Light Detection And Ranging (LiDAR) sensor is studied. With the recent development of drone technology, drones have become more readily available to the public and can only be expected to rise in popularity in the future. Drone detection is an important problem to tackle when it comes to tasks such as interception of uncooperative drones [1] or localization of drones in swarm [2] [3]. The methods presented in this thesis are intended for the relative localization of both cooperating and non-cooperating drones. Compared to absolute localization methods, relative localization does not need to rely on pre-existing ground infrastructure and can be used in more environments. LiDAR sensor has been utilized on drones on multiple occasions [4] [5] [6] and an RGB camera provides an easily accessible, cheap and lightweight sensor. Therefore, a fusion of data retrieved from both sensors and its impact on the overall drone detection problem is an interesting problem to tackle. The results can be useful for occasions where a drone is already equipped with a LiDAR sensor. Or they can provide further clarification, whether the addition of LiDAR sensor into the drone detection problem provides any advantage.

The goal of this thesis was to examine whether usage of LiDAR data coupled with RGB images from camera is useful for the localization of drones in contrast to the usage of image data alone. The LiDAR and RGB camera were mounted on top of the observer drone, which took pictures and point-clouds of the target drone. All the measurements were taken inside a virtual environment, with a realistic drone and sensor simulation. The dataset was then processed and used as the input for training and testing a convolutional neural network for the object detection as can be seen in Figure 1.1. The preprocessing was as follows:

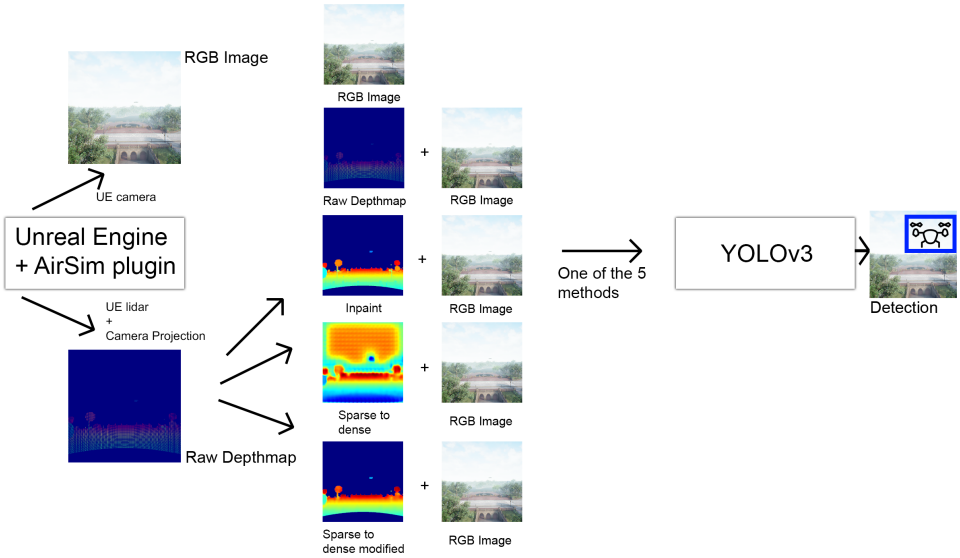


Figure 1.1: Schema of the process.

- coordinate transformation for the non-matching coordinate systems,
- projection of 3D points into a 2D image,
- utilizing different processing methods on sparse LiDAR depthmap to make it denser,
- fusing RGB images and LiDAR data into Red Green Blue Depth (RGBD) images.

The output metrics were then compared with the RGB trained convolutional neural network metrics.

1.1 Related Work

Solving the problem of drone detection has been tackled in different ways. They mostly differ with the sensors that are utilized or by placing markers on flying drones.

- **Static sensors** Current state-of-the-art drone detection techniques utilizing static sensors such as radars, acoustic sensors or Wi-Fi rely

on pre-existing infrastructure. According to the results presented in [7], radar technology faces multiple challenges when detecting smaller aircrafts such as drones because of their small size and high altitude of their flight but shows promising results. When it comes to accoustic sensors, sound of the drone can be a very good source of information but depends heavily on the ambient sound of its environment [7]. In [8], the authors presented a method for drone detection using Wi-FI packets that are sent between the drone and its user. This method relies on the fact that most commercially available drones utilize Wi-Fi packets when it comes to communication with the end user. However, only the presence of the drone in the vicinity of the sensor can be detected using this method and not its exact location, which is crucial for most multi-robotic tasks.

- **Marker-based detection** Some of the relative localization techniques utilize a set of markers that are placed on a target. In [9], an observer drone detects and localizes drones equipped with markers using a spherical camera with a 360° field of view. This system is applied in real-time and provides accurate localization with 4 cm precision. Another approach is provided in [10], where ultra-violet emitters are equipped on a drone and serve as detection markers for detection using an ultra-violet camera sensor. This approach proved to be successful in real-life situations. The downside of utilizing visual markers on target drones is that a hardware modification of the target drone is required. Therefore, this approach is not applicable to all situations, especially when the target drone is not cooperating.

- **Convolutional neural networks** Due to its speed, accuracy and advancements, object detection convolutional neural networks have become a popular technique for relative drone detection such as YOLOv3 [11], CenterNet [12], RetinaNet [13] and Faster R-CNN [14]. One approach of using a convolutional neural network is described in [15]. This approach is similar to the approach presented in this thesis in the sense that it fuses RGB and LiDAR data and use them as input into the modified YOLOv2 [16] convolutional neural network. This approach represents the LiDAR pointcloud as a birds-eye-view map and outputs 7 degrees of freedom, which includes the position, size and rotation of the bounding box for the detected objects. Another approach described in [17] uses a convolutional neural network for the use of relative micro aerial vehicle detection. This approach uses tiny-YOLO architecture [16] and estimates the distance of the detected vehicle from the size of its bounding box. The results prove to be applicable in real-world scenarios. For this thesis, a YOLOv3 architecture is utilized due to its speed and simplicity, making the utilization of the LiDAR data easier.

