UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

Deutsches Zentrum
für Luft- und Raumfahrt
German Aerospace Center

TREBALL DE FI DE GRAU

TFG TITLE: Real-time Vehicle Speed Estimation using Unmanned Aerial Vehicles for Traffic Surveillance

DEGREE: Double Bachelor's degree in Aerospace Systems Engineering and Telecommunications Systems

AUTHOR: David Saiz Vázquez

ADVISOR: Pablo Royo Chic

SUPERVISOR: Miguel Ángel Fas Millán

DATE: September 6, 2023

Títol: Real-time Vehicle Speed Estimation using Unmanned Aerial Vehicles for Traffic Surveillance
Autor: David Saiz Vázquez

Director: Pablo Royo Chic

Supervisor: Miguel Ángel Fas Millán

Data: 6 de setembre de 2023

Resum

Els drons són una eina emergent per a la vigilància de tràfic; així i tot, manquen de la capacitat d'obtenir la velocitat dels vehicles a la carretera per si mateixos. Aquest treball de fi de grau presenta el disseny, implementació i estudi d'un sistema per a detectar la posició, velocitat i el tipus de vehicle utilitzant el flux de vídeo aconseguit dels drons. La solució ha sigut creada per a ser funcionar amb qualsevol vehicle aeri i està adaptada als drons del projecte europeu LABYRINTH, del que ha format part el treball.

L'eina fa servir el senyal de vídeo d'una única càmera i les dades de telemetria del dron per a detectar, rastrejar i projectar els objectes presents a la carretera des de la imatge a la realitat. Això ens permet estima la posició i velocitat. L'algorisme de detecció i seguiment implementat és l'algorisme Simple Online Real Time, denominat comunament SORT. Un cop adquirida la posició, es genera un altre flux de vídeo que mostra la imatge original amb els vehicles identificats emmarcats amb la seva velocitat, amb un temps total de càlcul menor que l'interval entre fotogrames. Després de la seva implementació, es presenten proves executades en un entorn simulat amb un estudi de les dades obtingudes i els resultats de les proves de vol reals fetes durant la vigilància de tràfic del projecte LABYRINTH.

D'aquesta forma s'han assolit els objectius originals amb la utilització mínima de recursos, lògica fàcilment accessible i codi obert, per a aconseguir un equilibri òptim entre la funcionalitat en temps real i la precisió pel que fa a la posició dels vehicles.

Title : Real-time Vehicle Speed Estimation using Unmanned Aerial Vehicles for Traffic Surveillance

Author: David Saiz Vázquez

Advisor: Pablo Royo Chic

Supervisor: Miguel Ángel Fas Millán

Date: September 6, 2023

Overview

Drones are an emerging tool for traffic surveillance; however, they inherently lack the capability to solely obtain vehicle speed on the road. This Bachelor's thesis presents the design, implementation and study of a system to detect the position, velocity and type of vehicles using the video stream obtained from drones. The solution is created to be used with any kind of aerial vehicle but is tailored for the drones in the European project LABYRINTH, of which the thesis has been a part.

The tool utilizes the video feed from a sole camera and the telemetry data from the drone to detect, track and project the objects present on the road from the image into reality. This allows for an estimation of their position and speed. The detection and tracking algorithm implemented is the Simple Online Real Time algorithm, which is often referred to as SORT. Once the position has been acquired, another stream is generated that displays the same video, but with the bounding boxes, velocity and confidence ratings of all identified vehicles, with an overall computing time lower than the frame rate. After implementation, the tool underwent testing in a simulated environment to determine its assets and shortcomings, and was used during the LABYRINTH traffic monitoring flight tests.

The Bachelor's thesis achieves the aimed objectives with minimum resource utilization, using readily available logic and open-source software to strike an optimal balance between real-time functionality and precise detection of vehicle position.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

CCTV Closed-circuit television. 34

CIAR *Centro de Investigación Aeroportada de Rozas*. 55

CNN convolutional neural network. 14, 16, 25, 57

COCO Commmon Objects in Context. 30, 54, 57

CPU Central Processing Unit. 25, 38

CUDA Compute Unified Device Architecture. 25, 26, 30

CuDNN CUDA Deep Neural Network. 25, 26

deepSORT deep Simple Online Realtime Tracker. 19

DGT *Dirección General de Tráfico*. 20

DLR German Aerospace Center. 5, 58

EC2 Elastic Computing. 24, 33

FPN Feature Pyramid Network. 14, 15

GPU graphics processing unit. 23–25, 30

HLS HTTP Live Streaming. 33, 34, 37, 39

HTML HyperText Markup Language. 38

INTA Instituto Nacional de Técnica Aeroespacial. 23, 29, 32, 34, 39

IoU intersection-over-union. 17, 19

JSON JavaScript Object Notation. 35, 36, 38

LiDAR light detection and ranging. 19, 20, 40, 41, 57, 58

NED North East Down. 32, 33

OpenCV Open Computer Vision. 25, 26, 30, 33

OS Operating System. 35

OSDK Onboard Software Developing Kit. 32

R-CNN Region-Based Convolutional Neural Networks. 15

RTCP Real Time Control Protocol. 34

RTMP Real-Time Messaging Protocol. 33, 34

RTP Real Time Protocol. 34

RTSP Real Time Streaming Protocol. 33, 34

SORT Simple Online Realtime Tracker. 16, 18, 19, 23, 26

TCP Transmission Control Protocol. 33, 35, 36, 38

UAS Unmanned Aircraft System. 5, 21, 40, 58

UDP User Datagram Protocol. 29, 34, 35

UI User Interface. 37

UML Unified Modeling Language. 28

YAML YAML Ain't Markup Language. 36

YOLO You Only Look Once. 15, 16, 24, 29, 30, 42, 50

NOMENCLATURE

α_e	Angle used $[degree]$
Δe	Error $[m]$
Δh	height of the centroid $[m]$
γ	Skew factor $[px]$
ϕ	Roll angle $[degree]$
ψ	Yaw angle $[degree]$
θ	Pitch angle $[degree]$
a	Aspect ratio
f_x	Focal length in the X axis $[px]$
f_y	Focal length in the y axis $[px]$
K	Intrinsic matrix
R	Rotation matrix
s	Depth factor $[m]$
T	Displacement matrix
u	Position in picture in the X axis $[px]$
v	Position in picture in the Y axis $[px]$
x_0	Offset in the X axis $[px]$
x_{px}	Position in picture in the X axis $[px]$
y_0	Offset in the Y axis $[px]$
y_{px}	Position in picture in the Y axis $[px]$

INTRODUCTION

The LABYRINTH 2020 project is a European project to develop and validate swarm drone applications that improve safety, efficiency and security through the use of 4D path planning algorithms and a new set of U-space services to support autonomous flight.

In this 3-year-long project, the DLR (German Aerospace Center) was responsible for the implementation of U-space services whilst partners from Italy and Spain developed the flight planning system and provided the testing infrastructure. Various use cases were supported: airport monitoring, emergency management, waterborne and port surveillance and finally traffic surveillance. As the project approached its final stage, a method was required to gauge the velocity of the vehicles, thus this Bachelor's thesis was created.

The dissertation proposes and implements an approach to obtain the position and velocity of the automobiles on the road through the utilization of drones in real-time. The fundamental concept was to utilize solely the data that may be procured from a commercial drone to be able to execute it with the fleet presently employed for the project. Concretely, the drone selected for the tests was a DJI Matrice 300 RTK, possessing excellent flight capabilities. It was equipped with a Zenmuse H20T camera, which provided exceptional image resolution.

Objectives

The tool's objectives align with the LABYRINTH project's needs. The primary aim of the thesis is to extract the location and velocity of cars and other vehicles via static drones. Given its application by the police, it is imperative to acquire real-time information. In most cases, a minimum of two drones flying simultaneously is envisioned, and the solution should operate seamlessly on all road types, independent of conditions and type.

Another requirement of the system is that the detector must track not only cars but also trucks, motorbikes and bycicles. As such, techniques geared towards identifying only cars are not suitable. Additionally, employing the tool to detect boats and even other Unmanned Aircraft System is suggested.

As the objective is to obtain the speed for traffic surveillance, it must be easily and conveniently readable by the end user. This can be achieved through the creation of tools that offer access without any requirement of technical knowledge.

Thesis Timeline

In order to test the tool developed in a real use case, it had to be implemented prior to scheduled flights. The timely achievement of objectives was possible thanks to prior knowledge of software engineering and drone technology.

A proficiency in Linux, C++, Python, Neural Networks and other technologies was useful as they form the foundational pillars of the project. Figure 1 illustrates a simplified Gantt diagram with the planned tasks. During the LABYRINTH timeline, flight tests were conducted in Italy and Madrid. Coordinating with various partners for these tests took a considerable amount of time. The complete diagram is available in the annex A.

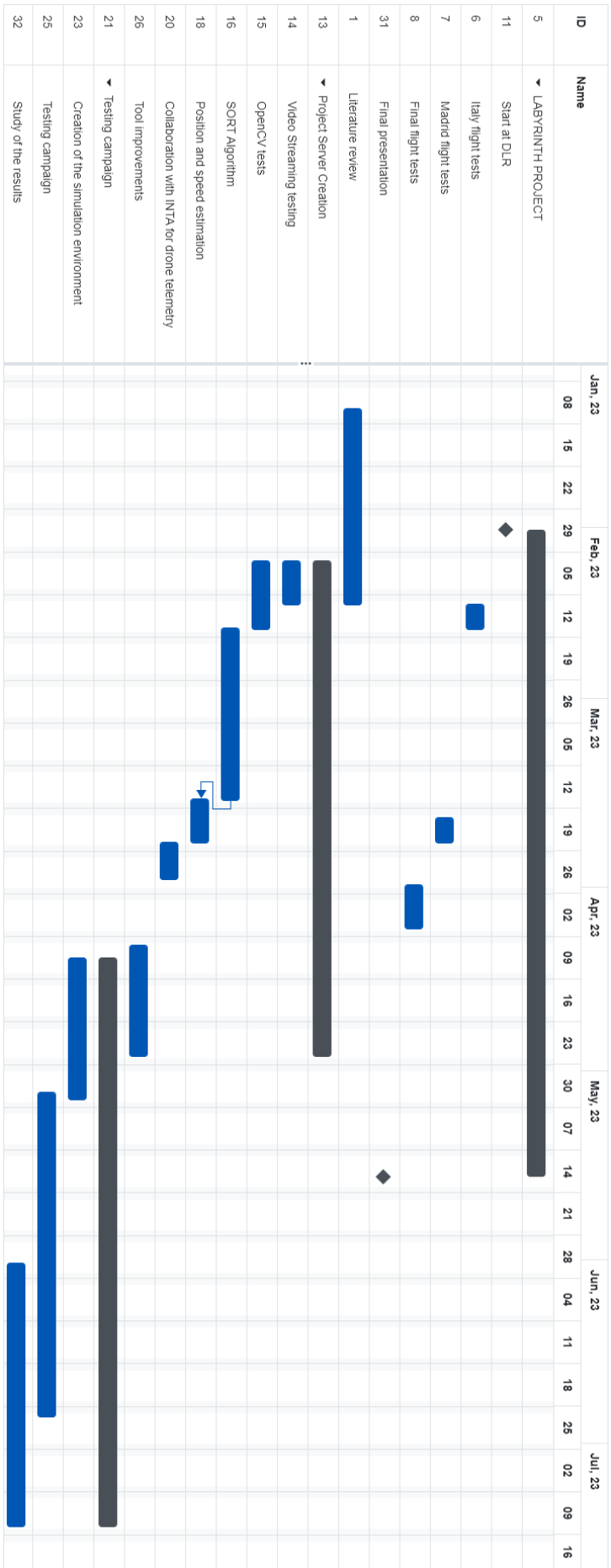


Figure 1: Simplified Gantt diagram for the project duration

CHAPTER 1. STATE OF THE ART AND THEORETICAL BASIS

1.1. Overview of Vehicle Speed Measuring Systems

Speed measuring is an active field of study, and as such multiple approaches are being developed at the same time. The most basic system for detecting the speed of a vehicle is the time radar, in simpler terms, measuring the time it takes a vehicle to go from one measuring spot to the next one and obtain the mean speed on that road fragment. Even though it is a working method, is not the optimal solution. Another caveat of this method is that it cannot obtain the instantaneous speed, the true velocity in the moment.

Regardless of trying to measure mean or instantaneous speed, the amount of variability makes it less efficient, so other methods based on sensors have been developed. The method currently used for the vast majority of traffic enforcement systems is the speed radar. Speed radars use the Doppler effect to calculate the speed of the approaching vehicles.

The project employs a computer vision approach. However, prior to explaining how it will work, it is worthwhile presenting current alternatives with their respective strengths and weaknesses. This will provide a wider perspective on radar systems for traffic surveillance.

1.1.1. Doppler Speed Radars

The Doppler Effect, also known as the Doppler shift, is the change of wavelength caused by the relative speed between the observer and the source. This shift is due to the wavefronts being closer to each other when the source is moving toward the observer and vice versa, as it can be seen in Figure 1.1.

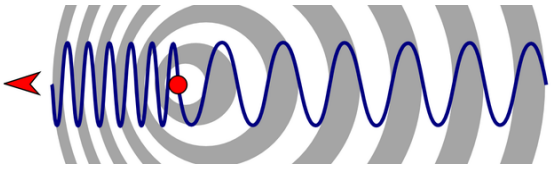


Figure 1.1: Doppler effect

Doppler radars use this shift in frequency to detect the speed of the vehicle. The radar generates an electromagnetic wave and then analyzes the returning bounce. Doppler radars are also used in aviation because they discern between moving aircraft and clutter such as clouds by the speed detected by the sensor.

The generated wave is usually sent in the shape of pulses, giving it the possibility to also obtain the distance to the targets and to detect them with the same radar. Some radars might be continuous-wave radars, which send constantly the electromagnetic signal. These sensors are not able to detect multiple vehicles at the same time because there will be a constant rebounded signal, and if there were multiple vehicles it would be impossible to differentiate them, it would look as if there is just a bigger vehicle.

When it comes to traffic enforcement, these radars are installed either beside or over the road. They are paired with a camera so once an infraction is detected, an image of the vehicle can be obtained and automatically detect the license plate number. Usually, the radar and the camera are built with a very small window of vision. This assures that only one vehicle is detected by the radar and visible in the camera at the same time.

Even if the basic principle is simple, Doppler radars have a set of caveats that affect their performance. The range of the system is proportional to the size of the antenna and the power it sends. Its antennas have to be small to fit, so it does not have a very good range. In some cases, due to the power limitations, it is necessary to use a continuous-wave radar, and therefore for it to work it can only point to one vehicle at a time. Furthermore, it is affected by clutter in the surrounding area causing incorrect readings. Finally, these radars are costly. Even with all these defects, the Doppler radar is the most precise tool and therefore is the standard speed measuring system for all radars commercially available.

1.1.2. Pavement-based Speed Detectors

A simpler method for detecting the speed of a vehicle is using sensors on the pavement. Detecting vehicles using systems such as induction rings has been already implemented in cities all around the globe. Inductive loops use the fact that the cars are made mostly of metal by measuring the inductance of a loop where an alternating current is applied. Similar to adding a ferrous core to an induction coil, the car causes a measurable change in the inductance on the loop. Even if the principal concept is simple, in reality, more factors play a role and measuring the impedance is more complicated. For example, eddy currents created by the edges of the car can be strong enough to completely disrupt the expected inductance changes. Inductive loops are largely used, for example, in the coordinated traffic system of Sydney [1].

Assuming the detection of a vehicle, two induction loops can be used to calculate the duration taken by the car to travel from one point to another, thus providing an estimate of the speed. [2] conducted a study on the possibility of using one induction loop for this purpose. Induction loops are highly effective for monitoring traffic due to their cost-effectiveness and ease of detection of cars. However, their suitability for speed estimation is limited.

1.2. Vision-based Vehicle Speed Estimation

While Doppler radars may currently be the most effective solution, developments in technology are providing alternative options. As cameras become more affordable and computational power increases, vision-based methods are becoming increasingly viable. Given the widespread installation of traffic management cameras worldwide, calculating speed from video frames would provide significant benefits in terms of road safety. [3] presents a thorough survey of methods studied up to 2021. The main process of a vision-based system is straightforward. When provided with a set of camera frames, the first objective is to recognise and detect the vehicles in the images. After detection, the vehicle must be accurately tracked to acquire its trajectory. Finally, a method is required to convert the pixel-based coordinates into real-world coordinates. These three stages can be carried

out using various techniques based on accuracy, system capabilities, and the recognized characteristics of the environment.

1.2.1. Camera properties

One of the largest sources of error in computer vision is the physical situation and properties of the camera, which can greatly vary depending on the situation and device. Intrinsic parameters refer to the aspects that affect the camera itself, while extrinsic parameters concern the position and attitude of the camera in relation to world coordinates. Extrinsic and intrinsic parameters establish the connection between the camera's viewpoint and the surrounding world. Therefore, obtaining these values enables the system to recover the actual coordinates of the objects represented in the image. Figure 1.2 exemplifies the effect of each parameter on the image. To experiment a different settings the head to the Jupyter notebook found in appendix B.

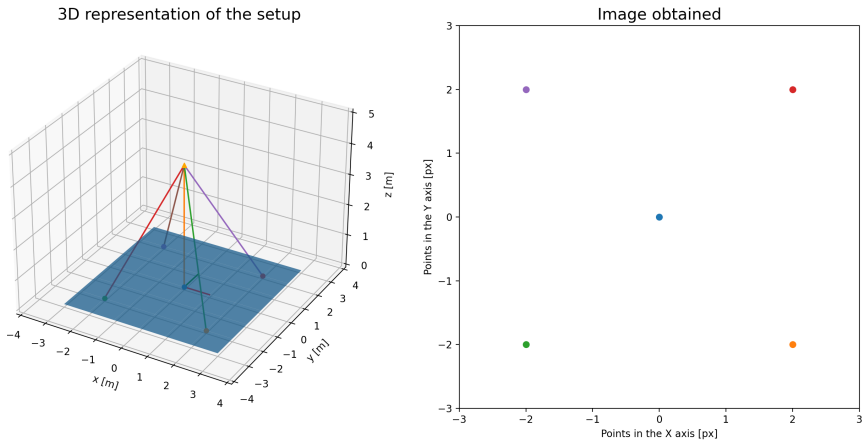


Figure 1.2: Base image to see the effects of the parameters

a Extrinsic parameters

Extrinsic parameters of a camera can be obtained by determining the relationship between the camera frame and the ground. This relationship can be broken down into two components: displacement (Figure 1.3) and rotation (Figure 1.4). The displacement refers to the distance from the world center of reference to that of the camera.

The direction of the vector is relevant, as the extrinsic parameters are given always going from the world coordinates to the camera coordinates. This also applies to the rotation matrix, which can be constructed from different sources. For our purposes, the rotation matrix will be determined using the air navigation Euler angles, known as roll (ϕ), pitch (θ) and yaw (ψ).

These parameters are normally in the format of a matrix size of 4x3 in such a way that the multiplication of the world coordinates becomes the real position. The matrix can be found in Equation 1.1.

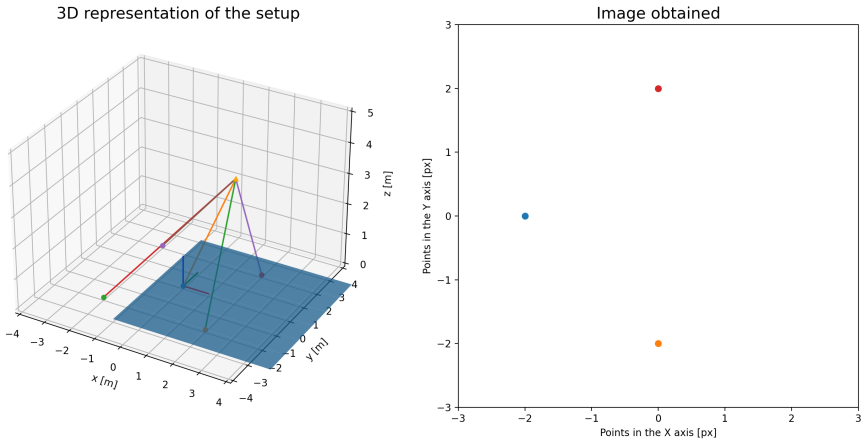


Figure 1.3: Effect of lateral displacement in the image obtained

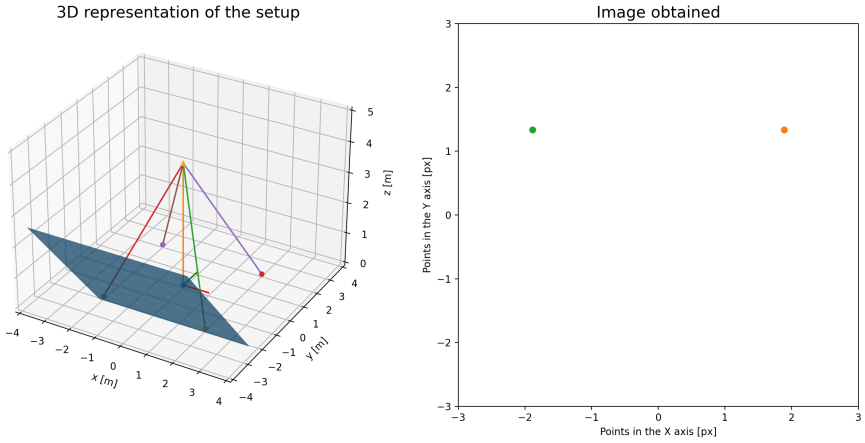


Figure 1.4: Effect of changing the angle of view for the image

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.1}$$

Instead of using a three-dimensional world vector, a fourth value is incorporated. This is widely employed in planar projection as it enables to do a full transformation between two planes using a sole matrix. The determination of the object's rotation and displacement can vary depending on the scenario. Fixed cameras necessitate a one-time measurement, while systems such as drones must continuously adjust the matrices to keep pace with the movements.

b Intrinsic parameters

The intrinsic parameters of a camera are the properties defined by the sensor of the camera and the relationship between the focal point and the image plane. These parameters can be separated by the effect they cause on the resulting image.

One parameter is the focal length (f), which is the distance between the sensor and the equivalent focal plane. The focal or camera plane is the plane that would where the objects are projected when the image is captured. This can also be understood as the ratio between the pixels in the image and their position in the plane. Normally given in milimeters, it may require a unit conversion having into account the ratio between the sensor size and the pixel density. Focal length can be either split into f_x and f_y or used alongside the sensor ratio. Figure 1.5 displays the effects of using a shorter focal length.

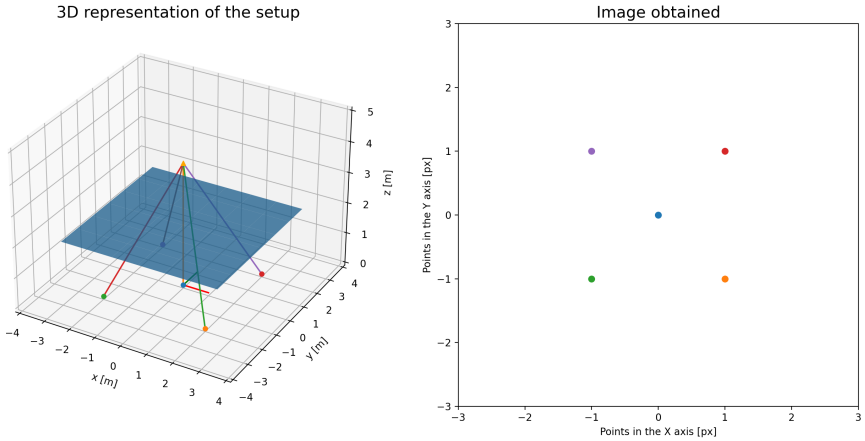


Figure 1.5: Effect of using a shorter focal length

Therefore, the second parameter for offset must be adjusted for both the x and y-axis. The offset parameter determines the origin of the image, while the other parameters determine how the image is processed. The camera is usually centered, but images have their origin point in the top corner. Figure 1.6 shows that the result is the same, but with a closer look, the axes differ. The offset parameter determines the origin of the image, while the other parameters determine the framing and scope.

Furthermore, certain sensors exhibit skew, indicating that the true pixels are not arranged in a perfect grid, but rather are misaligned. This skew can be seen as an angle between the expected rectangle and the real parallelogram shape of the sensor (see Figure 1.7).

