

Detection of Unmanned Air Systems Using Multi-Camera Architectures

Ana Margarida Fernandes Veiga

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisor(s): Prof. Afzal Suleman
Prof. Paulo Jorge Coelho Ramalho Oliveira

Examination Committee

Chairperson: Prof. Fernando José Parracho Lau
Supervisor: Prof. Paulo Jorge Coelho Ramalho Oliveira
Member of the Committee: Prof. Rita Maria Mendes de Almeida Correia da Cunha

October 2021

Acknowledgments

I would like to express my gratitude to my dissertation supervisors Prof. Afzal Suleman and Prof. Paulo Oliveira, for their guidance and assessment of my work, which helped me to improve.

I would also like to thank everyone at CfAR for their helpful advice and suggestions during the weekly meetings.

Finally, I would like to thank my friends and family for their friendship, support and encouragement, and without whom this project would not be possible.

Resumo

O número de veículos aéreos não tripulados (UAVs) leves disponíveis no mercado está em crescimento. Devido às suas limitações de carga, UAVs pequenos têm restrições nos sensores que podem transportar e muitos fazem uso de câmeras monoculares ou de profundidade, por serem leves e energeticamente eficientes. UAVs pequenos são adequados para operar em ambientes com muitos obstáculos, que apresentam alto um risco de colisão, tornando essencial a capacidade de detectar obstáculos usando câmeras como sensores.

Adicionalmente, devido à versatilidade e grande disponibilidade deste tipo de UAVs, eles podem também ser explorados para atividades perigosas ou criminosas. Desta forma, a capacidade de detectar e localizar UAVs maliciosos é desta forma também muito importante.

Este trabalho concentra-se numa primeira parte na avaliação das capacidades de fusão de mapas de profundidade monoculares e *stereo* para detecção de obstáculos. Para o efeito, as estimativas de profundidade de uma rede neural são combinadas com medições de uma câmera de profundidade, a fim de obter um mapa de profundidade mais preciso e denso.

O segundo foco deste trabalho consiste em avaliar a possibilidade de utilizar um grupo de UAVs equipados com câmeras monoculares para localizar um UAV intruso. É utilizada uma rede neural de detecção de objetos para detectar o alvo, e de seguida a sua localização é determinada por triangulação, sendo avaliados e comparados três algoritmos de triangulação distintos.

Palavras-chave: Fusão Sensorial, Triangulação, Localização de UAV Intruso, Detecção de Obstáculos

Abstract

The number of lightweight Unmanned Aerial Vehicles (UAVs) available on the market is increasing. Due to limited payload, small UAVs are restricted in the sensors they can carry, and many make use of monocular or depth cameras, since they are lightweight and power efficient. These types of UAV are suitable for operating in cluttered environments, where they are at a high risk of collisions. Therefore, the ability to detect obstacles with camera sensors is essential.

Additionally, because of their versatility and availability of access, these types of UAV can be exploited for dangerous or criminal activities. Being able to detect and localize malicious UAVs is then very important.

This work will focus on evaluating the capabilities of monocular and stereo depth fusion for obstacle detection. Depth predictions from a neural network will be combined with measurements of a depth camera, in order to obtain a more accurate and dense depth map.

The second focus of this work will be to evaluate the possibility of utilizing a group of UAVs equipped with monocular cameras to localize an intruder UAV. An object detector network will be employed for the task of detecting the target, and then the location of the target will be found by triangulation. Three distinct triangulation algorithms will be evaluated and compared.

Keywords: Sensor Fusion, Triangulation, Intruder UAV Localization, Obstacle Detection

Contents

Acknowledgments	iii
Resumo	v
Abstract	vii
List of Tables	xi
List of Figures	xiii
Nomenclature	xv
1 Introduction	1
1.1 Motivation	1
1.2 Related Works	1
1.2.1 Obstacle Detection for Small UAVs	1
1.2.2 Detection and Localization of an Intruder UAV	2
1.3 Objectives	3
1.4 Contributions	3
1.5 Thesis Outline	4
2 Background and Literature Review	5
2.1 Monocular Depth Estimation	5
2.2 Visual Object Detection	6
2.3 Camera Fundamentals	7
2.3.1 Projective Space and Homogeneous Coordinates	8
2.3.2 Pinhole Projection Model	8
2.4 Coordinate Systems	10
2.4.1 Local North-East-Down Coordinate System (NED)	11
2.4.2 Vehicle-Carried North-East-Down Coordinate System	11
2.4.3 Body Coordinate System	11
2.4.4 Camera Coordinate System	12
2.5 Performance Metrics	12
2.5.1 Depth Estimation Metrics	12
2.5.2 Object Detection Metrics	13

3	Methodology	16
3.1	Obstacle Detection for Small UAVs	16
3.1.1	Monocular Depth Estimation	17
3.1.2	Sky Segmentation	17
3.1.3	Fusion of Stereo and Monocular Depth Estimates	18
3.2	Detection and Localization of an Intruder UAV	20
3.2.1	Visual Object Detection	21
3.2.2	Triangulation Algorithms	21
4	Simulations	25
4.1	AirSim Simulation Environment	25
4.1.1	Simulated Environment	25
4.1.2	Data Collection	26
4.2	Trajectories/AirSim Simulations	27
4.2.1	Trajectory 1	28
4.2.2	Trajectory 2	29
4.2.3	Trajectory 3	29
5	Results	31
5.1	Obstacle Detection for Small UAVs	31
5.1.1	Monocular Depth Estimation	31
5.1.2	Fusion Results	33
5.2	Target UAV Localization	36
5.2.1	YOLO Detector	36
5.2.2	Trajectory 1 Simulations	38
5.2.3	Trajectory 2 Simulations	40
5.2.4	Trajectory 3 Simulations	42
5.2.5	Discussion	43
6	Conclusions	45
6.1	Findings	45
6.2	Future Work	45
	Bibliography	47

List of Tables

- 5.1 Error metrics of the predicted depth map for two example images. 32
- 5.2 Average error metrics of the predicted monocular depth map. 33
- 5.3 Average error metrics of the final predicted depth maps. 35

List of Figures

2.1	Representation of the pinhole camera model.	9
2.2	Coordinate systems used for the construction of the camera matrix.	10
2.3	Illustration of the intersection over union (IOU) metric.	14
2.4	Example of a precision-recall curve.	15
3.1	Proposed architecture for the monocular and stereo fusion.	16
3.2	Comparison of CNN depth prediction before and after alignment with the ground truth. . .	18
3.3	Comparison of CNN depth prediction before and after sky removal.	18
3.4	Graphic representation of the steps involved in the attainment of the $W_{c(x,y)}$ weighting factor.	19
3.5	Proposed architecture for the target localization method.	20
3.6	Example images from the Dettfly dataset.	21
4.1	Two examples of scenes from the City Park simulated environment.	26
4.2	Simulated UAV.	26
4.3	Segmentation and corresponding RGB images, captured by the same camera.	27
4.4	Diagram of the AirSim simulations performed.	27
4.5	Representation of trajectory 1.	28
4.6	Representation of trajectory 2 with formation 1.	29
4.7	Representation of trajectory 2 with formation 2.	30
4.8	Representation of trajectory 3.	30
5.1	Monocular depth prediction results for two example images from the Diode Dataset. . . .	32
5.2	Graphic representation of the depth estimation errors obtained.	33
5.3	Representation of the steps taken for the obtainment of the final estimated depth map. . .	34
5.4	Qualitative results on some example images.	35
5.5	YOLO detector F1-score and P-R curves.	36
5.6	Performance of the YOLO detector on the trajectory 1 simulations.	37
5.7	Average RMSE obtained as a function of the distance to the target, and the distance between the two cooperative UAVs.	39
5.8	Real and estimated target trajectory for trajectory 1.	39
5.9	Results obtained for trajectory 2.	40
5.10	Real and estimated target trajectory for trajectory 2 and formation 1.	41

5.11 Real and estimated target trajectory for trajectory 2 and formation 2.	42
5.12 Results obtained for trajectory 3.	43

Nomenclature

Greek symbols

θ Pitch angle.

ϕ Roll angle.

ψ Yaw angle.

Roman symbols

E Camera transformation matrix.

K Camera calibration matrix.

P Camera matrix.

R Rotation matrix.

W_c Confidence weighting factor.

W_s Ratio weighting factor.

Z Final depth estimate.

Z_m Monocular depth estimate.

Z_s Stereo depth estimate.

n Number of cooperative UAVs.

\mathbf{t} Translation vector.

\mathbf{u} Target detection vector.

u, v Target detection image coordinates.

\mathbf{x} Target location vector.

x, y, z Target location Cartesian components.

Subscripts

i Computational index.

p Pixel.

x, y, z Cartesian components.

Superscripts

\sim Homogeneous coordinates.

\wedge Estimate.

T Transpose.

Chapter 1

Introduction

1.1 Motivation

Nowadays, there is a large number of affordable, lightweight quadrotors commercially available on the market. Due to their small size, they are particularly suited for operating at low altitude in cluttered environments [1], where the risk of colliding against unknown obstacles is much higher. Therefore, autonomous obstacle detection and, ultimately, obstacle avoidance play a key role in the safety of these types of unmanned aerial vehicles (UAVs). However, obstacle detection in unknown environments is a challenging task, especially for small UAVs that are limited in the sensors they can carry due to weight constraints and power requirements [2].

Visual detection of UAVs, on the other hand, has received increasing attention in recent years since its use is fast expanding in a wide range of applications, including agriculture [3], 3D mapping [4] and infrastructure inspection [5]. They have proven to be both autonomous and versatile in a wide range of tasks, and are also increasingly becoming a part of citizens' daily lives in the recreational sector, due to decreasing equipment cost and relatively easy maneuvering [6]. However, because of their versatility and availability of access, they can be exploited for dangerous or criminal activities. Malicious UAVs, for example, can endanger key infrastructure or events, or even interfere with manned aircraft. In recent years, numerous instances involving UAVs flying over restricted areas, around important infrastructure, or during public events have been reported in the media [7], [8]. The visual identification of unknown aerial vehicles is a crucial step in the development of necessary UAV defensive systems.

1.2 Related Works

1.2.1 Obstacle Detection for Small UAVs

Considering the limitations of smaller UAVs, many reported studies make use of lightweight and power efficient sensors like monocular cameras and depth cameras for obstacle detection. In Yang et al. [9], a probabilistic convolutional neural network (CNN) was designed for monocular depth prediction,

with the goal of obstacle avoidance. Since only a monocular camera was used, the depth is predicted up to a scale factor. Deep learning was also used by Wang et al. [10], where a CNN was used in combination with a depth camera for obstacle avoidance. First, the CNN was employed to obtain the obstacle's classification and bounding box. Then, the obstacle's profile and 3D spatial information are extracted from the depth map provided by the depth camera.

While depth cameras provide metric information about the localization of the obstacles, their range is very limited, since the types of cameras possible on a quadrotor have necessarily a small baseline. Additionally, they may have limited performance in regions with low texture or where objects are partly occluded [2].

When it comes to monocular cameras, their main advantage compared to stereo vision is that since only one view is considered, theoretically their only range limitation is imposed by the image resolution. Thus, monocular cameras should perform well related with very close or very far away objects [11]. On the other hand, in contrast to the case with depth cameras, depth estimation methods that make use of monocular cameras are typically unable to provide metric information, and instead rely on additional sensors for absolute depth retrieval. For example, in Teixeira et al. [12] a neural network that performs depth completion takes as input the RGB image captured by a monocular camera as well as LiDAR measurements.

With a depth camera it is possible to take advantage of both the depth map and the RGB image as complementary information sources, and mitigate some of the limitations of these two sensors individually. In Fácil et al. [13] a method for the fusion of single- and multi-view depth estimates was developed, and in Martins et al. [11], a method for the fusion of stereo and monocular depth estimates was presented. In Zhang and Funkhouser [14], monocular depth estimation was used to complete the depth channel of an RGB-D image, by making use of surface normal and occlusion boundaries.

1.2.2 Detection and Localization of an Intruder UAV

The detection and localization of intruder UAVs is also an important task, and many reported studies make use of multi-sensor methods to tackle this problem. Fasano et al. [15] presents a methodology for radar and electro-optical data fusion, with the goal of non-cooperative sense and avoid. First, a radar detection defines an image search window, whose size is related to the distance the object was detected at. Then, the camera is used to detect the potential UAV, and both measurements are combined in a Kalman filter for track estimation. Another strategy presented in Park et al. [16] consists of the fusion of vision and LiDAR sensors. An object detector neural network is used to detect aerial vehicles in the camera image, while a clustering method is utilized to detect objects in LiDAR point cloud data. Both measurements are then combined to obtain the UAV's position.

The task of detection and localization of intruder drones is again made especially challenging for lightweight UAVs, due to the limitations in the type of sensors that they can carry. In Huang and Lai [17], a method for the distance detection of a UAV using only a monocular camera was proposed. First, a neural network for object detection called YOLO (You Only Look Once) was used to detect the UAV

in the image captured by the camera. Then, another neural network was employed to estimate the distance to the target. In Zahedi et al. [18], two neural networks were utilized for accurate mobile target localization and tracking. More traditional methods were used in Husodo et al. [19] and Laurito et al. [20], where algebraic expressions and prior knowledge of the target's dimensions were used to calculate its position. Husodo et al. [19] also proposes a method for following the target UAV, in order to obtain better results.

The use of multiple cooperative UAVs for the detection of another has also been studied in the literature. In Shinde et al. [21] a YOLO network was used to detect the target in images captured by the group of cooperative UAVs, and subsequently its position was ascertained, while in Arnold and Brown [22] different methods of swarm formation for the purposes of malicious UAV tracking are evaluated. Moreover, in Khanapuri and Sharma [23] a neural network based approach was proposed to improve the accuracy in the estimated positions of multiple targets. In this approach, a neural network is used to predict the required azimuth and elevation angles for each cooperative UAV so that the targets remain in their fields of view.

1.3 Objectives

The first goal of this thesis is a feasibility study to determine whether monocular depth estimation strategies, making use of current deep learning methods, are viable to improve the measurements of an onboard depth camera, for UAV obstacle detection. In particular, the objective is to leverage the information contained in the RGB image provided by the depth camera to fill in the stereo depth values missing due to the depth camera's range limitations.

This thesis also aims to explore the use of triangulation algorithms for the localization of uncooperative Unmanned Air Vehicles by a group of cooperative UAVs, each equipped with a monocular camera. The proposed approach utilizes the YOLO algorithm [24] to detect the target in each captured image, which provides the necessary information for its localization. Simulations should be performed in order to compare the performance of three triangulation algorithms in diverse situations.

1.4 Contributions

The contributions of this work are:

- Adoption of a multi-sensor fusion methodology for UAV obstacle detection.
- Training and evaluation of a YOLO network for UAV detection.
- Evaluation and comparison of three triangulation algorithms for target UAV localization, using the AirSim Simulator tool.
- Provided suggestions for the improvement of the triangulation results.

1.5 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 presents the current state-of-the-art algorithms for monocular depth estimation as well as visual obstacle detection. In addition, background knowledge for the present work is provided. Chapter 3 explains the methods related to the obstacle detection for lightweight UAVs, as well as detection and localization of a target. Chapter 4 describes the AirSim Simulation Environment, and discusses the different simulations made to evaluate the target localization results. Chapter 5 presents and discusses the results obtained, first for the monocular and stereo depth fusion method and later for the target localization through the triangulation algorithms. Finally, Chapter 6 summarizes the main conclusions of this thesis and outlines possible directions for future research.

Chapter 2

Background and Literature Review

This chapter starts by presenting a literature review on the the use of artificial intelligence methods for the tasks of monocular depth prediction, in section 2.1, and visual object detection, in section 2.2. It then presents an overview of some necessary background information, namely camera parameters (section 2.3), reference frames (section 2.4) and performance metrics (section 2.5).

2.1 Monocular Depth Estimation

Monocular depth estimation is a computer vision research problem that has been explored by several research groups, with varying degrees of success, over the past decades.

Early work on monocular depth estimation used simple geometric assumptions. In Hoiem et al. [25], rather than predicting depth explicitly, rudimentary 3D models are generated from outdoor photos by first labeling parts of the input image as “ground”, “sky” or “vertical”. Using a series of simple assumptions, these labels are then utilized to “cut and fold” the image into a pop-up model, onto which the original image is then texture mapped. These produced models are necessarily limited by the very strict geometric constraints that have been imposed.

Ladicky et al. [26] proposed that the problems of depth estimation and semantic segmentation should be addressed together by taking advantage of a property of perspective geometry, which states that the perceived size of the objects scales inversely with the distance from the camera. This property was used to condition the depth of an object on its inferred semantic class, thereby reducing the need for a pixel-wise depth classifier to a much simpler classifier that predicts only the likelihood of a pixel being at a given depth. This method effectively addressed one major flaw in classical methods, namely the difficulty in accurately estimating an object’s depth if it hasn’t been seen at the same depth during training.

More recently, significant progress has been made by directly regressing scene depth from the input image using convolutional neural networks. Eigen et al. [27] addresses the depth predicting task by employing two deep network networks, one that produces a coarse global prediction based on the entire image and another that refines it locally. This model is free of geometric priors and hand-engineered features, and it learns everything from the training data. In future work [28], a more general multiscale

convolutional network is employed, which is used for depth prediction, surface normal estimation, and semantic labeling. This network is able to adapt to each task with only small modifications.

These methods require extensive datasets with ground-truth depth for training, which is commonly acquired using RGB-D cameras or LiDAR sensors. Such scanning methods have important limitations. RGB-D cameras are constrained in terms of range, and laser scanners are cumbersome to operate or produce sparse depth maps. Garg et al. [29] proposed to use multiple views of a scene for self-supervised learning. The training is done using pairs of stereo images, with known camera motion between the two. This significantly simplifies the acquisition of training data, since labelled ground-truth depth is no longer required.

Another way to leverage self supervision is to exploit apparent motion. For example, Zhou et al. [30] presents an unsupervised learning approach for estimating dense 3D geometry and camera motion from unstructured video sequences. This method requires only a sequence of images as input, and estimates the camera pose parameters of the input set, along with the dense depth of the scene. However, due to the nature of these methods, they are difficult to apply to dynamic scenes.

Nowadays, the success of monocular depth estimation depends upon large and diverse training sets. Recent improvements have been made from combining datasets with different characteristics. In Ranftl et al. [31], new tools were developed that enable mixing multiple datasets during training, even if their annotations are incompatible. In addition to the use of traditional datasets, 3D movies were also utilized as a data source.

