TÉCNICO LISBOA

Non-cooperative UAV Detection and Relative Position Estimation

A Deep Learning Approach Using LiDAR and Camera Data

Mariana Ricardo Santos

Thesis to obtain the Master of Science Degree in

Aerospace Engineering

Supervisors: Professor Afzal Suleman
Professor Rita Maria Mendes de Almeida Correia da Cunha

Examination Committee

Chairperson: Professor Paulo Jorge Coelho Ramalho Oliveira
Supervisor: Professor Afzal Suleman
Member of the Committee: Professor Hugh Liu

December 2021

To my mom and dad

Acknowledgments

I would like to thank my supervisors, Professor Afzal Suleman and Professor Rita Cunha for the availability, guidance, advice and critiques on my work that help me improve my knowledge and achieve better results.

On a more particular note, I would like to thank Professor Afzal Suleman for giving me the opportunity of doing my Master's thesis in Canada. I have always admired its culture and the experience of living abroad was always part of my goals. It was a very rewarding experience, not only on a professional but also personal level, for which I am truly grateful.

I would like to thank my mom and dad for everything they did for me throughout the years. Regardless of the situation, they could always find a way to make sure I could grab all the opportunities I encountered and fight for my future. I couldn't be more proud of being their daughter. I hope that one day I can give them the deserved life they never had.

I would also like to mention my godfather Romeu Gomes, who has always been an enormous inspiration to me. The first person in my family to go to university, he has built an amazing professional career and has a beautiful family.

I would like to thank my friends who came with me to Canada for all the amazing adventures and experiences. Thank you Beatriz, Daniele, Diogo, Hugo C., Hugo F., Pedro and especially Mariana, who shared workspace with me and was always there listening and caring, especially when I needed a friend the most. Thank you Mariana for being one of the most caring and kind people I have ever met.

In this thesis, I continued the work developed by Daniel Justino, who I have to thank for the incredible availability and countless advice that always helped me throughout this work. I could not have reached my results without his input.

I also want to thank everyone at CfAR for all the support and for accompanying my work during this journey, offering advice, providing hardware and sharing their vast knowledge with me.

Finally, I would like to thank Sara and Ana for reminding me that, no matter how far away I am, when I come home, our friendship will always remain as if I had never left. Thank you *Xico* for making me feel that someone cares about what I have to say, regardless of how *nerdy* it might be.

Resumo

O trabalho apresentado nesta tese propõe um sistema capaz de detectar e estimar a posição relativa de UAVs (*Unmanned Aerial Vehicles*) não cooperativos. O sistema utiliza medições de um LiDAR (*Light Detection and Ranging*) e imagens de uma câmara para detectar e seguir UAVs próximos através da aplicação de algoritmos de deep learning. Em vez de anotar manualmente imagens para treinar o detetor de objetos escolhido - YOLO (You Only Look Once), foram criadas imagens sintetizadas e automaticamente anotadas com o *open-source software AirSim*. O YOLO foi treinado com 4761 imagens e validado com 530 imagens, para as quais apresentou mAP (*Mean Average Precision*) igual a 99.03%, precisão de 98.00%, *recall* de 98.00% e um IoU (*Intersection over Union*) de 83.11%. O YOLO fornece coordenadas de *bounding boxes* que, em conjunto com medições do LiDAR, são utilizadas para estimar a posição relativa dos UAVs. Filtros de Kalman são utilizados para suavizar as estimativas obtidas. O sistema pode ser usado em situações nas quais sistemas convencionais de localização não são uma boa solução, como *sense and avoid*. Simulações realizadas com o AirSim apresentaram um RMSE (*Root-Mean Squared Error*) máximo de $8.60m$ para a estimativa de distância com uma câmara de resolução $720p$ e $7.80m$ para uma câmara de resolução $1080p$, quando o UAV se encontra a $Z \approx 50m$. Finalmente, simulações com dois UAVs foram realizadas para confirmar que o sistema funciona para qualquer número de UAVs presentes nas imagens, sem qualquer informação ou suposição sobre os mesmos.

Palavras-chave: Câmara, LiDAR, YOLO, AirSim, Filtro de Kalman

Abstract

This work proposes a system designed to run onboard a UAV capable of detecting and estimating the relative position of encountered non-cooperative UAVs. The system utilizes both LiDAR (Light Detection and Ranging) measurements and images from a camera to detect and track nearby UAVs using a deep learning approach. Instead of manually labelling a dataset to train the chosen object detector - YOLO (You Only Look Once), synthesized images were automatically created and annotated using the open-source software AirSim. YOLO was trained with 4761 training and 530 validation images to which presented a mAP (Mean Average Precision) of 99.03%, precision of 98%, recall of 98% and an IoU (Intersection over Union) of 83.11%. YOLO outputs bounding box coordinates that are combined with measurements given by the LiDAR to estimate the relative position of the encountered UAVs. Kalman Filtering is used to smooth the obtained estimations. The system can be used in situations where conventional localization systems are not a good solution, such as sense and avoid. Simulations performed with AirSim presented a maximum RMSE (Root-Mean Squared Error) of $8.60m$ for the distance estimation with a camera with a $720p$ resolution and $7.80m$ for a camera with $1080p$, when the UAV is at $Z \approx 50m$. When tested in the simulation images, YOLO presented a precision of 100% and the lowest recall value of 92%. Finally, simulations with two UAVs were performed to confirm that the system works for any number of encountered UAVs without any a priori information or assumption.

Keywords: Camera, LiDAR, YOLO, AirSim, Kalman Filter

Contents

List of Tables

List of Figures

Nomenclature

Greek symbols

α	Angle between vectors a_1 and a_2 [$^\circ$].
γ	Aspect Ratio [adimensional].
ψ	Azimuth [$^\circ$].
θ	Elevation [$^\circ$].

Roman symbols

d	Distance [m].
h	Height [pixels].
r	Apparent Physical Size of an UAV [meters].
w	Width [pixels].

Subscripts

i, j, k, n	Computational indexes.
u, v	Component of the bounding box centre [pixels].
X, Y, Z	Cartesian components [m].
x, y, z	Cartesian components [pixels].

Superscripts

$\dot{}$	First time derivative.
i, j	Computational indexes.
T	Transposed.

List of Acronyms

AI	Artificial Inteligence
AP	Average Precision
API	Application Programming Interface
CfAR	Centre for Aerospace Research
CNN	Convolutional Neural Network
cpdf	conditional probability density function
CPU	Central Processing Unit
Deep SORT	Simple Online and Realtime Tracking With a Deep Association Metric
DNN	Deep Neural Network
EO	Electro-Optical
FOV	Field of View
GPU	Graphics Processing Unit
IoU	Intersection Over Union
IPPE	Infinitesimal Plane-Based Pose Estimation
LiDAR	Light detection and ranging
LSS	Low, Slow and Small
mAP	Mean Average Precision
MAV	Micro Air Vehicle
MaxAE	Maximum Absolute Error
MOT	Multi-Object Tracking
NATO	North Atlantic Treaty Organization
PnP	Perspective-n-Point
RADAR	Radio Detection and Ranging
RAM	Random Access Memory
RMSE	Root Mean Square Error
ROI	Region of Interest
SAA	Sense and Avoid
SORT	Simple Online and Realtime Tracking
UAS	Unmanned Aircraft Systems
UAV	Unmanned Aerial Vehicle

YOLO You Only Look Once

Chapter 1

Introduction

An overview of the potential of autonomous Unmanned Aerial Vehicles (UAVs) is presented in this chapter as a contextualization for the current work. Afterwards, an analysis of the project for which the presented system was developed is performed as well as an overview of the state of the art current methods that serve as a base for the algorithm. This chapter ends with the thesis objectives and contributions, followed by the outline of the thesis.

1.1 Context and Motivation

As the technological capabilities of UAVs increase, these aircraft are becoming more popular due to their wide and various applications. UAVs are affordable, functional, versatile and continue to become easier to acquire and operate. These vehicles present a robust solution for many industries and are capable of serving several commercial and individual needs such as search and rescue [1], precision agriculture [2], aerial photography [3] and infrastructure monitoring [4].

However, UAVs still have limitations in practical scenarios due to their lower payload capacity, shorter endurance and operational range. The potential of UAVs could be further explored if autonomous operations were possible, without the need for a human pilot. However, proving these UAVs are safe and capable, so they can be allowed to perform desired applications, is challenging. Most UAV operations are constrained to designated airspace areas or within temporary restricted airspace areas, commonly known as segregated airspace. On special operations, UAVs can be permitted to fly in a limited environment outside segregated airspace [5]. There has been an effort to integrate these vehicles in the non-segregated airspace [6], but for this to be possible UAVs need to be equipped with Artificial Inteligence (AI) algorithms capable of handling difficult problems and reacting in complex scenarios. Many of these algorithms perform tasks that are executed by humans in manned aircraft, such as environmental sensing and perception, trajectory planning and collision avoidance.

North Atlantic Treaty Organization (NATO) industrial advisory group (NIAG) categorizes Low, Slow and Small (LSS) Unmanned Aircraft Systems (UAS) according to their weight, operating altitude, mission range, and payload capacity. Table 1.1 presents the details of these classifications. The work presented

Table 1.1: NATO classification of LSS-UAS. [7].

Categories	Weight[kg]	Payload [kg]	Coverage [km]	Altitude [m]
Micro	< 2	< 1	< 10	< 1500
Mini Light	< 10	< 5	< 25	< 3000
Mini Heavy	< 25	< 12	< 50	< 4000
Small	< 150	< 50	< 150	< 6000

in this thesis is focused on Micro, Mini Light and Mini Heavy UAS and it is also focused in only a part of the capabilities associated with an autonomous system: the sense and relative localization estimation tasks that can be integrated with trajectory-planning algorithms for collision avoidance protocols.

For the sensing task, the ability to use inexpensive and lightweight sensors such as cameras for collision-avoidance purposes has become increasingly important. Visual detection of outdoor flying small drones is challenging because they are relatively small and can fly in environments with large amounts of background clutter and difficult lighting conditions. However, this problem has been tackled successfully in the automotive world, for example, there are now commercial products [8, 9] designed to sense and avoid both pedestrians and other cars. The problem of extending the algorithms used for pedestrian and automotive detection to the world of small UAVs is difficult, as flying objects detection poses some unique challenges [10]:

- The motions are more complex in the three-dimensional space, with objects moving in any direction and appearing in any part of the frame;
- Flying objects can have very different shapes and can fly against very complex backgrounds making them hard to detect, even for the human eye;
- Taking into account the speeds at which flying objects travel, potentially dangerous UAVs must be detected when they are still far away, at distances much greater than the intruder size, which means they may appear very small in the images;
- For small UAVs, having a powerful computer on-board is not feasible due to the limited payload of these vehicles.

Algorithms developed for the sensing task of flying objects are usually integrated with collision-avoidance algorithms, providing Sense and Avoid (SAA) capabilities to unmanned flying vehicles.

1.1.1 Sense and Avoid

Sense and avoid systems have four components (see Figure 1.1): sensing, conflict detection, collision avoidance and flight controller [11]. The first three components should replace the pilot's ability to "see and avoid". The detection of a potential collision must be at a minimum range that depends on the performance of both aircraft (such as the cruise speed, turn rate, and climb or descent rates). Furthermore, this detection should be possible in all weather conditions the UAS may encounter and even in

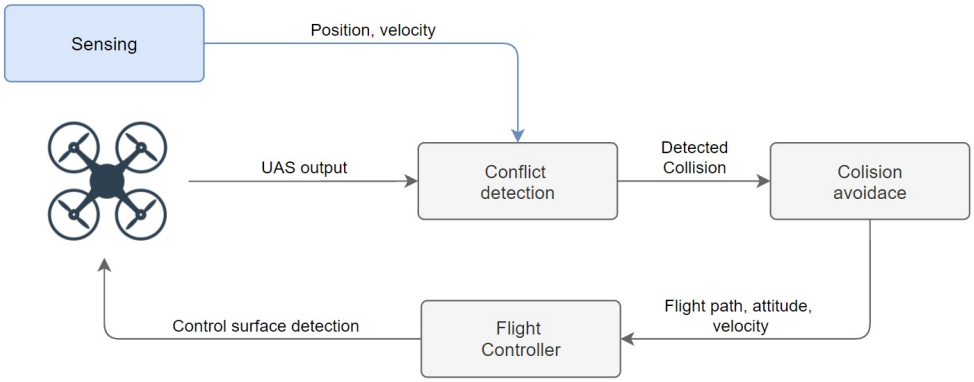


Figure 1.1: Sensing task as part of a collision-avoidance system. [11].

the case of loss of direct command, control, or communications with the command ground station [12]. Thus, it is of paramount importance to consider all these factors when designing the sensing subsystem for the UAS.

Non-cooperative Sensors

Sensors for sense and avoid applications can be divided into cooperative and non-cooperative depending on whether it is necessary for the encountered UAS to transmit information about its position or not, respectively. UAS which will not actively cooperate to resolve a conflict are classified as non-cooperative. For this project, the sensors used are non-cooperative to guarantee the safety of the flying vehicle against any type of intruder obstacle. These sensors can be classified as passive or active sensors, depending on whether or not they transmit the energy needed for detecting the object [11].

Two examples of passive non-cooperative sensors include:

- **EO/IR Cameras**: Electro-Optical (EO) and infrared (IR) cameras have the same basic principles, differing on the wavelength of the captured radiation. These sensors are lightweight and can provide high-resolution perceptions of the scene. EO cameras use visible light and can only operate during the daytime, but IR imagining sensors can be helpful under extreme lighting conditions and night-time [13].

Two examples of active non-cooperative sensors include:

- **RADAR:** Radio Detection and Ranging (RADAR) can work during day or night and in all weather conditions. A trade-off in low-power and long-range is expected from an effective sense and avoid radar system. It identifies a target based on the returned pulses reflected by the detected UAV and therefore, presents problems detecting small targets, which have a low radar cross-section (RCS). Solutions to capture small targets include multiple-input multiple-output (MIMO) radars [14–16].
- **Light detection and ranging (LiDAR):** LiDAR works similarly to a RADAR. Instead of radio waves, short and precise laser light impulses are emitted with high frequency, preceding sensor measurement of reflected light. It provides 3D sparse point cloud representations of the environment and can achieve high update rates $(5 - 20Hz)$ over ranges of $100m$.

Data Fusion

It might be useful to use multiple sensors types to detect and track encountered UAVs since different sensors have different advantages and disadvantages [17]. Robustness and accuracy can be improved as well as the sensing range. False positive and false negative detections can be also be reduced. However, combining data from multiple sensors adds challenges to the algorithms because the system must decide which data correspond to the same objects.

1.2 Project Overview

The work presented in this thesis is part of the research performed at the Centre for Aerospace Research (CfAR) of the University of Victoria. The task consisted in continuing the work developed in the previous year by Daniel Justino [18] where a sensing system detects and tracks potential intruder UAVs in images captured using a LiDAR and a Camera. A LiDAR can provide 3D point clouds with representations of the environment at ranges over $100m$ at high update rates $(5 - 20Hz)$ in low visibility conditions such as at night. However, the 3D point clouds provided are sparse and detections worsen in the presence of rain, fog, snow or dust. Cameras can provide rich texture-based and colour-based information, and object detection methods using deep learning have exceeded many traditional methods in both accuracy and speed. By combining both measurements from a LiDAR and Camera it is possible to improve the deficiencies from each individual system.

The system should fly onboard a UAV and be used for integration with avoidance trajectory planning algorithms for SAA applications. The work developed in the previous year was capable of detecting and tracking UAVs in an image but no estimation regarding relative position or velocity was accomplished. Therefore, the method presented in this thesis aims to continue the previous work by estimating the relative localization of the encountered UAVs in (X, Y, Z) coordinates relative to the world coordinate system.

1.3 Related Work

Numerous approaches to vision-based UAV detection and tracking have been presented in the literature. Some of the most effective approaches with low false positives utilise multi-stage-detection pipeline, machine learning and deep learning [19].

Multi-stage detection pipelines have become less popular with the arising of machine learning and deep learning methods. The key stages include image pre-processing, temporal filtering and detection logic. In the pre-processing stage, background clutter is suppressed and small pixel sized aircraft are highlighted. Popular methods for this stage include morphology [20] and machine learning [21]. Temporal filtering is required to emphasize and extract features that are usually associated with flying vehicles. In [20] Hidden Markov Model (HMM) filtering is used and [22] proposed the use of an extended Kalman filter. The next stage, detection logic, uses the information from the image pre-processing and temporal

filtering to assess whether there is a flying vehicle on the scene or not. A multi-stage-detection pipeline used to compute the 3D position of a flying target in images captured with a depth camera is proposed in [23].

More recently, deep learning approaches have been explored for the vision-based detection and relative localization estimation of objects. In [24] a single camera and the deep learning algorithm You Only Look Once (YOLO) [25], are used to estimate the distance to a Micro Air Vehicle (MAV) given its known physical size. A similar approach is presented in [26] where the algorithm can be used for a larger number of drones by training YOLO with a dataset that is created while systematically flying a quadcopter in front of a static camera and applying background subtraction to the images. In [27], a monocular camera and YOLO are used in a collision-avoidance system where a nested Kalman Filter is used to improve the estimations of distance and velocity. All these methods use single cameras and require a priori information about the vehicles, so they cannot be applied to non-cooperative UAVs.

By introducing a LiDAR in the system, it is possible to extend these approaches to non-cooperative UAVs, as it will be shown in the next chapters. An example of a LiDAR Obstacle Warning and Avoidance System (LOWAS) employed in a system for UAS sense and avoid applications is presented in [28]. In [29], a single camera and a LiDAR are used for the detection of beacons, with results improving by using information from both sensors and overcoming each sensor's individual deficiencies. A neural network is trained with data from the LiDAR and the camera so that it could estimate the position of encountered beacons. These estimations were compared with LiDAR 3D point clouds and fuzzy logic was used to compute the score of the final outputted distance.

1.4 Objectives

In this thesis, the study is focused on the relative localization estimation of non-cooperative UAVs which can be encountered in airspace using data from a LiDAR and an EO camera. The work presented in this thesis is capable of performing detection and relative localization tracking of UAVs without relying on any cooperative sensor, marker, information or previous knowledge about the possible encountered UAVs. This means that it can be used in scenarios where it may not be expected that the UAV will co-operate in any way. The developed system should be lightweight, removing the need for extra hardware to be carried, which may be important in UAVs that have limited carrying capacity.

1.5 Thesis Outline

This thesis is organized as follows: Chapter 2 introduces a theoretical background that serves as a base for the system developed in this thesis. Chapter 3 discusses the method proposed as a solution for the problem of relative localization estimation of UAVs with an overview of the system followed by details of the algorithm used for the detection and position estimation. Chapter 4 presents the simulations performed with open-source software and the correspondent results obtained. Finally, conclusions, contributions and future work are discussed in Chapter 5.

Chapter 2

Theoretical Background

This chapter begins with an overview of the theoretical work that has been done in the fields of visual object detection and tracking, where deep learning algorithms for the detection and tracking of objects are discussed. A summary of the principles and models used for camera projection and sensor calibration involving cameras is presented afterwards. LiDAR physics fundamentals and the mathematics behind Kalman Filtering are also presented. This chapter ends with the presentation of some metrics that will be used for the evaluation of the algorithms proposed in this thesis work.

2.1 Visual Object Detection

Humans are capable of looking at an image and instantly know which objects are presented and their positions. The human visual system is fast and accurate, allowing people to perform complex visual tasks. One of the fundamental computer vision problems is to completely understand an image by trying to precisely estimate the concepts and locations of the objects contained in each image. This task is referred to as object detection and its objective is to surround an object within a bounding box and classify which type of object it is.

Traditional object detection models have a pipeline that can be mainly divided into the following stages: informative region selection, feature extraction and classification. [30]. For the informative region selection stage, to find all the possible positions of objects, a usual but exhaustive strategy is to scan the image with a sliding window. The next stage, feature extraction, focuses on the problem of extracting visual features that allow to recognize different objects but the diversity of backgrounds, shapes, illumination conditions makes it difficult to design a robust feature extractor. Finally, a classifier is necessary to distinguish objects from different categories.

2.1.1 Object Detection With Deep Learning

Deep Neural Networks (DNNs) can be referred to as neural networks with deep structures and have deeper architectures with the capacity to detect more complex features and training algorithms that allow the networks to learn object representations without the need to design features manually. The history of

neural networks can date back to the 1940s [31] but stopped being largely used in the early 2000s due to lack of large-scale training data, limited computational power and lower performance when compared with other machine learning methods. Deep learning became popular in 2006, after a breakthrough in speech recognition.

Generic object detection can be divided into two main categories: region proposed based and regression/classification based. R-CNN [32] is an example of the first approach, which follows the traditional pipeline, generating firstly potential bounding boxes in an image and then running a classifier on them. Post-processing is used to refine the bounding box and eliminate duplicate detections. The second treats object detection in a unified framework where categories and localizations are achieved directly. An example is YOLO [25] in which the object detection is treated as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Overall, region proposal-based methods perform better, but classification based methods can be processed in real-time at the cost of a drop in accuracy. [30].

2.1.2 YOLO

The current state-of-the-art deep learning detector YOLO is a Convolutional Neural Network (CNN) that can be trained to detect different classes of objects in an image.

The input image is divided into an SxS grid. Each cell of this grid is responsible for the prediction of the object centred in that cell. Each cell proposes B bounding boxes and their correspondent confidence scores. These scores are a means of measuring the confidence YOLO has that the box contains an object and how accurate is the object's classification. The confidence scores are defined as $Pr(Object)\times IOU_{pred}^{truth}$, which indicates how likely there are objects ($Pr(Object) \geq 0$) and the confidence in their prediction (IOU_{pred}^{truth}), which is defined in 2.6.

The output of YOLO is a set of predicted bounding boxes that consist of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the centre of the box and (w, h) the width and height of the box. Because many bounding boxes have a low prediction score, YOLO keeps only the ones with the highest scores by imposing a confidence threshold.

For each cell, C conditional class probabilities are predicted, $Pr(Class_i \mid Object)$, translating the probability of the classification given, knowing that the cell contains an object. Only one set of class probabilities per grid cell is predicted, regardless of the number of bounding boxes, B. Yields the class-specific confidence scores for each bounding box,

$$Pr(Class_i \mid Object) \times Pr(Object) \times IOU_{pred}^{truth} = Pr(Class_i) \times IOU_{pred}^{truth}. \tag{2.1}$$

The object detector predictions are encoded as an $S \times S \times (B \times 5 + C)$ grid, see Figure 2.1.

Since the release of the first version of YOLO, many improvements have been made to the object detector. YOLOv1 suffers from a variety of deficiencies relative to other state-of-the-art object detectors regarding, for example, localization errors or low recall when compared to region proposal-based methods.

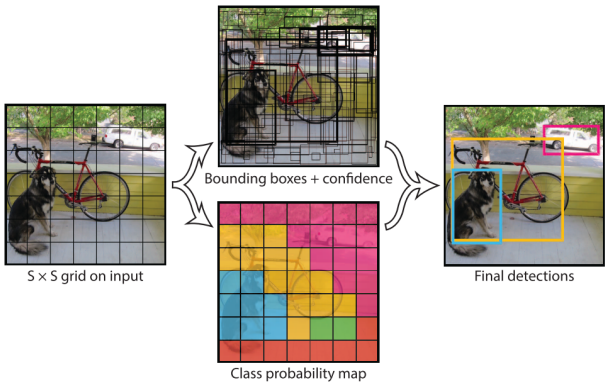


Figure 2.1: Detection method of YOLO. The input image is divided into an $S \times S$ grid and for each cell there are B predicted bounding boxes with correspondent scores, and C class probabilities. Taken directly from [25].

2.1.3 YOLO Frameworks

The release of YOLOv2 [33] intended to present a more accurate detector that was still fast. Batch Normalization [34] is a method used to accelerate the training of a DNN and was added to the convolutional layers of YOLO, improving the convergence during training and the Mean Average Precision (mAP). Bounding boxes are predicted using anchor boxes increasing the model's recall at the cost of a small decrease in the mAP. Dimension clusters are used on the training set to automatically find good values for the dimensions of the anchor boxes. The size of the input images changed from 448×448 to 416×416 and instead of fixing the input image size, every 10 batches the network randomly chooses a new size, forcing the network to learn to predict well for a variety of input sizes.

YOLOv3 [35] achieved a higher mAP and was better capable of detecting smaller objects. It predicts bounding boxes at three different scales and a similar approach to the concept of pyramid networks [36] is used to extract features. The method allows getting more meaningful semantic information from the later convolutional layers and high-resolution features from the earlier layers.

In YOLOv4 [37], a "bag of freebies" i.e., methods that increase the training time without affecting the inference time are implemented. Methods that only affect the inference time are also added to the network, called "Bag of Specials". It can be easily be trained on a single GPU, unlike many other detection algorithms. YOLO is the state-of-the-art for real-time single-stage detectors.

A comparison between the discussed frameworks is presented in Table 2.1.

Table 2.1: Comparison of YOLO detection frameworks. Models marked with * are compared on PASCAL VOC 2012 (PASCAL Visual Object Classes Challenge) [38], while others on MS COCO (Microsoft Common Objects in Context) [39]. [40]

Frameworks	Year	Input size	$AP_{0.5}$[%]	FPS[Hz]
YOLO*	2015	448	57.9	45
YOLOv2	2016	352	44.0	81
YOLOv3	2018	320	51.5	45
YOLOv4	2020	512	64.9	31

2.2 Multi-Object Tracking

Multi-Object Tracking (MOT), also called Multi-Target Tracking (MTT), is one of the many computer vision tasks. Its objective is to analyse videos to identify and track objects without any prior knowledge about their appearance and number. The MOT task can be applied to both 2D and 3D data, but in this thesis, we focus our study on 2D data extracted by a single camera. [41]

Usually, MOT algorithms share part of the following steps: detection (an object detector analyses the input frame), motion prediction (predicts the next position of each tracked target), affinity stage (compute a similarity/distance score between pairs of detections) and association stage (associate detections from different frames belonging to the same target). Many MOT algorithms skip the detection stage focusing more on the association algorithm.