The parameters are given in the shape of a matrix that applies all the modifications. It is called the intrinsic matrix of a camera (see Equation 1.2).

$$K = \begin{bmatrix} f_x & \gamma & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f & s & x_0 \\ 0 & af & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

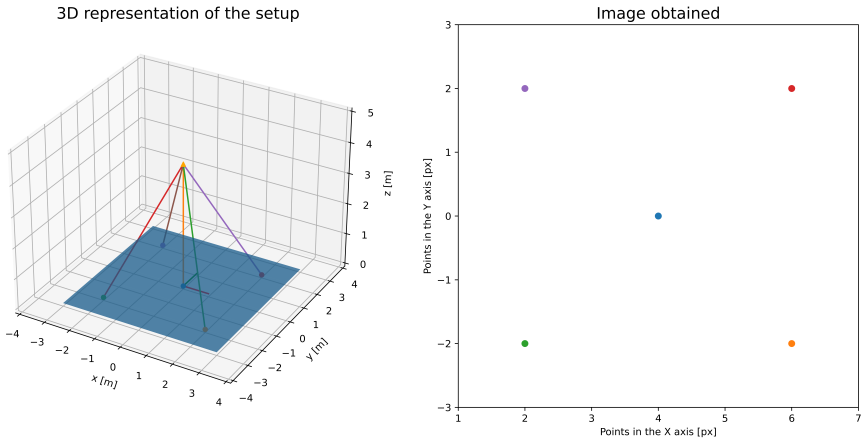


Figure 1.6: Effect of using offset in the image

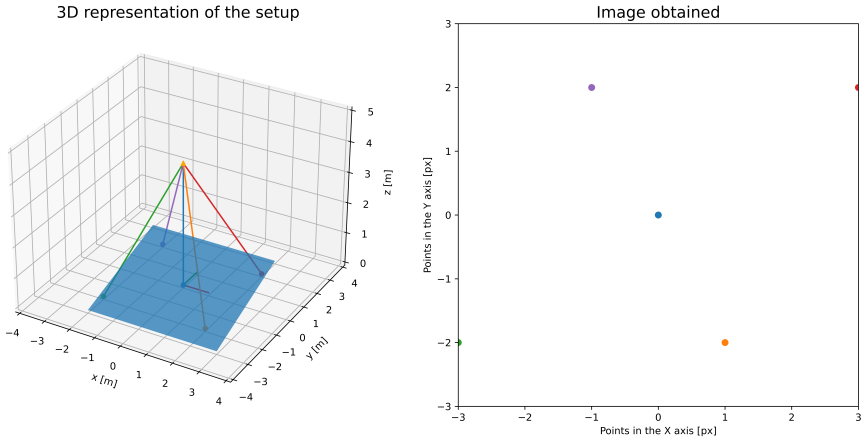


Figure 1.7: Effect of using skew in the image

c Other parameters

The intrinsic matrix assumes that the image is captured by an almost perfect pinhole camera, and it only has into account linear errors. Due to the nature of lenses, most cameras suffer from nonlinear deformations called distortions. The unequal bending of light can cause the images to bend in the corners producing barrel effects (As seen in Figure 1.8). It is also possible that the lens and the camera sensor are not perfectly aligned, resulting in a stretching effect. To address these issues, the image undergoes a series of corrections based on the camera model's specific parameters, which are typically applied prior to receiving the video, resolving the aforementioned issue.

When using both the intrinsic and extrinsic matrices, it becomes feasible to transform the position of an object in the real world to its corresponding position in the camera plane, as shown in Equation 1.3.

(a) Barrel effect caused by the lens (b) Resulting image after corrections

Figure 1.8: Barrel effect on an image

$$s \begin{bmatrix} x_{px} \\ y_{px} \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{1.3}$$

Obtaining an accurate estimate of these properties is crucial for approximating the speed, as they establish the correlation between the pixels in the image and real-world positions. Depending on the camera used, these may already be known, but in some cases, they must be determined from the image, increasing the task's difficulty. Some solutions such as the method proposed in [4] calibrate the camera with the vanishing point, whereas others like [5] calibrate and rely solely on the focal length.

From an image it is impossible to determine the distance between the object and the image plane, resulting in the loss of the depth information. It can be seen in the factor s added to the resulting vector, which is the z-axis in the camera coordinates.

The thesis relies heavily on camera properties, as they establish the correspondence between pixels in the image and real-world objects. Chapter 1.2.4. provides a detailed explanation of how this relationship can be utilized.

1.2.2. Vehicle detection

Object detection is the task of localizing and classifying objects within an image. There are numerous methods available for object detection. If the camera is static, most methods utilize the the static background to isolate the vehicles as the moving entities (See [6]). While simple, this approach cannot handle occlusion and is ineffective when the camera is in motion, such as with drones. More advanced methods use vehicle features instead. After clearing the background, the images are processed to identify edges and other object characteristics. This method can tolerate some occlusion, however, it can slow down the system.

Another approach that has been receving a great deal of attention is learning-based detectors. With the advancements in machine learning it is now viable to use neural networks as the detector, obtaining a bounding box of every vehicle with relative ease. This method is based on convolutional neural networks.

a Convolutional neural networks

A CNN (convolutional neural network) is a type of neural network that uses the properties of the recurring patterns and features to decrease the computational cost of processing the input. CNNs have a broad use in image recognition due to the fact that in images patterns repeat, making it more efficient. A neural network is an algorithm that imitates the process of information travelling through neurons. In this context, a neuron is a weighted sum of a set of inputs that then is used in an activation function. These neurons are arranged in layers, separating the input and the output. What it is done is to tweak to weights of those sums using a dataset of inputs and the expected output of the system. With sufficient data and a neural network of the appropriate size, the system will be capable of predicting the resulting output for values that have not yet been trained. Typically, each neuron layer is fully connected, such that its inputs consist of the outputs of all the neurons in the previous layer. When considering images, every pixel may be viewed as an input. For a color image of 640x480 pixels, each neuron in the initial layer must optimise 1,288,800 weights. Due to the high number of inputs, an average computer cannot process this.

Using full images for training as input is excessive, but they tend to exhibit repeatable patterns, rathen than training the neural network pixer by pixel, it is possible to create a filter that takes into account only region of the image and apply it multiple times. This can be viewed as doing a convolution between a filter and the image, hence the name. This filter will be trained such as with a usual neural network layer, but with significantly fewer weights. Normally, convolutional neural network apply multiple filters to the image at the same time, adding a dimension to the layer called depth. These filters end up working as feature detectors, finding edges or specific color schemes within the image. This is exemplified in the 2012 winner of the ImageNet challenge (See [7]). In order to process the images he used a CNN with 96 filters of size 11x11x3 (11 by 11 pixels with 3 colors).

While these basic convolutional networks are useful for classifying images, issues arise when localization is necessary. In a real image, objects may appear in various positions and sizes. For instance, a car located in a far lane will appear substantially smaller compared to another positioned much closer to the camera. However, the objective is to detect both.

To resolve this problem, convolutional networks must firstly identify the areas of interest before classifying them accordingly. Originally, feature extraction was achieved using traditional approaches like the use of Gaussian windows to detect corners, but most current implementations now use a CNN. Concretely, the method that is giving the best results is called FPN (Feature Pyramid Network) [8]. A FPN is a feature proposal system that uses the already pyramidal structure of a CNN to extract the features.

Convolutional neural networks take the image as input and decrease in size with each step, capturing more intricate features with each layer. However, for larger objects, these features may generate smaller detections within the object due to the lack of scale perception. Instead of solely studying the last output, in a FPN evaluations are performed on multiple network levels. This allows the neural network to obtain details at all the scales of the image, as lower levels will receive pattern information of upper levels. A Feature Pyramid Network has has to be paired with a Region proposal network, as it is only a feature detector. Region Proposal Networks extract feature information from the Pyramid and convert it into a collection of boxes that can be classified objectively. This approach typically generates boxes that overlap and subsequently require suppression.

There are two families of detection convolutional neural networks, R-CNN (Region-Based Convolutional Neural Networks) and YOLO (You Only Look Once). Despite using similar methods for obtaining the features and classification, the R-CNN family first detects and then classifies, whereas YOLO performs both steps at the same time. The time used by YOLO is orders of magnitude smaller than R-CNN, allowing real time processing of the frames.

The YOLO network segments the image into a grid and then it classifies what is most probable to be, creating a set of bounding boxes anchored on that cell. The result is a huge set of boxes that have multiple classifications with different degrees of confidence. The name derives from the concept that it predicts and classifies at the same time, using the same feature extraction, making you only look once the image. It is significantly faster than its R-CNN counterpart, but it comes with limitations in object size due to the grid directly limiting what is detected in each box, and it is overall less precise. YOLO neural networks have been advancing and the newest iteration is YOLOv8.

YOLO versions

The original paper was authored by Joseph Redmon [9], and since its beginning it has undergone 8 revisions. The original YOLO, even though it was quite advanced for his time, had still some shortcomings that the following versions addressed. A year after YOLO, Redmon J. created YOLOv2 [10], which made a general improvement of the tool as well as increassing the number of classifiable categories to 9000. This is the reason it can be referred as YOLO9000.

YOLOv3 [11] enhanced the convolutional network and is the initial version to implement the concept of Feature Pyramid Networks. This version is the last one that Joseph Redmon worked on. Even if the following versions are labelled YOLO they do not directly follow the project. Consequently, different teams worked independently on each version, causing a disorganised advancement process.

Version 4 [12], version 6 [13] and version 7 [14] of YOLO improved on the technical aspects of the neural network, while keeping the vision of Redmon. Versions 5 and 8 were developed by Ultralytics [15], a team that focused on making the network more accessible for general use. Both versions have a low entry barrier, and YOLOv5 continued to be widely used even after the release of newer versions. Version 8 was released on January 2023 and offers outstanding performance while retaining the user-friendly Python libraries and training environment.

During testing, versions 5, 6 and 7 were used. The tool is able to interchange them as necessary, and for the results the one used was YOLO7, as it had the best results overall.

1.2.3. Tracking

Once the location has been identified, the next stage is to commence tracking. Detection is performed on a frame-by-frame basis, which implies that it has no recollection of previous events. In order to track an object, data from the previous frame needs to be compared with the current one.

Similar to detection, tracking can be done utilizing features of the vehicle or directly the

complete box of the vehicle. One instance of feature tracking via optical flow is documented in [16]. Optical flow is defined as the pattern of apparent motion of objects. It is based on capturing consecutive frames of the image and finding the pixel movements between them. This approach is based on two main assumptions, that the pixel intensities remain unchanged between two frames and that neighboring pixels experience a similar motion. The most commonly used method of detecting the movement is Lucas-Kanade, which uses the fact that the optical flow should be similar in a small window of the image to create an over-determined system of equations and then solve it with the least squares solution, estimating the speed at which the pixel has moved. The Lucas-Kanade method does not identify regions of interest, therefore, the first step in setting up an optical flow system is to locate them. This implies that this algorithm can only be applied to identify sparse points in the image. At times, it may be more desirable to obtain the complete field of motion for the image, known as dense optical flow. The Gunnar Farneback's algorithm solves the dense optical flow of two sequential images, however it is more computationally intensive.

Optical flow has its uses in basic scenarios. Nonetheless, it proves inadequate in tracking complicated routes and is vulnerable to occlusion. Furthermore, its efficacy relies heavily on the state of the footage. For instance, the intensity variation may be excessive and result in the system losing track of the vehicle if the road has shaded regions.

In the case of using a convolutional neural network detector, alternative solutions are utilized. By utilising the bounding boxes from the objects, it is feasible to link the position from one frame to the next by proximity. This straightforward solution can be improved to achieve higher quality tracking, for instance by implementing the SORT (Simple Online Realtime Tracker). This algorithm is used in the thesis, therefore it will be explained in detail.

1.2.3.1. Simple Online Realtime Tracker

SORT (Simple Online Realtime Tracker) [17], is an algorithm that uses the bounding boxes obtained from a YOLO-based detector along with a Kalman Filter and the Hungarian algorithm to track the objects. This technique of a quick detector combined with straightforward algorithms results in a fast tracker that can monitor multiple objects in real-time applications. Its limitations are due to its heavy reliance on the detector, as well as its lower precision compared to similar algorithms. However, the algorithm makes up for this with its impressive speed.

The tracking cycle is based on four phases. The initial stage processes the frame via a CNN to obtain the detections. In the subsequent stage, a Kalman Filter predicts the future bounding box for the vehicle on a frame-by-frame basis. Then, the Hungarian algorithm assigns the detections to the objects. Finally, the Kalman Filter updates the prediction with the detection to further improve the result.

The SORT algorithm stablishes a tracker equipped with a Kalman Filter for every object it detects. The Hungarian algorithm subsequently identifies the most promising detection for the tracker, which is then utilized to update said tracker.

Figure 1.9 presents a visual depiction of the SORT algorithm, illustrating the processing required for each frame. The complexity of the process increases with each subsequent frame.

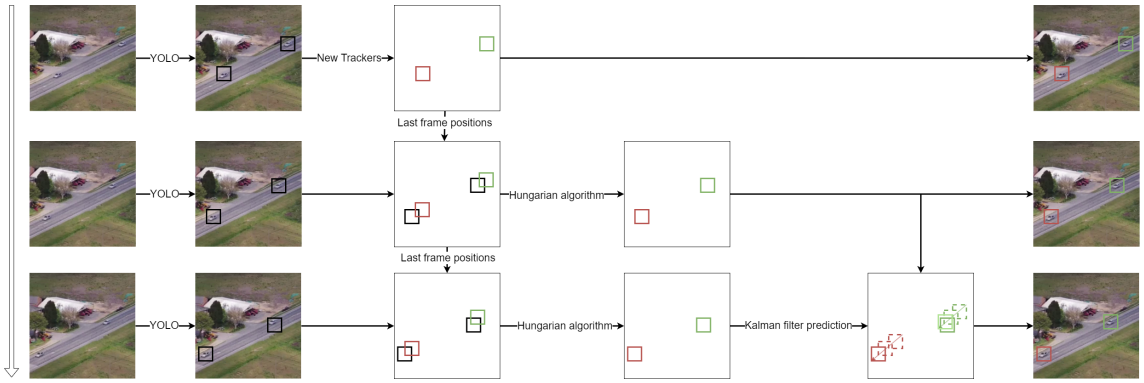


Figure 1.9: High level representation of the SORT algorithm

a Kalman Filter algorithm

The Kalman Filter algorithm is used to predict the system state while considering measurement and predictor uncertainty. This allows for the creation of a prediction matrix. It balances the measured value and system's predicted state to determine the system's response.

The algorithm begins with the state space representation, a connection between the system state and its derivative. The initialization parameters comprise this prediction matrix, the initial variance of the measurement, the uncertainty resulting from the error of initializing values, and an initial estimation.

The algorithm consists of two main steps, prediction and update. Upon initialization, a Kalman filter will predict the subsequent state while factoring the initial conditions. It will return a position and the variance for the prediction. Then, using a measurement, the filter will revise the state estimation and diminish the uncertainty to reduce the error in the procedure. The result of this update is optimal, meaning it will always minimize the uncertainty of the state. This process is visually represented in Figure 1.10.

b Hungarian algorithm

The Hungarian algorithm is a straightforward and optimal algorithm for determining the most cost-effective combination of two sets of data. Its aim is to compare the predicted boxes and the detections obtained and assign them in a way that minimizes the cost, represented by the IoU (intersection-over-union) of the bounding boxes. The IoU is the ratio of the intersection to the union of two boxes, portraying the extent of overlap between them as a fraction of their combined area.

As part of the process of initialization, the costs are stored in a grid. Then, the algorithm is comprised of four steps. Firstly, subtract the lowest element of each row from all elements within that row. Secondly, subtract the lowest value of each column from all numbers within that column. Thirdly, employ vertical or horizontal lines to cover the zeros in the grid. If the number of lines necessary to cover all the zeroes equals the number of lines, the algorithm is complete, and that is the solution for the algorithm. In the case of needing fewer lines, extra zeroes have to be created. To accomplish this, subtract the smallest number by all uncovered values and add double its value to the covered values. Then steps 3 and 4 are

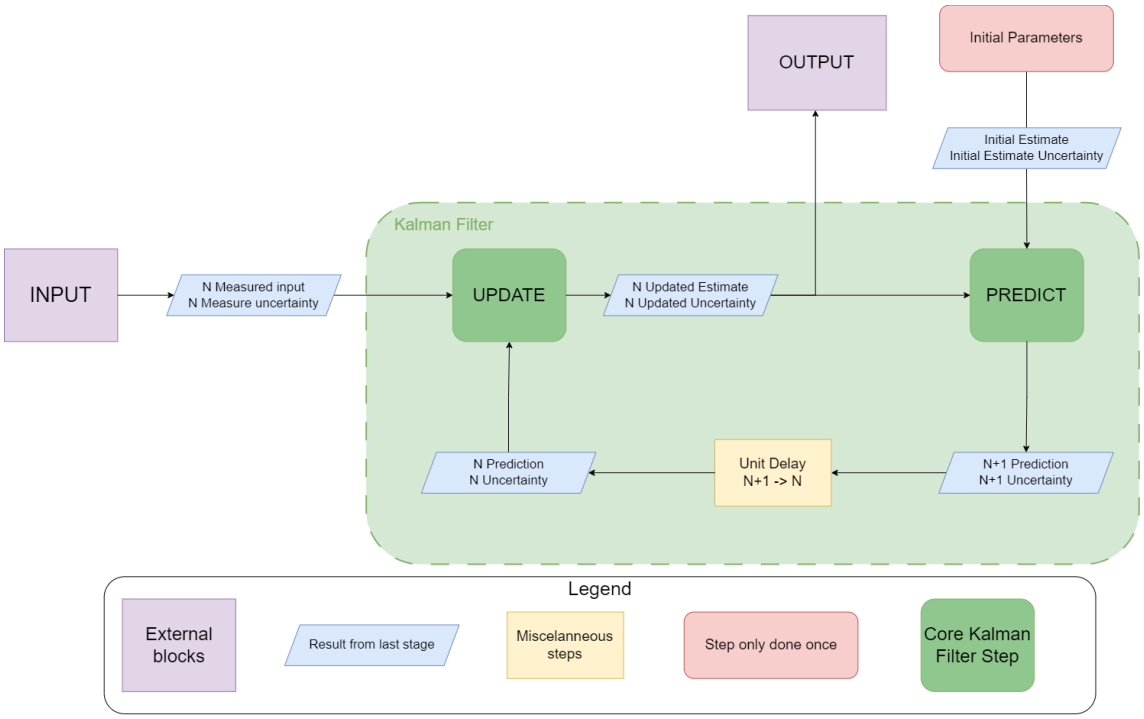


Figure 1.10: Kalman filter visual representation

repeated until a solution is reached. A visual representation can be seen in Figure 1.11.

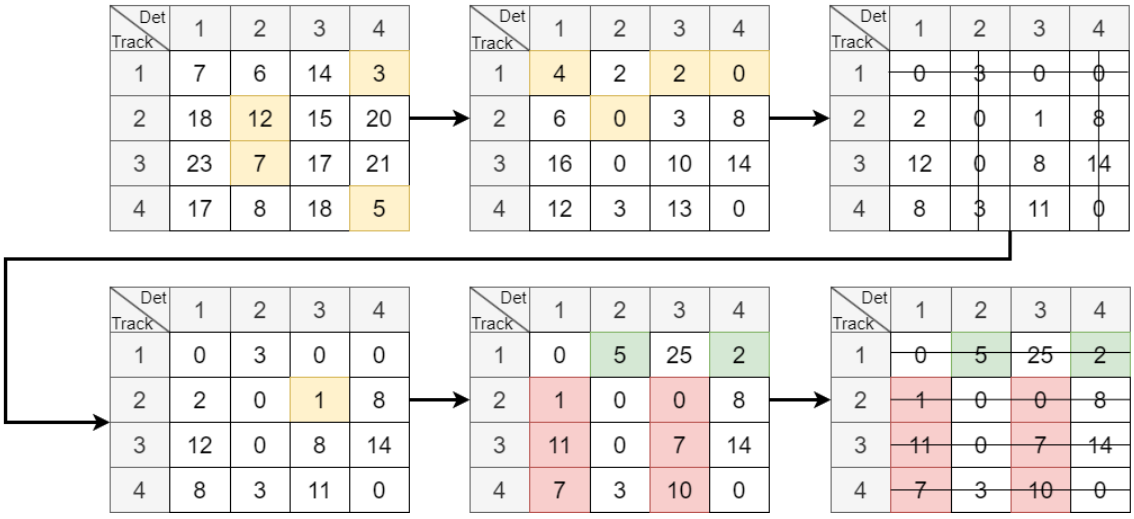


Figure 1.11: Hungarian algorithm representation

In the SORT algorithm, some boxes may lack an assigned match. For unassigned detections, novel trackers will be generated whilst those trackers lacking an assigned detection will utilize the prediction as the bounding box for that particular step. If a tracker is without detection spanning multiple consecutive frames, then this designates the object has exited the image and will be deleted.

1.2.3.2. DeepSORT

The SORT algorithm is a very simple yet profoundly effective approach for object tracking. Nonetheless, it presents certain drawbacks, such as constant identity switches, mainly in situations of high occlusion, such as crowded streets or busy roads. Hence, deepSORT [18] was created. deepSORT uses the same basic concept as the SORT algorithm, but instead of using IoU for associating the predictions and the detections it leverages appearance information and motion distance.

The appearance information refers to the similarity between features extracted from the detection box using another convolutional neural network. These features are then compared to those from the last frame to determine the degree of similarity between the images. Motion distance is computed using the squared Mahalanobis distance rather than the Euclidean distance. The Mahalanobis distance takes into account the covariance matrix of the system in calculating the distance between two multivariate vectors. The distance is determined by utilising the state variables, with the covariance matrix obtained through the Kalman Filter. This provides not just a quantification of the distance between both boxes, but also an understanding of the direction in which the prediction is heading, thus enhancing the outcome beyond that of a traditional Euclidean distance.

The resulting system is a better version of SORT that is more resistant to identity switches. While it is more complex, it still can maintain a good processing speed. Even if it is better with occlusion, it is too complex for the benefit it adds to the project. Furthermore, occlusion does not play an important enough role because it can be compensated by changing the height at which the drone is flying.

The decision to opt for the SORT algorithm over the superior deepSORT was based on its speed and simplicity. Although deepSORT would yield improved occlusion results, in reality, most occlusion occurs at low angles of vision, which is not typical for aerial scenarios. The benefit of a faster and simpler solution was the ability to handle larger swarms on a single machine. The authors of the publication (see [18]) achieved a processing speed of 40 frames per second for the deepSORT algorithm, whereas for SORT it was 60Hz.

1.2.4. Position and speed estimation

When attemptinh to determine the position of the vehicle the process is contrary to that of the previously mentioned camera matrices. The aim is to move from the image plane position to the actual position in the real world.

Given the fact that the depth is always unknown, it is necessary to estimate it. The optimal solution is to incorporate to the video a depth perception method such as LiDAR [19]. If an image and a corresponding depth map are available, the 3D position of any object can be accurately recreated. Alternatively, stereo vision can be employed, which involves obtaining the distance to the object from the small differences between the two camera angles of view. This method has two major caveats; the first is the importance of the synchronization of the cameras and the second is the limitations on the range. For the system to work correctly, the frames have to be perfectly in sync, because any error of time is treated as part of the difference of perspective, skewing the resulting depth. The stereo vision system's ability to see objects is dependent on the distance between the cameras. If they are further apart from each other, they will have more depth of field, but it

will become more difficult to find shared points of interest between the images.

It is also possible to use various features from the vehicles, such as the license plate of a car [20] or the width of the lanes [21] as distance reference. An alternative approach is to assume that the road is considerably flat and the vehicle's height makes negligible impact on position. This method, despite being the most imprecise, is the only one that fits the objectives as it offers the most freedom. While it would be desirable to incorporate car-specific characteristics like license plate length, it is unfeasible as the tool must detect all vehicles. Additionally, utilising car lanes is not possible due to the LABYRINTH tests taking place on a non-standard sized national road. Furthermore, as the drones are presently restricted to utilising only one camera, it was impossible to develop a resolution using stereo vision or LiDAR (light detection and ranging).

Additionally, the camera angle significantly impacts the accuracy of the speed estimation. Hence, we must consider these variables when estimating velocity. The distance the object has moved between frames relies on both its actual position and how much of the object is visible in each frame. For a low camera angle, the vehicle may partially obstruct the point of contact with the ground, leading to an inaccurate instant speed measurement.

This issue could be resolved by positioning the camera at an elevated angle above the street. This may be challenging for a stationary camera, but it is comparatively simpler when utilising a drone.

From the positions and frame rate, the speed of the vehicles can be estimated. Instantaneous speed may be theoretically achievable, but it is practically impossible due to the small shifts caused by drone movement or the Kalman filter, which create noise. This noise can be filtered by using the mean to reduce error. To obtain the mean speed, the newest value is added to the prior estimation. Instead of being equally weighted, the sum is weighted in favour of the most recent result. The purpose of weighting the mean in favour of more recent results is that some values may be incorrect, and this way older values are swiftly eliminated from the mean. Essentially, it creates a window of values.

1.3. Vehicle speed estimation with the use of drones

Drones, and multi-rotor drones in particular, offer a great deal of flexibility when it comes to traffic monitoring. They are easy to deploy and operate. With the current evolution of regulations, their use is becoming more common and the DGT (*Dirección General de Tráfico*), responsible for traffic management, monitoring and regulation in Spain, is already using them as a means of detecting infractions.

While drones are very useful, there are no commercial implementations for speed detection. It is not easy to mount a full Doppler-based camera radar on a drone due to weight limitations, and computer vision approaches are not yet accurate enough. Regardless of the method used, it is clear that having eyes in the sky is valuable for surveillance, hence the amount of research on the topic.

There are several studies in this area with varying degrees of complexity (see [22]). There are great examples of solutions such as [23]. They approached the problem with a hybrid between a learning-based and an optical flow solution. Its algorithm is explained in great detail. Other solutions such as [24] focus on testing what would be the speed accuracy

when the UAS (Unmanned Aircraft System) and the vehicles are moving, giving accurate results.

Drones can be very versatile, but with that versatility comes a lot of variability. Lighting can vary rapidly with a moving vehicle and the angle of vision affects the accuracy of the position estimation. There is also the issue of computation power. The more accurate the solution, the greater the resources required for implementation on board. If the solution is outsourced to another device, the results may experience delay, as it depends on the latency of the network in which the device is located. There are various challenges and every publication boasts distinct benefits and drawbacks. Therefore, the chosen solution depends largely on the objectives and constraints established.

CHAPTER 2. DESIGN AND IMPLEMENTATION

After going through the current research in computer vision, the decided approach was to use the SORT Algorithm for the detection and tracking of vehicles and basic plane projection to transform the objects detected on the image into positions in a three-dimensional space. To solve the depth problem it was assumed that the road was a flat surface. Finally, speed is measured using a pondered mean.