2.2 Visual Object Detection

Existing approaches for visual object detection can be divided into two categories: classical and deep learning methods.

Classical techniques are comprised of two steps. First it is necessary to extract object features using a feature descriptor, for example Histogram of Oriented Gradients (HOG), and then categorize the features using machine-learning techniques such as Support Vector Machine (SVM) or Adaboost.

The HOG feature descriptor, proposed by Dalal and Triggs [32], serves as a foundation for many object detectors. By analyzing the image's pixel gradients, it can extract the shape or structure of an object, since areas with large gradients often represent edges of an object. The image is first divided into different regions, and for each region a histogram based on the gradients and orientation of the pixel values is created.

The Deformable Part-based Model (DPM) [33] is one of the detection methods based on the HOG feature descriptor. In order to tackle the problem of geometric deformations, this model uses various parts of the image separately in order to determine if and where an object of interest exists. For example, detections of small parts of a body like "head", "leg" or "arm" can be used to determine whether an image contains a person.

In contrast to classical approaches that rely on hand-crafted features, deep-learning-based approaches rely on CNN features and, as a result, have a greater ability to represent complicated objects.

The disadvantage of employing a CNN, however, is that it has significant computational requirements and requires large datasets to train. Deep learning approaches to object detection are broadly classified into two types: two-stage detectors and single-stage detectors.

An early example of a two-stage detector is Regions with CNN features (R-CNN) [34]. R-CNN combines region proposals with CNNs. It creates 2000 rectangular regions in the image that might contain an object via a selective search algorithm, and in each region a CNN acts as a feature extractor. These features are classified by a SVM, and finally a bounding box is predicted for each identified object. In future work the Fast R-CNN model [35] is proposed, in which CNNs are used for both feature extraction and classification. This solution overcomes some of the speed limitations of R-CNN, since the CNN is fed with one region per image instead of 2000, and generates a convolutional feature map where the region proposals are identified. A second improvement to R-CNN was made with the Faster R-CNN model [36]. Instead of using a selective search algorithm on the feature map to identify the region proposals, a separate network called Region Proposal Network (RPN) is used to predict the region proposals. This way, the algorithm applied a CNN end-to-end.

Single-stage approaches perform both the object localization and classification in a single step. Compared to two-stage solutions, they have faster processing speeds at the cost of moderate detection accuracies.

The Single Shot MultiBox Detector (SSD), proposed by Liu et al. [37], completely eliminates region proposal generation, and is able to detect objects using a single deep neural network. It does so by creating default bounding boxes over different aspect ratios and scales. The network then generates a score for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Compared to the previously mentioned methods, SSD is less computationally demanding and still achieves good results.

Another popular single-stage method is the You Only Look Once (YOLO) detection algorithm [38]. YOLO approaches object identification as a straightforward regression problem rather than a classification problem with a pipeline in which regions of interest are produced and subsequently categorized. It separates the input image into a grid, and for each cell, a specific number of bounding boxes are predicted and assigned a confidence score. The boxes with low confidence can be removed, and the confidence threshold can be modified, affecting the number of boxes that remain. Furthermore, because many bounding boxes can be associated with the same object, overlapping boxes are removed as well. Subsequent improvements were made to the algorithm [39] [24]. The most recent version, YOLOv3, was selected as the object detector for this thesis due to its accuracy and fast executing speed [40].

2.3 Camera Fundamentals

This section introduces some fundamental knowledge required for the object localization approaches presented in Chapter 3. Subsection 2.3.1 explains homogeneous coordinates, which are necessary to describe the pinhole camera projection model, detailed in subsection 2.3.2.

2.3.1 Projective Space and Homogeneous Coordinates

Any point $\mathbf{x} = (x_1, \dots, x_n)^T \in \mathbb{R}^n$ can be described in the n -dimensional Euclidean space. In particular, the three-dimensional Euclidean space \mathbb{R}^3 can be used to describe points in the three-dimensional environment we live in. The projective space \mathbb{P}^n , however, can be utilized to simplify some equations, particularly in the context of projections. This space extends the Euclidean space by adding an additional coordinate designated w , and makes use of homogeneous coordinates. A point $\tilde{\mathbf{x}} = (\tilde{x}_1, \dots, \tilde{x}_n, \tilde{w})^T$ in homogeneous coordinates represents the point

$$\mathbf{x} = \left(\frac{\tilde{x}_1}{\tilde{w}}, \dots, \frac{\tilde{x}_n}{\tilde{w}} \right) \quad (2.1)$$

in Cartesian coordinates. As a result, when a point's homogeneous coordinates are multiplied by a non-zero scalar, the resultant coordinates represent the same point. Thus unlike Cartesian coordinates, a single point can be represented by infinitely many homogeneous coordinates, therefore w can be selected arbitrarily when converting a point from Cartesian to homogeneous coordinates. A simple method is setting $w = 1$.

The number of coordinates required to represent a point in homogeneous coordinates is one more than the dimension of the projective space under consideration. For example, a point in the projective plane \mathbb{P}^2 , of particular importance for this thesis, requires three homogeneous coordinates. In this thesis, vectors expressed in homogeneous coordinates will have a tilde sign, while the absence of a tilde indicates that the vector is expressed in Cartesian coordinates.

2.3.2 Pinhole Projection Model

The pinhole camera projection model [41] describes the projection of points in three-dimensional space onto a two-dimensional image plane. It does so by considering that the camera aperture can be described as a single point (called pinhole), and that all the light captured by the camera must pass through it before reaching the optical sensor. Despite its approximations, it is a reasonable description of how a camera depicts a 3D scene, and is widely used in computer vision applications [42].

This concept is illustrated in Figure 2.1. Let \mathbf{X} denote a point with coordinates (X, Y, Z) and \mathbf{u} its image projection with coordinates (u, v, w) . Both points are expressed in the camera reference frame. Since \mathbf{u} lies in the image plane, it is known that $w = f$.

Since points \mathbf{X} , \mathbf{u} and the pinhole point (represented in the figure as optical center) are collinear, there is some scalar λ so that

$$\begin{cases} u = \lambda X \\ v = \lambda Y \\ f = \lambda Z \end{cases} \Leftrightarrow \lambda = \frac{u}{X} = \frac{v}{Y} = \frac{f}{Z} \quad (2.2)$$

therefore, the image coordinates of the projection of point \mathbf{X} are given by

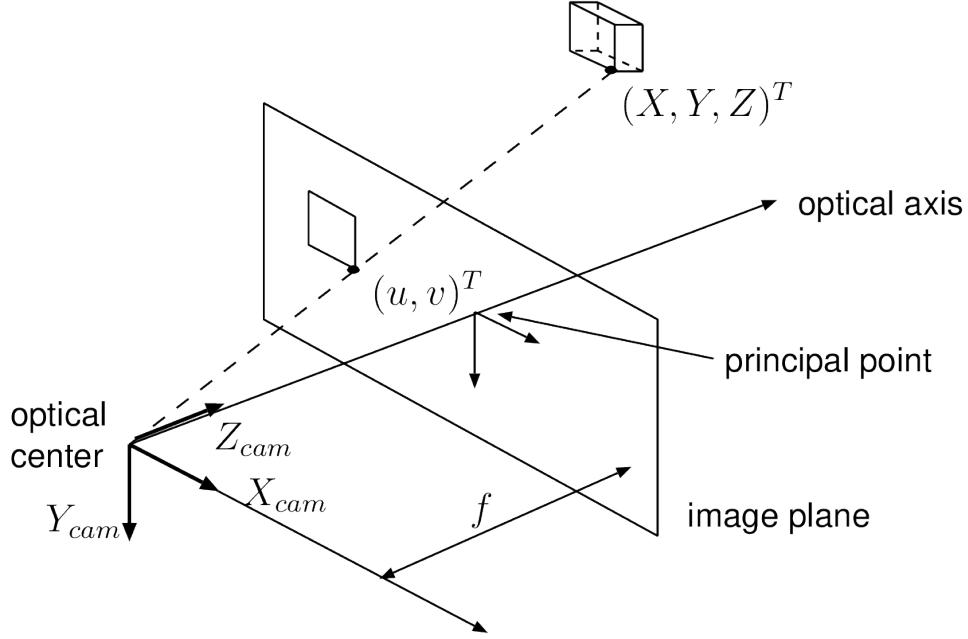


Figure 2.1: Representation of the pinhole camera model, which describes the relationship between a 3D point $\mathbf{X} = (X, Y, Z)^T$ and its corresponding 2D projection $\mathbf{u} = (u, v)^T$ onto the image plane [43].

$$\begin{cases} u = f \frac{X}{Z} \\ v = f \frac{Y}{Z} \end{cases} \quad (2.3)$$

The result above is only valid when point \mathbf{X} is expressed in the camera reference frame. It is often more useful to be able to find the image projection of a point expressed in the world reference frame directly.

The camera matrix $P \in \mathbb{R}^{3 \times 4}$ describes the relation between a point in the world reference frame and its projection in the camera reference frame [44]. The projection $\tilde{\mathbf{u}}$ of a three-dimensional point $\tilde{\mathbf{x}}$ onto the two-dimensional image plane is then given by

$$\tilde{\mathbf{u}} = P\tilde{\mathbf{x}} \quad (2.4)$$

where $\tilde{\mathbf{u}}$ and $\tilde{\mathbf{x}}$ are both expressed in homogeneous coordinates.

The camera matrix P consists of two camera elements. Firstly, the camera intrinsic parameters are described by matrix $K \in \mathbb{R}^{3 \times 3}$, and include information about camera specifics like its focal length and principal point. Second, the camera extrinsic parameters express how a point in the world coordinate system is transformed into the camera coordinate system. This transformation, described by matrix $E \in \mathbb{R}^{3 \times 4}$, is characterized by a rotation matrix $R \in \mathbb{R}^{3 \times 3}$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$. Matrix E is then determined as

$$E = [R|\mathbf{t}] \quad (2.5)$$

and the camera matrix P , defined as $P = KE$, thus corresponds to

$$P = K[R|t] \quad (2.6)$$

2.4 Coordinate Systems

For the construction of the camera matrix, described in section 2.3.2, knowledge of the transformation from world coordinates into coordinates in the camera reference frame is necessary. To make this calculation easier, other intermediate reference frames are introduced as well [45]. Each of the coordinate systems is depicted in Figure 2.2 and is described next, along with the transformations between them.

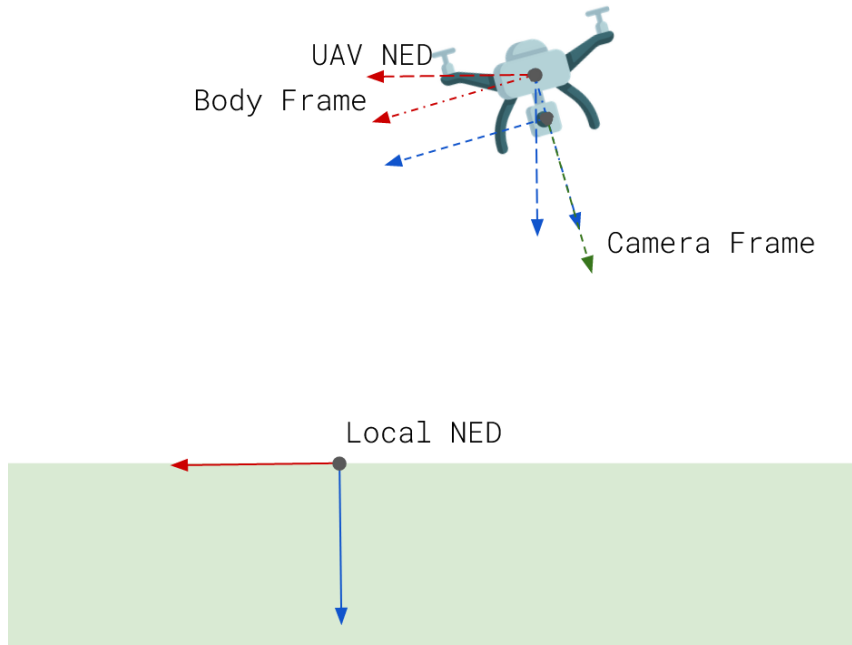


Figure 2.2: Coordinate systems used for the construction of the camera matrix. The X-axis is red, the Y-axis is green and the Z-axis is blue.

The transformations between frames are expressed as 4 by 4 matrices, where $T_A^B \in \mathbb{R}^{4 \times 4}$ denotes the transformation from coordinate system A to coordinate system B. Each transformation matrix can be constructed from a rotation matrix R and a translation vector t , in such a way that the resulting matrix is given by

$$T_A^B = \begin{bmatrix} R_{00} & R_{01} & R_{02} & t_0 \\ R_{10} & R_{12} & R_{12} & t_1 \\ R_{20} & R_{21} & R_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.7)$$

These transformation matrices can be multiplied by points or vectors, expressed in homogeneous coordinates, in order to obtain their transformed counterpart. Several transformations can also be combined by multiplying their corresponding matrices together.

2.4.1 Local North-East-Down Coordinate System (NED)

The first coordinate system used is the Local North-East-Down (NED) coordinate system. This system was used as the world reference frame, and both the cooperative UAVs and the target's positions are provided in it. It is constructed according to the following rules:

1. The origin is arbitrarily fixed to a point on the Earth's surface.
2. The X-axis points toward the North direction.
3. The Y-axis points toward the East direction.
4. The Z-axis is perpendicular to both other axis and points towards the center of the Earth.

2.4.2 Vehicle-Carried North-East-Down Coordinate System

The next coordinate system used is the Vehicle-Carried NED, or UAV NED, coordinate system. This system is constructed similarly to the local NED system, but in this case its origin is located at the center of gravity of the flying vehicle it corresponds to.

Rigorously speaking, the axis directions of the vehicle-carried NED frame change slightly according to the UAV's current position, and so are not strictly aligned with those of the local NED frame. However, since the small UAVs that this thesis deals with fly only in a short area, it is acceptable to assume that the vehicle-carried and local NED coordinate systems always point in the same direction. The transformation between the local NED and vehicle-carried NED reference frames is then simplified and corresponds to

$$T_{\text{LOCAL_NED}}^{\text{UAV_NED}} = \begin{bmatrix} 1 & 0 & 0 & x_{\text{UAV}} \\ 0 & 1 & 0 & y_{\text{UAV}} \\ 0 & 0 & 1 & z_{\text{UAV}} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.8)$$

where $(x_{\text{UAV}}, y_{\text{UAV}}, z_{\text{UAV}})$ corresponds to the UAV's position.

2.4.3 Body Coordinate System

The third needed coordinate frame is the Body coordinate system, which has the same origin as the vehicle-carried system, but its axes match the principle axes of a UAV. This system is therefore constructed according to the following rules:

1. The origin is located at the center of gravity of the UAV.
2. The X-axis points forward, lying in the symmetric plane of the UAV.
3. The Y-axis points toward the right side of the UAV.
4. The Z-axis is perpendicular to both other axis and points downward.

The rotation matrix from the vehicle-carried NED frame to the body frame, $R_{UAV_NED}^{BODY_FRAME}$, is given by

$$R_{UAV_NED}^{BODY_FRAME} = \begin{bmatrix} \cos(\theta) \cos(\psi) & \cos(\theta) \sin(\psi) & -\sin(\theta) \\ \sin(\phi) \sin(\theta) \cos(\psi) - \cos(\phi) \sin(\psi) & \sin(\phi) \sin(\theta) \sin(\psi) + \cos(\phi) \cos(\psi) & \sin(\phi) \cos(\theta) \\ \cos(\phi) \sin(\theta) \cos(\psi) + \sin(\phi) \sin(\psi) & \cos(\phi) \sin(\theta) \sin(\psi) - \sin(\phi) \cos(\psi) & \cos(\phi) \cos(\theta) \end{bmatrix}$$

where ϕ , θ and ψ correspond to the UAV's roll, pitch and yaw angles, respectively. In agreement with equation (2.7), the full transformation matrix $T_{UAV_NED}^{BODY_FRAME}$ corresponds to

$$T_{UAV_NED}^{BODY_FRAME} = \begin{bmatrix} R_{UAV_NED}^{BODY_FRAME} & \mathbf{0} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.9)$$

2.4.4 Camera Coordinate System

The final coordinate system used is the camera coordinate system. For simplicity, it was considered that the camera coordinate system has its origin coincident with the origin of the body reference frame, that is, the displacement between the UAV's center of mass and the camera's principal point was ignored. Nonetheless, the orientation of the camera system's axes is fundamentally different than that of the other systems. The camera coordinate system is built in accordance with the following procedure:

1. The origin is located at the center of gravity of the UAV.
2. The X-axis points to the right of the camera, matching the direction of the X-axis in the captured images.
3. The Y-axis points downwards, also matching the direction of the Y-axis in the captured images.
4. The Z-axis points in the view direction of the camera.

As a consequence, the transformation from the body frame to the camera frame corresponds to

$$T_{BODY_FRAME}^{CAMERA_FRAME} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.10)$$

2.5 Performance Metrics

2.5.1 Depth Estimation Metrics

To measure the accuracy of the depth estimation algorithm, error metrics commonly found in the literature [46] were employed.

Given a predicted depth map and the corresponding ground truth, where \hat{d}_p and d_p denote the estimated and ground-truth depths respectively at pixel p , and T is the total number of pixels for which there exist both valid ground truth and predicted depth, the following metrics can be defined:

- **Absolute Relative Error**

The absolute relative error corresponds to the average of the relative error obtained across all pixels that have a valid ground truth and predicted depth, and is given by

$$\frac{1}{T} \sum_p \frac{|d_p - \hat{d}_p|}{d_p} \quad (2.11)$$

- **Linear Root Mean Square Error (RMSE)**

Similarly, the linear root mean square error can be defined as

$$\sqrt{\frac{1}{T} \sum_p (d_p - \hat{d}_p)^2} \quad (2.12)$$

- **Accuracy Under a Threshold**

Finally, the accuracy under a threshold corresponds to the percentage of pixels for which the predicted depth falls under a certain threshold of the ground-truth, and is given by

$$\% \text{ of } d_p \text{ s.t. } \max\left(\frac{\hat{d}_p}{d_p}, \frac{d_p}{\hat{d}_p}\right) = \delta < th \quad (2.13)$$

where th is a predefined threshold, usually equal to 1.25 , 1.25^2 and 1.25^3 .

2.5.2 Object Detection Metrics

This subsection presents the evaluation metrics relating to object detection, which will later be used to evaluate YOLO's performance. These metrics are detailed in Padilla et al. [47] and Padilla et al. [48].