One usual approach to identify helpful observations in MOT algorithms is tracking-by-detection. In this method, a set of bounding boxes identifying the targets in the images are used in the tracking process, usually by associating them together and constructing a track. This approach is standard and very effective. Another example is tracking-by-matching, in which there is a model for how the object moves and appears. The object's appearance model is constructed with information from previous frames and objects in the current frame are selected based on the distance to the samples. This method performs well if the target does not change abruptly its appearance or the background is not very complex.

2.2.1 Deep SORT

Simple Online and Realtime Tracking (SORT) [42] proposes a tracking-by-detection framework for the MOT problem which can be used for real-time applications, combining both speed and accuracy. The detections of each frame are represented as bounding boxes and the tracker only considers detections from the current and previous frames.

Simple Online and Realtime Tracking With a Deep Association Metric (Deep SORT) [43] integrates appearance information to improve the performance of SORT, allowing to track objects during longer periods of obstructions while keeping it easy to implement and efficient. It assumes the camera is not calibrated and there is no ego information available. They approximate the inter-frame displacements of each object with a linear constant velocity model which is independent of other objects and camera motion. For the tracking step, it models the object state x as

$$x = \begin{bmatrix} u & v & \gamma & h & \dot{x} & \dot{y} & \dot{\gamma} & \dot{h} \end{bmatrix}^T, \tag{2.2}$$

where u and v represent the bounding box centre position, γ is the aspect ratio, h is the height and the remaining values represent their respective velocities in image coordinates.

For each new UAV assignment, the algorithm creates a new track that stays in a tentative state until A_{max} consecutive associations are successful. Afterwards, the track transitions into an active state (a1). Tracks that are not successfully associated with a measurement within their first A_{max} frames are deleted by transitioning into an inactive state (a2). The track remains in an active state for as long as

observations keep being assigned to them (a3). When no associations are made, the track transitions into a lost state (a4) where the Kalman Filter from Deep SORT continues to predict the target position based on previous observations (a5). If it associates one observation with a track, this track transitions back to the active state (a6), but if no associations are made for N iterations, the track is removed (a7).

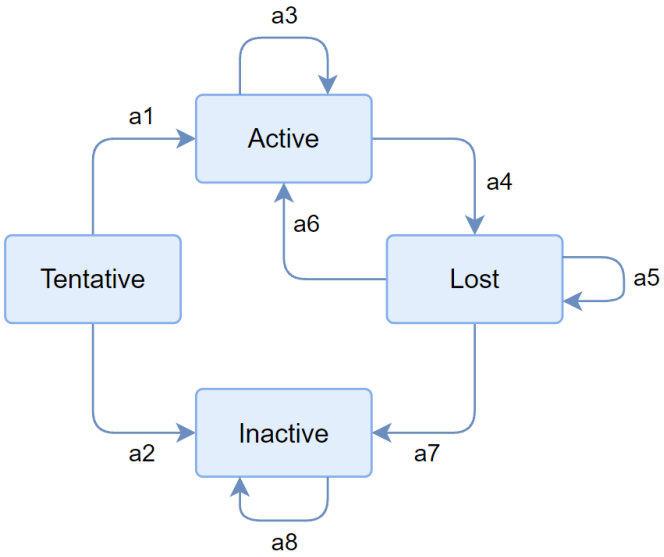


Figure 2.2: State machine of the Deep SORT tracker.

For the assignment task, Deep SORT integrates motion and appearance information. To incorporate motion information, the squared Mahalanobis distance between the newly arrived measurements and the predicted Kalman Filter states is used, accordingly with

$$s^{(1)}(i,j) = (s_j - s_i)^T S_i^{-1}(s_j - y_i), \tag{2.3}$$

where (y_i, S_i) is the projection of the i^{th} track distribution into measurement space and s_j is the j^{th} bounding box detection. A second metric is integrated into the assignment problem, for each bounding box detection s_j an appearance descriptor, g_j is computed. Then, the second metric measures the smallest cosine distance between the i^{th} track and the j^{th} detection is appearance space,

$$s^{(2)}(i,j) = min\{1 - g_j^T g_k^{(i)} | g_k^{(i)} \in \mathcal{G}_i\}. \tag{2.4}$$

In practice, a pre-trained CNN is applied to compute bounding box appearance descriptors.

2.3 Geometric Camera Parameters

2.3.1 Pinhole Respective Projection Model

The pinhole perspective (also called central perspective) projection model was first proposed at the beginning of the fifteenth century and is a mathematically convenient and simple approximation of the imaging process. Let P denote a scene point with coordinates (X, Y, Z), p denote its image with coordi-

nates (x, y, z) and O the ideal pinhole of the camera, as shown in Fig. 2.3.

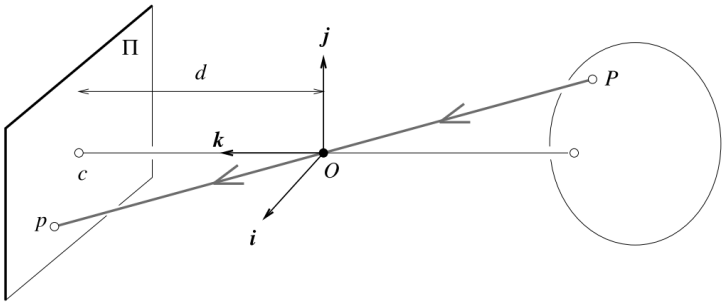


Figure 2.3: Collinearity of the point P, its image p, and the pinhole O from which the perspective projection equations are derived. [44]

Since p is a point in the image plane, $z = d$ and because p, P and O are collinear yields,

$$\begin{cases} x = \lambda X \\ y = \lambda Y \\ d = \lambda Z \end{cases} \Leftrightarrow \begin{cases} x = d\frac{X}{Z} \\ y = d\frac{Y}{Z} \end{cases} . \tag{2.5}$$

It is possible to notice that if we have information regarding a point in the camera frame, (x, y), it is not possible to recover the correspondent coordinates in the scene, (X, Y, Z) without knowing any more information.

2.3.2 Homogeneous Coordinates

Consider a coordinate vector P to be $(X, Y, Z)^T$ in $\mathbb{R}^3$ in some fixed world coordinate system. Its homogeneous coordinate vector $\mathbf{P} = (X, Y, Z, 1)^T$ in $\mathbb{R}^4$ and the correspondent vector $\mathbf{p} = (x, y, 1))^T$ of its image p in the camera's reference frame are related by the perspective projection equation [44],

$$\mathbf{p} = \frac{1}{Z}\mathcal{M}\mathbf{P}, \tag{2.6}$$

where $\mathcal{M}$ is the matrix that provides an algebraic representation of the perspective projection process.

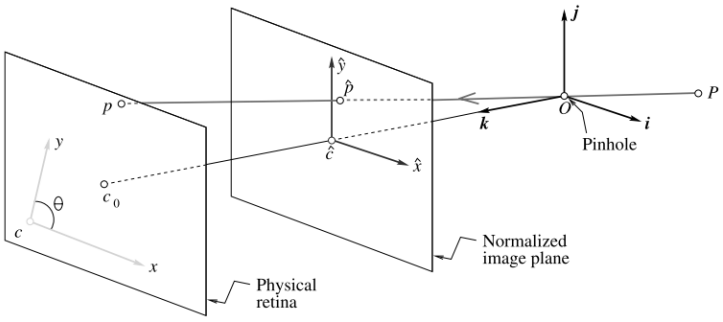


Figure 2.4: Normalized and physical coordinate systems. [44]

2.3.3 Intrinsic Parameters

Consider a normalized image plane parallel to the image plane but located at a distance $d = 1$ from the pinhole, O, with its own coordinate system. Let $\hat{\mathbf{p}} = (\hat{x}, \hat{y}, 1)^T$ be the homogeneous coordinates of the projection of point P into the normalized image plane. Equation (2.5) can be written in this normalized coordinate system as

$$\begin{cases} \hat{x} = \frac{X}{Z} \\ \hat{y} = \frac{Y}{Z} \end{cases} \Leftrightarrow \hat{\mathbf{p}} = \frac{1}{Z} \begin{bmatrix} Id & 0 \end{bmatrix} \mathbf{P} \tag{2.7}$$

The physical retina of the camera usually is located at a distance $d \neq 1$ from the pinhole and the coordinates of the image points are usually expressed in pixels instead of meters. Additionally, pixels may be rectangular instead of square and the actual origin of the camera coordinate system is at a corner c of the retina and not at its centre. Finally, the camera coordinate system might be skewed, so the angle θ between the two image axes is not equal to 90 degrees (see Fig. 2.4).

For these reasons, the relation between the vector of homogeneous coordinates in the image frame and the homogenous coordinates in the normalized image plane is obtained through the camera calibration matrix $\mathcal{K}$:

$$\mathbf{p} = \mathcal{K}\hat{\mathbf{p}} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \hat{\mathbf{p}}, \tag{2.8}$$

and, consequentially, equation 2.6 can be rewritten as

$$\mathbf{p} = \frac{1}{Z}\mathcal{K} \begin{bmatrix} Id & 0 \end{bmatrix} \mathbf{P} \tag{2.9}$$

The pinhole model provides only a simple approximation for the projection of points from the real world to the camera coordinate system and is not valid when high accuracy is required. In these cases, a more realistic camera model should be used. One method is to correct the radial and tangential lens distortion that are responsible for the radial and tangential displacements of the image points in the image plane.

Remembering that $(\hat{x}, \hat{y})^T$ are the coordinates of the projection of point P in the normalized image plane, the radial distortion can be approximated by

$$\begin{cases} \delta x^{(r)} = \hat{x}(k_1 r^2 + k_2 r^4 + ...) \\ \delta y(r) = \hat{y}(k_1 r^2 + k_2 r^4 + ...) \end{cases}, \tag{2.10}$$

where k_1, k_2... are the radial distortion coefficients and $r = \sqrt{\hat{x}^2 + \hat{y}^2}$. The tangential distortion can be approximated by

$$\begin{cases} \delta x^{(t)} = 2p_1\hat{x}\hat{y} + p_2(r^2 + 2\hat{x}^2) \\ \delta y^{(t)} = 2p_2\hat{x}\hat{y} + p_1(r^2 + 2\hat{y}^2) \end{cases}, \tag{2.11}$$

where p_1 and p_2 are the tangential distortion coefficients. It is concluded that, in this model, the set

of intrinsic parameters (f_x, f_y, s, c_x, c_y) is augmented with the distortion coefficients $k_1, ..., k_n$, p_1 and p_2 [45]. Since these parameters have a certain physical meaning, they are known as physical camera parameters. Let $\tilde{\mathcal{K}}$ be the augmented matrix of $\mathcal{K}$, equation (2.9) is written as

$$\mathbf{p} = \frac{1}{Z}\tilde{\mathcal{K}} \begin{bmatrix} Id & 0 \end{bmatrix} \mathbf{P}. \tag{2.12}$$

2.3.4 Extrinsic Parameters

Equation (2.12) is written in a coordinate frame attached to the camera, C. Consider the case where this frame does not correspond to the world coordinate system, W. Extrinsic camera parameters are necessary to transform the world coordinates to the camera centred coordinate frame.

Let ${}^{C}\mathbf{P}$ denote the vector of homogeneous coordinates of point P expressed in C and ${}^{W}\mathbf{P}$ the same vector expressed in W. It is possible to change between C and W using the following rigid transformation

$${}^{C}\mathbf{P} = \begin{bmatrix} \mathcal{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} {}^{W}\mathbf{P}, \tag{2.13}$$

where $\mathcal{R}$ is a rotation matrix and $\mathbf{t}$ is a translation vector. In this case, it is possible to write equation 2.12 as

$$\mathbf{p} = \frac{1}{Z}\tilde{\mathcal{K}} \begin{bmatrix} \mathcal{R} & \mathbf{t} \end{bmatrix} {}^{W}\mathbf{P}, \tag{2.14}$$

which is the most general form of the perspective projection equation.

2.3.5 The PnP Problem

The Perspective-n-Point (PnP) problem is the problem of finding matrices $\mathcal{R}$ and $\mathbf{t}$ given the camera calibration matrix $\mathcal{K}$, the distortion coefficients $k_1, ..., k_n$, p_1 and p_2, and a set of 3D points in the world and their corresponding 2D coordinates in the camera frame. A solution for this problem is the Infinitesimal Plane-Based Pose Estimation (IPPE) [46] which is based on finding a point where the transformation is best estimated and constraining the pose using only the transformation at this point. This method is fast and leads to accurate pose estimates.

2.4 LiDAR Principles

LiDAR sensors have a working principle similar to RADAR sensors and use information acquired by the light emitted by a laser beam or pulse. The laser emits coherent beams or pulses since all photons are emitted at the same time and have the same phase. They are also monochromatic because all photons have the same frequency or wavelength. Finally, they are said to be collimated because the photons emitted move on parallel rays. The emitted power must be kept within a safety envelope and an adequate frequency must be selected to guarantee safety in human environments [47]. For example, Class 1 laser systems are safe under all conditions of normal use.

The standard LiDAR equation [48] is derived from the RADAR equation and relates the power of the received and transmitted signals. In the case of objects that are distributed in space, the received power is given by

$$P_r(t) = \frac{D^2}{4\pi\lambda^2} \int_0^H \frac{\eta_{sys}\eta_{atm}}{R^4} P_t\left(t - \frac{2R}{v_g}\right)\sigma(R)dR, \tag{2.15}$$

where t is the time, D is the aperture diameter of the receiver optics, P_r the received power, P_t the emitted power, λ the wavelength, H the flying height, R the distance from the system to the target, η_{atm} and η_{sys} respectively the atmospheric and system transmission factors, v_g the group velocity of the laser pulse, and $\sigma(R)dR$ the apparent effective differential cross-section. The cross-section is called "apparent" because an object can be occluded by another object that is reflecting the signal.

Usually, a very short but intense pulse of laser radiation is used to compute distances by measuring its Time of Flight (TOF), accordingly with,

$$d = \frac{c \cdot (t_r - t_t)}{2}, \tag{2.16}$$

where d is the distance between the LiDAR and the object, c is the speed of electromagnetic radiation and t_r and t_t are the received and transmitted time measurements, respectively.

The relationship between the distance measurements and the LiDAR's coordinate frame is given by

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = d \begin{bmatrix} cos(\psi) \cdot cos(\theta) \\ cos(\psi) \cdot sin(\theta) \\ sin(\psi) \end{bmatrix} \tag{2.17}$$

where ψ and θ are the elevation and azimuth angles of the emitted laser beam, respectively [49].

2.5 Kalman Filtering

The Kalman Filter [50] is a mathematical procedure for situations in which noisy sensor outputs are used to estimate the state of a system with uncertain dynamics. The filter applies to linear, discrete-time, time-varying systems characterized by a sequence of noisy observations that allow to evaluate the state estimate through the minimization of the mean square error.

2.5.1 Discrete Kalman Filter

Consider a linear, discrete-time system having dynamics [51]

$$x(k+1) = A(k)x(k) + B(k)u(k) + G(k)w(k), \qquad t \geq 0 \tag{2.18a}$$

$$y(k) = H(k)x(k) + v(k) \tag{2.18b}$$

$$x(0) = x_0 \tag{2.18c}$$

where $x(k)$, $u(k)$, $y(k)$, $w(k)$ and $v(k)$ represent the state, control, measurements and measurements noise vectors, respectively, and $A(k)$, $B(k)$, $G(k)$ and $H(k)$ represent the system transition, input distribution, system noise and observation matrices, respectively. $w(k)$ and $v(k)$ are Gaussian, white noises processes satisfying

$$\begin{aligned} &E\{w(k)\} = 0, && E\{v(k)\} = 0, \\ &E\{w(k)w(j)^T\} = \begin{cases} 0 & k \neq j \\ Q(k) & k = j \end{cases}, && E\{v(k)v(j)^T\} = \begin{cases} 0 & k \neq j \\ R(k) & k = j \end{cases}, \\ &E\{w(k)v(j)^T\} = 0, \end{aligned} \tag{2.19}$$

and have covariance matrices $Q(k)$ and $R(k)$, respectively, both positive definite [51].

It is assumed that the initial condition is also modelled as a Gaussian random variable with $E\{x(0)\} = x_0$ and $E\{x(0)x(0)^T\} = P(0)$. For this case, the dynamics and observations are linear and Gaussian, white noises and the conditional probability density functions (cpdfs), $p[x(k)|\{y(1), ...y(k)\}, \{u(0), ...u(k-1)\}]$, are Gaussian for any k. The Kalman Filter propagates this cpdf in order to obtain the state estimates. Let

$$p[x(k)|\{y(1), ...y(k)\}, \{u(0), ...u(k-1)\}] \sim \mathcal{N}(\hat{x}(k)^+, P(k)^+), \tag{2.20}$$

represent a Gaussian cpdf. The state estimate $\hat{x}(k)^+$ is the mean of this cpdf and the uncertainty of the state's estimate is expressed by the covariance matrix $P(k)^+$, with

$$\hat{x}(k)^+ = E\{x(k)|\{y(1), ...y(k)\}, \{u(0), ...u(k-1)\}\}, \tag{2.21a}$$

$$P(k)^+ = E\{(x(k) - \hat{x}(k)^+)(x(k) - \hat{x}(k)^+)^T|\{y(1), ...y(k)\}, \{u(0), ...u(k-1)\}\}. \tag{2.21b}$$

This means that the Kalman Filter only propagates the first (mean) and second (covariance) moments because the cpdfs are Gaussian. To evaluate $\hat{x}(k+1)^+$, the filter only uses previous estimate $\hat{x}(k)^+$ and new observation $y(k+1)$.

The Kalman Filter performs two steps: prediction, with $\hat{x}(k+1)^-$ representing the predicted estimate and update, with $\hat{x}(k+1)^+$ representing the updated estimate. During the prediction step, an error between the estimated state, $\hat{x}$, and the true state, x, and an error covariance matrix are computed accordingly with

$$\tilde{x}(k+1)^- = x(k+1) - \hat{x}(k+1)^-, \tag{2.22a}$$

$$P(k+1)^- = E\{\tilde{x}(k+1)^- \cdot (\tilde{x}(k+1)^-)^T\}. \tag{2.22b}$$

The same applies to the update step:

$$\tilde{x}(k+1)^+ = x(k+1) - \hat{x}(k+1)^+, \tag{2.23a}$$

$$P(k+1)^+ = E\{\tilde{x}(k+1)^+ \cdot (\tilde{x}(k+1)^+)^T\}. \tag{2.23b}$$

Considering an observer of the form [51]

$$\hat{x}(k+1)^{+} = \hat{x}(k+1)^{-} + K(k+1)[y(k+1) - H(k+1)\hat{x}(k+1)^{-}], \tag{2.24}$$

where $K(k+1)$ is the Kalman Gain and represents a weighted factor that aims to minimize the error covariance matrix $P(k+1)^{+}$. The Kalman Gain that minimizes equation (2.23b) can be written as

$$K(k+1) = P(k+1)^{-}H(k+1)^{T}[H(k+1)P(k+1)^{-}H(k+1)^{T} + R(k+1)]^{-1}, \tag{2.25}$$

and the error covariance matrix is updated using

$$P(k+1)^{+} = P(k+1)^{-} - P(k+1)^{-}H(k+1)^{T}[H(k+1)P(k+1)^{-}H(k+1)^{T} + R(k)]^{-1}H(k+1)P(k+1)^{-}. \tag{2.26}$$

As a consequence, the following equations summarize the functioning of the Kalman Filter.

Prediction Step:

$$\hat{x}(k+1)^{-} = A(k)\hat{x}(k)^{+} + B(k)u(k), \tag{2.27a}$$

$$P(k+1)^{-} = A(k)P(k)^{+}A(k)^{T} + G(k)Q(k)G(k)^{T}, \tag{2.27b}$$

Update Step:

$$\hat{x}(k+1)^{+} = \hat{x}(k+1)^{-} + K(k+1)[y(k+1) - H(k+1)\hat{x}(k+1)^{-}], \tag{2.28a}$$

$$P(k+1)^{+} = [I - K(k+1)H(k+1)P(k+1)^{-}]. \tag{2.28b}$$

For every step, the Kalman filter only needs to remember the previous step making it computationally efficient. It does not require high computational power making it a good choice for several applications.

2.6 Evaluation Metrics

In order to evaluate the algorithms used or developed for this thesis work, it is necessary to introduce adequate performance metrics [52, 53].

2.6.1 Object Detection

Intersection Over Union

The Intersection Over Union (IoU) predicts how well adjusted to the objects are the bounding boxes returned by the algorithms. It is given by

$$IoU = \frac{Area(B_p \cap B_{gt})}{Area(B_p \cup B_{gt})}, \tag{2.29}$$

where B_p and B_{gt} are the predicted and ground truth bounding boxes, respectively. If the two bounding boxes do not overlap, the IoU is equal to 0% and if they overlap entirely, it is equal to 100%. Two bunding boxes match if IoU $\geq 50\%$, as proposed in the PASCAL VOC competition [54].

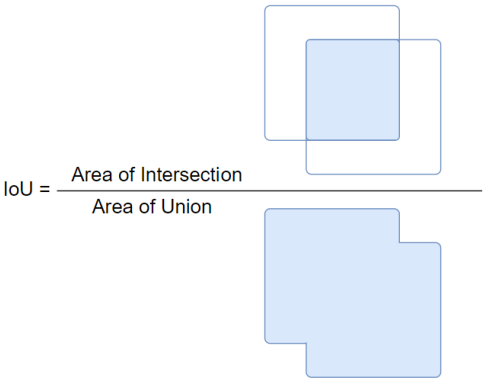


Figure 2.5: Visual representation of IoU.

Precision

The precision, P, translates the capacity to perform correct detections since it gives the relationship between the number of objects detected versus the total number of detections. It is defined as

$$P = \frac{TP}{TP + FP}, \tag{2.30}$$

where TP and FP are equal to the number of true positives and false positives detected, respectively. A TP is characterized by presenting an IoU superior to 50%, otherwise, it is considered to be a FP. In percentage, precision values vary from 0% to 100%, with $P = 100\%$ corresponding to an algorithm that does not detect any false positives, when there are TP detections.

Recall

The recall, R, translates the ability to find all the objects in the scene since it gives the relationship between the number of objects detected versus the total number of objects. It is defined as

$$R = \frac{TP}{TP + FN}, \tag{2.31}$$

where FN corresponds to the number of false negatives in a set. In percentage, recall values vary from 0% to 100%, with $R = 100\%$ corresponding to an algorithm that detects all the objects in the scene, when there are TP detections.

$\mathbf{F_1}$ Score

The F_1 score is the harmonic mean of the precision, P, and recall, R. It is defined as

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}. \tag{2.32}$$

Precision-Recall Curve

As explained before, when YOLO detects an object in an image, this prediction as a confidence score associated. When this score is lower than a certain confidence_threshold, the prediction is discarded. For each value of this threshold, the precision and recall presented by YOLO for the same set of images will be different. It is possible to plot the curve of the recall versus precision for each confidence score. If the confidence score is high, YOLO will present high precision but low recall values. On the other hand, if it is low, YOLO will have high recall values, but low precision.

Average Precision

The Average Precision (AP) corresponds to the area under the Precision-Recall curve. AP is given by

$$AP = \sum^{n} (R_n - R_{n-1}) \times P, \tag{2.33}$$

where R_n is the recall value of point n, and P is the maximum precision for any recall value larger than R_n, i.e., $P = max\{P(R) : R \geq R_n\}$.

Mean Average Precision

The mAP is the average of the AP across different classes, defined as

$$mAP = \frac{1}{N} \sum_{N}^{j=1} AP_j, \tag{2.34}$$

where N represents the number of total classes and AP_j the AP of the j^{th} class.

2.6.2 Model Performance

RMSE

The Root Mean Square Error (RMSE) has been used as a standard statistical metric to measure model performance. It is a measure of accuracy and it is always non-negative. It is defined as

$$RMSE = \sqrt{\frac{1}{N} \sum_{n=1}^{N} \left(\hat{x}_n - x_n\right)^2}, \tag{2.35}$$

where N is the total number of measurements, $\hat{x}_n$ is the estimated value of the n^{th} predicted measurement and x_n is its correspondent actual value.

A value of 0 would indicate a perfect fit to the data. In general, a lower RMSE is better than a higher one. Comparisons across different types of data are invalid since the measure is dependent on the scale of the numbers used.

MaxAE

The Maximum Absolute Error (MaxAE) is given by

$$MaxAE = max(|\hat{x}_n - x_n|), \tag{2.36}$$

where $\hat{x}_n$ is the estimated value of the n^{th} predicted measurement and x_n is its correspondent actual value. The MaxAE has the same unit of measurement as the data being predicted. Similarly to the RMSE, a value of 0 would indicate a perfect fit to the data.

Chapter 3

Methodology

This chapter begins with a discussion of the previous work developed at CfAR for the detection and tracking of non-cooperative UAVs that serves as a starting point for the work developed in this thesis. Subsequently, an overview of the final architecture is presented and three main tasks are discussed: detection, relative position estimation and algorithm adaptation for the problem of multi-UAV. A discussion of the method's limitations is also presented.

3.1 Previous Work

The work presented in this thesis carries on the work previously developed by Daniel Justino in [18] during his time at CfAR. Therefore, this section presents an overview of the algorithm that he developed for the task of detection and tracking of non-cooperative UAVs in an image using a LiDAR and a camera.

Using an object detector such as YOLO to inspect a whole image and detect UAVs is computationally expensive and can lead to the acquisition of a large number of false positives. For this reason, one of the largest contributions of the developed work was the implementation of a method that is capable of generating a Region of Interest (ROI) in the captured images, in which YOLO searches for the UAV.

