Universiteit Leiden
The Netherlands

Computer Science

YOLO-based Obstacle Avoidance for Drones

Christof W. Corsel

Supervisors:
Erwin M. Bakker
Michael S. Lew

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

August 25, 2020

Abstract

One basic requirement for autonomous operation of UAVs is obstacle avoidance. Because of the strict power and weight constraints of UAVs, traditional sensors like LiDAR and ultrasound are often omitted in favour of a visual camera sensor. This thesis presents a method for obstacle avoidance utilising only these common, versatile sensor systems by using a deep learning (DL) approach, which aims to show the validity of using Convolutional Neural Networks (CNN) object detectors in the obstacle avoidance pipeline. The YOLOv4 object detector is used to generate bounding boxes around possible obstacles. These located obstacles are tracked and their sizes are compared in subsequent frames. The size ratio values gathered determine if a detected obstacle must be avoided. The proposed method is evaluated and compared against a previous non-DL SIFT-based method in various real-life obstacle avoidance scenarios. Here, results show an avoidance accuracy of 88.7% compared to 63.3% of the previous work. The proposed method performs on par with the previous work in a simple, single object scenario where accuracy reached 90%. Furthermore, results from more realistic scenarios including multiple objects and textured backgrounds show significant improvements in accuracy from 71.0% to 90.0% and from 18.3% to 83.3% respectively. Lastly, the validity of using obstacle class labels gathered from the YOLOv4 network in obstacle avoidance was investigated, showing an increase in avoidance accuracy from 40.0% to 90.0% in the selected scenario.

Contents

Chapter 1

Introduction

Unmanned Aerial Vehicles (UAVs) have made their way from the military into the hand of companies and consumers. Thanks to their low cost and high manoeuvrability, they have become popular tools in many fields, such as agriculture [1], delivery service [2], surveillance [3], film-making [4] and photography [5].

For continuous operation and the successful execution of designated tasks, the UAV must not collide with obstacles in the environment. A human pilot is often used to prevent this from happening, requiring much training and focus of the operator. Autonomous operation of the UAV could alleviate the need for intensive human supervision, saving time and money. For such autonomous systems, obstacle avoidance is one of the basic underlying functions. Many existing solutions for obstacle avoidance rely on special sensors such as laser (LiDAR) [6, 7], ultrasound [8], depth cameras [9] or stereo cameras [10]. These sensors, however, suffer from lower accuracy in areas with unpredictable lighting or reflective surfaces. Furthermore, with the recent trend of shrinking UAV size to reduce cost and power requirements, such sensors are often omitted in favour of more versatile visual camera sensors. These sensors can be used to mimic visual obstacle avoidance methods seen in nature. For example, pigeons are found to employ reactive path planning when encountering cluttered environments [11]. A monocular camera, which is generally included in a UAV, can be used to perform such obstacle avoidance [12]. However, using only this single sensor without the ability to gather depth information easily increases the difficulty of detecting obstacles at high accuracy and speed.

Deep learning (DL) has shown excellent results in areas such as object or speech recognition as well as robotics tasks like positioning and control [13]. In particular, Convolutional Neural Networks

(CNN) have been successful in the computer vision field, which includes tasks like object detection and localisation. Their ability to process visual data from cameras make them a natural candidate to be considered in the field of UAV obstacle avoidance, especially if only a monocular camera is available. Previous applications of DL to obstacle avoidance learn manoeuvre commands from a custom made dataset [14], or predict depth information to generate a depth map of the environment to find a safe passage [15].

In this thesis, an obstacle avoidance system based on the YOLOv4 [16] object detector is proposed, used in a size expansion algorithm based on the paper by Al-Kaff et al. [17]. It replaces the non-DL based method using SIFT proposed in this paper by a high speed CNN object detection network trained on the MS COCO dataset [18]. Using the object localisation ability of this network, the size changes of frontal objects in a live video feed can be detected by comparing the size of the bounding boxes of those objects in consecutive frames. The size ratio is sent to a control system which decides movement commands for the UAV. By combining a state-of-the-art object detector and the size expansion algorithm, objects can be detected and avoided using a single monocular camera in real time. Furthermore, the classification capability of object detection networks can be used in the avoidance algorithm. This is done by utilising class labels obtained by the detector to improve avoidance decisions based on the object type. The proposed method can be easily integrated in UAV systems using an object detector for tasks like real-time object detection [19].

This thesis aims to improve the accuracy of obstacle avoidance for drones using only a single camera sensor. The proposed method of obstacle avoidance is compared against the original size expansion algorithm using the SIFT algorithm by performing various real-life experiments. These experiments are designed to test the accuracy of obstacle avoidance both methods in various environments and conditions.

The remainder of this thesis is organised as follows: Chapter 2 gives an overview of related work concerning obstacle avoidance for drones, ranging from methods using sensors to deep learning based methods. In Chapter 3, the size expansion algorithm as described in [17] and some fundamentals in object detection are discussed. Chapter 4 explains the implementation of the proposed method, discussing both the object detection and obstacle avoidance modules that are used. In Chapter 5, the validity of the implementation is tested in several scenarios by comparing it against the original method by Al-Kaff et al. [17]. Chapter 6 analyses the results of the experiments. Finally, in Chapter 7, a conclusion to this thesis is given as well as suggestions for future research.

Chapter 2

Related Work

2.1 Obstacle Avoidance Systems

Many different ways of handling collision avoidance are described in literature. In Simultaneous Localisation and Mapping (SLAM) [20] techniques, a map of the surroundings is generated and updated whilst using this map to find the robot's location in the environment. This allows for the detection of obstacles and traversable spaces. The authors of [9] used an RGB-Depth camera to incrementally build a map of the environment and find a collision-free path. In [6] a Light Detection And Ranging (LIDAR) sensor is attached to a small quadcopter to generate occupancy grids which are used to track walls and avoid obstacles. However, using these types of sensors in mini- to medium-sized drones is not always viable due to the higher cost and power consumption. To combat this issue, the authors of [8] used a combination of low-cost ultrasonic and infrared sensors to enable autonomous flight. This results in a less accurate but much more energy and computationally efficient solution.

Another approach to obstacle avoidance is to use depth information generated by stereo vision [21] techniques. Here two cameras looking at the same object are used to calculate depth based on the disparity of the pictures of both cameras. This depth information can then be used to enable 3D world navigation. As demonstrated in [22], stereo vision sensors can also be combined with global position system (GPS) to plan a flight path. Here, safe areas for the UAV to fly through are found by analysing depth images obtained by a Semi-Global Matching (SGM) algorithm. In [10], a stereo-camera is used on a tiny quadcopter to enable velocity and depth measurements. The algorithm is able to run on an embedded microprocessor and uses edge distributions to calculate optical flow and stereo disparity.

As monocular cameras are found in virtually all modern UAVs it is only logical to include these sensors in obstacle avoidance, possibly eliminating the need to add additional sensors. The authors of [23] used Extended Kalman Filter (EKF) to estimate the location of features in an image. These feature points are used to create a terrain map which can be used in obstacle avoidance. In [24], Harris Corner Detection and Scale Invariant Feature Transformation (SIFT) were used to compare the size of an incoming object between frames so an obstacle can be detected. The method proposed in [17] also uses a size expansion algorithm but includes an avoidance system which was tested by the authors in an indoor environment.

With the rise of deep learning (DL) in the field of computer vision, it has been found to be a promising method to apply to the obstacle avoidance problem [13]. The authors of [25] used self-supervised Convolutional Neural Networks (CNN) trained on a custom database to predict the distance-to-collision. The distance to the closest object in multiple directions is estimated to change the direction of the drone so collisions can be avoided. The implementation was found to have high accuracy in a real-world indoor environment.

In [14], the DenseNet-161 CNN was trained on a custom-made dataset containing imagery of an indoor corridor environment. The model classifies images gathered from a monocular, front-facing camera on a UAV. The output of the model returns the probability of different class labels which represent movement commands for the drone. Experiments showed an average accuracy of 77.3% but the authors state that this could be improved by adding more diverse training images into the dataset.

[26] combines two CNN networks to predict a 3D trajectory through different environments. First, depth and surface normals are predicted to give the UAV 3D obstacle positions which are then used to predict paths more precisely by a second CNN. Using the NYUv2 public dataset, containing 260k images with their corresponding depth information, the authors showed an improvement in accuracy and robustness from 42.68% to 50.05% compared to a direct prediction approach.

The authors of [15] used a CNN trained on the public NYUv2 database to predict depth of single images generated by a quadrotor. The authors further finetuned the network by adding their own samples. The depth information predicted by the CNN is fed to a control system which steers the UAV away from obstacles in real time. Testing was done in both a simulated environment and the real world.

2.2 Object Detection Networks

Modern object detectors often use CNNs to classify and localise objects in a frame. Here, two-stage and one-stage detectors are differentiated. Popular two-staged detectors are Fast R-CNN [27] and Faster R-CNN. Such networks deliver high detection accuracy but at a high computational cost. One-stage detectors like Single Shot Detector (SSD) [28] or the You Only Look Once (YOLO) [16] family generally do not reach the same accuracy as two-staged detectors, but are able to run significantly faster. Some of the relevant networks are further discussed in Section 3.2.2.

Chapter 3

Background

3.1 Size Expansion

Detecting objects using a single monocular camera is considered challenging because of the absence of depth information. Size expansion enables the detection of obstacles in this situation by comparing the size of an object in consecutive frames. An object can be determined to be an obstacle by estimating the size and position of the object within the frame. Calculation of the differential size ratio value and setting a threshold enables the drone to know when action needs to be taken to avoid collision, as can be seen in Figure 3.1. To estimate the size of an object, a bounding box is generated in the frame, encapsulating the possible object. The size of the object is then determined by the amount of pixels the bounding box contains in the frame.

3.2 Object Localisation and Detection

Because bounding boxes are used as a means to estimate object sizes, they need to be drawn accurately around possible objects in the image. The task of finding objects in an image is called object localisation. It provides an extension to image classification, which tries to state what object can be seen in an image, to also specifying where the object classified is located in the image. Here we are limited by detecting a single, main object in the image. The object detection problem deals with locating and classifying multiple objects in an image, by generating bounding boxes and labels for all the objects it can detect (Figure 3.2).