At its most basic level, evaluating the performance of an object detector involves determining whether or not a given detection is valid. In order to characterize the subsequent evaluation metrics, the following definitions are necessary:

- True positive (TP): A correct detection of a ground-truth bounding box;
- False positive (FP): An erroneous detection of a nonexistent object or a misplaced bounding box when detecting an existing object;
- False negative (FN): A ground-truth bounding box missed by the model;

It is worth noting that a true negative (TN) result is not applicable in the context of object detection since there is an endless number of bounding boxes which should not be detected in any given image.

- **Intersection over Union**

The previous definitions require specifying what constitutes a “correct detection” and a “incorrect detection”. In order to determine the validity of a detection, a supporting metric known as Intersection over Union (IoU) is required.

The IoU divides the overlapping region between the predicted bounding box B_p and the ground-truth bounding box B_{gt} by the area of union between them, that is

$$\text{IoU} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})} \quad (2.14)$$

A visual interpretation of the IoU metric is present in Figure 2.3. This metric ranges from 0, signifying no overlap between the ground-truth and the predicted bounding boxes, and 1, implying perfect overlap between the two bounding boxes.

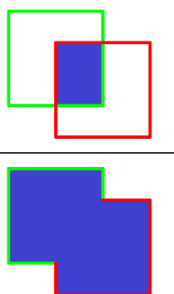
$$\text{IOU} = \frac{\text{area of overlap}}{\text{area of union}} = \frac{\text{img}}{\text{img}}$$


Figure 2.3: Illustration of the intersection over union (IOU) metric [48].

Using this metric, a true positive can be redefined as a detection for which $\text{IoU} \geq 0.5$ and, conversely, a detection for which $\text{IoU} \leq 0.5$ can be classified as a false positive.

- **Precision**

Precision refers to a model’s ability to recognize only the relevant objects, and corresponds to the percentage of correct predictions. It is given by

$$P = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}} \quad (2.15)$$

- **Recall**

The ability of a model to find all relevant objects, that is, all ground-truth bounding boxes, is known as recall. It corresponds to the percentage of all ground-truth bounding boxes that were correctly predicted, and is given by

$$R = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}} \quad (2.16)$$

- **Precision-Recall Curve**

A plot of precision as a function of recall is known as the precision-recall (PR) curve, of which Figure 2.4 illustrates an example. It depicts the trade-off between the two metrics for varying model detection confidence values. If the number of false positives is low, the precision score will be high but more object instances may be overlooked, resulting in a high number of false negatives and a poor recall score.

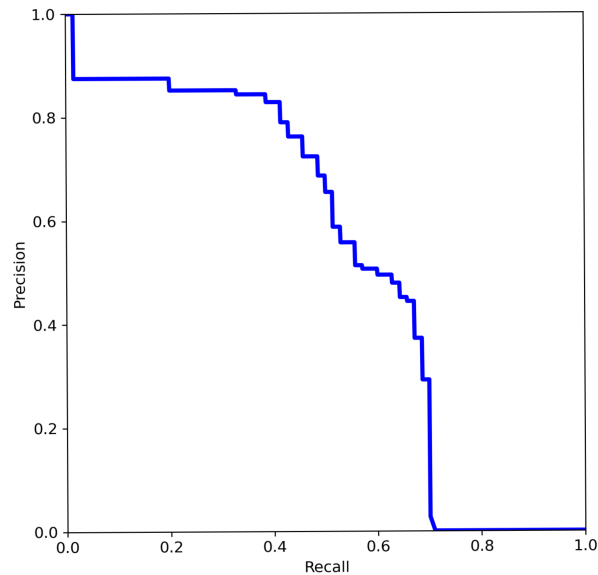


Figure 2.4: Example of a precision-recall curve.

Conversely, lowering the model detection confidence threshold to accept more positives would increase the recall value, but false positives may increase as well, lowering the precision score. A successful model should maintain high precision and recall scores for varying confidence threshold values.

- **F1 Score**

The F1-score is a measure of a model's accuracy on a certain dataset, and is defined as the harmonic mean of the precision and recall of a given model, that is

$$F1\ score = 2 \frac{P \cdot R}{P + R} \tag{2.17}$$

The F1-score ranges from 0 to 1, with 0 indicating that precision or recall scores (or both) are 0%, and 1 indicating that both precision and recall are 100%. Because it is calculated for a single confidence threshold, it can be used to compare a model's performance for different confidence threshold values.

Chapter 3

Methodology

This chapter presents the methods of the proposed approaches for both obstacle detection and target localization, in sections 3.1 and 3.2 respectively. Both sections start with an overview of the proposed architectures, and then discuss necessary algorithms and concepts.

3.1 Obstacle Detection for Small UAVs

An obvious strategy to take advantage of all the information provided by the depth camera is to use the RGB-D image (RGB + *depth*), which encompasses the color image and the depth map, directly as input of a neural network, and train it so that it can fill in the depth of the missing pixels. However, according to Zhang and Funkhouser [14], this strategy does not give good results, especially for large regions with missing distance measurements.

It was then decided to use an architecture similar to the one described in Martins et al. [11], which is exemplified in Figure 3.1.

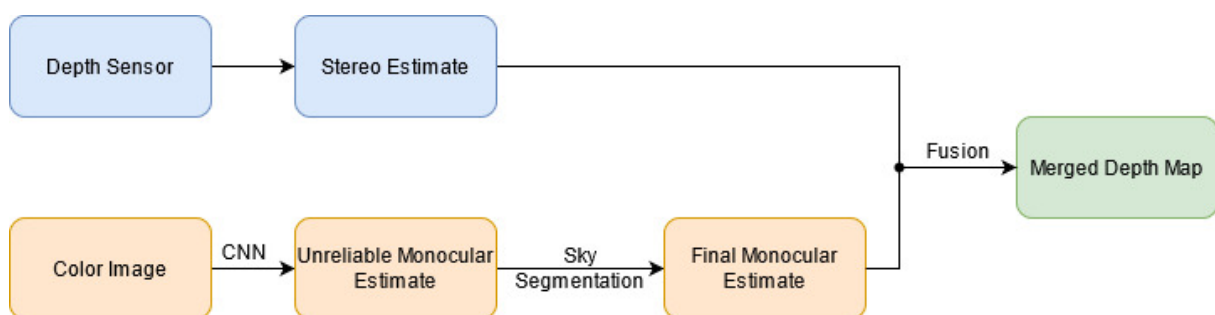


Figure 3.1: Proposed architecture for the monocular and stereo fusion.

This architecture can be divided into three blocks: the stereo estimate (blue), the monocular estimate (orange) and the merging of both estimates (green). The stereo estimate is given directly by the depth camera, and may have missing values for some pixels. On the other hand, the monocular estimate is given by a convolutional neural network, which receives the RGB image as an input and outputs the estimated distance of each pixel to the camera. Because the monocular depth predictions in sky areas

are unreliable, a sky segmentation step is included, where sky areas are removed from the monocular estimate.

Then, both estimates are merged, resulting in a more complete and accurate depth map than the previous ones.

3.1.1 Monocular Depth Estimation

To perform the monocular depth estimation, the MiDaS v2.1 Small [31] neural network was chosen, due to its robustness to various types of environment and fast execution speed. This network is a CNN that estimates the distance of each pixel to the camera from an RGB image. Additionally, to evaluate results, the DIODE dataset [49] was used, since it includes varied environments and types of obstacles and has dense ground truth measurements over a large range of distances (from 0.6 to 350m).

The output of the CNN corresponds to a relative and inverted depth map. It is then necessary to align it with the ground truth before analyzing the errors obtained. This procedure is described in Li and Snavely [50]. Several points from the ground truth depth map are selected, and from their corresponding points in the monocular depth map, two correction factors necessary to adjust the monocular estimate are identified. These are the scale and shift factors, a and b respectively. These corrections are then applied and the monocular estimate is aligned to the ground truth depth measurements. In summary, this process corresponds to the following steps:

1. Invert the ground truth.
2. Align the monocular estimate with points from the inverted depth camera measurements using the least squares method, according to the following:

$$\text{inverted ground truth} = a \times \text{monocular estimate} + b$$

3. Invert the aligned monocular estimate in order to obtain the depth in meters.

An analogous procedure for the monocular depth map alignment can also be followed in a real-life situation, simply replacing the ground truth with some other measurements, in the case of this thesis, the stereo depth map obtained by the depth camera.

A comparison between the monocular estimate before and after the alignment is exemplified in Figure 3.2. In this figure, yellow tones represent larger distances, while blue tones indicate shorter distances. The depth map of Figure 3.2 (b) is inverted in regards to the ground truth, and relative, that is, it doesn't contain metric information. The depth map of Figure 3.2 (c), after alignment, matches the ground truth in Figure 3.2 (d) in terms of scale much more closely.

3.1.2 Sky Segmentation

Often, the CNN depth predictions in sky areas will be incorrect, and if those areas weren't removed there could be false obstacle detections. The adopted procedure for sky segmentation was the one described in Mashaly [51], due to its fast execution speed. The monocular estimate before and after the sky removal is exemplified in Figure 3.3.

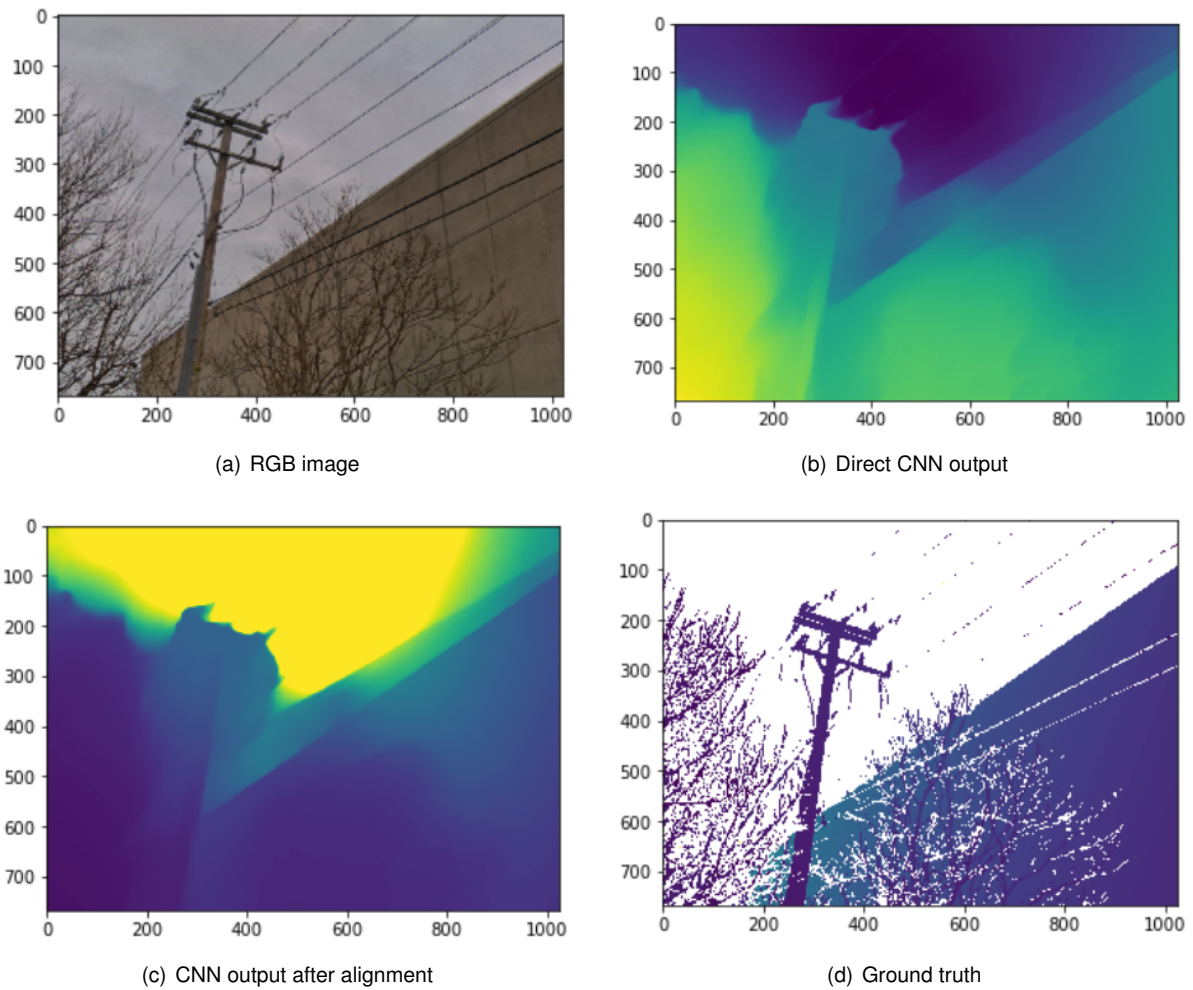


Figure 3.2: Comparison of CNN depth prediction before and after alignment with the ground truth.

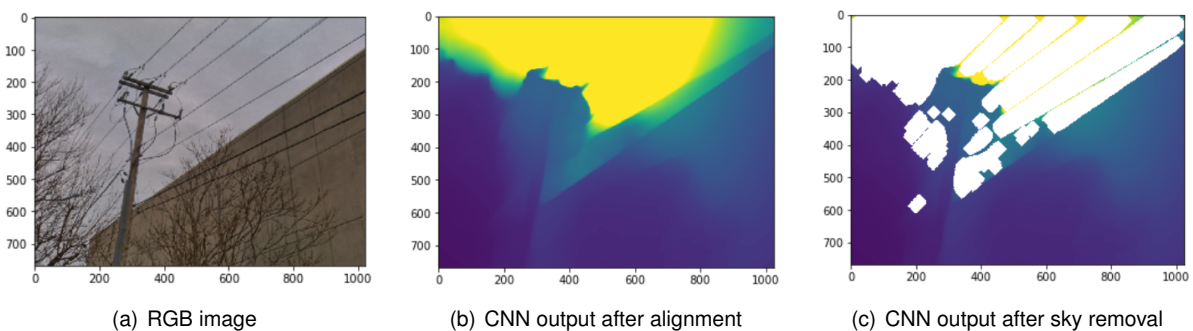


Figure 3.3: Comparison of CNN depth prediction before and after sky removal.

3.1.3 Fusion of Stereo and Monocular Depth Estimates

The algorithm for the fusion of the two individual depth estimates was adapted from Martins et al. [11]. The changes made arise from taking into account that in this case the two estimates do not have the same range of distances, since the stereo estimate is bound by the depth camera limitations in terms of range, while the monocular estimate is not. For pixels for which only one depth estimate, monocular or stereo, is available, that estimate is preserved in the final depth map. When for a given pixel both the stereo and the monocular depth estimates are available, the fusion algorithm can be summarized by the

following points:

1. When the stereo estimate for a given pixel is considered reliable, it is preserved.
2. When the stereo estimate for a given pixel is missing, the monocular estimate is preserved.
3. When the stereo estimate for a given pixel isn't considered reliable:
 - (a) if the two depth estimates are dissimilar then the monocular estimate is trusted more,
 - (b) otherwise the stereo estimate is trusted more.

In practice, these rules correspond to the following equation

$$Z_{(x,y)} = W_{c(x,y)} \times Z_{s(x,y)} + \left(1 - W_{c(x,y)}\right) \cdot \left[\left(1 - W_{s(x,y)}\right) \times Z_{m(x,y)} + W_{s(x,y)} \times Z_{s(x,y)} \right] \quad (3.1)$$

where $Z_{(x,y)}$ is the final depth estimate of pixel (x, y) , $Z_{m(x,y)}$ and $Z_{s(x,y)}$ are the monocular and stereo estimates of pixel (x, y) , respectively, $W_{c(x,y)}$ is a weighting factor dependent on the confidence of the stereo map at pixel (x, y) , and $W_{s(x,y)}$ is a weighting factor dependent on the ratio between the monocular and stereo estimates at pixel (x, y) .

To calculate the first weighting factor, $W_{c(x,y)}$, it was considered that since the stereo matching operation works well for vertical edges, the confidence of the stereo depth estimate at a certain pixel is related to the distance of that pixel to the closest edge in the image.

These edges were identified by applying a Canny filter [52] to the image. The output of this filter is 1 if the pixel belongs to an edge, and 0 otherwise, as is represented in Figure 3.4 (b).

The weighting factor $W_{c(x,y)}$ is given by

$$W_{c(x,y)} = \frac{1}{1 + e^{0.25 \times d}} \quad (3.2)$$

where d corresponds to the distance in pixels between pixel (x, y) and the closest edge, until a maximum distance of 5. A graphic representation is displayed in Figure 3.4 (c).

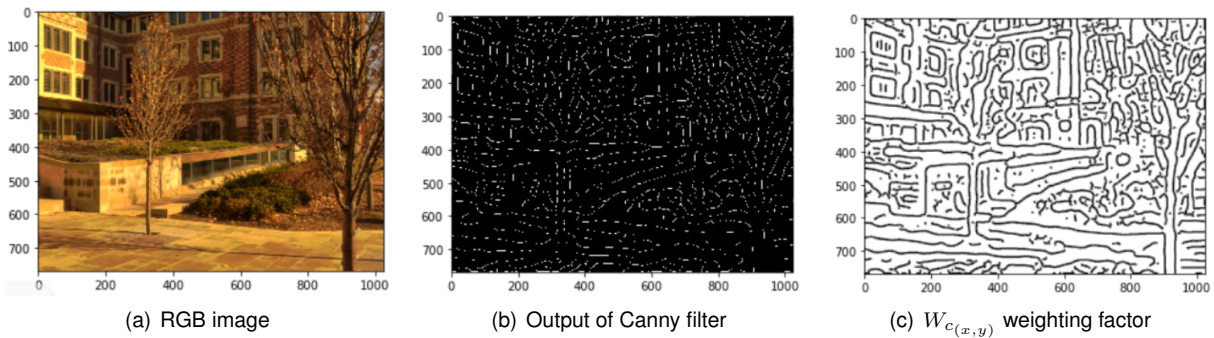


Figure 3.4: Graphic representation of the steps involved in the attainment of the $W_{c(x,y)}$ weighting factor.

In turn, the weighting factor $W_{s(x,y)}$ is simply given by

$$W_{s(x,y)} = \begin{cases} \frac{Z_m(x,y)}{Z_s(x,y)} & \text{if } Z_s(x,y) > Z_m(x,y) \\ \frac{Z_s(x,y)}{Z_m(x,y)} & \text{if } Z_s(x,y) < Z_m(x,y) \end{cases} \quad (3.3)$$

As stated earlier, the output of the CNN corresponds to a relative and inverted depth map. Therefore, before the two depth estimates can be merged, the monocular depth estimate has to be scaled according to the depth camera measurements. The adopted procedure corresponds to the one detailed in section 3.1.1, where the ground truth data is substituted by the depth camera measurements.

