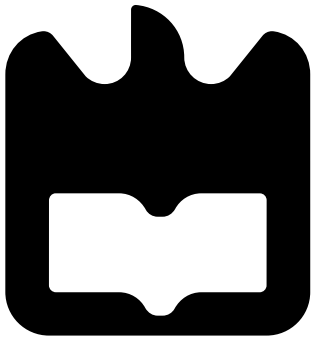
Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2019

Luís Marques Santos

People tracking using drones for Smart Spaces

Seguimento de pessoas com Drones em Espaços Inteligentes



Luís Marques Santos

People tracking using drones for Smart Spaces

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Electrónica e Telecomunicações, realizada sob a orientação científica da Professora Doutora Susana Sargento, Professora Catedrática do Departamento de Electrónica e Informática da Universidade de Aveiro e co-orientação científica do Professor Doutor António José Ribeiro Neves, Professor Auxiliar da Universidade de Aveiro.

o júri / the jury

presidente / president	**Professor Doutor Filipe Miguel Teixeira Pereira da Silva** Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro
vogais / examiners committee	**Professor Doutor Miguel Armando Riem de Oliveira** Professor Auxiliar do Departamento de Mecânica da Universidade de Aveiro
	Professora Doutora Susana Isabel Barreto de Miranda Sargento Professora Catedrática do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (Orientadora)

agradecimentos

Em primeiro lugar queria agradecer à minha mãe por todo o apoio ao longo do meu percurso académico. À professora Susana Sargento por me ter dado a oportunidade de integrar o ambicioso grupo de investigação que é o NAP assim como pelo tempo e paciência dispendidos durante a orientação e consequente criação deste documento. Ao meu co-orientador, o professor António Neves, pelos conselhos e sugestões. Agradeço também ao Dr. André Reis pela disponibilidade, espírito crítico e apoio constante ao longo do desenvolvimento deste trabalho. Finalmente, agradeço ao Nuno Humberto e Bruno Areias pela ajuda crucial nos testes práticos realizados ao longo desta dissertação.
Muito obrigado a todos.

Palavras Chave

Drones, Aprendizagem Automática, Redes Neuronais Convolucionais, Detecção de Objectos, Visão por Computador.

Resumo

O recente progresso tecnológico registado nas últimas décadas no campo da Visão por Computador introduziu novos métodos e algoritmos com um desempenho cada vez mais elevado. Particularmente, a criação de algoritmos de aprendizagem automática tornou possível a detecção de objetos aplicada a feeds de vídeo capturadas em tempo real. Paralelo com este progresso, a tecnologia relativa a veículos aéreos não tripulados, ou drones, também beneficiaram de avanços tanto na miniaturização dos seus componentes de hardware assim como na optimização do software. Graças a essas melhorias, os drones emergiram do seu passado militar e são agora usados tanto pelo público em geral como pela comunidade científica para aplicações tão distintas como fotografia e monitorização ambiental.

O objectivo da presente dissertação pretende tirar proveito destes recentes avanços tecnológicos e aplicar algoritmos de aprendizagem automática de última geração para criar um sistema capaz de realizar seguimento automático de pessoas com drones através de visão por computador.

Para realizar a detecção de objetos, dois algoritmos distintos de aprendizagem automática são apresentados. O primeiro é dotado de uma abordagem baseada em Support Vector Machine (SVM), enquanto o segundo é caracterizado por uma arquitetura baseada em Redes Neuronais Convolucionais. Ambos os métodos serão avaliados usando uma base de dados de imagens criada para os propósitos da presente dissertação.

As avaliações realizadas relativas ao desempenho dos algoritmos de detecção de objectos demonstraram que o método baseado numa arquitetura de Redes Neuronais Covolucionais foi o melhor tanto em termos de tempo de processamento médio assim como na precisão das detecções, revelando-se portanto, como sendo o método mais adequado de acordo com os objectivos pretendidos.

O sistema desenvolvido foi testado num contexto real, com os resultados obtidos a demonstrarem que o sistema é capaz de realizar o seguimento de pessoas a velocidades comparáveis a um ritmo normal humano de caminhada.

keywords Drones, Machine Learning, Convolutional Neural Networks, Object Detection, Computer Vision.

Abstract Recent technological progress made over the last decades in the field of Computer Vision has introduced new methods and algorithms with ever increasing performance results. Particularly, the emergence of machine learning algorithms enabled class based object detection on live video feeds. Alongside these advances, Unmanned Aerial Vehicles (more commonly known as drones), have also experienced advancements in both hardware miniaturization and software optimization. Thanks to these improvements, drones have emerged from their military usage based background and are now both used by the general public and the scientific community for applications as distinct as aerial photography and environmental monitoring.

This dissertation aims to take advantage of these recent technological advancements and apply state of the art machine learning algorithms in order to create a Unmanned Aerial Vehicle (UAV) based network architecture capable of performing real time people tracking through image detection.

To perform object detection, two distinct machine learning algorithms are presented. The first one uses an SVM based approach, while the second one uses an Convolutional Neural Network (CNN) based architecture. Both methods will be evaluated using an image dataset created for the purposes of this dissertation's work.

The evaluations performed regarding the object detectors performance showed that the method using a CNN based architecture was the best both in terms of processing time required and detection accuracy, and therefore, the most suitable method for our implementation.

The developed network architecture was tested in a live scenario context, with the results showing that the system is capable of performing people tracking at average walking speeds.

Contents

List of Figures

List of Tables

Acronyms

AP	Average Precision
API	Application Programming Interface
ANN	Artificial Neural Networks
CNN	Convolutional Neural Network
FC	Fully Connected
FIFO	First in First out
FN	False Negatives
FP	False Positives
GPU	Graphics Processing Unit
HOG	Histogram Oriented Gradients
HTTP	Hypertext Transfer Protocol
IoU	Intersection over Union
JSON	JavaScript Object Notation
mAP	Mean Average Precision
ReLU	Rectified Linear Unit
REST	Representational State Transfer
SIFT	Scale-Invariant Feature Transform

SURF	Speeded Up Robust Feature
SVM	Support Vector Machine
TN	True Negatives
TP	True Positives
UAV	Unmanned Aerial Vehicle
VTOL	Vertical Take-off and Landing
XML	Extensible Markup Language
YOLO	You Only Look Once

Chapter 1

Introduction

Unmanned Aerial Vehicle (UAV)s, more commonly known as drones, have benefited from recent technological advances in the last few decades that contributed towards their miniaturization and optimization. This progress expanded their utility and usage into a wide range of possible applications for both scientific purposes (e.g.: atmospheric monitoring and data gathering, topographic monitoring of coastal areas, oceanographic research) as well as more casual uses for the general public (e.g.: photography). This recent development presents a paradigm shift from the inception of UAVs that, previously, focused mainly on military applications.

Parallel to the UAVs development over the years, the fields of image classification and object detection have also seen great strides, with the advent of machine learning algorithms that leverage huge image databases to produce fast and accurate computer based image assessment.

The present work aims to take advantage of both UAV's characteristics and state of the art machine learning algorithms to implement a network architecture capable of real-time person tracking through image detection using UAVs.

1.1 Objectives

The main goal of this dissertation is the implementation of a network architecture capable of conducting real-time person tracking through image detection using UAVs. With this goal in mind, the present dissertation has the following objectives:

- Design the overall elements of the architecture required to gather live video data from

an airborne UAV, perform object detection on said data and issue drone commands based on this input so as to correctly track a person present on an open terrain;

- Evaluate current state of the art object detectors and their suitability to perform real-time person detection on a given video feed;
- Create the UAV control logic based around the constraints found when performing object detection on the live video footage gathered by an airborne UAV;
- Evaluate the overall performance of the proposed solution in real world scenarios;

1.2 Contributions

The work developed in this dissertation led to the following contributions:

- Creation of an image Database comprised of frames depicting people on an open terrain environment and shot from the point-of-view of an airborne UAV, complete with boundary box labels for each person depicted;
- Comparison of two machine learning based object detectors, one using the SVM approach and the other one using a Convolutional Neural Network (CNN) architecture, when performing person detection on footage recorded by an airborne UAV;
- Development of a REST based server capable of performing person detection on an input video feed and issue movement based commands over to an active UAV in order to perform a tracking mission. The commands are based on the UAV platform designed by Bruno Areias et al. [24].
- Experimentation of a multi-technology network approach for the real-time image detection and drone control.

1.3 Document Structure

This section outlines the structure of this dissertation.

- **Chapter 1 -** Provides a brief description of this dissertation's work along with the context, motivation and objectives;

- **Chapter 2 -** Presents the advancements in areas relevant to this dissertation, covering drones, current machine learning algorithms used for object detection and the methodology used to evaluate their performance;
- **Chapter 3 -** Contains the description of the proposed architecture, covering each element present and its respective description;
- **Chapter 4 -** Contains the algorithm description and implementation used for issuing commands to the UAV based around the output from an object detector;
- **Chapter 5 -** Presents the results obtained with regards to the performance of two different types of machine learning based object detectors when used solely for detecting people from images gathered by an airborne UAV. It also provides the final results obtained when conducting tracking experiments using the proposed solution;
- **Chapter 6 -** Presents the conclusions of this dissertation and proposes additional improvements that can be implemented in order to enhance the overall performance of the proposed architecture.