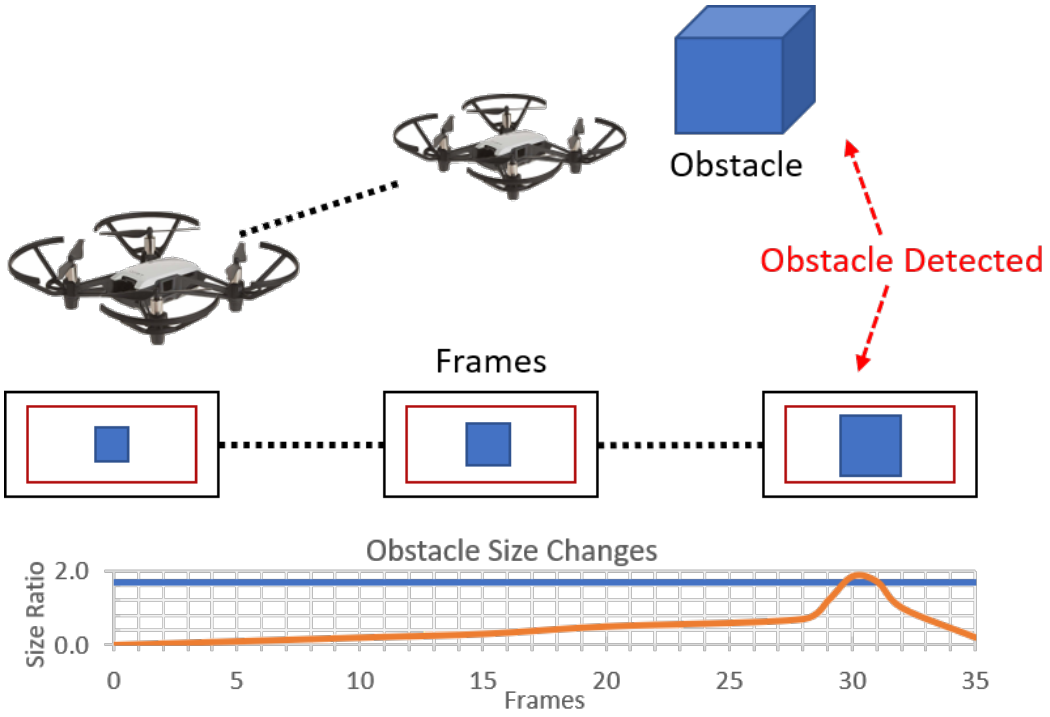


Figure 3.1: Size expansion concept

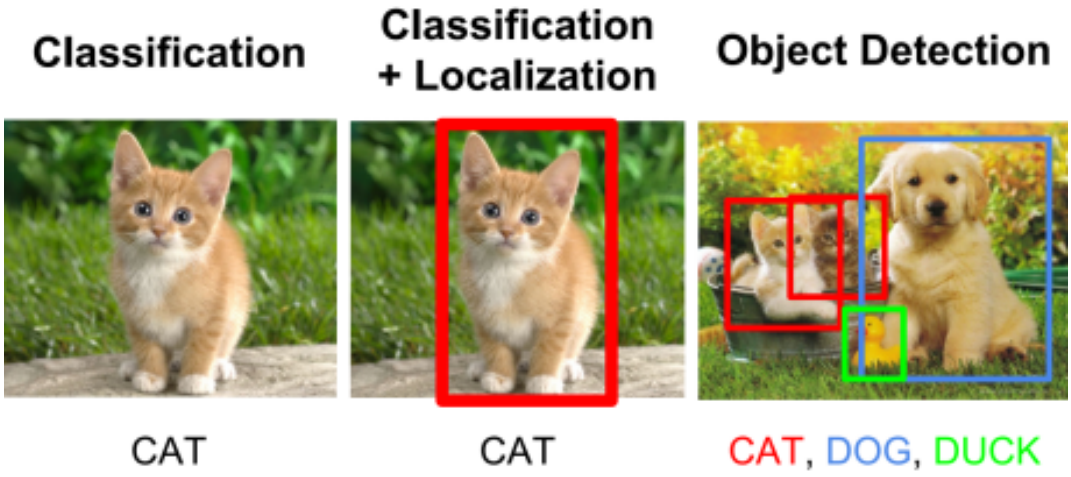


Figure 3.2: Object classification, localisation and detection. Taken from [29]

3.2.1 Point Feature Matching

One way to detect an object in an image is through feature matching. This is done by detecting important features in both the input image and an example image of the object that needs to be detected. Important features are often edges or corners, as these depict unique characteristic of the object in question. These features can be found using algorithms like Harris Corner Detection [30], Scale-Invariant Feature Transform (SIFT) [31] and Oriented FAST and Rotated BRIEF (ORB) [32]. The features found in the input image are then matched with those in the example image using a Brute-Force or Fast Library for Approximate Nearest Neighbours (FLANN) based matcher [33]. The matches found will show if and where an object is located in the input image.

In [17] SIFT features and a Brute-Force Matcher are used to detect objects. SIFT offers the detection of features which are robust to rotation and scale. The object detection algorithm described in [17] consists of five stages:

1. **Scale-space extrema detection:** A difference-of-Gaussian function is used to find potential scale and orientation invariant interest areas.
2. **Keypoint localisation:** The results from the previous step are refined, eliminating low-contrast and edge keypoints based on threshold values.
3. **Orientation assignment:** Each keypoint is assigned an orientation to achieve invariance to image rotation.
4. **Keypoint description:** Keypoint descriptors are created to create robust, distinct features based on the neighbourhood of the keypoint. This generates multiple keypoints in an image, as can be seen in Figure 3.3.
5. **Keypoint matching:** The detected features are used for object recognition by matching the features of a training image with the ones detected in the frame (Figure 3.4).

Matched keypoints are used calculate the size difference of an object in subsequent frames. This is described further in Section 4.2.

Figure 3.3: Detected SIFT features. Taken from [34]

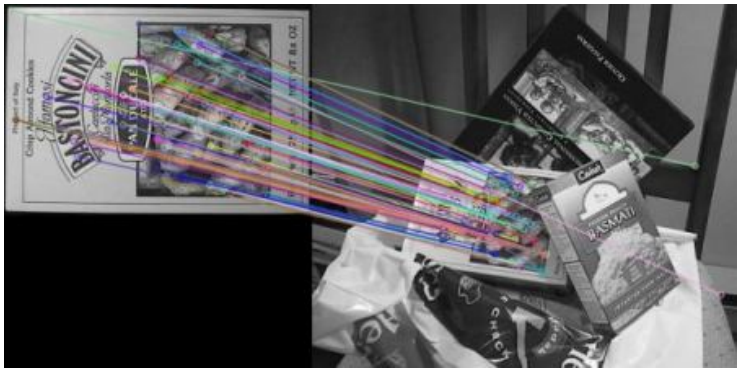


Figure 3.4: Matched keypoints from a Brute-Force matcher. Taken from [34]

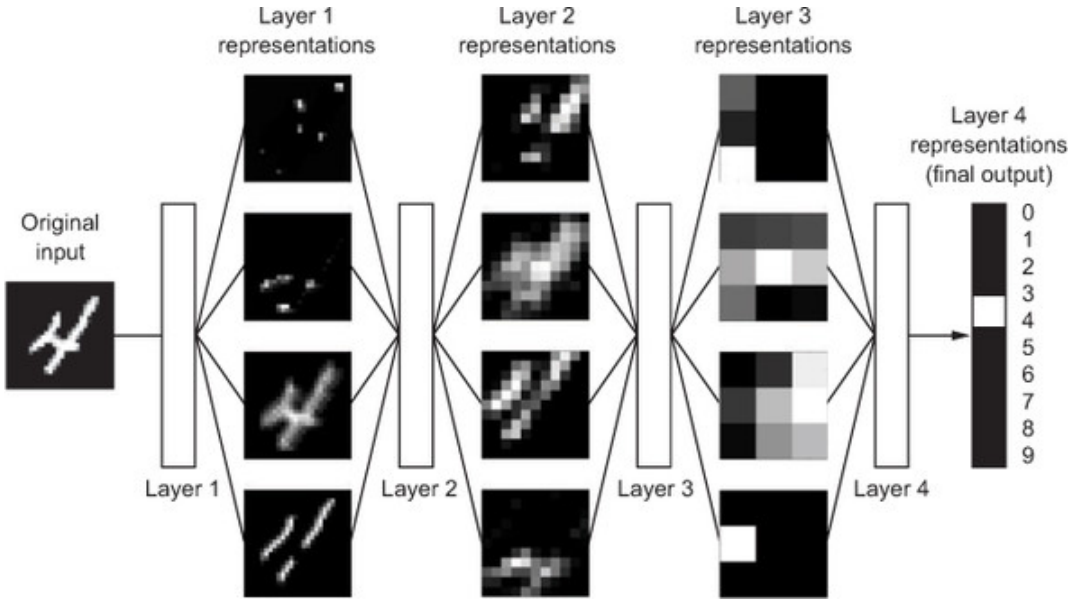


Figure 3.5: Deep representations learned by a digit-classification model. Taken from [35]

3.2.2 Deep Learning

Another popular method for localising and detecting obstacles is using deep learning. These methods are usually based on convolutional neural networks (CNN). Such networks are able to detect local patterns in the input image in the first convolution layer, combining these into larger patterns in subsequent layers, allowing these networks to efficiently learn complex structures. This can be seen in Figure 3.5, where an example CNN can classify handwritten digits by combining local features detected in the first layer, like edges and corners, into larger, more abstract concepts in the latter layers, finally giving a classification label to the image.

Object detectors can be separated in multiple categories where two-stage detection and single-stage

detection are the most common. Two-stage detectors, like Faster R-CNN [36], use a region proposal network to find interesting regions possibly containing objects in the image first. These regions are then used by the second stage to apply classification or bounding box regressions, as can be seen in Figure 3.6. Such methods are found to be the most accurate CNN object detectors, but often require long processing times.