- **Dense depthmap** A LiDAR produces sparse depthmaps, which are not always satisfactory for further use. Therefore, ways to denser the sparse depthmaps have been explored. In [18] dense depthmaps are generated

by combining LiDAR with Stereo Imagery using a convolutional neural network with a self-supervised training process. This method shows that the method used can create dense depthmaps even from low-resolution LiDAR measurements and therefore reduce the cost by utilizing a low-resolution LiDAR sensor. Another method is described in [19]. It uses sparse depthmaps combined with RGB images as an input to the convolutional neural network to create a dense depthmap. This method is used in the thesis. In [20] an image inpainting method is utilized, which fills in unknown pixel values based on the neighbouring known pixel values. It is usually used for inpainting RGB or grayscale images, but it can be used for sparse depthmaps as well and is used in this thesis.

1.2 Problem Statement

An RGB camera and a LiDAR sensor are assumed to be mounted on an observer drone so that their fields of view overlap. The transformation between coordinate frames of the LiDAR and the camera must be known. The target drone is located in a range of 0.9 meters to 100 meters and has length, width and height dimensions of 0.6, 1.0 and 0.3 meters, respectively. The drone is located in the field of view of both sensors and is not obstructed by any other object. The location of the target drone on the image is required to be determined from the data obtained from both sensors.

Chapter 2

Methodology

In this chapter, five different approaches to the detection problem were tackled:

- RGB-only,
- Sparse-to-dense RGBD,
- Sparse-to-dense modified RGBD,
- Inpaint RGBD,
- Raw RGBD.

The baseline approach in this thesis is to use RGB image and use it as an input into the YOLOv3 [11] convolutional neural network for drone detection.

The second approach utilizes the Sparse-to-dense convolutional neural network [19], which takes sparse LiDAR depthmap and RGB image as an input and outputs dense LiDAR depthmap. The neural network is pre-trained and its output is concatenated with RGB images and used as an input into the YOLOv3 for drone detection

Another approach is to use the output of Sparse-to-dense network and further process the data, eliminating sections of the depth image that are very sparse and using the output concatenated with RGB images to use as input for YOLOv3.

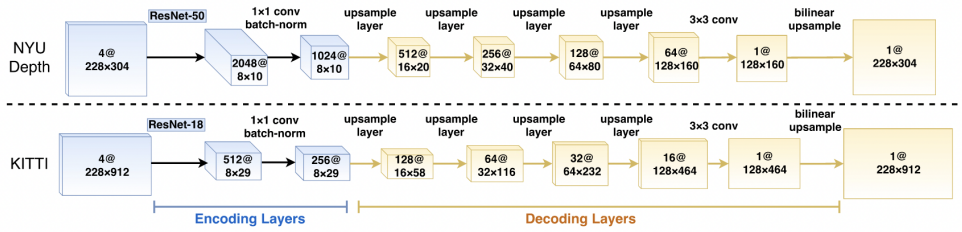


Figure 2.1: Example of different network architectures available. Taken from [19].

A separate approach is to use the inpainting method from the OpenCV Python library[1], which takes sparse LiDAR depthmap and fills in unknown values based on the nearby known values, outputting dense depthmap. The output of the inpainting method is further processed to filter inpainted values that are very far from known values and concatenated with RGB images to use as an input into YOLOv3.

The final approach is to concatenate sparse LiDAR depthmap with RGB images and use it as an input into YOLOv3.

A video game engine Unreal Engine[2]paired with plugin AirSim[3] is used for simulating real-life environments and for generating the dataset.

2.1 Sparse-to-dense

Sparse-to-dense [19] is a convolutional neural network with an implementation in PyTorch publicly available[4]. The input into the network is a sparse depthmap and an RGB image and the output is a dense depthmap. The sparse depthmap is a matrix with few values that have known depth value and the rest are unknowns. In the dense depthmap every value is known. The size of the network is modifiable and can be chosen as a meta-parameter. For the encoding layers, a ResNet-50 or ResNet-18 can be chosen depending on the size of the input image as can be seen in Figure 2.1. The decoding layers consist of 4 upsampling layers and a deconvolutional layer with either stride 2 or 3 or uprojection layer or upconvolutional layer as a choice for training. The loss function used for backpropagation is the least absolute deviations

[1] https://docs.opencv.org/4.x/d7/d8b/group__photo__inpaint.html
[2] https://www.unrealengine.com
[3] https://microsoft.github.io/AirSim/
[4] https://github.com/fangchangma/sparse-to-dense.pytorch

error also known as L_1 error:

$$L_1 = \sum |y_{true} - y_{predicted}|, \tag{2.1}$$

where

- y_{true} are the ground-truth depth values,
- $y_{predicted}$ are the predicted depth values.

The depth input D is sampled from the ground truth depth map D^* with the following formula:

$$D(i,j) = \begin{cases} D^*(i,j), & \text{with probability } p, \\ 0, & \text{otherwise,} \end{cases} \tag{2.2}$$

where

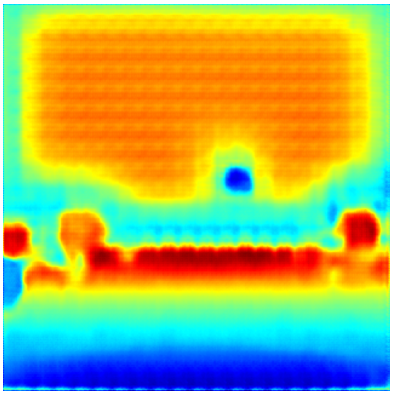
- i, j are coordinates of the input image,
- $p = \frac{m}{n}$, where m number of depth samples to be chosen at the start of training and n is the total amount of available depth samples.