Chapter 2

State of the Art

This chapter introduces the main concepts and topics involved in this dissertation's work. Firstly, there is a brief discussion about drones and their usage in today's landscape.

An overview on methods of object detection in the field of image processing is then followed. The main approaches discussed follow the most recent trends in this field and will focus mainly on deep learning methodologies developed over the last recent decades with a primary focus on the approaches using Convolutional Neural Network (CNN) and Support Vector Machine (SVM).

2.1 Drones

Ever since the first documented flight accomplished in 1903 by the Wright brothers, drones or Unmanned Aerial Vehicle (UAV)'s were seen as a possibility for scientists and engineers at the time. The ability of controlling an aircraft remotely and without an on-board pilot opened a wide range of possible applications. UAV technology took its first steps during the first World War [25]. The ability to transmit real-time intelligence, surveillance, and reconnaissance information from hostile territory made these types of technology very appealing to military leaders. This lead to continued financial backing from governments to pursue new, more advanced prototypes. Nowadays, thanks to: (1) new advances; (2) UAV's characteristics of small size; (3) strong mobility; (4) low communication overhead; (5)the emergence of new sensors with improved geometric and radiometric resolution; (6) new platforms that provide robustness and increased autonomy; (7) new software developed that ranges from navigation and communications; and (8) the processing and analysis of

the data gathered by sensors [26], made a whole new host of applications in several different possible fields, as explained below.

Agriculture Estimate harvest volumes using digital images collected by UAVs [27]. Crop monitorization, analyses and protection [28] [29].

Surveillance UAVs are deployed in either coastal or land borders in order to perform patrol missions [30].

Environmental Monitoring UAVs are used to monitor harmful gas concentrations in the atmosphere [31].

Atmospheric Data Gathering UAVs are used to retrieve various types of localized atmospheric readings [32].

Figure 2.1: Fixed wing UAV. Figure 2.2: Single-rotor UAV. Figure 2.3: Multi-rotor UAV.

UAVs can vary greatly in shape and size depending on the applications' requirements where the given UAV is deployed. The type of technology used to keep the drone airborne determines the type of drone. The three main types of drones are: fixed-wing, single-rotor and multi-rotor, depicted in Figures 2.1, 2.2 and 2.3 respectively.

A brief summary of the different types of UAVs and their main characteristics is shown in Table 2.1.

2.2 Image Processing

Computer vision took its first steps in the 1970s. Back then it was considered that solving the "visual input" problem would be an easy step along the path to solving more

Table 2.1: Types of UAVs and corresponding features (adapted from [22]).

	Pros	**Cons**	**Typical Uses**
Multi-Rotor	• Accessibility • Ease of use • Vertical Take-off and Landing (VTOL) and hover flight • Good camera control • Can operate in a confined area	• Short flight times • Small payload capacity	• Aerial photography and video • Aerial inspection
Fixed-Wing	• Long endurance • Large area of coverage • Fast flight speed	• Requires take-off and landing area • No VTOL/hover • Harder to fly, more training required • Expensive	• Aerial mapping • Pipeline and Power line inspection
Single-Rotor	• VTOL and hover flight • Long endurance • Heavier payload capability	• Less reliable • No VTOL/hover • Harder to fly, more training required • Expensive	• Aerial LIDAR laser scanning • Pipeline and Power line inspection

difficult problems such as higher-level reasoning and planning [1]. However, this task has proven to be more difficult to solve than previously imagined.

Figure 2.4 shows a brief overview of the most significant advances in the field of computer vision over the last few decades.

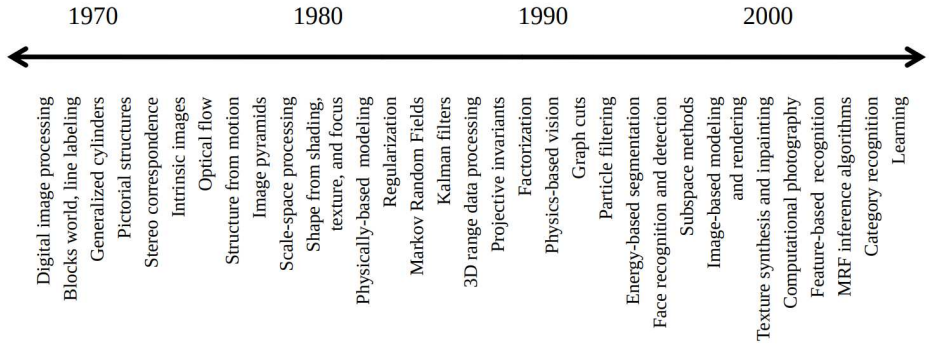


Figure 2.4: A timeline overview of some of the most active topics of research in computer vision (from [1]).

The efforts that were made pushed the boundaries of computer vision to new frontiers. Each new step brought us closer to what is possible nowadays, with object detection being used in different areas such as Automated Parking Systems [33] (shown in Figure 2.5) and detection of tumors [3] (shown in Figure 2.6).

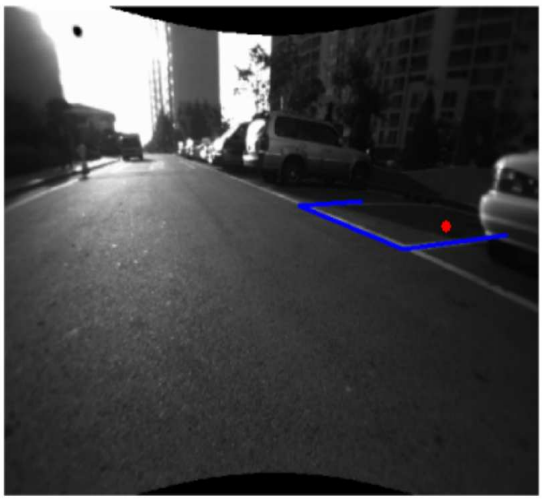


Figure 2.5: Object detection used in parking application (from [2]).

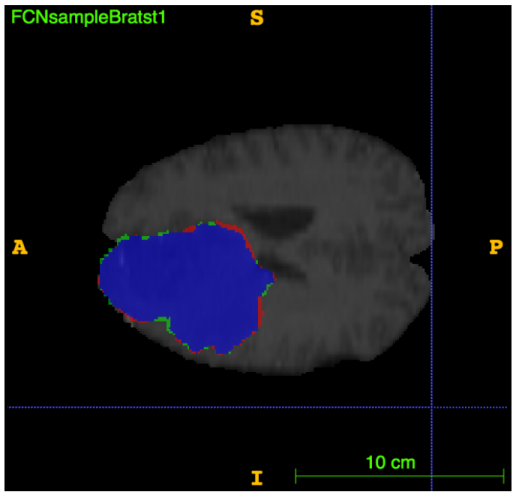


Figure 2.6: Brain Tumor Segmentation in MRI Images (from [3]).

2.2.1 Background

One of the most sought-after goals in the field of computer vision is the analysis of a given image, and the recognition and labelling all of the objects present in the input image (Figure 2.7).

The difficulty behind this problem arises from the fact that, in the real world, objects are usually, not arranged in an organized manner and separated from each other. They can appear to have different shapes and sizes depending on the perspective. Furthermore, each class belonging to a specific object (e.g., dogs, chairs, people...) has an intrinsic variability. For example, in the object class belonging to birds there can be extreme variations in size and shape due to the many different types of bird breeds that exist (shown in Figure 2.8). This makes it unlikely that we can simply perform exhaustive matching against a database of exemplars [1].

The most challenging aspect in object recognition in computer based algorithms arises from this fact alone: to determine whether a certain type or class of object is present (e.g., dogs, people, cars...), it is needed to take into account all the different factors (e.g., lighting conditions, object orientation, deformation, class variability...) and context behind the analyzed image: this is where Artificial Neural Networks (ANN) have played a major role in tackling these problems over the most recent years.