3.2 Detection and Localization of an Intruder UAV

The monocular and stereo vision fusion method presented in the previous section does not yield good results for the distance estimation of an intruder UAV, since “floating objects” were not contemplated in the training of the monocular depth estimation network. Therefore, a different method is needed to localize intruder UAVs. Since the type of depth cameras that can be installed in UAVs have severe limitations in terms of range, it was decided to use several cooperative UAVs, each equipped with a monocular camera, for this task. Figure 3.5 illustrates the proposed architecture for the localization of a target UAV.

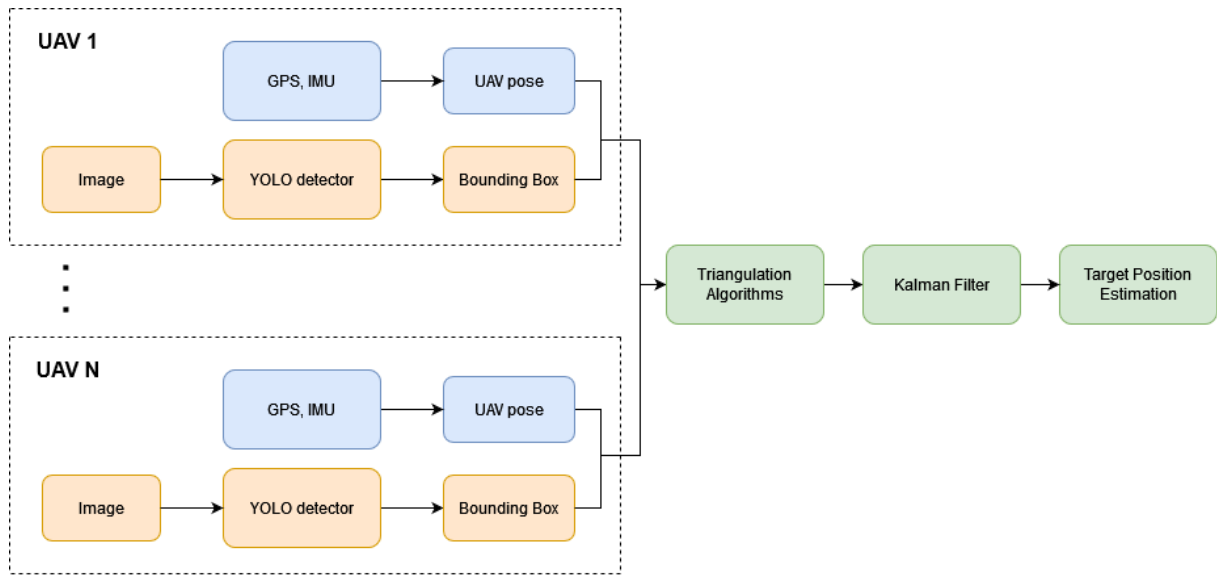


Figure 3.5: Proposed architecture for the target localization method.

First, for each cooperative UAV that sees the target, the YOLOv3 [24] algorithm provides the respective target’s image coordinates. Then that information, along with the position and orientation of each cooperative UAV, is used by a set of triangulation algorithms in order to estimate the target’s location. To improve the estimated intruder UAV position, a Kalman filter was used after the triangulation step, resulting in the final estimated target position.

3.2.1 Visual Object Detection

Two criteria were used to select the visual detection algorithm to perform the target UAV's detections: accuracy and processing speed. To construct a real-time solution, the latter is required. The YOLOv3 algorithm was chosen because it offered a good compromise between both factors [53].

A PyTorch implementation of the algorithm¹ was adopted. During the training of the network, a technique known as transfer learning was used. Transfer learning is a machine learning technique in which a model created for one task is used as the starting point for a model intended for a different task. In this case, a model pre-trained on the MS COCO dataset [54], able to recognize objects of 80 different classes, was used as the starting point to train the network to be able to detect target UAVs.

The Detfly dataset [40], which consists of more than 13,000 labeled images of a flying target UAV (DJI Mavic2), was chosen for training. This dataset was selected because it includes a variety of realistic scenarios with an assortment of background scenes, viewing angles, relative distances, flying altitudes, and lightning conditions. Figure 3.6 presents some example images from this dataset. The images were divided into training and validation sets in an 80/20 split.

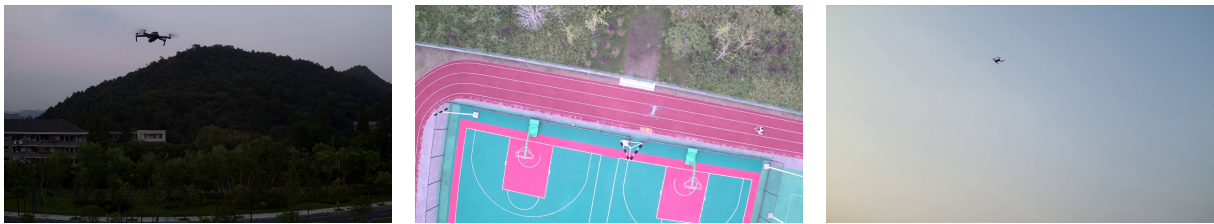


Figure 3.6: Example images from the Detfly dataset [40].

3.2.2 Triangulation Algorithms

Triangulation algorithms [55] deal with the problem of finding the position of a point $\mathbf{x} \in \mathbb{R}^3$ given its projection $\mathbf{u}_1, \dots, \mathbf{u}_n \in \mathbb{R}^2$ in n images taken with cameras with known calibration and pose, that is, with known camera matrices $P_1, \dots, P_n \in \mathbb{R}^{3 \times 4}$. For the triangulation to be possible, at least two non-collinear detections are necessary.

In the absence of noise, the triangulation problem is trivial, since all rays will intersect. In practice however, the rays will not generally meet in one single point. This may be due to uncertainties in relative camera poses or intrinsics (i.e. errors in P_i), or to errors in the detection pixel coordinates (i.e. errors in \mathbf{u}_i). Different methods exist that try to find the best point of intersection. In this thesis, three will be discussed: the linear method, the midpoint method and the L2 method.

- **Linear Triangulation**

The linear triangulation algorithm, described in Hartley and Sturm [55] and Hartley et al. [56], is a simple and efficient method to solve the triangulation problem.

¹<https://github.com/ultralytics/yolov3>

The projection of a point in space into an image plane is can be expressed in homogeneous coordinates by

$$\tilde{\mathbf{u}} = w(u, v, 1)^T \quad (3.4)$$

where (u, v) are the observed point coordinates in the image and w is an unknown scale factor. If we denote by \mathbf{p}_i^T the i th row of the matrix P , then the projection equation for the pinhole camera $\tilde{\mathbf{u}} = P\tilde{\mathbf{x}}$ can be written as

$$\begin{cases} wu = \mathbf{p}_1^T \tilde{\mathbf{x}} \\ wv = \mathbf{p}_2^T \tilde{\mathbf{x}} \\ w = \mathbf{p}_3^T \tilde{\mathbf{x}} \end{cases} \Leftrightarrow \begin{cases} u\mathbf{p}_3^T \tilde{\mathbf{x}} = \mathbf{p}_1^T \tilde{\mathbf{x}} \\ v\mathbf{p}_3^T \tilde{\mathbf{x}} = \mathbf{p}_2^T \tilde{\mathbf{x}} \end{cases} \quad (3.5)$$

where the simplification comes from eliminating w using the third equation.

In summary, each detection results in two linearly independent equations. All these equations can be combined in the form

$$A\tilde{\mathbf{x}} = 0 \quad (3.6)$$

with $A \in \mathbb{R}^{2n \times 4}$.

For example, for the two-view case where the detections are given by $\tilde{\mathbf{u}} = P\tilde{\mathbf{x}}$ and $\tilde{\mathbf{u}}' = P'\tilde{\mathbf{x}}$, matrix A is given by

$$A = \begin{bmatrix} u\mathbf{p}_3^T - \mathbf{p}_1^T \\ v\mathbf{p}_3^T - \mathbf{p}_2^T \\ u'\mathbf{p}'_3{}^T - \mathbf{p}'_1{}^T \\ v'\mathbf{p}'_3{}^T - \mathbf{p}'_1{}^T \end{bmatrix} \in \mathbb{R}^{4 \times 4} \quad (3.7)$$

where two equations have been included from each detection, giving a total of four equations.

In the presence of noise, equation (3.6) does not have an exact solution. A common approach to find an approximate solution is to use the Homogeneous method [44], which minimizes $\|A\tilde{\mathbf{x}}\|$ subject to the condition $\|\tilde{\mathbf{x}}\| = 1$. This problem can be solved using Single Value Decomposition (SVD) [57].

Although this method is simple and computationally fast, solving the least square solution of equation (3.6) can lead to large errors, since the minimized algebraic error is not geometrically meaningful [58].

• Midpoint Triangulation

The midpoint triangulation algorithm is described in Beardsley et al. [59] for the two-views case, and further extended for the general case of n -views in Ramalingam et al. [60].

For every point of view $i \in \{1, \dots, n\}$, we can construct a detection ray that starts at the camera position $\mathbf{c}_i = -R_i^T \mathbf{t}_i$ and passes through a point $\mathbf{v}_i \in \mathbb{R}^3$ given in homogeneous coordinates by $\tilde{\mathbf{v}}_i = \tilde{P}_i^{-1} \tilde{\mathbf{u}}_i$. We can write this detection ray as

$$\mathbf{r}_i(t_i) = \mathbf{c}_i + t_i \mathbf{d}_i \quad (3.8)$$

with $t_i \geq 0$ and the direction vector \mathbf{d}_i given by

$$\mathbf{d}_i = \frac{\mathbf{v}_i - \mathbf{c}_i}{\|\mathbf{v}_i - \mathbf{c}_i\|} \quad (3.9)$$

The midpoint triangulation algorithm determines the point $\hat{\mathbf{x}}$ which is closest on average to all rays, that is

$$\hat{\mathbf{x}} = \underset{\hat{\mathbf{x}}}{\operatorname{argmin}} \sum_{i=1}^n d(\hat{\mathbf{x}}, \mathbf{r}_i)^2 \quad (3.10)$$

where $d(*, *)$ denotes the Euclidean distance between a point and a line. In the two-view case, $\hat{\mathbf{x}}$ corresponds to the midpoint of the common perpendicular to the two rays.

Ramalingam et al. [60] provides a closed-form solution for $\hat{\mathbf{x}}$ via

$$\hat{\mathbf{x}} = \frac{1}{n} (I_3 + DD^T A) \sum_{i=1}^n \mathbf{c}_i - A \sum_{i=1}^n \mathbf{d}_i \mathbf{d}_i^T \mathbf{c}_i \quad (3.11)$$

where n is the number of detections, I_3 is the 3×3 Identity matrix, $D = [\mathbf{d}_1 | \dots | \mathbf{d}_n] \in \mathbb{R}^{3 \times n}$ and the matrix $A \in \mathbb{R}^{3 \times 3}$ is given by

$$A = (nI_3 - DD^T)^{-1} \quad (3.12)$$

The triangulation of a 3D point with the midpoint algorithm can then be carried out very efficiently, using only matrix multiplications and the inversion of a symmetric 3×3 matrix.

Unlike the linear method, the midpoint method minimizes an error that has physical meaning. However, since it operates in the Euclidean space, it does not recognize the projective properties of pinhole cameras. Therefore, compared to methods that minimize the reprojection error, it might have a worse performance in situations when some cameras are much closer to the target than others.

• L2 Triangulation

The two previous triangulation algorithms presented do not take into account the projective properties of pinhole cameras. In contrast, the L2 triangulation method, outlined in Sturm and Hartley [61] and Chen et al. [58], operates in the two-dimensional space of the image planes.

This method makes use of the fact that the image points \mathbf{u}_i are likely to be in the correct area of the image plane, that is, they are a small distance from the ground truth detections. The goal of the L2 algorithm is then to find the $\hat{\mathbf{x}}$ that minimizes the reprojection error, which corresponds to

$$\hat{\mathbf{x}} = \underset{\hat{\mathbf{x}}}{\operatorname{argmin}} \sum_{i=1}^n d(\mathbf{u}_i, \hat{\mathbf{u}}_i)^2 \quad (3.13)$$

where $d(*, *)$ denotes the Euclidean distance between two points, and $\hat{\mathbf{u}}_i = P_i \hat{\mathbf{x}}$.

For two views, in Hartley and Sturm [55] the minimalization problem (3.13) is reformulated to finding the roots of a polynomial of degree six, but for the general case with $n \geq 2$ no linear solution exists. In the general case, equation (3.13) can be rewritten as

$$\hat{\mathbf{x}} = \operatorname{argmin}_{\hat{\mathbf{x}}} \sum_{i=1}^n \left(\mathbf{u}_i - \frac{\mathbf{p}_{i_{1,2}} \hat{\mathbf{x}}}{\mathbf{p}_{i_3} \hat{\mathbf{x}}} \right)^2 \quad (3.14)$$

where $\mathbf{p}_{i_{1,2}}$ is the matrix containing the 1st and 2nd rows of matrix P_i and \mathbf{p}_{i_3} the 3rd row of the matrix P_i .

Equation (3.14) corresponds to a non-linear least squares problem, therefore is not solvable in a trivial matter. A common approach to solve this problem is the Levenberg-Marquardt method [62].

Because it minimizes the reprojection error, the L2 method is expected to perform better than the two methods previously mentioned in cases where some cameras are significantly closer to the target than others. However, the minimization function (3.13) contains multiple minima [63], which the Levenberg-Marquardt method is subject to. Therefore, a non-optimal solution may be found.

Chapter 4

Simulations

This chapter presents the various simulations performed in order to evaluate the method for the localization of a target UAV. It starts with an introduction on the AirSim Simulation Environment in section 4.1, and then describes the specifics of each simulation in section 4.2.

4.1 AirSim Simulation Environment

Due to the need for a photorealistic environment, the AirSim (Aerial Informatics and Robotics Simulation) platform [64], developed by Microsoft, was chosen to perform the simulations required to test the algorithms presented in section 3.2.2. AirSim is an open-source simulator for drones and ground vehicles, built on Unreal Engine. It supports physically and visually realistic simulations, necessary for the use of a visual detector like YOLO.

One of the main purposes of AirSim is to function as a platform for AI research, deep learning and computer vision. It also has the great benefit of hardware abstraction, making it possible to control the simulated UAVs using simple custom Python or C++ scripts. This makes it very easy to define trajectories and velocities for each UAV that respect its kinematic and dynamic constraints, for example. Scripts can also be used to define what data should be recorded during the simulation, like the images captured by the cameras and the pose of each UAV, as well as to start and stop the recording automatically.

4.1.1 Simulated Environment

Since AirSim was developed as an Unreal plug-in, it can be used with any Unreal environment. Rather than creating an environment from scratch, the environment from the package *City Park Environment Collection*¹ was used for all simulations. Two example scenes from this environment are pictured in Figure 4.1. It simulates a park and includes realistic features that are important for use with an object detector like YOLO, for example trees and plants moving with the wind, clouds, and realistic lighting.

Since YOLO was trained on a dataset that features the DJI Mavic 2, the default UAV mesh present in AirSim was altered to look like the DJI Mavic 2 as well. The simulated UAV is pictured in Figure 4.2.

¹www.unrealengine.com/marketplace/en-US/product/city-park-environment-collection



Figure 4.1: Two examples of scenes from the City Park simulated environment.



Figure 4.2: Simulated UAV.

4.1.2 Data Collection

The recording feature of AirSim allows the recording of data such as position, orientation, and velocity, as well as the images captured by each camera, at specified intervals. It also allows the recording of segmentation images, which correspond to the RGB image seen by each camera, yet each object is assigned a specific color.

For all simulations, it was recorded the position and orientation of all cooperative UAVs, the position of the target, and the RGB images as seen by the front facing camera of each cooperative UAV. This data was recorded with a frequency of about 4.5Hz. To make the simulation more realistic, error was added to the pose of each cooperative UAV, simulating GPS and IMU inaccuracies. A zero-mean Gaussian error with a standard deviation of 0.5m was added to the UAVs' position, and a zero-mean Gaussian error with a standard deviation of 2° was added to the UAVs' orientation.

For the evaluation of the YOLO detector performance, segmentation images of the target were also recorded, of which an example is pictured in Figure 4.3 (a). In these images the target appears as white against a dark background, which allowed a script to be developed that obtains the accurate coordinates of the target in each image, that later served as ground truth for the evaluation of YOLO on AirSim data. Figure 4.3 (b) pictures the color image that corresponds to 4.3 (a), overlapped with a red box that indicates the ground truth coordinates of the target in the image.

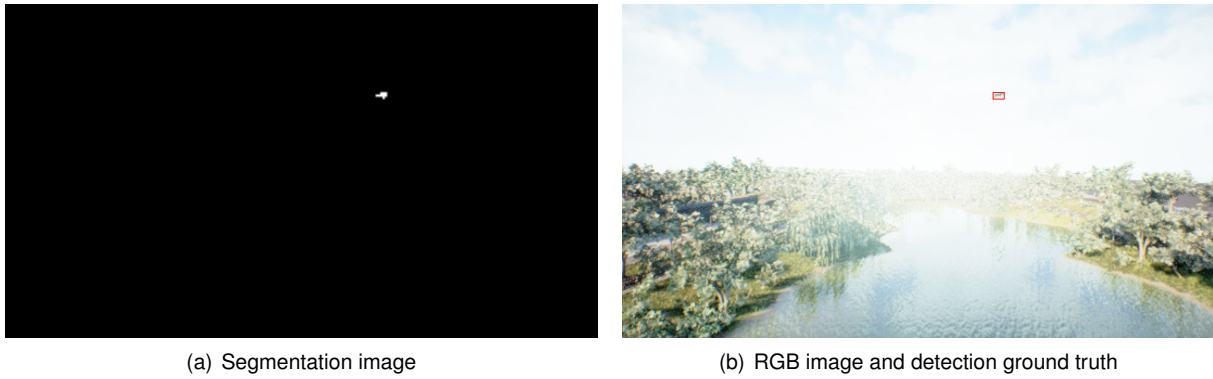


Figure 4.3: Segmentation and corresponding RGB images, captured by the same camera. From the segmentation image the ground truth for YOLO detections can be obtained (red box).