During training several input data augmentations take place. These augmentations include:

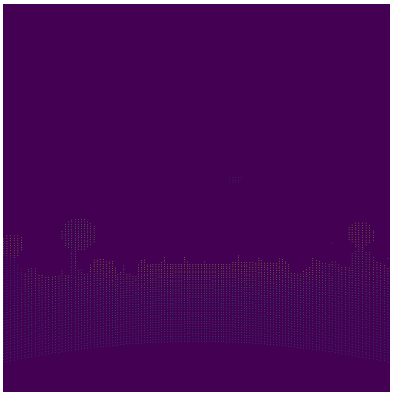
- scaling the input image by a random number $s \in [1, 1.5]$,
- rotating the input image by a random degree $r \in [-5, 5]$,
- scaling the brightness, contrast and saturation of the RGB component of the image by a random number $k \in [0.6, 1.4]$,
- normalizing the RGB component of the image,
- flipping the image horizontally with a 50% chance.

The output of the network is a dense depthmap with the dimensions of the input. Every pixel contains predicted depth measurement in meters. The output of the Sparse-to-dense network will be used for further training later and can be seen in Figure 2.2.

(a) : Input image.



(b) : Output depthmap.



(c) : All depthmap points.

(d) : Selected depthmap points.

Figure 2.2: Sparse-to-dense training.

Sparse-to-dense modified is another approach used in this thesis. It uses the dense depthmap outputted from the trained Sparse-to-dense network and a sparse depthmap and further filters the dense one. At each pixel of the dense depthmap if there exists a known depth value in sparse depthmap within the range of 3 pixels, the dense depthmap value is kept. Otherwise, the value is overwritten to -1.0. The example output of Sparse-to-dense modified can be seen in Figure 2.3.

2.2 YOLOv3

You Only Look Once (YOLO) is a convolutional neural network model used mainly for object detection and recognition [21]. The main advantage is its simplicity in comparison to similar convolutional neural networks, resulting in faster detection speeds. The network belongs to the state-of-the-

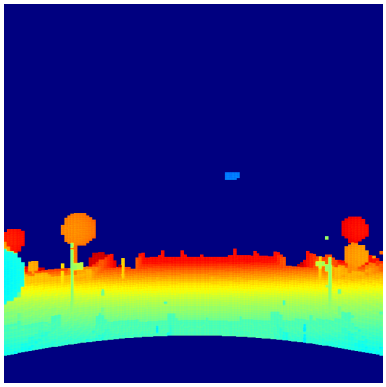


Figure 2.3: Applied filter on Sparse-to-dense output.

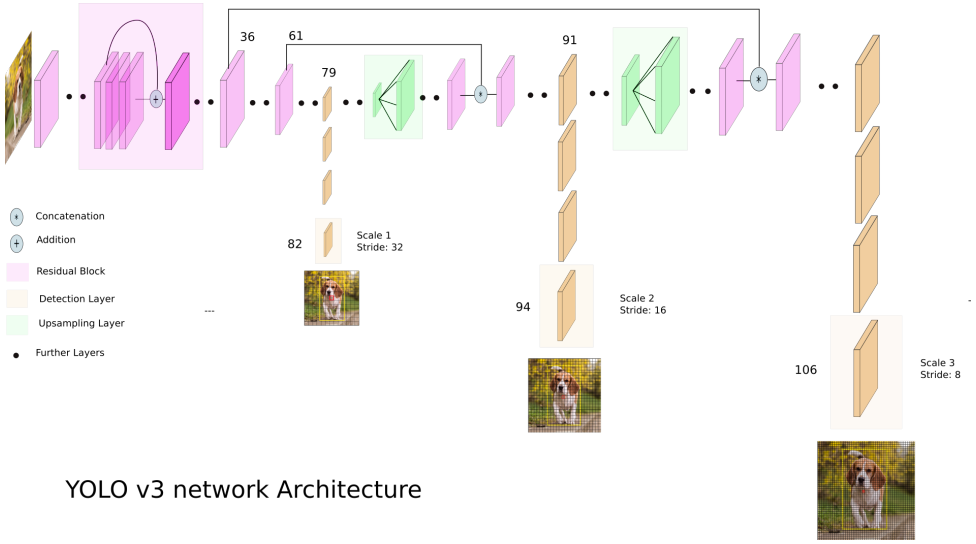


Figure 2.4: YOLOv3 network architecture[5].

art convolutional neural networks for object detection and recognition. The version used in this work is the third version, called YOLOv3[11]. YOLOv3 takes n-channel images as the input. Each of the detection layers outputs three bounding boxes for each of the cells. The backbone called Darknet53 consists of 53 convolutional layers. The original detector consists of 3 detection layers each responsible for detecting objects of various sizes as shown in Figure 2.4.

At each detection layer, the image is divided into multiple grid cells, where each grid cell detects three bounding boxes. The content of one bounding box is as follows:

- t_x, t_y, t_w, t_h are bounding box coordinates,
- p_O is objectness score,

[5] `https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b`

- p_c is class score for each class in the dataset.

The bounding box t_x and t_y coordinates are relative to the upper-left corner of its respective cell, while t_w and t_h are relative to one of the anchors. Anchors are pre-defined default bounding box sizes and can be modified before training. To transform these bounding box coordinates to be relative to the image the following transformation is applied:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x, \\ b_y &= \sigma(t_y) + c_y, \\ b_w &= p_w e^{t_w}, \\ b_h &= p_w e^{t_h}, \end{aligned} \tag{2.3}$$

where

- $\sigma(x)$ is a sigmoid function, defined as $\sigma(x) = \frac{1}{1+e^{-x}}$,
- c_x, c_y are grid cells offsets from the top left corner of the image,
- p_w, p_h are the anchor's width and height, respectively,
- b_x, b_y, b_w, b_h are the bounding box coordinates relative to the image size.

The loss function used during the training is the sum squared error loss or L_2 error described as follows:

$$L_2 = \sum(\hat{\mathbf{t}} - \mathbf{t})^2, \tag{2.4}$$

where

- $\hat{\mathbf{t}}$ is a vector of the predicted bounding box coordinates,
- $\mathbf{t}$ is a vector of the ground truth bounding box coordinates which can be obtained by inverting the transformation in eq. (2.3).