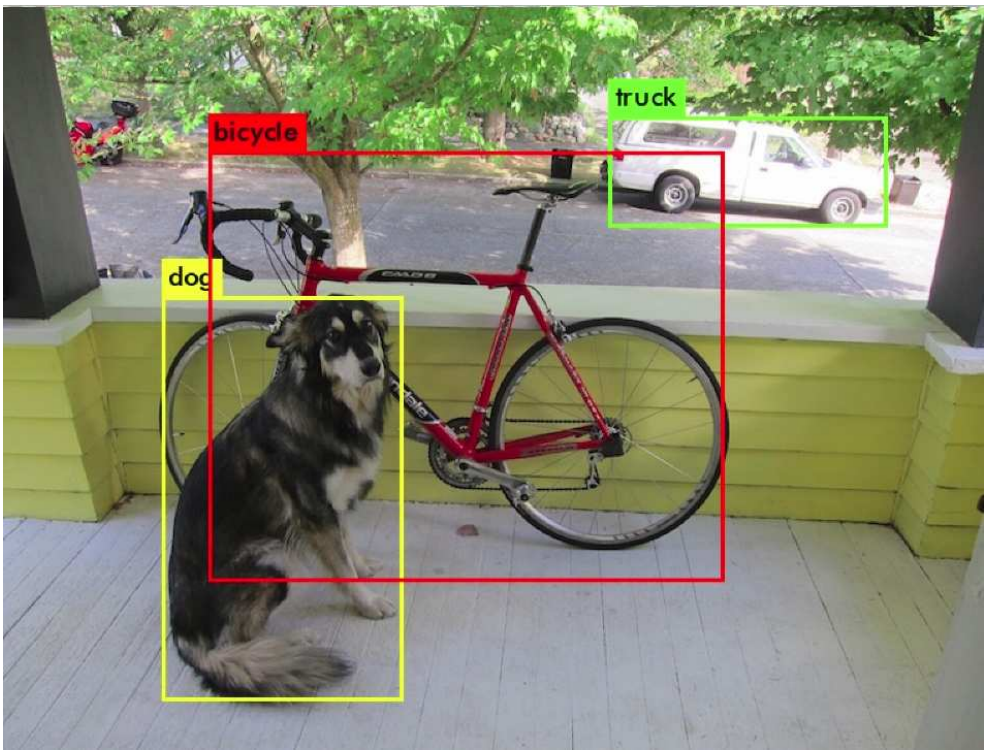


Figure 2.7: Example of Object Detection (from [4]).

2.2.2 ImageNet challenge

The ImageNet challenge [34] was created in a collaboration between the Standford and Princeton Universities, and was first presented in 2009 at the Conference on Computer Vision and Pattern Recognition. It provides a large database composed of millions of images where each one depicts a class of object and is labelled accordingly. In it, at least one million images bounding boxes are also provided, pertaining to the pixel position of the object shown in the image.

The challenge was originally a classification task where the final objective is to identify the object present in the input image. In this case, the input images represent only a single object belonging to one class (e.g., dogs, person, chair). The challenge has evolved to further include a multi-classification task where the objective is to label each object shown with the respective class and bounding box.

Traditional methods of image classification were first applied to tackle this challenge. However, in 2012 a significant performance boost was achieved using Deep Neural Networks [35]. The results showed an error rate of 15.3%, outperforming considerably the

Figure 2.8: Birds shown as an example of intra-class variability (from [5]).

second best contest entry that achieved an error rate of 26.2%.

2.2.3 Deep Neural Networks

The term Deep Learning or Deep Neural Network refers to ANN with multiple layers. ANNs are commonly used in artificial intelligence and machine learning applications. Its importance in pattern recognition has grown over the last decades [36]. Deep neural networks were first inspired by the fields of Neuroscience or Biology and Mathematics. More specifically, ANN emulate what happens in the visual cortex when it receives sensory input from the eyes. It was shown in an experiment designed by D.H. Hubel and T.N. Wiesel [37] that certain individual cells, present in the visual cortex, only fire when exposed to certain patterns present in the image being perceived (these patterns include, for example, vertical or horizontal edges). The notion of specialized components inside a network that respond to specific patterns is an integral part of the theory behind Deep Neural Networks. The first approach at achieving a computational model of neural networks was introduced by the neurophysiologist Warren McCulloch, and by the mathematician Walter Pitts in 1943 [38]. The model proposed ushered in a new field of study: ANN. ANNs are composed of an interconnected set of artificial neurons, Figure 2.9 depicts a simple model of one such neuron.

The artificial neurons are responsible for using the inputs received by their neighbours and obtain an output signal to be propagated to other neurons. A basic artificial neuron is comprised of a set of inputs $\mathbf{x}_i$, each with an associated weight $\mathbf{w}_i$ and a corresponding bias θ. The output $\mathbf{y}$ is obtained by applying an activation function to the weighted sum between inputs and their associated weights, and is expressed in Equation 2.1:

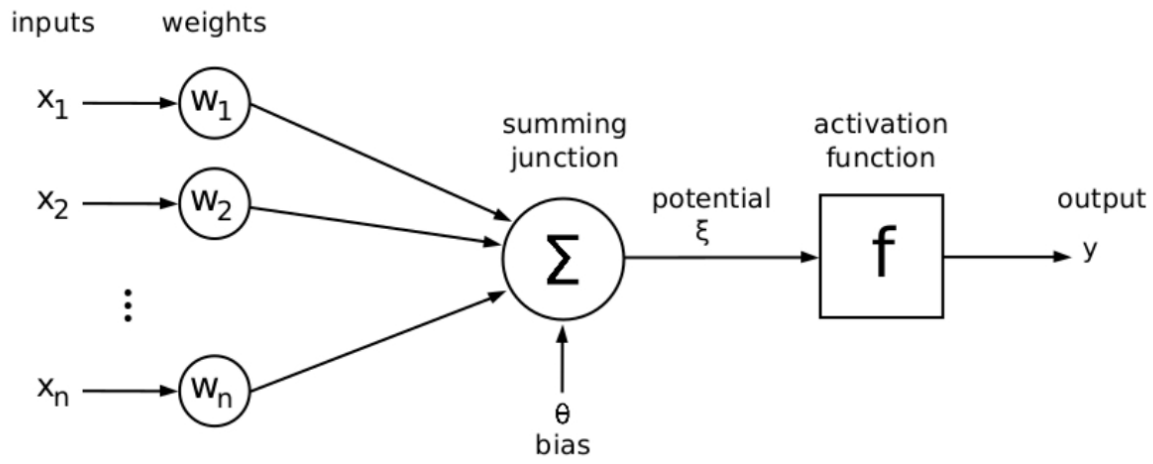


Figure 2.9: Model of an artificial neuron (from [6]).

$$y = f(\xi) = f(\sum_{i=1}^{n} x_i.w_i + \theta) \tag{2.1}$$

There are two different types of ANN topologies: Feedforward and Feedback [7]. On Feedforward ANN the output signal is always sent over to the next layer of neurons; in other words, the data flow is unidirectional. By contrast, in Feedback ANN neurons are able to use data from succeeding layers. Figures 2.10 and 2.11 depict examples of a Feedforward and Feedback topologies respectively.

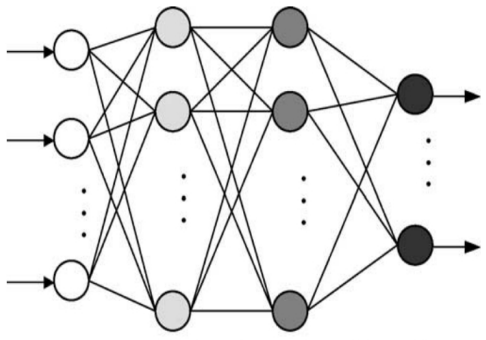


Figure 2.10: Example of a Feed Forward topology ANN (from [7]).

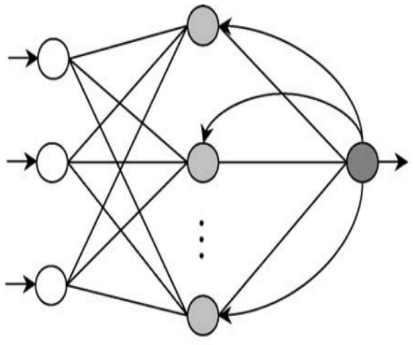


Figure 2.11: Example of a Feedback topology ANN (from [7]).

When dealing with object recognition in images, one specific type of Deep Neural

Networks has been shown to be particularly efficient: Convolutional Neural Networks (CNNs). When compared to standard feed forward neural networks with similarly-sized layers, CNNs have much fewer connections and parameters, and so, they are easier to train, while their theoretically best performance is likely to be only slightly worse [35].

2.2.3.1 Convolutional Neural Networks

CNNs are multi-layer feed-forward networks. Their architecture is very similar to the architecture of ANNs discussed in the previous chapter: they are comprised of neurons that have learnable weights and biases. Each neuron has a set of inputs used to obtain that output given by performing a dot product between them followed, in some cases, by a non-linearity function.

Typical applications when designing these types of networks consist of object recognition, however, CNNs have also been successfully designed for various different types of tasks [39] [40].

The main building blocks behind the architecture of CNNs are convolutions, hence their name. In CNNs convolutions are mainly used to perform feature extraction on the input image. A two-dimensional convolution can be expressed by:

$$f[m,n] \circledast g[m,n] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i,j].g[m-i,n-j] \tag{2.2}$$

In CNNs the matrices used to perform the convolutional operation and, consequently, feature extraction are very small when compared to the inputted data thus preserving the spacial relationship between pixels. These matrices can be designated as: kernels, filters or feature extractor. The convolution operation is achieved by sliding the kernel along the input matrix and obtaining the dot product as shown in equation 2.2. An example of a filter matrix for horizontal edge detection applied to an image and the corresponding feature map is depicted in Figure 2.12.