The project can be divided into a primary module that handles the video frames and a series of auxiliary applications that enable the system to process several streams at once, save the generated data and present the findings.

Drones might not possess adequate computational power to utilize the SORT algorithm as it relies on a GPU. Therefore, rather than attempting to accommodate the application on the drone's onboard computer, the information is processed by a server (more information in section 2.1.). This approach provides additional advantages as it comprises a disconnection between the drone and the image processor.

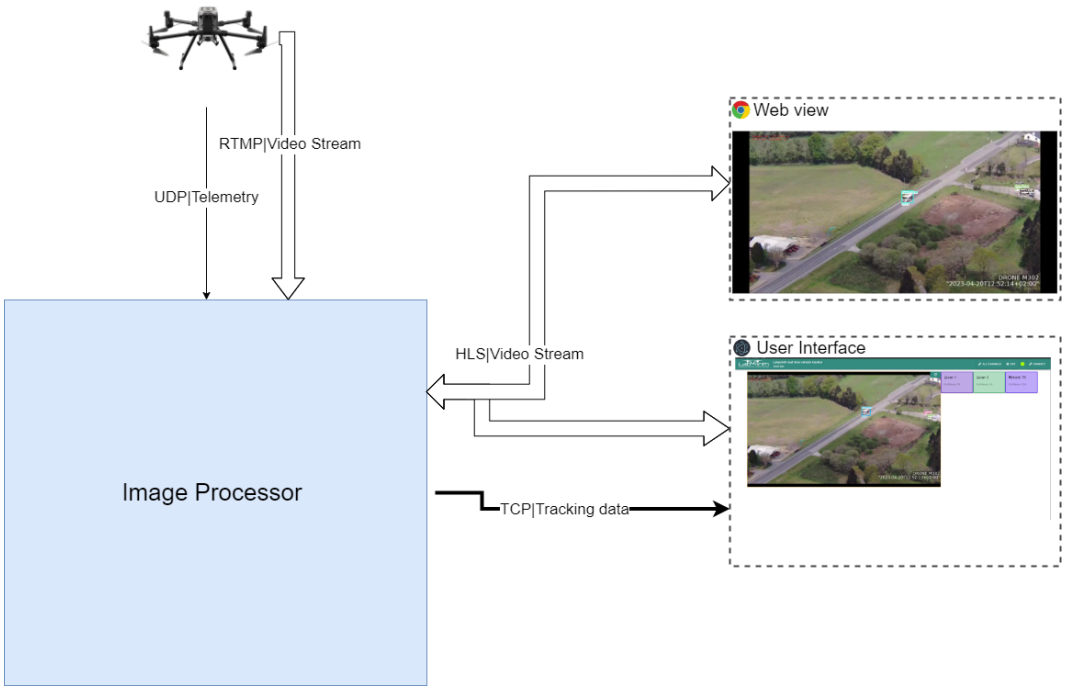


Figure 2.1: General diagram of the system

In Figure 2.1, a simplified illustration of the tool is presented. Although the primary project is carried out in the Image Processor, additional actors are necessary for the project to function. The flow information is simple: the drone transmits information to the Image Processor, and subsequently, the various user applications receive the video and information from the server. The labels in the line explain the protocol used for each connection. Double lines are reserved for video streams and single lines for only data.

The official LABYRINTH drones were used as the source of video and telemetry. These are programmed by INTA, a government association located in Spain. Therefore, cooperation with them was needed to establish the communication protocol between the drone and

the image processor server. Usually, the video stream is transmitted through the drone controller, but in their case the telecommunications link was used. Their telemetry used a redundancy system that could use multiple antennae and 4G at the same time, using the optimal telecommunications channel all the time.

The two applications act as video receivers for end users. Initially, only a website was provided for users to access the video. However, it was discovered that during heavy traffic, the labels and boxes would cause clutter, making it difficult to read. To improve the user's experience, a computer application was developed, presenting the video and vehicle speed in a more legible format. As an additional advantage, an endpoint has been established that any other application can connect to and calculate the speed of the vehicles without the necessity of extracting it from the video.

Finally, one of the aims of the LABYRINTH project was to utilize the tool with multiple drones simultaneously, so an additional screen was included to display the processed streaming from four drones at once. The server permits more, but this exceeded the requirements for the LABYRINTH project, which employed two camera-equipped drones at most during the flight tests.

2.1. Image processor: detection, tracking and positioning server

This server was built on an EC2 (Elastic Computing) instance from Amazon Web Services [25]. This enables the software to operate on a robust machine while also remaining easily accessible through the internet. Alternatively, the system could function on an edge computer with sufficient computing capability. The crucial characteristic of the server is its GPU model. Older cards might not support the YOLO models used for tool. In Figure 2.2 the image server can be seen in more detail. The code for the image processor is publicly available here.

This computer hosted three distinct services: a streaming server, an image processing server, and a MySQL database. The video and image processing servers were segregated due to the intricacies involved with video streaming, which entails numerous protocols and edge cases. Programming the streaming server in its entirety was deemed impractical and was therefore outsourced (refer to [26]).

2.1.1. Hardware Components and Specifications

The Elastic Computing Service from Amazon offers fast and scalable deployment of virtual machines. This service allows for the creation of virtual machines or instances of popular operating systems in mere seconds. The same virtual machine can also be run on different systems, enabling scalability without the need to purchase and configure your own server. For our purposes, the primary advantage provided by it was the capability to have highly potent GPU (graphics processing unit)s accessible from any location and to deploy the server with relative ease and safety.

For the testing and deployment of the tool, two different types EC2 instances were used. Initially, the system was tested and built using the model called p2.xlarge. This instance

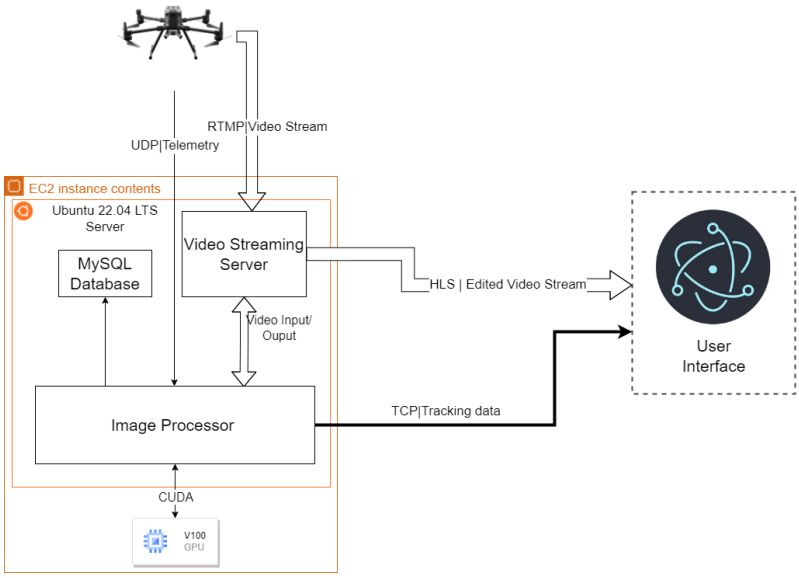


Figure 2.2: Diagram of the server hardware

was cost-effective but had a GPU which does not receive software updates anymore. As a result, it is incompatible with the latest versions of artificial intelligence libraries and dependencies.

Once the tool was prepared for testing, it was upgraded to the p3.2xlarge server, which features a V100 NVIDIA GPU and can run the most up-to-date versions of major computer vision libraries. This allowed for the implementation of more effective neural networks and an overall improvement in performance. With this upgrade, the server's processing speed doubled and enabled the use of more drones simultaneously.

2.1.2. Software tools and libraries

Before describing the software implementation, it is essential to provide an overview of the employed technologies. The program is entirely coded in C++, chosen due to its superior speed compared to other languages, such as Python. Python is generally favoured as an entry point for most artificial intelligence frameworks for its relative ease of use, but the most widely used computer vision library is written in C++.

OpenCV (Open Computer Vision) [27] is an open-source library that has all the required functions and structures necessary for any project that involves image processing. It is also one of the top libraries for matrix algebra, making it suitable for all mathematical operations. Additionally, it is able to run convolutional neural networks and can use CUDA (Compute Unified Device Architecture) to communicate with the graphics card of the computer, allowing for GPU computation. By using GPUs to perform operations, CUDA turns a routine that would take 400 ms on a CPU into just 4 ms. To run a CNN on CUDA, one must utilize CuDNN (CUDA Deep Neural Network), a library for deep learning that is accelerated by GPU. This library contains optimised algorithms for operating neural networks on NVIDIA GPUs.

The library's drawback is the absence of packages compiled with the requisite functions.

Specifically for the thesis, it was necessary to compile OpenCV with streaming capabilities, CUDA, CuDNN, and some other optional packages. As straightforward as it may seem, the compilation encountered numerous difficulties, particularly when utilizing the p2.xlarge instance. This was due to the software's outdated version being incompatible with the most recent one, thus requiring the downgrading of the compiler from the standard Ubuntu version.

OpenCV is extensively used for the project. It serves as the detection library, the Kalman filter implementation and the matrix operations. It provides a series of structures and classes that aid in the administration of images and positions. In essence, the project would have been impracticable without this library.

Although the library is impressive, it possesses certain limitations, particularly regarding detection. The core objective of the OpenCV library is not to execute neural networks; hence, it is not in line with the current conventions. However, this shortcoming is inconsequential for the thesis, as it does not require the most advanced detector to produce satisfactory outcomes.

For streaming video, FFmpeg software is utilized, which is a collection of free software capable of recording, streaming and converting any audio or video. It is the standard for video handling and allows control over the parameters of the video, from the types of frames created to the streaming protocol used. In general, it is operated via the terminal but it can also be implemented as a library within the code. OpenCV incorporates it for the management of video creation.

The Hungarian Algorithm is not found in the OpenCV library, thus it had to be implemented manually. The Hungarian Algorithm employed in this study is based primarily on Markus Buehren's implementation (See [28]) since it has been heavily optimized.

2.1.3. Software implementation

The image processor fullfils the main objectives of the thesis. The main loop is heavily based in the SORT algorithm, but since the original code was built to be used in Python it had to be rewritten in C++ from scratch.

It functions by analyzing the video frames and extracting the bounding boxes for every vehicle. From this data, the vehicles are tracked and their respective speeds are recorded. Following this, a new video stream is generated which displays the identified vehicles alongside their recorded speed for the user. This must be accomplished regularly and must be resilient to disruptions in the connection due to the drones' continual movement. The top-level architecture can be seen in Figure 2.3.

The software implementation can be viewed from two different perspectives. Firstly, from the perspective of the video frame as it moves through the various components. The image processor must first identify the object, track it, and lastly utilize this data to estimate its position.

Secondly, from the point of view of the tracker itself. A tracker is responsible for monitoring an object on the screen, specifically a car in this thesis. It functions as the code's object and comprises all the essential data to predict the position. For the trackers to work properly the tracker handler is used, an entity that has the overall information and therefore can assign the detections to the appropiate tracker.

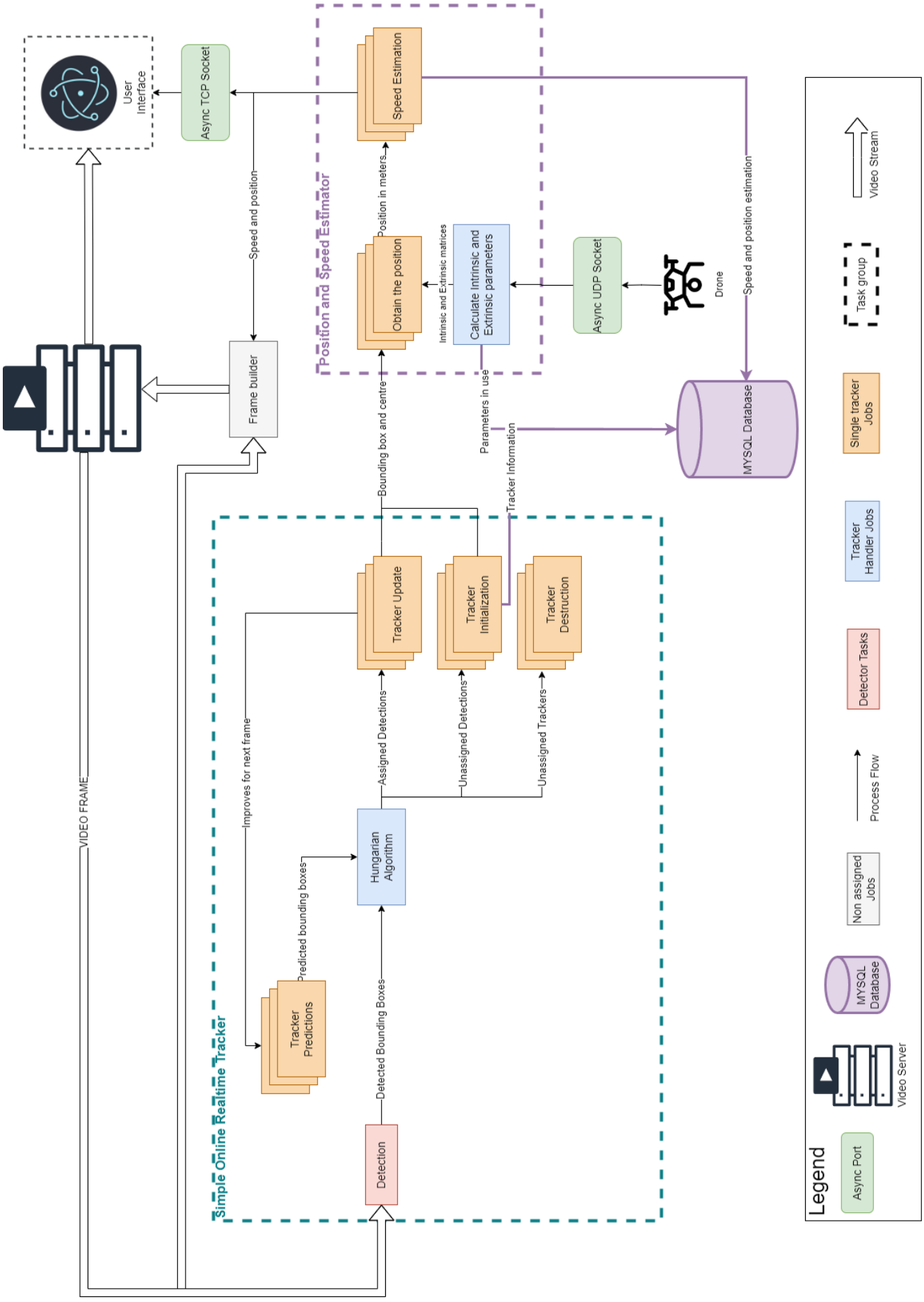


Figure 2.3: Software diagram of the image processor

The Figure 2.3 depicts the dichotomy of various entities, designated by different colors. The image processor's flow is indicated by the general shape and directive arrows, while the boxes' colors and shapes demonstrate the tracker's perspective. Subsequent to the detection process, identified by the red box, the remaining process is initiated. The inclusion of this red box is necessary since it needs to be accomplished before a tracker can be formed, and it provides an abstraction layer. Once the detections are obtained, the tracker handler (blue) can utilize the set of trackers it contains (orange) and predict their next position. The Detected boxes and the predicted ones are then compared and assigned using the Hungarian Algorithm. Following this, the tracker handler is able to update the set of trackers and subsequently update the camera parameters. These parameters are then provided to the trackers, which are responsible for calculating the position and speed. Finally, the Handler can retrieve all positions from the trackers and overlay the data onto the original image to create the frame.

To illustrate these dependencies, another effective tool is utilising the UML (Unified Modeling Language) class diagram. The UML is a set of rules for creating software diagrams. During the planning phase, software development projects apply these diagrams to demonstrate how the programme will be used and how various processes will interconnect. While UML defines multiple types of diagrams, the UML class diagram is the most well-known. It connects the different classes in the planned software, defining the relationships between them. Figure 2.4 provides a simplified version of the class diagram of the code, highlighting the main structure and reducing clutter by depicting only the most important dependencies.

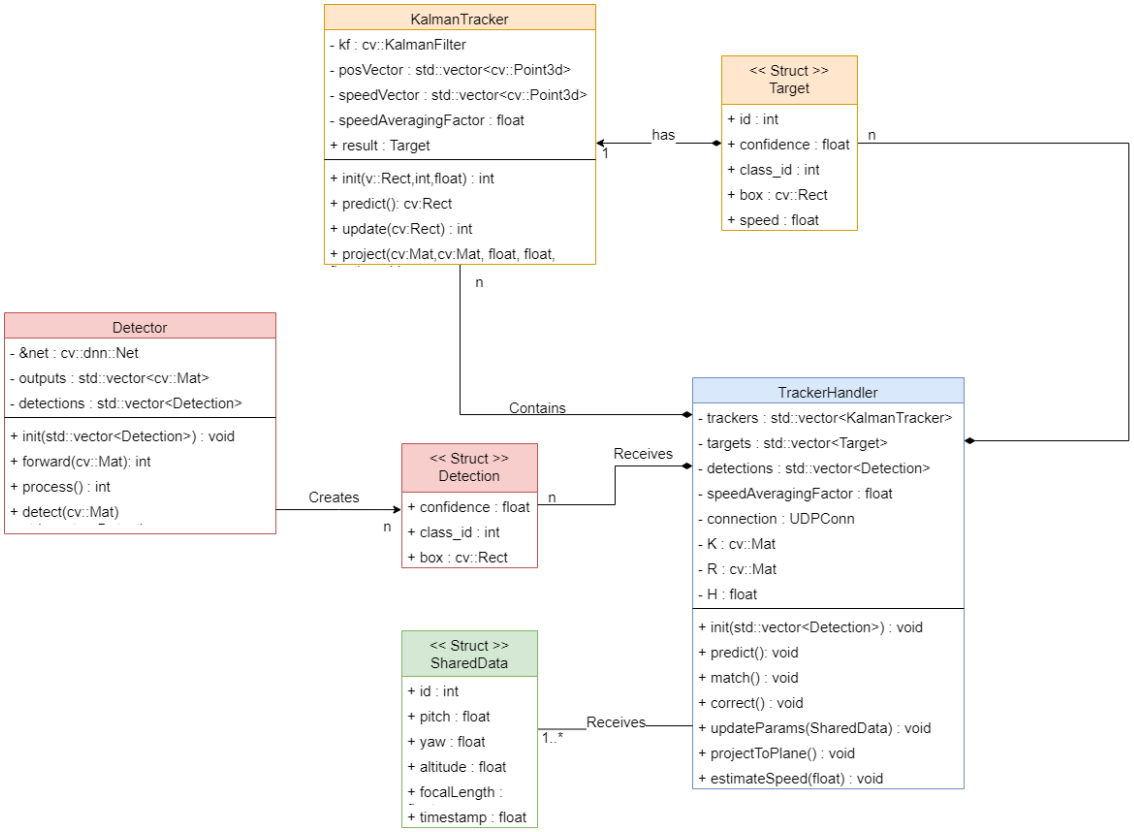


Figure 2.4: UML diagram of the code classes

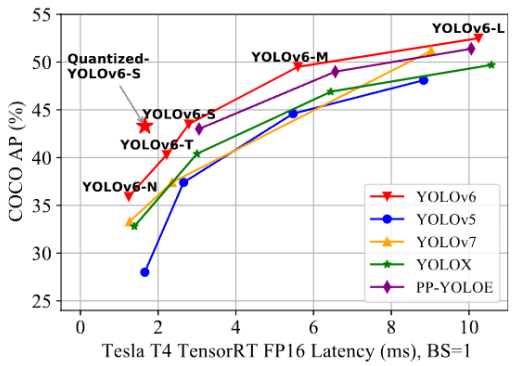
Structs serve as the communication packages between components. The Target struct

encompasses the information managed by the tracker, and it is the final objective of the application. The detection struct provides the specifics on each detection: a cv::Rect containing the bounding box, its confidence and the class id, the category of the object detected. Finally, there is SharedData, a package shared between the image processor and INTA's drones for coordinated communication. The parameters are received via a UDP port (hence the green color) and used by the handler.

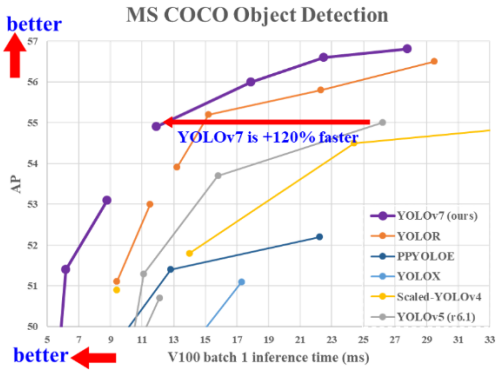
2.1.3.1. Object detection using YOLO

YOLO (You Only Look Once) is a convolutional neural network employed for object detection. within this thesis, YOLOv5, YOLOv6 and YOLOv7 have been used. Each version offers distinct options for input image resolution and neural net dimensions. A trade-off between speed and precision is evident when it comes to neural net sizes. In this particular use case, a neural network of medium size is considered to be the optimal choice. Using higher-resolution images is advantageous for our purposes as it aids in detecting vehicles that are farther from the camera. However, it is computationally expensive and limits the swarm capabilities. Therefore, although better in detection, the use of higher-resolution YOLO models was dismissed.

Due to YOLO versions being non-consecutive, consistent metrics for determining version superiority are difficult to obtain. Rather than relying on a single YOLO neural net, the detector can alternate between networks. The Figures in 2.5 utilize inference time and mAP parameters. Inference time denotes the duration taken to process an image. The mean average precision (mAP) indicates the mean precision for all objects that can be classified. mAP is provided because, depending on the category, the neural network may exhibit bias and yield better overall outcomes.



(a) YOLOv6 graph of mAP and latency from ReadME of [13].



(b) YOLOv7 graph giving a 120% increase in speed. Extracted from ReadME of [14].

Figure 2.5: Inconsistency in the results produced by machines running identical yolo versions.

There are three primary factors contributing to the inconsistency in the results: the non-consecutive nature of YOLO versions, hardware variability, and discrepancies in the neural network software used.

Firstly, each YOLO iteration is not merely a copied version of its predecessor with enhancements. Instead, each iteration modifies the fundamental structure of the neural network,

resulting in variations in layer types or the manner in which information is presented in the output layer. For instance, although YOLOv5 and YOLOv6 deliver a vector containing the bounding boxes and the corresponding confidence levels, YOLOv7 generates an anchor list that specifies the confidence level and different bounding boxes discovered for an object.

The hardware's electronic design is critical when running a neural network on a graphics processing unit. The memory size and chip arrangement can cause an impact on the algorithms used to execute them. Furthermore, various convolutional neural networks can be optimised better than others, resulting in better speeds depending on the layer type or the CUDA version used.

Finally, the performance of the neural network can also be influenced by the software being used. There are countless configurations available for running the neural network. First, a framework must be chosen, such as OpenCV or TensorFlow, each of which may have better optimization for specific neural network structures, which in turn affects the overall outcome. Additionally, the storage method for the neural network's weights and structure can also impact performance. There are various formats, some are exclusive to specific frameworks, and others are meant to be more generic. Even when utilising the same standard, different versions may interpret the neural network and its weights diversely.

The YOLO version provides a basic framework for training, but it does not impose a specific set of weights. In this study, we utilized a pre-trained network from the Microsoft-created COCO (Commmon Objects in Context) dataset, which comprises various image categories, ranging from people and toothbrushes. This dataset serves as a reference to compare precision. Training the neural network might have enhanced the general outcomes of the tool, but it would require amassing a database of drone images of all relevant vehicles to be achievable. This option would represent a considerable undertaking beyond the realms of the project's abilities and scope.

After testing all neural networks, the YOLOv7 standard neural network for images of 640 to 640 pixels yielded the optimal results. It exhibits superior performance detecting cars compared to other competitors while maintaining a high speed application. To determine the optimal version, videos of heavy traffic were used for comparison. These challenging scenarios demonstrated only a marginal variance in performance, but significant enough to consider version 7 as the optimal one.

2.1.3.2. Tracker Implementation

The tracker is designed to track the position of a vehicle and is built using the Kalman Filter. This filter employs a straightforward matrix consisting of 8 state variables, including the centre position in the x and y axes, the height and width of the bounding box, and the corresponding rates of change for these four variables. This is sufficient for monitoring vehicles due to the minuscule relative movement between two frames. Typically, vehicles on the road do not accelerate beyond the system's predictive capacity.

Apart from the state space matrix, the Kalman filter is initialized by three additional parameters: process noise, measurement noise and error covariance. These parameters determine the filter's performance in tracking the object. Process noise arises from model errors; however, its value is small since the model is fairly accurate. Measurement noise is the noise arising from the measurement and is generally higher than the model's noise.

This is due to the variability in detector's measurement between frames, resulting in substantial differences in the size of the box. The error covariance of the system represents the initial state of the covariance matrix. The covariance matrix's high values signify inaccurate initial predictions. This last value (as the initial position) would vary with every prediction, improving over time.

The Kalman filter is applied in pixel coordinates. Following the Hungarian algorithm, the position is updated to enhance the estimation. This updated position is the one considered for the forthcoming steps, as it will possess the smaller error by definition. The camera matrices are utilized to acquire a three-dimensional position from the pixels. The initial setup that necessitates resolving can be expressed mathematically as demonstrated in Equation 2.1.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \left(R \left(\begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} - \begin{bmatrix} Cx_w \\ Cy_w \\ Cz_w \end{bmatrix} \right) \right) \tag{2.1}$$

In Equation 2.1, the positions of the pixel, u and v, are defined alongside the intrinsic matrix of the system, K, the rotation matrix, R, and the world coordinates of the pixel, X_w, Y_w, and Z_w. Additionally, the camera coordinates, Cx_w, Cy_w, and Cz_w, are specified within the frame of reference of the world.