Note the scheme present in Fig. 3.1 that represents the architecture proposed. The LiDAR detects and acquires a UAV by giving a 3D point cloud that is clustered and filtered by the algorithm to obtain just the points relative to the encountered UAV. These points are then projected into the 2D image using the geometric camera parameters and the projection perspective projection equation (2.14). Afterwards, a ROI centred in the 2D projected point is created with a fixed size of 128x128 pixels. The object detector YOLOv3 scans this ROI and tries to detect the UAV inside it. If a UAV is detected, the tracking algorithm based on Deep SORT is used to predict the position of the UAV in the following frames, i.e, for each instant t, the modified Deep SORT predicts the position of the ROI in the next frame, at $t+1$. For this reason, at instant $t+1$, YOLO will be inspecting ROIs created by the LiDAR 3D-2D projection and ROIs predicted by the modified Deep SORT. The modified Deep SORT tracker is especially important when LiDAR measurements are not available.

The discussed architecture detects and tracks UAVs in an image and outputs the UAV's pixel co-

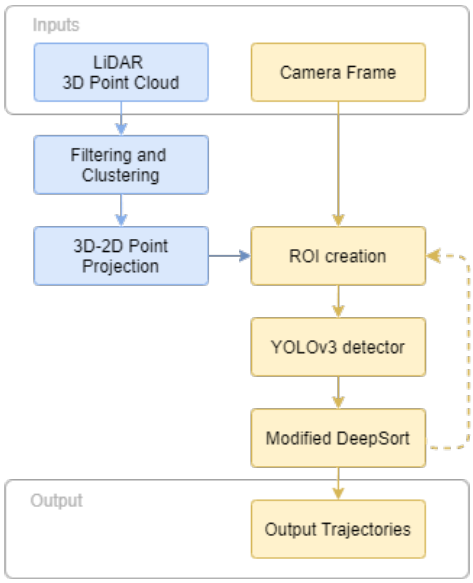


Figure 3.1: Previously proposed architecture. The dashed line represents a link between consecutive iterations [18].

ordinates in the image camera frame. The architecture is not capable of providing relative position estimations of the detected UAV, (X, Y, Z), in the world coordinate frame (see Fig.3.2). Therefore, the work presented in this thesis aims to continue the work discussed thus far and find a solution for the problem of relative position estimation of non-cooperative UAVs using a LiDAR and a camera.

Figure 3.2: Example of an output of the previously proposed system. LiDAR detections are identified by red dots, the ROI, ground truth and YOLOv3 detections correspond to the yellow, light blue and dark blue bounding boxes, respectively. Taken directly from [18].

3.1.1 LiDAR Point Cloud Filtering and Clustering

When filtering the point cloud given by the LiDAR, the objective is to remove points that do not correspond to any of the encountered UAVs. These points may result from reflections of the LiDAR beam on the ground, trees, buildings, etc. In this system, the filtering algorithm uses information from the sensor system's orientation and altitude to determine the boundaries corresponding to the region in which the UAVs may be flying.

Clustering is used to find the points reflected by the same UAV and assign them into clusters correspondent to the UAVs present in the scene. This task is performed by a density-based clustering solution called DBSCAN (Density-Based Spatial Clustering of Applications with Noise) [55] that requires the following two parameters: maximum distance between two points for them to be considered in the neighbourhood of the other (set to $0.5m$) and the minimum number of points necessary to form a cluster (set to 1).

3.1.2 Modified Deep SORT

The prediction of the position of the UAVs in the following frame is based on the MOT algorithm Deep SORT since its distance metric is a good fit for scenarios that deal with a small number of sparse flying UAVs. Deep SORT was planned for the tracking of pedestrians, where CNN appearance descriptors add value to the algorithm but, for the specific case of small UAVs, the CNN appearance descriptors are not particularly useful. For this reason, these descriptors were discarded and the algorithm is called modified Deep SORT.

3.2 System Overview

The previous architecture developed by Daniel Justino detects and tracks UAVs in images but cannot provide localization estimations for the encountered UAVs in the world coordinate frame. Therefore, this thesis aims to find a solution for the task of relative position estimation of non-cooperative UAVs using information from a LiDAR and a camera, providing relative coordinates (X, Y, Z) of the UAVs relative to the onboard payload, in the world coordinate frame.

The method proposed can be divided into two different steps: detection and relative localization estimation, as presented in Figure 3.3.

For the detection task, most of the work presented in section 3.1 was kept. For each frame, the 3D point cloud given by the LiDAR is filtered and clustered by the algorithm. When there are points likely to be related to an encountered UAV, a ROI is created using the 3D-2D projection. YOLO looks for UAVs in the ROIs created by the projection and by the modified Deep SORT tracker. If for a specific frame no ROI is created, the output of that frame is empty. The same happens if YOLO does not detect a UAV in any of the ROIs created for that frame. The modified Deep SORT is used to predict the position of the UAVs in the following frames. The algorithm constructs a track with the previous detections and uses this information to predict the position of the UAV in the following frame.

The main work of this thesis is focused on the second task, relative localization estimation. The position of the UAVs is estimated with two different methods. When YOLO detects a UAV with a correspondent LiDAR measurement, i.e., if there is a LiDAR 2D projection inside the bounding box returned by YOLO, the correspondent LiDAR 3D coordinates are used directly. If there are no correspondent LiDAR points for the detected UAV, position estimation is performed using the camera.

Because the LiDAR 3D point clouds are sparse, most of the estimations will be based on information

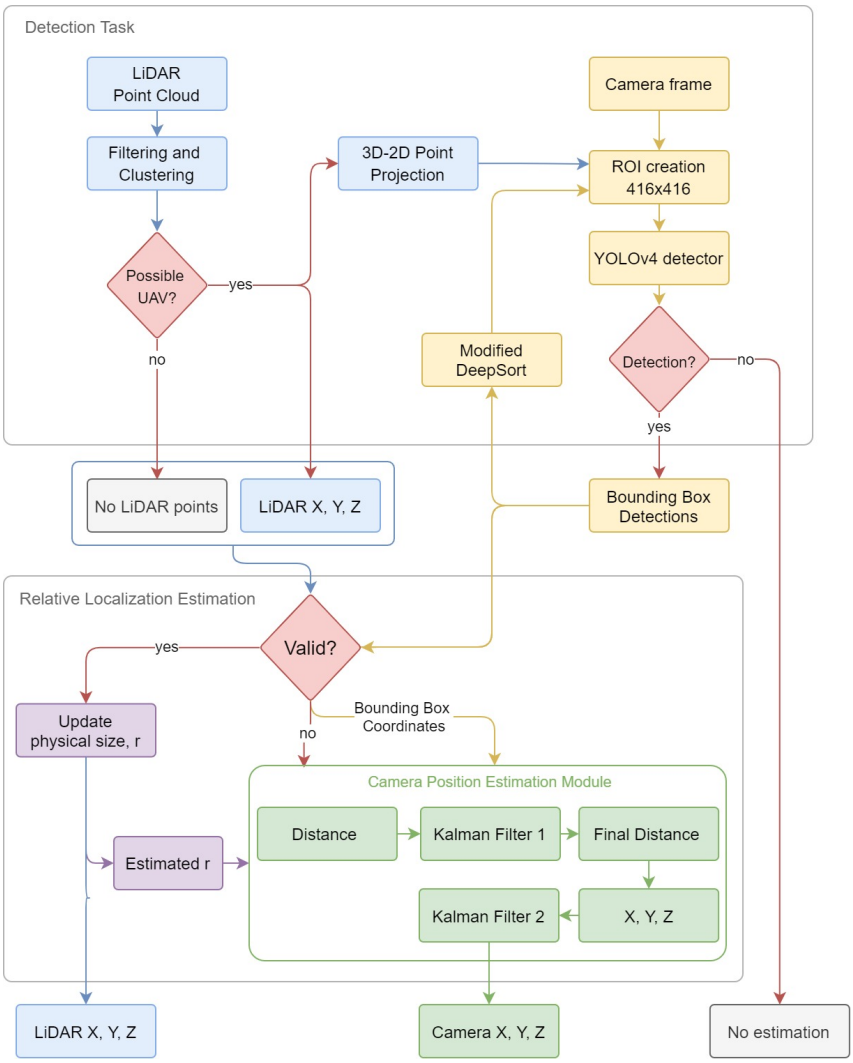


Figure 3.3: Diagram of the proposed system.

from the camera. Most of the solutions presented for this task that use a single camera assume a priori knowledge about the encountered UAVs. By introducing a LiDAR in the system, these methods can be generalized to cases where no a priori information about the target is necessary while maintaining a system that is simple, affordable and of small dimensions and weight.

To estimate the UAV's position with the camera, it is necessary to estimate its relative distance first. The method used to achieve this requires knowing the apparent physical size of the target, r, which will be discussed in detail in section 3.4, the bounding boxes returned by YOLO and the geometric camera parameters. The value of r is called apparent since it changes accordingly with variations in the attitude of the UAV. A Kalman Filter is used to smooth the estimates of the distance that afterwards is used to compute the relative 3D coordinates of the UAV in the LiDAR's coordinate frame.

The proposed system is suitable to be used in situations where conventional localization systems cannot be used and works for any number of UAVs present in the scene and without any previous information or assumption about these UAVs.

3.3 Detection Task

As mentioned before, for the detection task, most of the work presented in 3.1 was kept but some key updates were implemented.

Firstly, YOLO was updated to the last released version at the time of the training, YOLOv4, since this version was computationally faster and presented a larger $AP_{0.5}$ than the previously released frameworks. Also, the size of the ROI was changed from 128×128 pixels to 416×416 pixels, which is the standard size of the input images of YOLOv4, because it was observed experimentally that this size provided better detection results for the model trained.

The Deep SORT tracker was configured to keep a new track in the tentative state until $A_{max} = 3$ consecutive associations are successful and to remove this track if no associations are made for $N = 9$ iterations.

The coordinates of the bounding boxes returned by YOLO are used in the estimation of distance and 3D coordinates when using only the camera. As it will be explained in section 3.4, these bounding boxes need to fit around the target almost perfectly in order to achieve good results (see Figure 3.4(b)). For this reason, it was necessary to create a dataset that satisfied this requirement so that YOLO could be trained to return bounding boxes in these conditions. Most of the available datasets to train YOLO for UAV detection present bounding boxes similar to Figure 3.4(a) and do not satisfy the system's requirements, so it was necessary to develop a synthetic customized dataset using the open-source software AirSim [56].

(a) Actual label of the image. (b) Desired label for the image.

Figure 3.4: Example of a labelled image from the Drone-net dataset [57].

3.3.1 AirSim Overview

AirSim [56] is an open-source simulator that enables the development and testing of algorithms for autonomous vehicles and facilitates the collection of annotated data for the training of object detectors such as YOLO. AirSim was built on Unreal Engine [58] which offers realistic simulations, enabling the realization of Software-in-the-loop (SITL) and Hardware-in-the-loop (HITL) experiments (see Figure 3.5).

Figure 3.5: Snapshot that shows two aerial vehicles flying in a rural environment. The inset shows depth, object segmentation and front camera streams generated in real-time.

The core components of AirSim include environment model, vehicle model, physics engine, sensor models, rendering interface, public Application Programming Interface (API) layer and an interface layer for vehicle firmware (see Figure 3.6).

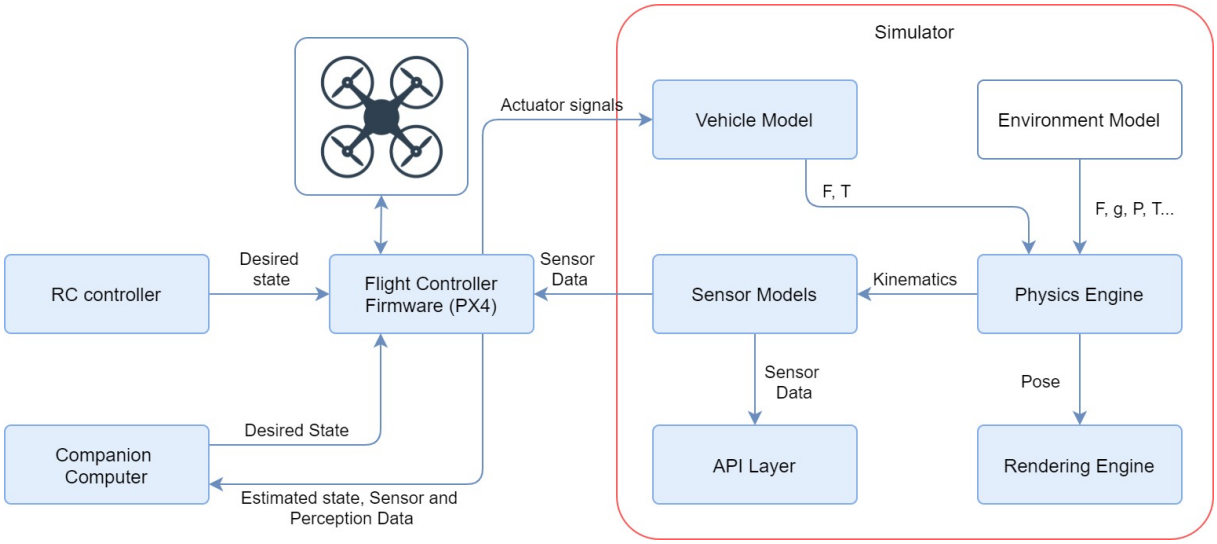


Figure 3.6: Architecture of AirSim that depicts the core components and their interactions. [56]

The simulator provides sensor data from the simulated world to the flight controller which outputs the actuator signals that are taken as input by the vehicle model.

AirSim is implemented as a plug-in for the Unreal engine and can be dropped into any Unreal project. The Unreal engine platform offers a marketplace with hundreds of pre-made detailed environments, turning AirSim into a powerful tool for gathering simulated training data in a large number of different environments. AirSim simulations can be personalized using the file *settings.json* which is a file in the JSON format [59] that allows the user to change the simulation accordingly with their needs.

Amongst other functionalities, the file allows the following modifications:

- **Type of simulated vehicles:** AirSim has built-in physics models for both cars and quadrotors. It is possible to configure the number of vehicles and their type.
- **Sensors:** AirSim allows the full personalization of the payload of each vehicle. Currently, AirSim supports cameras (scene, depth and segmentation images), LiDARs, barometers, IMUs, GPS, Magnetometers and Distance Sensors.
- **PawnPaths:** It is possible to specify the vehicle pawn blueprints and replace the default quadrotor in AirSim with another UAV mesh while maintaining the original physics model. It is also possible to change the physics model.
- **APIs:** AirSim exposes APIs so you can interact with the vehicle in the simulation programmatically, using Python code. It is possible to use these APIs to retrieve images, get the state of the vehicle, control it, etc.
- **Controller:** AirSim comes with its own controllers, but can be personalized to use external software, such as PX4 Autopilot [60]. When using the built-in controller, the vehicle can be moved using APIs and when using PX4 it can be moved using RC controllers or tools such as QGroundControl [61].
- **Recording:** The recording feature allows to record data such as segmented and scene images in real-time at an adjustable rate.

For the creation of the dataset to train YOLO, the simulations used PX4 Autopilot as a controller that received the desired state from the QGroundControl.

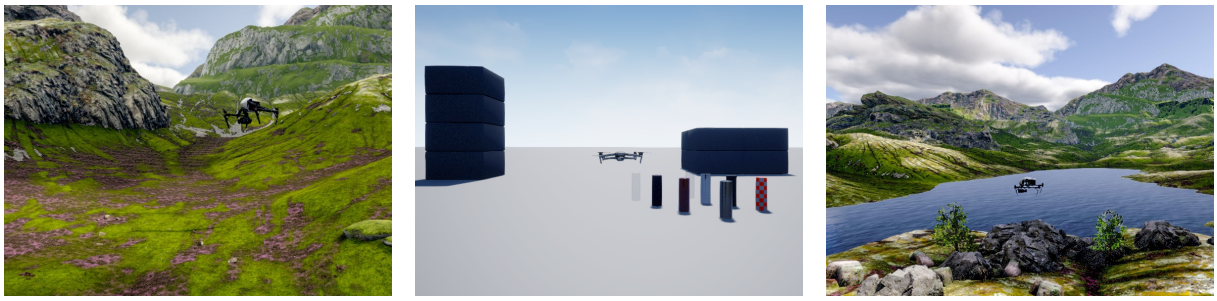
PX4 is an open-source flight control software for drones and other unmanned vehicles. It utilizes a port-based architecture so when developers add components, the extended system does not lose robustness or performance, making it highly modular and extensible in terms of both software and hardware. The system is designed to be coupled with embedded computer vision for autonomous capabilities, making it a good solution for developers working on localization and obstacle detection algorithms. PX4 can be used in various simulators such as AirSim or Gazebo [62].

QGroundControl provides full flight control and mission planning for drones with the Micro Air Vehicle Link (MAVLink) communication protocol [63] and all the code is open-source.

3.3.2 AirSim Simulation

During the simulations performed to gather training data, 4 different environments were used: *Blocks*, *Landscape Mountains*, *A Boy and His Kite*, and *Sun Temple*.

Two UAVs flew in every simulation so that the first UAV (carrying one camera) could record images of the second UAV. It was decided to train YOLO with images of two drones: DJI Inspire 1 and Mavic Pro considering that these were the drones available at CfAR and used in the experiments of previous



(a) Landscape from *A Boy and His Kite*. (b) Landscape from *Blocks*. (c) Landscape from *A Boy and His Kite*.

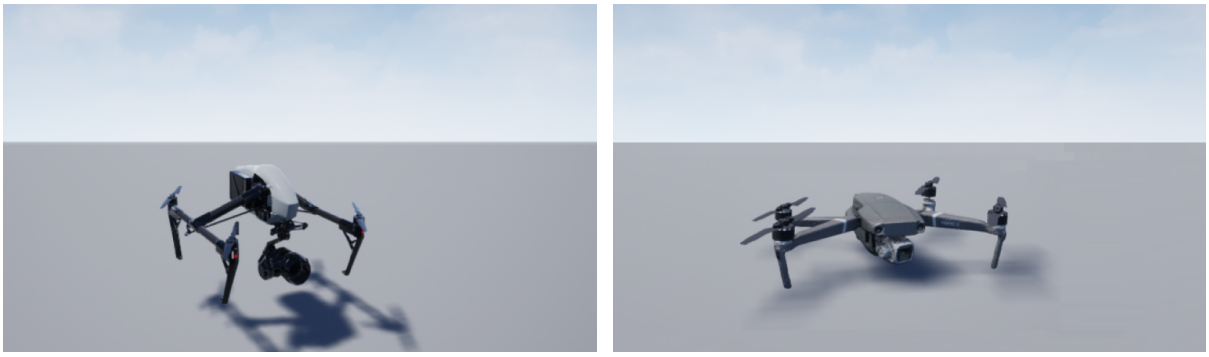
Figure 3.7: Images captured on different environments using AirSim and Unreal engine.

years. The specifications of these drones can be found in Table 3.1. For this reason, two *PawnPaths* meshes were created in Unreal engine, as presented in Figure 3.8. These meshes were taken from the *Sketchfab* website [64, 65] and edited in Unreal Engine so that the propellers could work with the quadrotor physics model built-in AirSim. YOLO could have been trained for other UAVs and even for other classes of objects, but in the scope of this thesis and taking into account the limited time available to build a dataset, it was trained just for these two.

Table 3.1: Properties of the vehicles used in the AirSim simulations.

	Dimensions [cm]	Weight [g]	Nº of Motors	Autonomy [min]
DJI Inspire 1	$44 \times 45 \times 30$	$2,845$	4	18
DJI Mavic Pro	$32 \times 24 \times 8$[1]	734	4	21

[1] unfolded



(a) PawnPath of the DJI Inspire 1. (b) PawnPath of the DJI Mavic Pro.

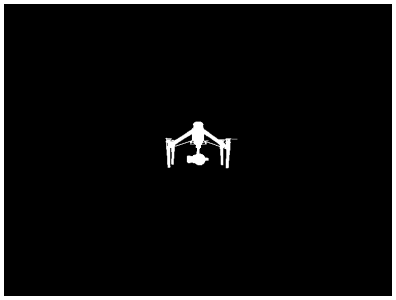
Figure 3.8: PawnPaths developed on Unreal Engine for AirSim.

In the *Blocks* and *Landscape Mountains* environments, the drones were recorded flying from the front, back, rotated by $\pm 90°$, rotated by $\pm 45°$ and moving randomly at both close and long distances. In the *A Boy and His Kite*, and *Sun Temple* environments, several different flights were performed with random trajectories that allowed to capture the drones with different attitudes and at different distances.

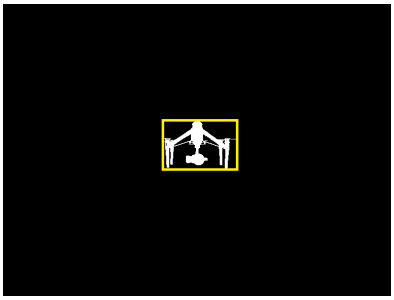
The camera frames were programmed to have 1920×1080 pixels, a horizontal Field of View (FOV) of $90°$ and a vertical FOV of $50.6°$. Alongside the scene camera frames, segmented images were collected. A Python code was created to use APIs to change the colours of the segmentation images

so that every object in the environment would appear black and only the drones would appear in white. This facilitated the process of finding the bounding boxes of the drones present in the image, using the Python library OpenCV [66]. These points were written to a text file accordingly to the YOLO darknet format, i.e., one text file per image containing annotations and a numeric representation of the label (0 or 1 in the case of DJI Mavic Pro or Inspire 1, respectively). The annotations were normalized to the range [0, 1] which makes them easier to work with even after scaling or stretching images.

(a) Scene image from *Landscape Mountains*.



(b) Segmented image correspondent to (a).



(c) Boundary points of the drone in (b).

Figure 3.9: Process for the creation of a dataset in the YOLO Darknet format.

A total of 4761 training images and 530 validation images were gathered to train the object detector. After the training, YOLO was tested on the 530 images from the validation set and presented a mAP of 99.03%, precision of 98.00%, recall of 98.00% and IoU of 83.11%.

3.4 Relative Localization Estimation

LiDARs are capable of giving 3D point clouds with high accuracy and at high frame rates but these point clouds are sparse and the UAVs are not detected when flying between the LiDAR beams. In addition, LiDAR beams can have small ranges. On the contrary, cameras can capture sequenced frames in which the UAV is always visible and YOLO can always perform UAV detection in these frames. However, single camera position estimation state-of-the-art methods require an a priori knowledge of the encountered UAVs and are usually used in cooperative UAV missions. By using information from the previous LiDAR detections as the necessary a priori knowledge for the implementation of single camera methods, these can be expanded to non-cooperative flying vehicles and the system can continue to estimate the UAV's position only with the camera even when there is no LiDAR data.

3.4.1 Position Estimation Using LiDAR

The LiDAR measurements are used directly whenever available and validated by YOLO, i.e., if the LiDAR 3D-2D projection results in points that are inside the bounding box returned by YOLO. In [67], the accuracy of distance measurement of a previous model of the LiDAR available at CfAR is tested in both outdoor and indoor environments. The data-sheet for this LiDAR displayed a range accuracy equal to $< 5cm$ (1σ at $50m$) and the results obtained are presented in table 3.2.

Table 3.2: M8-Quanergy LiDAR performance results.[67]

Experiment	Range of distances[m]	Error range[cm]
Indoor	2 to 14	1.79 to 4.47
Outdoor	10 to 40	1.70 to 6.86

It is possible to notice that the algorithm presented is fully dependent on the detections performed by YOLO. This decision was made taking into account that only an object detector can distinguish a UAV from a bird or the tip of a tree. For this reason, to use information from the LiDAR clusters, YOLO must detect the UAV in the same position too.

3.4.2 Position Estimation Using Camera

Whenever there are no measurements from the LiDAR, the relative position estimation is computed using only the camera. The presented method has been used for distance estimation using a single camera in several works such as [24] and [26], but always assumed knowledge about the encountered UAVs. In this work, we present a solution that expands this method to any non-cooperative UAV.

The task of relative position estimation can be divided into two phases: distance estimation and 3D coordinate estimation.

It is possible to notice from Fig. 3.10 that the distance d, from the centre of the UAV and the origin of the camera coordinate system, O_c, is estimated from the angle α which corresponds to the angle between vectors a_1 and a_2. These are directional vectors of 3D lines intersecting with O_c and the centres of the horizontal edges of the bounding box of the detected UAV.

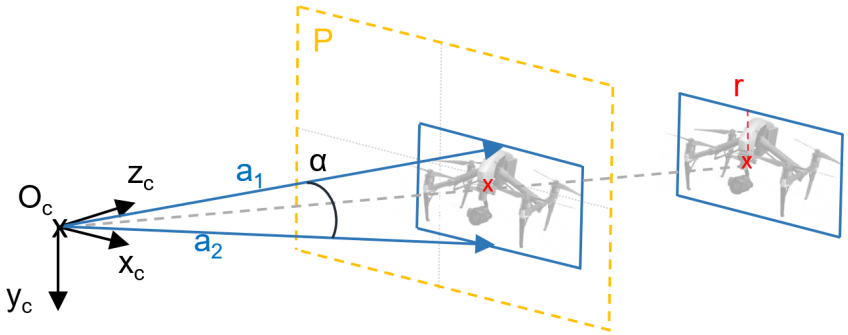


Figure 3.10: Scheme of the projection of the detected UAV to the camera projection plane P. The UAV is specified by the physical size r. The camera coordinate frame is defined by the camera origin O_c, vectors x_c, y_c and z_c.

The angle α between vectors a_1 and a_2 is given by

$$\alpha = acos\left(\frac{a_1 \cdot a_2}{\|a_1\| \cdot \|a_2\|}\right), \tag{3.1}$$

and, as a consequence, the approximate distance, d, to the detected object is given by

$$d = \frac{r}{tan(\alpha/2)}, \tag{3.2}$$

where r is the estimated physical size of the UAV.