2.3 Image inpainting method based on the Fast Marching Method

Image inpainting is a method used for reconstructing missing values in the image. One such method based on [20] called the Image inpainting method

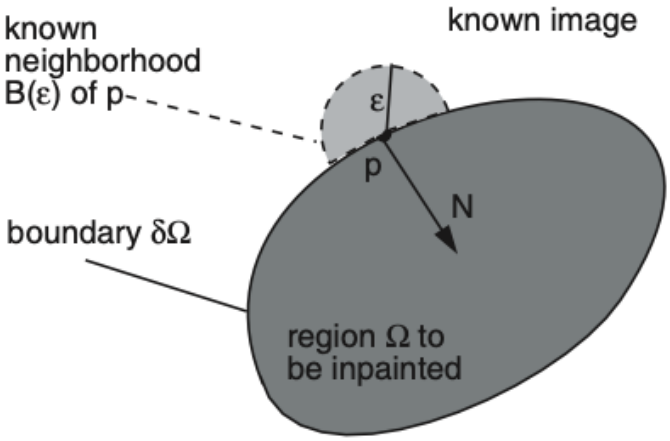


Figure 2.5: Inpainting principle. Image from: [20].

based in the Fast Marching Method was used in this thesis and is described in this section. This method is utilized in this thesis for densing a sparse depthmap. A sparse depthmap is provided as the input and the output is dense depthmap. The same method can be extended for RGB images, but is not needed for this thesis.

The depth value of a pixel to be inpainted is determined by the known neighbouring pixel values. To compute a depth value from one close pixel the following formula is used:

$$I_q(\mathbf{p}) = I(\mathbf{q}) + \nabla \mathbf{I}(\mathbf{q}) \cdot (\mathbf{p} - \mathbf{q}), \tag{2.5}$$

where

- $\mathbf{q}$ is a vector of pixel coordinates with a known depth value,
- $\mathbf{p}$ is a vector of pixel coordinates with an unknown depth value,
- $I(\mathbf{x})$ is a depth value at pixel coordinates $\mathbf{x}$,
- $\nabla \mathbf{I}(\mathbf{x})$ is a gradient vector at pixel coordinates $\mathbf{x}$.

To get a final value for the unknown pixel, the Equation (2.5) is applied on all known pixels in a specified region $B_\varepsilon(\mathbf{p})$ from the unknown pixel as can be seen in Figure 2.5. The resulting function can be expressed as:

$$I(\mathbf{p}) = \frac{\sum_{\mathbf{q} \in B_\varepsilon(\mathbf{p})} w(\mathbf{p}, \mathbf{q}) I_{\mathbf{q}}(\mathbf{p})}{\sum_{\mathbf{q} \in B_\varepsilon(\mathbf{p})} w(\mathbf{p}, \mathbf{q})}, \tag{2.6}$$

Figure 2.6: Example of the inpainting technique. Image from: [20].

where $w(\mathbf{p}, \mathbf{q})$ is a weighting function designed for propagating sharpness of the image and is obtained as the product of the following expressions:

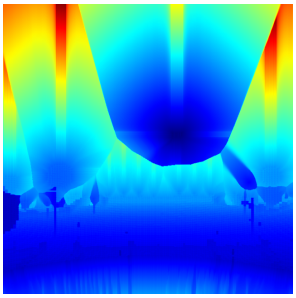
$$\begin{aligned} dir(\mathbf{p}, \mathbf{q}) &= \frac{\mathbf{p} - \mathbf{q}}{||\mathbf{p} - \mathbf{q}||} \cdot \mathbf{N}(\mathbf{p}), \\ dst(\mathbf{p}, \mathbf{q}) &= \frac{1}{||\mathbf{p} - \mathbf{q}||^2}, \\ lev(\mathbf{p}, \mathbf{q}) &= \frac{1}{1 + |T(\mathbf{p}) - T(\mathbf{q})|}, \end{aligned} \tag{2.7}$$

where

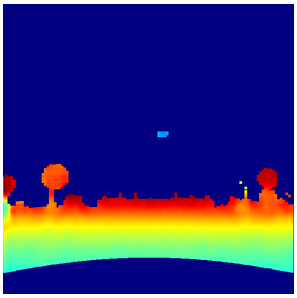
- $\mathbf{N}(\mathbf{x})$ is a normal vector of the boundary to be inpainted at pixel $\mathbf{x}$,
- $T(\mathbf{x})$ is distance of pixel $\mathbf{x}$ to the inpainting boundary.

Eq. (2.6) is iteratively applied to all pixels on the inpainting boundary and advances inside the region to be inpainted until the whole region has been filled. This is implemented via the Fast Marching Method algorithm. Example results of inpainting method can be seen in Figure 2.6. The algorithm is provided in the OpenCV Python library[6]. The results were processed with the same filtering method as for the results of Sparse-to-dense modified method. For further training, only the filtered depth map was used. The results of this approach can be seen in Figure 2.7.

[6] `https://docs.opencv.org/4.x/d7/d8b/group__photo__inpaint.html`



(a) : Non filtered result.



(b) : Filtered result.

Figure 2.7: Inpaint results.

2.4 Coordinate systems

To correctly label the data for training, a position of the target drone in relation to the sensor mounted on the observer drone is required. The AirSim API returns the position of each drone in respect to their starting points. The starting point is a point where the drone spawns in the map. Therefore, a transformation from the starting point of the target drone to the camera mounted on the observer is required. This transformation is obtained as:

$$\mathbf{T} = \mathbf{T}_o^c \mathbf{T}_{os}^o \mathbf{T}_{ts}^{os}, \tag{2.8}$$

where

- $\mathbf{T}_{ts}^{os}$ is a transformation from the starting point of the target drone to the starting point of the observer drone,
- $\mathbf{T}_{os}^{o}$ is a transformation from the starting point of the observer drone to the body of the observer drone,
- $\mathbf{T}_{o}^{c}$ is a transformation from the body of the first drone to the cameras coordinate system.