Each filter has a specific set of weights attributed to it. The weights are treated as neuron parameters while the convolution operation replaces the logical operations of a regular ANN. These weights determine which features are to be detected on the input data. Changing these values produces different feature maps and, consequently, produces different outputs between each layer of the CNN. During the training process of one CNN these values are adjusted between each iteration, to minimize the error rate at the output.

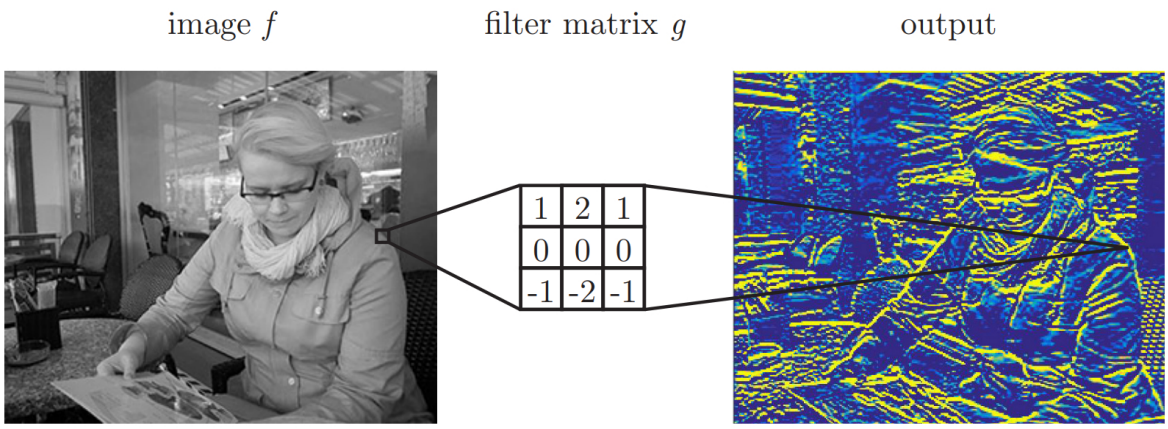


Figure 2.12: Examples of 2D image filters (adapted from [8]).

Sets of different filters can be combined to form a convolutional layer. Besides convolutional layers, a CNN architecture is usually comprised of the following layers:

Input Layer Composed of the raw pixel data from the image (usually a 3 dimension matrix corresponding to the three color channels R,G,B).

Convolutional Layer This layer is made up of a set of filters whose parameters are learnable. These filters convolve across the width and height of the input matrix. The result is an activation matrix that shows the presence, or not, of a certain feature intrinsic to that filter. These features typically correspond to simple shapes in the low level Convolutional layers (e.g. edges with a specific orientation), to more complex ones in the high level layers (e.g. roundness patterns).

Pool Layer Given an input matrix this layer performs a downsampling across the height and width of the matrix.

Fully-Connected Layer This layer outputs each class confidence score, corresponding to the level of certainty that the network has about the fact that the object of class X is present in the image.

A simple overview of a CNN is shown in Figure 2.13.

The Network receives an input image (3 dimensional vector: width, height, channels). This input is then transformed through a set of hidden layers, changing the shape and size

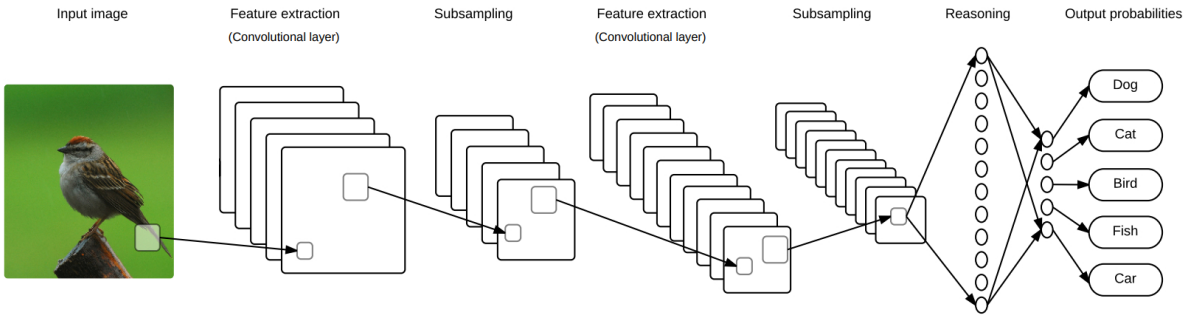


Figure 2.13: Example of a Convolutional Neural Network (from [9]).

of the original data to obtain an output vector consisting of each class scored according to the level of certainty about the fact that the class is present in the input image.

Each layer has an input volume and outputs a certain volume to the next layer. The size of each output volume is dependent on the layers' hyperparameters:

Depth This first parameter corresponds to the number of filters present in the layer. These filters have different goals, each producing different feature maps. They can be used to detect edges, gradients, downsample the input, etc.

Stride This represents the number of pixels when shifting the filter across the input. Increasing this parameter will reduce the output volume and reduce computational effort at the cost of possibly decreasing the overall accuracy of the network.

Zero-Padding This final parameter is used to signal the number of zeros added, or not, around the input volume. It is commonly used to obtain an output volume equal to the input volume when applying the layers' filters.

Each convolutional layer is preceded by a Rectified Linear Unit (ReLU) operation, and can be expressed by:

$$f(x) = max(0, x) \tag{2.3}$$

The purpose behind this operation is to introduce nonlinearity to the network since, without this, the network would consist of only computing linear operations.

Previous work used the nonlinear functions *tanh* and *sigmoid*, but recently researchers found that ReLU operators work better because the network is able to train faster (because

of the computational efficiency) without making a significant difference to the accuracy benchmarks [41].

A strong advantage behind CNN is the fact that convolutions are an integral part of computer graphics (at the hardware level, convolutions are implemented in the GPU), making this task very efficient and fast.

2.2.3.2 Examples of Convolutional Networks

You Only Look Once (YOLO) is an example of an implemented CNN. It presents a new approach to object detection when compared to previous attempts made in the field of object recognition. Prior detection systems repurpose classifiers or localizers to perform detection. They apply the model to an image at multiple locations and scales in order to detect a wether a certain object is present. High scoring regions of the image are considered detections. Instead, YOLO uses a different approach where a single neural network is applied to the full image. The YOLO network divides the input image into grids and proceeds to predict bounding boxes and probabilities associated to a given class for each grid element as shown in Figure 2.14. These bounding boxes are then weighted by the predicted probabilities to infer the location of each detection [11].

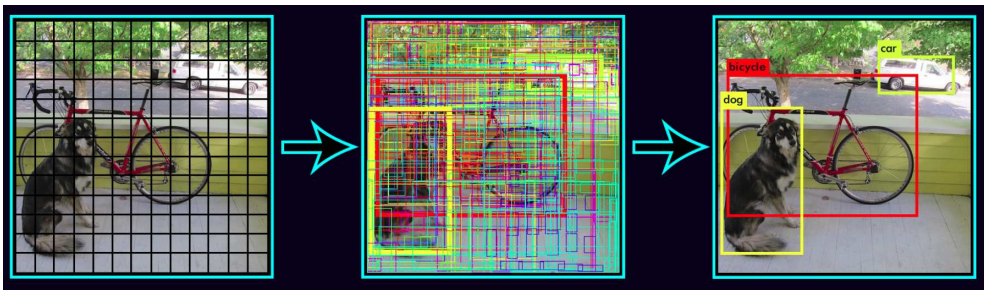


Figure 2.14: Visual representation of YOLO network output (from [10]).

Figure 2.15 shows the architecture present in the YOLO network. This network consists of 24 convolutional layers followed by 2 Fully Connected (FC) layers.

The initial convolutional layers of the network are used to extract low level features from the image while the FC layers predict and output conditional probabilities and coordinates for each class.

The images used when training this CNN belong to the PASCAL VOC2007 detection project [12]. Resources provided by this project are the following:

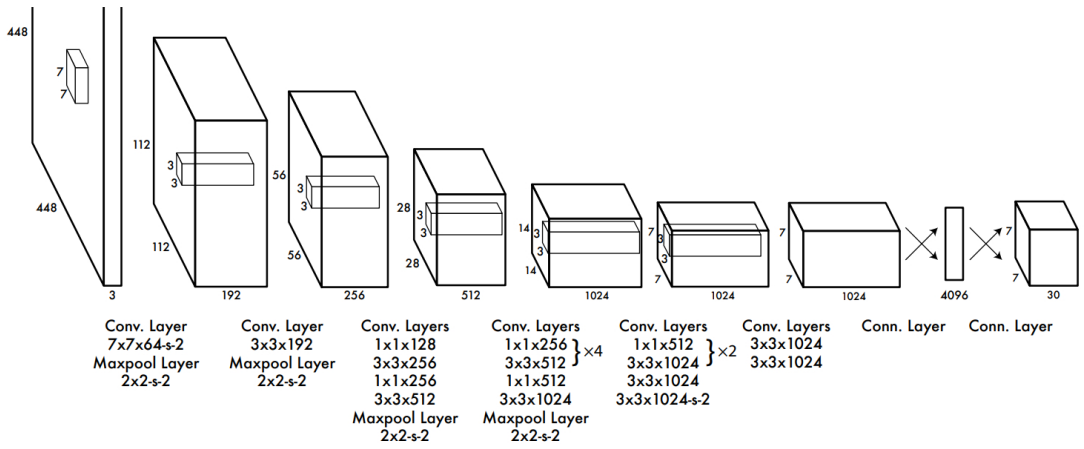


Figure 2.15: Example of a Convolutional Neural Network Architecture (from [11]).