4.2 Trajectories/AirSim Simulations

This section presents a brief explanation of the different simulations that were executed in the AirSim simulator. A simplified diagram of the types of simulations that were performed is presented in Figure 4.4. In order to test the influence of different factors on the performance of the algorithms, three trajectories for the target and cooperative UAVs were created. These fall under two distinct categories: *moving target*, where the cooperative UAVs are stationary in relation to the environment and the target performs a defined trajectory, and *moving sensors*, where the opposite is true.

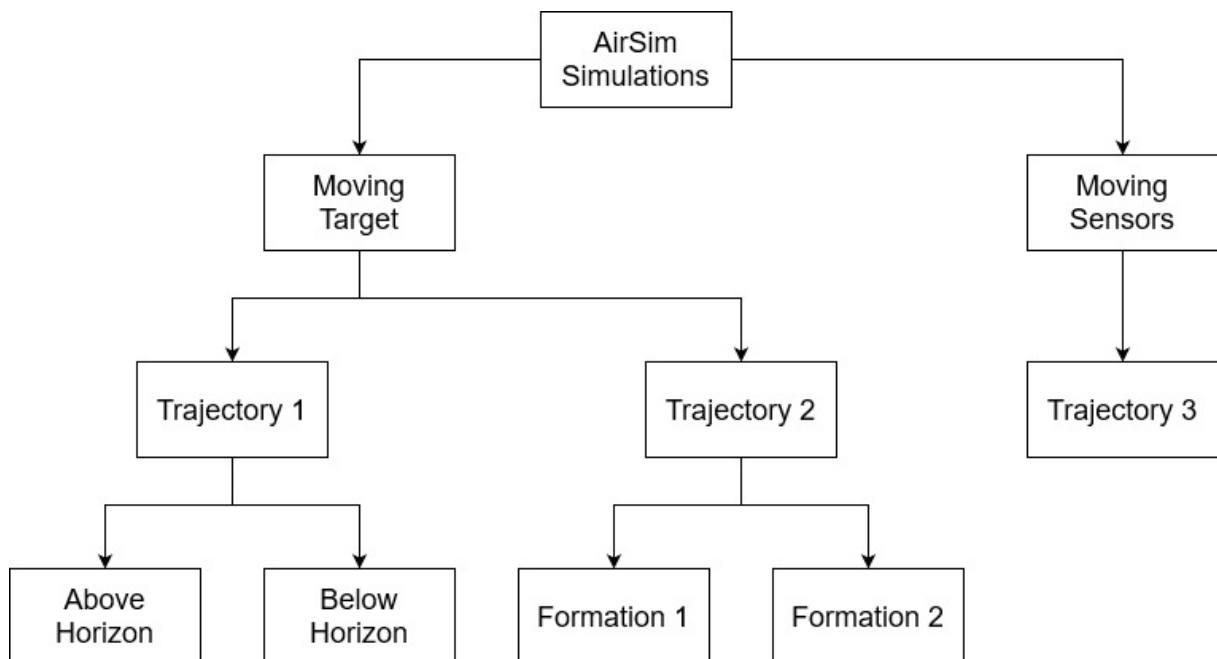


Figure 4.4: Diagram of the AirSim simulations performed.

4.2.1 Trajectory 1

The first trajectory falls under the *moving target* category, hence the two cooperative UAVs remain in hover mode, stationary relative to the surrounding environment for the duration of the simulation, while the target follows a predetermined zig-zag trajectory, flying at 5m/s. This is represented in Figure 4.5, where the numbers indicate the cooperative UAVs and the letters indicate the target's path.

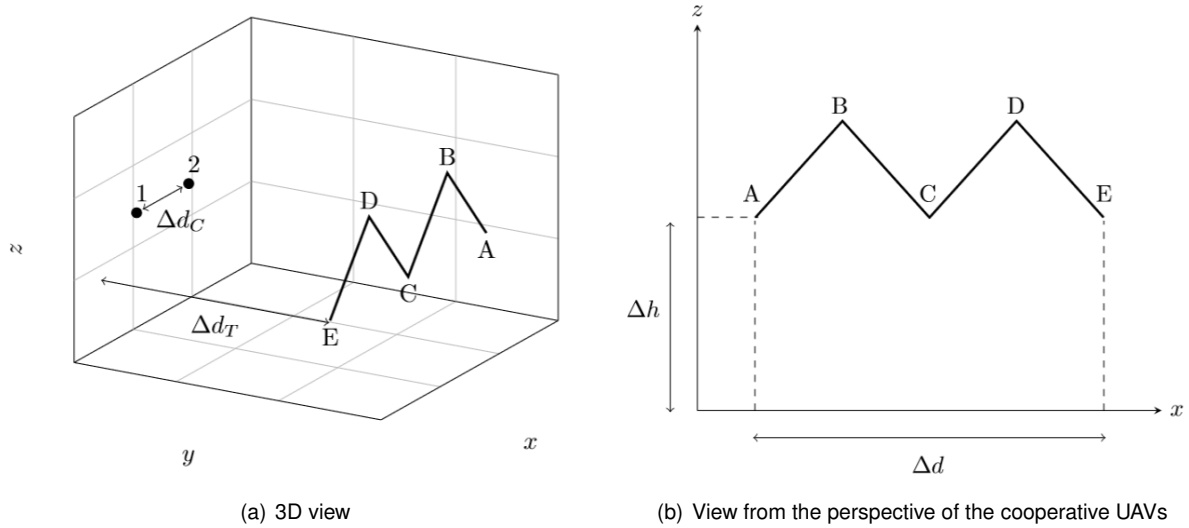


Figure 4.5: Trajectory 1, where two cooperative UAVs (1 and 2) are stationary and the target follows the path A-B-C-D-E.

The target's trajectory was designed to be parallel to the y-axis so that it keeps a constant distance to the plane that contains the cooperative UAVs, indicated in Figure 4.5 (a) by Δd_T . The simulation was then run for different values of Δd_T , ranging from 10m to 60m, in order to evaluate the relationship between the distance to the target and the errors obtained in the triangulation.

The relationship between the sensor locations and the triangulation error obtained was also studied. For this, the distance between the cooperative UAVs, indicated in Figure 4.5 (a) by Δd_C , was changed across simulations, between a range of 5m to 35m.

The target's trajectory involves a series of climbs and descents, hence the name *zig-zag trajectory*. The purpose of this trajectory is to diversify the vertical detection location of the target, that is, to ensure that the target passes through many different regions of the captured images, in order to better evaluate the YOLO performance and have the captured data be more realistic.

In Figure 4.5 (b) are represented the two trajectory parameters, Δh and Δd . Δh refers to the height that the trajectory is performed at, and was chosen for each simulation in order to keep the target detections either always above or below the horizon, with the purpose of better evaluating the YOLO performance under these two different conditions. Additionally, Δd refers to the horizontal length of the trajectory and was adjusted so that in each simulation, the target was always in the field of view of both cameras, and the cooperative UAVs could capture it during the entire maneuver.

4.2.2 Trajectory 2

The second trajectory was developed with the purpose of studying the influence of the number of cooperative UAVs on the target localization results. It also falls under the *moving target* category, and like in the previous trajectory, the cooperative UAVs remain in hover while the target follows a predetermined trajectory.

In this case, the target flies at 5m/s along a rectangular path, and for each simulation a different number of cooperative UAVs, between 2 and 5, was used. The cooperative UAVs position themselves according to two distinct configurations, called *formation 1* and *formation 2*, which are represented in Figures 4.6 and 4.7, respectively.

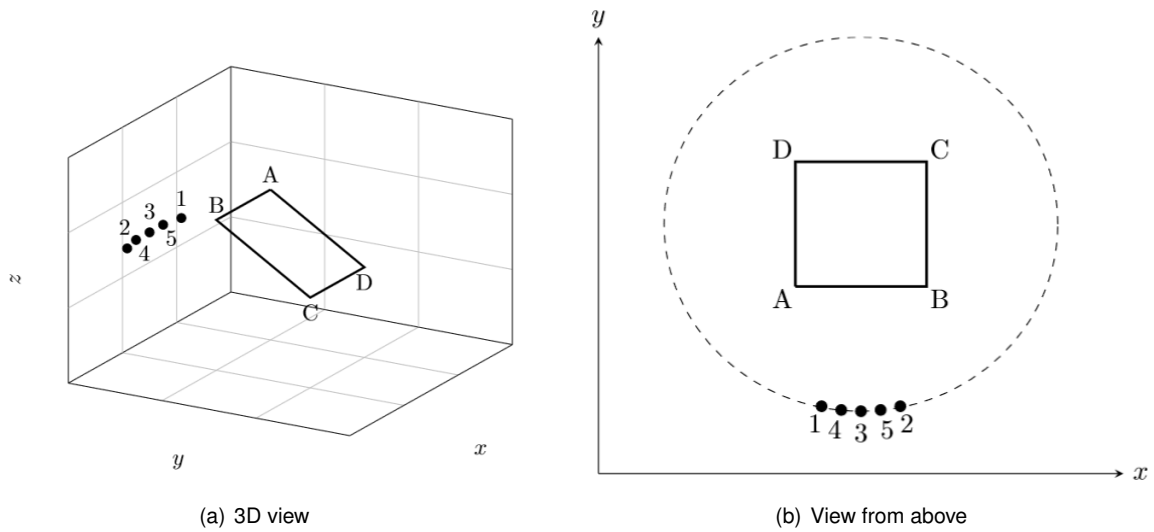


Figure 4.6: Trajectory 2 and formation 1, where the cooperative UAVs (1-5) are stationary and the target follows the path A-B-C-D-A.

In the case of formation 1, the cooperative UAVs position themselves relatively close to each other, along a circle centered around the target's trajectory. This configuration is represented in Figure 4.6. The number next to each UAV represents the order in which they are added to the group, so in the simulation with two cooperative UAVs they will be positioned in positions 1 and 2, in the simulation with three cooperative UAVs they will be positioned in positions 1, 2 and 3, and so on. UAVs 1 and 2 are positioned in the extremities of the group so that the effect of adding more viewpoints without increasing the maximum parallax angle between detections can be studied.

In the case of formation 2, represented in Figure 4.7, the cooperative UAVs position themselves equally spaced along a circle centered around the target's trajectory. Again, the number next to each UAV shows the order in which they are added to the simulation. This configuration is intended to study the effect that additional varied points of view have in the accuracy of the estimated location of the target.

4.2.3 Trajectory 3

The third and final trajectory was developed with the goal of studying the performance of the triangulation algorithms when the distances between each cooperative UAV to the target are not similar to each

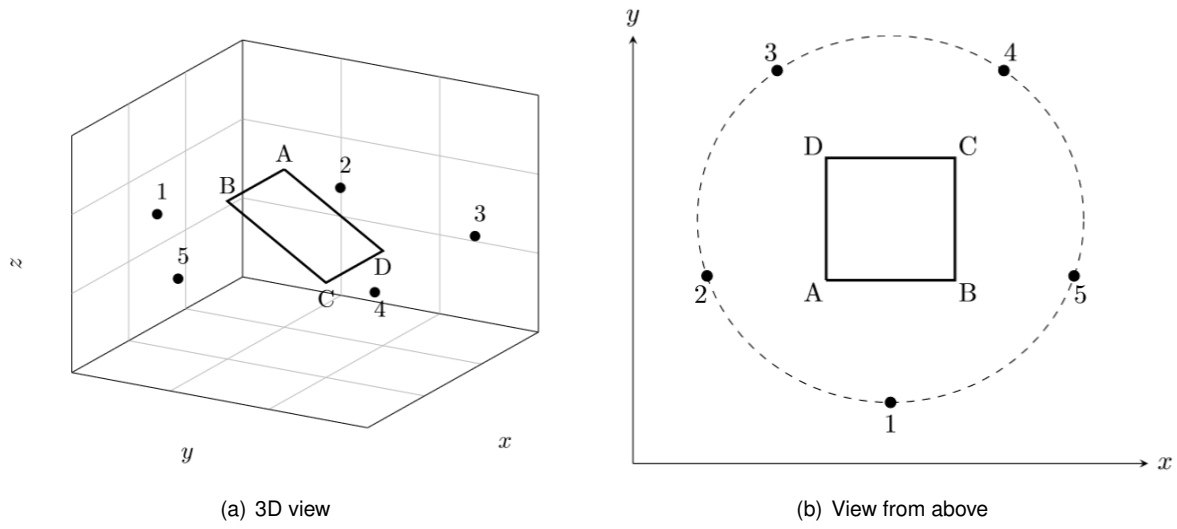


Figure 4.7: Trajectory 2 and formation 2, where the cooperative UAVs (1-5) are stationary and the target follows the path A-B-C-D-A.

other, that is, when some UAVs are much closer to the target than others. This trajectory falls under the *moving sensors* category, meaning that the cooperative UAVs move throughout the simulation, while the target remains in hover mode and is stationary in relation to the environment.

A depiction of trajectory 3 is present in Figure 4.8. As stated above, the target, represented by the letter T, remains stationary. Two cooperative UAVs start at the same distance from the target, and move according to a predetermined path: the first cooperative UAV moves from point A to point B, getting closer to the target, while the second cooperative UAV moves in the opposite direction, from point C to point D.

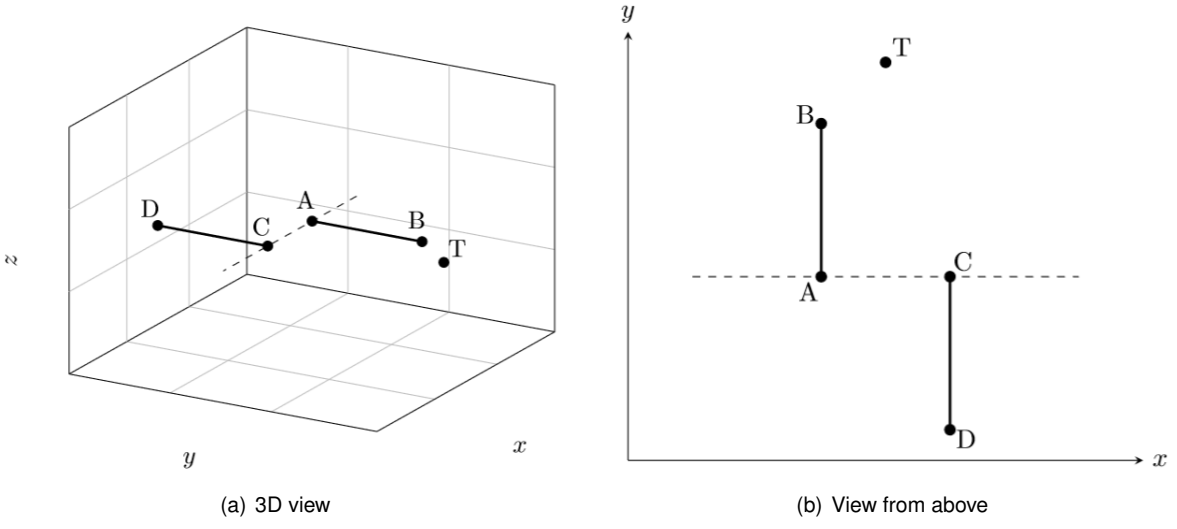


Figure 4.8: Trajectory 3, where the target (T) is stationary and two cooperative UAVs follow the paths A-B and C-D.

Chapter 5

Results

This chapter starts by presenting the results relating to the obstacle detection problem, in section 5.1. First, the performance of the monocular depth estimation network is discussed, and then the results from the monocular and stereo depth fusion are presented. Subsequently, section 5.2 approaches the outcomes of the target UAV localization. Initially, the performance of the YOLO object detector is evaluated, and follows an analysis on the results obtained for each of the trajectory simulations. This section finishes with a discussion on the overall performance of this method.

All the evaluation metrics discussed in this chapter are introduced in section 2.5.

5.1 Obstacle Detection for Small UAVs

In order to study the feasibility of using a monocular depth estimation network to complement the measurements of a depth camera, the possible output of a depth camera was simulated from the ground truth of the DIODE Dataset. For this, the ground truth measurements smaller than 10 meters, since that is a common range for depth cameras commonly used in UAVs, were considered, and a Gaussian zero-mean error with a standard deviation 0.1m was introduced.

A subset of the Diode dataset, containing 446 images, was used to evaluate the results of the fusion of stereo and monocular depth estimates. For each image, a monocular depth prediction was generated, and subsequently it was aligned to the simulated depth camera measurements according to the procedure described in section 3.1.1. Then, after the sky segmentation step, the monocular depth map and the simulated stereo depth map were combined, following the procedure detailed in section 3.1.3.

All metrics presented in this section are detailed in section 2.5.1.

5.1.1 Monocular Depth Estimation

First, the performance of the monocular depth estimation network, when the ground truth is available for alignment, was evaluated. As previously stated, the depth estimation CNN outputs a depth map that is relative and inverted in relation to the ground truth, therefore before comparing the results obtained the monocular estimate needs to be adjusted according to the steps described in section 3.1.1. Figure 5.1

depicts the depth estimation network’s performance in regards to two example images from the Diode dataset. The white regions of figures 5.1 (b) and (e) represent points for which there are no ground truth measurement available.