The homogeneous transformation matrix $\mathbf{T}$ is generally described as:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{bmatrix}, \tag{2.9}$$

where

- $\mathbf{R}$ is a 3x3 rotation matrix,

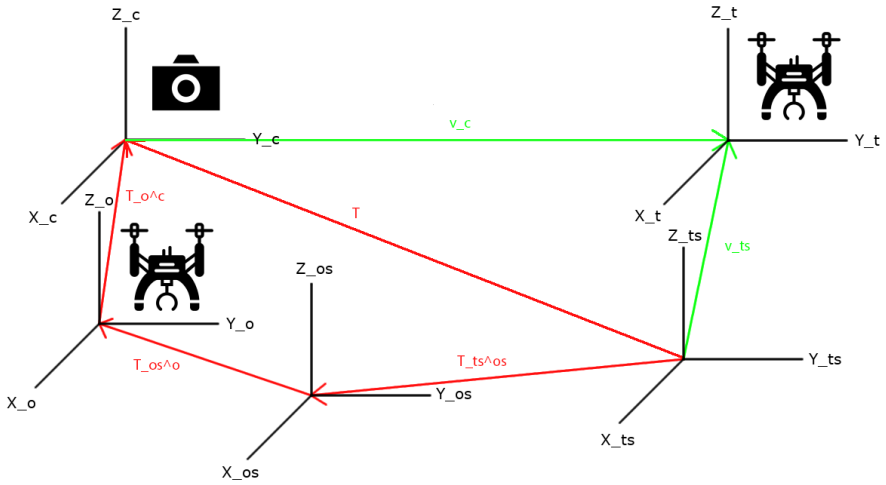


Figure 2.8: Visualization of transformation.

- $\mathbf{p}$ is a 3x1 translation column vector,
- $\mathbf{0}^T$ is a 1x3 row vector of zeros.

To transform a location of the drone represented by vector $\mathbf{v_{ts}}$ into the coordinate system of the camera, as can be seen in Figure 2.8, the following transformation is applied:

$$\mathbf{v_c} = \mathbf{T}\mathbf{v_{ts}}. \tag{2.10}$$

2.5 Camera Model

For the creation of the bounding boxes used for training of the neural networks and for projecting the LiDAR pointcloud to a depthmap, a transformation from the coordinate system of the camera to the pixel values of the image is required. For this task, a pinhole camera model is used as seen in Figure 2.9. The transformation is defined as:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} f\frac{X_c}{Z_c} + c_x \\ f\frac{Y_c}{Z_c} + c_y \end{bmatrix}, \tag{2.11}$$

where

- u and v are pixel coordinate values on the image,

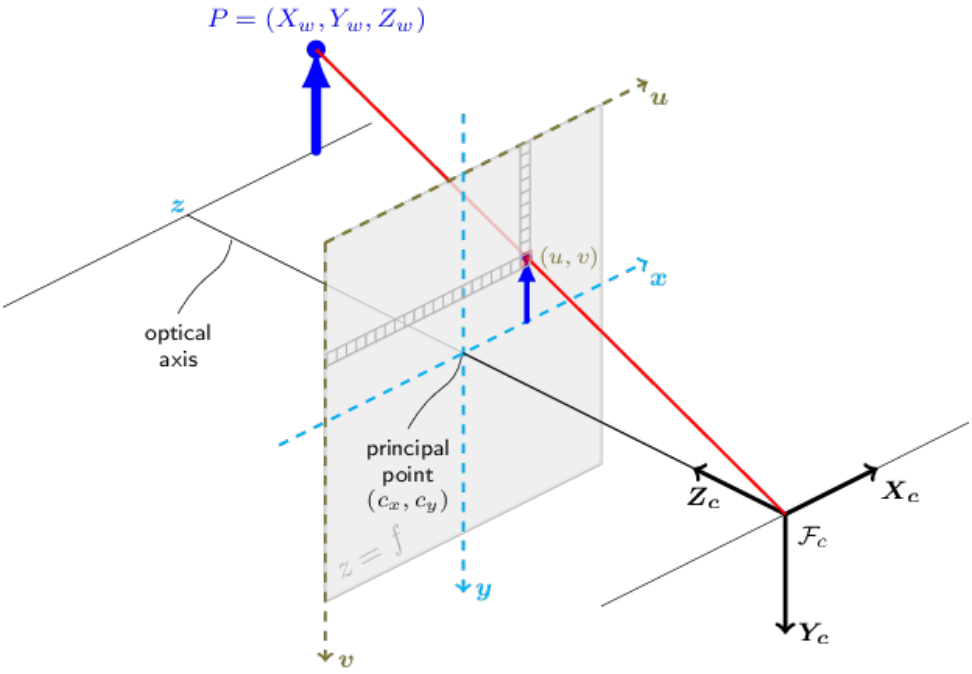


Figure 2.9: Pinhole camera model[7].

- X_c, Y_c, Z_c are coordinate values of a point in the coordinate system of the camera,
- f is the focal length of the camera,
- c_x,c_y are pixel coordinates of the center of the image plane.

The pinhole camera model is only an idealization of a real life camera and no lens distortion is considered. The AirSim simulator simulates an ideal pinhole camera so no other processing was done for the purposes of this thesis.

2.6 Training and testing dataset

The dataset in this thesis was used for training and testing of all used approaches for drone detection. It can be generated in two ways. The first is real-life drone shots mixed with pointclouds from LiDAR mounted on top of a drone. The second is generating a dataset using a realistic virtual environment where a drone, camera and LiDAR are being emulated very close to their real-life counterparts. An advantage to this approach is that a great variety of

[7] `https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html`

Figure 2.10: Unreal Engine user interface.

environments can be chosen a lot of them often inaccessible otherwise (power plant, airport, snowy mountains out of season etc.). Therefore, this approach was chosen for the task. The dataset is publicly available online[8].