Since we are developing a system that works for any non-cooperative UAV, it is necessary to find a way to estimate the target's physical dimension, r. To accomplish this, the last LiDAR measurement before losing the LiDAR information is used. Since the distance given by the LiDAR from frame $t-1$ and the corresponding bounding box returned by YOLO at $t-1$ are known, yields,

$$r(t-1) = d(t-1) \cdot tan(\alpha(t-1)/2). \tag{3.3}$$

In subsequent camera frames, at t, $t+1$ and so on, it is assumed r constant and equal to the last estimate, until a new LiDAR measurement occurs and the value of r is updated. The performance of the distance estimation improves as the number of LiDAR measurements increases since the value of r is updated more often and the algorithm takes into account more changes in the UAV's attitude.

Vectors a_1 and a_2 are directional vectors with origin in the camera coordinated frame (x_c, y_c, z_c) and end in the image plane. Thus far, the only information available is the coordinates $\mathbf{p} = (x, y)$ of the two points of the horizontal edges of the bounding boxes. To use this method to compute the 3D coordinates in the world coordinate frame, it is necessary to know the value of the coordinate z as well. However, as shown in section 2.3, it is impossible to calculate this value because the distance from the image plane to the camera origin, O_c, is unknown. For this reason, vectors a_1 and a_2 are obtained from the normalized image plane, which is at the known distance $\hat{z} = 1$. The relation between the vector of homogeneous coordinates in the image frame and the homogeneous coordinates in the normalized image plane was given by equation (2.12) and therefore, the points in the normalized plane can be obtained using equation 3.4,

$$\hat{\mathbf{p}} = \tilde{\mathcal{K}}^{-1}\mathbf{p}, \tag{3.4}$$

where $\tilde{\mathcal{K}}$ is the matrix with the intrinsic camera parameters and the radial and tangential distortion coefficients.

The position of the encountered UAV, in 3D coordinates and relative to the camera coordinate frame, is obtained using vectors a_1, a_2 and distance, d. Let vector a_c be the vector from the camera origin towards the centre of the detected UAV, C, yields:

$$a_c = \frac{a_1 + a_2}{2}, \tag{3.5a}$$

$$C = d\frac{a_c}{\|a_c\|}. \tag{3.5b}$$

The coordinates of point C are given with respect to a coordinate system centred in the camera. It is necessary to transform these coordinates to the LiDAR coordinate frame. This can be achieved using equation (2.13) and the extrinsic camera parameters.

Vectors a_1 and a_2 are computed using the horizontal edges of the bounding box instead of the vertical because the rotation of the propellers causes variations in the width of the bounding box which would increase the errors of the estimations. Notice Figures 3.11(a) and 3.11(b) in which the UAV presents the same attitude but due to the position of the propellers the bounding box in 3.11(b) presents a larger width.

(a) (b)

Figure 3.11: Effect of the propeller's movement in the bounding boxes returned by YOLO.

3.5 Method Limitations

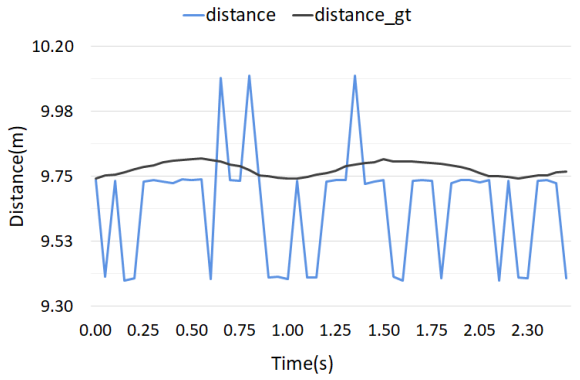
The bounding boxes returned by YOLO can vary sudden and significantly in size because the bounding boxes do not always fit around the detected UAV in the same way, inducing a random and unpredictable error. In addition, changes in the attitude of the UAV during its flight can lead to estimation errors. Notice Figure 3.12(a) and Figure 3.12(b) correspondent to similar positions (X, Y, Z) of a UAV but different attitudes. If the computation of the physical size, r, is made for the bounding box represented in Figure 3.12(a) and a few frames later, in the frame of Figure 3.12(b), the value of r has not been updated, the algorithm will predict that the UAV is at a closer distance than it is, because the height of the bounding box increased and the value of r remained the same. Situations where the UAV is predicted to be closer than it is are safer than the opposite case, where the UAV is predicted to be further than reality. In some cases, with very complex backgrounds, YOLO can detect only a part of the UAV, not bounding its body entirely (see Figure 3.12(c)). In this case, the height of the bounding box is usually smaller than expected and a strong increase in distance is estimated.

Since the distance estimation is dependent on the height of the bounding box returned by YOLO, which is given in pixels, there exists a quantization phenomenon associated with the smallest measurement unit - 1 pixel. In Figure 3.13 the distance estimated by the algorithm for distances of 10, 20, 30 and $40m$ is presented. It is possible to see that the values outputted always take discrete values that correspond to variations of a small number of pixels. It is also possible to notice that for distances of

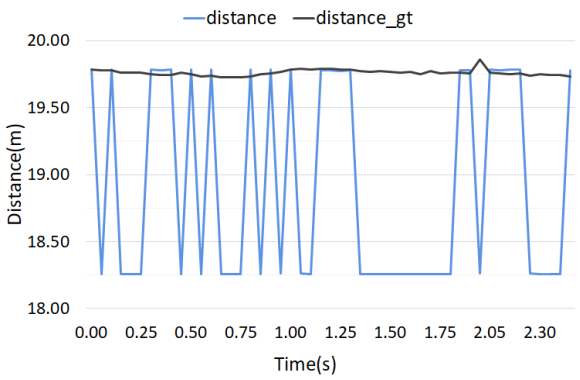
(a) (b) (c)

Figure 3.12: Examples of bounding boxes returned by YOLO.

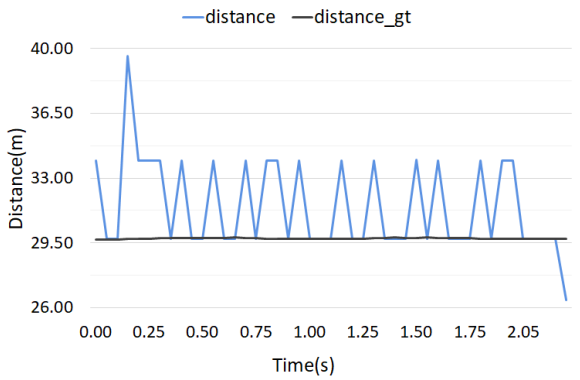
$10m$ a variation of 1 pixel corresponds to an absolute error in distance of $30cm$ but for a distance of $40m$, a variation of 1 pixel corresponds to an absolute error of $6.8m$ in distance, which is significantly larger. These results correspond to images taken with a camera with a resolution of $720p$ (images of size 1280×720 pixels) because this is the resolution of the camera available at CfAR. In later chapters, it will be studied the effect of using a camera with a higher resolution ($1080p$).



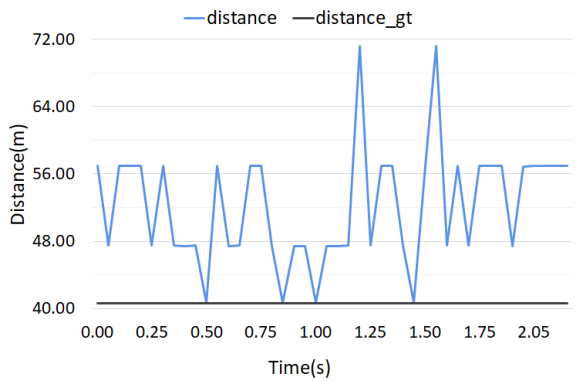
(a) Estimated distance at $10m$.



(b) Estimated distance at $20m$.



(c) Estimated distance at $30m$.



(d) Estimated distance at $40m$.

Figure 3.13: Estimated distance versus ground truth.

A Kalman Filter was implemented for the distance estimation and 3D coordinate stabilization (when using the camera) to smooth all the variations discussed.

3.6 Kalman Filtering

The Kalman Filter does not require high computational power, making it a good choice for the applications intended in this thesis and can be used to smooth the algorithm's estimations, avoiding sudden and strong variations.

3.6.1 Distance Stabilization

For the distance estimation, it was considered a very simple linear, discrete-time system having dynamics

$$x(k+1) = A(k)x(k) + B(k)u(k) + G(k)w(k), \qquad t \geq 0 \tag{3.6a}$$

$$y(k) = H(k)x(k) + v(k), \tag{3.6b}$$

$$x(0) = x_0, \tag{3.6c}$$

where $x(k)$, $u(k)$, $y(k)$, $w(k)$ and $v(k)$ represent the state, control, measurements and measurements noise vectors, respectively, and $A(k)$, $B(k)$, $G(k)$ and $H(k)$ represent the system transition, input distribution, system noise and observation matrices, respectively. $w(k)$ and $v(k)$ are Gaussian, white noises processes satisfying Equation (2.19) and have covariance matrices $Q(k)$ and $R(k)$, respectively.

The considered state transition matrix is based on a linear velocity model,

$$A(k) = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}_{2\times 2}, \tag{3.7}$$

and the input $u = 0$ since there are no control inputs. The Kalman Filter is initialized with estimates of $\hat{x_0}$, and predicted (a priori) estimate covariance, P_0, given by

$$\hat{x_0} = \begin{bmatrix} d(t=0) \\ \dot{d}(t=0) \end{bmatrix}_{2\times 1}, \tag{3.8a}$$

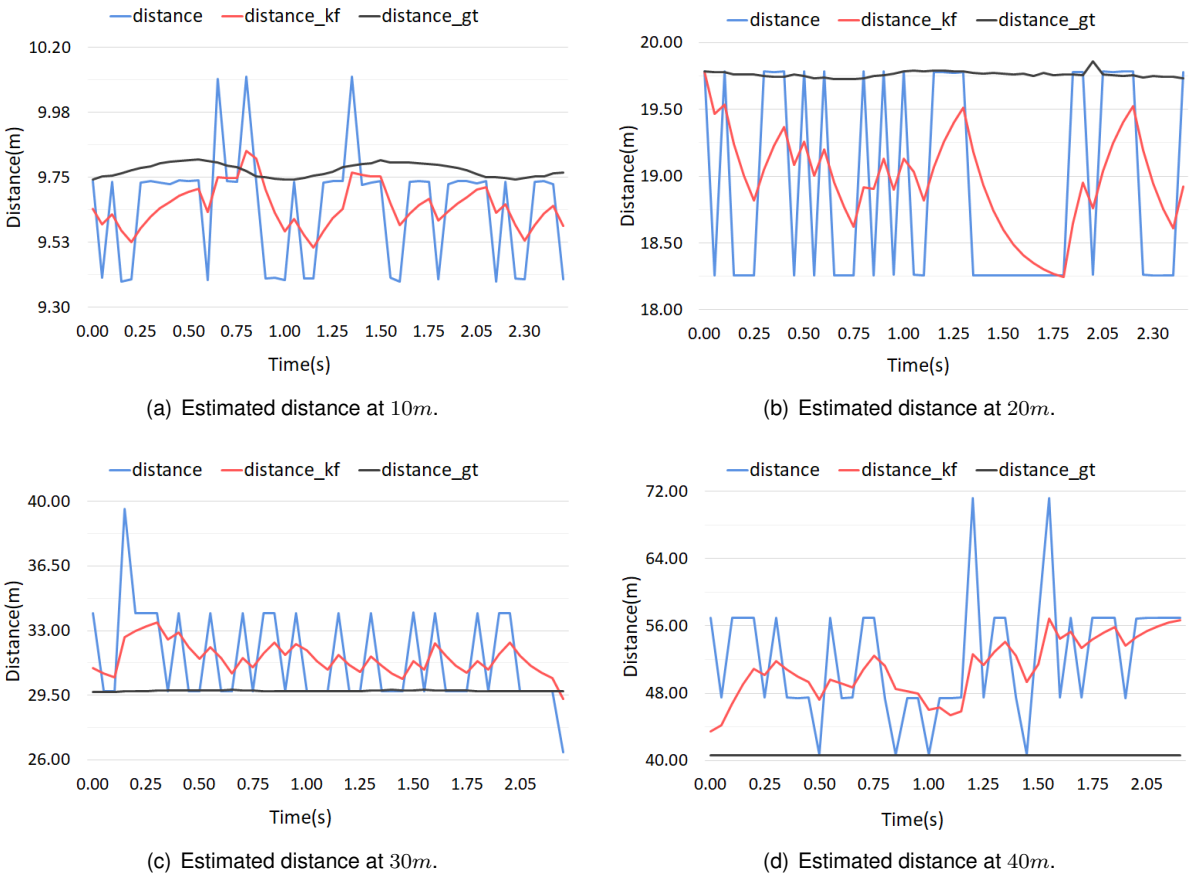
$$P_0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}_{2\times 2}, \tag{3.8b}$$

where $d(t=0)$ is computed with the last available LiDAR measurement and $H(k) = \begin{bmatrix} 1 & 0 \end{bmatrix}_{1\times 2}$.

The covariance of the measurements noise matrix, $R(k)$, and the process noise matrix, $Q(k)$, were tuned manually. The results obtained for constant distances of 10, 20, 30 and $40m$ are presented in figure 3.14.

3.6.2 3D Coordinates Stabilization

For the 3D coordinates stabilization, it was used a linear, discrete-time system of the form presented in Equation 3.6c. The state transition matrix is based, once again, in a linear velocity model with no



(a) Estimated distance at $10m$.

(b) Estimated distance at $20m$.

(c) Estimated distance at $30m$.

(d) Estimated distance at $40m$.

Figure 3.14: Estimated distance before and after Kalman Filtering.

control inputs ($u = 0$). The Kalman Filter is initialized with estimates of $\hat{x_0}$, and predicted (a priori) estimate covariance, P_0, given by

$$\hat{x_0} = \begin{bmatrix} x(t=0) & y(t=0) & z(t=0) & \dot{x}(t=0) & \dot{y}(t=0) & \dot{z}(t=0) \end{bmatrix}^T_{6\times 1}, \tag{3.9a}$$

$$P_0 = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \\ 0 & & 1 \end{bmatrix}_{6\times 6}, \tag{3.9b}$$

where $x(t=0)$, $y(t=0)$ and $z(t=0)$ are given by the last available LiDAR measurement and

$$H(k) = \begin{bmatrix} 1 & \dots & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & & \vdots \\ 0 & & 1 & \dots & 0 \end{bmatrix}_{3\times 6}. \tag{3.10}$$

Matrices $Q(k)$ and $R(k)$ were tuned manually once again.

3.7 Implementation for Multi-UAV

In order to be able to detect and estimate the position of several UAVs in a sequence of images, it is necessary to find a method to associate each detected UAV in a new frame with its previous detection in the last frame, creating the UAV's track. This is an assignment problem that will be solved using a simple distance metric. Future work involves the usage of Deep SORT's association metrics and possibly its CNN appearance feature descriptors to associate UAVs between frames. However, this method has not yet been implemented in the algorithm proposed in this work.

For simplicity, the algorithm creates a list containing the information of all the UAVs present in the scene. For each UAV, the list has information regarding:

- the coordinates of the centre of the last bounding box returned by YOLO, (c_x, c_y);
- the estimate of the apparent physical size, r;
- a Kalman Filter for the distance and a Kalman Filter for the 3D coordinates;
- a parameter that counts the number of frames since the last successful association, COUNTER_DEAD;
- a flag that allows to know if the UAV has already been updated in the current frame, COUNTER_FRAME;
- the UAV identification, UAV_ID;
- the last LiDAR measurement and correspondent YOLO detection for the computation of the physical size, r.

For each frame, the algorithm searches for UAVs in the list that have been lost for more than LOSS_THRESHOLD frames and removes them. This guarantees that the list does not accumulate data regarding lost UAVs, occupying more memory and slowing down the execution during the assignment task. This also prevents wrong assignments between the detected UAVs and old lost UAVs. It is desired to set the LOSS_THRESHOLD to a value greater than 0 since YOLO can fail to detect a UAV in some frames because the background is complex, for example, but still be able to detect the UAV a few frames later. In this situation, it is desirable to be able to associate the UAV with its previous detection at N frames ago, so that it is possible to know its apparent physical size, r and estimate its relative localization.

Note Figure 3.15, for each ROI, YOLO tries to detect a UAV. If a new detection is confirmed, the algorithm checks if the list of UAVs is empty. When it is, the new detected UAV is added to the list, if not, the algorithm tries to assign the detected UAV with the ones already present in the list.

The assignment is performed based on the coordinates of the centre of the bounding box returned by YOLO, (c_x, c_y), in pixels. These coordinates are compared to the centres of the bounding boxes of all the other UAVs present in the list, according to

$$s = \sqrt{(c_x - c_x^i)^2 + (c_y - c_y^i)^2}, \tag{3.11}$$

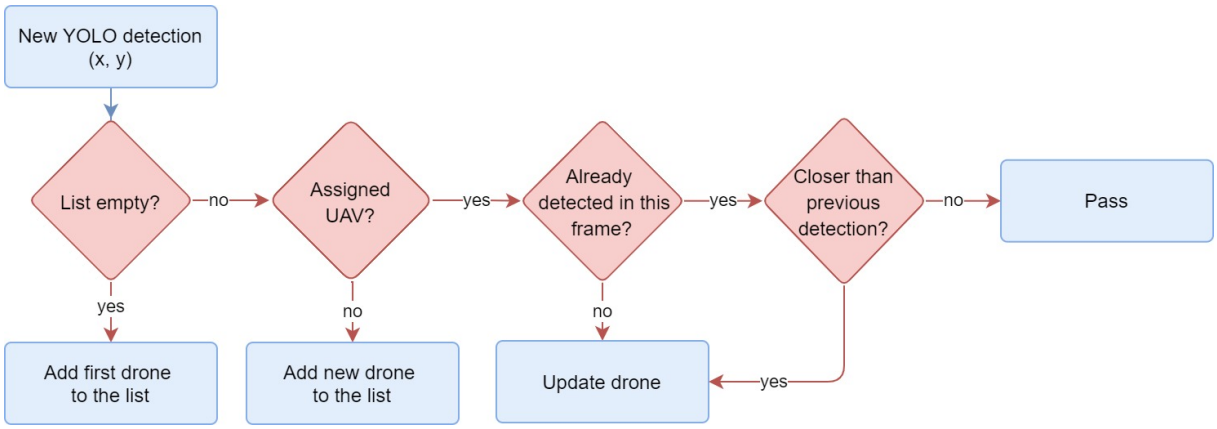


Figure 3.15: Architecture of the Multi-UAV algorithm.

where (c_x^i, c_y^i) represents the centre of the bounding box of the UAV with correspondent index i in the list of vectors. This represents a circle of radius s around the centre of the bounding box. The algorithm chooses the UAV correspondent to the smallest value of s and compares this value with a threshold. If $s \leq$ DETECTION_THRESHOLD, the assignment is validated and the algorithm proceeds, if not, the detection is considered to be a new UAV that just entered the scene and is added to the list.

Because ROIs have a size equal to 416×416 pixels, a ROI may capture more than one UAV. Sometimes, one of the UAVs could even be cropped and as consequence the bounding box returned by YOLO could not bound entirely the UAV, providing a smaller height than what was expected (see Figure 3.16). As a consequence, the estimated distance would be much further than the real value. The solution found to correct this problem is to verify, after the assignment, if the UAV has already been updated for that specific frame and if so, choose the estimation that presents the closest distance, since this is the safest approach when trying to avoid a potential collision.

Figure 3.16: Example of a ROI that captures two UAVs.

The developed algorithm can only fail to assign and update the detected UAV properly in the following two situations:

- If two or more UAVs are flying at a distance that is closer than the DETECTION_THRESHOLD;
- If the detected UAV is at a distance closer than the DETECTION_THRESHOLD from the last known

position of another UAV that was lost at N frames ago with $N \leq$ LOSS_THRESHOLD. This corresponds to a time window of LOSS_THRESHOLD$\times T$ seconds, where T is the camera's sampling rate.

In both these situations, the assignment fails only if the distance metric, s, computed between the detected UAV in the i^{th} frame and all the UAVs in the $i^{th} - 1$ frame gives a smaller number of pixels between two different UAVs. One of the solutions to guarantee good results is having a high frame rate. However, the system is limited by the execution time of YOLO and as the number of UAVs increases, YOLO needs to analyse a larger number of ROIs and the execution time increases as well.

It is possible to conclude that the algorithm only needs to store information regarding the last detection (for each UAV) making it computationally less expensive.

Chapter 4

Simulations and Results

This chapter presents simulations performed for the testing of the proposed system and their correspondent results. The algorithm is tested with images gathered during simulations in Unreal Engine with AirSim. Firstly, a flight with varying pre-defined yaw is performed to study the influence of attitude variations in the estimations performed by the algorithm. Also, a flight simulation with a camera of higher resolution is used to study the influence of camera resolution on the system's output. Finally, a multi-UAV flight test is presented to test the assignment problem and show the system's capabilities regarding the detection and tracking of multiple encountered UAVs.

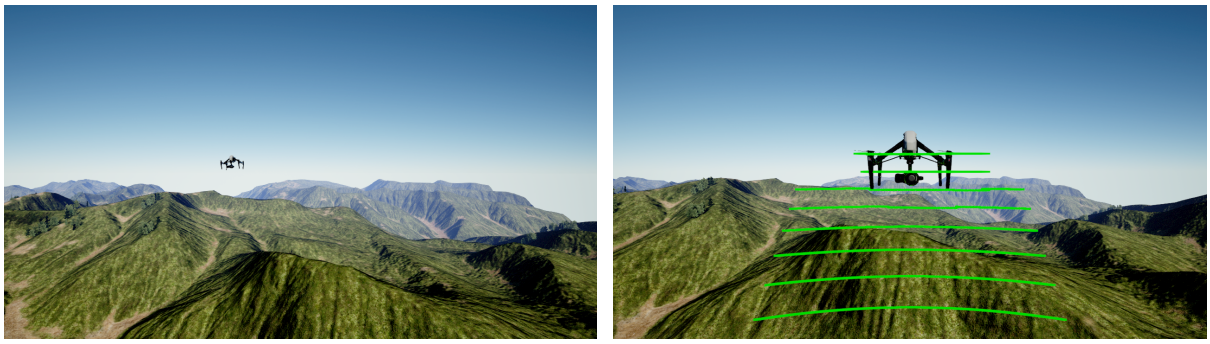
4.1 AirSim Setup

The simulations in AiSim were performed with QGroundControl and transmitted to AirSim by the PX4 controller. The UAV used was a DJI Inspire 1 (see Table 3.1) since this is one of the drones available at CfAR.

The chosen environment for the flight simulation was different from the ones used to train YOLO so that the results could be more realistic. A picture of the environment is presented in Figure 4.1(a) and it is composed of a landscape with a clear sky, where a UAV should be easily detected and mountains simulating complex backgrounds with dark areas where it is difficult to see a UAV flying at large distances, even for the human eye.

A UAV carrying a camera and a LiDAR stayed hovering at a fixed position while the other UAVs, representing encountered non-cooperative flying vehicles, followed trajectories given by the QGroundControl. The LiDAR was simulated using a built-in tool from AirSim (see Figure 4.1(b)) and presented the same characteristics as the LiDAR available at CfAR: 8 detection layers, 420000 points per second, a range of $60m$, a horizontal FOV of 360° , a vertical FOV of 21° $(+3^\circ / -18^\circ)$ and 10 rotations per second.

Python code was developed to capture LiDAR measurements, ground truth and camera images during the flights, using APIs. The trajectory was planned so that the maximum velocity of the flying UAVs was equal to $4m/s$. The camera was configured to output images of size 1280×720 since this is the resolution of the camera available at CfAR. It presented a horizontal FOV of 90° and a vertical FOV



(a) Environment for the flight simulations.

(b) Buil-in LiDAR reprojection points.

Figure 4.1: AirSim flight simulations.

of $50°$. The simulations were configured to give a data rate of $20Hz$.

The values of the thresholds used in the simulations were the following: LOSS_THRESHOLD$= 15$ frames, DETECTION_THRESHOLD=64 pixels and LIDAR_TRESHOLD=15 pixels. Finally, the matrices $Q(k)$ and $R(k)$ from the Kalman Filters implemented for the smoothing of both the distance and 3D coordinate estimations are presented in Equations 4.1 and 4.2.

Distance Kalman Filtering

$$R(k) = 65, \qquad Q(k) = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix}. \tag{4.1}$$

3D Kalman Filtering

$$R(k) = \begin{bmatrix} 9 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 25 \end{bmatrix}, \qquad Q(k) = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 \end{bmatrix}. \tag{4.2}$$

In Figure 4.2 it is presented the appearance of the simulated UAVs at different distances ranging from $10m$ to $50m$ in pictures of size 416×416 since this is the input size of YOLO. It is possible to notice that for $Z \approx 40m$ and $Z \approx 50m$ the UAV appears very small and with a camera of resolution equal to $720p$ it is expected that attitude variations will not influence the algorithm's estimations as much as for closer distances. In fact, for such values of Z, the distance estimation variations associated with the quantization of the height of the bounding boxes (in pixels) will have a much larger impact on the position estimations.

To test the developed algorithm with the data gathered in AirSim, it was necessary to estimate the camera's intrinsic and extrinsic parameters, discussed in section 2.3.

(a) UAV appearance at $10m$.

(b) UAV appearance at $20m$.

(c) UAV appearance at $30m$.

(d) UAV appearance at $40m$.

(e) UAV appearance at $50m$.

Figure 4.2: Appearance of the UAV at Z ranging from 10 to $50m$ with a $720p$ camera and in a ROI with size 416×416.