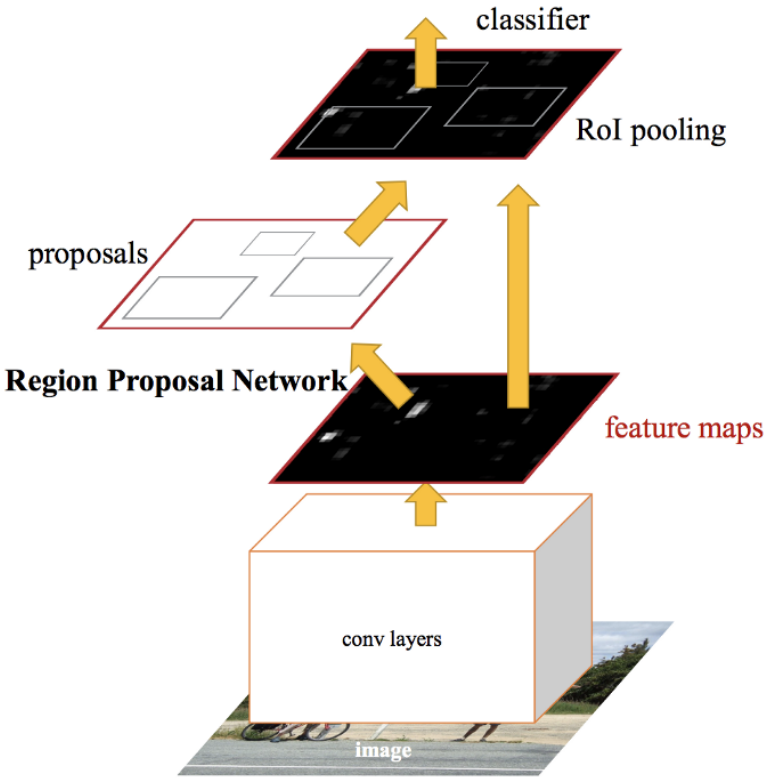


Figure 3.6: Faster R-CNN. Taken from [36]

Single-stage detectors like YOLO (You Only Look Once) [37] or SSD (Single Shot MultiBox Detector) [28] only utilise a single network, i.e. stage. Here, object detection is treated as a regression problem where class categories and bounding box coordinates are learned from an input image. Only using a single network increases the speed of the detection significantly, but hurts detection accuracy. Figure 3.8 shows the general stages for both one- and two-staged networks.

Performance measures

To measure the accuracy of object detectors, Average Precision (AP) is often used. Object detection challenges like COCO [18] use this measure to compare the performance of models on their datasets. The AP is calculated by using the precision and recall values, where precision refers to the percentage

of correct positive predictions and recall measures what percentage of positives are found, as seen in Equations 3.1 and 3.2. AP is defined as the area under the plotted precision-recall curve, using Equation 3.3.

$$p = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{3.1}$$

$$r = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{3.2}$$

$$AP = \int_0^1 p(r)dr \tag{3.3}$$

When measuring accuracy for object localisation, Intersection over Union (IoU), as shown in Equation 3.4, is often used as a measure of how well the predicted bounding box represents the ground truth. A threshold is set to differentiate a generated bounding box between a true and false positive.

$$IoU = \frac{area\ of\ overlap}{area\ of\ union} \tag{3.4}$$

YOLO

The You Only Look Once (YOLO) family of one-staged object detectors promises high performance and fast processing times. YOLO [37] is a single shot detector which divides the image into a grid, where each cell predicts bounding boxes, confidence scores for these boxes, and a class probability of the cell. These properties are processed during test time to create the final bounding boxes and labels in a single network evaluation, resulting in a higher efficiency compared to two-stage detectors. However, it makes more localisation errors and is less accurate than two-staged detectors.

YOLOv2 [38] was proposed to deal with these issues by applying batch normalisation to the convolutional layers, and implementing multi-scale training, fine-grained features, and anchor boxes. This dramatically increased accuracy compared to the YOLO but is still lacking good accuracy for smaller and densely packed objects.

The next iteration of the model, YOLOv3 [39], comprised changes to further improve accuracy. It introduced multi-scale predictions and a new feature extractor network which greatly improved

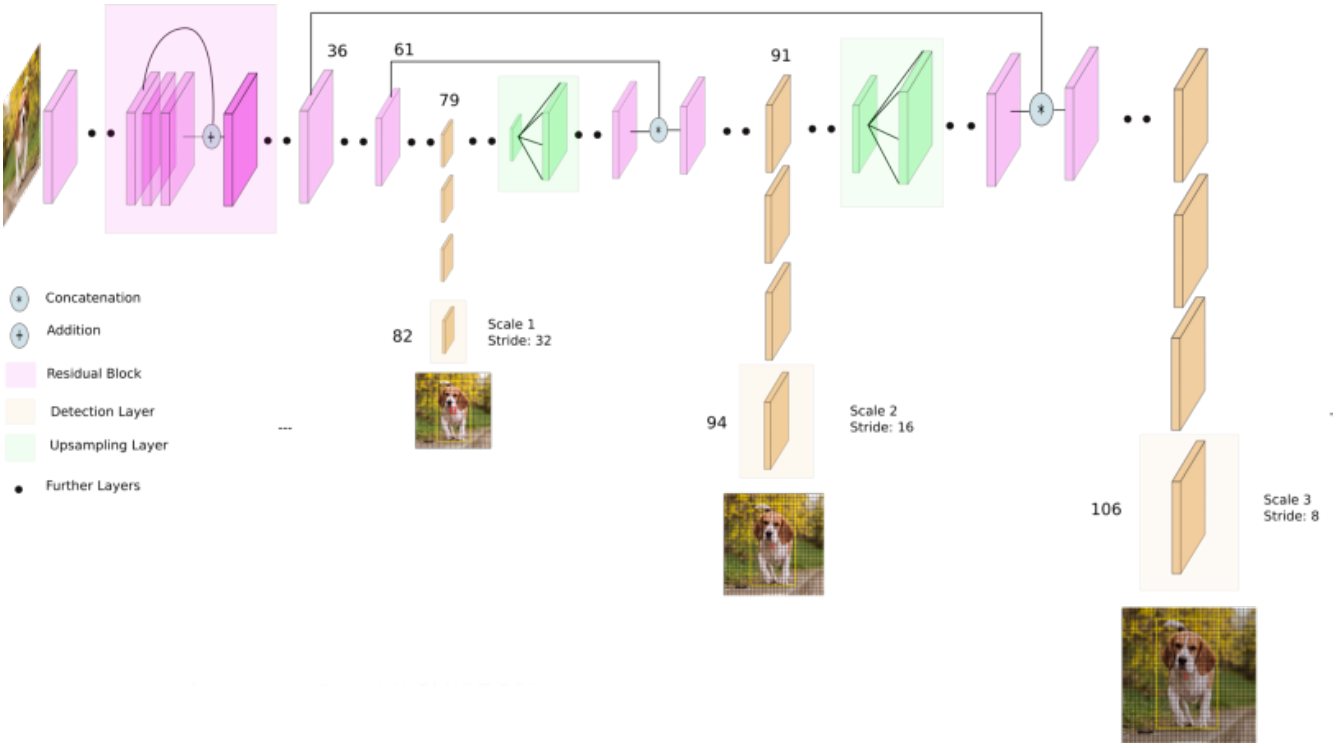


Figure 3.7: YOLOv3 architecture. Taken from [40]. Single numbers denote the layer number

the detection of small objects. However, the AP still lags behind teh AP obtained by some of the heavier two-staged networks. It makes up for this by having a much higher performance, i.e. it is able to process an image faster than state-of-the-art two-staged methods on the same hardware. The network architecture of YOLOv3 can be seen in figure 3.7. The backbone feature extractor network used is Darknet53 which performed on par with other detector backbone networks at a higher max processing speed.

YOLOv4, the latest addition to the family, aims at an optimal trade-off between speed and accuracy for object detection, in addition to being efficient, offering real-time performance. Furthermore, the network was optimised so that it will be easy to train on a single high-end consumer GPU, for example a single Nvidia GeForce RTX 2080 Ti. Bochkovskiy et al. accomplished this by comparing the performance of the network whilst using different methods for each of the detectors stages: backbone, neck and dense prediction, as can be seen in the one-stage portion of Figure 3.8. Furthermore, the authors compared and added more methods from literature to improve accuracy. They distinguished two categories:

- **Bag of Freebies (BoF):** These methods increase the accuracy of the network without impacting inference performance. An often used example of this is data augmentation.

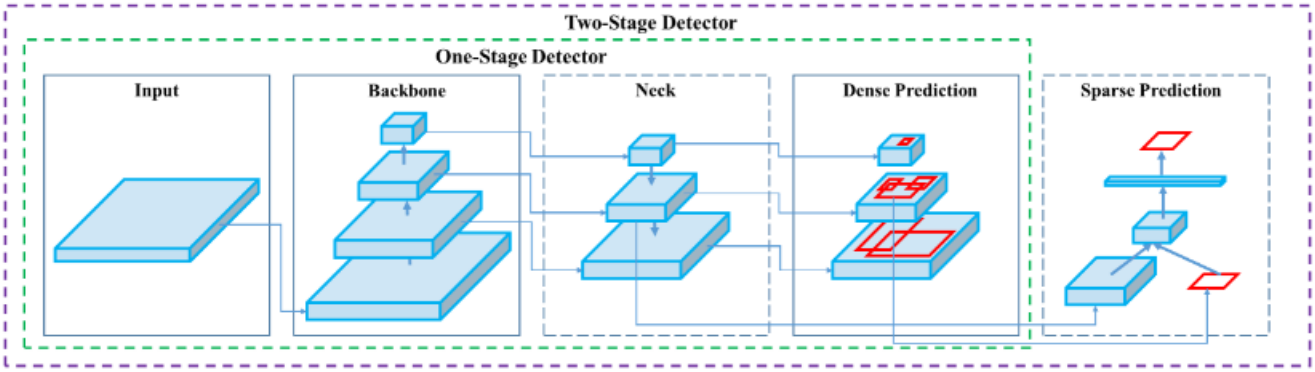


Figure 3.8: One- and two-stage object detector layout. Taken from [16]

- **Bag of Specials (BoS):** These methods increase the accuracy while only impacting inference performance by a small amount.

The authors compared many of these methods to find the optimal combination to be used in YOLOv4. The model was tested and compared to others (Figure 3.9) using the MS COCO dataset [18] and shows comparable accuracy performance to other state-of-the-art models with a much higher FPS. This makes this model suitable for obstacle avoidance, where high accuracy as well as high performance are required.

YOLOv4 Architecture

The final YOLOv4 design includes the optimal backbone and neck found by experimentation to be used by the YOLOv3 head (Figure 3.7). The following components were used:

1. **Backbone:** CSPDarknet53 [41]. Cross-Stage-Partial-connections (CSP) are implemented into the old Darknet53 backbone. This reduces the algorithm's complexity by separating the input into two parts where only one is processed by the network layer (see Figure 3.10).
2. **Neck:** SPP [42], PAN [43]. The neck is used to enrich information gained by the backbone feature extraction network for use in the head object detection network. Spatial Pyramid Pooling (SPP) allows for the detection of objects at multiple scales by splitting feature maps into multiple blocks, creating a spatial pyramid. Path Aggregation Network (PAN) is used to speed up information flow through the network. An example of this can be found in Figure 3.11. The green line represents a shortcut path, allowing fine-grained information to be available to later layers.