2.6.1 Unreal Engine

Unreal Engine is a software tool used for creating realistic 3D environments, most often used as a video game engine. It is written in C++ and open-source supporting a variety of pre-built environments and assets. Example of an Unreal Engine interface can be seen in Figure 2.10. For this work, three different environments were used for the creation of the dataset, that are illustrated in Figure 2.11:

- City Park Environment Collection (2256 samples taken),
- Automotive Winter Scene (1813 samples taken),
- Downtown West Modular Pack (1251 samples taken).

Together, 5320 pictures and labels were generated using two drones. The observer drone was equipped with an RGB camera and the LiDAR sensor and was taking pictures using the camera and pointclouds from the LiDAR.

[8] `https://drive.google.com/file/d/1169jGntZzNYojkDpVVXsOw5izrO1FVOl/view?usp=sharing`

(a) : Automotive Winter Scene.

(b) : City Park Environment Collection.

(c) : Downtown West Modular Pack.

Figure 2.11: Sample photos from each environment.

Figure 2.12: Parrot AR.Drone 2.0. Image taken from [22].

The Parrot AR.Drone 2.0 shown in Figure 2.12 was used as a model for the drone detection target.

2.6.2 AirSim

An open-source plugin for Unreal Engine called AirSim was used for the generation of the dataset. It simulates realistic flight motions of drones as well as seven types of sensors, including RGB cameras and LiDARs, which are the types used for this thesis. AirSim supports both a C++ API as well as a Python API, the latter of which was used for controlling motion of the observer and target drones and capturing the dataset. The location of the second drone was generated through API call, which produces a location of the drone in local coordinates relative to its starting point, which is later transformed to the local coordinates of the first drone carrying the LiDAR

and RGB sensors using eq. (2.10). The location is then projected onto the camera image using eq.(2.11). The ground-truth bounding boxes required for the training of the convolutional neural network were generated the same way but instead of the target's center, corner points of the 3D bounding box were transformed using eq.(2.10) and eq.(2.11). The bounding box dimensions were set to $0.6\,\mathrm{m} \times 1.0\,\mathrm{m} \times 0.3\,\mathrm{m}$. The LiDAR pointcloud was generated via an API call, which returns a set of 3D coordinates of the points measured by the sensor in a coordinate frame of the observer drones camera. Therefore transformation (2.11) was utilized, transforming the LiDAR pointcloud into depthmap. The capturing drone travelled on each map on a 3D cube grid.

2.7 Training

For training and testing purposes, 5320 samples were taken using the AirSim simulator. Each sample consists of:

- a $640px \times 640px$ RGB image from the camera,
- a $640px \times 640px$ sparse depth image from the LiDAR sensor,
- a label file containing coordinates of the ground-truth bounding boxes.

This dataset was split into 3662 training, 407 validation, 1251 testing samples.

2.7.1 Sparse-to-dense

The Sparse-to-dense neural network was trained using RGB images and sparse depthmaps. The training was done for 15 epochs using a batch size of 8. The backbone was Resnet18 and the decoder was set to Deconv3 as described in section 2.1. According to the authors of Sparse-to-dense [19] this provided the best overall results. A processing algorithm was applied to the output depth map, further filtering points that were not in the vicinity of the ground truth depth points. Both filtered and unfiltered depth maps were used for further training and testing to clarify their overall impact.

Figure 2.13: Sample YOLOv3 outputs.

2.7.2 YOLOv3

The following parameters were used for training all the methods:

- number of epochs was set to 15,
- batch size was set to 64,
- learning rate was set to 0.001.

A PyTorch implementation of YOLOv3 was used for training[9]. Further modifications were made. The original implementation of YOLOv3 supports 3-channel RGB images as inputs. For the sake of this work, an RGBD input option using the H5 file system was implemented. The dataset consisted of drones of various sizes ranging from very small (few pixels) to very large closeups. Therefore, 5 detection layers were implemented instead of the original 3. The addition of two more detection layers provide finer cell division when determining the location and the size of the bounding box, which helps detecting smaller objects. The sample outputs can be seen in Figure 2.13. During training, the validation AP followed the training loss and started converging after around the 8th epoch, as is apparent from the graphs in Figure 2.14. Considering this, the chosen weights ensured that the network does not overfit the training and validating dataset. Table 2.1 shows selected weights for each method.

[9] `https://github.com/eriklindernoren/PyTorch-YOLOv3`

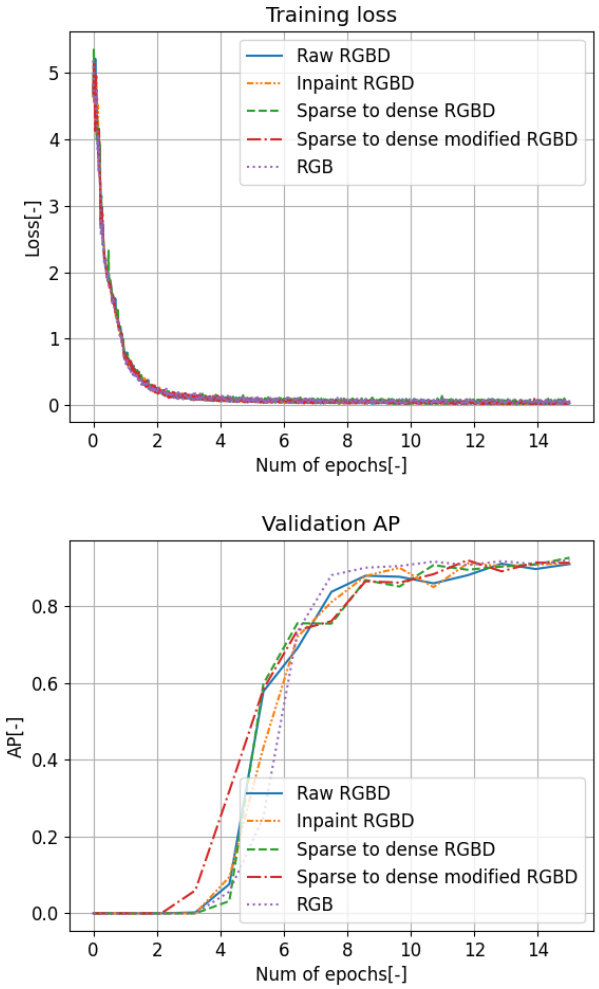


Figure 2.14: Comparison of the training loss and validation AP scores for the different methods during training.