Usually, instead of separating the rotation and displacement matrices, a single matrix is provided. To achieve this, the position of the world frame centre in camera coordinates must be known. Since the camera coordinates move with the constantly moving drone, it is more convenient to reverse this process.

Given the initial state of the system, it is evident that a solution cannot be obtained due to the presence of four unknown variables (s, X_w, Y_w, and Z_w) and only three equations. To solve this, it is assumed that all the objects are situated on a level ground plane. This assumes that for all positions, the Z_w coordinate is equal to zero. Although this does not account for the depth of the image, the calculated position is sufficiently accurate. Please refer to Equation 2.2. As can be observed, it is necessary to invert the matrices to separate X_w and Y_w. This process is carried out directly in the tracker.

$$\begin{bmatrix} X_w \\ Y_w \\ 0 \end{bmatrix} = R^{-1} K^{-1} s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - \begin{bmatrix} Cx_w \\ Cy_w \\ Cz_w \end{bmatrix} \tag{2.2}$$

To solve the system, first the s is determined using the last equation of the system 2.2. Next, the X_w and Y_w are obtained. The tracker handler provides the matrices for the camera's position, rotation, and intrinsic parameters, which are calculated using telemetry data from the drone.

2.1.3.3. Drone telemetry and camera parameters

All camera parameters are extracted from the drone's telemetry data, including position, camera state and attitude. The drone's position is the distance between the world frame and the camera, with only the drone's height being the crucial parameter for defining the

ground plane. Without the height, it is impossible to assume the ground plane and alternative solutions would be required. Having only the height, the system functions correctly when the drone is stationary.

However, when the drone is in motion, its frame of reference changes. To enable the system to function when the drone is moving, the displacement of the x and y axes must also be measured. This can be resolved by using the home point as the origin of the world axes and calculating the distance from the coordinates.

For this project only the height is used, restricting the drone movement. This is because transmitting more data would have complicated the communication between the drone and the server, and INTA required it to be as straightforward as possible for timely implementation and testing. The height is obtained from using the OSDK (Onboard Software Developing Kit) and is measured relative to the take-off location. This may introduce potential errors in areas with significant terrain variation, but obtaining a true altitude using this drone model is not feasible.

From the camera state, only the focal length is required as a parameter. The drones used have a variable focal length as it is tied to the zoom, therefore it is necessary to update it in the intrinsic matrix. The skew and other parameters may be sourced from the camera properties or the specifications published by DJI. Therefore, it is unnecessary to send them each time and can be utilized as initialization parameters.

Focal length refers to the relationship between the focal plane and the image sensor and is typically expressed in millimeters. For our purposes, it is necessary to convert this measurement to pixels. Additionally, most camera sensors are not square and their ratio (a) can be found in the camera's specifications, which must be input as an initial parameter. To obtain pixel measurements, one only needs to consider the ratio between the size of the sensor and the image resolution, which yields the two focal lengths 2.3 and 2.4.

$$f_x = \frac{f[mm]}{sensorWidth[mm]} frameWidth[px] \tag{2.3}$$

$$f_y = \frac{f[mm]}{sensorheight[mm]} frameHeight[px] = f_x a \tag{2.4}$$

The camera attitude refers to its roll, pitch, and yaw. Drones often feature a gimbal-mounted camera, so the camera's attitude differs from that of the drone. The angles are commonly given in the NED frame of reference. Based on the conventional aircraft, the North East Down frame defines the angles that position the X axis towards the head of the aircraft, the Y axis towards the right wing, and the Z axis downwards. When considering the camera, the X axis points towards the front, the Y to the right and the Z to the bottom. This is used as the standard frame of reference for the ground as well, so the rotation matrix used can be directly derived from the angles. It can be calculated using the matrix 2.5 where $c(x)$ stands for to $cos(x)$ and $s(x)$ stands for $sin(x)$.

$$R = \begin{bmatrix} c(\theta)c(\phi) & s(\theta)s(\psi)c(\phi) - c(\theta)s(\phi) & c(\theta)s(\psi)c(\phi) + s(\theta)s(\phi) \\ c(\theta)s(\phi) & s(\theta)s(\psi)s(\phi) + c(\theta)c(\phi) & c(\theta)s(\psi)s(\phi) - s(\theta)c(\phi) \\ -s(\psi) & s(\theta)c(\psi) & c(\theta)c(\psi) \end{bmatrix} \tag{2.5}$$

This rotation converts the points from the world to the camera coordinates. However, it is

still not the correct frame of reference. The images have the X axis horizontally oriented to the image and the Y axis pointing downward. This standard is set by OpenCV when referring to pixel coordinates (see Figure 2.6).

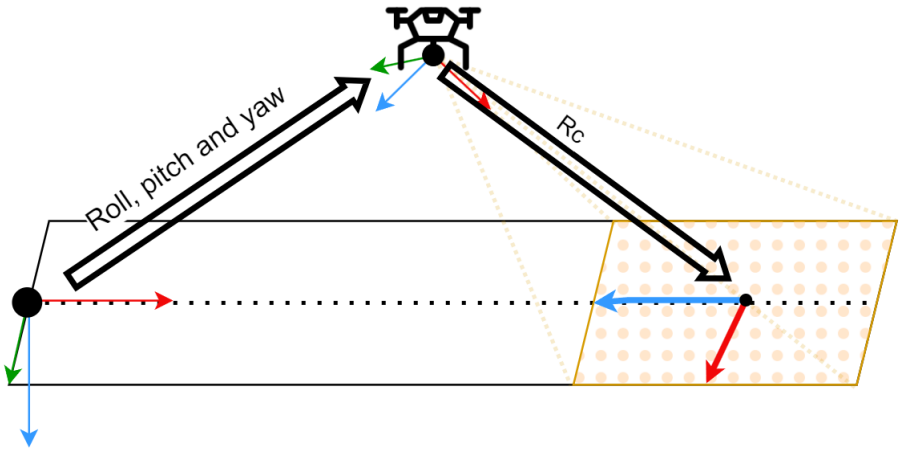


Figure 2.6: Relationship between frames of reference

To rectify the axis mismatch, a second rotation is added to convert from NED to the image system of reference. This relationship remains constant, as the drone camera and the image move synchronously. The resulting rotations consist of a 90 degree rotation on the Z axis followed by another 90 degree rotation on the X axis.

2.1.3.4. Video streaming

For the application to function effectively, the streaming system must be dependable, speedy, and user-friendly. The streaming server employed is an open-source solution capable of seamlessly handling multiple protocols (see [26] for mfurther details). Streaming all types of content has been resolved numerous times utilizing various protocols, each with its own advantages and drawbacks. By operating the streaming service on the EC2 instance, it is feasible to use them interchangeably, as this only modifies communication with the server. One of its top features is its ability to receive input in one protocol and produce output in another without causing significant delays.

Having this option enables the use of the most suitable streaming protocol for each task. Three primary protocols were used: RTMP (Real-Time Messaging Protocol), RTSP (Real Time Streaming Protocol), and HLS (HTTP Live Streaming).

a Real Time Messaging Protocol

The Adobe-owned communication protocol, RTMP, is frequently used for streaming in Flash applications. Despite being outdated compared to newer protocols, it is still widely supported and functions on most machines. Typically, it is used with TCP, aiming to maintain low-latency connections while transmitting high volumes of data, including audio and video. Due to its foundation on TCP, it can achieve high speeds.

This protocol is widely utilized for streaming from the source to a central server, but is no longer compatible with modern applications. Therefore, end-users must use either RTSP

or HLS depending on their platform. In this project, drones utilize RTMP to transmit the streaming to the server due to its simplicity and flexibility.

b Real Time Streaming Protocol

RTSP is an application-level protocol for controlling video streams. It differs from other protocols as it is not a streaming protocol in itself. Instead, it uses RTP (Real Time Protocol) and RTCP (Real Time Control Protocol) to transmit the video streams. RTSP's primary objective is to offer users options such as play, pause, and record for the receiver. The stack of RTSP with RTP and RTCP is typically referred as RTSP. It was designed with HTTP compatibility in mind, allowing it to function similarly to websites. As a result, RTSP is normally compatible with any HTTP network, which facilitated its initial rapid expansion.

It is primarily utilized for CCTV (Closed-circuit television) cameras and occasionally for live broadcast sources, although its use has been reduced. In the context of this project, it functions mainly as the VLC player's receiver and for video testing due to its user-friendliness and modernity compared to RTMP, while still yielding swift results.

c HTTP Live Streaming

While RTMP and RTSP are useful for sending video from the source to the server (also known as ingest), the primary goal of HLS is to deliver the video feed to the end user. HLS is based on HTTP making it fully compatible with contemporary browsers. It is the most used protocol for streaming due to its adaptive bitrate, reliability and broad support. However, latency remains its most significant drawback. By itself, HLS is too slow to meet the project's objectives, as it can have a latency of between 5 and 20 seconds. Fortunately, for the project, HLS has a low latency version that the server supports, so this issue can be resolved.

In the project, HLS is used as the end-user protocol, receiving the video with the bounding boxes and speeds imprinted on it. It could be replaced by webRTC, the latest protocol, however, due to the project's emphasis on compatibility, HLS was deemed more suitable. A Summary of various streaming protocols and their applications are presented in Table 2.1.

Protocol	Latency	Compatibility	Use Case
RTMP	$<$2 seconds	streaming servers only	Stream Ingestion
RTSP	0.5-2 seconds	streaming servers	Stream Ingestion
HLS (low latency)	2-8 seconds	web browsers	Stream Playing

Table 2.1: Streaming protocols summary

2.1.3.5. Telemetry management of the image processor

While the video is received through the streaming service, telemetry was acquired using a different method. A byte-level package was used to send the data via a UDP (User Datagram Protocol) socket for the telemetry. This approach is employed by INTA for their

internal communications, and it is the most straightforward method. The only necessary components were a C++ struct and a UDP endpoint to receive the data.

The C++ struct is a variable type that enables bundling variable types together and creating a larger object. It is possible to work at the byte level and use this struct as a data package directly. This implies that there will be no overhead to the package other than the UDP information. UDP is a protocol that allows sending messages through the internet without the need of a connection. The emitter sends the package with an IP and a port, and if the receiver is listening it might obtain it. There is no guarantee against potential package losses. However, since the telemetry is transmitted once per second, the loss is not considered critical.

Using C++ structs has one significant limitation, compatibility. The method relies on the way the compiler organises data. An OS (Operating System) , the space allocated to a particular variable differs depending on whether it is 64 or 32 bits. Additionally, depending on the compiler and C++ version running in the OS, there may also be variations in storage of values. This approach is exclusively applicable when the emitter and receiver have similar properties and utilize the same standards, thereby ensuring compatibility. To extend the application of the tool to other systems, an alternative approach would be required.

While the information about the vehicle can be read in the videos of the cars, it could be overwhelming if there are too many vehicles on the road. To address this issue, the data of the vehicle is also transmitted in a readable format to the user. The end-user system is not programmed using C++ and usually will use a dynamic IP, thus requiring an alternative solution. The approach employed involves generating a TCP server that awaits user connections and transmits data via a JSON (JavaScript Object Notation) packet.

TCP is a low-level protocol that resembles UDP but is connection-oriented. This indicates that, prior to sending any data, the server awaits connection requests and then establishes a reliable tunnel for communication. This enables the server to connect to users in any network, no matter the type. The server's IP is fixed and known, making it simple to establish a connection with it from the user application.

The messages are encoded using the standard JSON (JavaScript Object Notation). JSON is a file standard designed to store pairs of keys and values in an organized, easily serializable manner. It originated from the way the programming language JavaScript handles its objects and their properties. This is a simple yet powerful approach to transmitting objects in strings. All languages possess numerous libraries that are capable of producing, modifying, and interpreting JSONs, rendering it very useful for cross-communication among applications built in different programming languages. Furthermore, it is a readable standard, which means the data is not encoded in binary and can be read without the need for decoding. The main structure of a JSON is straightforward. All the data is contained within $\{\}$ and is assigned a corresponding tag. Refer to figure 2.7 for a JSON example.

The UDP socket has to constantly listen for incoming messages without interrupting the running code. Otherwise, receiving messages would slow down processing to only one frame per second, since telemetry comes in at 1Hz. To avoid this, the port is executed on a parallel thread using an asynchronous socket.

Since the timing of package deliveries on the thread is uncertain, there is a risk of an error occurring when the UDP socket attempts to write to a variable at the same time as the tracker handler tries to read it. To prevent this issue, the memory space is blocked during

```
{
  "input":"http://...",
  "status": 1,
  "cars":
  [
      {
          "id":3,
          "speed": 22,
          "color": "1212FF"
      },
      {
          "id":4,
          "speed": 0.002,
          "color": "123421"
      },
      {
          "id":5,
          "speed": 40,
          "color": "24B4AF"
      }
  ]
}
```

Figure 2.7: Example of a JSON file with the car information

handling to ensure secure usage. This may cause a slight delay in the main process, however, it is negligible. The TCP should be asynchronous too to accommodate users appearing at random intervals, or possibly not at all. To establish user connections with the port, it is necessary for the port to continuously listen and operate independently from the main thread.

2.1.3.6. Initialization parameters of the image processor

The image processor possesses numerous adjustable parameters. Although it is feasible to configure the variables directly in code (also known as hardcoding), this approach is not advisable. Hardcoding forces recompiling for every test during tool parameter optimization, which is time-intensive and reduces its portability, since a compiled version of the code can be used regardless of the machine's configuration.

To enable the full parameterisation of the system, the application is loaded with a YAML (YAML Ain't Markup Language) file. The YAML format is a human-readable data serialisation language used for configuring files. Its main advantage over other formats is its simplicity; a YAML file has no overhead and can directly present a set of variables in order, while also having options such as lists, arrays or other YAMLs. It is possible to effortlessly code strings, scalars, and lists by making use of key-value pairs in a comparable manner to JSON. This entails assigning a variable to a keyword. Although the formatting differs, the objective remains identical. Figure 2.8 displays a simple YAML file.

```
"output":"http://...",
"framesPerSecond": 15,
"InitializationMatrix":[[3,0,0],[0,3,0],[0,0,1]],
"initialPackage": {
  "input":"http://...",
  "status": 1,
  "cars":[]
}
```

Figure 2.8: Example of a YAML file

2.1.3.7. Database connection

The database stores the positions and the trackers in a user-friendly format. Initially, a less elaborate method was to be used, but due to the nature of the data, a relational database was found to be more convenient. There are two tables, one for trackers and another for positions. The tracker table includes the flight parameters (roll, pitch, yaw, height). The positions table contains the position saved in pixels and real-world coordinates, along with the speed calculated at that moment. To reflect the sequential order of the positions, a counter is included in the table. Each position is assigned a unique identifier by the tracker. A C library is available to assist with executing queries and extracting the results. The database was utilized during the testing stage to evaluate the accuracy of the tool within the simulation environment.

2.2. User Interface

The image processor handles most of the task for the tool; nonetheless, it cannot independently display the resulting video stream, and a receiver had to be build. Initially a single UI (User Interface) was going to be used, a website displaying the HLS video stream (see Figure 2.9). While the website is a practical and straightforward means of presenting information, it can be challenging to discern the speed of each vehicle when there is a substantial number of cars. Furthermore, there is no information provided about the state of the stream or potential errors.

These uncertainties led to the creation of an alternative application and the video feed was kept as a secondary option for the user, requiring no installation. The UI contains the essential data to display the video stream, machine state updates, and facilitates switching between drones with a single click.

2.2.1. Technologies used

The application is developed with Electron [29] and Vue [30]. Electron is a Javascript-based framework for desktop application development. Due to the progress of web development, web browsers are now able to fulfil the functions of desktop applications. Electron embeds a small web browser and backend server in a local application, providing a complete website environment within the client's computer. This enables the creation of

Figure 2.9: Website application in use

compiled executables that are fully compatible with website development.

Electron applications have become increasingly popular, particularly for web-only apps seeking to export content to desktop applications. This enables the development of an application directly from the website code, providing a consistent look-and-feel for the user. Little modification is required to shift a website to an Electron App and vice versa.

Vue is a JavaScript framework for creating dynamic websites. Unlike static websites based on HTML, Vue allows websites to be constantly updated depending on the data they receive. Its structure is highly optimized to create dynamic pages that flow seamlessly. They are built from simple blocks that can be moved, altered and edited to fulfill any objective.

Although this combination is excellent for building stunning applications, it lacks optimization. Web browsers are very resource-intensive applications, and Electron relies on them to display information. Consequently, apps built with these technologies use more temporary memory and CPU cycles than is strictly necessary. Whilst this is not usually problematic for modern computers, it could present difficulties for older or less powerful ones.

2.2.2. Development

The user application had two distinct roles: to comfortably display one drone and to simultaneously show multiple drones. Consequently, two separate views were created.

Upon opening the application, the user is presented with a mostly blank screen, with the options located in the top bar. These options include changing to an all-channels mode, selecting the output (i.e. the drone output stream to connect to), displaying information about the machine and app state, and connecting to the server. A colored icon shows the connection status, either red or green. The server connection is made via the TCP port which, when open, will establish the connection and begin to receive packages. The data is provided in JSON format and includes the server status and detected vehicle properties.

If the application cannot connect to the server, a message will notify the user. Additionally, if a connection is established but there is no ongoing stream, the user will also be informed.

Once a connection is established, and a drone is sending data to the image processor, the display will switch to Figure 2.10.

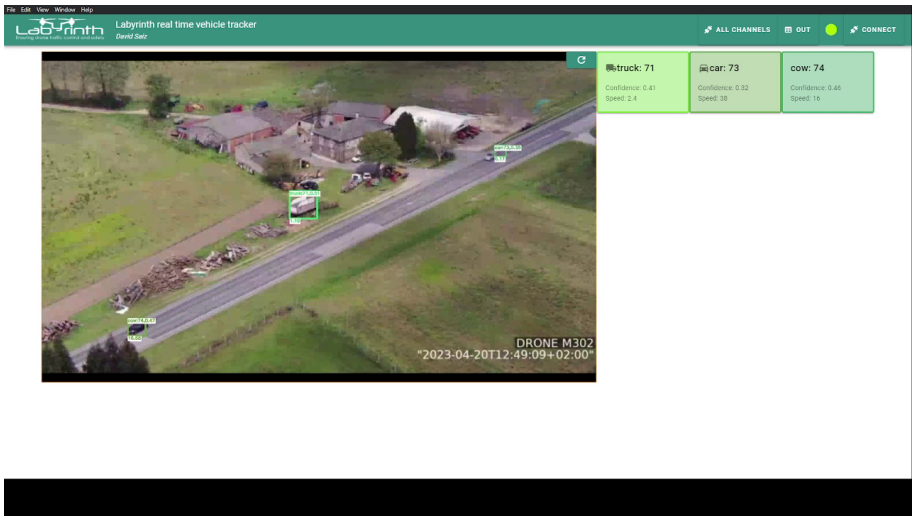


Figure 2.10: Application main screen with stream

Two additional items are visible on the screen: a video visualiser and a set of boxes containing data about the vehicles on the road. The app features a visualisation of an HLS stream with a latency of 1 to 2 seconds, which allows for quick reaction in the event of detecting an infraction, although the video feed cannot be employed as the operational screen for the drone camera. Each box contains the id, the confidence, the speed and the color assigned. The color serves as a helpful visual aid for quickly matching the boxes with the vehicles on the road. The containers are regularly updated according to a user-defined quantity of frames. With the accompanying icons, cars, trucks, bicycles, and motorbikes can be easily distinguished. Although the image processor is capable of detecting other objects, they do not have personalized icons since they are unintended or erroneos detections. Figure 2.7 displays an example of a package.

This view is advantageous for focusing exclusively on a single drone, but it is incapable of displaying multiple drones simultaneously. To view all channels simultaneously, select the ALL CHANNELS option which will modify the display. This will provide a view similar to Figure 2.11, which features four video feeds from various drones. It's worth noting that not all the feeds are typically in operation, and any streams that are unavailable will be represented by a blank screen. The reload icon in the video's corner enables the visualizer to restart when a drone is added. This allows the window to be opened before all drones are operational.

2.3. Drone Setup

For the LABYRINTH project, the drones were operated predominantly by INTA. INTA used two Matrice 300 RTK from DJI, a Chinese company specialising in creating drones for personal and professional use. The Matrice 300 RTK is a high-end drone, which, according to DJI's website [31], has a top flight time of 55 minutes and can operate with up to 15 km of line-of-sight range. Not only that but it contains a multitude of built-in sensors for impact de-

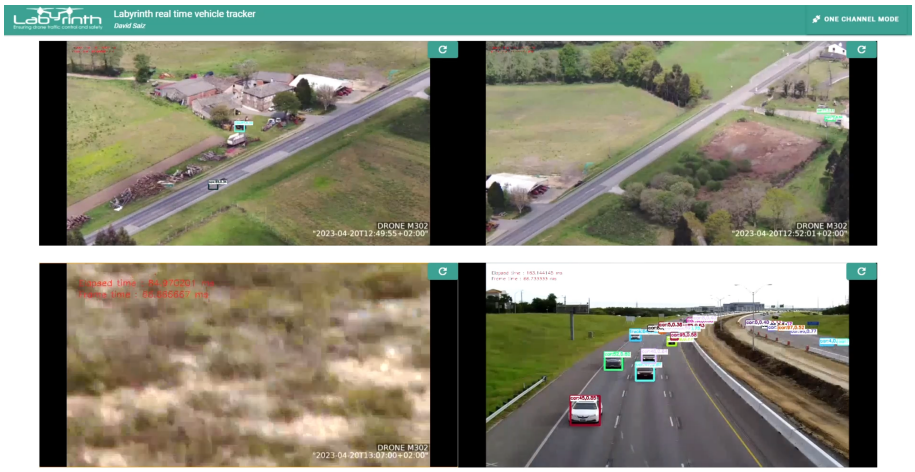


Figure 2.11: Application multiple streams view. The streams have been simulated

tection, battery health regulation, and autonomous flight capabilities, although the drone's functionalities rely on the controller. To address this problem, an onboard computer was used called Manifold V2 [32]. This system directly connects to the drone, enabling it to receive various commands and transmit telemetry data, effectively transforming the drone into a UAS (Unmanned Aircraft System). To ensure continuous communication with the drone, a partner of the project developed a smart link switcher. This system switches between a radio link and 4G depending on the drone's situation, ensuring an uninterrupted connection. The onboard computer runs C++ code that manages all operations, including camera zoom and video streaming.

The Matrice 300 RTK features a camera integrated into its frame, but it remains fixed towards the front. As a result, a payload camera is utilized instead. The team incorporated the Zenmuse H20T model for this particular project, which consists of a 20-megapixel zoom camera, a wide-angle camera, a thermal camera, and a laser rangefinder. The laser rangefinder provides the distance of a selected point via the accompanying software, but it cannot be utilized from the code or perform a full scan, rendering it unsuitable as a LiDAR substitute. The camera best suited for the project is the Zoom camera, capable of a 55x zoom utilizing a blend of optical and electronic zoom. The images captured are clear despite the drone's distance from the road, thereby enhancing the manoeuvrability of the UAS, which is crucial for expansive coverage, circumventing obstructed views and adhering to the no-fly-zones rule.

The employment of a high-capacity drone equipped with a camera yields a system capable of conducting surveillance in designated areas autonomously and securely. The drone's test flights typically lasted approximately 40 minutes, and with two drones available, it was attainable to keep them both in rotation, ensuring permanent visual coverage of the objective region.

CHAPTER 3. EXPERIMENTAL EVALUATION

The flights for the LABYRINTH project were conducted immediately after completing the tool. However, arriving at conclusions on the precision of the tool was challenging due to the short duration of the flight tests and the unknown speed of the cars. Consequently, the collected data could not be used for characterization. An alternative method to obtain reliable results had to be contemplated.

Instead of using actual cars, modern simulation technologies make it possible to generate virtual cars that offer just as high-quality an image as a real vehicle. This approach enables the creation of vehicles with a set, consistent speed and capture them from any desired camera angle, simulating the state of the drone.

3.1. Test setup and data collection

The setup consisted of a simulator, a database and a set of python scripts for data extraction. The simulator setup was built using Airsim, a Microsoft-designed AI research simulation platform with the goal of serving a physics-based simulation setting for piloting drones. The platform's API empowers direct object creation and manipulation via code within the simulation. For the purposes of the testing, although it is feasible to generate drones and the system can simulate cameras for stereo vision or a LiDAR, a basic camera suffices. References to pitch, yaw or altitude are properties that relate the camera in and the world frame of reference, and are independent of the drone, which remains static.

The most significant limitation relates to video recording. In order to acquire a video, every frame has to be extracted individually, so the number of frames per second obtained is low. Although a slower frame rate is not problematic for the thesis, it diminishes the information within the system. Additionally, there is a lack of consistency in the frame rate, making it more difficult to estimate the velocity accurately.

With Airsim, several videos were recorded under controlled conditions. Each video features a single car moving along a straight path on the x-axis. By comparing the expected trajectory of the car to the results obtained from the image processor, its properties can be characterized. Additionally, it is noteworthy to examine how the image processor handles multiple lanes of cars simultaneously. When considering roads, it is common for them to have at least two lanes, one in each direction. It is important to take into account the fact that the cars in the image may be far from the center, where errors are likely to be greater. To address this issue, the concept of a car lane is introduced to simulate highway traffic and control all road lanes simultaneously. Lanes are numbered from 1 to 9, with the first being furthest from the camera and the ninth being closest. Figure 3.1 illustrates the vehicles in each lane and demonstrates the variety of perspectives from a single point.