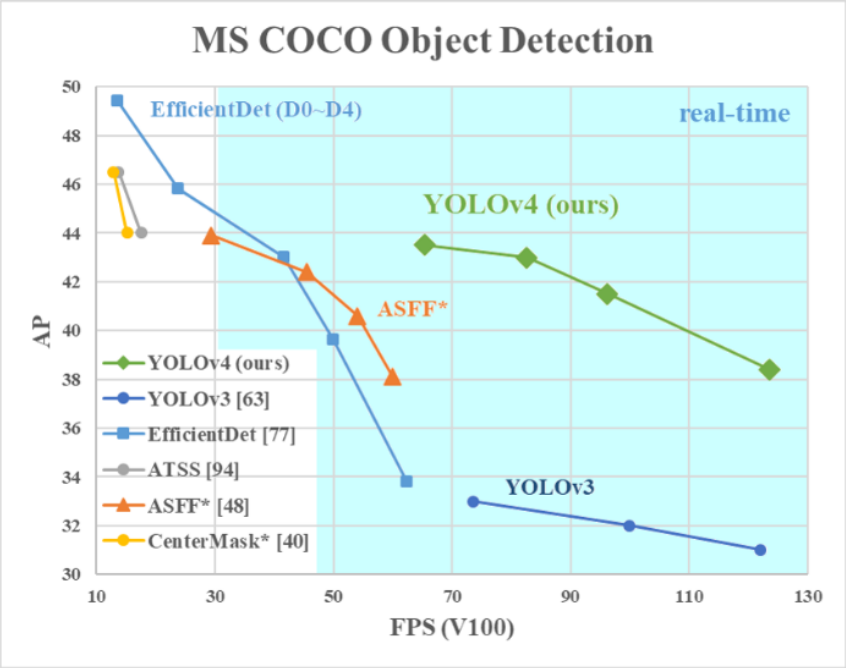


Figure 3.9: Comparison of YOLOv4 and other object detectors. Taken from [16]

3. **Head:** YOLOv3 [39]. The head of the YOLOv3 network is used as the head for the YOLOv4 network.

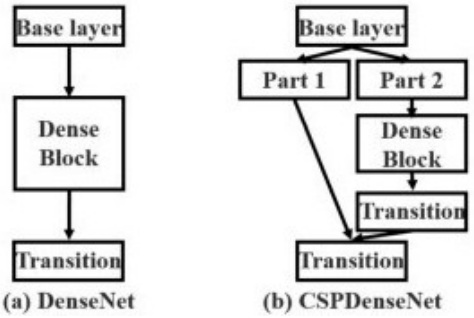


Figure 3.10: Cross-Stage-Partial-connections [41]

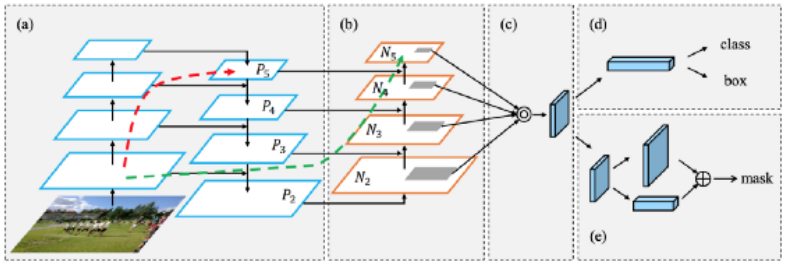


Figure 3.11: Path Aggregation Network (PAN) for an object detector [43]

Chapter 4

Methods

As seen in Chapter 2, there are multiple approaches to obstacle avoidance. This thesis proposes to implement the YOLOv4 CNN object detector into the size expansion algorithm described in the paper by Al-Kaff et al. [17], replacing the non-DL SIFT method for object detection. The problem of obstacle avoidance is divided into two subproblems:

- **Obstacle Detection:** Localising objects and generating bounding boxes to calculate object sizes which are used to specify if an object is regarded as an obstacle or not.
- **Obstacle Avoidance:** Avoiding the obstacle, knowing its location.

4.1 Movement commands

There are many different routes a drone can take to avoid an obstacle. For the sake of simplicity, the drone's avoidance manoeuvres are limited to a Left/Right or Up/Down motion, or a combination of the two. The concrete commands to be sent to the drone can be seen in Equation 4.1 and are used by the obstacle avoidance module to move the drone away from obstacles.

$$MoveRight(Amount)$$
$$MoveLeft(Amount)$$
$$MoveUp(Amount)$$
$$MoveDown(Amount) \tag{4.1}$$

This chapter will continue to describe the methods used and the changes made to the original paper in both subproblems and how these two modules work together.

4.2 SIFT based method

This section describes the method used in [17], which is used in the experiment as a baseline for comparing the performance of the proposed method.

4.2.1 Obstacle Detection

The baseline method [17] uses the SIFT algorithm to detect objects in front of the drone by detecting SIFT features, or keypoints, in two consecutive frames.

Before locating objects, the input image is preprocessed by cropping the image to extract the Region Of Interest (ROI) with a diagonal Field Of View (FOV) of 62°. This is done to limit the object detection area, preventing keypoints outside the collision area of the drone from being matched, and speeding up the computation. Objects outside of this region will not interfere with the drone's flight path and thus are not considered obstacles.

The found SIFT features within the ROI in two frames are matched using a brute-force matcher, returning keypoints present in both frames. These matched keypoints are filtered by excluding a match if the keypoint size in the second frame is not larger than the keypoint size in the first. This eliminates keypoints moving away from the drone.

Next, using the filtered matched keypoints, a single Object of Interest (OOI) is defined by creating

a convex hull containing all the matched keypoints. The convex hull serves as an estimation of the object location and size, which are used to decide if an detected OOI is an obstacle or not. This is done by calculating two size ratio values: the matched keypoint size ratio and the convex hull size ratio, calculated as follows:

$$ratio(mkp) = \frac{1}{N}\sum_{i=1}^{N}\frac{size(mkp_2(i))}{size(mkp_1(i))} \tag{4.2}$$

$$ratio(C) = \frac{size(C_2)}{size(C_1)} \tag{4.3}$$

Here, $mkp_1(i)$ and $mkp_2(i)$ denote a keypoint in the first and second frame respectively. C_1 and C_2 are the convex hulls containing the matched keypoints in both frames. If the respective threshold values of both size ratios are exceeded, the algorithm decides that the object is an obstacle and action will be taken by the obstacle avoidance system discussed in Section 4.2.2. Otherwise, the drone can continue undisrupted and the algorithm will process the next frame. The threshold values are selected based on the experiment described in Section 5.2.1.

4.2.2 Obstacle Avoidance

The obstacle avoidance module is tasked with sending the drone in the optimal direction to avoid the obstacle. This direction is decided by the location of the obstacle in the frame. The best direction is found by calculating four image zones: Up, Down, Left, Right. This is done by measuring the amount of pixels between the four sides of the obstacle and the border of the frame, as seen in Figure 4.1. The sizes of each opposite zone is compared and the largest ones are selected as the direction of the avoidance command. This ensures the manoeuvre moves the drone to the part of the frame that is the emptiest, away from the obstacle.

Avoidance commands

With the avoidance direction of the drone decided, a movement command is sent to the drone. Here, an empty safety area $SafetyArea = 0.30$ m around the obstacles is assumed where the drone is safe to move. This takes into account the dimensions of the drone and any inaccuracies that might arise from the drone's positional sensors. The movement commands that will be sent based

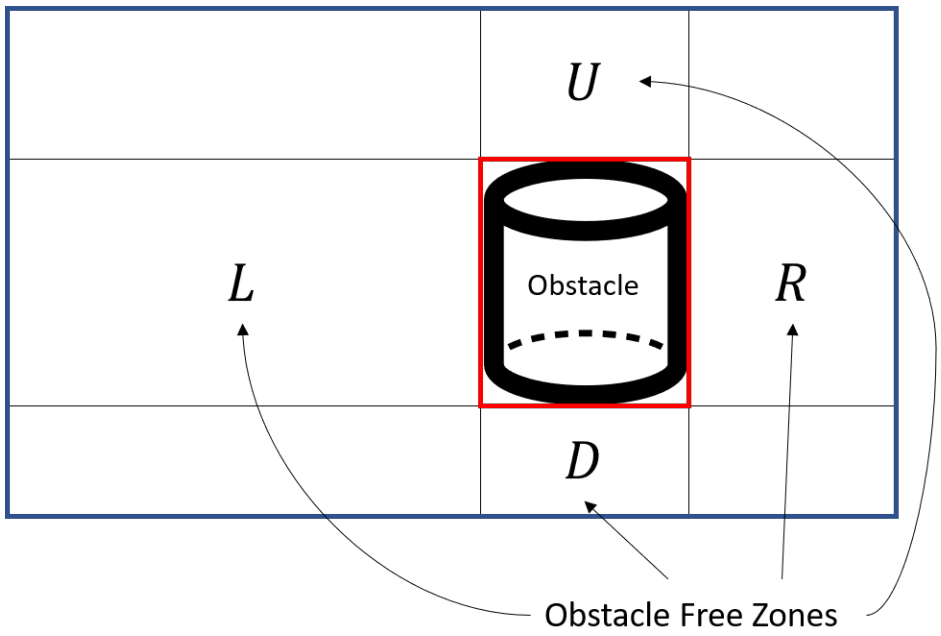


Figure 4.1: Obstacle free image zones

on the direction that was decided by the image zone sizes is shown in Equation 4.1.

In case any of the obstacle's size ratio exceeds 2.0, the object is considered too close or approaching too fast for the drone to avoid, and a hover command is sent to stop the drone.

4.3 YOLO based method

The method proposed in this thesis aims to improve the baseline method by utilising an object detection neural network to localise objects and estimate their sizes. This section presents the neural network used and the changes made to the method described in Section 4.2.

4.3.1 Obstacle Detection

The original method uses the SIFT algorithm to detect objects in front of the drone by matching SIFT features between two consecutive frames. The proposed method aims to improve the object detection by using the CNN YOLOv4 network to generate bounding boxes around detected objects. The sizes of these bounding boxes are used in the size expansion algorithm instead of the convex

hull and keypoint sizes. The CNN used must have fast inference times for the drone to be able to react quickly to incoming objects whilst also having a high localisation accuracy to reliably calculate the sizes of obstacles.