4.1.1 Camera Calibration

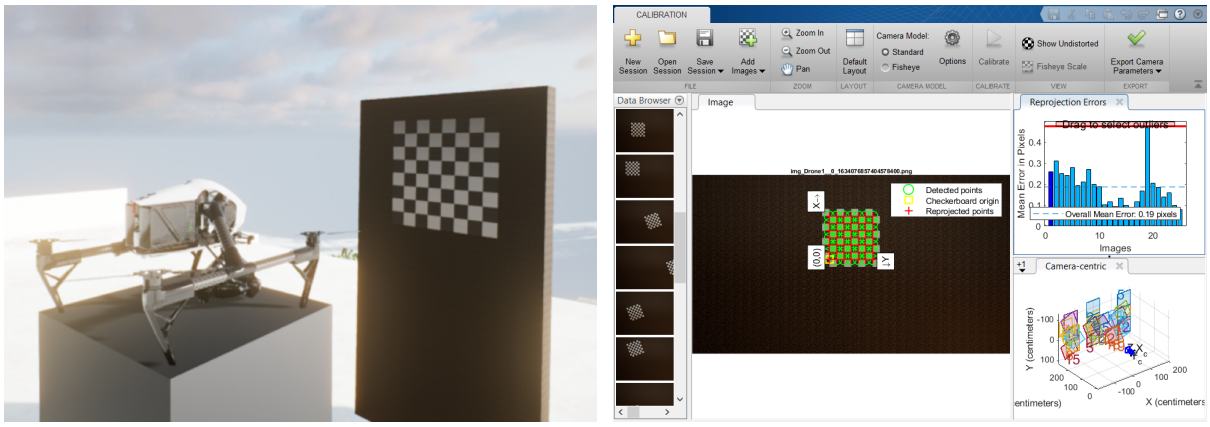
As mentioned throughout this document, the intrinsic and extrinsic camera parameters are necessary for the following tasks:

- project the 3D points given by the LiDAR into the camera frame and create the ROI in which YOLO will look for potentially encountered UAVs;
- transform the pixel coordinates of the bounding boxes returned by YOLO into distances and 3D coordinates in the camera frame;
- change the 3D coordinates from the camera's coordinate frame to the LiDAR's coordinate frame.

The Camera Calibrator App from Matlab [68] was used for the estimation of the intrinsic parameters. This app takes as input a set of images with a chessboard in different positions and the physical size of each square in the real world and computes the camera parameters that best fit the given data.

It was necessary to create a chessboard in Unreal Engine and record images with it in different positions and angles (see Figure 4.3).

The calibration app gave the results present in equation (4.3) which are close to the theoretical values expected for a camera in AirSim.



(a) Chessboard simulated in Unreal Engine.

(b) Results from the Camera Calibrator App.

Figure 4.3: Camera intrinsic calibration procedure.

$$\mathcal{K} = \begin{bmatrix} 644.34 & 0 & 0 \\ 0 & 644.27 & 0 \\ 639.76 & 359.91 & 1 \end{bmatrix} \tag{4.3}$$

$$k_1 = 0.0056, \quad k_2 = 0.0032, \quad k_3 = -0.0054$$

$$p_1 = 0, \quad p_2 = 0$$

For the extrinsic calibration between the LiDAR and the camera, IPPE was used to estimate the parameters. This algorithm was described in 2.3.5 and it was implemented using the Python library OpenCV which has a function that computes the parameters based on a set of pairs of 3D points in the real world and their correspondent 2D projections in the camera image. These points were gathered manually by visual inspection of images taken with the simulated system. An example of a 3D-2D projection after performing the camera calibration is presented in Figure 4.4.

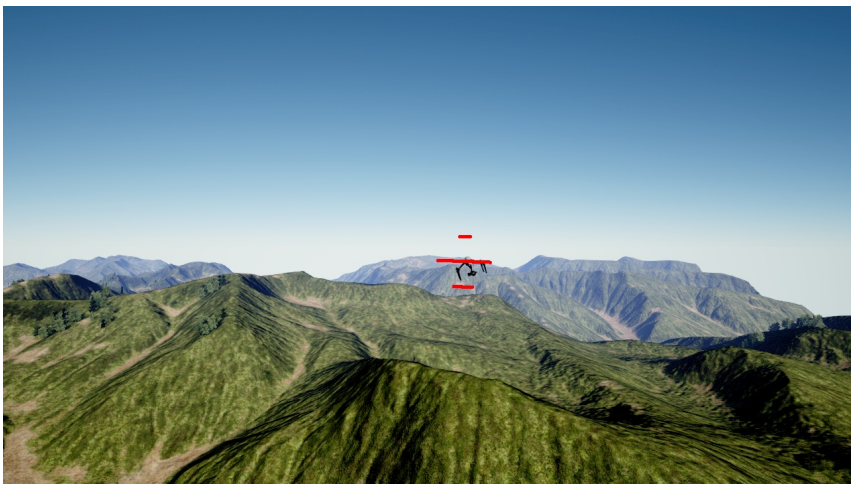


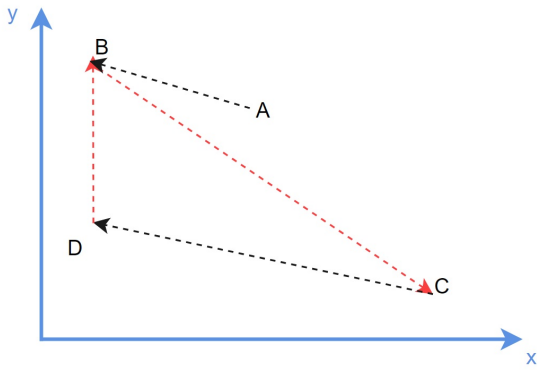
Figure 4.4: Example of 3D-2D projection after the extrinsic calibration.

4.2 Yaw Simulation

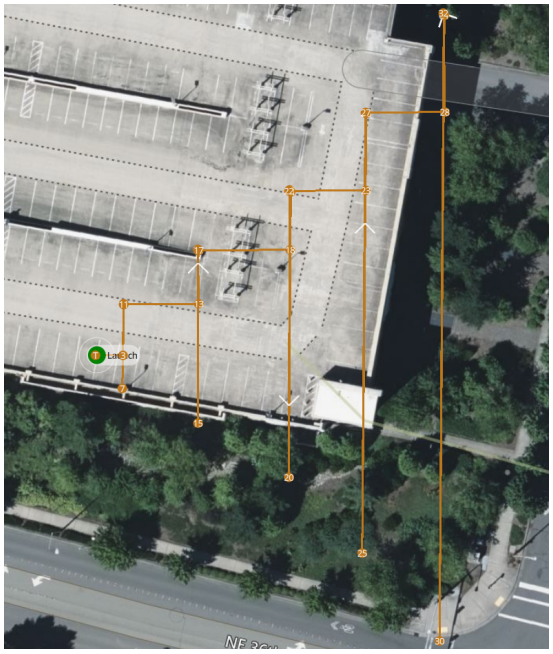
This flight was simulated to study the influence of attitude variations in the algorithm's predictions. QGroundControl only allows to control the yaw of the flying UAV but the height of the bounding box given by YOLO is influenced by roll and pitch as well. For this reason, the roll and pitch during the flight will change according to the UAVs trajectory and influence the results obtained. Also, when the UAV reaches a trajectory point in which it must change its yaw to the new designated value, yaw varies for a short period time. Regardless, a simulation with well-defined yaw was already enough to infer several conclusions regarding the influence of attitude variations.

For this flight, the relative position estimation was performed for just one UAV. It flew along the Z-axis at positions ranging from 10 to $50m$ in intervals of $10m$. In Figure 4.5(b) it is possible to see the trajectory plan from above in QGroundControl. For each distance, the UAV follows the trajectory represented in Figure 4.5(a), in the OXY plane, following the sequence A-B-C-D-B. Afterwards, the UAV flies at constant pair of (X, Y) coordinates to the following value of Z and performs the same sequence.

The UAV presents yaw equal to $0°$ from point A to B and from point C to D and yaw equal to $90°$ from point B to C and D to B. Images representing the appearance of the UAV in these two cases is presented in Figure 4.6. Notice that the height of the bounding box returned by YOLO is smaller when the UAV presents a yaw of $90°$.



(a) Trajectory followed in the OXY plane.

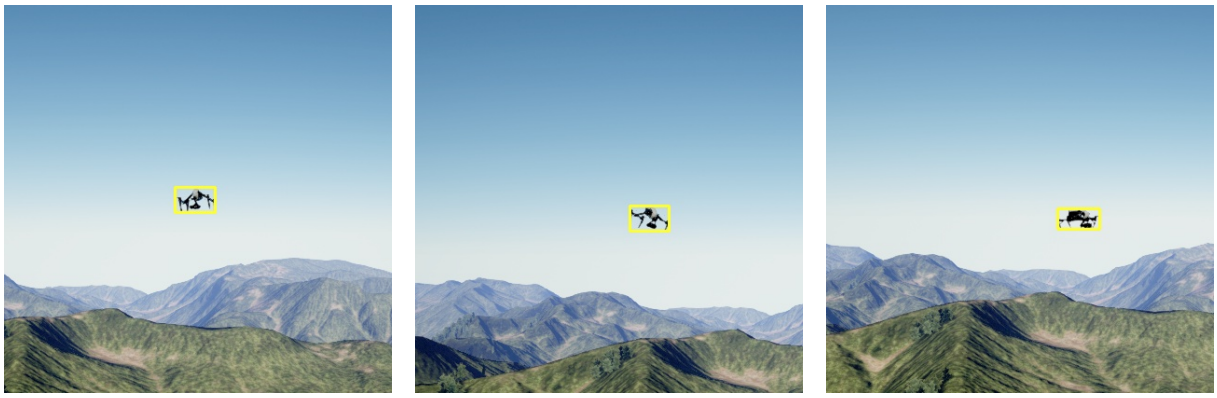


(b) Trajectory followed in the OXZ plane.

Figure 4.5: Trajectory followed by the UAV for the yaw simulation.

4.2.1 Results

The results obtained in this flight test are presented in Table 4.1 and Figures 4.7 to 4.9. In these figures, it is possible to distinguish values correspondent to the ground truth, given by AirSim, points



(a) UAV appearance with yaw of 0° at $Z \approx 10m$.

(b) Example of a UAV appearance when transitioning from yaw of 0° to 90° at $Z \approx 10m$.

(c) UAV appearance with yaw of 90° at $Z \approx 10m$.

Figure 4.6: UAV appearance with yaw varying from 0° to 90° at $Z \approx 10m$.

correspondent to the LiDAR measurements and points correspondent to positions estimated by the algorithm when there are no LiDAR measurements. The output of the system is exactly the one presented in the figures. The algorithm was tested on a Virtual Machine with 4 available INTEL(R) XEON(R) CPU E5-1620 V4 of $3.5GHz$ and a Random Access Memory (RAM) equal to $8GB$ and in Google Colaboratory [69] with a TESLA P100-PCIE-16GB GPU in a high RAM environment.

By analysing Table 4.1 it is possible to conclude that both RMSE and MaxAE increase as Z increases. The distance, d, presents the highest errors followed by the estimations along the Z-axis.

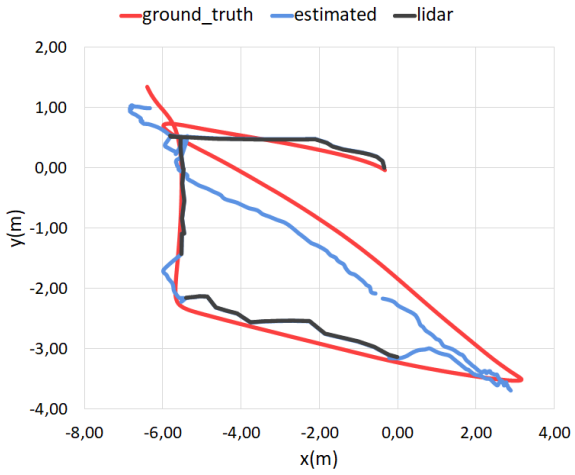
Table 4.1: RMSE and MaxAE results obtained for the yaw simulation.

	RMSE[m]				MaxAE[m]			
	X	Y	Z	d	X	Y	Z	d
$10m$	0.43	0.19	0.80	0.86	1.25	0.39	1.93	2.19
$20m$	0.80	0.34	1.28	1.45	2.93	1.21	3.78	4.73
$30m$	2.00	1.13	3.56	4.17	5.53	3.67	10.27	12.18
$40m$	2.12	1.30	4.03	4.62	7.11	3.83	10.53	12.91
$50m$	4.35	1.58	7.36	8.60	16.94	5.99	23.21	28.76

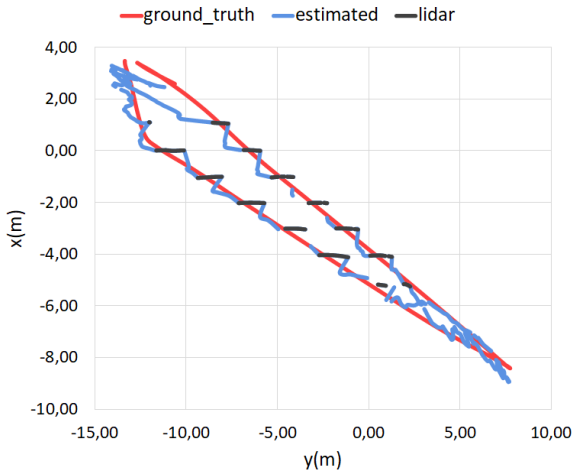
Noticing also the graphics from Figure 4.7 it is possible to conclude that position estimations in the OXY plane present good results, with RMSE values lower than $4.34m$ for estimations in X and $1.58m$ for estimations in Y. Note that the UAV moved between $-50m$ and $+40m$ in X and between $-12m$ and $+16m$ in Y, so the error associated with the Y-axis is smaller. The error in the Z-axis is the most affected, presenting a MaxAE of $23.21m$ and a RMSE of $7.38m$ for $Z \approx 50m$. Overall, it can be concluded that the algorithm performs well for positions such that $Z \leq 40m$ but for higher values of Z the estimates start to deviate significantly from the ground truth values. The fact that the estimates given by the algorithm improve as the UAV comes closer to the system is desirable because the system will have good estimates that can be used in avoidance algorithms for UAVs that are closer and that may be potentially dangerous.

Noticing the distance estimation errors, it is possible to verify that the distance influences directly

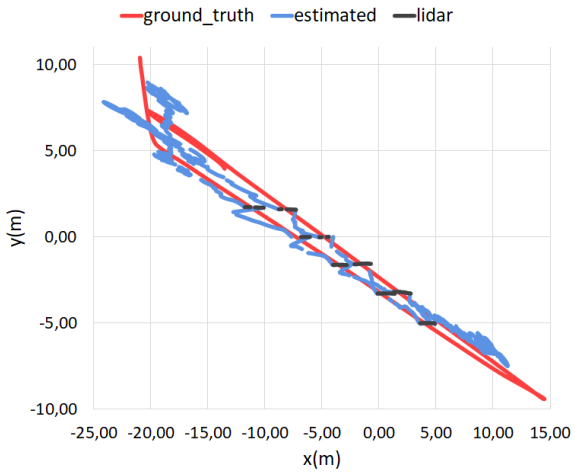
the error of the 3D coordinates, because it is used to compute them. When the error in the estimated distance increases, the error in each one of the three coordinates increases as well.



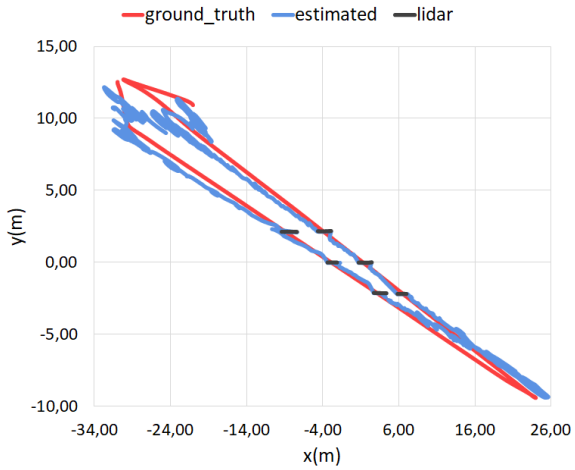
(a) Estimated X and Y at $10m$.



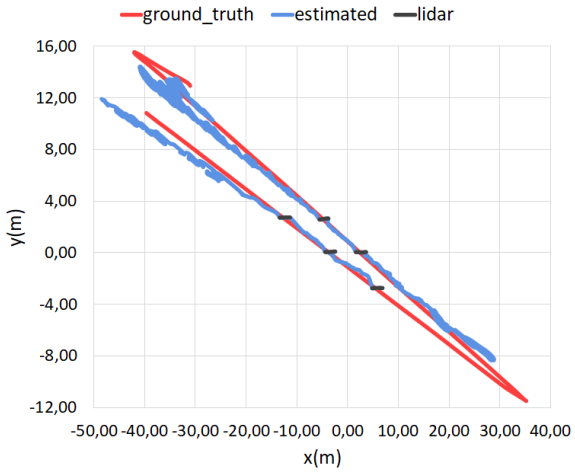
(b) Estimated X and Y at $20m$.



(c) Estimated X and Y at $30m$.



(d) Estimated X and Y at $40m$.



(e) Estimated X and Y at $50m$.

Figure 4.7: Results obtained for the OXY plane in the yaw simulation.

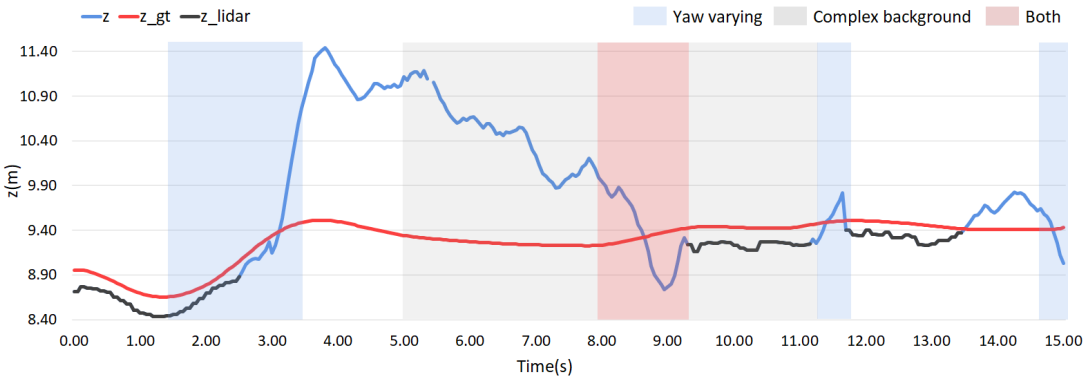
It is possible to distinguish different coloured regions in the graphics of Figures 4.8 and 4.9, correspondent to:

- **Attitude Variation** regions, where the attitude of the UAV is varying significantly in both yaw, pitch and roll due to transitions between trajectory points and/or different attitudes;
- **Complex background** regions, where the UAV was flying against a complex background;
- **Both** regions, where the UAV was changing its attitude while flying against a complex background.

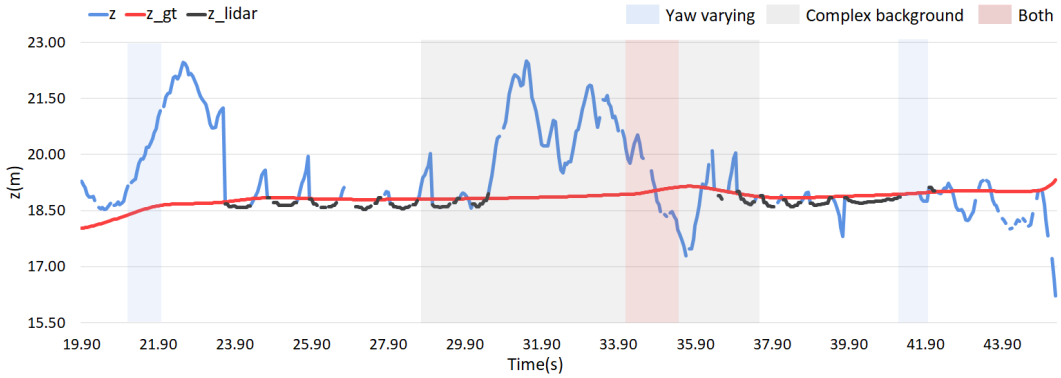
Also, the LiDAR measurements are represented in different colour. This is helpful because the transitions between LiDAR measurements and the points estimated by the algorithm are directly related to the moments where the value of the apparent physical size of the UAV, r, is updated. Attitude variations after the last update of r and the quantization problem associated with the minimum unit of measurement of the bounding boxes (1 pixel) are the two main sources of the errors associated with the position estimation method.

For $Z \approx 10m$ (see Figure 4.8(a)) the value of r is updated for the first time at $t = 2.65s$ while the UAV is changing its yaw from 0° to 90° . Therefore, the value of r for $t \geq 3.5s$ corresponds to an attitude different from the one the UAV presents. Since the height of the bounding boxes returned by YOLO is smaller in the case of yaw equal to 90° (see Figure 4.6), the algorithm predicts that the UAV is further than the ground truth value. However, because the UAV is at a close distance, 1-pixel variations in the height of the bounding box correspond to small variations in the distance estimation. As a consequence, the algorithm is capable of predicting the position of the UAV with an absolute error inferior to $1.93m$. From ts to $t = 11s$ the UAV enters a region with a strong background and starts to vary its attitude from 90° to 0° , changing to the attitude that it had when the last value of r was computed, at $t = 2.5s$. The algorithm predicts values closer to the ground truth and the observed errors are mainly due to the variations in the height of the bounding box. The value of r is updated again at $t = 11.5s$ and the UAV changes its attitude from 0° to 90° soon after. As a consequence, the system estimates a further distance again. At $t = 13.5s$ the value of r is updated and the estimations present values close to the ground truth.

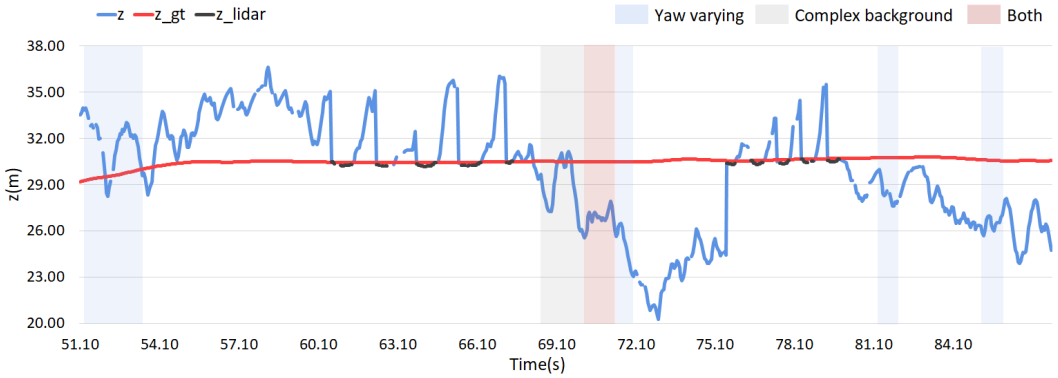
Regarding positions with $Z \approx 20m$ (see Figure 4.8(b)) the UAV starts with an attitude equal to 0° and changes it to 90° around $t = 21.90s$. It is possible to notice that the estimation error increases with predictions further than the ground truth values, similarly to the case of $Z \approx 10m$. The UAV is captured by the LiDAR from $t = 24s$ to $t = 30s$ but estimations keep presenting larger values than the ground truth due to height variations in the bounding boxes returned by YOLO. It is also possible to notice that the estimations worsen after $t = 31s$ although the value of r has just been updated. This happens due to fact that the UAV is passing through an area with a strong background that worsens the detections and correspondent bounding boxes. At $t = 34.90s$, the UAV changes its attitude back to 0° and because the value of r was estimated for 90° the estimation gives a closer distance than the ground truth. Note that since the last value of r was updated in an area with a complex background, where YOLO can output bounding boxes that do not fit the UAV as well, some unpredictable behaviour could have happened in



(a) Results for $Z \approx 10m$.



(b) Results for $Z \approx 20m$.



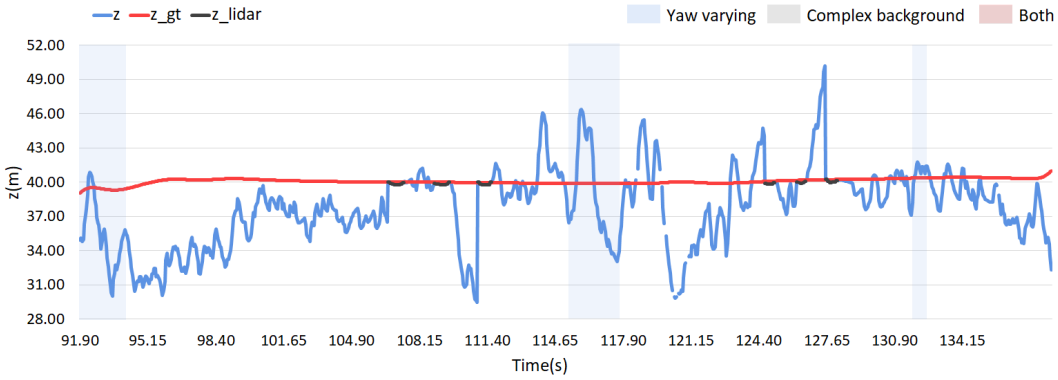
(c) Results for $Z \approx 30m$.

Figure 4.8: Estimation results for Z in the yaw simulation.

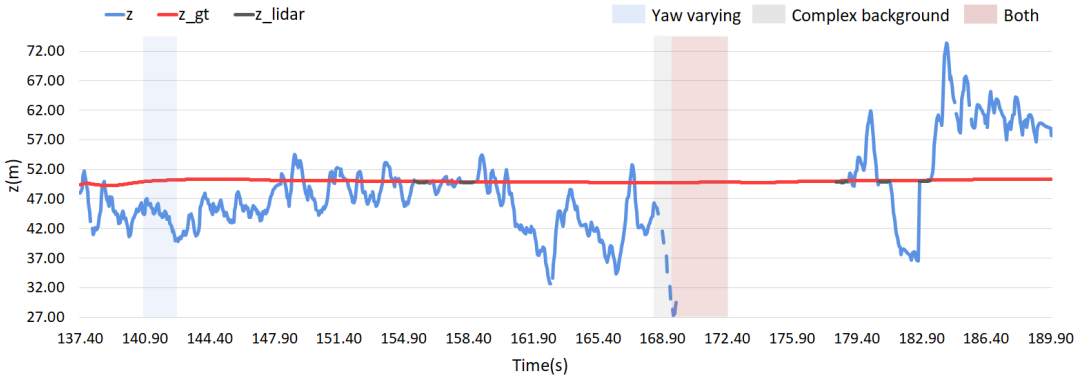
this area. However, YOLO was able to detect the UAV with bounding boxes coherent enough to perform the expected behaviour regardless of the complex background. The UAV changes its attitude once again at $t = 41.90s$ but the UAV is captured by the LiDAR soon afterwards so the following errors are small.