The closer the object is to the corners of the image, the more extreme the projection applied, resulting in a larger error due to imperfections. If no lane is specified, lane 5 is used. The tests are prepared with the assumption that lane 5 is the middle lane.

The variables chosen for testing are pitch, yaw, and height. These variables were selected because they are those sent by the drone, which implies that they can be optimised by moving it; roll could also be part of the test, but it was not taken into account because gimbals usually self-correct to keep the camera horizontal, which means that it is never

Figure 3.1: Cars in lanes from 1 to 9 in parallel

used in real cases. Furthermore, roll is unnecessary for steering the camera plane, as it results in rotation around its normal axis without altering its orientation.

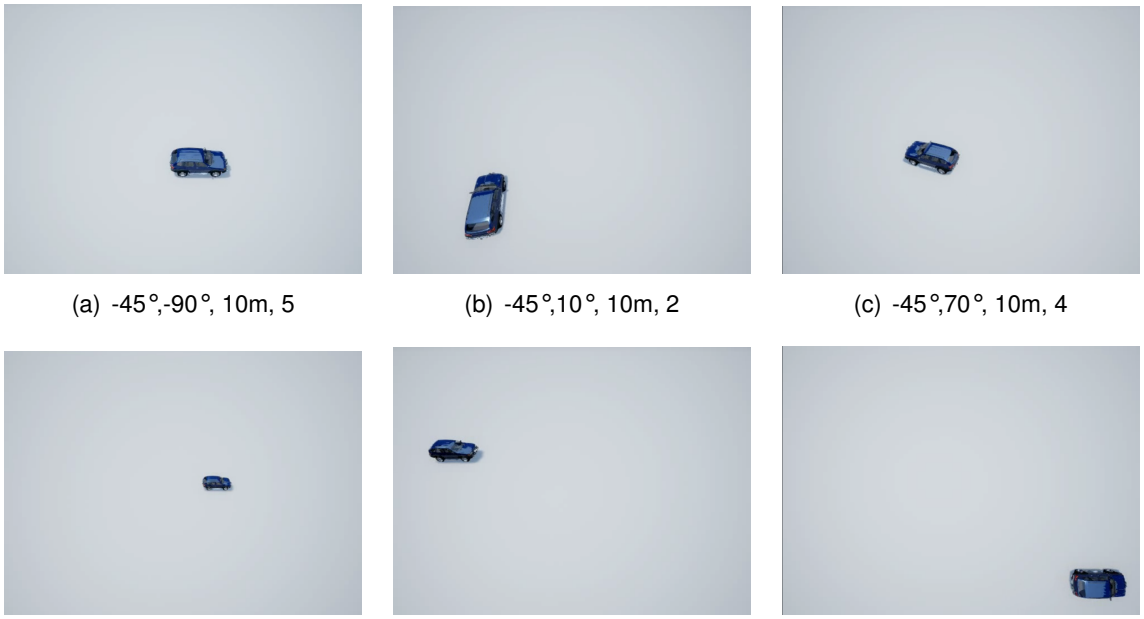
The videos produced possess four defining properties: the pitch, yaw, and height employed, along with the lane of the automobile's trajectory. Although not an official test parameter, this fourth element is crucial in determining the conditions under which the data was recorded. Figure 3.2 shows a series of videos showing the video creation process. Each video is played for one minute. Ensuring that each video has the same duration enables an accurate comparison of recorded data. The detector's effectiveness for a particular angle decreases as it detects fewer trackers, indicating a shorter detection time for the object.

Variable	Definition
Pitch	Tilt angle of the camera
Yaw	Pan angle of the camera
Height	Height of the camera
Car lane	Road lane the car is driving

Table 3.1: Definition of the tests variables

Table 3.1 presents a summary of the parameters used. To examine the detection and classification rates of the tool, the class id for each detection is recorded. A class id refers to a number assigned to a category in the YOLO detector. As detections are linked to a specific position, the class ids are stored in the positions database. This allows class switches to be monitored for a single tracked object.

When using the tool, inaccurate readings may occur during the initial and final stages of vehicle detection. Various factors can contribute to this, but common reasons include the object being detected intermittently, resulting in the Kalman filter failing to accurately predict, or the object being detected before fully appearing in frame, causing sudden forward or backward movement of its center. Both these situations are alleviated in real-time by the windowed mean and, in comparison to the tracking time, the number of impacted frames is typically low. However, with recorded data, these points can influence the resulting mean



(a) -45°,-90°, 10m, 5 (b) -45°,10°, 10m, 2 (c) -45°,70°, 10m, 4

(d) -45°,-90°, 18m, 5 (e) -50°,-90°, 10m, 3 (f) -85°,-90°, 10m, 9

Figure 3.2: AirSim video feed examples. The subtitles give the configuration in order: pitch, yaw, height and car lane

error. Once the videos were run through the image processor, these outliers were removed from the database.

Two separate sets of experiments were conducted. The first aimed to investigate the impact that camera attitude and position have on estimation. The second aimed to evaluate the impact of receiving inaccurate data. The objective of the two tests was to initially characterise optimal tool conditions and subsequently evaluate system performance when an error is encountered.The following sections describe each of the tests and their results in more detail.

3.2. Attitude and position results

To examine the camera's attitude and height, three different tests were performed with the simulated drone camera static. Each test investigated the impact of one of the variables. In order to do so, the selected variable was varied while the other two were kept constant, so that the effects could be compared. For each specific test, numerous videos were recorded, each featuring distinct angles, heights, and car lane configurations.

3.2.1. Pitch effects

The camera's pitch is arguably the most important parameter in the system. It accounts for most of the mean position error. To eliminate any influence from the height or yaw, all tests were conducted with a -90 degrees yaw and varying height to maintain continuous visibility of the cars. The use of a -90 degrees yaw is advantageous as it allows the car in the image to move horizontally, resembling the actual movement of a car. Figure 3.3

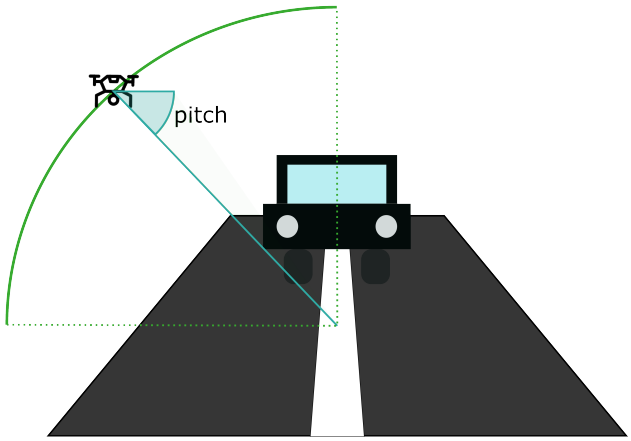


Figure 3.3: Pitch test setup

illustrates a simplified view of the arc where the test is conducted. The recordings are captured from positions within the arc, with intervals of 5 degrees in pitch.

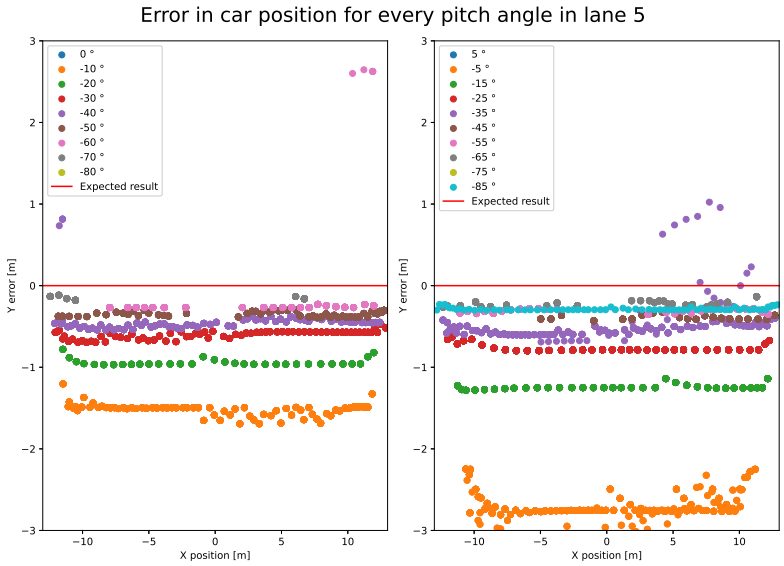


Figure 3.4: Effect of the pitch in the routes

Figure 3.4 illustrates the general test results for the position. It displays the error deviation from the expected trajectory for all examined angles. The observed trend is in line with the expected: the closer the angle is to -90 degrees, the smaller the average error. This is due to the presupposition of a flat world. In order to rectify this error source completely, it is necessary to know the size and height of the car, but since the tool is intended for use without any prior knowledge of the vehicle. It's worth noting that the error grows non-linearly. The difference in error between -5 and -10 degrees is much greater than between -85 and -80 degrees, as it is proportional to the tangent of the angle. The steeper the angle, the greater the tangent and therefore the error. The graph also illustrates that the variance for these measurements is small in the y-axis.

Tables 3.2 and 3.3 provide the mean error in the pitch value. The system is found to be in-

lane	**0°**	**-5°**	**-10°**	**-15°**	**-20°**	**-25°**	**-30°**	**-35°**	**-40°**	**-45°**
1	-8.92	-0.44	2.65	-	3.49	3.30	3.44	3.85	-	3.59
2	-7.66	-0.50	0.92	1.50	1.89	1.75	2.55	2.90	1.94	2.25
3	-	-1.96	-0.38	0.32	0.33	0.76	0.90	1.06	1.21	1.19
4	-9.27	-	-0.95	-0.44	-0.21	-0.08	0.04	0.17	0.04	0.25
5	-	-2.74	-1.52	-1.37	-0.94	-0.92	-0.67	-0.52	-0.52	-0.39
6	-5.46	-2.62	-1.82	-1.53	-1.51	-1.18	-1.07	-1.01	-1.07	-0.92
7	-	-2.68	-2.03	-1.84	-1.51	-1.43	-1.34	-1.49	-1.27	-1.26
8	-	-2.62	-	-1.71	-1.55	-1.63	-1.49	-1.57	-1.66	-1.47
9	-	-2.20	-1.70	-1.56	-1.47	-1.43	-1.41	-1.32	-1.30	-1.29

Table 3.2: Table of the position mean error in m for pitch angles from 0° to -45° and lanes

lane	**-50°**	**-55°**	**-60°**	**-65°**	**-70°**	**-75°**	**-80°**	**-85°**
1	3.46	3.35	3.27	3.09	3.02	2.87	2.70	2.60
2	2.17	2.15	2.14	2.06	2.05	1.99	1.97	3.65
3	1.15	1.21	1.19	1.25	1.30	1.29	-	-
4	0.31	0.38	0.65	0.43	0.46	-	-	-
5	-0.35	-0.31	-0.18	-0.17	0.11	-	-	0.04
6	-0.83	-0.84	-0.79	-0.77	-	-	-	-
7	-1.24	-1.18	-1.19	-	-	-	-	-
8	-1.50	-1.51	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-

Table 3.3: Table of the position mean error in m for pitch angles from -50° to -85° and lanes

effective when the angles and lanes have an excessively steep angle of vision. Predictably, larger errors occur in extreme car lanes. It is worth noting that the optimum lane for vision is not necessarily the middle lane and can change depending on the angle of the pitch. This is due to the fact that the optimal lane allows for the steepest angle of vision while maintaining the ability to detect it.

While the resulting position provides an advantageous view of the error, the parameter that requires calculation is the speed of the vehicle. Mean error has no impact on speed, but inaccuracies still exist. The two primary sources of error are detector shift or missing detection and variable frame rate. If information is available regarding frame rate shift, it would be easier to mitigate the effect; however, this is not applicable. The most effective means of presenting the findings would be through tables 3.4 and 3.5, which display the average error.

One conclusion that can be drawn is that the greater the lane's degree of curvature, the less accurate the speed estimation. This is because speed is calculated based on two consecutive positions, and the error is proportionate to the distance the vehicle has traveled between the two frames. When the vehicle is farther from the frame, the distance the vehicle covers will appear shorter, leading to an overestimation of speed. Conversely, closer lanes have a reverse effect. If the car is closer to the camera, the distance between points will appear longer and the expected speed will underestimate the actual speed. Both of these effects are visible in Figure 3.5. In the diagram, all the lines cover the same

lane	**0°**	**-5°**	**-10°**	**-15°**	**-20°**	**-25°**	**-30°**	**-35°**	**-40°**	**-45°**
1	221.41	-1.09	-0.49	-	0.59	0.04	0.90	0.77	-	-0.03
2	111.35	-1.52	-0.93	-0.35	-0.69	0.31	0.73	0.08	1.06	0.25
3	11.00	-2.59	-1.17	-1.33	-0.06	-0.70	-0.93	-0.47	1.24	-0.97
4	44.11	-	-2.01	-1.57	-1.97	-0.75	-1.90	-1.67	0.28	-1.00
5	11.00	-3.26	-2.93	-2.13	-1.75	-1.68	-1.78	-1.90	-0.31	-0.35
6	113.11	-3.89	-2.76	-2.08	-1.98	-2.36	-2.64	-2.45	-0.73	-0.49
7	11.00	-3.48	-3.18	-2.83	-2.21	-2.86	-2.41	-1.78	-1.93	-2.45
8	11.00	-4.00	-	-3.66	-2.93	-2.72	-3.38	-2.81	-1.21	-2.57
9	11.00	-3.66	-3.15	-1.45	-2.32	-1.33	-2.13	-2.23	-7.34	1.42

Table 3.4: Table of the speed mean error in m/s for pitch angles from 0° to -45° and lanes

lane	**-50°**	**-55°**	**-60°**	**-65°**	**-70°**	**-75°**	**-80°**	**-85°**
1	1.25	4.85	2.19	-1.04	4.94	-3.60	-1.71	-1.60
2	-1.25	0.06	-0.74	-0.36	-1.75	-3.00	-4.63	-3.13
3	-1.02	-0.20	1.56	-1.11	-2.91	-15.45	-	-
4	-1.26	2.48	0.51	-3.98	-3.93	-	-	-
5	-0.98	-1.40	-2.25	-3.47	-5.35	-	-	-2.15
6	-2.70	-2.19	-7.73	-6.78	-	-	-	-
7	-1.58	-5.36	-7.26	-	-	-	-	-
8	-4.84	-7.15	-	-	-	-	-	-
9	-	-	-	-	-	-	-	-

Table 3.5: Table of the speed mean error in m/s for pitch angles from -50° to -85° and lanes

distance in the real world, but due to the perspective, their lengths are dissimilar. This demonstrates that although advances should be similar when the plane moves, they are not directly proportional, causing errors.

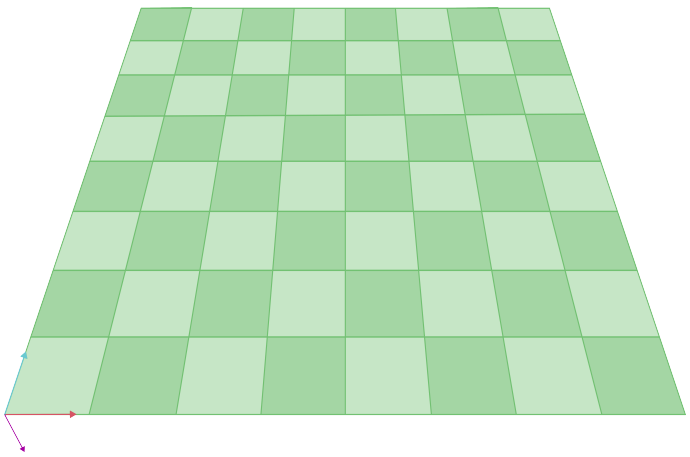


Figure 3.5: Checkboard showing equivalent distances when applying an angle

The optimal results are achieved with an angle of -45 degrees. This angle strikes a balance between accuracy and feasibility, as beyond this point the results start to deteriorate. It is

crucial to note that while better results can be obtained with higher angles of vision, the detection rate must be considered when selecting the angle. Steeper angles beyond -45 degrees have a lower success rate in accurately classifying the category, making -45 degrees more effective for general purposes. This is further explored in section 3.4..

3.2.2. Yaw effects

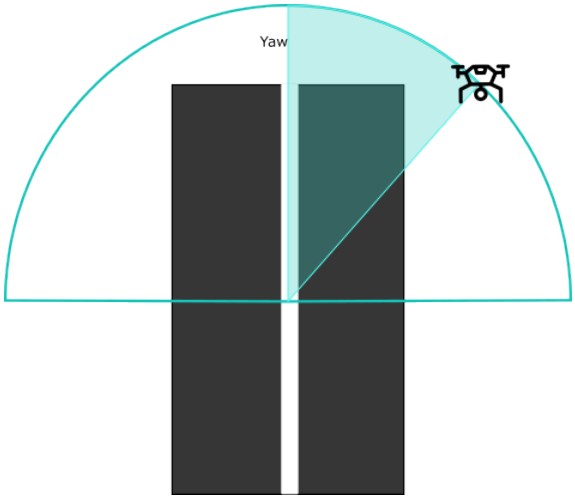


Figure 3.6: yaw testing setup

When considering yaw, one can assume that the effect is similar to that for pitch, but instead of a constant error, there should be more variance. To ensure isolation of the yaw, the chosen parameters for the test were -45 degrees of pitch and a height of 10 metres. The height remains constant throughout the process while rotation happens in the X-Y plane around the central position, as illustrated in Figure 3.6. While -45 degrees may cause a slight error, it can be isolated. The normalization is carried out, considering the distance between the camera and the vehicle across the x and y axes, ensuring that there is no distortion among the routes, allowing for a more straightforward analysis.

Figure 3.7 distinguishes between positive and negative angles to enhance visualization. From the results, several conclusions can be drawn. Firstly, all trajectories exhibit a mean error of 0.5 when undergoing -45 degree pitch. Secondly, routes shift diagonally as the angle changes. It is remarkable that the lines disappear entirely within a viewing angle of 40 to -40 degrees. In instances where the vehicle is not perceptible from a sideways viewpoint, the detector encounters complications in accurately recognising them as automobiles. If all positions regardless of the detector classification were plotted on the same graph, the trajectories would be visible.

When examining average error, there are no significant effects, thus the focus shifts to the analysis of variance. Tables 3.6 and 3.7 illustrate the corresponding variance and lanes matrices.

Interesting results can be seen from the variance tables. Initially, when the yaw is at 90 or -90 degrees, the variance is apparently 0, which is consistent with the plotted data. Nevertheless, certain values deviate from this trend, specifically the value for lane 6 and yaw -90 degrees. Despite this, including these findings would emphasise that erroneous

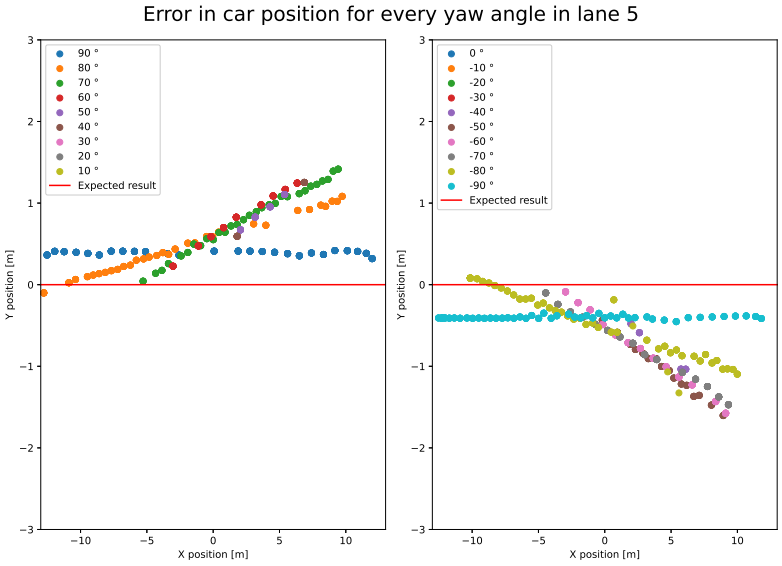


Figure 3.7: Effect of the yaw in the projected path

lane	**90°**	**80°**	**70°**	**60°**	**50°**	**40°**	**30°**	**20°**	**10°**
1	0.00	2.46	0.01	-	-	-	-	-	0.00
2	1.19	3.39	0.08	0.10	-	-	-	-	-
3	0.00	0.05	0.10	1.19	-	-	-	-	-
4	0.00	0.12	0.33	0.04	0.10	0.00	-	-	-
5	0.00	0.13	0.15	0.10	0.03	0.11	-	-	-
6	0.00	0.25	0.20	0.01	0.25	0.01	-	-	-
7	0.00	0.26	0.46	0.40	0.35	0.19	0.00	2.19	-
8	0.88	0.27	0.83	0.46	0.37	0.24	0.01	0.02	2.13
9	0.01	0.55	0.74	1.03	0.41	0.31	10.93	0.19	2.20

Table 3.6: Table of the position variance for the yaw going from 90° to 10°

lane	**0°**	**-10°**	**-20°**	**-30°**	**-40°**	**-50°**	**-60°**	**-70°**	**-80°**	**-90°**
1	0.62	0.03	0.07	0.09	0.13	0.68	0.74	0.68	0.93	0.00
2	0.73	0.46	-	-	0.29	0.39	0.50	0.90	0.60	0.00
3	-	-	0.11	0.01	-	0.37	0.32	0.55	0.39	0.01
4	-	-	-	0.03	0.06	0.17	0.31	0.25	0.12	0.00
5	-	-	-	-	0.06	0.10	0.20	0.17	0.16	0.00
6	-	-	-	-	-	0.05	0.06	1.15	2.81	0.00
7	-	-	-	-	-	-	0.00	0.12	0.04	1.63
8	0.28	-	-	-	-	0.00	2.23	0.05	0.03	0.00
9	2.58	-	-	-	-	-	-	0.00	0.00	0.00

Table 3.7: Table of the position variance for the yaw going from 0° to -90°

outcomes could still occur in optimal situations due to the innate randomness of employing artificial intelligence.

Finally, the variance tables suggest that a more extreme angle yields better results. However, this is not realistic as extreme angles lead to a shorter detection range, resulting in smaller spread. By considering the slope of these values, lack of measurements is not a factor. Tables 3.8 and 3.9 display a more logical result. It produces consistent findings and clearly indicates that extreme angles yield suboptimal performance, which is the expected result.

lane	**90°**	**80°**	**70°**	**60°**	**50°**	**40°**	**30°**	**20°**	**10°**
1	0.02	0.35	0.09	-	-	-	-	-	-0.04
2	0.00	0.16	0.08	0.13	-	-	-	-	-
3	0.00	0.04	0.07	0.11	-	-	-	-	-
4	0.00	0.05	0.09	0.10	0.13	0.20	-	-	-
5	-0.00	0.05	0.09	0.11	0.13	0.13	-	-	-
6	-0.00	0.07	0.11	0.15	0.15	0.15	-	-	-
7	0.07	0.16	0.14	0.17	0.17	0.17	0.10	0.13	-
8	-0.01	-0.16	0.16	0.18	0.19	0.19	0.15	0.13	0.13
9	0.00	0.08	0.17	0.15	0.21	0.14	-1.30	0.16	0.13

Table 3.8: Table of the slope for the yaw going from 90° to 10°

lane	**0°**	**10°**	**-20°**	**-30°**	**-40°**	**-50°**	**-60°**	**-70°**	**-80°**	**-90°**
1	-0.10	-0.12	-0.16	-0.23	-0.20	-0.23	-0.21	-0.18	-0.09	-0.00
2	-0.07	-0.12	-	-	-0.19	-0.19	-0.18	-0.15	-0.08	-0.00
3	-	-	-0.11	-0.11	-	-0.17	-0.16	-0.13	-0.07	0.01
4	-	-	-	-0.14	-0.13	-0.15	-0.12	-0.10	-0.07	0.00
5	-	-	-	-	-0.14	-0.13	-0.12	-0.09	-0.06	0.00
6	-	-	-	-	-	-0.13	-0.12	-0.05	-0.01	0.00
7	-	-	-	-	-	-	-0.07	-0.09	-0.04	0.04
8	0.06	-	-	-	-	0.70	-0.93	-0.08	-0.04	-0.00
9	0.09	-	-	-	-	-	-	-0.12	-0.13	-0.01

Table 3.9: Table of the slope for the yaw going from 0° to -90°

The optimal yaw angle is either 90 or -90 degrees as it minimises the variance suffered due to perspective.

3.2.3. Height effects

Height is the third parameter tested during the corrections. It determines the projected depth and can exacerbate any existing errors, rather than causing trajectory errors. The conditions of the experiment are to investigate the effect of using higher heights, so -90 degrees of yaw and -45 degrees of pitch are used. To simplify the study, only the middle lane is used. With this combination, various heights ranging from 12 to 28 metres have been tested. It is not worthwhile to test closer distances than 10 metres as the only noticeable effect is a reduction of the field of view and the test was discontinued at a height of 28 metres, at which point the vehicles became virtually invisible to the camera. Figure 3.8 illustrates the position for all distances.