Because of these requirements, YOLOv4 [16] is used as the object detection network for its fast inference performance and high accuracy compared to other models with the same speed. The model was trained on the open-source COCO [18] dataset, containing over 200K labelled images spread across 80 object classes.

Size Expansion

Using the object detector, we can localise objects and estimate their sizes by counting the amount of pixels the bounding box of the object contains. The original method using SIFT feature matching can only detect a single frontal object, as keypoints from the whole ROI are used by the matching algorithm. This means keypoints outside of the main obstacle can be included in its convex hull, making the detector less accurate and skewing size ratio calculations. The proposed CNN-based method can detect multiple objects separately since bounding boxes are drawn only around the part of the image the detected object is located in. These sizes are then used in the size expansion algorithm as depicted in Figure 4.2.

Before locating objects, the input image is, similar to the SIFT method, preprocessed by cropping the image to extract the Region Of Interest (ROI) with a diagonal Field Of View (FOV) of 62°. This ROI is used to exclude detected objects which middle points are outside this region, removing objects that are outside the drone's collision zone from being considered in obstacle detection.

The object detection model generates bounding boxes around objects in the ROI of every frame of the drone's video stream. This operation returns a list of the coordinates of the bounding boxes along with their respective class labels. When the detection model detects any objects, one or more Object Of Interests (OOIs) are chosen which will be tracked as potential obstacles. Being the most likely objects to interfere with the drone's flight path, objects closest to the middle of the frame are selected as OOIs. An OOI is tracked until it moves out of the frame or cannot be detected any more, which is checked by comparing the label and location of each OOI with the detected objects in the next frame.

After choosing and validating the OOIs, the size change ratio of each OOI will be calculated using the size of the OOI in the previous frame and its current size, using Equation4.3.1. Here O denotes

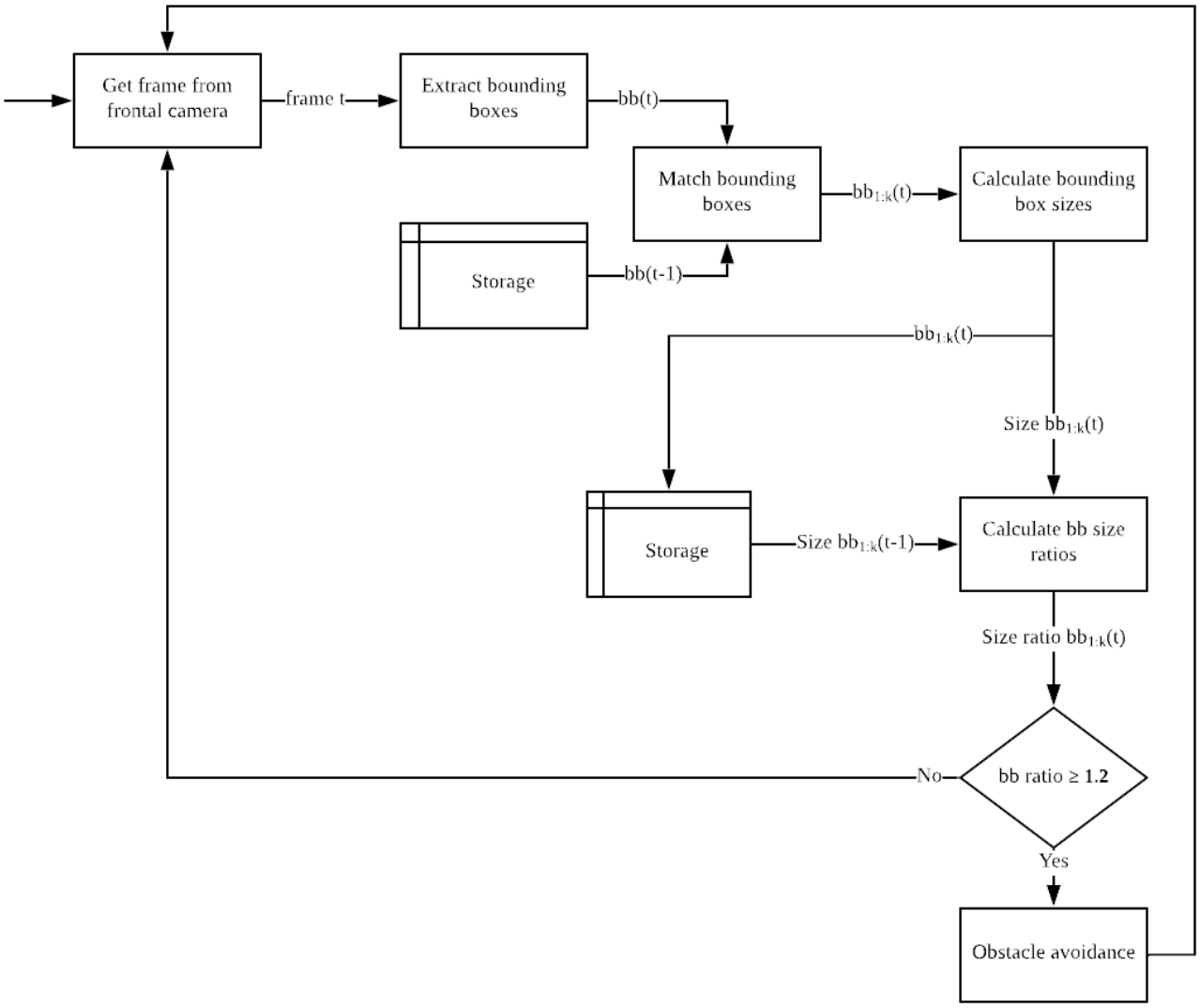


Figure 4.2: Detection system flowchart

the OOI and t the current frame.

$$ratio(O) = \frac{size(O(t))}{size(O(t-1))} \tag{4.4}$$

If the size ratio of any OOI exceeds the threshold value (1.15), as decided by the experiment described in Section 5.2.1, the object is considered an obstacle and must be avoided. The latest bounding box coordinates and class label of all OOIs are then sent to the obstacle avoidance module, where appropriate action will be taken to avoid the object.

4.3.2 Obstacle Avoidance

After one or more obstacles are detected by the object detector, they need to be avoided by the drone to avoid collision. The direction of the avoidance manoeuvre is decided by calculating the same four image zones as seen in Section 4.2.2, where the baseline method defines the image zones as the amount of pixels between the side of the image and the obstacle. The proposed method defines the Up, Down, Left and Right image zones as their respective half of the frame (See Figure 4.3). Because the proposed method can detect multiple obstacles at the same time, each zone is calculated by its size in pixels minus the surface area of the obstacle bounding box(es) located in its area. An example of this can be found in Figure 4.4, where three obstacles are detected. The impact of each OOI is decided by its size ratio. OOIs with high size ratios have a greater impact, as these objects need to be avoided more quickly. This OOI weight value is calculated using Formula 4.5.

$$\text{OOI weight} = \frac{\text{Size Ratio}}{\text{Size Threshold}} \tag{4.5}$$

Finally, thanks to the classification ability of the detection network, the class labels of the obstacles can be used to make smarter choices when choosing a direction. For each class, weights are assigned to the four image zones which are multiplied with the zone size values to give a bias to one or more image zones. A higher weight value for a zone would increase the probability of the direction of that zone to be chosen, and vice versa. This also allows for directions to never be chosen if the weight value is set to 0, which can be beneficial for object classes where some avoidance direction might almost never be viable (e.g. for wall-hung objects, going down has a low chance of being successful).

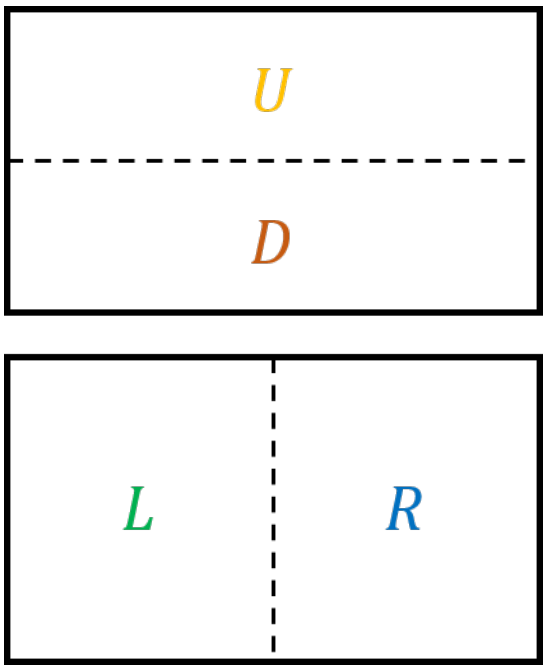


Figure 4.3: The four image zones of the proposed method

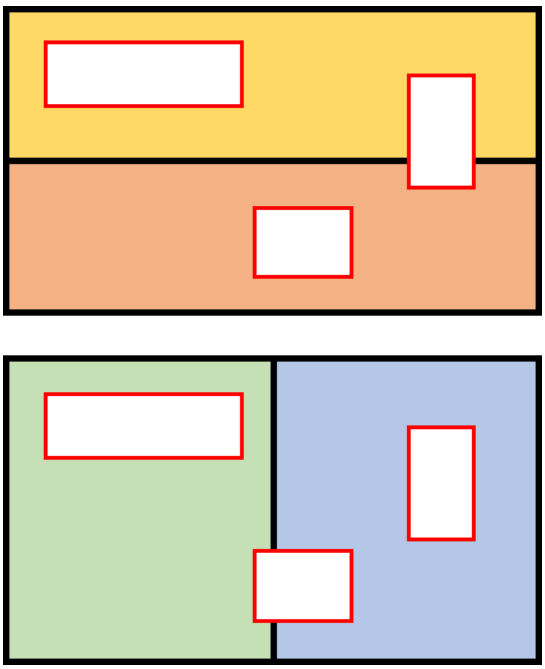


Figure 4.4: Example calculation of image zone sizes. The coloured parts show what area is included in the corresponding zone

After calculating the image zone sizes, the obstacle avoidance manoeuvre is decided, where the drone can execute a left or right, and up or down motion. Which commands are executed depends on the sizes of the zones in both pairs, choosing the smaller ones. These zones contain the most empty space, giving the drone a higher chance to avoid the obstacles. A general overview of the avoidance system is shown in Figure 4.5.