- Standardized image data sets for object class recognition.
- Common set of tools for accessing the data sets and annotations.
- Enables evaluation and comparison of different methods.

Figure 2.16 shows a description of the contents present in the provided image dataset.

2.2.4 SVM

Initial work done to tackle the detection of types of objects in static images, such as: traffic signs [42] and human faces [43], used template matching approaches where a set of rigid templates or handcrafted parameterized curves are used to determine whether a type of object is present in the given image. These methods, although intuitive and simple, are difficult to implement when the type of object intended to be detected and its background becomes more complex. This happens because these methods require a significant amount of prior information and domain knowledge [44]. To account for the increase in complexity of detection described previously, new methods were proposed that use learning-based algorithms. These new implementations leverage large sets of data to achieve higher levels detection accuracy in more complex imagery. One such implementation that showed promising results in the field of object detection uses SVM [45].

	train		val		trainval		test	
	img	**obj**	**img**	**obj**	**img**	**obj**	**img**	**obj**
Aeroplane	112	151	126	155	238	306	204	285
Bicycle	116	176	127	177	243	353	239	337
Bird	180	243	150	243	330	486	282	459
Boat	81	140	100	150	181	290	172	263
Bottle	139	253	105	252	244	505	212	469
Bus	97	115	89	114	186	229	174	213
Car	376	625	337	625	713	1,250	721	1,201
Cat	163	186	174	190	337	376	322	358
Chair	224	400	221	398	445	798	417	756
Cow	69	136	72	123	141	259	127	244
Dining table	97	103	103	112	200	215	190	206
Dog	203	253	218	257	421	510	418	489
Horse	139	182	148	180	287	362	274	348
Motorbike	120	167	125	172	245	339	222	325
Person	1,025	2,358	983	2,332	2,008	4,690	2,007	4,528
Potted plant	133	248	112	266	245	514	224	480
Sheep	48	130	48	127	96	257	97	242
Sofa	111	124	118	124	229	248	223	239
Train	127	145	134	152	261	297	259	282
Tv/monitor	128	166	128	158	256	324	229	308
Total	2,501	6,301	2,510	6,307	5,011	12,608	4,952	12,032

Figure 2.16: VOC2007 Image Dataset description (from [12]).

SVMs were first described by Vapnik and collaborators in 1992 [46]. SVM, in machine learning, are supervised learning models that are based on statistical learning theory [47]. These models have recently been successfully applied to classification and regression problems, leading to applications in various fields. Examples of these applications are: object recognition [48], handwritten digit recognition [49], speaker identification [50], face detection in images [51], text categorization [52] and many others. When compared to neural networks, SVMs are more intuitive and generally easier to implement, with the latter method being more opaque when trying to recognize the "intention" behind each neuron's function and the connections formed between them.

The general principle behind SVMs, when applied to object detection, uses feature vectors as points in a higher dimensional space to try and find planes that "slice" the volume in such a way as to optimally separate between different classes of objects present in the training data. The feature vectors can originate, in the field of image classification, through Histogram Oriented Gradients (HOG), wavelets, Scale-Invariant Feature Transform (SIFT), Speeded Up Robust Feature (SURF), etc. A simplified representation of the idea described is shown in Figure 2.17.

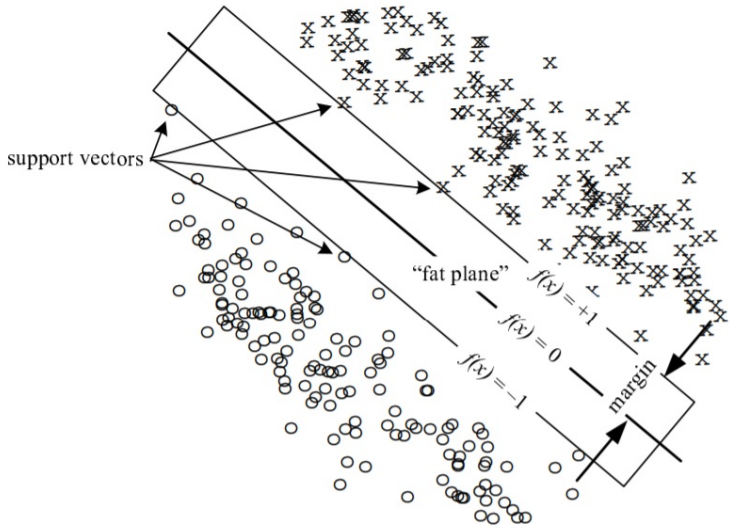


Figure 2.17: Visual representation of a SVM dataset containing 2 different classes (from [13]).

In the example shown, the data points consist of two different classes, one class represented by crosses and the other one by circles. Each point has 2 corresponding features, leading to the 2D mapping shown. The support vectors, where the name SVM is derived from, are the ones found closest to the hyperplane, forming the "frontier" between the two different classes.

In order to perform the task of classification optimally, there needs to exist an n-dimensional (corresponding to n different features) space where the different classes of objects are clearly separated. To achieve this, during the training phase, different types of feature extractors and image correction schemes are used. An example showing this concept is shown in Figure 2.18.

Following this phase, the next step is to find the most suitable hyperplane that separates the different classes in the most optimal way. This optimal hyperplane corresponds to the one that maximizes the distance between the two classes. This characteristic allows fewer classification errors when dealing with inputs that are near the hyperplane and thus leading to better accuracy. Depending on which methods are used during the development of the SVM model, the model can be either linear or non-linear. Once this is achieved, it is possible to map out any input data to the given class prediction. This is done simply by

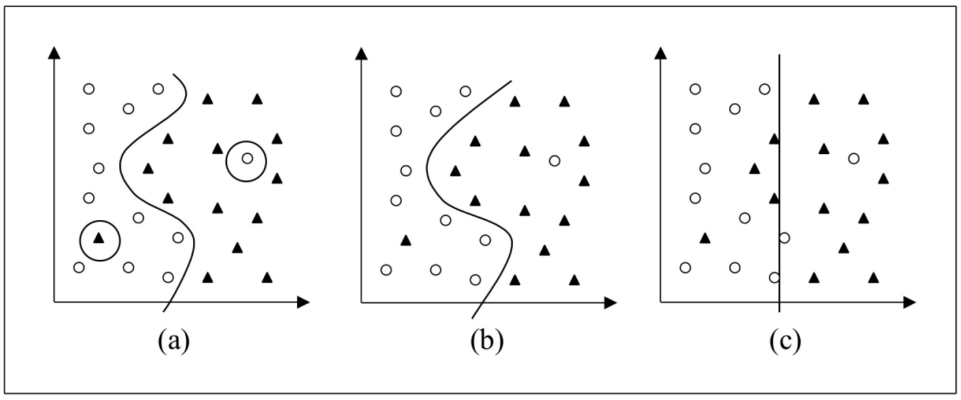


Figure 2.18: Examples of different 3 different SVM models using the same data(from [14]).

using the mathematical expression that encodes the extracted feature vector, from the input data, to the n-dimensional plane and the associated distance in relation to the hyperplane. When testing if the model is accurate, besides being able to predict the training data, the model must also be able to correctly predict new data fed into it. In cases where the developed model shows a high level of accuracy when predicting the training data but does not accomplish the same for non-training data leads to a model that suffers from overfitting. This is usually due to either a small set of training data, too many feature descriptors or a combination of the two. Underfitting is also a possible result; this occurs when the training samples under represent the scope of possible class variation or when the model is too simple.

Extrapolating these notions to feature vectors in higher dimensions: $f : \mathbb{R}^N \rightarrow \{-1, +1\}$, where n is the number of features present, and the value of the function refers to each class to be classified by the model.

2.3 Evaluation of Object Detection Algorithms

The task of evaluating the performance of object classifiers is a simple one since these classifiers are only required to predict whether a certain class of objects is present or not in the input image. When performing a classification on a given object class, we can obtain 4 different types of results at the output:

True Positives (TP) True positives represent the instances where the predicted class is equal to the actual class present in the image.

True Negatives (TN) True negatives represent the instances when the class is not present in the image and the classifier, correctly, does not predict that class.

False Positives (FP) False positives represent the instances where the predicted class is not present in the image.

False Negatives (FN) False negatives represent the instances where an image containing the object class is, incorrectly, predicted as having none.