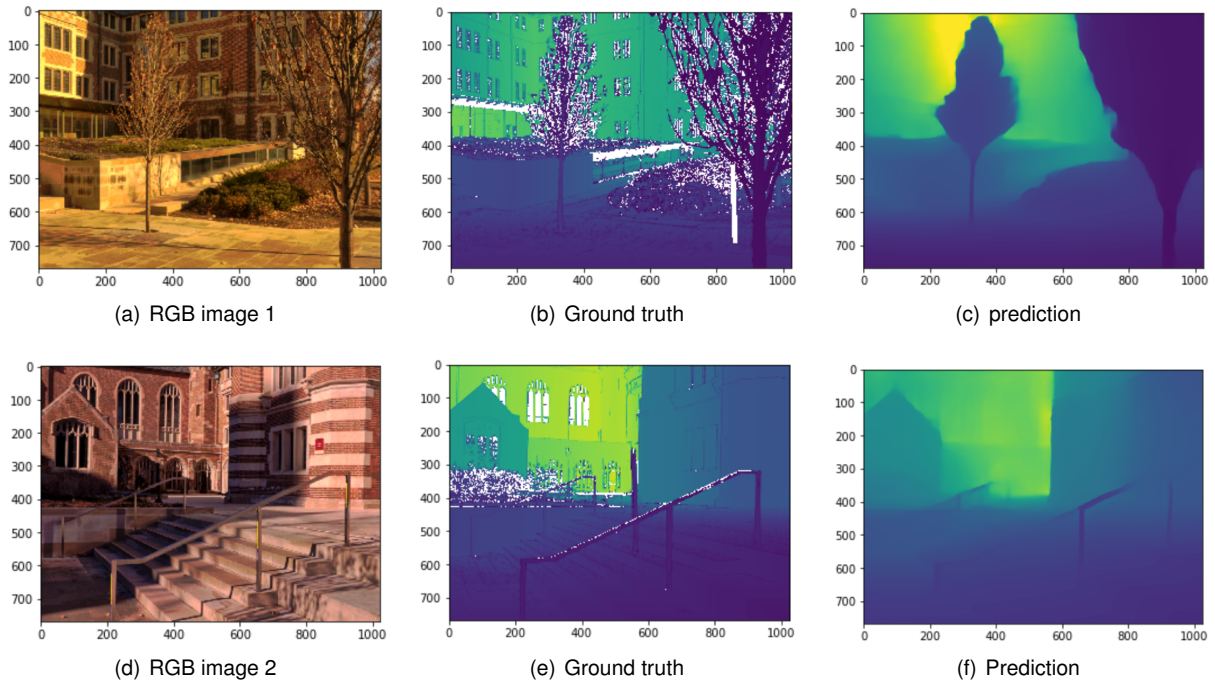


Figure 5.1: Monocular depth prediction results for two example images from the Diode Dataset [49].

The results of these two images illustrate a characteristic of the depth prediction network: the predicted depth maps are much less detailed than the original image, and small shapes with small details, like the tree branches in Image 1, appear blurred. However, for objects with clear-cut edges, like the buildings in Image 2, the contours of the predicted depth map are much more precise. Table 5.1 contains the error metrics for these two predictions. As expected the prediction for Image 1, which contains more detailed shapes, had worse results.

Table 5.1: Error metrics of the predicted depth map for two example images.

	RMSE	Absolute Relative Error	Threshold Accuracy		
			$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Image 1	8.83	0.721	0.489	0.692	0.786
Image 2	6.46	0.251	0.603	0.736	0.801

Figure 5.2 shows a graphical representation of the absolute error obtained for these two example predictions. In these figures, blue tones indicate a smaller prediction error, and yellow tones indicate the opposite. As predicted, for Image 1 the sections with the highest error correspond to the areas surrounding the tree branches, due to the imprecision of the network when dealing with small details. For Image 2 however, the region with the highest error corresponds to the furthest building. This illustrates another characteristic of the depth prediction network: even after the alignment described in section 3.1.1 it tends to significantly underestimate how close far away objects are from the camera.

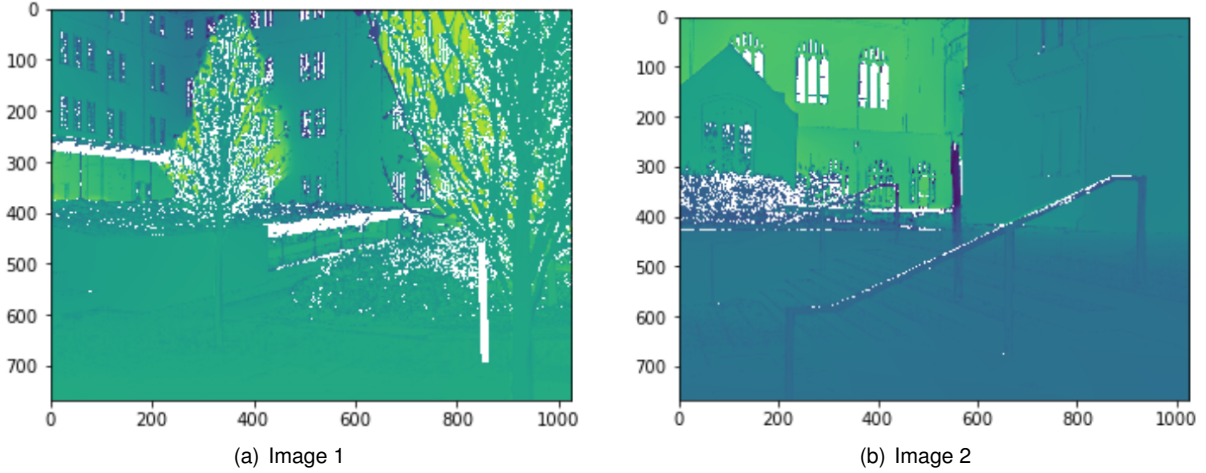


Figure 5.2: Graphic representation of the depth estimation errors obtained. Blue represents smaller errors, while yellow represents larger errors.

Table 5.2 presents the average error metrics for every image in the subset used for evaluation. As expected, the results are similar to the ones presented in Table 5.1.

Table 5.2: Average error metrics of the predicted monocular depth map.

RMSE	Absolute Relative Error	Threshold Accuracy		
		$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
8.26	0.641	0.543	0.693	0.795

5.1.2 Fusion Results

In this section, the full method for monocular and stereo depth fusion is analyzed similarly to the monocular depth estimation network in the previous section. Figure 5.3 illustrates the steps taken to arrive at the final depth prediction. From the comparison of Figure 5.3 (d), which represents the monocular estimate, with the ground truth depicted in Figure 5.3 (b), it can be exemplified that the monocular estimate has a tendency to underestimate the distance to faraway objects, like in this case the building behind the cars. This is in agreement with the results from the previous section. Moreover, in Figure 5.3 (e) it can be observed that the sky segmentation step removed two regions from the estimate wrongfully. One of the “holes” in the monocular estimate was filled in Figure 5.3 (f) since there were depth camera measurements for that area of the image, however the other “hole” remained. Situations like this happened in several instances and could probably be avoided with the use of a more precise sky segmentation algorithm.

Finally, it can be seen from the qualitative comparison of Figures 5.3 (c) and (f), depicting the depth camera simulated measurements and the final depth estimate respectively, that the final depth map is much more full than the stereo depth map.

More examples of results obtained with this method are pictured in Figure 5.4. Overall, there was an average 21.16% increase in the number of pixels with depth information for all the evaluated images.

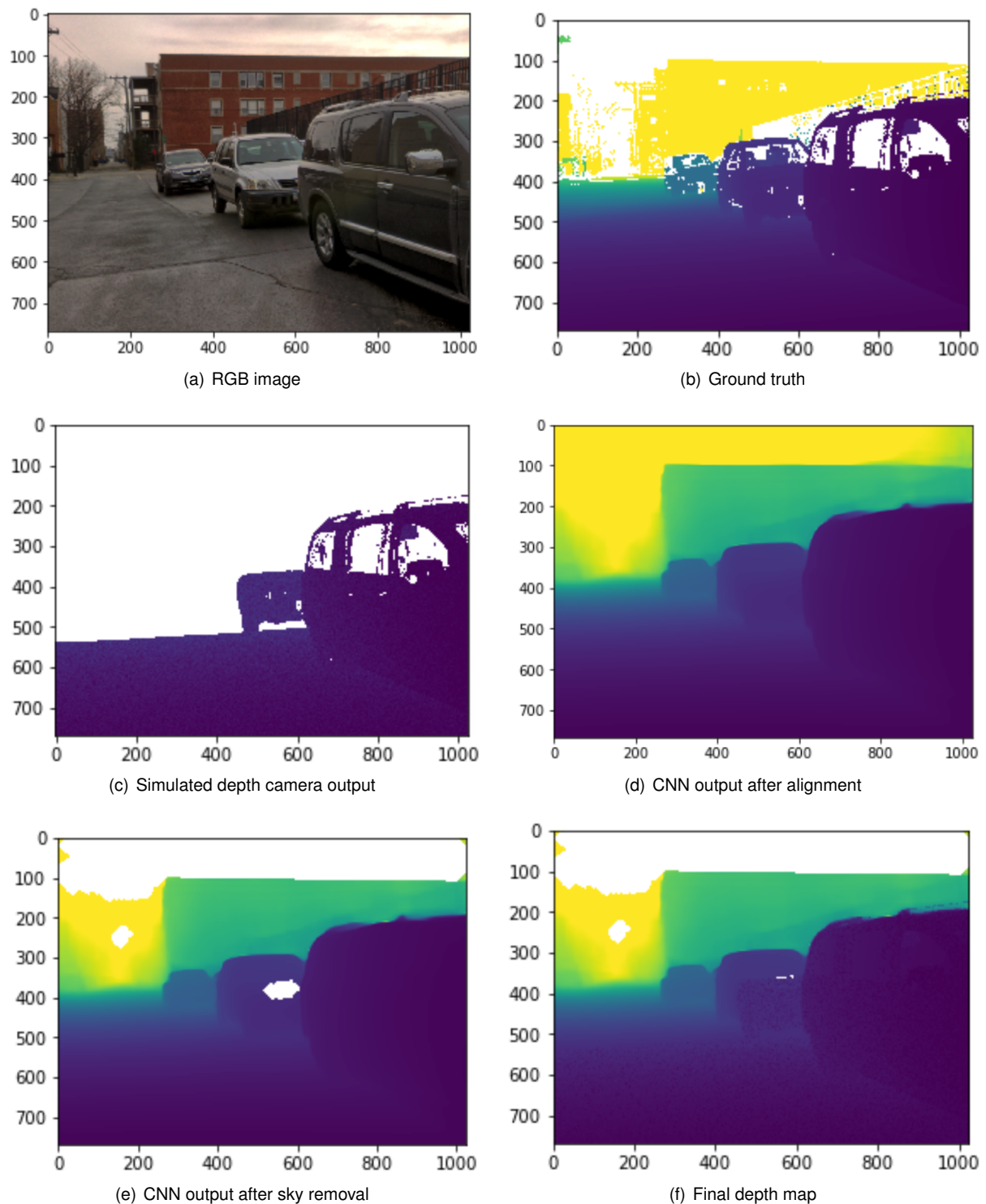


Figure 5.3: Representation of the steps taken for the obtainment of the final estimated depth map.

Again, this number could presumably be increased with the use of a more precise sky segmentation step, since in several instances pixels erroneously classified as sky are removed from the monocular estimate.

The results obtained for the set of images used for evaluation are shown in Table 5.3. The increase in the errors seen from Table 5.2 to Table 5.3 can be explained by the worse alignment of the monocular estimate in the latter case, since when only measurements of a depth camera are available, there is a

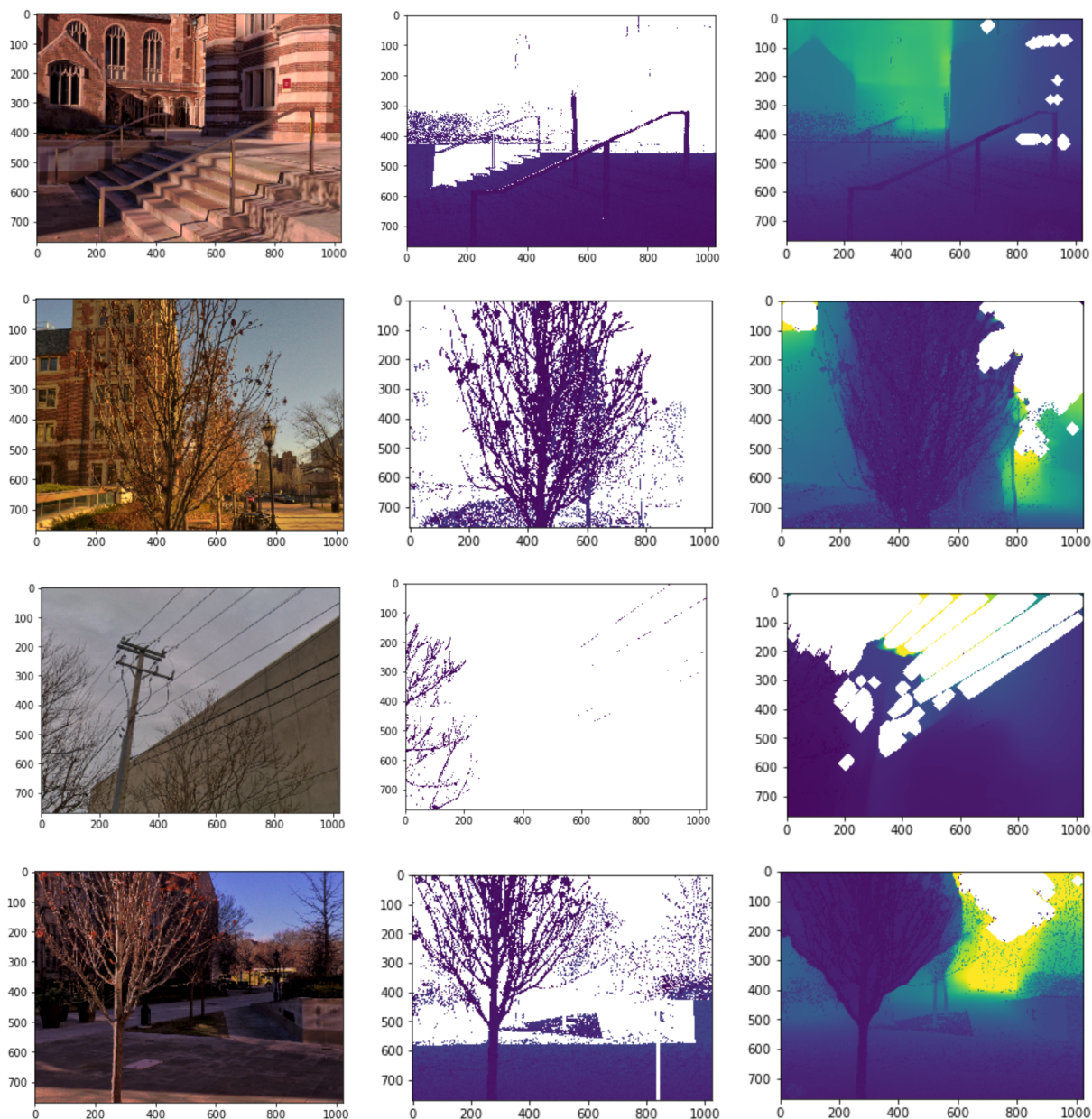


Figure 5.4: Qualitative results on some example images. From left to right, the RGB image, the simulated depth camera output and the final predicted depth map.

clear bias in the depths of the points available to perform the monocular estimate alignment.

Table 5.3: Average error metrics of the final predicted depth maps.

RMSE	Absolute Relative Error	Threshold Accuracy		
		$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
9.51	0.385	0.524	0.648	0.762

5.2 Target UAV Localization

This section presents the results obtained with the data collected during the simulations explained in section 4.2. It starts by presenting the results relative to the YOLO detector performance on the AirSim environment, in subsection 5.2.1. Then, it continues with an analysis of the influence of the distance to the target, as well as the distance between the cooperative UAVs, on the results obtained, in subsection 5.2.2, followed by a comparison of the results obtained for different numbers of cooperative UAVs, in subsection 5.2.3. Finally, the influence of the relative distance between cooperative UAVs to the target on the results obtained is also presented, in subsection 5.2.4.

5.2.1 YOLO Detector

YOLOv3, presented in section 2.2, is the algorithm responsible for detecting the target in the camera images of each cooperative UAV. As mentioned in section 3.2.1, the YOLOv3 network was trained on the Dettfly dataset [40]. It was then tested on a selection of representative data collected on the AirSim simulator, with the objective of ascertaining how well the acquired knowledge translated into the simulated environment. The collected data included images with the target UAV in various positions on the screen and at various distances away from the camera, as well as the true position of the target so that the detections could be evaluated.

The following F1-score and precision-recall (P-R) curves were obtained, and are presented in Figure 5.5 (a) and (b), respectively.

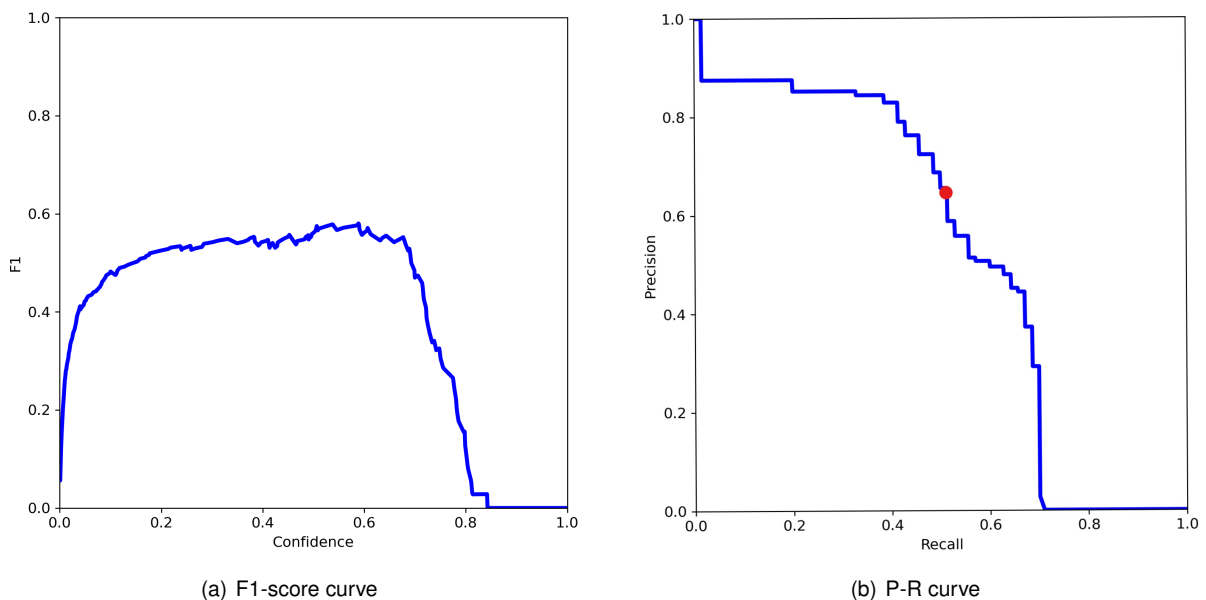


Figure 5.5: YOLO detector F1 score and P-R curves, with the highest F1-score point represented by a red dot.

The F1-score, as explained in detail in section 2.5.2, is a measure of a model's accuracy on a dataset. Figure 5.5 (a) represents the F1-score of the model as a function of its confidence threshold. As can be seen from the figure, the highest F1-score achieved is 60%, for a confidence value of 0.6.

Therefore it can be concluded that this is the confidence threshold value for which the model has the best performance.

This point, indicated by the red dot in Figure 5.5 (b), corresponds to a precision of 65% and a recall of 53%.