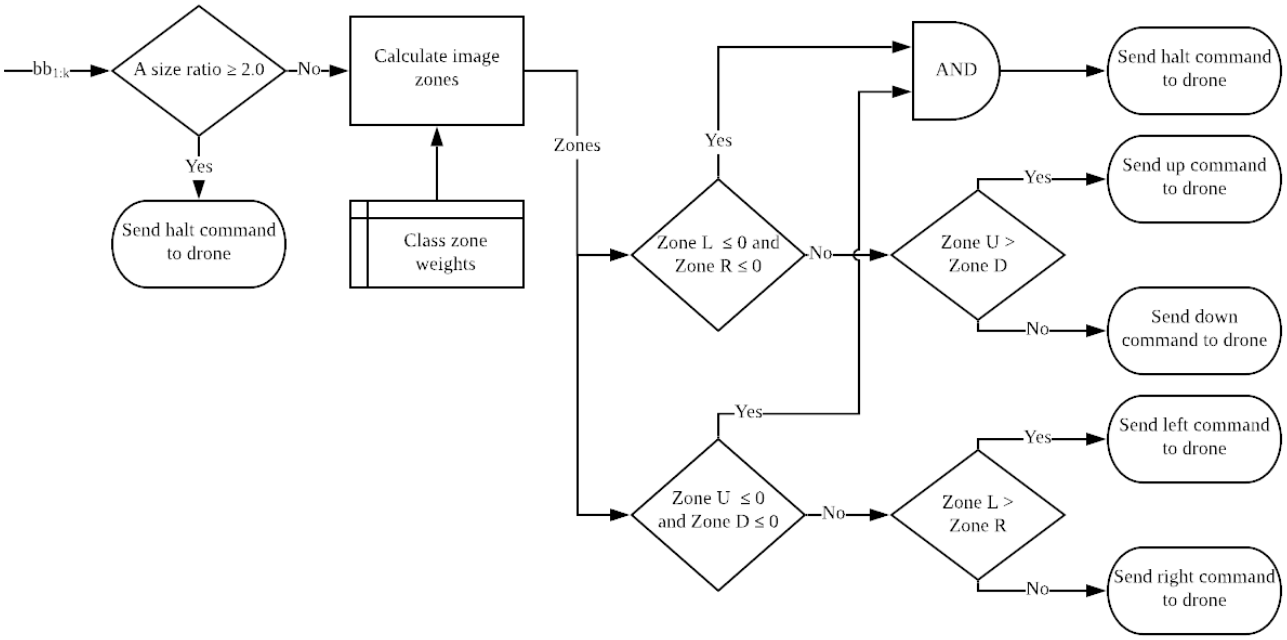


Figure 4.5: Obstacle Avoidance system flowchart

Chapter 5

Experiments

5.1 Hardware Platform and Specifications

To test the proposed obstacle avoidance method the Ryze Tello drone was used. This consumer quadcopter is low cost (∼€100), small, light, and very manoeuvrable (See Figures 5.1, 5.1). The video output from its single camera (1280 × 720 at 30 fps) is used as an input to detect and avoid obstacles.

The drone is connected to a base station PC using WiFi. This station will receive the drone's video stream, process it and return movement commands if needed. The specifications of the base station can be found in Table 5.2. The average performance during flight of the proposed and previous methods are displayed in Table 5.3.

Figure 5.1: Ryze Tello drone

Specification	Value
Weight	80g
Dimensions	98 × 92.5 × 41 mm
Wi-Fi	2.4GHz 802.11n
Camera	5MP
Video	720p at 30 fps
Features	IR Altitude Detector, IMU

Table 5.1: Ryze Tello specifications

Table 5.2: Base station specifications used in experiments

Component	Specification
Processor	Intel Core i7-4770k 4.0 GHz 4 cores 8 threads
RAM Storage	16 GB
Graphics Card	Nvidia GTX 1070 8 GB
Operating System	Ubuntu 20.04 LTS 64-bit

Table 5.3: Performance comparison

Method	Performance (fps)
SIFT	11
YOLO	15

In all experiments, the input stream is processed ten times per second (10 fps) to provide a consistent comparison. The drone flies at a constant 0.5 m/s in all scenarios.

5.2 Scenarios

Multiple experiments were performed to both optimise the proposed method and to compare its performance to the SIFT based method. The details of each experiment are discussed in this section.

5.2.1 Threshold Optimisation

This experiment tries to find the optimal values for the obstacle and keypoint size ratio threshold values as described in Section 4.3.1. These needed to be recalculated from the values used in [17] to account for the difference in drone speed and processing time between frames.

The test is performed on stationary objects on a white background to prevent SIFT features from the background to be included in the obstacle convex hull. The drone approached the obstacle with a constant speed, taking a backwards manoeuvre once the obstacle was detected, meaning the obstacle size or keypoint size ratio exceeds the threshold. The distance flown was recorded using the drone's internal sensors at the moment of detection to calculate the distance to the obstacle. These distance values are measured to determine the optimal threshold values used in later experiments. Measurements are gathered until the drone cannot reliably detect the frontal object due to too high threshold values.

5.2.2 Inside Obstacle Avoidance

This scenario compares the performance of obstacle avoidance of the described method against the method presented in [17] in an indoor environment. The drone travels in a predefined straight line, encountering one or more objects in its path. Information gathered from the drone's internal sensors and camera during the flight was used to evaluate and compare the performance of the implementation. The indoor environment included a white background to make it easier for both methods to detect differently coloured objects. Obstacles were approached at different angles and distances to give a varied comparison.

5.2.3 Outside Obstacle Avoidance

The outside scenario has the same goal as the indoors scenario described above: testing the avoiding performance of the proposed method. The difference to the indoor environment is the inclusion of a noisy background, increasing the difficulty of detecting foreground objects. A person obstacle is used to test the methods robustness against noisy environments.

5.2.4 Smart Car Obstacle Avoidance

This experiment is designed to show the validity of using object classes in a real-life scenario. A single obstacle representing a car will be placed in front of the drone at low altitude. Because of this altitude specification, taking a downward avoidance manoeuvre has a high chance of the drone colliding with the ground. We compare the performance of the proposed method, both with and without the use of class weights as described in Section 4.3.2. The performance of the SIFT method will also be analysed.

Chapter 6

Results

6.1 Threshold optimization

The results in Figure 6.1 show the average distance of detection for various threshold values. A safe detection distance of 1.5 m was chosen, taking into account connection and processing latencies between the obstacle detection and the avoidance manoeuvre taken by the drone. The figure shows the Hull size ratio threshold to be 1.15 and the Keypoint size ratio threshold to be 1.09 at the chosen distance. These thresholds are used in subsequent experiments, where the drone processes the video feed at 10 fps and flies at a constant 0.5 m/s.