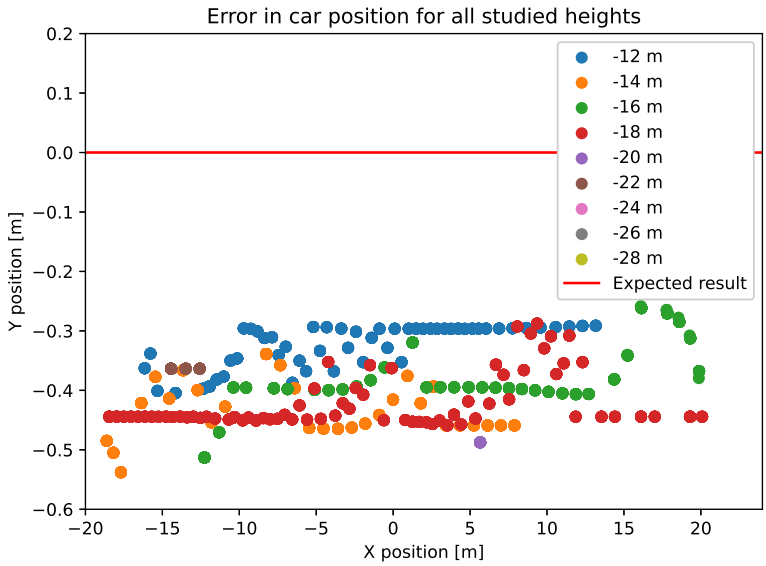


Figure 3.8: Effect of the height in the projected path

The height has a proportional effect on the error, which increases slightly with the distance. It is also worth noting the camera's range of vision, as this improves with distance. If the aim is to capture a greater number of cars simultaneously, a wider field of view would be beneficial. Another noteworthy observation is the decreased rate of detection at 20 metres and above. From 20 metres to 24 metres, the number of detections reduces significantly. Above 24 metres, there are no detections as the objects have become too small for the YOLO detector to operate.

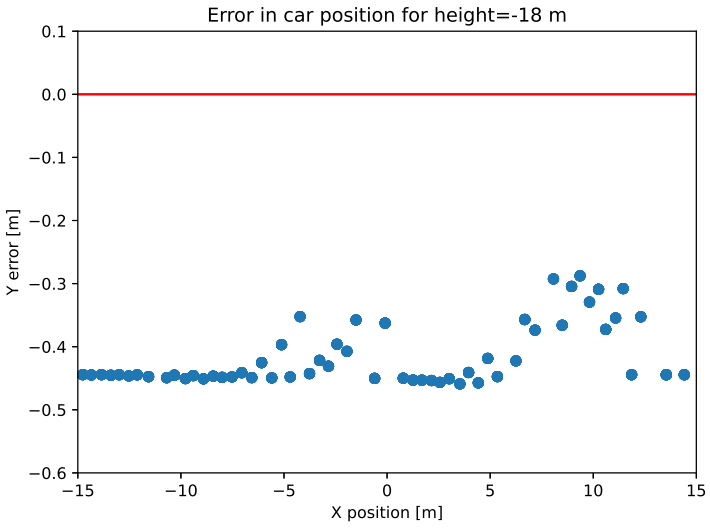


Figure 3.9: Effect of detaching from the vehicle in the routes

It is evident that certain graphs exhibit a recurring tendency to lose the line as demonstrated in Figure 3.9. This graph visually displays the vehicle's tracking loss. The Kalman filter is responsible for calculating the next position when the vehicle loses track, and during these values the approximation is the only input, slowly drifting. This phenomenon

occurs in nearly all routes, as longer distances increase the likelihood of detector failure and tracker failure.

Knowing the effects it had on position, it can be deduced that the speed will deteriorate with increasing heights. The findings demonstrate that there is a correlation between speed variance and altitude, while the mean error is consistent across all heights. The impact of the errors is more acute at higher altitudes, which means that the extremes suffer more deformation, resulting in a higher variance. When taking all the samples from one end of the graph to the other, the average speed is calculated. Any error caused by the extreme ends is averaged out and the primary source of error becomes the loss of connection.

3.3. Effects of an error in the telemetry

Having established the impact of drone positioning, the subsequent step is to investigate the impact of providing an incorrect telemetry. The test was conducted under the parameters of a 10-metre height, -45 degrees pitch and -90 degrees yaw. The telemetry packages were altered while displaying the same video in order to test the impact of incorrect data input.

The values were compared with the outcome generated by using the appropriate parameters, instead of the actual route as with the other instances. Figures 3.10, 3.11(a), and 3.12 exhibit the error in the tracked routes for each varaible.

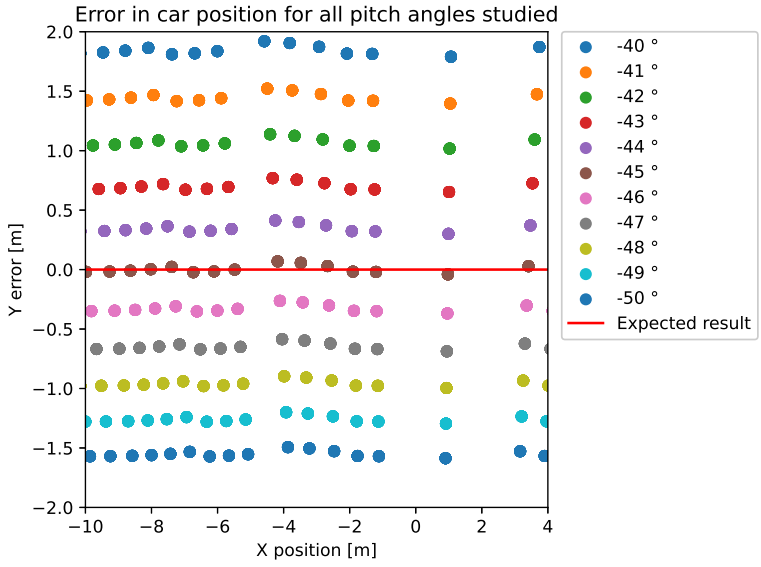


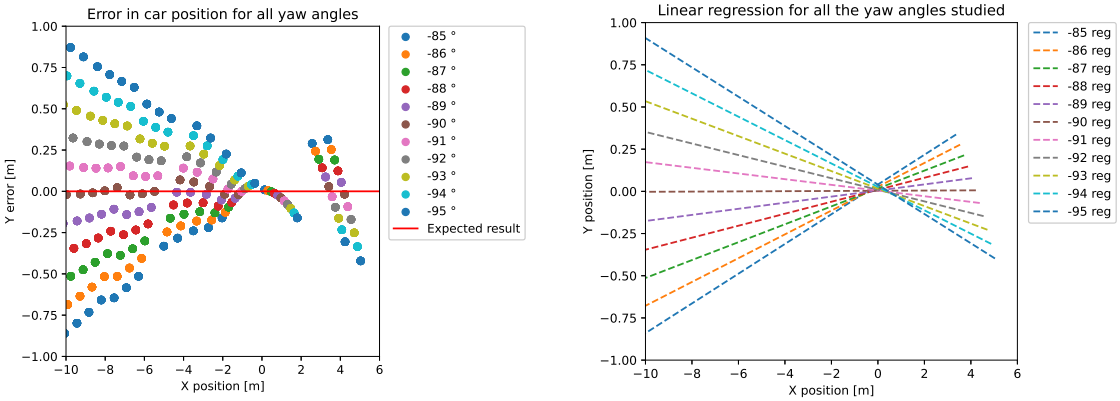
Figure 3.10: Effect of an angle change in the pitch

In Figure 3.10 the pitch variations are depicted. With each increase in the angle, the error becomes positive, indicating that the car is further away. This is reasonable because a more positive angle also means a lower viewing angle. Similarly, as the pitch angle becomes more negative, the visual angle of the vehicle becomes more vertical. Although the change in distance between angles appears linear, it is not. This discrepancy in position originates from the disparity between the original angle and the new one, thus, it is

proportionate to the difference in tangents. Formula 3.1 illustrates the correlation between the position error (Δe) and the angle employed (α_e) to cause it. The Δh in the equation references the height difference between the contact point with the ground and the center of the vehicle.

$$\Delta e = \Delta h(tan(-45[deg]) - tan(\alpha_e))[m] \tag{3.1}$$

Another noticeable outcome of the experiment is that all the routes are essentially identical but displaced. The points are marked in identical X positions with the same displacement. This occurs because the video frames are consistently the same, hence the convolutional neural network outcome will be indistinguishable for every angle, and discrepancies are only caused by the correction parameters. In the other experiments, this did not occur as the videos were different.



(a) Effect of the error in the yaw

(b) Linear regression of the error in the yaw

Figure 3.11: Effect of the error in the yaw and regression

In Figure 3.11(a), an investigation is conducted on the impact of a yaw error. The error in yaw can be explained by the fact that the vehicles are seen to be following a diagonal path, and therefore correcting the trajectory in the opposite angle around the centre point. Although this graph displays the actual data, it is more valuable to examine the linear regression of these trajectories, depicted in Figure 3.11(b).

From figure 3.11(b) it can be seen that all the routes coincide at the centre point, since in this position the yaw doesn't affect the estimate. Contrary to the pitch, the variation caused by incrementing the error is solely determined by the angle, not the height of the vehicle. Therefore, the angle between the different approximations is consistently 1 degree.

Figure 3.12 displays the effects of height. Similar to pitch, the error is caused in the Y axis. Higher heights create the impression that the object is further away, while lower heights suggest it is nearer. Another noticeable impact is the dispersion of points. The distance between points horizontally is relative to the perceived depth, so if the camera is lower, the distances in the image are calculated as shorter, leading to compression of the points on the x-axis. The spacing between lines is 1 metre, however, this is only proportional due to the -45 degree pitch angle. When the tangent is at -45, elevating the camera by 1 metre is equivalent to adding 1 metre to the y-axis. If the angle were different, the proportion would not remain the same.

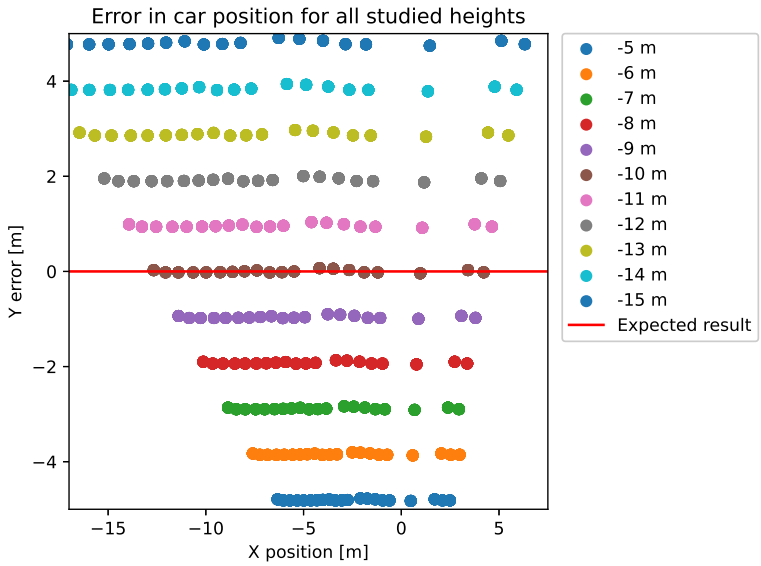


Figure 3.12: Error in the height caused by wrong telemetry

With pitch and yaw, errors predominantly affect position rather than speed, which is determined by the relationship between points. However, this is not the case with height. Height is proportional to the depth constant, and therefore any inaccuracies in height can result in a significant difference in the expected distance between points, causing the system to produce inaccurate outcomes. This relationship can be demonstrated by calculating the speed in relation to the height error, as shown in Figure 3.13.

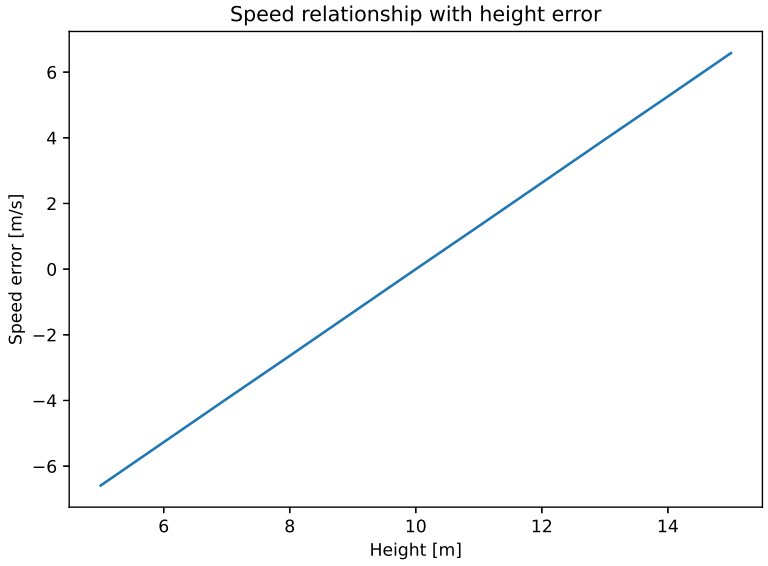
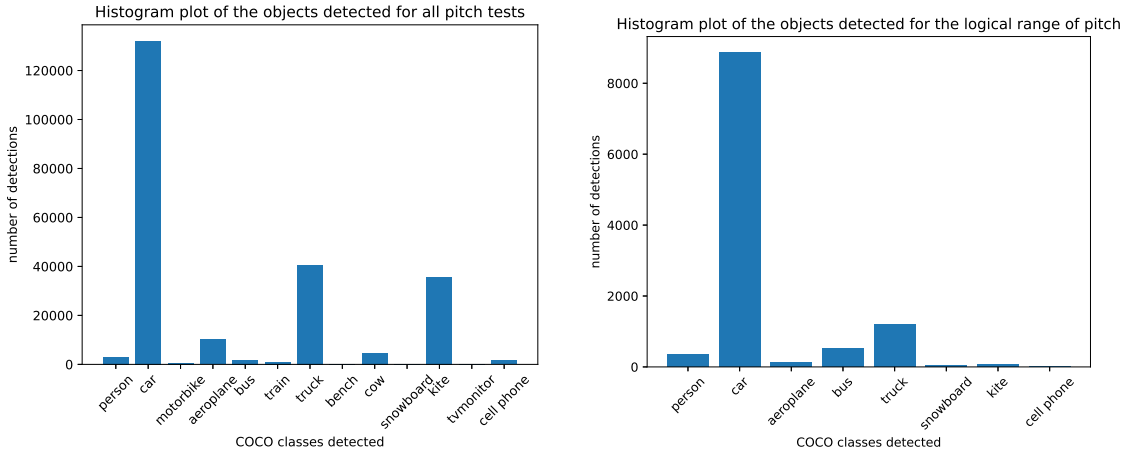


Figure 3.13: Relationship between height and speed

3.4. Detector precision derived from the experiments

The tests concentrate on the accuracy of the tool in determining position and speed, which is the primary objective of the application. The detection and tracking setup is also a potential source of error which impacts the tool's capability. Although the tool worked properly for general car detection, there were some edge cases where other objects were detected in place of the cars. These results also provide an insight into the consequences of not having trained the neural network for the task. The histogram of the classes for pitch is shown in Figure 3.14(a).



(a) Histogram from the complete pitch data

(b) Histogram from the optimal range

Figure 3.14: Histograms of the classes obtained from pitch

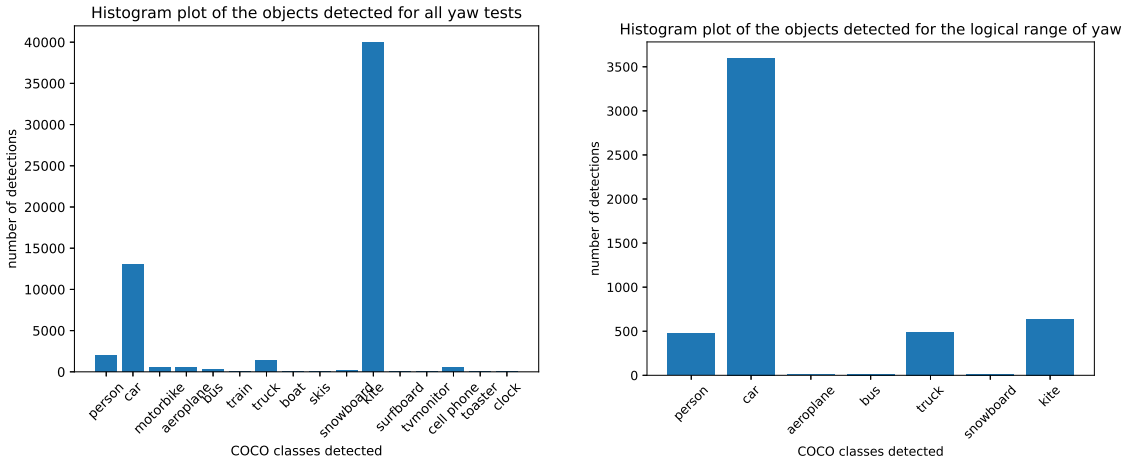
From the 80 categories of COCO, it only shows the ones that happen. As anticipated, the bulk of the detections are cars. While the test seemingly highlights multiple erroneous detections, it fails to provide any insights into their cause.

Most of the errors are out of the optimum range. Based on previous tests, the most favourable position falls within a yaw of -90 and a pitch of between -70 and -40. When using the middle lanes, rather than the outermost ones, the histogram (see Figure 3.14(b)) shows that most errors disappear. Taking all angles into account, the accurate detection rate was 57%, which increased to 79% within the limited range. To avoid misclassifications, the number of COCO classes employed can be limited, resulting in either detection or non-detection with no erroneous classifications. This approach was maintained in the experiments to determine the impact of using more classes not under investigation.

The yaw experiments can be replicated to obtain the results displayed in Figure 3.15(a). Yaw showcases an unexpected outcome as it was intended to detect cars, yet the majority of detections are kites. This phenomenon is attributed to the angle of vision: if the car is too vertical, confidence in the car detection drops while confidence in the kite detection increases.

For angles with an absolute value lower than 60 degrees, kite detections become more frequent. Above those 60 degrees of absolute yaw, the values are comparable to those of pitch, with an accuracy of 70%. The histogram featuring the usable range can be found in Figure 3.15(b).

These tests do not encompass the full range of potential detections. The outcomes for cars



(a) Histogram from the complete yaw data

(b) Histogram from the optimal range

Figure 3.15: Histograms of the classes obtained from yaw

are generally positive, but the detector has not been evaluated with bikes or actual mixed traffic.Accomplishing such evaluations would have required redesigning the simulator from scratch, as AirSim lacks this functionality.

3.5. Flight tests of the LABYRINTH project

For the road transport, flight tests were conducted in the CIAR (*Centro de Investigación Aeroportada de Rozas*). The tests were carried out on national roads with low traffic density. The view of the roads was captured from a high altitude, enabling a wider field of vision. The primary challenge faced during the tests was the flight duration that was limited by battery capacity and other flight objectives. While the flight tests of the LABYRINTH project were inconclusive regarding the functionality of the application, the videos were recorded, and the telemetry stored, enabling the repetition of valuable tests and the extraction of as much content as possible.

The number of real tests was insufficient to draw significant conclusions. However, the results revealed a higher number of incorrect detections and recorded more instances of the detector losing tracking, which is to be expected. As for the reported speed, it remains unknown if it is accurate, since the tests were only conducted using regular traffic.

It remained possible to observe the road, but the delay would have been too high to utilize the video feed for operating a drone camera, as the lag would cause substantial discomfort to the operator. The main sources of delay were the latency in drone streaming and internet connectivity performance in the field. In laboratory conditions, the most noteworthy delay was caused by the HLS streaming protocol, while in real-world tests the connection was not as stable, causing buffering issues which continued to add to the latency. Reducing the video stream's quality mitigated the effects but did not completely eliminate them.

CONCLUSION

The chapter aims to reflect on the objectives of the project and to outline a number of improvements and future work that could further extend the capabilities of the system.

The first objective was to accurately position the vehicles on a 2D plane. The objective was met with half a meter precision, but it relied heavily on the drone's location. While its precision is proportional to the quality of the parameters sent, the overall outcome is quite satisfactory, considering the hardware and technical limitations. To increase accuracy, it may be useful to restrict the vehicle pool to cars only or utilize a LiDAR to obtain depth measurements. Similar to the position, speed was obtained with various degrees of success. Speed, like vehicle position, was measured to varying degrees of success. While the average error does not affect the speed, its accuracy is not sufficient to be used for fining. Currently, the application could provide an initial estimate and then confirm it as an infraction via a certified speed radar. The user interface was utilized to demonstrate the tool to the partners who participated in the tests. An unforeseen advantage of the software was that, thanks to its user-friendly connectivity, all partners had access to the camera video in real time, without having to be near the operators.

Finally, the project was capable of employing any type of drone, but this limited the project considerably. The tool has no dependency on the drone, which means that any model can be used as long as the video stream from the camera and telemetry are available in real time. However, this results in a loss of precision, as better depth measurements or position optimization without the necessity of an operator could have been obtained.

Due to the difficulty of conducting real test flights in the context of the LABYRINTH project and the absence of available resources, there is insufficient evidence to fully evaluate the tool's performance in real-world conditions. However, the simulated tests provide valuable insights that would not have been possible on the road, as they can eliminate external factors such as bad lighting or wind.

One LABYRINTH project participant was a business committed to assisting traffic control authorities to implement new technologies. Through testing, they determined that the technology had potential as a preliminary screening device rather than as a means of imposing fines. A principal limitation of existing camera radars is that only one car can be scrutinised at any given time. For instance, their flagship helicopter, the Pegasus, boasts exceptional versatility. However, it has the limitation of only being able to detect one car's speed at a time, making extensive surveillance impossible. They discovered the tool's most significant advantage to be its ability to detect multiple vehicles on the road simultaneously.

Future work

The objectives have been fulfilled, however, there is room for improvement in certain areas. One possibility for detector enhancement could be training the neural network to fit the specific task. For the project, the tool development and CNN training was omitted due to time constraints and the availability of all the expected objects for detection in the COCO database. To enhance the neural network, a thorough database of images of at least cars, trucks, bicycles and motorcycles, all taken from the drone's perspective and in different conditions, would be needed. That, alongside training and optimising hyperparameters,

was beyond the scope of the project. Nevertheless, it is indisputable that it would enhance the system. Alternatively, adding a secondary detection technique, for instance optical flow, could be an option, but it may restrict real-time functionality.

The positional projection is simple but innacurate. It is possible to employ more advanced methods, like utilising drones with dual cameras or even a LiDAR, a feature present in certain DJI drones. Alternatively, one could incorporate data concerning the objects to be detected, such as the typical height of cars, or use items on the road as a gauge for scale. Both solutions would enhance the tool, yet they impose restrictions on its usage.

In conclusion, the project can be improved in several ways. If continued, any of the aforementioned options could be considered. Personally, tailoring the neural network to the drone camera presents the most intriguing prospects for the future.

Final words

While the thesis can be viewed as a project completed over a six-month period, I must acknowledge that my prior knowledge was pivotal in enabling me to complete it. My three-year employment at ICARUS Research group alongside Pablo Royo was an outstanding introduction to UAS (Unmanned Aircraft System). It enabled me to come to DLR already knowing all the basics of the project, which in turn allowed me to focus on the computer vision aspects that were completely new to me.

I also wanted to talk about my experience in DLR. Through collaborating with Miguel, I have contributed to my own project while also supporting the LABYRINTH initiative. This opportunity has enabled me to engage with people from across Europe and contribute to the final stages of a 3-year endeavour. I even had the chance to contribute to presentations. The experience provided me with invaluable resources, aid, and insights, far beyond what I could have anticipated for a Bachelor's thesis.

I am grateful to the ICARUS Research group and DLR for providing me with the necessary resources to work with drones. Working on a project like LABYRINTH for the last 6 months has been a pleasure.