Using these definitions, we can evaluate the object classifier in terms of: accuracy, precision, recall and specificity. Accuracy corresponds to the models' correct predictions compared to the total number of possible correct predictions, and it can be expressed by,

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.4}$$

Precision is a measure that shows the proportion of positive predictions that are actually true expressed by,

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

Recall measures the ratio of true object detections to the total number of objects in the data set, and is expressed by,

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

Since object detectors are tasked with assigning, in an image, a bounding box to the predicted object there needs to be a new metric designed to evaluate its precision. To achieve this, the set of image data must first contain correct labels in each image with the bounding box values for each class. These bounding box values are usually hand labelled and referred to as ground truth. The most widely metric used to evaluate the predicted bounding box level of accuracy is Intersection over Union (IoU). This metric measures the area of overlap between the predicted bounding box B_p and the ground truth bounding box B_{gt} in relation to their area of union [53], as shown in Figure 2.19, according to the formula:

$$IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{2.7}$$

Figure 2.19: Example of an IoU from an arbitrary Object Detector (from [15]).

This metric can be used to evaluate a given Object Detector performance by establishing a minimum value of IoU required for a TP to be validated.

Using the concepts described, we can evaluate a given algorithm in terms of Average Precision (AP). AP computes the average precision for equally spaced recall values that range from 0 to 1 evaluated for a particular class of objects. For example, in the VOC2007, the AP is calculated as the mean precision at a set of eleven equally spaced recall levels [0,0.1,0.2,...,1], using the formula [54]:

$$AP = \frac{1}{11} \sum_{r \epsilon \{0,0.1,...,1\}} p_{interp}(r) \tag{2.8}$$

The precision at each recall value of r is interpolated by taking the maximum precision obtained for a method for which the corresponding recall exceeds r:

$$p_{interp}(r) = \max_{\widetilde{r}:\widetilde{r} \geq r} p(\widetilde{r}) \tag{2.9}$$

where $p(\widetilde{r})$ is the measured precision at recall $\widetilde{r}$. This method of evaluation proposed in [55] is the most widely used method for evaluating Object Detectors with slight variations on the formula being used in different visual object recognition research papers. Finally, when evaluating, for a dataset comprised of multiple different classes, the AP scores obtained for each class are averaged out to obtain the *Mean Average Precision (mAP)*. The mAP score measures the overall performance accuracy across multiple types of classes Figure 2.20 shows the mAP scores obtained in the PASCAL VOC 2012 competition with the results obtained by the YOLO network being highlighted.

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	**73.9**	**85.5**	**82.9**	**76.6**	**57.8**	**62.7**	**79.4**	77.2	86.6	**55.0**	**79.1**	**62.2**	87.0	**83.4**	**84.7**	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	**79.8**	87.7	49.6	74.9	52.1	86.0	81.7	83.3	**81.8**	**48.6**	**73.5**	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
Fast R-CNN + YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	**89.4**	49.4	75.5	57.0	**87.5**	80.9	81.0	74.7	41.8	71.5	68.5	**82.1**	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	**68.8**	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	**87.5**	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

Figure 2.20: PASCAL VOC 2012 Leaderboard(from [11]).

2.4 Summary

This Chapter provided a brief overview to the field of image processing with a special focus in machine learning algorithms used in object detection tasks. The two main algorithms discussed were the ones that implement the SVM and CNN approaches. These methods will be the ones used in this dissertation's architecture to perform person detection, and will be explored more thoroughly in Chapter 5 in order to decide which one performs this task more efficiently.

Furthermore, an introduction to UAV technology was presented, characterizing the different types of UAVs available to today's developers, and the different types of applications in which they are inserted.

Chapter 3

Proposed Architecture

This chapter provides an insight on the architecture implemented for both the drone and ground systems in order to achieve the goal set out of using an Unmanned Aerial Vehicle (UAV) to track a person through the video captured on the on-board camera. Section 3.1 presents an overview of the proposed network architecture along with the main challenges it faces. Section 3.2 describes the main hardware and software components of the architecture.

3.1 Scenario Overview

The aim of this work is to build an architecture capable of conducting real-time person tracking with UAVs through image detection. The architecture proposed is based on an UAV platform [24] which already contains features such as UAV control and monitoring implemented for multi-rotor UAVs. The scenario envisioned, depicted in Figure 3.1, consists of a drone equipped with a camera that gathers video while airborne. This video is streamed to a server located the ground side of the architecture. Since video streaming requires the use of a significant amount of bandwidth, WiFi is chosen to deliver the video to the server located on the ground. This server will produce the positional commands to be sent to the active UAV based on the relative position of the person inside each frame. The resulting commands are issued over to the drone platform presented in [24] that proceeds to communicate them over to the UAV through cellular technology. The use of cellullar technology by the drone platform to monitor and control UAVs is meant to prevent them from being in a situation where they find themselves out of range. These commands aim to

continuously update the UAV's position relative to the person targeted, so as to effectively track its movement.

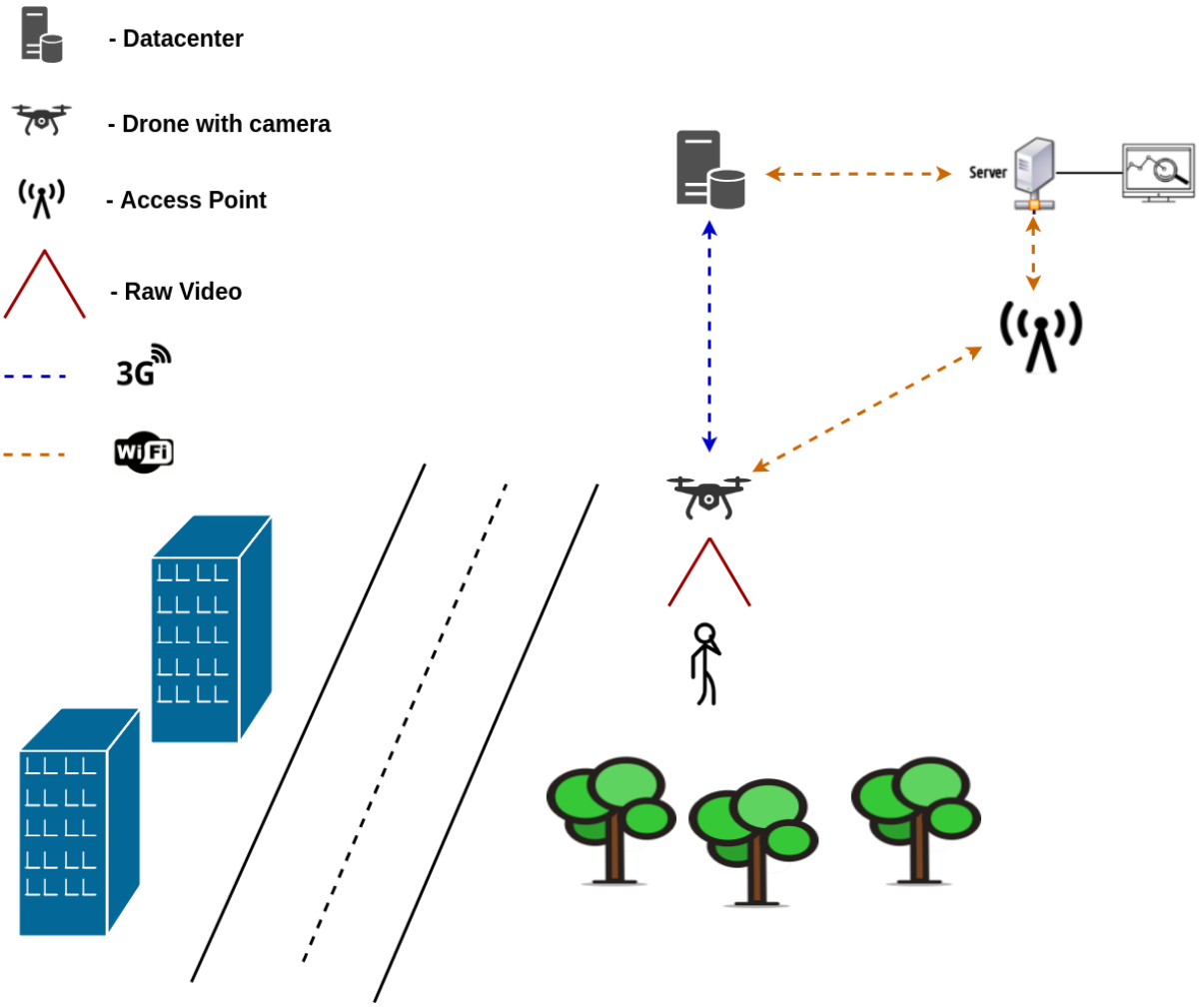


Figure 3.1: Scenario Overview.

The main challenges faced by this architecture to achieve the goal of real-time people tracking are:

Video feed In order to provide real-time tracking, a low delay in the video feed between camera and Ground Side elements is required. Also, in order to perform object detection, the captured video needs to have a base level of quality and resolution, where the improvement of both these factors will result in better object detection performance. The trade-off in this case will be between bitrate levels required to transmit the video feed and its quality.