	RGB (baseline)	Inpaint RGBD	Sparse-to-dense RGBD	Sparse-to-dense modified RGBD	Raw RGBD
Weights [number of training epochs]	8	9	11	12	9

Table 2.1: Chosen weights.

Chapter 3

Results

After the training was completed, the different metrics on the validation dataset were compared. These metrics include:

- precision,
- recall,
- Intersection over Union (IoU),
- Average Precision (AP).

These metrics are given by the following formulas:

$$\begin{aligned} precision &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}, \\ recall &= \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}, \\ IoU &= \frac{\text{Area of overlap of two bounding boxes}}{\text{Area of union of two bounding boxes}}. \end{aligned} \tag{3.1}$$

where:

- N is number of classes.

AP is calculated by plotting Precision against Recall obtained in eq. (3.1) and calculating the area under the resulted curve.

Results	RGB (baseline)	Inpaint RGBD	Sparse-to-dense RGBD	Sparse-to-dense modified RGBD	Raw RGBD
AP	0.41	0.46	0.36	0.43	**0.48**
precision	0.79	0.89	**0.92**	0.60	0.59
recall	0.48	0.48	0.37	0.48	**0.53**
IoU	0.83	**0.84**	**0.84**	0.83	0.79

Table 3.1: Comparison of different metrics on the testing dataset for the evaluated detection methods.

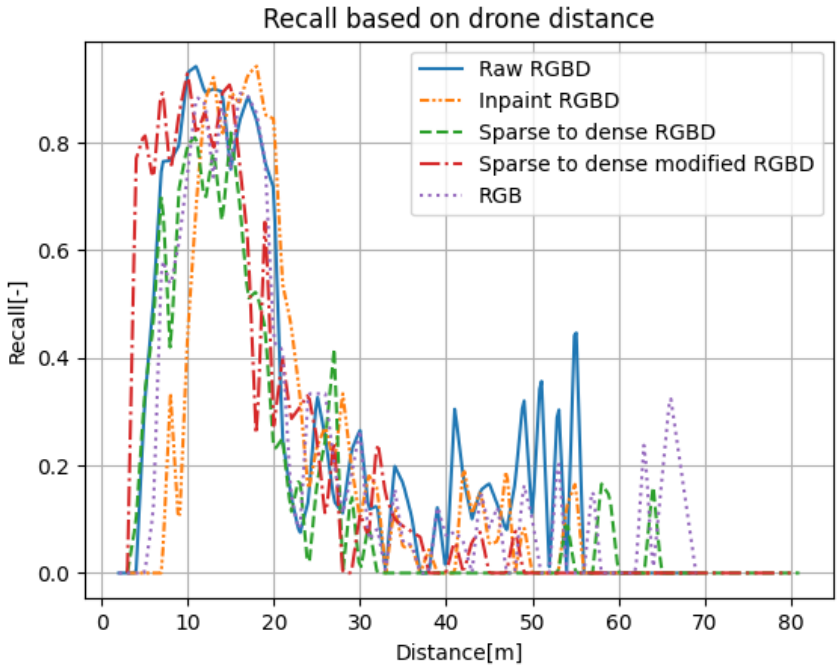


Figure 3.1: Recall over distance of the drone.

After the weights were chosen, the network was validated on the test dataset. The confidence threshold was chosen to be 0.2 and IoU threshold was set to 0.5. These settings showed best overall results for all the methods. From the results presented in Table 3.1, Raw RGBD offers the best improvement in terms of AP by 7% when compared to the baseline RGB-only method. Every method except Sparse-to-dense RGBD offers some improvement in terms of AP over the baseline. In terms of precision, Sparse-to-dense RGBD and Inpaint RGBD offer an improvement over RGB-only by 13% and 10% respectively. Other methods namely Raw RGBD and modified Sparse-to-dense RGBD offer a decrease of 20% and 19% respectively. When it comes to the recall, not a big improvement is made. The best method is Raw RGBD with an increase of 5% in comparison to RGB-only. Sparse-to-dense RGBD suffers a decrease of 11% in comparison to RGB-only, while the remaining methods

Speed	RGB (baseline)	Inpaint RGBD	Sparse-to-dense RGBD	Sparse-to-dense modified RGBD	Raw RGBD
Duration [ms]	**39**	542	67	76	61

Table 3.2: Average processing duration of all methods.

are unchanged. This can be observed in Figure 3.1 where all methods perform the best in the range from 4 to 19 meters reaching a maximum recall of around 0.95. Modified Sparse-to-dense RGBD offers the highest recall in around 4 meters but starts to fall after 18 meters. Raw RGBD shows a few spikes in the range from around 48 to 55 meters, the highest reaching 0.5 recall. RGB-only shows a spike at around 67 meters with a recall of around 0.7. In terms of the IoU, all methods perform very similarly with best-performing methods Inpaint RGBD and Sparse-to-dense RGBD showing improvement of 1%.

Table 3.2 presents the total processing durations of the evaluated methods from acquiring the image to providing the detection output. The times shown represent a mean from 1251 test samples. From the results, the 4th channel depth input almost doubles the inference time of YOLOv3, while Sparse-to-dense network alone performs quite fast. The longest time for inference belongs to the Inpaint RGBD method with 542 milliseconds.

Overall Raw RGBD provides an increase in AP and recall in comparison to RGB-only method but suffers a decrease in precision. The inference time is longer, which may prove to be a problem in real-life applications.