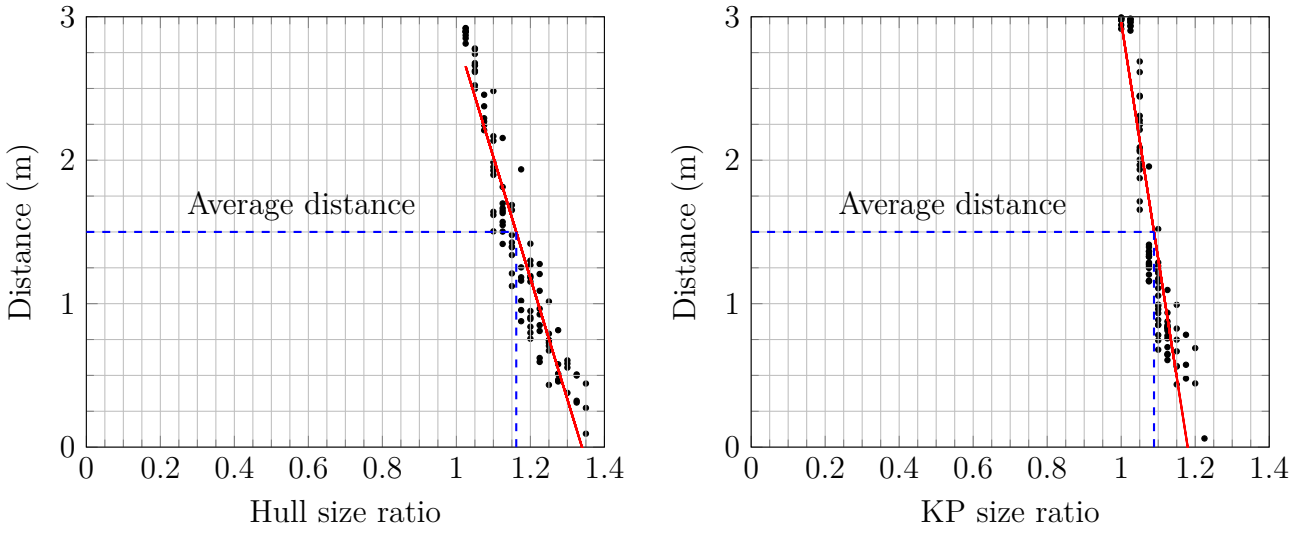


Figure 6.1: Threshold optimization results

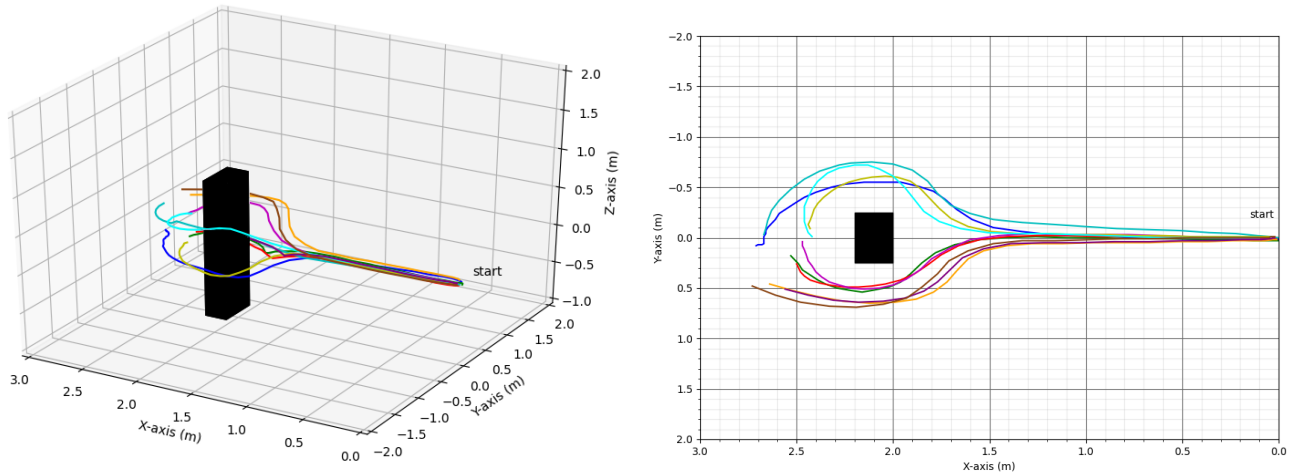


Figure 6.2: SIFT Avoidance manoeuvres: 3D and 2D top view

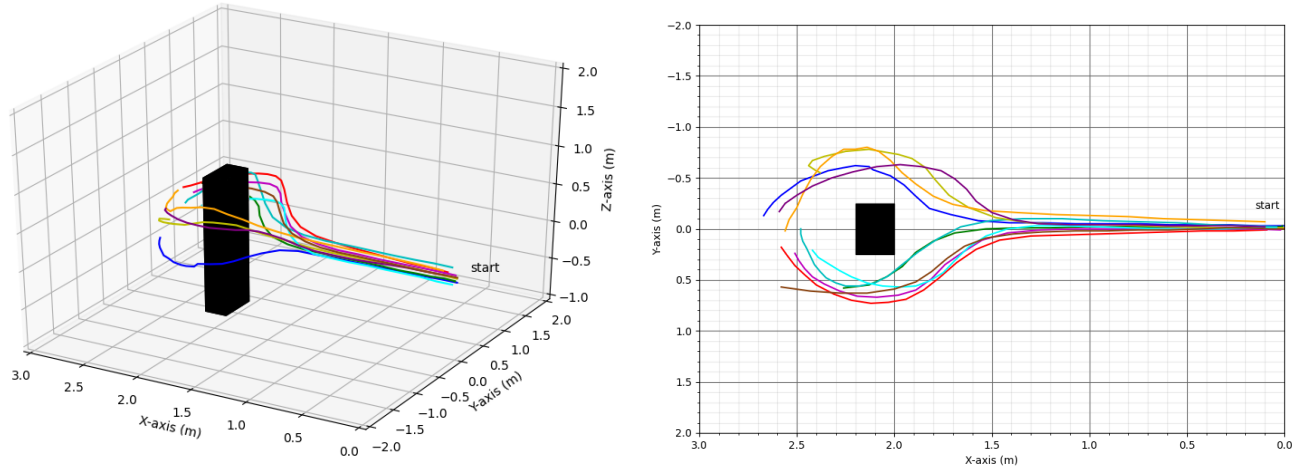


Figure 6.3: YOLO Avoidance manoeuvres: 3D and 2D top view

6.2 Obstacle Avoidance

The obstacle avoidance performance of the proposed method was tested in multiple scenarios. Figures 6.2 and 6.3 show an example of 10 flights taken by the drone for the SIFT and YOLO methods respectively. It demonstrates the drone's ability to avoid frontal obstacles in multiple directions. In these examples, the drone starts at the *start* position, travelling forward until it detects an obstacle, denoted by the black box. It shows the avoidance manoeuvres of the proposed method being comparable to the performance of those of the SIFT method. This is to be expected as the same avoidance commands were used.

The overview of the total performance of both methods is shown in Table 6.1. It shows that the

Table 6.1: Obstacle avoidance accuracy (%) of the selected algorithms

Algorithm	Accuracy (%)				
	Inside			Outside	
	Person	Obstacles	Car	Person	Total
SIFT size expansion	91.0	71.0	42.5	18.3	63.3
proposed CNN algorithm	90.0	90.0	90.0	83.3	88.7

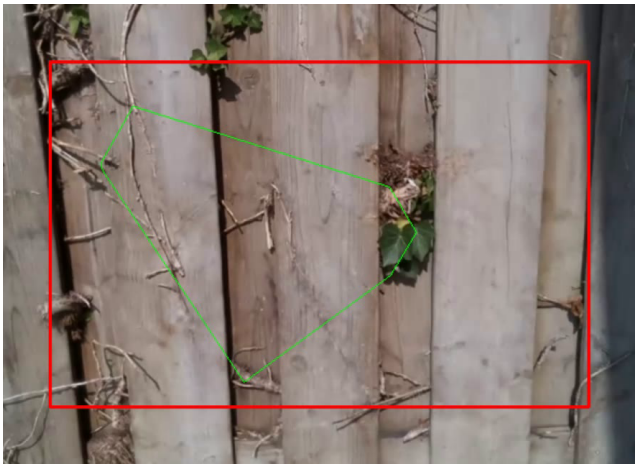
(a) (b)

Figure 6.4: Bounding box inconsistency example. Large size change from frame (a) to (b) on the truck object

proposed method has the highest avoidance accuracy of 83.7% compared to SIFT method which reached 63.3% accuracy.

The main cause of failure for the YOLO method is the inaccuracy of object localization, causing the bounding box around an object to change suddenly in subsequent frames. This triggers the obstacle avoidance too early or too late, causing the drone to collide with the obstacle. An example of this can be seen in Figure 6.4. Here the size of the bounding box around the truck object increases dramatically within one frame, causing a premature avoidance manoeuvre.

Another cause of failure is the inability of the network to detect objects classes not present in its training dataset which causes no bounding boxes to be generated. Thus, the drone will not consider these objects as obstacles and not perform any evasive manoeuvres. An example can be found in Figure 6.5. Where the SIFT method can use keypoints to detect the wall object, the YOLO method fails as the object was not contained in its training dataset. This can be solved by including more objects in the training dataset or by using a more specialised dataset for the working environment, including expected object classes.



(a) SIFT

(b) YOLO

Figure 6.5: Object detection on a wall obstacle

Table 6.2: Obstacle avoidance results SIFT method

	Person	**Inside Obstacles**	**Car**	**Outside Person**	**Total**
Situations	100	100	40	60	300
Avoided	91	71	17	11	190
Failed	9	29	23	49	110
Accuracy (%)	91.0	71.0	42.5	18.3	63.3

The separate results for the SIFT and YOLO methods in all experiments are listed in Tables 6.2 and 6.3 respectively. We see similar performance in the indoor person scenario. This shows our method performs on par with the previous work when detecting a single frontal object on a white background.

The inside obstacle test includes multiple objects to be avoided at once. Using the multiple object tracking capability of our method, the ability to avoid multiple objects is improved compared to the previous work, as accuracy increases from 71.0% to 90.0%. The examples in Figure 6.6 show

Table 6.3: Obstacle avoidance results YOLO method. Using class weights for the car test. (Section 6.3)

	Person	**Inside Obstacles**	**Car**	**Outside Person**	**Total**
Situations	100	100	40	60	300
Avoided	90	90	36	50	266
Failed	10	10	4	10	34
Accuracy (%)	90.0	90.0	90.0	83.3	88.7

(a) SIFT (b) YOLO

Figure 6.6: Object detection on multiple obstacles

that the SIFT method represents both objects as a single convex hull. This causes the size of the hull to increase slower than necessary to avoid the obstacles on time. Because the proposed method generates separate bounding boxes for both obstacles, it can track the size changes of both obstacles at the same time, predicting more accurately when avoidance must take place.

In the outside scenario we see a slightly lower accuracy in comparison to the inside environment for the YOLO method. This can be explained by the harsher lighting conditions found outside compared to the more consistent lighting in an indoor environment. The SIFT method shows a poor result with an accuracy of 18.3% in this scenario. This can be explained by the textured background present behind the obstacle, causing the algorithm to include keypoints from the background in the obstacle convex hull. This means the convex hull increases slower between frames, causing the UAV to detect the object too late or not at all. An example of this can be seen in Figure 6.7, showing the inability of the SIFT method to differentiate close objects to a noisy background. Because of the proposed method's object detector, it can easily separate frontal objects from the background, leading to a significantly higher accuracy of 83.3% in this scenario.

6.3 Smart Avoidance

This experiment shows the influence of the class weights, described in Section 4.3.2, on the avoidance accuracy. The results seen in Table 6.4 show that the accuracy of the YOLO detector can be significantly improved by using the object class in the avoidance procedure. By eliminating the

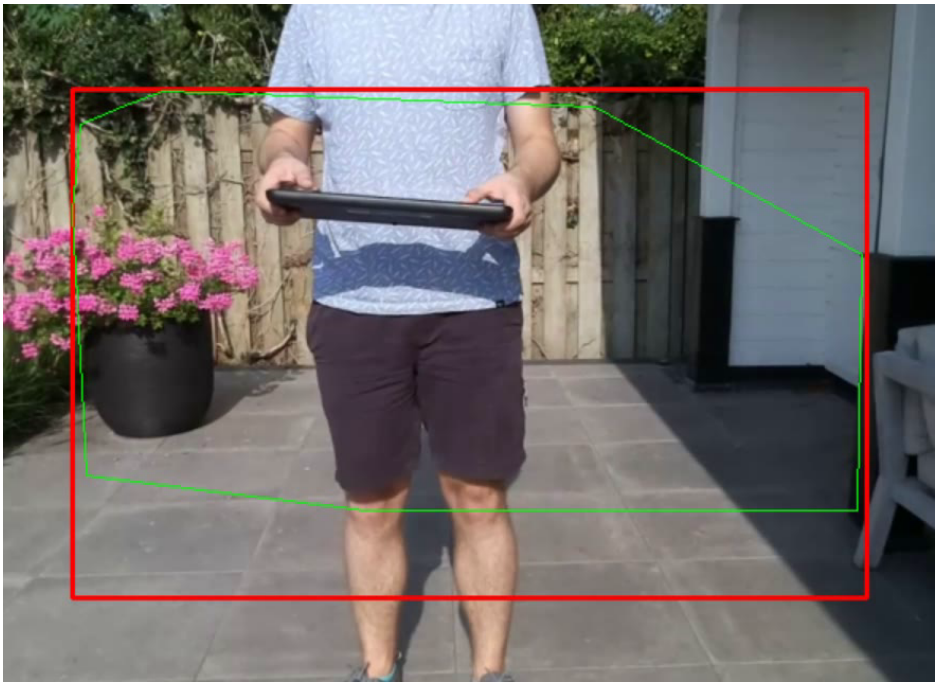


Figure 6.7: SIFT: Person outside example

downward avoidance direction for car objects, the accuracy increased from 40.0% to 90.0%. This example scenario shows the improvements using rich information like the object class can bring to obstacle avoidance. This concept can be used in different ways based on the object present in the dataset and the use case by assigning weights to the zones of object classes if called for.