Detection time vs Accuracy Each frame collected needs to be processed by an object detector to detect the presence and position of a person relative to the frame. The predicted position obtained through the Object detector allows for an estimation of the person's position relative to the UAV. This information is crucial when calculating the positional commands sent to the UAV. Also, since the purpose is to achieve real-time tracking, the time needed for the Object detector to process each frame must be minimized as much as possible, while maintaining an acceptable level of performance in regards to its detection accuracy (see Section 5.3.1).

Delay Between Commands The latency between each positional command sent to the drone is crucial when performing real-time tracking, since high levels of latency may compromise the whole process.

3.2 Network Elements

This section describes the various modules and their interconnection, Subsection 3.2.1 gives some details regarding the hardware equipment used, and Subsection 3.2.2 describes the software architecture.

3.2.1 Hardware Equipment

3.2.1.1 Drone

To carry out the proposed mission scenario the choice of the type of drone to use is limited to multi-rotor drones, since the capability of stationary flight is crucial for both clear video acquisition and point-to-point movement when tracking a specific target. Another characteristic to be taken into consideration is the payload capacity since the drone needs to carry external equipment required to perform the mission. To accommodate these characteristics, the hexacopter Flame Wheel 550 (F550) developed by DJI, depicted in Figure 3.2, was chosen as the preferred drone to be used. Table 3.1 presents a brief summary of the characteristics of the chosen drone.

Figure 3.2: DJI Flame Wheel 550 (F550) (from [16]).

Table 3.1: DJI Flame Wheel 550 (F550) specifications(from [16]).

DJI Flame Wheel 550 (F550)	
Frame Weight	478g,
Diagonal Wheelbase	550mm
Takeoff Weight	1200g 2400g
Recommended Propeller	10 x 3.8in ; 8 x 4.5in
Recommended Battery	3S 4S LiPo

3.2.1.2 Raspberry Pi 2 Model B

Raspberry Pi, depicted in Figure 3.3, is a single-board computer mainly used in applications that require low-power consumption and computational effort, with the hardware specifications described in Table 3.2.

Table 3.2: Raspberry Pi 2 Model B specifications (from [17]).

Raspberry Pi 2 Model B	
Processor	900MHz quad-core ARM Cortex-A7 CPU
RAM Memory	1 GB
Operating System	Raspbian 4.9.2

This single-board computer was chosen to provide an interface between the on-board camera (Raspberry Pi Camera Module v2) deployed on the UAV, as well as providing the required transcoding and transmission of the video feed created using FFmpeg [56]. Since

Figure 3.3: Raspberry Pi 2 Model B (from [17]).

this version of Raspberry Pi does not have an embedded WiFi interface, a USB Wireless adapter TL-WN722N was used.

3.2.1.3 Camera

The camera used is a Raspberry Pi Camera Module v2 with a Sony IMX219 sensor. This camera has a native resolution of 8 megapixels, and has a fixed focus lens on board. It is capable of 3280x2464 pixel static images, and it also supports 1080p at 30 frames per second (fps), 720p at 60 fps and 640x480 at 90 fps video [18]. The video compression format used is H.264 (MPEG-4). The interface between the camera and the Raspberry Pi module is made through a ribbon cable connected to the CSI camera port. The Raspberry Pi Camera Module v2 is shown in Figure 3.4.

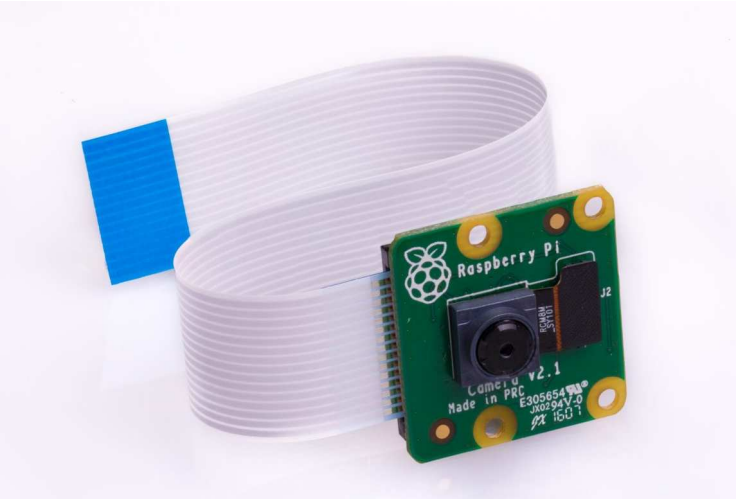


Figure 3.4: Raspberry Pi Camera Module v2 (from [18]).

3.2.2 Software Architecture Overview

Figure 3.5 overviews the software architecture implemented, showing the network's elements as well as their interconnections. Both the Drone Manager and the Drone Controller as well as their interconnection belong to the drone platform developed by Bruno Areias et al. in [24]. The desired tracking behaviour is achieved through the loop formed by the video feed collected on the drone side and the drone control logic performed by the network elements present on the ground side. The proposed architecture presents a modular approach with the following components:

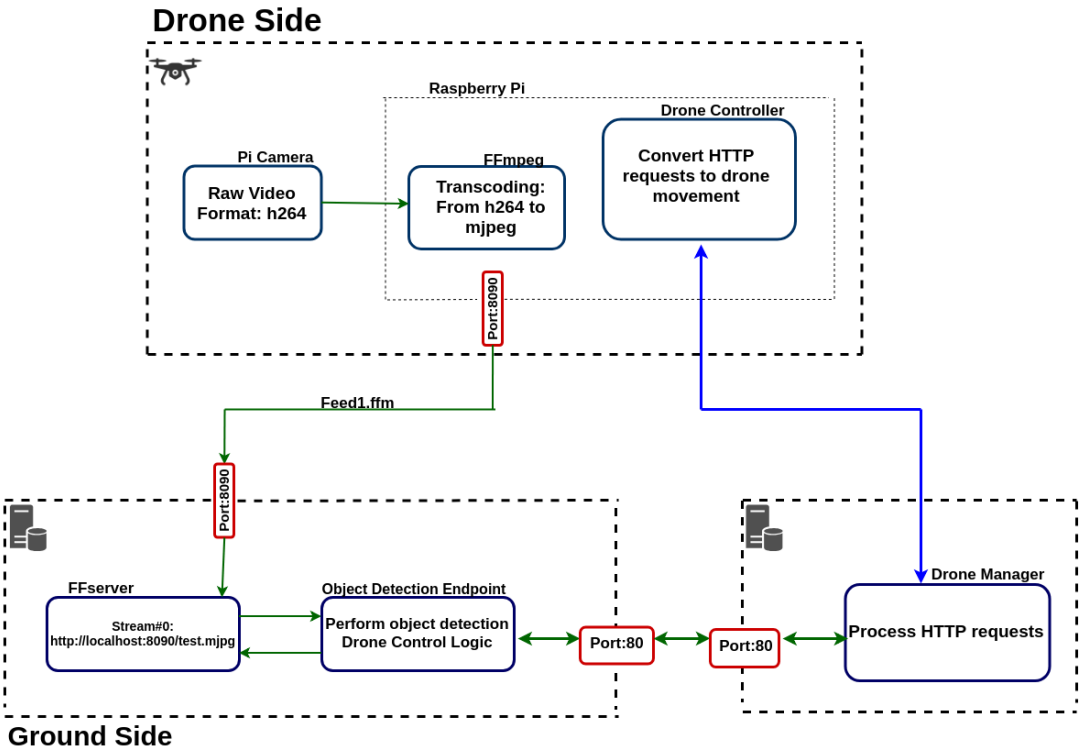


Figure 3.5: Overview of the implemented architecture.

UAV: Carries out the tracking mission, as well carries the required hardware payload for the network elements present in the drone side.

On-Board Camera: Responsible for gathering video data.

FFmpeg: Used for transcoding the raw video data collected to the required format. The resulting output of this element is a video feed used as an input for the FFserver.

FFServer: Used to stream the live video feed captured by the camera to the Object Detection Endpoint and other clients if required.

Object Detection Endpoint: Receives as an input the video feed captured by the on-board camera published by the FFServer, and implements the drone control logic behind each output command issued to the UAV.

Drone Manager: Enables high-level drone control available to users through Hypertext Transfer Protocol (HTTP) requests.

Drone Controller: Implements the commands issued from the Drone Manager on the target UAV.

3.2.2.1 FFmpeg

FFmpeg is an open source, very powerful multimedia framework, widely used for format transcoding. This platform supports multiple transmission protocols, media container formats, as well as video / audio coding standards, and it provides a unified data structure to store the information extracted from multimedia data, thus it effectively solves the difficulty in analysis of wide range of media data formats. In addition, *FFmpeg* provides highly efficient transcoding algorithms that can meet requirements of real-time video analysis [57], such as the one aimed for this dissertation's work. Present in the Drone Side of the architecture, this module is used to encode the data collected by the Pi Camera (H.264 raw video data) to a *mjpeg* video feed. In addition, *FFmpeg* allows to define the characteristics of the video feed created such as: bitrate, frames per second transmitted and resolution. This video feed is used as an input for the FFserver module.