In order to compare the performance of the YOLO detector in the cases where the target is above or below the horizon, simulations were run following the *trajectory 1* template, explained in detail in section 4.2.1. In these simulations, two cooperative UAVs, separated 10m from each other, captured images of the target as it followed a zig-zag trajectory. The distance from the target to the cooperative UAVs was varied between 10m and 60m, and the height at which the target flew was chosen in order to keep the target either always above or below the horizon. The results obtained are represented in Figure 5.6.

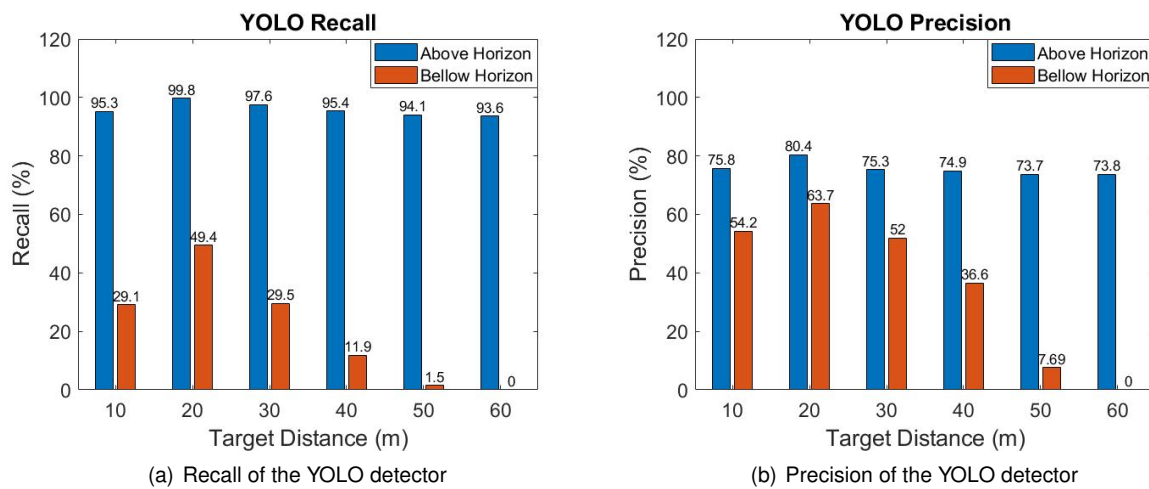


Figure 5.6: Performance of the YOLO detector on the *trajectory 1* simulations.

This figure presents a stark contrast between the performance of the YOLOv3 algorithm when the target is above or below the horizon, both in terms of recall and precision. In the above-horizon simulations the recall was above 90% across all target distances, and therefore the algorithm could detect the target in the majority of cases. However, in the below-horizon simulations there was a dramatic decrease in true target detections, especially as the distance to the target increased. At the highest distance of 60m, the target was never detected below ground.

Contrary to what might be expected, the recall for a target distance of 10m is smaller than for a larger distance of 20m. This might be a consequence of the lack of images in the Dettfly Dataset that depict the UAV at closer distances to the camera, compared to images where the UAV is farther away.

As for the differences in precision, in both experiments the cameras were in the same pose, therefore the higher precision in the above horizon case comes from the higher number of true positives that were detected, that is, since the target was correctly detected more times, the false detections were much more diluted among the true ones.

The results of Figure 5.6 (b) might lead to the conclusion that, in the below-horizon scenario, YOLO had more false detections than in the above-horizon scenario. However, a closer inspection reveals that in both cases, since the cameras were in the same pose, YOLO mistook similar amounts of ground clutter

for the target. The difference in precision might then be explained by the lower number of detections, true and false, in the below-horizon simulations, that caused the false detections to be a higher percentage of the detections overall.

Due to the much higher performance of the YOLO detector in the above-horizon scenario, all subsequent simulations were made with the target above the horizon for all the cameras involved.

5.2.2 Trajectory 1 Simulations

As explained in section 4.2.1, the *trajectory 1* simulations had the cooperative UAVs stationary, while the target followed a predetermined zig-zag trajectory. The simulation was repeated several times with the target at different distances from the cooperative UAVs, as well as with the two cooperative UAVs at different distances apart from each other.

After the detection data was collected for each simulation, the triangulation algorithms were run 20 times with a random position and orientation error, as detailed in section 4.1.2, and the RMSE for the estimated target position was collected. Figure 5.7 shows the average RMSE obtained with each of the triangulation algorithms considered, for each simulation. So as to increase the readability of the graphs, the z-axis is in a logarithmic scale.

As we can see from the figure, when the distance between the two cooperative UAVs is small in comparison to the distance between the two UAVs and the target, the error obtained is very large. Therefore, to accurately position a target that is far away, the cooperative UAVs should be as far away from each other as possible, in order to capture varied points of view.

In almost all simulations, the midpoint triangulation algorithm obtained the best results. The linear triangulation algorithm was the second best, closely followed by the L2 triangulation algorithm. However, the difference is the most relevant in the yellow portions of the graphs, where the target is located far away from the cooperative UAVs and these are positioned close together. Here, the midpoint algorithm clearly outperformed the other two. In the worst case, with the cooperative UAVs separated by only 5m, and 60m away from the target, the error obtained with the midpoint algorithm (53.4m) was less than half the error obtained with the linear (158.4m) or L2 (172.9m) algorithms. Although these errors are too large for any real application in these conditions, they serve to illustrate some of the differences between the three triangulation strategies.

In the conditions corresponding to the blue areas of the graph, an application of these algorithms is much more realistic. In the best case scenario, the linear, midpoint and L2 algorithms obtained an average RMSE of 1.43m, 1.42m and 1.41m respectively.

Figure 5.8 depicts an example of the estimated target trajectory, for the case where the distance to the target was 30m, and the cooperative UAVs were separated by 15m. This example illustrates how the obtained error is distributed: the estimated target position is much more accurate in the xz-plane, which corresponds to the camera's image plane, than in the y-direction, which corresponds to the direction that the cameras are facing. Judging from these results, the position estimate along the y-axis could possibly be improved with the addition of target detections from different points of view in regards to the target's

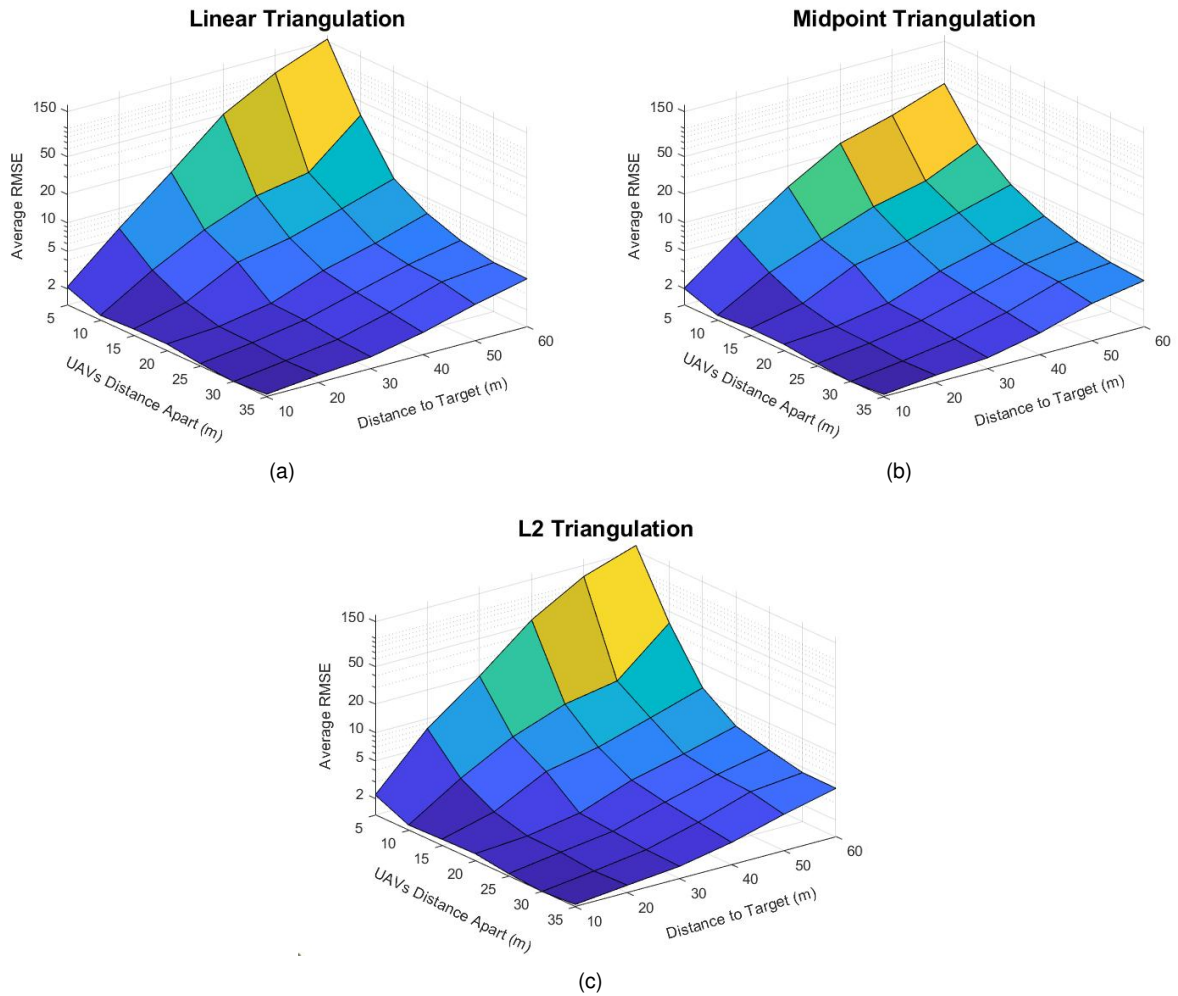


Figure 5.7: Average RMSE obtained as a function of the distance to the target, and the distance between the two cooperative UAVs, for each algorithm: (a) linear triangulation, (b) midpoint triangulation and (c) L2 triangulation.

position in the y-axis.

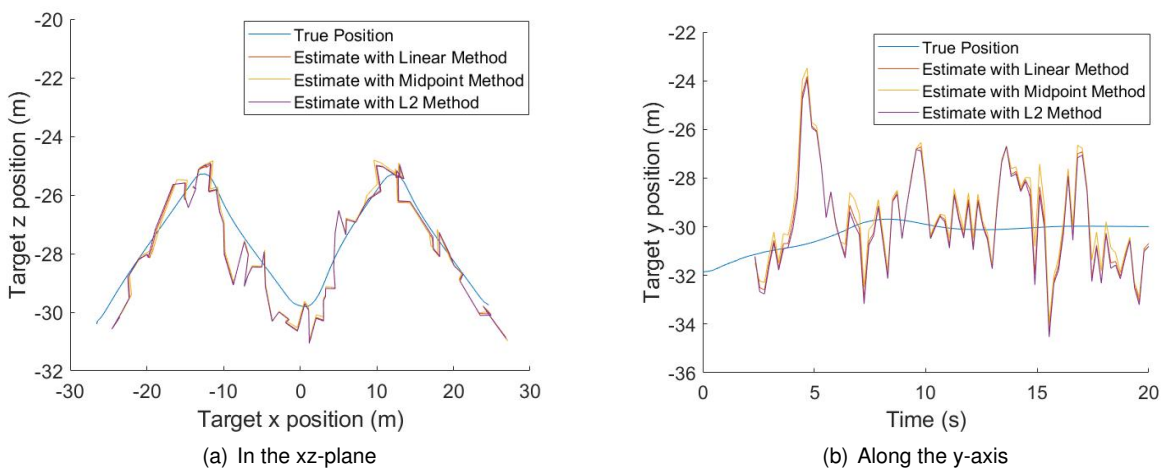


Figure 5.8: Real and estimated target trajectory for the *trajectory 1* simulation where the distance to the target was 30m, and the cooperative UAVs were separated by 15m.

5.2.3 Trajectory 2 Simulations

The effect of the number of cooperative UAVs on the obtained results was also studied, making use of the *trajectory 2* simulations detailed in section 4.2.2. As explained previously, all cooperative UAVs were stationary for the duration of the simulation, while the target followed a predetermined rectangular path. Configurations with a different number of cooperative UAVs, positioned according to two different dispositions, were examined. The target was at an average distance of 30m from the cooperative UAVs.

Similarly to the previous case, the results for this experiment were obtained by running the triangulation algorithms 20 times with a random position and orientation error for the cooperative UAVs, and the RMSE for the estimated target position during the trajectory was collected. The results obtained are presented in Figure 5.9.

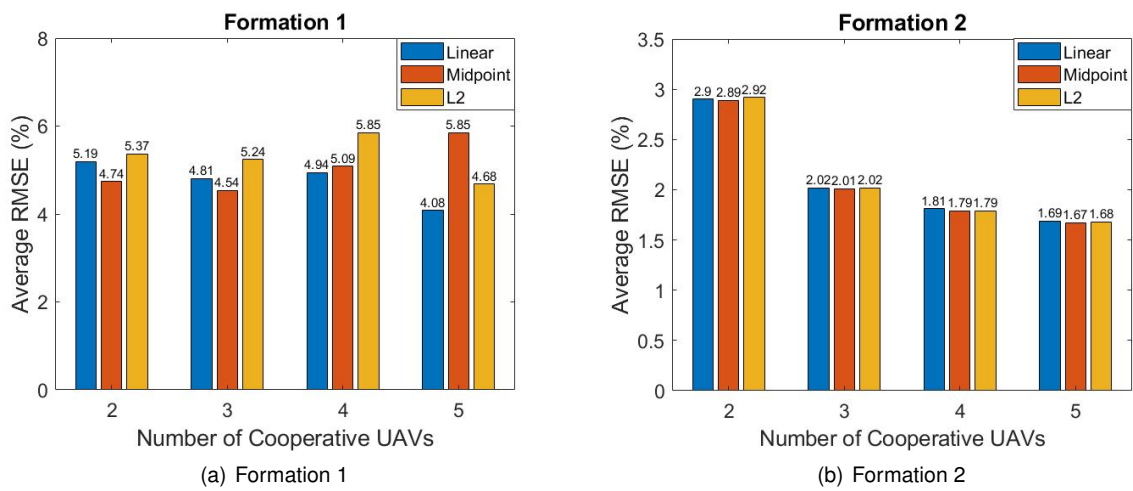


Figure 5.9: Results obtained for the *trajectory 2* simulation, with the cooperative UAVs disposed according to two different formations.

Figure 5.9 (a) refers to the RMSE obtained with *formation 1*, where the cooperative UAVs are positioned close to each other. For this case, the initial two cooperative UAVs are located 10m apart from each other, and subsequent UAVs are added between them so as not to increase the maximum angle between detections. The purpose is to study the benefit of adding additional UAVs independently from the effect of having additional distinct points of view, which is the case when the cooperative UAVs are placed farther apart.

The results in this figure seem to indicate no meaningful performance increase as a result of the extra UAVs. A closer inspection revealed that the benefit of having more sensors is outweighed by the higher probability of having some detections that are not perfectly centered around the target, since when the detections are close to parallel, even small inaccuracies in the detection coordinates can lead to considerable errors in the estimated target position. This effect might be mitigated by training the YOLO detector to provide more accurate detections, that is, bounding boxes that are centered around the target more accurately.

Figure 5.9 (b) refers to the RMSE obtained with *formation 2*. In this formation, the cooperative UAVs surround the target in a circle, with the aim of analyzing the benefit of additional points of view

for the triangulation of the target location. In this case, as expected, with the increase in the number of cooperative UAVs the error decreases. Even though with more cooperative UAVs there is a higher probability that at a certain time step, there will be some detections that are not perfectly centered around the target, the additional points of view are sufficient to improve results. Since in this circumstance the detection directions are not close to being parallel with each other, small errors in the detection coordinates do not affect the final estimated position as significantly.

Nonetheless, it can be inferred that increasing the number of cooperative UAVs seems to have a diminishing return, since the biggest performance gain happens with the increment from two to three cooperative UAVs, and after that any additional increases lead to smaller improvements. Additionally, it can also be concluded that in this case the discrepancies observed between the three triangulation algorithms are not very significant.

Figure 5.10 depicts two examples of the estimated target trajectory, where the detections were made by two and five UAVs positioned according to *formation 1*. These examples illustrate how the estimated trajectory is affected by an increase in the number of cooperative UAVs capturing very similar points of view.

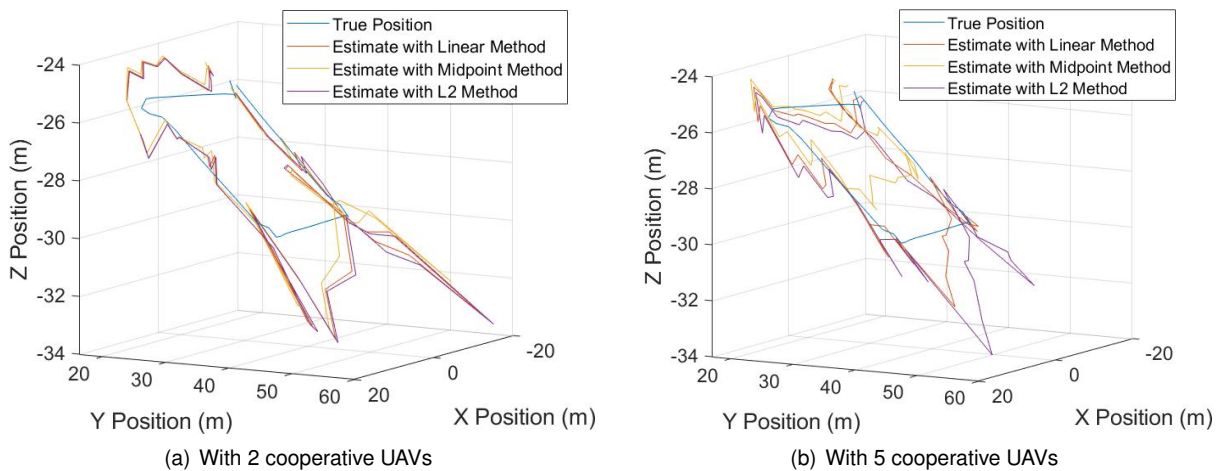


Figure 5.10: Real and estimated target trajectory for the *trajectory 2* and *formation 1* simulation.