BIBLIOGRAPHY

[1] A.G. Sims and K.W. Dobinson. The sydney coordinated adaptive traffic (scat) system philosophy and benefits. *IEEE Transactions on Vehicular Technology*, 29(2):130–137, 1980. 8

[2] Xiao Yun Lu, Pravin Varaiya, Roberto Horowitz, Zhaomiao Guo, and Joe Palen. Estimating traffic speed with single inductive loop event data. *Transportation Research Record*, pages 157–166, 12 2012. 8

[3] David Fernández Llorca, Antonio Hernández Martínez, and Iván García Daza. Vision-based vehicle speed estimation: A survey. 1 2021. 8

[4] F W Cathey and D J Dailey. A novel technique to dynamically measure vehicle speed using uncalibrated roadway cameras. pages 777–782, 2005. 13

[5] Ruimin Ke, Sung Kim, Zhibin Li, and Yinhai Wang. Motion-vector clustering for traffic speed detection from uav video. pages 1–5, 2015. 13

[6] Tarun Kumar and Dharmender Singh Kushwaha. An efficient approach for detection and speed estimation of moving vehicles. volume 89, pages 726–731. Elsevier B.V., 2016. 13

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60:84–90, 5 2017. 14

[8] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. 12 2016. 14

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 6 2015. 15

[10] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. 12 2016. 15

[11] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. 4 2018. 15

[12] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. 4 2020. 15

[13] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. Yolov6: A single-stage object detection framework for industrial applications. 9 2022. Source code available here: https://github.com/meituan/YOLOv6. 15, 29

[14] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. 7 2022. Source code available here: https://github.com/WongKinYiu/yolov7. 15, 29

[15] Ultralytics Inc. Ultralytics official site. Available online: https://ultralytics.com/. Accessed: 2023-09-04. 15

[16] Ya Liu, Yao Lu, Qingxuan Shi, and Jianhua Ding. Optical flow based urban road vehicle tracking. pages 391–395, 2013. 16

[17] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. 2 2016. 16

[18] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. 3 2017. 19

[19] Jiaxing Zhang, Wen Xiao, Benjamin Coifman, and Jon P. Mills. Vehicle tracking and speed estimation from roadside lidar. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:5597–5608, 2020. 19

[20] Elnaz Vakili, Maryam Shoaran, and Mohammad R. Sarmadi. Single–camera vehicle speed measurement using the geometry of the imaging system. *Multimedia Tools and Applications*, 79:19307–19327, 7 2020. 20

[21] Shengnan Lu, Yuping Wang, and Huansheng Song. A high accurate vehicle speed estimation method. *Soft Computing*, 24:1283–1291, 1 2020. 20

[22] Konstantinos Kanistras, Goncalo Martins, Matthew J. Rutherford, and Kimon P. Valavanis. A survey of unmanned aerial vehicles (uavs) for traffic monitoring. pages 221–234, 2013. 20

[23] Jing Li, Shuo Chen, Fangbing Zhang, Erkang Li, Tao Yang, and Zhaoyang Lu. An adaptive framework for multi-vehicle ground speed estimation in airborne videos. *Remote Sensing*, 11, 5 2019. 20

[24] Debojit Biswas, Hongbo Su, Chengyi Wang, and Aleksandar Stevanovic. Speed estimation of multiple moving objects from a moving uav platform. *ISPRS International Journal of Geo-Information*, 8, 5 2019. 20

[25] Amazon Web Services. Amazon web services official site. Available online: https://aws.amazon.com/. Accessed: 2023-09-05. 24

[26] blueviron. Mediamtx. Available online: https://github.com/bluenviron/mediamtx. 24, 33

[27] OpenCV team. Opencv official site. Available online: https://opencv.org/. Accessed: 2023-08-31. 25

[28] Markus Buehren. hungarian-algorithm-cpp. Available online: https://github.com/mcximing/hungarian-algorithm-cpp. Accessed: 2023-09-01. 26

[29] OpenJS Foundation. Electron official site. Available online: https://www.electronjs.org/. Accessed: 2023-09-01. 37

[30] Evan You. Vue official site. Available online: https://vuejs.org/. Accessed: 2023-09-01. 37

[31] DJI. Matrice 300 rtk official site. Available online: https://enterprise.dji.com/matrice-300. Accessed: 2023-09-03. 39

[32] DJI. Manifold 2 official site. Available online: https://www.dji.com/manifold-2?site=brandsite&from=insite_search. Accessed: 2023-09-04. 40

APPENDIX A. FULL GANTT DIAGRAM

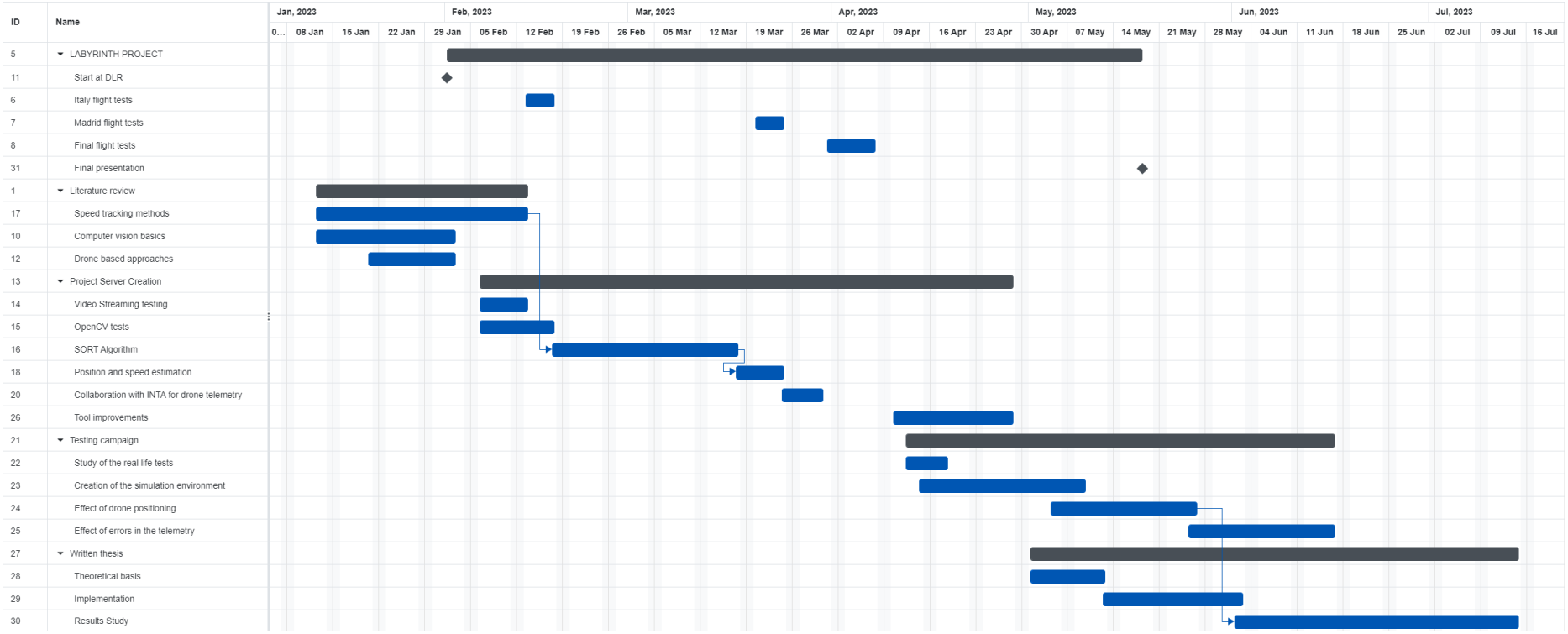


Figure A1: Gantt diagram for the project duration.

APPENDIX B. CAMERA SIMULATION JUPYTER NOTEBOOK

In [ ]:
```
%matplotlib widget

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from utils import *
```

In [ ]:
```
# define point parameters
# points
points = np.array([[0,0,0],
                   #[2,0,0],[-2,0,0],[0,2,0],[0,-2,0],
                   [2,-2,0],[-2,-2,0],[2,2,0],[-2,2,0],
                   #[2,1,0],[2,-1,0],[-2,1,0],[-2,-1,0],
                   #[1,2,0],[1,-2,0],[-1,2,0],[-1,-2,0],
                   #[2,0.5,0],[-2,0.5,0],[2,-0.5,0],[-2,-0.5,0],
                   #[0.5,2,0],[0.5,-2,0],[-0.5,2,0],[-0.5,-2,0],
                   #[2.5,-2,0],[-2.5,-2,0],[-2.5,2,0],[2.5, 2,0],
                   #[2,2.5,0],[-2,2.5,0],[2,-2.5,0],[-2,-2.5,0],
                   #[2,1.5,0],[2,-1.5,0],[-2,1.5,0],[-2,-1.5,0],
                   #[-2.5,1,0],[-2.5,-1,0],[2.5,1,0],[2.5,-1,0],
                   #[1.5,2,0],[1.5,-2,0],[-1.5,2,0],[-1.5,-2,0],
                   #[-1,2.5,0],[-1,-2.5,0],[1,-2.5,0],[1,2.5,0]
                   ])
```

In [ ]:
```
# Define Camera properties

#angles = [+np.pi/4]
angles = [0]
order = 'x'
R = create_rotation_transformation_matrix(angles, order)

# World position of the camera
T = np.array([0,0,4])

# image plane
img_size = (7, 7)
h, w = img_size
f=4
a=1
s=0
cx=0
cy=0

K = np.array([[f,s,cx],
              [0,a*f,cy],
              [0,0,1]]
             )
```

In [ ]:
```
def convert_grid_to_homogeneous(xx, yy, Z, img_size):
    '''
    Extract coordinates from a grid and convert them to homogeneous coordinates
    '''
    h, w = img_size
    pi = np.ones(shape=(3, h*w))
    c = 0
    for i in range(h):
```

```
        for j in range(w):
            x = xx[i, j]
            y = yy[i, j]
            z = Z[i, j]
            point = np.array([x, y, z])
            pi[:3, c] = point
            c += 1
    return pi

def convert_homogeneous_to_grid(pts, img_size):
    '''
    Convert a set of homogeneous points to a grid
    '''
    xxt = pts[0, :].reshape(img_size)
    yyt = pts[1, :].reshape(img_size)
    Zt = pts[2, :].reshape(img_size)

    return xxt, yyt, Zt
```

In [ ]:

```
# Plot All
def plotcamera(img_size,K,R,T,filename):
    h,w=img_size

    fig = plt.figure(figsize=(16, 7))
    ax = fig.add_subplot(121,projection='3d')
    ax.set(xlim=(-4, 4), ylim=(-4, 4), zlim=(0, 5))
    ax = pr.plot_basis(ax)

    # plot plane
    xx, yy, Z = create_image_grid(-f, img_size)
    pt_h = convert_grid_to_homogeneous(xx, yy, Z, img_size)
    T2= np.dstack([T]*h*w)
    pt_h_transformed = np.linalg.inv(R) @ (pt_h ) + T2
    xxt, yyt, Zt = convert_homogeneous_to_grid(pt_h_transformed[0], img_size)

    # Plot camera plane
    ax.plot_surface(xxt, yyt, Zt, alpha=0.75)

    #plot camera
    ax.scatter(T[0],T[1],T[2], color="orange",marker='^')
    ax.set_xlabel("x [m]")
    ax.set_ylabel("y [m]")
    ax.set_zlabel("z [m]")
    ax.set_title("3D representation of the setup", fontsize="16")
    # plot points

    for point in points:
        ax.scatter(point[0],point[1],point[2])
        ax.plot([point[0],T[0]],[point[1],T[1]],[point[2],T[2]])
    ax = fig.add_subplot(122)
    ax.set(xlim = (-(h // 2), w // 2), ylim = (-(h // 2), w // 2))
    V_points = np.zeros((np.shape(points)[0],3))
    i=0

    last_point =None
    for point in points:

        V = K.dot(R.dot(point-T))
        ax.scatter(-V[0]/V[2], -V[1]/V[2])
        V_points[i,:]=[V[0]/V[2], V[1]/V[2],1]
```

```
        i+=1
    ax.set_xlabel("Points in the X axis [px]")
    ax.set_ylabel("Points in the Y axis [px]")
    ax.set_title("Image obtained", fontsize="16")
    plt.savefig(filename,dpi=200)
    return V_points


# Plot All
def plotcameraOffset(img_size,K,R,T,filename):
    h,w=img_size

    fig = plt.figure(figsize=(16, 7))
    ax = fig.add_subplot(121,projection='3d')
    ax.set(xlim=(-4, 4), ylim=(-4, 4), zlim=(0, 5))
    ax = pr.plot_basis(ax)

    # plot plane
    xx, yy, Z = create_image_grid(-f, img_size)
    pt_h = convert_grid_to_homogeneous(xx, yy, Z, img_size)
    T2= np.dstack([T]*h*w)
    pt_h_transformed = np.linalg.inv(R) @ (pt_h ) + T2
    xxt, yyt, Zt = convert_homogeneous_to_grid(pt_h_transformed[0], img_size)

    # Plot camera plane
    ax.plot_surface(xxt, yyt, Zt, alpha=0.75)

    #plot camera
    ax.scatter(T[0],T[1],T[2], color="orange",marker='^')
    ax.set_xlabel("x [m]")
    ax.set_ylabel("y [m]")
    ax.set_zlabel("z [m]")
    ax.set_title("3D representation of the setup", fontsize="16")
    # plot points

    for point in points:
        ax.scatter(point[0],point[1],point[2])
        ax.plot([point[0],T[0]],[point[1],T[1]],[point[2],T[2]])
    ax = fig.add_subplot(122)
    ax.set(xlim = (-(h // 2 )+4, w // 2+4), ylim = (-(h // 2 ), w // 2 ))
    V_points = np.zeros((np.shape(points)[0],3))
    i=0

    last_point =None
    for point in points:

        V = K.dot(R.dot(point-T))
        ax.scatter(-V[0]/V[2], -V[1]/V[2])
        V_points[i,:]=[V[0]/V[2], V[1]/V[2],1]
        i+=1
    ax.set_xlabel("Points in the X axis [px]")
    ax.set_ylabel("Points in the Y axis [px]")
    ax.set_title("Image obtained", fontsize="16")
    plt.savefig(filename,dpi=800)
    return V_points


# Plot All
def plotcameraSkew(img_size,K,R,T,filename):
    h,w=img_size
```

```
    fig = plt.figure(figsize=(16, 7))
    ax = fig.add_subplot(121,projection='3d')
    ax.set(xlim=(-4, 4), ylim=(-4, 4), zlim=(0, 5))
    ax = pr.plot_basis(ax)
    x = [1,-3,-1,3]
    y = [-2,-2,2,-2]
    z = [0,0,0,0]
    x = [ 3, -3, -3, 3]
    y = [-3, -2,  3,2]
    z = [0,0,0,0]
    verts = [list(zip(y,x,z))]
    ax.add_collection3d(Poly3DCollection(verts,alpha=0.75))

    #plot camera
    ax.scatter(T[0],T[1],T[2], color="orange",marker='^')
    ax.set_xlabel("x [m]")
    ax.set_ylabel("y [m]")
    ax.set_zlabel("z [m]")
    ax.set_title("3D representation of the setup", fontsize="16")
    # plot points

    for point in points:
        ax.scatter(point[0],point[1],point[2])
        ax.plot([point[0],T[0]],[point[1],T[1]],[point[2],T[2]])
    ax = fig.add_subplot(122)
    ax.set(xlim = (-(h // 2), w // 2), ylim = (-(h // 2), w // 2))
    V_points = np.zeros((np.shape(points)[0],3))
    i=0

    last_point =None
    for point in points:

        V = K.dot(R.dot(point-T))
        ax.scatter(-V[0]/V[2], -V[1]/V[2])
        V_points[i,:]=[V[0]/V[2], V[1]/V[2],1]
        i+=1
    ax.set_xlabel("Points in the X axis [px]")
    ax.set_ylabel("Points in the Y axis [px]")
    ax.set_title("Image obtained", fontsize="16")
    plt.savefig(filename,dpi=800)
    return V_points
```

In [ ]:

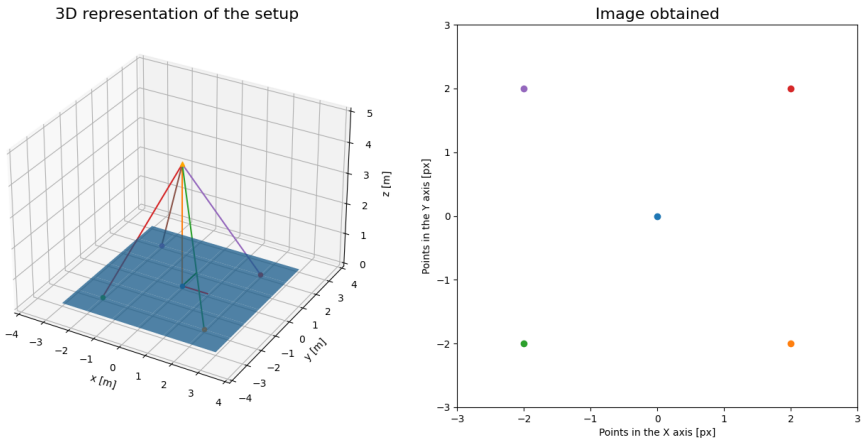
```
# BASE
#angles = [+np.pi/4]
angles = [0]
order = 'x'
R = create_rotation_transformation_matrix(angles, order)

# World position of the camera
T = np.array([0,0,4])

# image plane
img_size = (7, 7)
h, w = img_size
f=4
a=1
s=0
cx=0
cy=0
```

```
K = np.array([[f,s,cx],
              [0,a*f,cy],
              [0,0,1]]
             )

V_points = plotcamera(img_size,K,R,T,"plot/base.png")

# offset
angles = [0]
order = 'x'
R = create_rotation_transformation_matrix(angles, order)

# World position of the camera
T = np.array([2,0,4])

# image plane
img_size = (7, 7)
h, w = img_size
f=4
a=1
s=0
cx=0
cy=0
K = np.array([[f,s,cx],
              [0,a*f,cy],
              [0,0,1]]
             )

V_points = plotcamera(img_size,K,R,T,"plot/lateral_disp.png")

# angle
angles = [np.pi/4]
order = 'x'
R = create_rotation_transformation_matrix(angles, order)

# World position of the camera
T = np.array([0,0,4])

# image plane
img_size = (7, 7)
h, w = img_size
f=4
a=1
s=0
cx=0
cy=0
K = np.array([[f,s,cx],
              [0,a*f,cy],
              [0,0,1]]
             )

V_points = plotcamera(img_size,K,R,T,"plot/angle.png")

# focal
angles = [0]
order = 'x'
R = create_rotation_transformation_matrix(angles, order)

# World position of the camera
T = np.array([0,0,4])
```

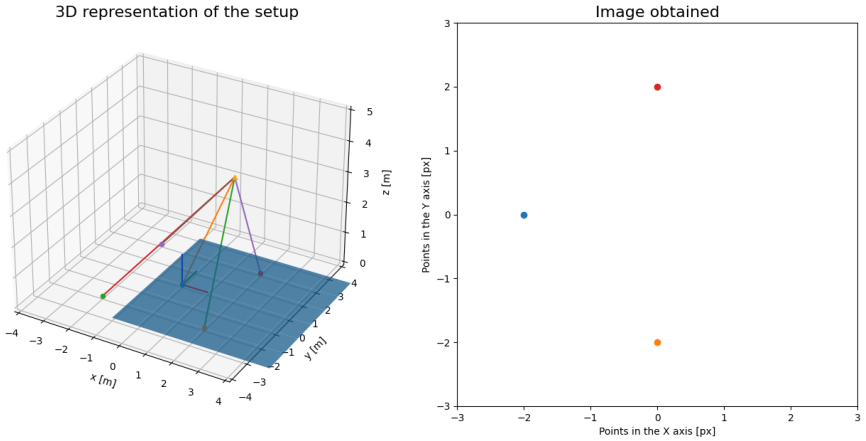
```
# image plane
img_size = (7, 7)
h, w = img_size
f=2
a=1
s=0
cx=0
cy=0
K = np.array([[f,s,cx],
              [0,a*f,cy],
              [0,0,1]]
             )

V_points = plotcamera(img_size,K,R,T,"plot/shorter_focal.png")

# offset_image
angles = [0]
order = 'x'
R = create_rotation_transformation_matrix(angles, order)

# World position of the camera
T = np.array([0,0,4])

# image plane
img_size = (7, 7)
h, w = img_size
f=4
a=1
s=0
cx=-4
cy=0
K = np.array([[f,s,cx],
              [0,a*f,cy],
              [0,0,1]]
             )

V_points = plotcameraOffset(img_size,K,R,T,"plot/offset.png")

# skew
angles = [0]
order = 'x'
R = create_rotation_transformation_matrix(angles, order)

# World position of the camera
T = np.array([0,0,4])

# image plane
img_size = (7, 7)
h, w = img_size
f=4
a=1
s=2
cx=0
cy=0
K = np.array([[f,s,cx],
              [0,a*f,cy],
              [0,0,1]]
             )
```

```
V_points = plotcameraSkew(img_size,K,R,T,"plot/skew.png")
```

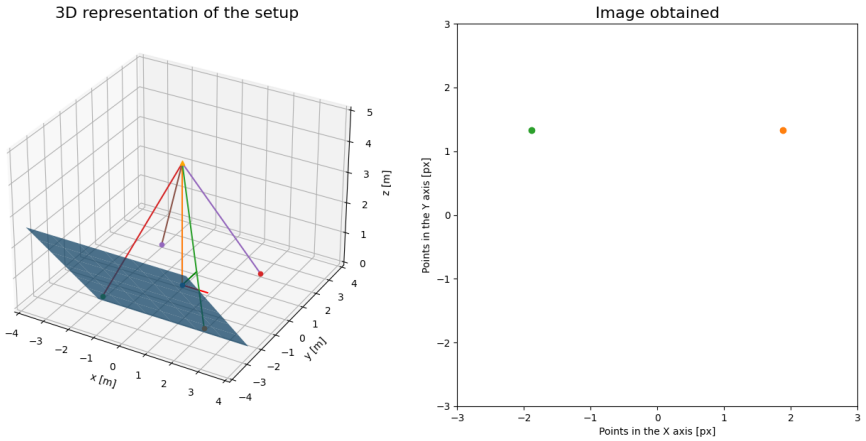
Figure



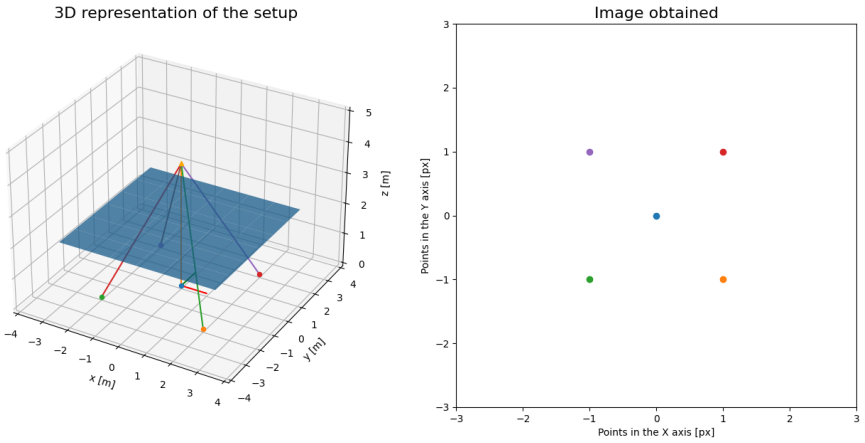
Figure



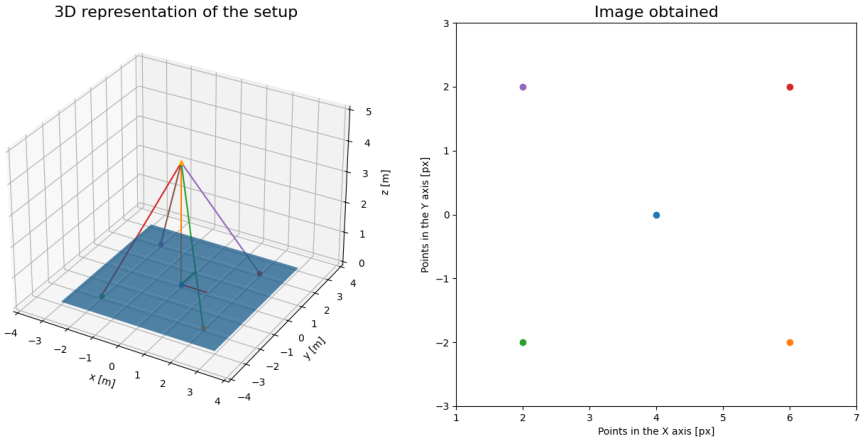
Figure



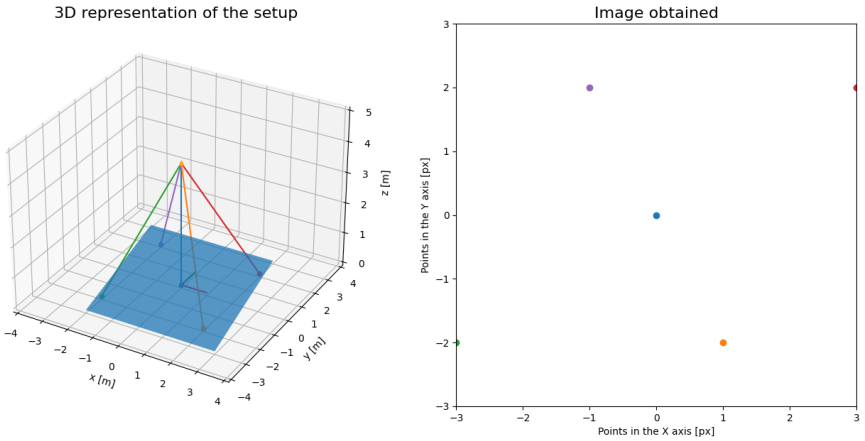
Figure



Figure



Figure



APPENDIX C. RESULT PLOTTING JUPYTER NOTEBOOKS

Preliminary necessities

In [ ]:
```
%matplotlib widget

import mysql.connector
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import csv
pd.options.display.float_format = '{:20,.2f}'.format
```

In [ ]:
```
# Open a connection to the MySQL database
conn = mysql.connector.connect(
    host='127.0.0.1',
    user="root",
    password="XXX",
    database="RTVT"
)
```

In [ ]:
```
# Execute the SQL query to retrieve the positions
target_id = 80 # Replace with the ID of the target you're interested in
query = f"SELECT x, y FROM positions WHERE target_id = {target_id} ORDER BY `num
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
x_values = [row[0] for row in results]
y_values = [row[1] for row in results]
print((x_values[:10]))
print((y_values[:10]))
```

In [ ]:
```
# Definition of all angles and lanes
#check number of targets for each value
lanes = range(1,10)
angles = range(0,-91,-5)
data =np.zeros([len(lanes),len(angles)])
i =0
for lane in lanes:
    j=0
    for angle in angles:
        query = f"SELECT COUNT(*) AS target_count FROM targets WHERE pitch = {ar
        cursor = conn.cursor()
        cursor.execute(query)
        results = cursor.fetchall()
        val = [row[0] for row in results]
        data[i,j]=int(val[0])
        j+=1
    i+=1

pd.DataFrame(data,columns=angles)
```

In [ ]:
```
yaw = -90
R=np.sqrt(2)*10
lanes = range(1,10)
angles = range(0,-91,-5)
expected_y = np.empty([len(lanes),len(angles)])
```

```
i=0
for pos in lanes:
    j = 0
    for pitch in angles:
        y = 10 + R*np.cos(np.deg2rad(pitch))
        z = R*np.sin(np.deg2rad(pitch))
        vector_cam = np.array([130,y,z])
        vector_car = np.array([100,2*(pos) , 0])
        diff = vector_car-vector_cam
        expected_y[i,j] = diff[1]
        j+=1
    i+=1

df = pd.DataFrame(expected_y,columns=angles)
display(df)
```