For $Z \approx 30m$ (see Fig. 4.8(c)) it will be possible to conclude that attitude variations still influence the estimations, but variations in the height of the bounding boxes become more significant than before. The UAV changes its attitude from $0°$ to $90°$ at $t = 54s$ and the behaviour of the algorithm is to estimate distances further than the ground truth, as seen in the previous graphics. Comparing this graphic with the ones presented for $Z \approx 10m$ and $Z \approx 20m$, it is possible to see the same behaviour throughout the flight. The estimations worsen when the UAV is in a complex background, with YOLO detecting the UAV

at a closer distance than for the previous values of Z at $t = 69s$. Although the graphics present the same shape, it is clear that the errors for $Z \approx 30m$ are larger than the ones at $Z \approx 10m$. The distance error associated with a 1-pixel variation in the height of the bounding box increases and therefore, although the attitude behaviour is the same, the results worsen.



(a) Results for $Z \approx 40m$.



(b) Results for $Z \approx 50m$.

Figure 4.9: Remaining estimation results for Z in the yaw simulation.

Regarding the position with $Z \approx 40m$ and $Z \approx 50m$ in the Z-axis (see Fig. 4.9) it is possible to notice that the curves do not present the same shape as the curves for $Z \leq 30m$ and that result from the attitude variations throughout the flight. In fact, for $Z \geq 40m$, the UAV appears very small and variations in the attitude do not influence the errors significantly. The verified errors are mainly due to the variations in the height of the bounding boxes. As seen in Section 3.5, for $Z \approx 40n$ a variation of 1 pixel in the height of the bounding box corresponds to variations of $6.8m$ when using a camera with a resolution of $720p$. Therefore, the estimation errors are expected to be large. The estimations presented for $40m$ are smaller than $12.91m$ and it is concluded that the system performs well up until this point. For $Z \approx 50m$, the maximum error reaches $23.21m$ which is a significantly large error. Also, for $Z \approx 50m$ the UAV is so small that YOLO cannot detect it when the background is complex and at $t = 167s$ the UAV is lost, being captured again by the LiDAR at $t = 179s$ when flying against a clear sky background.

A simulation in the same conditions as this flight but with constant yaw was performed to study further the impact of attitude variations in the proposed method. However, because the roll and pitch still varied throughout the flight, although the results for Z between 10 and $30m$ were better, the improvements were

of the order of centimetres and it was not possible to infer significant conclusions.

Figures with the simulation results throughout the entire flight and not just for specific values of Z as well as the error over time for X, Y, Z and d are presented in Appendix A.

YOLO Performance

The precision and recall of YOLO were equal to 100% and 93.50%, respectively. This means that YOLO never identified a false positive in the created ROIs and that YOLO failed to detect the UAV in 242 ROIs of a total of 3721. This can cause problems if YOLO does not detect the target for more than LOSS_THRESHOLD frames. In this case, the UAV would be lost by the Deep SORT tracker and deleted from the list of drones. Therefore, the UAV could only be acquired in the future by the LiDAR. However, YOLO never missed the detection of the UAV for more than LOSS_THRESHOLD consecutive frames for positions with $Z \leq 40m$ so the Deep SORT tracker was able to continue to predict the positions of the UAV in the upcoming frames. The UAV was lost at $Z \approx 50m$ for $10s$ which corresponds to 3.3% of the flight.

The results for the number of empty frames are presented in Table 4.2. It was defined as the fraction between the number of frames without a position estimation output and the number of total frames. A frame without output is considered to be a frame for which no estimations were performed, neither with LiDAR measurements nor with the camera. This happened because YOLO was not able to detect the UAV in those frames.

Table 4.2: Empty frames as a function of Z for the yaw simulation.

	$10m$	$20m$	$30m$	$40m$	$50m$
Empty frames[%]	0.33	8.84	9.93	1.29	17.70

The number of empty frames increases as the position of the UAV along with the Z-axis increases, except for the case of $Z \approx 40m$, since the UAV becomes smaller and harder to detect, especially when flying against complex backgrounds. YOLO presented a speed of $1.73Hz$ when running with Central Processing Unit (CPU) and a speed of $9.5Hz$ when running with Graphics Processing Unit (GPU). Real-time capabilities are still to be explored in the future.

LiDAR Measurements

The results of the LiDAR measurements are presented in Table 4.3 where it is shown the RMSE values, MaxAE and percentage of the output points correspondent to LiDAR measurements. The LiDAR measurements are very close to the ground truth values, as expected because LiDARs have high accuracy. It is possible to notice that the number of LiDAR points decreases as the distance increases. This is expected since the LiDAR beams are spaced by 3° in elevation, which corresponds to a larger height as the distance in Z increases. In fact, as the position in Z increases, the number of LiDAR beams that

capture the UAV decreases. For example, at a position $Z \approx 50m$, the UAV is captured only by one of the beams of the LiDAR at a time.

Table 4.3: RMSE and MaxAE results for the LiDAR measurements of the yaw simulation.

	RMSE[m]				MaxAE[m]				LiDAR Points[%]
	X	Y	Z	d	X	Y	Z	d	
$10m$	0.29	0.13	0.18	0.27	0.56	0.29	0.28	0.43	42.33
$20m$	0.29	0.25	0.16	0.22	0.56	0.60	0.26	0.40	36.83
$30m$	0.34	0.28	0.17	0.19	0.64	0.58	0.26	0.41	14.91
$40m$	0.37	0.26	0.18	0.20	0.64	0.60	0.25	0.35	8.37
$40m$	0.37	0.25	0.18	0.20	0.58	0.55	0.26	0.36	7.43

Taking into account that the results obtained in Table 4.1 are significantly worse than the ones correspondent to LiDAR measurements only (present in Table 4.3) the decision of using the LiDAR points directly whenever they are available is corroborated.

4.3 Camera Resolution Simulation

The results from section 4.2 are influenced mainly by two phenomena: attitude variations during flight and variations in the height of the bounding boxes returned by YOLO. To study the impact of the camera resolution, a flight simulation with a $1080p$ camera was performed. The objective is to verify if, by increasing the number of pixels in the image, it is possible to achieve better results, especially for further positions, where pixel variations correspond to larger distance errors.

The flight simulated presented the same trajectory of the flight from section 4.2 so that a comparison between both flights could be performed and conclusions inferred. The trajectory was presented in Figure 4.5 and only the size of the camera image was changed to 1920×1080 pixels. All the other parameters regarding simulated hardware, velocity, data rate, thresholds and Kalman Filtering were kept. The intrinsic and extrinsic calibration had to be performed again in such a way that it would not improve or worsen the results.

In Figure 4.10, a comparison between ROIs cropped for a camera with resolution $1080p$ versus a camera with resolution $1080p$ is presented. It is possible to see that since the number of pixels of the ROI remained the same (416×416) but the number of pixels in the images increased, the UAVs appear closer and more clearly.

4.3.1 Results

The results obtained for this simulation are presented in Table 4.4 and Figures 4.11 to 4.13. When comparing the results from Table 4.4 with the ones from Table 4.1 is possible to see that, overall, the results for $1080p$ have a higher impact on the estimations of Z and d. For all the cases except $Z \approx 40m$, the estimations of Z and d present a smaller RMSE and MaxAE. In fact, for $Z \approx 50$ the RMSE

(a) UAV appearance at $40m$ with a camera of resolution $720p$.

(b) UAV appearance at $40m$ with a camera of resolution $1080p$.

(c) UAV appearance at $50m$ with a camera of resolution $720p$.

(d) UAV appearance at $50m$ with a camera of resolution $1080p$.

Figure 4.10: UAV at Z approximately equal to 40 and $50m$ with a $720p$ and a $1080p$ camera in a ROI of size 416×416.

associated with d decreased from $8.60m$ to $7.80m$ and the MaxAE decreased from $28.76m$ to $18.36m$, which corresponds to a decrease of 9.3% and 36.16%, respectively.

However, for $Z \approx 40m$ the RMSE of the estimated distance increases from $4.62m$ to $6.34m$ and the MaxAE increases from $12.91m$ to $14.76m$ which corresponds to a growth of 37.23% and 14.33%, respectively. These results do not follow the trend presented by the other values of Z. Comparing Figure 4.9(a) with 4.13(a), it is possible to notice that the curves do not present a similar shape, especially in the beginning. At $t = 98s$ the algorithm estimates a value of Z further than the ground truth for $1080p$ and for $720p$ it estimated a value of Z that was closer. This difference is related to the height of the bounding box predicted by YOLO at the time of the last LiDAR measurement. When comparing the curves obtained for Z approximately 10, 20, 30 and $50m$ for both $1080p$ and $720p$, the shapes are identical. This indicates that the execution of the algorithm was more similar for these values of Z. For this reason, when comparing the results from both simulations, conclusions regarding values of Z different from $40m$ will be taken more into account. Consequently, it is concluded that the $1080p$ camera provides overall better results than the $720p$.

When comparing Figure 4.9(b) and 4.13(b) it is possible to notice that the UAV is no longer lost

Table 4.4: RMSE and MaxAE results obtained for the $1080p$ simulation.

	RMSE[m]				MaxAE[m]			
	X	Y	Z	d	X	Y	Z	d
$10m$	0.44	0.18	0.65	0.71	0.96	0.42	1.73	1.81
$20m$	0.83	0.48	0.97	1.23	2.05	1.65	2.83	3.50
$30m$	1.40	0.84	1.94	2.36	4.06	2.68	7.28	8.49
$40m$	2.60	1.04	5.86	6.34	6.80	3.91	14.05	14.76
$50m$	3.56	1.00	7.01	7.80	12.55	4.07	16.34	18.36

at $t = 167s$, when using a camera with a $1080p$ resolution. The UAV is tracked throughout the entire simulation without ever being lost for more than LOSS_THRESHOLD frames. Since the camera has a better resolution, the UAVs appear more clearly for further distances and YOLO is capable of detecting the UAVs during a larger number of frames.

YOLO Performance

YOLO continued to present precision of 100% and recall improved to 94.42%. This means that YOLO never detected a false positive in the searched ROIs and failed to detect the UAV in 212 of a total of 3800 ROIs. The percentage of empty frames as a function of Z is presented in Table 4.5. When comparing these results with the ones obtained previously, it is possible to notice improvements for Z equal to 10, 20, 30 and $50m$. However, once again, for $Z \approx 40m$ the number of empty frames worsened.

YOLO presented a frequency of $1.64Hz$ when running with CPU and $11.92Hz$ when running with GPU. Since the size of the ROI remained the same (416×416) the processing time of YOLO should not be affected by the fact that images of higher resolution are being analysed. However, the algorithm execution can be delayed since uploading an image with a $1080p$ resolution is computationally more expensive than uploading one with $720p$.

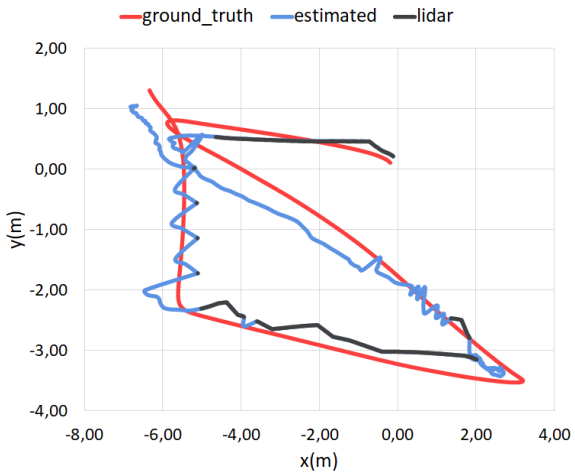
Table 4.5: Empty frames as a function of Z for the $1080p$ simulation.

	$10m$	$20m$	$30m$	$40m$	$50m$
Empty frames[%]	0.00	0.59	8.03	11.39	3.62

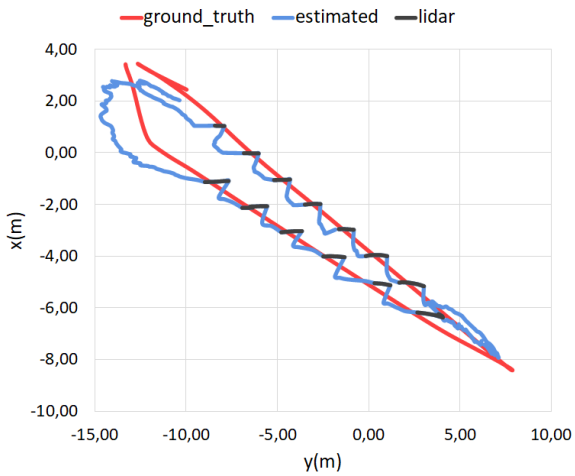
LiDAR Measurements

The LiDAR measurements were given by AirSim and therefore present RMSE and MaxAE values very close to the ground truth values and to the errors obtained for $720p$ (see Table 4.3), since the LiDAR specifications were not modified. It is possible to notice that the number of LiDAR points decreases as the distance increases, as expected.

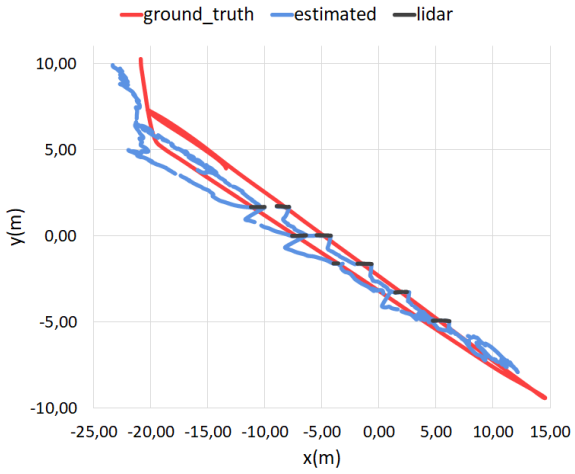
The percentage of LiDAR measurements during the flight is inferior to the $720p$ case, but this is because a new extrinsic calibration had to be performed. The fact that the number of measurements is



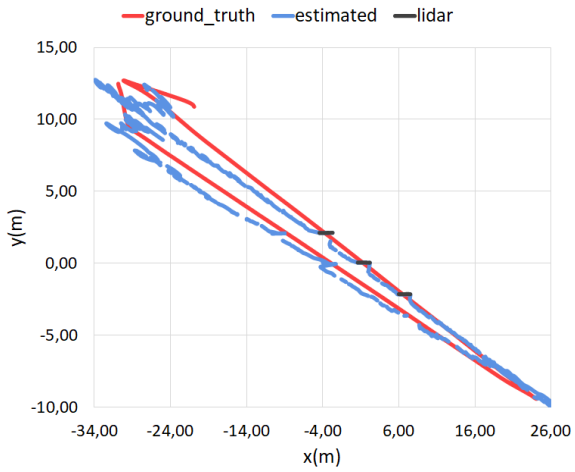
(a) Estimated values at $10m$.



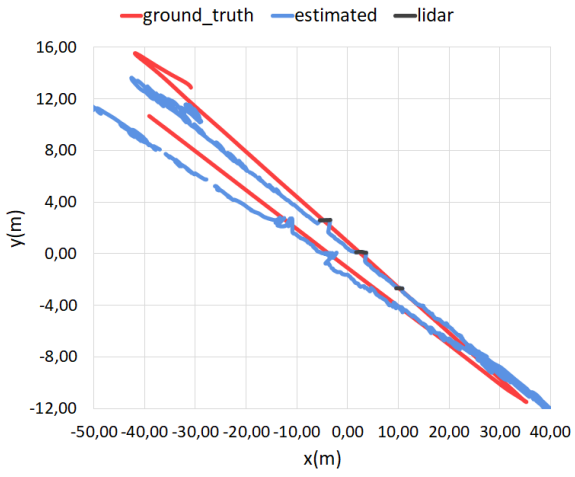
(b) Estimated values at $20m$.



(c) Estimated values at $30m$.



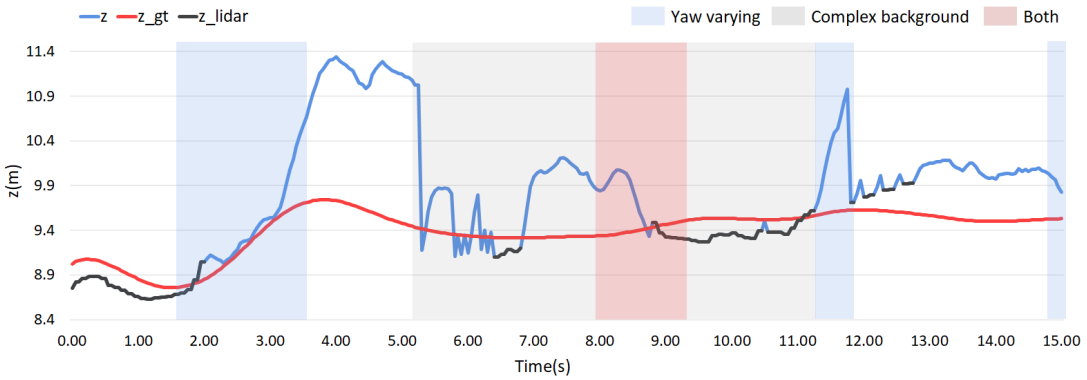
(d) Estimated values at $40m$.



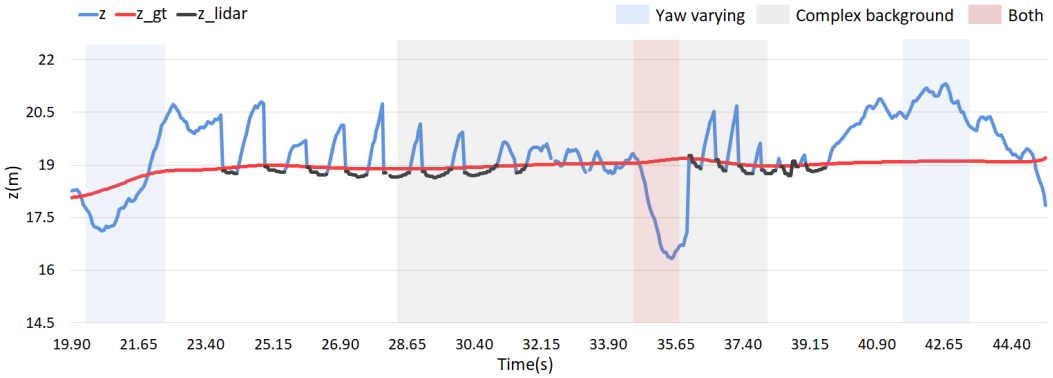
(e) Estimated values at $50m$.

Figure 4.11: Estimations results in the OXY plane for the $1080p$ simulation.

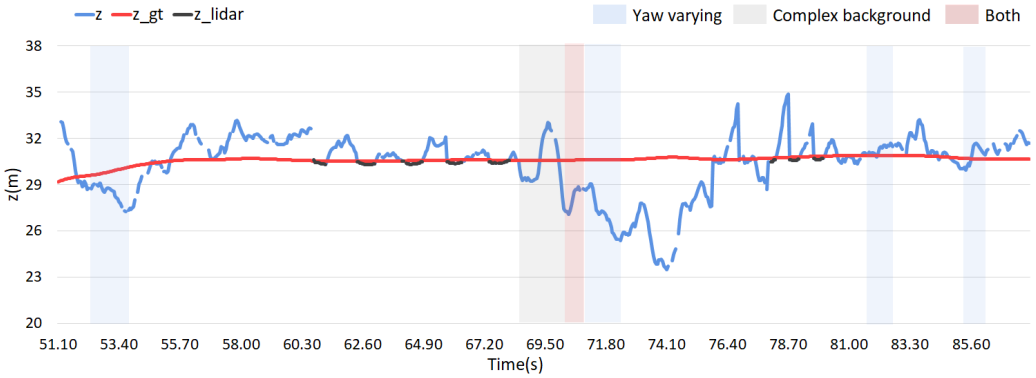
inferior than before could only worsen the results, not improve them, so it does not present a problem to the analysis of the results.



(a) Results for $Z \approx 10m$.



(b) Results for $Z \approx 20m$.

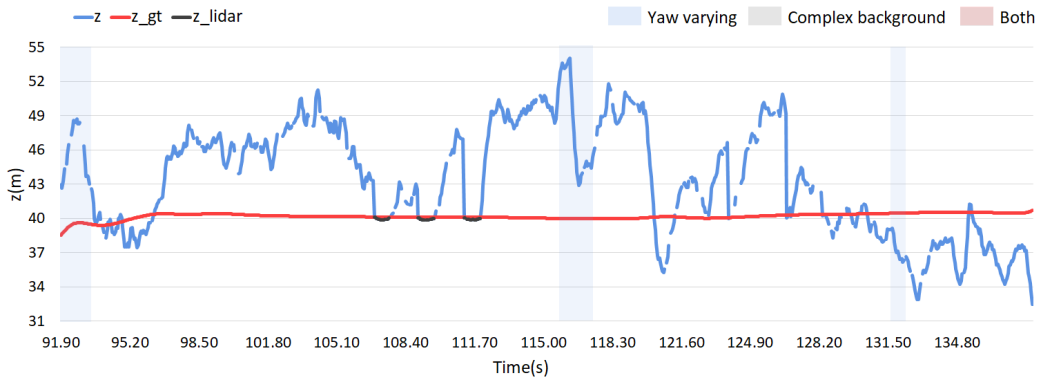


(c) Results for $Z \approx 30m$.

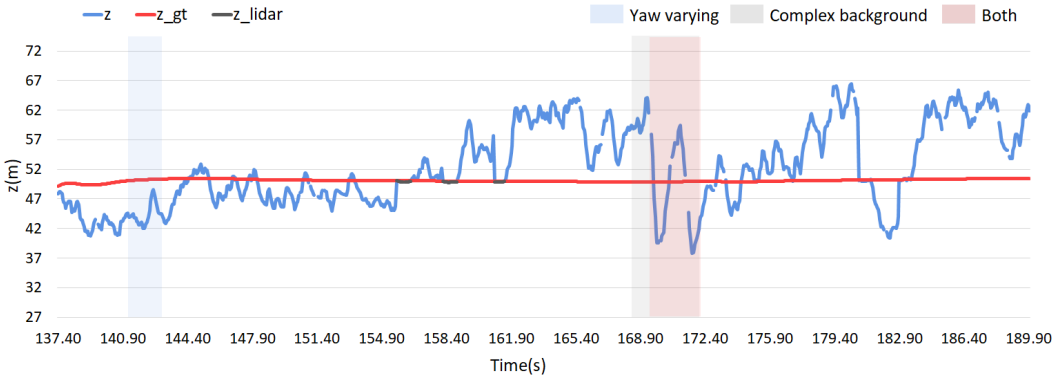
Figure 4.12: Estimation results for Z in the $1080p$ simulation.

Table 4.6: RMSE and MaxAE results for the LiDAR measurements of the $1080p$ simulation.

	RMSE[m]				MaxAE[m]				LiDAR Points[%]
	X	Y	Z	d	X	Y	Z	d	
$10m$	0.36	0.18	0.18	0.26	0.62	0.36	0.34	0.42	39.87
$20m$	0.32	0.25	0.17	0.22	0.81	0.60	0.28	0.40	28.66
$30m$	0.32	0.26	0.17	0.22	0.65	0.58	0.25	0.42	14.05
$40m$	0.25	0.27	0.17	0.19	0.41	0.55	0.25	0.29	5.70
$40m$	0.36	0.28	0.17	0.20	0.68	0.57	0.24	0.31	5.13



(a) Results for $Z \approx 40m$.



(b) Results for $Z \approx 50m$.

Figure 4.13: Remaining results for Z in the $1080p$ simulation.

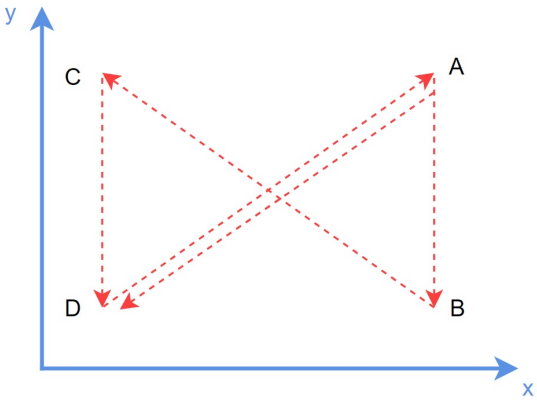
Figures with the simulation results throughout the entire flight and not just for specific values of Z as well as the error over time for X, Y, Z and d are presented in Appendix A.

4.4 Multi-UAV Simulation

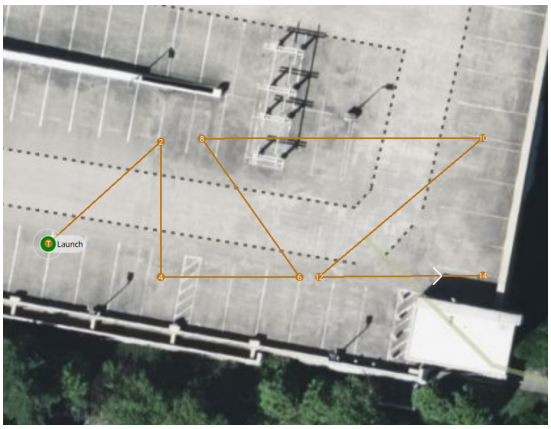
A simulated flight with two UAVs was performed to exemplify the response of the algorithm when more than one UAV is present in the scene. This simulation was carried out with the $720p$ camera since this is the specifications of the camera available at CfAR.