Inpaint RGBD produces overall better results in every metric compared to RGB-only but suffers heavily in long inference times, making it inappropriate for real-life applications where fast tracking is required.

Sparse-to-dense RGBD offers a decrease in AP and recall but provides overall the best precision out of any method tested. Sparse-to-dense network output time is slowing down the inference speed by only a small amount.

Sparse-to-dense modified RGBD provides little improvement in terms of AP and suffers a decrease in precision. Out of any other method, it provides the highest recall in close detection distances. The modifying algorithm is responsible for a very little slowdown compared to Sparse-to-dense RGBD.

Chapter 4

Conclusion

In this thesis, four different methods of fusion of RGB data with LiDAR data for the purpose of drone detection were proposed and compared to an RGB-only method. Virtual environments were used for the generation of the dataset. A detection convolutional neural network was trained from scratch and tested using various metrics that were compared with the RGB-only baseline CNN. The advantage of these methods is that they can be easily implemented on drones already carrying these sensors, or on drones already carrying a LiDAR sensor as RGB camera is lightweight and cheap. Since these methods provide detection of drones, there is no need to rely on pre-existing sensor infrastructure as in the case of absolute localization methods. These methods also do not rely on markers placed on target drones, so they can be used in a wider variety of scenarios, where f.e. the target is an uncooperative drone.

The dataset used in this method was generated using a variety of virtual environments as described in section 2.6. The engine used for the creation of the dataset proves to be useful when trying to simulate real-life situations and environments. The main advantage is that environments can be created based on specific drone scenarios, without the need to physically simulate such scenarios. Expanding on the number of different environments and the number of different drone models could be an improvement for the overall generalisation and size of the dataset in future works.

The main detection convolutional neural network was described in section 2.2. The network was modified from its original implementation to process RGBD data and to detect smaller objects as flying drones that are farther away are only a few pixels in size. A more specific custom-designed architecture could prove to bring an overall improvement on the problem and should be further studied and compared.

Overall, the results show that no single approach excels over others in all tested metrics. In situations where high precision is required, the best method is the one utilizing the Sparse-to-dense preprocessing CNN. It suffers no significant slowdowns in terms of inference time and provides the best precision out of all tested methods. A modified method from section 2.1 proves to provide a higher recall compared to other methods in terms of close distance detection. Compared to other methods, RGB-only method provides the most balanced results in all the metrics and the best result in terms of inference speed, making it the best default method for drone detection out of all the tested methods. When it comes to more specific scenarios the fusion of LiDAR and RGB data will excel more.

This thesis aimed to compare different approaches of fusion of LiDAR and RGB data and compare them with RGB-only method. This assignment was satisfied and proved that some approaches of LiDAR and RGB fusion can have better results in some specific scenarios. The fusion of LiDAR and RGB has got potential for future works and with larger dataset methods tested in this thesis could overall be better then RGB methods.

Bibliography

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

[2] T. Krajník, M. Nitsche, J. Faigl, P. Vanundefinedk, M. Saska, L. Přeučil, T. Duckett, and M. Mejail, "A practical multirobot localization system," *J. Intell. Robotics Syst.*, vol. 76, p. 539–562, dec 2014.

[3] A. Tahir, J. Böling, M.-H. Haghbayan, H. T. Toivonen, and J. Plosila, "Swarms of unmanned aerial vehicles — a survey," *Journal of Industrial Information Integration*, vol. 16, p. 100106, 2019.

[4] S. Lee, D. Har, and D. Kum, "Drone-assisted disaster management: Finding victims via infrared camera and LiDAR sensor fusion," in *2016 3rd Asia-Pacific World Congress on Computer Science and Engineering (APWC on CSE)*, pp. 84–89, 2016.

[5] H. Kramer, S. Mücher, and H. van der Hagen, "Hotspot vegetation structure and terrain monitoring of dutch coastal dunes with LiDAR and optical camera's mounted on drones," in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pp. 739–742, 2021.

[6] D. R. Alves de Almeida, E. Broadbent, A. M. Almeyda Zambrano, M. P. Ferreira, and P. H. Santin Brancalion, "Fusion of LiDAR and hyperspectral data from drones for ecological questions: The gatoreye atlantic forest restoration case study," in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pp. 714–715, 2021.

[7] X. Shi, C. Yang, W. Xie, C. Liang, Z. Shi, and J. Chen, "Anti-drone system with multiple surveillance technologies: Architecture, implementation, and challenges," *IEEE Communications Magazine*, vol. 56, no. 4, pp. 68–74, 2018.

[8] I. Bisio, C. Garibotto, F. Lavagetto, A. Sciarrone, and S. Zappatore, "Unauthorized amateur UAV detection based on WiFi statistical fingerprint analysis," *IEEE Communications Magazine*, vol. 56, pp. 106–111, 04 2018.

[9] S. Holter, A. Tsoukalas, N. Evangeliou, N. Giakoumidis, and A. Tzes, "Relative visual localization for unmanned aerial systems," *arXiv preprint arXiv:2003.01954*, 2020.

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

[11] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[12] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[13] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[14] 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, 2016.

[15] W. Ali, S. Abdelkarim, M. Zidan, M. Zahran, and A. El Sallab, "YOLO3D: End-to-end real-time 3d oriented object bounding box detection from lidar point cloud," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, pp. 0–0, 2018.

[16] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

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

[18] J. Zhang, M. S. Ramanagopal, R. Vasudevan, and M. Johnson-Roberson, "LiStereo: Generate dense depth maps from LiDAR and stereo imagery," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7829–7836, IEEE, 2020.

[19] F. Ma and S. Karaman, "Sparse-to-dense: Depth prediction from sparse depth samples and a single image," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4796–4803, IEEE, 2018.

[20] A. C. Telea, "An image inpainting technique based on the fast marching method," *Journal of Graphics Tools*, vol. 9, pp. 23 – 34, 2004.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[22] P. Martinez, *Vision-based Algorithms for UAV Mimicking Control System*. PhD thesis, 11 2017.