Table 6.4: Results car experiment

	SIFT	**YOLO Default**	**YOLO Class Weights**
Situations	40	40	40
Avoided	17	16	36
Failed	23	24	4
Accuracy (%)	42.5	40.0	90.0

Chapter 7

Conclusions and Further Research

This thesis proposed a Convolutional Neural Network (CNN) object detector-based obstacle avoidance system for drones using the YOLOv4 [16] network trained on the COCO dataset [18]. It uses only information gathered from the video feed of the onboard camera for detecting and avoiding obstacles. By calculating the size changes of generated bounding boxes around objects in subsequent frames, obstacles can be differentiated from other objects by comparing the size change values to set thresholds.

The proposed YOLOv4 method was compared to non-deep learning Scale-Invariant Feature Transform (SIFT) based method proposed in [17] in various real-life obstacle avoidance experiments. Results show a significant increase in avoidance accuracy from 63.3% using the SIFT method to 83.7% using the proposed YOLO method. Accuracy was found to be on par with the previous work in a simple scenario containing a single obstacle. However, significant improvements were measured in scenarios with multiple obstacles, where accuracy increased from 71.0% to 90.0%. Furthermore, results from experiments containing obstacles on a textured background, an increase of accuracy from 18.3% to 90.0% was measured, where the SIFT method struggles to differentiate background from obstacle. These results show that the proposed method has a consistent high accuracy in various environment, where the SIFT method lacks in more difficult circumstances.

The proposed method struggles in certain scenarios where a detected object is not well represented in the used database. This can cause an unstable generation of bounding boxes between frames, resulting in a premature or delayed avoidance manoeuvre. Furthermore, the proposed method is clearly unable to detect objects that are not represented in the training dataset. However, this can be avoided by including more expected obstacle classes in the training dataset for the environment the UAV is used in.

Based on these conclusions, recommendations for future research are provided. These can focus on two main categories:

- Improve the YOLO method. This can be done by experimenting with different datasets, adjusting the CNN's architecture, or by implementing the algorithm using small, efficient networks like YOLOv4-tiny [16]. This can remove computational restraints allowing a faster processing speed. This can possibly increase responsiveness when performing avoidance manoeuvres.
- Combine the YOLO method with the SIFT approach. Using an object detector or region proposal network to find regions where SIFT convex hulls will be generated may increase accuracy whilst keeping a real-time operating speed. This approach can decrease the impact of bounding box generation inaccuracies whilst still allowing for the detection of multiple obstacles.

Bibliography

[1] UM Rao Mogili and B B V L Deepak. Review on application of drone systems in precision agriculture. *Procedia Computer Science*, 133:502 – 509, 2018. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2018.07.063. URL http://www.sciencedirect.com/science/article/pii/S1877050918310081. International Conference on Robotics and Smart Manufacturing (RoSMa2018).

[2] J. Lee. Optimization of a modular drone delivery system. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–8, 2017.

[3] G. Ding, Q. Wu, L. Zhang, Y. Lin, T. A. Tsiftsis, and Y. Yao. An amateur drone surveillance system based on the cognitive internet of things. *IEEE Communications Magazine*, 56(1): 29–35, 2018.

[4] Julien Fleureau, Quentin Galvane, Francois-Louis Tariolle, and Philippe Guillotel. Generic drone control platform for autonomous capture of cinema scenes. In *Proceedings of the 2nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pages 35–40, 2016.

[5] AK Puttock, AM Cunliffe, K Anderson, and Richard E Brazier. Aerial photography collected with a multirotor drone reveals impact of eurasian beaver reintroduction on ecosystem structure. *Journal of Unmanned Vehicle Systems*, 3(3):123–130, 2015.

[6] Subramanian Ramasamy, Roberto Sabatini, Alessandro Gardi, and Jing Liu. Lidar obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology*, 55:344 – 358, 2016. ISSN 1270-9638. doi: https://doi.org/10.1016/j.ast.2016.05.020. URL http://www.sciencedirect.com/science/article/pii/S1270963816301900.

[7] A. Ferrick, J. Fish, E. Venator, and G. S. Lee. Uav obstacle avoidance using image processing techniques. In *2012 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 73–78, 2012. doi: 10.1109/TePRA.2012.6215657.

[8] Nils Gageik, Paul Benz, and Sergio Montenegro. Obstacle detection and collision avoidance for a uav with complementary low-cost sensors. *IEEE Access*, 3:599–609, 2015. ISSN 2169-3536. doi: 10.1109/ACCESS.2015.2432455.

[9] Massimiliano Iacono and Antonio Sgorbissa. Path following and obstacle avoidance for an autonomous uav using a depth camera. *Robotics and Autonomous Systems*, 106:38 – 46, 2018.

ISSN 0921-8890. doi: 10.1016/j.robot.2018.04.005. URL http://www.sciencedirect.com/science/article/pii/S0921889018301027.

[10] Kimberly McGuire, Guido de Croon, Christophe De Wagter, Karl Tuyls, and Hilbert Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017. ISSN 2377-3774. doi: 10.1109/lra.2017.2658940. URL http://dx.doi.org/10.1109/LRA.2017.2658940.

[11] AA Biewener H-T Lin, IG Ros. Through the eyes of a bird: modelling visually guided obstacle flight. *J. R. Soc. Interface*, 2014. doi: http://dx.doi.org/10.1098/rsif.2014.0239.

[12] H. Alvarez, L. M. Paz, and D. Sturm, J.and Cremers. *Collision Avoidance for Quadrotors with a Monocular Camera*, pages 195–209. Springer International Publishing, Cham, 2016. ISBN 978-3-319-23778-7. doi: 10.1007/978-3-319-23778-7_14. URL https://doi.org/10.1007/978-3-319-23778-7_14.

[13] Paula Fraga-Lamas, Lucía Ramos, Víctor Mondéjar-Guerra, and Tiago M. Fernández-Caramés. A review on iot deep learning uav systems for autonomous obstacle detection and collision avoidance. *Remote Sensing*, 11(18), 2019. ISSN 2072-4292. doi: 10.3390/rs11182144. URL https://www.mdpi.com/2072-4292/11/18/2144.

[14] Ram Prasad Padhy, Sachin Verma, Shahzad Ahmad, Suman Kumar Choudhury, and Pankaj Kumar Sa. Deep neural network for autonomous uav navigation in indoor corridor environments. *Procedia Computer Science*, 133:643 – 650, 2018. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2018.07.099. URL http://www.sciencedirect.com/science/article/pii/S1877050918310524.

[15] P. Chakravarty, K. Kelchtermans, T. Roussel, S. Wellens, T. Tuytelaars, and L. Van Eycken. Cnn-based single image obstacle avoidance on a quadrotor. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6369–6374, May 2017. doi: 10.1109/ICRA.2017.7989752.

[16] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv*, 2020.

[17] Abdulla Al-Kaff, Fernando García, David Martín, Arturo De La Escalera, and José María Armingol. Obstacle detection and avoidance system based on monocular camera and size expansion algorithm for uavs. *Sensors*, 17(5), 2017. doi: 10.3390/s17051061. URL https://www.mdpi.com/1424-8220/17/5/1061.

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10602-1.

[19] Subrahmanyam Vaddi, Chandan Kumar, and Ali Jannesari. Efficient object detection model for real-time uav applications, 2019. URL https://lib.dr.iastate.edu/etd/17592.

[20] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006. ISSN 1558-223X. doi: 10.1109/MRA.2006.1638022.

[21] Rahul Kala. 2 - basics of autonomous vehicles. In Rahul Kala, editor, *On-Road Intelligent Vehicles*, chapter 2.2.2, pages 14 – 15. Butterworth-Heinemann, 2016. ISBN 978-0-12-803729-4. doi: https://doi.org/10.1016/B978-0-12-803729-4.00002-7. URL http://www.sciencedirect.com/science/article/pii/B9780128037294000027.

[22] Xiaojie Zhao, Gang Wang, Maozhi Cai, and Hongkun Zhou. Stereo-vision based obstacle avoidance by finding safe region. *International Journal of Control, Automation and Systems*, 15(3):1374–1383, 2017. ISSN 1598-6446. doi: 10.1007/s12555-015-0354-y.

[23] Daniel Magree, John Mooney, and Eric Johnson. Monocular visual mapping for obstacle avoidance on uavs. *Journal of Intelligent & Robotic Systems*, 74(1-2):17–26, 2014. ISSN 0921-0296. doi: 10.1007/s10846-013-9967-7.

[24] N. Aswini and Satyanarayana Visweswaraiya Uma. Obstacle detection in drones using computer vision algorithm. In *Advances in Signal Processing and Intelligent Recognition Systems*, pages 104–114. Springer Singapore, 2019. ISBN 978-981-13-5758-9.

[25] Alexandros Kouris and Christos Bouganis. Learning to fly by myself: A self-supervised cnn-based approach for autonomous navigation. In *Intelligent Robots and Systems (IROS 2018), 2018 IEEE/RSJ International Conference*, 2018. doi: 10.1109/IROS.2018.8594204.

[26] Shichao Yang, Sandeep Konam, Chen Ma, Stephanie Rosenthal, Manuela M. Veloso, and Sebastian A. Scherer. Obstacle avoidance through deep networks based intermediate perception. *CoRR*, abs/1704.08759, 2017. URL http://arxiv.org/abs/1704.08759.

[27] R. Girshick. Fast r-cnn. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[28] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016. ISSN 1611-3349. doi: 10.1007/978-3-319-46448-0_2. URL http://dx.doi.org/10.1007/978-3-319-46448-0_2.

[29] Palash Baranwal. Computer vision applications, 2019. URL https://medium.com/@palashbaranwal/computer-vision-applications-fab5a3d96c03.

[30] Christopher G. Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[31] David Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–, 11 2004. doi: 10.1023/B:VISI.0000029664.99615.94.

[32] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[33] M. Muja and D. G. Lowe. Fast matching of binary features. In *2012 Ninth Conference on Computer and Robot Vision*, pages 404–410, 2012.

[34] OpenCV. Opencv-python documentation, 2020. URL https://docs.opencv.org/master/da/df5/tutorial_py_sift_intro.html.

[35] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017. ISBN 1617294438.

[36] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39, 06 2015. doi: 10.1109/TPAMI.2016.2577031.

[37] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *arXiv*, 2015.

[38] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *arXiv*, 2016.

[39] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[40] Ayoosh Kathuria. What's new in yolo v3?, 2018. URL https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b.

[41] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1571–1580, 2020.

[42] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37 (9):1904–1916, 2015.

[43] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia. Path aggregation network for instance segmentation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8759–8768, 2018.