The trajectory followed by UAV 1 is presented in Figure 4.14. The UAV started at the point that presented the furthest value of Z and got closer to the camera as the simulation time increased. On the OXY plane, the UAV moved between points accordingly with the sequence A-B-C-D-A, as presented in Figure 4.14(a). The yaw was configured to be kept constant and equal to $0°$ during the entire flight but since the roll and pitch vary, these variations will be identified in the results to provide a better understanding of them.

The trajectory followed by UAV 2 is presented in Figure 4.15. This UAV also started at the point that presented the furthest value of Z and got closer to the camera as the simulation time increased. On the OXY plane, the UAV moved between points accordingly with the sequence A-B-C-B-A presented in Figure 4.15(a). The yaw was defined in the same way as explained for UAV 1.



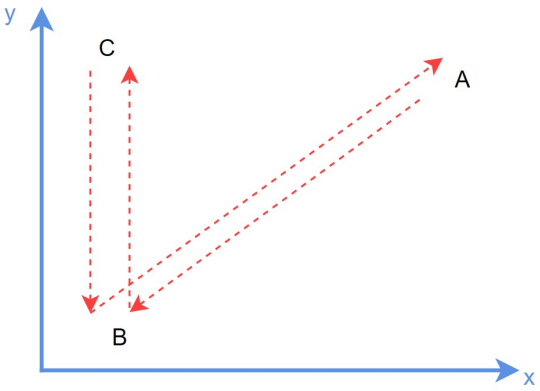
(a) Trajectory followed in the OXY plane.



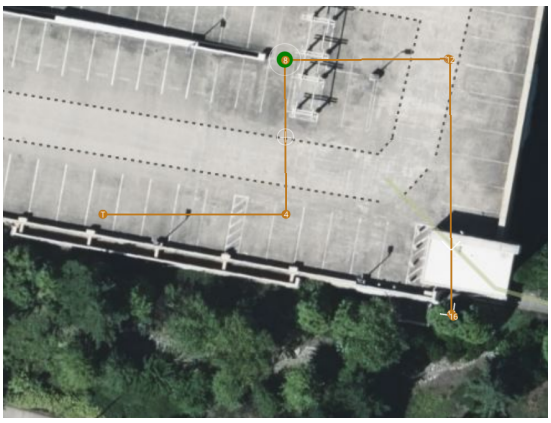
(b) Trajectory followed in the OXZ plane.

Figure 4.14: Trajectory followed by UAV 1 for the multi-UAV simulation.

The minimum distance between the two UAVs was 50 pixels, which is smaller than the DETECTION_THRESHOLD defined previously for the simulation (64 pixels). The correspondent frame is presented in Figure 4.16.



(a) Trajectory followed in the OXY plane.



(b) Trajectory followed in the OXZ plane.

Figure 4.15: Trajectory followed by UAV 2 for the multi-UAV simulation.

Figure 4.16: Minimum distance between UAVs during the multi-UAV simulation.

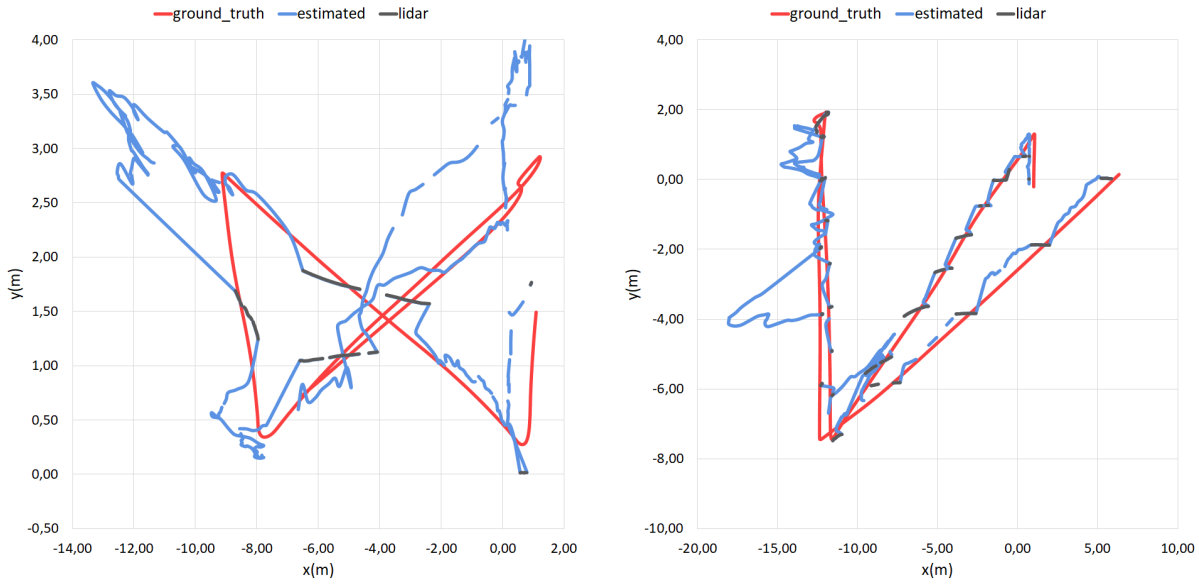
4.4.1 Results

The results of the multi-UAV flight are presented in Table 4.7 and Figures 4.17 and 4.18. The assignment of the UAVs in the frames in which YOLO was able to detect them was always performed correctly, even when the UAVs were flying at a close distance of 50 pixels.

When analysing Table 4.7 it is possible to notice that the RMSE and MaxAE are concordant with the values obtained in the previous simulations. It is also presented the percentage of empty frames and the precision and recall of YOLO for each UAV. It is possible to notice that YOLO continues to present precision of 100%. The precision results could worsen in the case of pictures of non-simulated flight tests, with the presence of birds and other real-world objects that could be mistaken as flying UAVs. The recall of YOLO for UAV 1 is equal to 92.00% and 92.18% which is smaller than the values obtained before, but still high enough to be considered a good result.

Table 4.7: Results for the multi-UAV simulation.

	RMSE[m]				MaxAE[m]				Empty Frames[%]	YOLO[%]	
	X	Y	Z	d	X	Y	Z	d		P	R
UAV 1	1.19	0.40	3.85	4.02	4.24	1.11	12.81	13.51	9.07	100	92.00
UAV 2	1.16	0.58	3.12	3.36	5.63	2.62	13.59	14.60	9.47	100	92.18

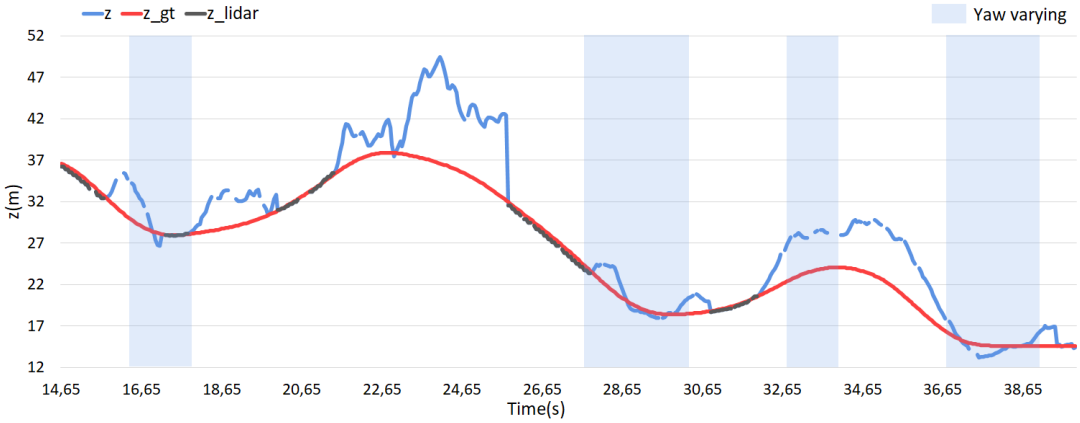


(a) Results for UAV 1.

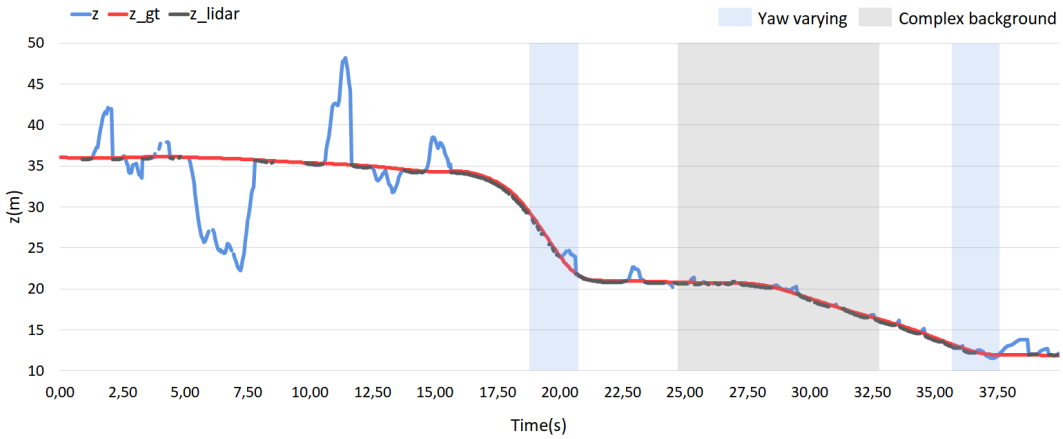
(b) Results for UAV 2.

Figure 4.17: Results in the OXY plane for the multi-UAV simulation.

Analysing Figure 4.18(b) it is possible to identify that at instant $t = 8.75s$ YOLO was not capable of detecting the UAV for more than LOSS_THRESHOLD since the following frames are empty. The UAV is captured by the LiDAR again at $t = 9.85s$. It is also possible to notice that, during this flight, the LiDAR was able to acquire the UAVs in a larger number of frames, because these UAVs presented trajectories that intersected more times the LiDAR beams. As a consequence, the position outputted by the system is very accurate even for positions in which the UAV is flying in an area with complex background, such



(a) Results for UAV 1.



(b) Results for UAV 2.

Figure 4.18: Results in the OXZ plane for the multi-UAV simulation.

as from $t = 25s$ to $t = 33s$.

Since the UAV kept its yaw constant throughout the flight, the errors presented in the curves of Figure 4.18 result from variations in pitch and roll and the variations in the height of the bounding boxes outputted by YOLO.

The results from this flight test show that the algorithm makes no assumption regarding the number of UAVs present in the scene and is capable of solving the assignment problem between UAVs from different frames, even when the encountered UAVs are flying at close distances.

4.5 Discussion

In this chapter, three simulations were presented. The first one permitted to study the influence of attitude variations and pixel quantization in the estimations performed by the algorithm. The second simulation studied the influence of the camera resolution by simulating the same flight with a camera of $1080p$ instead of a camera of $720p$. Finally, a multi-UAV simulation was used to study the capabilities of the algorithm when more than one UAV is present in the scene.

Overall, it is shown that the attitude influences the estimations for distances along the Z-axis up

to $30m$. For further values of Z, the UAV appearance is too small and the results are mostly affected by variations in the height of the bounding box. There is a quantization associated with the smallest measurement unit - 1 pixel - and the error in distance associated with a variation of 1 pixel increases as the distance increases. Therefore, for further distances, the errors associated with height variations also increase.

A maximum RMSE value of $4.62m$ is obtained for the distance, d, when the UAV is at $Z \approx 40m$. For $Z \approx 50m$ the RMSE increased to $8.60m$. Both these results are promising, however, even though the MaxAE for $Z \approx 40m$ is equal to $12.91m$, for $Z \approx 50m$ it is equal to $28.76m$ which is a significantly high error. For this reason, it is concluded that the system performs well for positions with $Z \leq 40m$.

When comparing the results obtained with a camera of resolution equal to $1080p$, it is concluded that the results improve for all the values of Z except for $Z \approx 40m$. This happened because the execution of the algorithm for $40m$ was considerably different. Since the execution for the other values of Z was closer in both cases, it was concluded that the system's results improved when using a camera with higher resolution and the system can be used to estimate the position of UAVs that are at positions with $Z \leq 50m$.

The system was capable of detecting and tracking multiple UAVs in the frames of the multi-UAV simulation without ever failing to assign the two UAVs correctly, even when they were at distances inferior to the pre-defined DETECTION_THRESHOLD.

In conclusion, the system behaved as expected. Further improvements that can be implemented in the system will be discussed in the next chapter.

Chapter 5

Conclusions and Future Work

This chapter presents the main conclusions of this thesis work, its contributions and possible future work approaches that might improve the system and its capabilities.

5.1 Conclusions

The presented work proposes a system for the detection and relative position estimation of non-cooperative UAVs using a deep learning approach and data from a LiDAR and a Camera.

The system uses the state-of-the-art object detector YOLO to detect nearby UAVs in images taken with an EO camera. LiDAR measurements are used to acquire the UAV and create ROIs of size 416×416 pixels in which YOLO searches for the flying objects. Distance and position estimations are computed by processing the LiDAR measurements or by using the bounding boxes returned by YOLO and previous LiDAR data. The system is capable of detecting and estimating the relative position of any number of encountered UAVs without any a priori knowledge. Furthermore, the system can be expanded to any type of class, including pedestrians and automotive vehicles.

Simulations were performed to study the influence of attitude variations and pixel quantization in the estimations performed by the algorithm, the influence of the camera resolution and the system's capabilities to detect and track multiple UAVs simultaneously.

Overall, the system provided good estimation results for values of Z up to $40m$ whenever using a camera with a resolution of $720p$ and up to $50m$ when using a camera with $1080p$. The maximum RMSE value obtained for $Z \approx 40m$ and a camera with $720p$ was equal to $4.62m$, with correspondent MaxAE equal to $12.91m$. For the case of the camera with $1080p$, the maximum RMSE obtained for $Z \approx 50m$ was $7.80m$ with correspondent MaxAE $18.36m$.

For the multi-UAV simulation, the system was capable of detecting and tracking multiple UAVs in the same frame without ever failing to assign the two UAVs correctly.

5.2 Contributions

The method presented can provide not only detections but also relative positions of encountered UAVs using a LiDAR and a camera. The system can be used in sense and avoid applications, relative stabilization of several UAVs in a formation or swarm-like behaviour and the detection and tracking of intrusive UAVs.

In addition, a labelled dataset of synthesized images with 4761 training and 530 validation images is developed in the YOLO Darknet format, for the training of the object detector YOLO.

A demonstration of the procedure necessary to use the open-source software AirSim to create a dataset to train an object detector is included. It is also presented the procedure necessary to find AirSim's camera intrinsic parameters and the extrinsic calibration parameters between the LiDAR and camera sensors.

5.3 Future Work

Several improvements can be implemented in the system. Regarding the detection task, YOLO can be trained for a larger number of UAVs and for different classes, such as birds and other objects that are similar to small UAVs. Another interesting addition would be to train YOLO with images of size 416×416 pixels since this is the input size of the ROIs the algorithm creates. This could provide better detection results, increasing the recall of the object detector.

Regarding relative position estimation, some possibilities could be explored for the update of the apparent physical size, r. One of them would be to consider the implementation of appearance descriptors to detect attitude variations. The system could detect instants with strong attitude variations and move to an extra layer of decision.

The assignment problem could be improved by complementing the distance metric used with others, such as the cosine distance, for example. In addition, whenever two or more UAVs are very close, a second layer of decision could be implemented, to guarantee a more reliable assignment between the UAVs. It would be interesting to investigate if Deep SORT could also be used in the assignment problem since it performs its own assignments throughout the execution of the algorithm.

An important task for future work is to estimate the velocities of the encountered UAVs since these are necessary for the integration with collision-avoidance trajectory planning algorithms. Once this is achieved, the algorithm could use the estimated velocity of the UAV and the frame rate of the camera to automatically update the values of the thresholds defined throughout this thesis. In this case, each flying UAV would have its own LOSS_THRESHOLD and DETECTION_THRESHOLD.

Finally, algorithm optimization and evaluation on an onboard computer would be needed to study potential real-time capabilities.

Bibliography

[1] J. Qi, D. Song, H. Shang, N. Wang, C. Hua, C. Wu, X. Qi, and J. Han. Search and Rescue Rotary-Wing UAV and Its Application to the Lushan Ms 7.0 Earthquake. *Journal of Field Robotics*, 33(3): 290–321, 2016. ISSN 15564959. doi: 10.1002/rob.21615.

[2] W. H. Maes and K. Steppe. Perspectives for Remote Sensing with Unmanned Aerial Vehicles in Precision Agriculture. *Trends in Plant Science*, 24(2):152–164, 2019. ISSN 13601385. doi: 10.1016/j.tplants.2018.11.007.

[3] J. A. Gonçalves and R. Henriques. UAV photogrammetry for topographic monitoring of coastal areas. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:101–111, 2015. ISSN 09242716. doi: 10.1016/j.isprsjprs.2015.02.009.

[4] Y. Ham, K. K. Han, J. J. Lin, and M. Golparvar-Fard. Visual monitoring of civil infrastructure systems via camera-equipped Unmanned Aerial Vehicles (UAVs): a review of related works. *Visualization in Engineering*, 4(1):1–8, 2016. ISSN 22137459. doi: 10.1186/s40327-015-0029-z.

[5] A. Simpson and J. Stoker. Safety challenges in flying UAVS (unmanned aerial vehicles) in non segregated airspace. pages 81–88, 2006. doi: 10.1049/cp:20060206.

[6] M. Correa, J. B. Camargo, M. A. Rossi, and J. R. Almeida. Improving the resilience of UAV in non-segregated airspace using multiagent paradigm. *Proceedings - 2012 2nd Brazilian Conference on Critical Embedded Systems, CBSEC 2012*, pages 88–93, 2012. doi: 10.1109/CBSEC.2012.21.

[7] H.-w. Warnke. Reconnaissance of LSS-UAS with Focus on EO-Sensors. *Military Sensing Symposium*, 2017.

[8] Mercedes-Benz Innovation: Autonomous. URL `https://www.mercedes-benz.com/en/innovation/autonomous/`.

[9] Autopilot — Tesla. URL `https://www.tesla.com/autopilot`.

[10] A. Rozantsev, V. Lepetit, and P. Fua. Detecting Flying Objects Using a Single Moving Camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(5):879–892, 2017. ISSN 0162-8828. doi: 10.1109/TPAMI.2016.2564408.

[11] M. Skowron, W. Chmielowiec, K. Glowacka, M. Krupa, and A. Srebro. Sense and avoid for small unmanned aircraft systems: Research on methods and best practices. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 233(16):6044–6062, 2019. ISSN 20413025. doi: 10.1177/0954410019867802.

[12] X. Prats, L. Delgado, J. Ramírez, P. Royo, and E. Pastor. Requirements, issues, and challenges for sense and avoid in unmanned aircraft systems. *Journal of Aircraft*, 49(3):677–687, 2012. ISSN 15333868. doi: 10.2514/1.C031606.

[13] P. Andraši, T. Radišić, M. Muštra, and J. Ivošević. Night-time Detection of UAVs using Thermal Infrared Camera. *Transportation Research Procedia*, 28:183–190, 2017. ISSN 23521465. doi: 10.1016/j.trpro.2017.12.184.

[14] P. Zhao, C. X. Lu, B. Wang, N. Trigoni, and A. Markham. 3-D motion capture of an unmodified drone with single-chip millimeter wave radar. *arXiv*, 2020. ISSN 23318422.

[15] T. Multerer, A. Ganis, U. Prechtel, E. Miralles, A. Meusling, J. Mietzner, M. Vossiek, M. Loghi, and V. Ziegler. Low-cost jamming system against small drones using a 3D MIMO radar based tracking. *European Microwave Week 2017: "A Prime Year for a Prime Event", EuMW 2017 - Conference Proceedings; 14th European Microwave Conference, EURAD 2017*, 2018-Janua:299–302, 2017. doi: 10.23919/EURAD.2017.8249206.

[16] L. R. Sahawneh, J. Mackie, J. Spencer, R. W. Beard, and K. F. Warnick. Airborne radar-based collision detection and risk estimation for small unmanned aircraft systems. *Journal of Aerospace Information Systems*, 12(12):756–766, 2015. ISSN 23273097. doi: 10.2514/1.I010284.

[17] S. Ramasamy, R. Sabatini, and A. Gardi. Avionics sensor fusion for small size unmanned aircraft Sense-and-Avoid. *2014 IEEE International Workshop on Metrology for Aerospace, MetroAeroSpace 2014 - Proceedings*, pages 271–276, 2014. doi: 10.1109/MetroAeroSpace.2014.6865933.

[18] D. Justino. *LiDAR and Camera Sensor Fusion for Onboard sUAS Detection and Tracking*. PhD thesis, Instituto Superior Técnico, 2021.

[19] J. James, J. J. Ford, and T. L. Molloy. Learning to Detect Aircraft for Long-Range Vision-Based Sense-and-Avoid Systems. *IEEE Robotics and Automation Letters*, 3(4):4383–4390, 2018. ISSN 23773766. doi: 10.1109/LRA.2018.2867237.

[20] J. Lai, J. J. Ford, L. Mejias, and P. O'Shea. Characterization of Sky-region Morphological-temporal Airborne Collision Detection. *Journal of Field Robotics*, 30(2):171–193, mar 2013. ISSN 1556-4959. doi: 10.1002/rob.21443.

[21] S. Petridis, C. Geyer, and S. Singh. Learning to detect aircraft at low resolutions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in*

Bioinformatics), 5008 LNCS:474–483, 2008. ISSN 03029743. doi: 10.1007/978-3-540-79547-6_46.

[22] A. Nussberger, H. Grabner, and L. Van Gool. Aerial object tracking from an airborne platform. *2014 International Conference on Unmanned Aircraft Systems, ICUAS 2014 - Conference Proceedings*, pages 1284–1293, 2014. doi: 10.1109/ICUAS.2014.6842386.

[23] M. Vrba, D. Hert, and M. Saska. Onboard Marker-Less Detection and Localization of Non-Cooperating Drones for Their Safe Interception by an Autonomous Aerial System. *IEEE Robotics and Automation Letters*, 4(4):3402–3409, 2019. ISSN 23773766. doi: 10.1109/LRA.2019.2927130.

[24] M. Vrba and M. Saska. Marker-Less Micro Aerial Vehicle Detection and Localization Using Convolutional Neural Networks. *IEEE Robotics and Automation Letters*, 5(2):2459–2466, 2020. ISSN 23773766. doi: 10.1109/LRA.2020.2972819.

[25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016-Decem:779–788, 2016. ISSN 10636919. doi: 10.1109/CVPR.2016.91.

[26] F. Schilling, F. Schiano, and D. Floreano. Vision-Based Drone Flocking in Outdoor Environments. *IEEE Robotics and Automation Letters*, 6(2):2954–2961, apr 2021. ISSN 23773766. doi: 10.1109/LRA.2021.3062298.

[27] Q. Lim, Y. He, and U.-X. Tan. Real-Time Forward Collision Warning System Using Nested Kalman Filter for Monocular Camera. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 868–873. IEEE, dec 2018. ISBN 978-1-7281-0377-8. doi: 10.1109/ROBIO.2018.8665220.

[28] S. Ramasamy, R. Sabatini, A. Gardi, and J. Liu. LIDAR obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology*, 55:344–358, 2016. ISSN 12709638. doi: 10.1016/j.ast.2016.05.020.

[29] P. Wei, L. Cagle, T. Reza, J. Ball, and J. Gafford. LiDAR and camera detection fusion in a real-time industrial multi-sensor collision avoidance system. *Electronics (Switzerland)*, 7(6), 2018. ISSN 20799292. doi: 10.3390/electronics7060084.

[30] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu. Object detection with deep learning: A review. *arXiv*, 30 (11):3212–3232, 2018. ISSN 23318422.

[31] W. Pitts and W. S. McCulloch. How we know universals the perception of auditory and visual forms. *The Bulletin of Mathematical Biophysics*, 9(3):127–147, 1947. ISSN 00074985. doi: 10.1007/BF02478291.

[32] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference*

on Computer Vision and Pattern Recognition, pages 580–587, 2014. ISSN 10636919. doi: 10.1109/CVPR.2014.81.

[33] J. Redmon and A. Farhadi. YOLO9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:6517–6525, 2017. doi: 10.1109/CVPR.2017.690.

[34] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *32nd International Conference on Machine Learning, ICML 2015*, 1:448–456, 2015.

[35] J. Redmon and A. Farhadi. YOLOv3: An Incremental Improvement. 2018.

[36] X. Li, T. Lai, S. Wang, Q. Chen, C. Yang, and R. Chen. Weighted feature pyramid networks for object detection. *Proceedings - 2019 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking, ISPA/BDCloud/SustainCom/SocialCom 2019*, pages 1500–1504, 2019. doi: 10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00217.

[37] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. apr 2020. ISSN 2331-8422.

[38] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, 2012.

[39] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. Microsoft coco: Common objects in context, 2015.

[40] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee. A Survey of Modern Deep Learning based Object Detection Models. pages 1–18, 2021.

[41] G. Ciaparrone, F. Luque Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera. Deep learning in video multi-object tracking: A survey. *Neurocomputing*, 381:61–88, 2020. ISSN 18728286. doi: 10.1016/j.neucom.2019.11.023.

[42] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft. Simple online and realtime tracking. *Proceedings - International Conference on Image Processing, ICIP*, 2016-Augus:3464–3468, 2016. ISSN 15224880. doi: 10.1109/ICIP.2016.7533003.

[43] N. Wojke, A. Bewley, and D. Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, sep 2017. ISBN 978-1-5090-2175-8. doi: 10.1109/ICIP.2017.8296962.

[44] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2 edition, 2003. ISBN 978-0-13-085198-7.

[45] J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 147:1106–1112, 2019. ISSN 03783839. doi: 10.1109/CVPR.1997.609468.

[46] T. Collins and A. Bartoli. Infinitesimal plane-based pose estimation. *International Journal of Computer Vision*, 109(3):252–286, 2014. ISSN 15731405. doi: 10.1007/s11263-014-0725-5.