As can be seen from the figure, with two cooperative UAVs the estimates from the three triangulation algorithms are very similar. As expected, the error along the y -axis is more significant, since that is the direction that all cameras are facing. Additionally, the error is larger in the segment of the trajectory with $y=40\text{m}$, since that is the point where the target is the farthest away from the sensors.

A similar effect can be seen in the case with five cooperative UAVs performing the target detections. However, in this case there is a larger difference between the results of the three triangulation algorithms, with the linear algorithm performing the best. This seems to indicate that when the target detections are close to parallel, errors in the target detection coordinates or in the cooperative UAVs' pose have a large influence on the final estimated target position, in a way that highlights the differences between each algorithm.

Figure 5.11 illustrates two examples of the estimated target trajectory, where this time the detections were made by two and five UAVs positioned according to *formation 2*. These examples showcase how

the estimated target trajectory is affected by an increase in the number of viewing angles captured by the cooperative UAVs. In this case, the estimated trajectory improves with the larger number of UAVs seeing the target. Furthermore, since the predictions are more accurate the difference between the results of the three triangulation algorithms is not as pronounced.

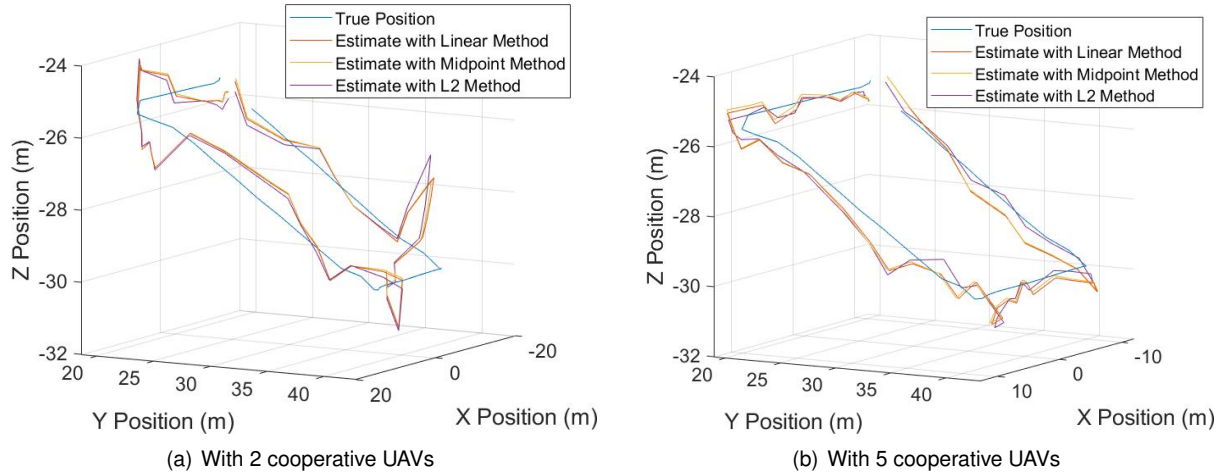


Figure 5.11: Real and estimated target trajectory for the *trajectory 2* and *formation 2* simulation.

5.2.4 Trajectory 3 Simulations

The third and final trajectory has the goal of studying the difference in performance of each triangulation algorithm when the relative distances between each cooperative UAV to the target change. As explained in section 4.2.3, in this trajectory two cooperative UAVs change their distance to the target, one moving closer to it and the other moving in the opposite direction.

If we consider that UAV number 1 is moving towards the target and UAV number two is moving away, we can define the ratio of their respective distances as

$$D_{ratio} = \frac{d_2}{d_1} \quad (5.1)$$

where d_1 and d_2 are the distance from UAVs 1 and 2, respectively, to the target. This ratio quantifies how much closer one UAV is to the target than the other. The change of D_{ratio} over the course of the trajectory is plotted in Figure 5.12 (a). At the start of the simulation, both cooperative UAVs are at the same distance from the target, therefore $D_{ratio} = 1$. As the simulation progresses, this difference in respective distances increases until UAV 1 is 9 times closer to the target than UAV 2.

Figure 5.12 (b) depicts the error of each of the three triangulation algorithms throughout the simulation. At the start of the trajectory the midpoint algorithm obtained the best results, which is in agreement with the results for *trajectory 1*. However, as D_{ratio} increases, the difference in performance between the algorithms becomes less apparent, and at the end of the trajectory all algorithms have a very comparable performance.

The maximum value of D_{ratio} is limited by the ability of the furthest away cooperative UAV to detect the target. It would be expected that the L2 triangulation algorithm would outperform the other two when

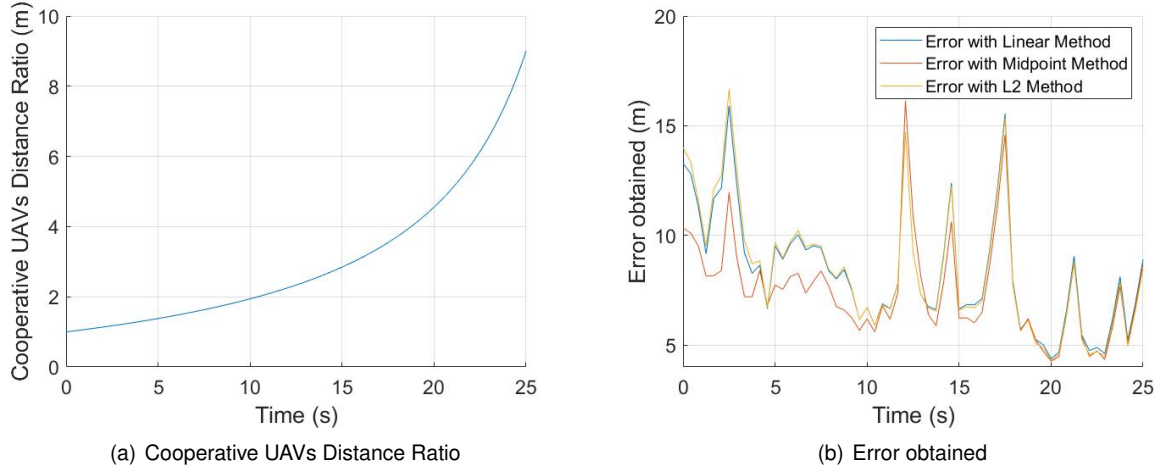


Figure 5.12: Results obtained for the *trajectory 3* simulation.

some sensors are much closer to the target than others, since it minimizes the reprojection error as detailed in section 3.2.2. However, the maximum value of D_{ratio} does not seem to be high enough to see this effect.

5.2.5 Discussion

The YOLOv3 algorithm trained on the Dettfly dataset was able to adequately detect the target when it was above the horizon. The discrepancy in performance when the target was below the horizon might be due to the lower contrast between the target and the background in this case, as well as differences in the drone's appearance between the real photographs in the dataset used for training and the images captured in the AirSim simulations, which can be more significant in situations of lower contrast.

As for the triangulation algorithms, the midpoint algorithm seems to produce the most accurate results in the majority of the situations studied. In addition, it is a simple and fast algorithm to implement, which makes it a good candidate for the target UAV localization problem.

The L2 algorithm proved to not be advantageous in the range of realistic cooperative UAV positions. Although it is meant to have a better performance in comparison with the others when some sensors are significantly closer to the target than others, due to the limitations in the distance that the cooperative UAVs can be away from the target while still being able to detect it that effect seemed to not be significant.

Finally, some recommendations can be made to improve the estimated target's position accuracy. First, the cooperative UAVs should be as far away from each other as realistically possible, so that they can capture varied points of view. Additionally, they should position themselves in a way that surrounds the target, in order to reduce the estimation error in all axis. Furthermore, increasing the number of cooperative UAVs seems to have a benefit, but not if they are positioned in high proximity to each other.

Chapter 6

Conclusions

6.1 Findings

This thesis provides a multi-sensor methodology for UAV obstacle detection, as well as for the localization of an intruder UAV, taking advantage of current deep learning algorithms.

The examined results suggest that there is a benefit in using the presented method for the fusion of monocular and stereo depth, in order to use monocular depth estimation to complement the measurements of a depth camera for UAV obstacle detection. This method is specially useful when it comes to filling in the gaps in the depth map provided by the depth camera due to the obstacles being out of range. This improves the information available to be used by an obstacle avoidance algorithm, for instance. Having information about potential obstacles at greater distances can allow the UAV to fly faster, for example.

In regards to the target UAV detection, the results also look promising in that monocular cameras are suitable sensors to detect and localize an intruder UAV. First, the YOLO detector was shown to be capable of performing the task of detecting a flying UAV above the horizon, and recommendations on how to improve its performance below the horizon were made. This research also evaluated three different triangulation algorithms in different scenarios. These are the linear triangulation, the midpoint triangulation, and the L2 triangulation methods. It was concluded that the midpoint triangulation algorithm is the most appropriate for the task from among those considered. It also presented suggestions of how to minimize the target position error obtained, both in terms of improved YOLO training and cooperative UAV positioning.

6.2 Future Work

In regards to the monocular and stereo depth fusion method, future steps would include more thorough evaluation, which could be performed with data from the AirSim simulation environment, as well as evaluation onboard a UAV in real time. Further work could also include the integration of this method with an obstacle avoidance algorithm.

When it comes to the intruder UAV localization, potential future work includes improvements in the YOLO detector training, both in terms of increasing its detection capabilities below the horizon and also in augmenting its bounding box precision, in order to reduce the error introduced in the triangulation algorithms. Additionally, the creation of a dynamic region of interest where the target is expected to be in the captured images, based on previous localization results, would allow YOLO to run faster and therefore increase the results precision.

Further work for both methods could also include the algorithms optimization for performance on an onboard computer, and proving their real time capabilities.

Bibliography

- [1] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive uav control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE, 2013.
- [2] Z. Zhang, M. Xiong, and H. Xiong. Monocular depth estimation for uav obstacle avoidance. In *2019 4th International Conference on Cloud Computing and Internet of Things (CCIoT)*, pages 43–47. IEEE, 2019.
- [3] D. C. Tsouros, S. Bibi, and P. G. Sarigiannidis. A review on uav-based applications for precision agriculture. *Information*, 10(11):349, 2019.
- [4] I. Colomina and P. Molina. Unmanned aerial systems for photogrammetry and remote sensing: A review. *ISPRS Journal of photogrammetry and remote sensing*, 92:79–97, 2014.
- [5] K. Máthé and L. Buşoniu. Vision and control for uavs: A survey of general methods and of inexpensive platforms for infrastructure inspection. *Sensors*, 15(7):14887–14916, 2015.
- [6] C. Stöcker, R. Bennett, F. Nex, M. Gerke, and J. Zevenbergen. Review of the current state of uav regulations. *Remote sensing*, 9(5):459, 2017.
- [7] Guardian. Gatwick drone disruption cost airport just £1.4m. <https://www.theguardian.com/uk-news/2019/jun/18/gatwick-drone-disruption-cost-airport-just-14m>. Accessed: 2021-10-10.
- [8] T. Age. Prisons struggle to swat drug-smuggling drones. <https://www.theage.com.au/national/victoria/prisons-struggle-to-swat-drug-smuggling-drones-20201115-p56ep1.html>. Accessed: 2021-10-10.
- [9] X. Yang, J. Chen, Y. Dang, H. Luo, Y. Tang, C. Liao, P. Chen, and K.-T. Cheng. Fast depth prediction and obstacle avoidance on a monocular drone using probabilistic convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems*, 2019.
- [10] D. Wang, W. Li, X. Liu, N. Li, and C. Zhang. Uav environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution. *Computers and Electronics in Agriculture*, 175:105523, 2020.

- [11] D. Martins, K. Van Hecke, and G. De Croon. Fusion of stereo and still monocular depth estimates in a self-supervised learning context. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 849–856, 2018. doi: 10.1109/ICRA.2018.8461116.
- [12] L. Teixeira, M. R. Oswald, M. Pollefeys, and M. Chli. Aerial single-view depth completion with image-guided uncertainty estimation. *IEEE Robotics and Automation Letters*, 5(2):1055–1062, 2020.
- [13] J. M. Fácil, A. Concha, L. Montesano, and J. Civera. Single-view and multi-view depth fusion. *IEEE Robotics and Automation Letters*, 2(4):1994–2001, 2017.
- [14] Y. Zhang and T. Funkhouser. Deep depth completion of a single rgb-d image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.
- [15] G. Fasano, D. Accardo, A. E. Tirri, A. Moccia, and E. De Lellis. Radar/electro-optical data fusion for non-cooperative uas sense and avoid. *Aerospace Science and Technology*, 46:436–450, 2015.
- [16] C. Park, S. Lee, H. Kim, and D. Lee. Aerial object detection and tracking based on fusion of vision and lidar sensors using kalman filter for uav. *International journal of advanced smart convergence*, 9(3):232–238, 2020.
- [17] Z.-Y. Huang and Y.-C. Lai. Image-based sense and avoid of small scale uav using deep learning approach. In *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 545–550. IEEE, 2020.
- [18] R. Zahedi, E. Ceh-Varela, R. Selje II, H. Cao, and L. Sun. Neural network based approaches to mobile target localization and tracking using unmanned aerial vehicles. In *AIAA Scitech 2020 Forum*, page 0392, 2020.
- [19] A. Y. Husodo, G. Jati, N. Alfiany, and W. Jatmiko. Intruder drone localization based on 2d image and area expansion principle for supporting military defence system. In *2019 IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*, pages 35–40. IEEE, 2019.
- [20] G. Laurito, B. Fraser, and K. Rosser. Airborne localisation of small uas using visual detection: A field experiment. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1435–1443. IEEE, 2020.
- [21] C. Shinde, R. Lima, and K. Das. Multi-view geometry and deep learning based drone detection and localization. In *2019 Fifth Indian Control Conference (ICC)*, pages 289–294. IEEE, 2019.
- [22] C. Arnold and J. Brown. Performance evaluation for tracking a malicious uav using an autonomous uav swarm. In *2020 11th IEEE Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0707–0712. IEEE, 2020.
- [23] E. M. Khanapuri and R. Sharma. Uncertainty aware geo-localization of multi-targets with multi-uav using neural network and extended kalman filter. In *AIAA Scitech 2019 Forum*, page 1690, 2019.

- [24] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- [25] D. Hoiem, A. A. Efros, and M. Hebert. Automatic photo pop-up. In *ACM SIGGRAPH 2005 Papers*, pages 577–584. 2005.
- [26] L. Ladicky, J. Shi, and M. Pollefeys. Pulling things out of perspective. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 89–96, 2014.
- [27] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. *arXiv preprint arXiv:1406.2283*, 2014.
- [28] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.
- [29] R. Garg, V. K. Bg, G. Carneiro, and I. Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *European conference on computer vision*, pages 740–756. Springer, 2016.
- [30] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1851–1858, 2017.
- [31] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *arXiv preprint arXiv:1907.01341*, 2020.
- [32] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.
- [33] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. Ieee, 2008.
- [34] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [35] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [36] 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*, 39(6):1137–1149, 2016.

- [37] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [38] 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*, pages 779–788, 2016.
- [39] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.
- [40] Y. Zheng, Z. Chen, D. Lv, Z. Li, Z. Lan, and S. Zhao. Air-to-air visual detection of micro-uavs: An experimental evaluation of deep learning. *IEEE Robotics and Automation Letters*, 6(2):1020–1027, 2021.
- [41] D. A. Forsyth and J. Ponc. *Computer Vision, A Modern Approach*. Prentice Hall, 2003.
- [42] P. Sturm. Pinhole camera model. In *Ikeuchi K. (eds) Computer Vision*, 2014. doi: 10.1007/978-0-387-31439-6_472.
- [43] Y. Morvan. Projective geometry. <http://epixea.com/research/multi-view-coding-thesisch2.html>. Accessed: 2021-10-05.
- [44] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge, 2000.
- [45] G. Cai, B. M. Chen, and T. H. Lee. *Unmanned rotorcraft systems*. Springer Science & Business Media, 2011.
- [46] C. Cadena, Y. Latif, and I. D. Reid. Measuring the performance of single image depth estimation methods. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4150–4157. IEEE, 2016.
- [47] R. Padilla, S. L. Netto, and E. A. da Silva. A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242. IEEE, 2020.
- [48] R. Padilla, W. L. Passos, T. L. Dias, S. L. Netto, and E. A. da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3):279, 2021.
- [49] I. Vasiljevic, N. Kolkin, S. Zhang, R. Luo, H. Wang, F. Z. Dai, A. F. Daniele, M. Mostajabi, S. Basart, M. R. Walter, et al. Diode: A dense indoor and outdoor depth dataset. *arXiv preprint arXiv:1908.00463*, 2019.
- [50] Z. Li and N. Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2041–2050, 2018.
- [51] A. S. Mashaly. Performance assessment of sky segmentation approaches for uavs. *International Journal of Image and Graphics*, 19(04):1950023, 2019.

- [52] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. doi: 10.1109/TPAMI.1986.4767851.
- [53] Y. Zheng, Z. Chen, D. Lv, Z. Li, Z. Lan, and S. Zhao. Air-to-air visual detection of micro-uavs: An experimental evaluation of deep learning. *IEEE Robotics and Automation Letters*, 6(2):1020–1027, 2021.
- [54] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
- [55] R. I. Hartley and P. Sturm. Triangulation. *Computer vision and image understanding*, 68(2):146–157, 1997.
- [56] R. I. Hartley, R. Gupta, and T. Chang. Stereo from uncalibrated cameras. In *CVPR*, volume 92, pages 761–764, 1992.
- [57] K. E. Atkinson. *An Introduction to Numerical Analysis*. Wiley, 1989.
- [58] J. Chen, D. Wu, P. Song, F. Deng, Y. He, and S. Pang. Multi-view triangulation: Systematic comparison and an improved method. *Ieee Access*, 8:21017–21027, 2020.
- [59] P. A. Beardsley, A. Zisserman, and D. W. Murray. Navigation using affine structure from motion. In *European Conference on Computer Vision*, pages 85–96. Springer, 1994.
- [60] S. Ramalingam, S. K. Lodha, and P. Sturm. A generic structure-from-motion framework. *Computer Vision and Image Understanding*, 103(3):218–228, 2006.
- [61] P. Sturm and R. Hartley. Triangulation. In *Image Understanding Workshop*, volume 11, pages 957–966, 1994.
- [62] K. Madsen, H. B. Nielsen, and O. Tingleff. *Methods for non-linear least squares problems*. 2004.
- [63] P. Lindstrom. Triangulation made easy. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1554–1561. IEEE, 2010.
- [64] S. Shah, D. Dey, C. Lovett, and A. Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