Position study

In [ ]:
```
fig, ax = plt.subplots()
pitch =-45
lane = 6
j = int(np.abs(pitch/-5))
i=lane-1


query = f"SELECT x, y FROM positions WHERE target_id  IN (SELECT id FROM targets
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
x_values = [row[0] for row in results]
y_values = [row[1] for row in results]
normalized_y = y_values+expected_y[i,j]
ax.scatter(x_values, normalized_y)
ax.plot([-10,10],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(2), 2))
plt.xlabel('X position [m]')
plt.ylabel('Y position [m]')
plt.title(f"normalized car position for all targets with pitch={pitch} and lane=
```

In [ ]:
```
pitch =-45
lane = 5
mean_error_y_table = np.ones([len(lanes),len(angles)])
sample_size = np.zeros([len(lanes),len(angles)])
subplot_base=10*90/5
for lane in range(1,10):
    i =lane-1
    for pitch in angles:
        query = f"SELECT x, y FROM positions WHERE target_id  IN (SELECT id FROM
        cursor = conn.cursor()
        cursor.execute(query)
        results = cursor.fetchall()

        x_values = [row[0] for row in results]
        y_values = [row[1] for row in results]
        j = int(np.abs(pitch/-5))
        normalized_y = y_values+expected_y[i,j]
        if len(x_values)>1:
```

```
            mean_error_y_table[i,j]=np.sum(normalized_y)/len(normalized_y)
            sample_size[i,j] = len((normalized_y))


plt.xlabel('X position')
plt.ylabel('Y position')
df = pd.DataFrame(mean_error_y_table,columns=angles)
display(df)
df = pd.DataFrame(sample_size,columns=angles)
display(df)
```

In [ ]:

```
fig, (ax1, ax2) = plt.subplots(1, 2)
pitch =-45
lane = 5
angles = range(0,-90,-5)
angles1 = range(0,-90,-10)
angles2 = range(5,-95,-10)

for pitch in angles1:
    query = f"SELECT x, y FROM positions WHERE target_id  IN (SELECT id FROM tar
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()

    x_values = [row[0] for row in results]
    y_values = [row[1] for row in results]
    j = int(np.abs(pitch/-5))
    i =lane-1
    normalized_y = y_values+expected_y[i,j]
    ax1.scatter(x_values, normalized_y)

ax1.set(xlim = (-(13), 13), ylim = (-(3), 3),xlabel='X position [m]',ylabel='Y p
ax1.plot([-16,16],[0,0],"r")
ax1.legend(angles1,loc='upper left')

for pitch in angles2:
    query = f"SELECT x, y FROM positions WHERE target_id  IN (SELECT id FROM tar
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()

    x_values = [row[0] for row in results]
    y_values = [row[1] for row in results]
    j = int(np.abs(pitch/-5))
    i =lane-1
    normalized_y = y_values+expected_y[i,j]
    ax2.scatter(x_values, normalized_y)

ax2.set(xlim = (-(13), 13), ylim = (-(3), 3),xlabel='X position [m]',ylabel='Y p
ax2.plot([-16,16],[0,0],"r")
ax2.legend(angles2,loc='upper left')

fig.suptitle(f"normalized position for all studied angles in lane {lane}")
```

Speed Study

In [ ]:
```
base_speed =5
fps =15
expected_speed = np.ones([len(lanes),len(angles)])
with open('videos.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=';')
    for row in spamreader:
        properties = row[0].split("_")
        pitch = float(properties[0])
        lane = int(properties[3])
        j = int(np.abs(pitch/-5))
        i =lane-1
        expected_speed[i,j] = base_speed*fps*float(row[1])
df = pd.DataFrame(expected_speed,columns=angles)
display(df)
```

In [ ]:
```
mean_error_speed_table = np.ones([len(lanes),len(angles)])
for lane in range(1,10):
    i =lane-1
    for pitch in angles:
        query = f"SELECT vx, vy FROM positions WHERE target_id  IN (SELECT id FR
        cursor = conn.cursor()
        cursor.execute(query)
        results = cursor.fetchall()

        vx_values = np.array([row[0] for row in results])
        vy_values = np.array([row[1] for row in results])

        j = int(np.abs(pitch/-5))
        mean_speed = np.sqrt(vx_values**2+vy_values**2)
        normalized_speed = mean_speed-expected_speed[i,j]
        if len(vx_values)>1:
            mean_error_speed_table[i,j]=np.sum(normalized_speed)/len(normalized_
df = pd.DataFrame(mean_error_speed_table,columns=angles)
display(df)
```

In [ ]:
```
fig, ax = plt.subplots()
pitch=-60
lane=5
query = f"SELECT id FROM targets WHERE pitch = {pitch} AND car_lane={lane};"
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
vx_values = [row[0] for row in results]
print(vx_values[3])

query = f"SELECT x, y,num FROM positions WHERE target_id = {vx_values[3]} AND cl
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()

x = np.array([row[0] for row in results])
y = np.array([row[1] for row in results])
num = np.array([row[2] for row in results])
print(num)
ax.quiver(x[:-1], y[:-1], x[1:]-x[:-1], y[1:]-y[:-1], scale_units='xy', angles='
ax.set(xlim = (-(13), 13), ylim = (-(10), 10))
plt.xlabel('X position')
plt.ylabel('Y position')
```

Preliminary necessities

In [ ]:
```
%matplotlib widget

import mysql.connector
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import csv
pd.options.display.float_format = '{:20,.2f}'.format
```

In [ ]:
```
# Open a connection to the MySQL database
conn = mysql.connector.connect(
    host='127.0.0.1',
    user="root",
    password="XXXX",
    database="RTVT_yaw"
)
```

In [ ]:
```
# Execute the SQL query to retrieve the positions
target_id = 34 # Replace with the ID of the target you're interested in
query = f"SELECT x, y FROM positions WHERE target_id = {target_id} ORDER BY `num
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
x_values = [row[0] for row in results]
y_values = [row[1] for row in results]
print((x_values[:10]))
print((y_values[:10]))
```

In [ ]:
```
# Definition of all angles and lanes
#check number of targets for each value
lanes = range(1,10)
angles = range(90,-91,-10)
data =np.zeros([len(lanes),len(angles)])
i =0
for lane in lanes:
    j=0
    for angle in angles:
        query = f"SELECT COUNT(*) AS target_count FROM targets WHERE yaw = {angl
        cursor = conn.cursor()
        cursor.execute(query)
        results = cursor.fetchall()
        val = [row[0] for row in results]
        data[i,j]=int(val[0])
        j+=1
    i+=1

pd.DataFrame(data,columns=angles)
```

In [ ]:
```
yaw = -90
R=10
expected_pos = np.empty([len(lanes),len(angles)])
i=0
for pos in lanes:
```

```
    j = 0
    for yaw in angles:
        y = 10 - R*np.sin(np.deg2rad(yaw))
        x = 130 + R*np.cos(np.deg2rad(yaw))
        z=-10
        vector_cam = np.array([x,y,z])
        vector_car = np.array([100,2*(pos) , 0])
        diff = vector_car-vector_cam
        expected_pos[i,j] = diff[1]
        j+=1
    i+=1

df = pd.DataFrame(expected_pos,columns=angles)
display(df)
```

Position study

In [ ]:
```
fig, ax = plt.subplots()
yaw =-80
lane = 5
i=lane-1
j = int(abs((yaw-90)/10))
query = f"SELECT xpx,ypx,x,y FROM positions WHERE target_id IN (SELECT id FROM t
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
px_values = [row[0] for row in results]
py_values = [row[1] for row in results]
x = np.array([row[2] for row in results])
y = np.array([row[3] for row in results])+expected_pos[i,j]
# ax.quiver(x[:-1], y[:-1], x[1:]-x[:-1], y[1:]-y[:-1], scale_units='xy', angles
ax.scatter(x, y)

ax.plot([-15,15],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(3), 3))
plt.xlabel('X position')
plt.ylabel('Y position')
plt.title(f"normalized car position for all targets and positions with yaw={yaw}
```

In [ ]:
```
mean_error_y_table = np.ones([len(lanes),len(angles)])
sample_size = np.zeros([len(lanes),len(angles)])

for lane in range(1,10):
    i =lane-1
    for yaw in angles:
        query = f"SELECT x, y FROM positions WHERE target_id  IN (SELECT id FROM
        cursor = conn.cursor()
        cursor.execute(query)
        results = cursor.fetchall()

        x_values = [row[0] for row in results]
        y_values = [row[1] for row in results]
        j = int(abs((yaw-90)/10))
        normalized_y = y_values+expected_pos[i,j]
        if len(x_values)>1:
            mean_error_y_table[i,j]=np.sum(normalized_y)/len(normalized_y)
            sample_size[i,j] = len((normalized_y))
```

```
plt.xlabel('X position')
plt.ylabel('Y position')
df = pd.DataFrame(mean_error_y_table,columns=angles)
display(df)
df = pd.DataFrame(sample_size,columns=angles)
display(df)
```

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(1, 2)

lane = 5
angles1 = range(90,0,-10)
angles2 = range(0,-91,-10)

for yaw in angles1:
    query = f"SELECT x, y FROM positions WHERE target_id  IN (SELECT id FROM tar
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()

    x_values = [row[0] for row in results]
    y_values = [row[1] for row in results]
    i =lane-1
    j = int(abs((yaw-90)/10))

    normalized_y = y_values+expected_pos[i,j]
    ax1.scatter(x_values, normalized_y)
#ax.set(xlim = (-(13), 13), ylim = (-(3), 3))
ax1.set(xlim = (-(13), 13), ylim = (-(3), 3),xlabel='X position [m]',ylabel='Y p
ax1.plot([-16,16],[0,0],"r")
ax1.legend(angles1,loc='upper left')

for yaw in angles2:
    query = f"SELECT x, y FROM positions WHERE target_id  IN (SELECT id FROM tar
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()

    x_values = [row[0] for row in results]
    y_values = [row[1] for row in results]
    i =lane-1
    j = int(abs((yaw-90)/10))

    normalized_y = y_values+expected_pos[i,j]
    ax2.scatter(x_values, normalized_y)
#ax.set(xlim = (-(13), 13), ylim = (-(3), 3))
ax2.set(xlim = (-(13), 13), ylim = (-(3), 3),xlabel='X position [m]',ylabel='Y p
ax2.plot([-16,16],[0,0],"r")
ax2.legend(angles1,loc='upper left')


fig.suptitle(f"normalized position for all studied angles in lane {lane}")
```

Speed Study

In [ ]:
```
base_speed =5
fps =15
print(angles)
expected_speed = np.ones([len(lanes),len(angles)])
print(expected_speed.shape)
with open('videosyaw.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=';')
    for row in spamreader:
        properties = row[0].split("_")
        yaw = float(properties[1])
        lane = int(properties[3].split(".")[0])
        j = int(abs((yaw-90)/10))
        i =lane-1
        expected_speed[i,j] = base_speed*fps*float(row[1])
df = pd.DataFrame(expected_speed,columns=angles)
display(df)
```

In [ ]:
```
mean_error_speed_table = np.ones([len(lanes),len(angles)])
for lane in range(1,10):
    i =lane-1
    j=0
    for yaw in angles:
        query = f"SELECT vx, vy FROM positions WHERE target_id  IN (SELECT id FR
        cursor = conn.cursor()
        cursor.execute(query)
        results = cursor.fetchall()

        vx_values = np.array([row[0] for row in results])
        vy_values = np.array([row[1] for row in results])

        mean_speed = np.sqrt(vx_values**2+vy_values**2)
        normalized_speed = mean_speed-expected_speed[i,j]
        if len(vx_values)>1:
            mean_error_speed_table[i,j]=np.sum(normalized_speed)/len(normalized_
        j+=1
df = pd.DataFrame(mean_error_speed_table,columns=angles)
display(df)
```

EXTRA

In [ ]:
```
def CardanAngles(a):
    return np.array([
        [ np.cos(a[1])*np.cos(a[2]),
        [ np.sin(a[0])*np.sin(a[1])*np.cos(a[2])-np.cos(a[0])*np.sin(a[2]),
        [ np.cos(a[0])*np.sin(a[1])*np.cos(a[2])+np.sin(a[0])*np.sin(a[2]),
    ])
```

In [ ]:
```
fig,ax = plt.subplots()
yaw=-70
lane=5
query = f"SELECT id FROM targets WHERE yaw = {yaw} AND car_lane={lane};"
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
vx_values = [row[0] for row in results]
```

```
query = f"SELECT xpx, ypx,x,y num FROM positions WHERE target_id = {vx_values[2]
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()

px = np.array([row[0] for row in results])
py = np.array([row[1] for row in results])
total = np.vstack((px-320,py-320*0.75))
total = np.vstack((total,np.ones(len(px))))
x = np.array([row[0] for row in results])
y = np.array([row[1] for row in results])
num = np.array([row[2] for row in results])
# for val in range (32,60):
#     print(f"np.array([{px_values[val]-320},{py_values[val]-320*0.75},1]),",ena
ax.quiver(px[:-1], py[:-1], px[1:]-px[:-1], py[1:]-py[:-1], scale_units='xy', ar
#ax.set(xlim = (-(13), 13), ylim = (-(10), 10))
ax.set(xlim = (0, 640), ylim = (0, 320))
ax=plt.gca()                            # get the axis
ax.set_ylim(ax.get_ylim()[::-1])
ax.xaxis.tick_top()                     # and move the X-Axis
```

In [ ]:
```
angles_e=np.array([0,-45,-70])*np.pi/180 # rotation angles
R=CardanAngles(angles_e)
angles_c=np.array([90,0,90])*np.pi/180
R_c=CardanAngles(angles_c) # rotation matrix from body to camera, it is fixed
#camera parameters
f=320
K=np.array([[f, 0, 0],
            [0, f*0.75, 0],
            [0, 0, 1]])

R_inv=np.linalg.inv(R)
R_c_inv=np.linalg.inv(R_c)
K_inv =np.linalg.inv(K)
points =np.array([[0,0,0]])
print(points.shape)
for pos in total.T:
    M = R_inv @ R_c_inv @ K_inv @ pos.T
    s= 10/M[2]
    P_r =M*s + np.array([0,0,-10]).T
    P_r = np.round(P_r,2)
    print(P_r.T)
    points= np.vstack((points, P_r.T))
print(points)
```

In [ ]:
```
angles_e=np.array([0,-45,-70])*np.pi/180 # rotation angles
R=CardanAngles(angles_e)
angles_c=np.array([90,0,90])*np.pi/180
R_c=CardanAngles(angles_c) # rotation matrix from body to camera, it is fixed
#camera parameters
f=320
K=np.array([[f, 0, 0],
            [0, f*0.75, 0],
            [0, 0, 1]])

R_inv=np.linalg.inv(R)
R_c_inv=np.linalg.inv(R_c)
K_inv =np.linalg.inv(K)
points =np.array([[0,0,0]])
```

```
print(points.shape)
for pos in total.T:
    M = R_inv @ R_c_inv @ K_inv @ pos.T
    s= 10/M[2]
    P_r =M*s + np.array([0,0,-10]).T
    P_r = np.round(P_r,2)
    print(P_r.T)
    points= np.vstack((points, P_r.T))
print(points)
```

Preliminary necessities

In [ ]:
```
%matplotlib widget

import mysql.connector
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import csv
pd.options.display.float_format = '{:20,.2f}'.format
plt.close("all")
```

In [ ]:
```
# Open a connection to the MySQL database
conn = mysql.connector.connect(
    host='127.0.0.1',
    user="root",
    password="XXX",
    database="RTVT_h"
)
```

In [ ]:
```
# Execute the SQL query to retrieve the positions
target_id = 34 # Replace with the ID of the target you're interested in
query = f"SELECT x, y FROM positions WHERE target_id = {target_id} ORDER BY `num
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
x_values = [row[0] for row in results]
y_values = [row[1] for row in results]
print((x_values[:10]))
print((y_values[:10]))
```

In [ ]:
```
# Definition of all angles and lanes
#check number of targets for each value
lanes = range(1,10)
heights=list(range(-12,-30,-2))
data =np.zeros([1,len(heights)])
i =0
j=0
for height in heights:
    query = f"SELECT COUNT(*) AS target_count FROM targets WHERE height = {heigh
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    val = [row[0] for row in results]
    data[i,j]=int(val[0])
    j+=1

pd.DataFrame(data,columns=heights)
```

In [ ]:
```
yaw = -90
lane = 5
pitch = -45
heights = np.array(heights)
y = 10 + heights
diff = y-2*5
```

```
print(diff)

#df = pd.DataFrame(expected_y,columns=angles)
#display(df)
```

In [ ]:
```
fig, ax = plt.subplots()
height = -18
query = f"SELECT x,y FROM positions WHERE target_id IN (SELECT id FROM targets W
print(height)
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
x = np.array([row[0] for row in results])
y = np.array([row[1] for row in results])+height
ax.scatter(x,y)
ax.plot([-10,10],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(2), 2))
plt.xlabel('X position [m]')
plt.ylabel('Y position [m]')
plt.title(f"normalized car position for all targets with height={height} and lar
```

Position study

In [ ]:
```
fig, ax = plt.subplots()
for height in heights:
    query = f"SELECT x,y FROM positions WHERE target_id IN (SELECT id FROM targe
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    x = np.array([row[0] for row in results])
    y = np.array([row[1] for row in results])+height
    ax.scatter(x,y)
ax.plot([-10,10],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(1), 1),xlabel='X position [m]',ylabel='Y po
ax.legend(heights,loc='upper left')
plt.title("normalized position for all studied heights")
```

Speed Study

In [ ]:
```
base_speed =5
fps =15
expected_speed = []
with open('videosdist.csv', newline='') as csvfile:
    spamreader = csv.reader(csvfile, delimiter=';')
    for row in spamreader:
        properties = row[0].split("_")
        height = float(properties[2])
        expected_speed.append(np.round(base_speed*fps*float(row[1]),2))
print(expected_speed)
print(len(expected_speed))
```

In [ ]:
```
mean_error_speed_table = []
j=0
for height in heights:
    query = f"SELECT vx,vy FROM positions WHERE target_id IN (SELECT id FROM tar
```

```
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    vx_values = np.array([row[0] for row in results])
    vy_values = np.array([row[1] for row in results])
    mean_speed = np.sqrt(vx_values**2+vy_values**2)
    normalized_speed = mean_speed-expected_speed[j]
    j+=1
    if len(vx_values)>1:
        mean_error_speed_table.append(np.round(np.sum(normalized_speed)/len(norm
    else:
        mean_error_speed_table.append(-1)
print(mean_error_speed_table)
```

Preliminary necessities

In [ ]:
```
%matplotlib widget

import mysql.connector
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import csv
pd.options.display.float_format = '{:20,.2f}'.format
```

In [ ]:
```
# Open a connection to the MySQL database
conn = mysql.connector.connect(
    host='127.0.0.1',
    user="root",
    password="XXX",
    database="RTVT_err2"
)
```

In [ ]:
```
# Execute the SQL query to retrieve the positions
target_id = 34 # Replace with the ID of the target you're interested in
query = f"SELECT x, y FROM positions WHERE target_id = {target_id} ORDER BY `num
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
x_values = [row[0] for row in results]
y_values = [row[1] for row in results]
print((x_values[:10]))
print((y_values[:10]))
```

In [ ]:
```
# Definition of all angles and lanes
#check number of targets for each value
pitchs = range(-40,-51,-1)
base_pitch = -45
yaws = range(-85,-96,-1)
base_yaw = -90
heights = range(-5,-16,-1)
base_height=-10
count_pitch = []
count_yaw = []
count_heigh = []

for pitch in pitchs:
    query = f"SELECT COUNT(*) AS target_count FROM targets WHERE pitch = {pitch}
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    val = [row[0] for row in results]
    count_pitch.append(int(val[0]))
print(count_pitch)

for yaw in yaws:
    query = f"SELECT COUNT(*) AS target_count FROM targets WHERE pitch = {base_p
    cursor = conn.cursor()
    cursor.execute(query)
```

```
    results = cursor.fetchall()
    val = [row[0] for row in results]
    count_yaw.append(int(val[0]))
print(count_yaw)

for height in heights:
    query = f"SELECT COUNT(*) AS target_count FROM targets WHERE pitch = {base_p
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    val = [row[0] for row in results]
    count_heigh.append(int(val[0]))
print(count_heigh)
```

In [ ]:
```
diff = -10 # correct
print(diff)
base_speed =5
fps =15
frame_time =0.15
expected_speed = base_speed*fps*frame_time
print(expected_speed)
```

Pitch Error

In [ ]:
```
fig, ax = plt.subplots()
pitch = -40
query = f"SELECT x,y FROM positions WHERE target_id IN (SELECT id FROM targets W
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
x = np.array([row[0] for row in results])
y = np.array([row[1] for row in results])+diff
ax.scatter(x,y)
ax.plot([-10,10],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(2), 2))
plt.xlabel('X position [m]')
plt.ylabel('Y position [m]')
plt.title(f"normalized car position for all targets with pitch={pitch}")
```

In [ ]:
```
fig, ax = plt.subplots()
for pitch in pitchs:
    query = f"SELECT x,y FROM positions WHERE target_id IN (SELECT id FROM targe
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    x = np.array([row[0] for row in results])
    y = np.array([row[1] for row in results])+diff
    ax.scatter(x,y)
ax.plot([-10,10],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(2), 2))
ax.legend(pitchs,loc='upper left')

plt.xlabel('X position [m]')
plt.ylabel('Y position [m]')

plt.title(f"normalized car position for all targets")
```

In [ ]:
```
mean_error_speed_table = []
j=0
for pitch in pitchs:
    query = f"SELECT vx,vy FROM positions WHERE target_id IN (SELECT id FROM tar
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    vx_values = np.array([row[0] for row in results])
    vy_values = np.array([row[1] for row in results])
    mean_speed = np.sqrt(vx_values**2+vy_values**2)
    normalized_speed = mean_speed-expected_speed
    j+=1
    if len(vx_values)>1:
        mean_error_speed_table.append(np.round(np.sum(normalized_speed)/len(norm
    else:
        mean_error_speed_table.append(-1)
print(mean_error_speed_table)
```

Yaw error

In [ ]:
```
fig, ax = plt.subplots()
yaw = -85
query = f"SELECT x,y FROM positions WHERE target_id IN (SELECT id FROM targets W
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
x = np.array([row[0] for row in results])
y = np.array([row[1] for row in results])+diff
ax.scatter(x,y)
ax.plot([-10,10],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(2), 2))
plt.xlabel('X position [m]')
plt.ylabel('Y position [m]')
plt.title(f"normalized car position for all targets with yaw={yaw}")
```

In [ ]:
```
fig, ax = plt.subplots()
for yaw in yaws:
    query = f"SELECT x,y FROM positions WHERE target_id IN (SELECT id FROM targe
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    x = np.array([row[0] for row in results])
    y = np.array([row[1] for row in results])+diff
    ax.scatter(x,y)
ax.plot([-10,10],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(2), 2))
ax.legend(yaws,loc='upper left')

plt.xlabel('X position [m]')
plt.ylabel('Y position [m]')

plt.title(f"normalized car position for all targets")
```

In [ ]:
```
mean_error_speed_table = []
j=0
for yaw in yaws:
```

```
    query = f"SELECT vx,vy FROM positions WHERE target_id IN (SELECT id FROM tar
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    vx_values = np.array([row[0] for row in results])
    vy_values = np.array([row[1] for row in results])
    mean_speed = np.sqrt(vx_values**2+vy_values**2)
    normalized_speed = mean_speed-expected_speed
    j+=1
    if len(vx_values)>1:
        mean_error_speed_table.append(np.round(np.sum(normalized_speed)/len(norm
    else:
        mean_error_speed_table.append(-1)
print(mean_error_speed_table)
```

Height error

In [ ]:
```
fig, ax = plt.subplots()
height = -12
query = f"SELECT x,y FROM positions WHERE target_id IN (SELECT id FROM targets W
cursor = conn.cursor()
cursor.execute(query)
results = cursor.fetchall()
x = np.array([row[0] for row in results])
y = np.array([row[1] for row in results])+diff
ax.scatter(x,y)
ax.plot([-10,10],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(5), 5))
plt.xlabel('X position [m]')
plt.ylabel('Y position [m]')
plt.title(f"normalized car position for all targets with height={height}")
```

In [ ]:
```
plt.close('all')
fig, ax = plt.subplots()
for height in heights:
    query = f"SELECT x,y FROM positions WHERE target_id IN (SELECT id FROM targe
    cursor = conn.cursor()
    cursor.execute(query)
    results = cursor.fetchall()
    x = np.array([row[0] for row in results])
    y = np.array([row[1] for row in results])+diff
    ax.scatter(x,y)
ax.plot([-10,10],[0,0],"r")
ax.set(xlim = (-(10), 10), ylim = (-(5), 5))
ax.legend(heights,loc='upper left')

plt.xlabel('X position [m]')
plt.ylabel('Y position [m]')

plt.title(f"normalized car position for all targets")
```

In [ ]:
```
mean_error_speed_table = []
j=0
for height in heights:
    query = f"SELECT vx,vy FROM positions WHERE target_id IN (SELECT id FROM tar
    cursor = conn.cursor()
    cursor.execute(query)
```

```
    results = cursor.fetchall()
    vx_values = np.array([row[0] for row in results])
    vy_values = np.array([row[1] for row in results])
    mean_speed = np.sqrt(vx_values**2+vy_values**2)
    normalized_speed = mean_speed-expected_speed
    j+=1
    if len(vx_values)>1:
        mean_error_speed_table.append(np.round(np.sum(normalized_speed)/len(norm
    else:
        mean_error_speed_table.append(-1)
print(mean_error_speed_table)
```