[47] C. V. Dolph, M. J. Logan, L. J. Glaab, T. L. Vranas, R. G. McSwain, Z. R. Johns, and K. Severance. Sense and avoid for small unmanned aircraft systems. *AIAA Information Systems-AIAA Infotech at Aerospace, 2017*, (10), 2017. doi: 10.2514/6.2017-1151.

[48] C. Mallet and F. Bretar. Full-waveform topographic lidar: State-of-the-art. *ISPRS Journal of Photogrammetry and Remote Sensing*, 64(1):1–16, 2009. ISSN 09242716. doi: 10.1016/j.isprsjprs.2008.09.007.

[49] J. Shan and C. Toth. *Topographic Laser Ranging and Scanning: Principles and Processing, Second Edition*. CRC Press, 2018. ISBN 9781351650427. URL `https://books.google.ca/books?id=N_ErDwAAQBAJ`.

[50] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering, Transactions of the ASME*, 82(1):35–45, 1960. ISSN 1528901X. doi: 10.1115/1.3662552.

[51] G. F. Franklin, M. L. Workman, and D. Powell. *Digital Control of Dynamic Systems*. Addison-Wesley Longman Publishing Co., Inc., USA, 3rd edition, 1997. ISBN 0201820544.

[52] R. Padilla, S. L. Netto, and E. A. Da Silva. A Survey on Performance Metrics for Object-Detection Algorithms. *International Conference on Systems, Signals, and Image Processing*, 2020-July(July): 237–242, 2020. ISSN 21578702. doi: 10.1109/IWSSIP48289.2020.9145130.

[53] A. Botchkarev. Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology. pages 1–37, 2018.

[54] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010. ISSN 0920-5691. doi: 10.1007/s11263-009-0275-4. URL `http://link.springer.com/10.1007/s11263-009-0275-4`.

[55] M. Daszykowski and B. Walczak. Density-Based Clustering Methods. *Comprehensive Chemometrics*, 2:635–654, 2009. doi: 10.1016/B978-044452701-1.00067-3.

[56] S. Shah, D. Dey, C. Lovett, and A. Kapoor. AirSim : High-Fidelity Visual and Physical. pages 1–14.

[57] D. Chuan-En Lin, L. Lai, I. Ibrahimli, and A. Kinsman. Drone-net: 2664 images of drones, labeled. URL `https://github.com/chuanenlin/drone-net`.

[58] B. Karis. Real shading in unreal engine 4 by. 2013.

[59] F. Pezoa, J. L. Reutter, and F. Suarez. Foundations of JSON Schema. pages 263–273, 2016.

[60] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys. PIXHAWK : A System for Autonomous Flight using Onboard Computer Vision. pages 2992–2997, 2011.

[61] QGC - QGroundControl - Drone Control. URL http://qgroundcontrol.com/.

[62] N. Koenig and A. Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3: 2149–2154, 2004. doi: 10.1109/iros.2004.1389727.

[63] L. Meier. MAVLink Developer Guide, 2009. URL https://mavlink.io/en/.

[64] G. Couppey. DJI INSPIRE 2 with ZENMUSE X5S, 2016. URL https://sketchfab.com/3d-models/dji-inspire-2-with-zenmuse-x5s-3979efe28b3a4221bdd462638582d0a6.

[65] DJI Mavic 2 Pro Photogrammetry, 2018. URL https://sketchfab.com/3d-models/dji-mavic-2-pro-photogrammetry-4496df7bf13841698f9a2a150a49cbfd.

[66] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[67] M. A. Mittet, H. Nouira, X. Roynard, F. Goulette, and J. E. Deschaud. Experimental assessment of the quanergy M8 LiDAR sensor. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 41:527–531, 2016. ISSN 16821750. doi: 10.5194/isprsarchives-XLI-B5-527-2016.

[68] MATLAB. *9.6.0.1072779 (R2019a)*. The MathWorks Inc., Natick, Massachusetts, 2010.

[69] Google Colaboratory, 2017. URL https://colab.research.google.com/.

Appendix A

Simulation Results

The results presented in Chapter 4 for the yaw and camera resolution simulations were divided for each value of Z. In this appendix, the results obtained throughout the entire simulations are presented.

A.1 Yaw Simulation

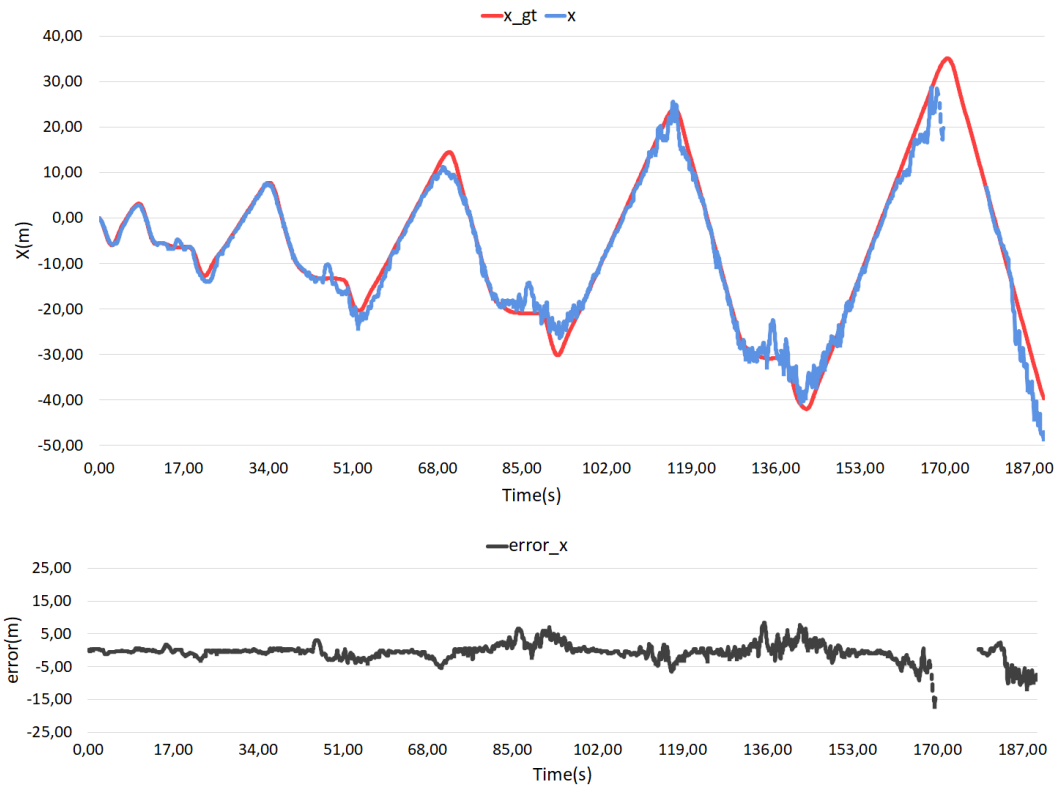


Figure A.1: Yaw simulation results for X over time.

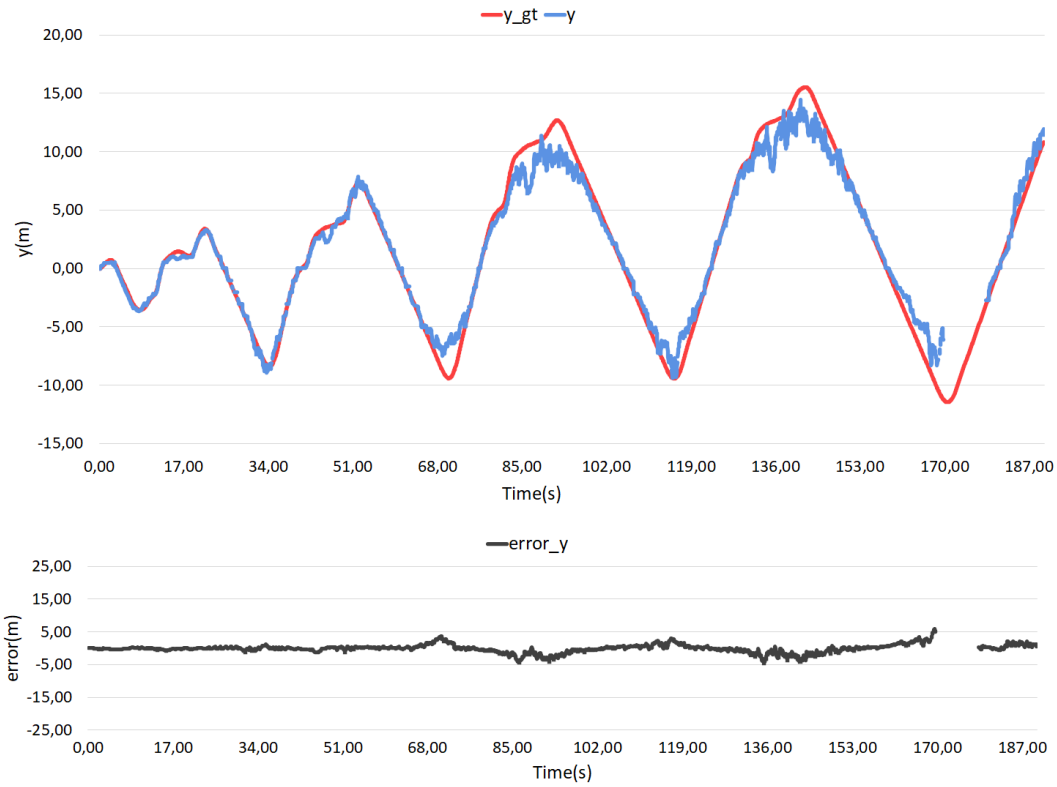


Figure A.2: Yaw simulation results for Y over time.

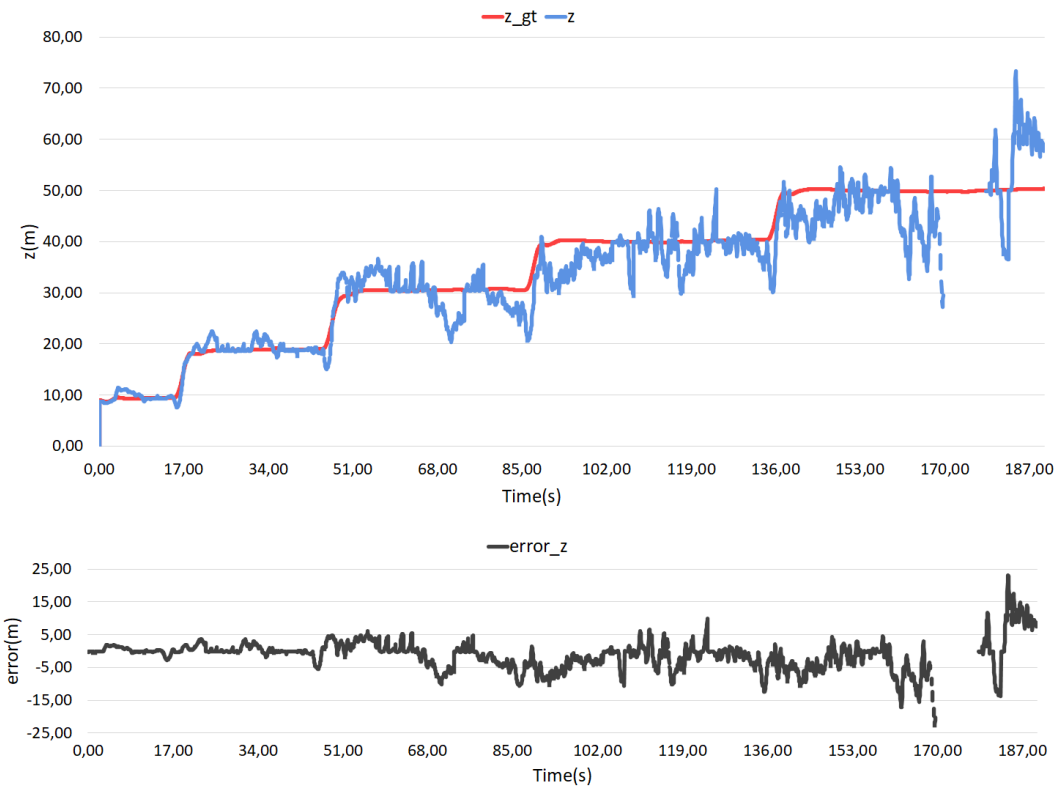


Figure A.3: Yaw simulation results for Z over time.

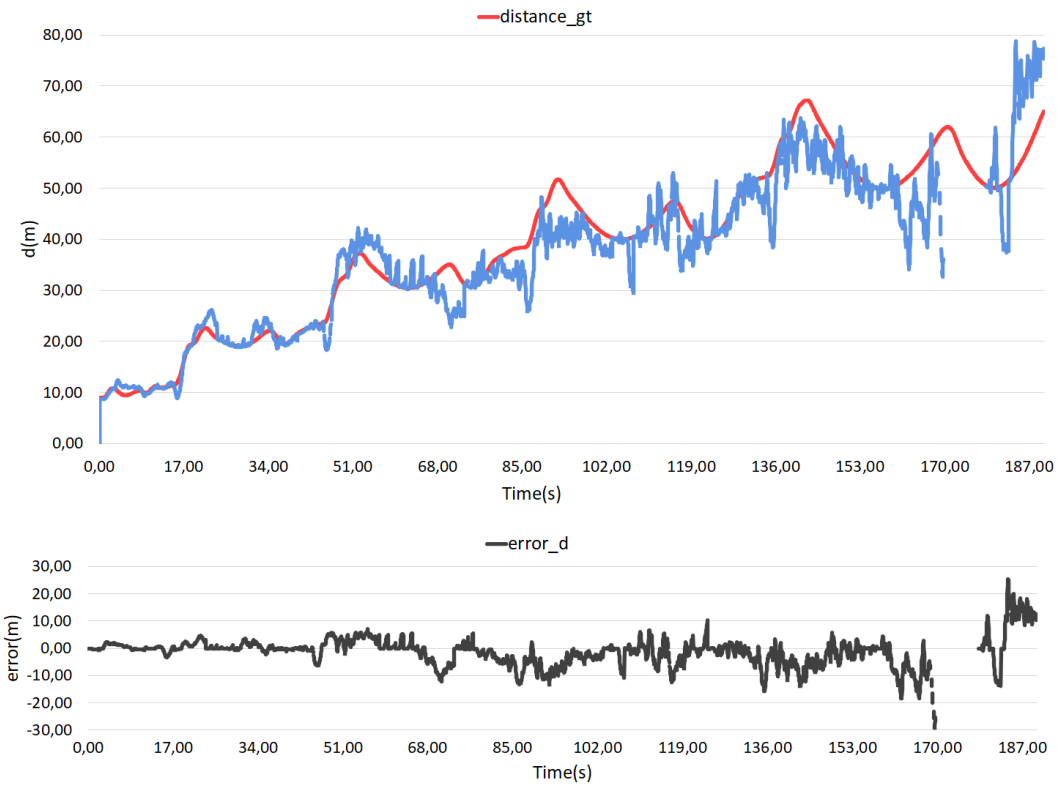


Figure A.4: Yaw simulation results for d over time.

A.2 Camera Resolution Simulation

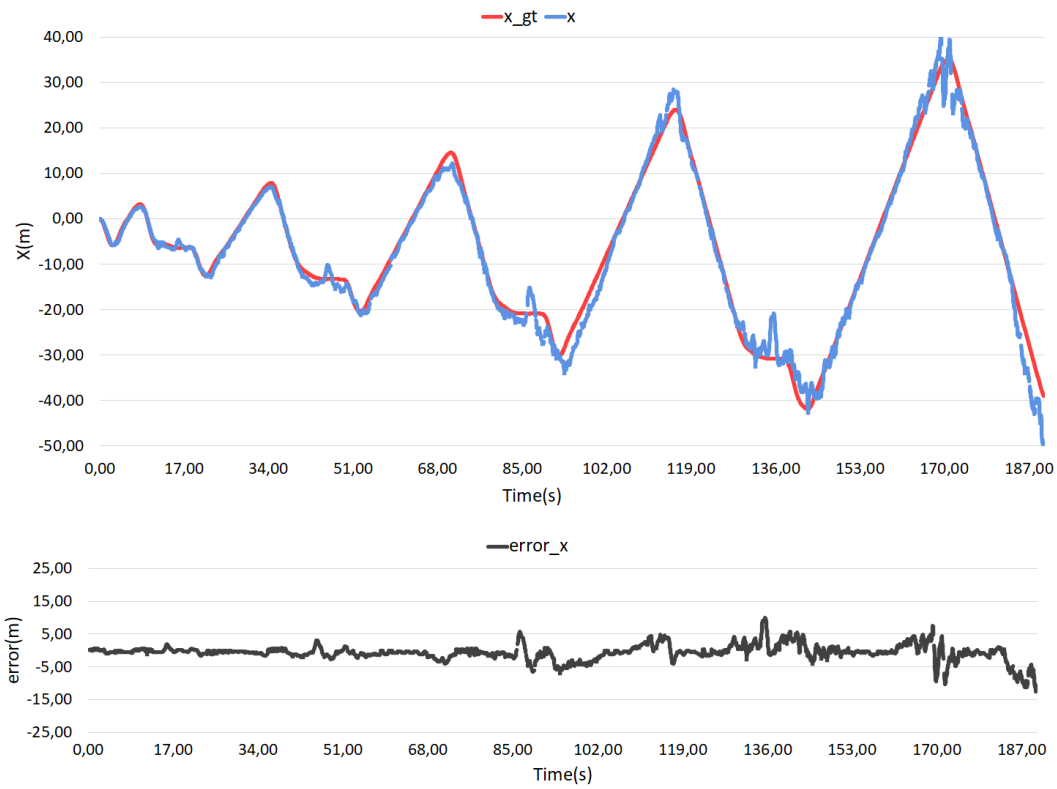


Figure A.5: Camera resolution simulation results for X over time.

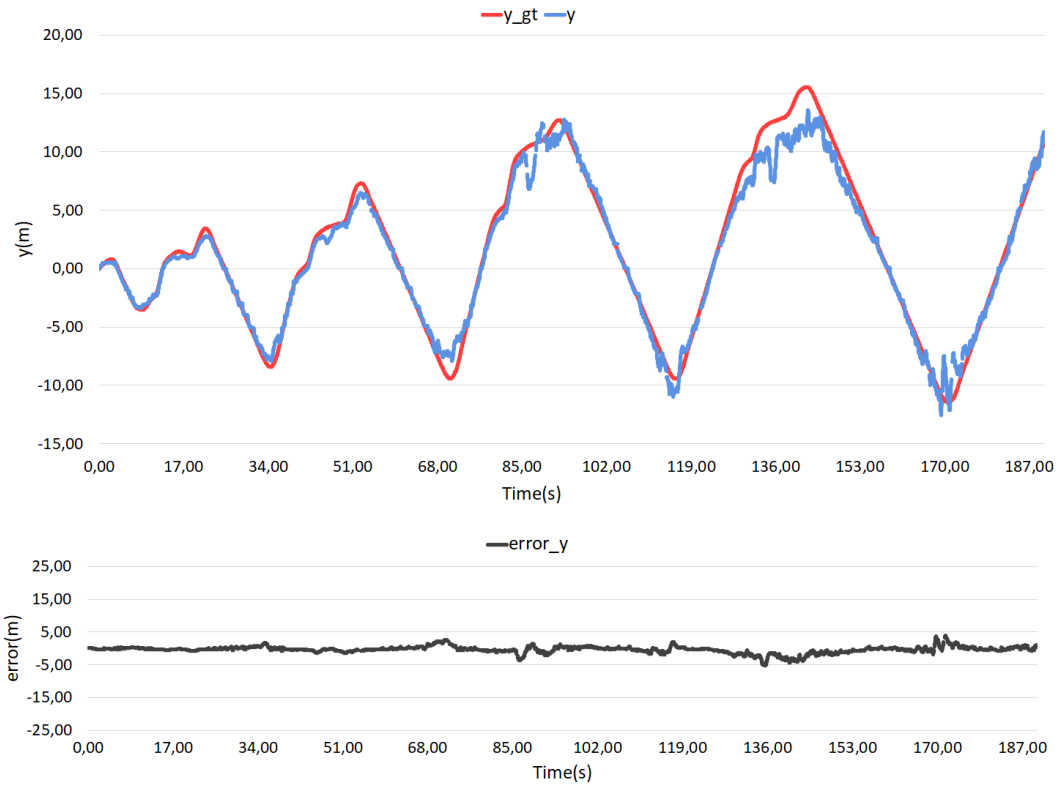


Figure A.6: Camera resolution simulation results for Y over time.

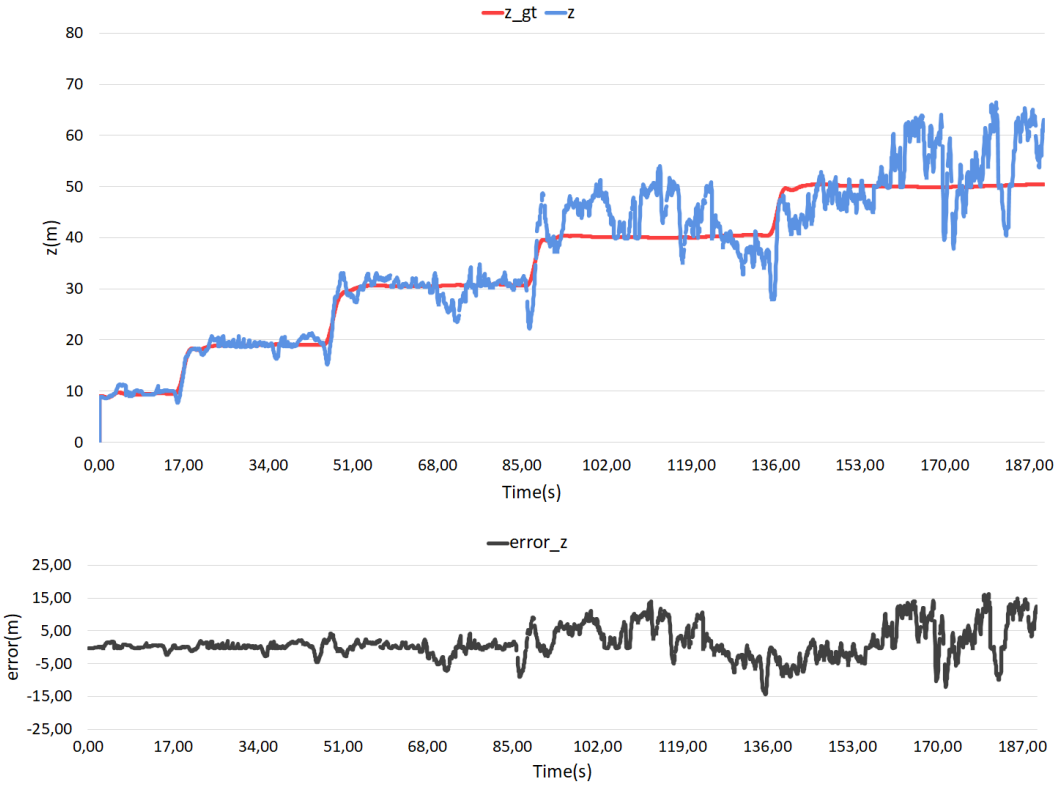


Figure A.7: Camera resolution simulation results for Z over time.

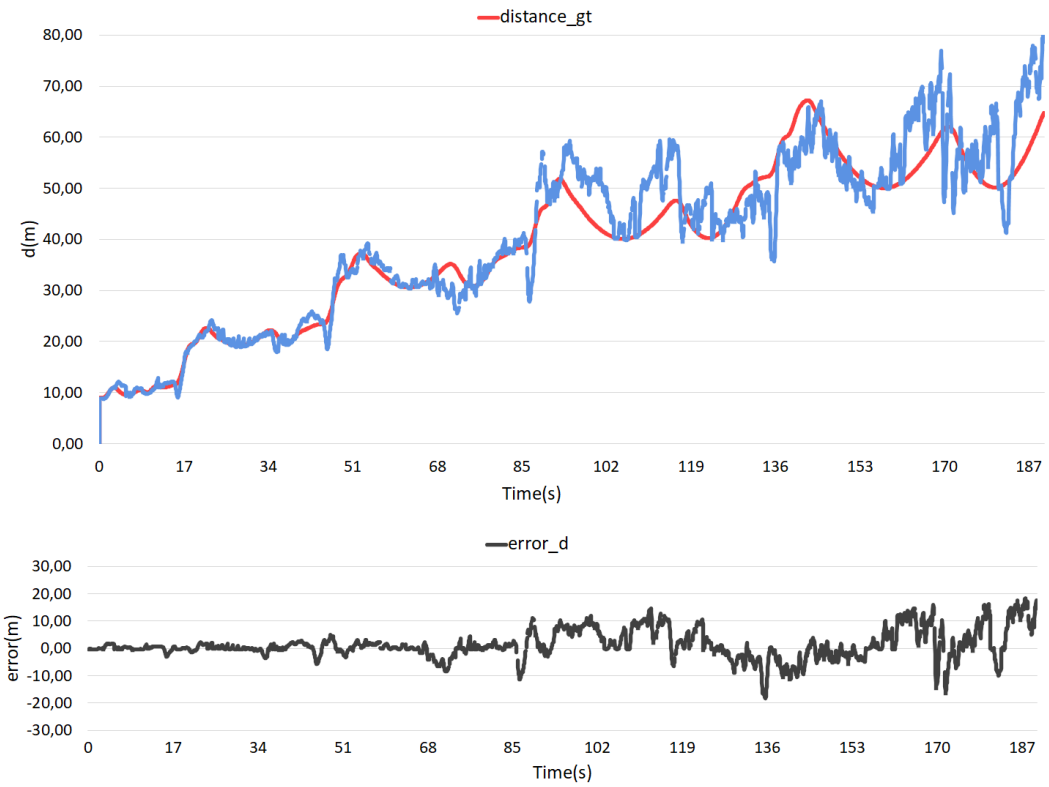


Figure A.8: Camera resolution simulation results for d over time.