3.2.2.2 FFServer

FFserver is a streaming server for both audio and video. It supports several live feeds, streaming from files in the device as well as time shifting on live feeds. FFserver receives as an input pre-recorded files or FFM streams from an FFmpeg instance as input, and produces as an output streams that can be transmitted over RTP, RTSP or HTTP. The FFserver process listens to a port as specified in the configuration file. The process can handle more than one instance of FFM streams sent over from FFmpeg. Input streams sent to the server are called feeds, and each one is specified by a <Feed> section in the configuration file. The configuration file is read at the startup. The configuration file used can be seen below:

`ffserver.conf`

```
HTTPPort 8090
HTTPBindAddress 0.0.0.0
MaxHTTPConnections 200
MaxClients 10
MaxBandwidth 100000
```

```
CustomLog -

<Feed feed1.ffm>
        File /tmp/feed1.ffm
        FileMaxSize 1M
</Feed>

<Stream test.mjpg>

Feed feed1.ffm
Format mpjpeg
VideoFrameRate 4
VideoQMin 2
VideoQMax 5
VideoBufferSize 200000
VideoSize 800x600
NoAudio
Strict -1

</Stream>
```

For each feed, there can be different output streams in various formats, each one specified by a <Stream> section in the configuration file.

FFserver acts as an HTTP server, accepting POST requests from FFmpeg to acquire the stream to publish, and serving RTSP or HTTP clients' GET requests with the stream media content.

Each feed is identified by a unique name, corresponding to the name of the resource published on FFserver, and is configured by a dedicated Feed section in the configuration file.

The stream formats that are supported by an FFServer instance are shown in Table 3.3.

3.2.2.3 Object Detection Endpoint

The implementation environment for the Object Detection Endpoint is an ASUS X550j laptop computer with an Intel Core i7-4710hq 2.50 GHz CPU, 8 GBs of RAM and an

Table 3.3: Stream formats supported by FFserver (from [23]).

Format	Brief Description
mpeg	MPEG-1 multiplexed video and audio.
mpegvideo	only MPEG-1 video.
mp2	MPEG-2 audio (use AudioCodec to select layer 2 and 3 codec).
ogg	Ogg format (Vorbis audio codec).
rm	RealNetworks-compatible stream. Multiplexed audio and video.
ra	RealNetworks-compatible stream. Audio only.
mpjpeg	Multipart JPEG (works with Netscape without any plugin).
jpeg	Generate a single JPEG image.
asf	ASF compatible streaming (Windows Media Player format).
swf	Macromedia Flash compatible stream.
avi	AVI format (MPEG-4 video, MPEG audio sound).

NVIDIA GeForce 850M GPU. The operating system used is Ubuntu 18.04.1 LTS. This server consists of a Representational State Transfer (REST) endpoint that provides person detection to the images posted to it, as well as produce the commands sent to the active UAV. This server receives as an input a single frame sent by the client; this frame is received via the HTTP POST method. The formats accepted are: png, jpg and jpeg.

To provide Object Detection, this node uses Darkflow [58]. Darkflow is an open-source object detector that implements the Convolutional Neural Network (CNN) architecture of YOLO [21] in Python. Darkflow uses the machine learning Framework Tensorflow 1.0 [59] and OpenCV libraries [60]. To enable Graphics Processing Unit (GPU) computation and achieve more efficient allocation of computational resources during Object Detection tasks, Darkflow was installed alongside CUDA v9.0 Toolkit [61] and cuDNN v7.1 [62]. A more detailed overview of this CNN is given in Section 5.3.

The output of this CNN, for an input frame, is given in the JavaScript Object Notation (JSON) format with the information on the class detected (label), the level of confidence (from [0,1]) and the pixel coordinates of the corresponding bounding box as shown in Figure 3.6.

Following the detection process and based on the output from the YOLO network, this REST endpoint then proceeds to determine which commands are to be issued in order

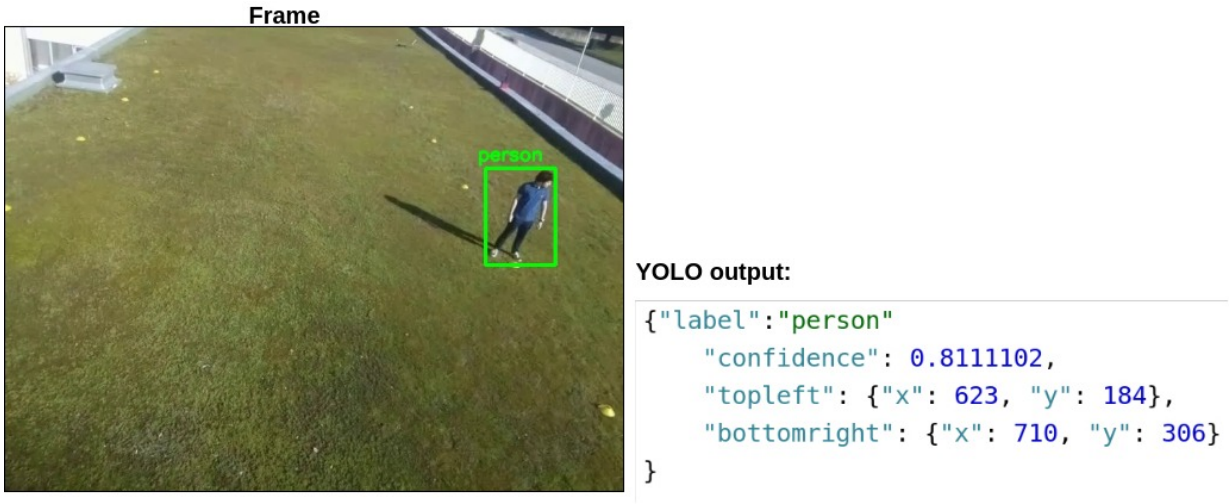


Figure 3.6: Example of the YOLO CNN output for a given input frame.

to continue (or start) the tracking procedure (the control logic implemented is detailed in Section 4.3). The commands are issued using the HTTP POST method to the Drone Manager. The frame size will influence this server's response time, since each frame needs to be processed by the YOLO network, with larger sized frames requiring more computational effort on the server side.

3.2.2.4 Drone Manager

The Drone Manager consists of a REST endpoint that enables high-level drone control and monitoring accessible to users through HTTP GET and POST requests. This node will provide the Object Detection Endpoint with drone control based on geographical coordinates as well as drone telemetry data acquisition. Table 3.4 describes the methods used by the Object Detection Endpoint when communicating with the Drone Manager.

Method	Command	Additional Parameters	Description
POST	horizontalChange	**droneID**: Drone Identifier **North**: distance in meters **East**: distance in meters	Routed to the Drone Controller. Commands the drone to move in the **North-South** and **East-West** bound direction by a specified number of meters.
GET	getheading	**droneID**: Drone Identifier	Routed to the Drone Controller. Retrieves the orientation of the drone in relation to the Geographic North.

Table 3.4: Methods used by the Object Detection Endpoint (from [24]).

3.3 Summary

In this chapter we presented the proposed architecture implemented to carry out person tracking missions using UAVs. The proposed solution is a computer vision based control architecture comprised of multiple software and hardware modules deployed in the drone side and the ground side. These modules interact with each other in order to carry out the tracking mission. The UAV control is accomplished using the Drone Platform presented in [24], which already provides GPS based control and monitoring of UAVs. Once implemented, the proposed architecture aims to be able to perform UAV tracking missions with people as targets without user inputs.

Chapter 4

Integration and Implementation

This chapter describes some of the challenges and limitations faced by the architecture proposed in the previous chapter, and the control choices that were implemented in order to circumvent them. Section 4.1 shows the method used to derive an estimate of the distance from the target, person, to the active Unmanned Aerial Vehicle (UAV). Section 4.2 presents the "Noise" introduced in the live video feed by the airborne UAV when capturing video from a stationary GPS position without any stabilization mechanisms applied to the on-board camera (e.g., gimbal support). Lastly, Section 4.3 provides an overview of the control algorithm implemented in the Object Detection Endpoint, taking in consideration the limitations imposed and described in the Sections 4.1 and 4.2.

4.1 Camera Calibration

To determine which commands are to be sent to the UAV when performing the tracking mission, an estimate of the distance from the target detected inside each frame relative to the camera is needed. In other words, there needs to be a translation of the position, in terms of pixel coordinates, of the tracked object (in this case a person) to the actual real world position of the object relative to the camera. To obtain this estimate, it is important to know that each pixel in the frame is going to translate to different lengths in the corresponding scene being depicted, as shown in Figure 4.1.

In order to get an estimate of pixel size as it relates to the real world, the UAV, with the camera mounted on it, flies to a marked terrain where the ground distance to the drone in known for several placed pins, depicted in Figure 4.2.

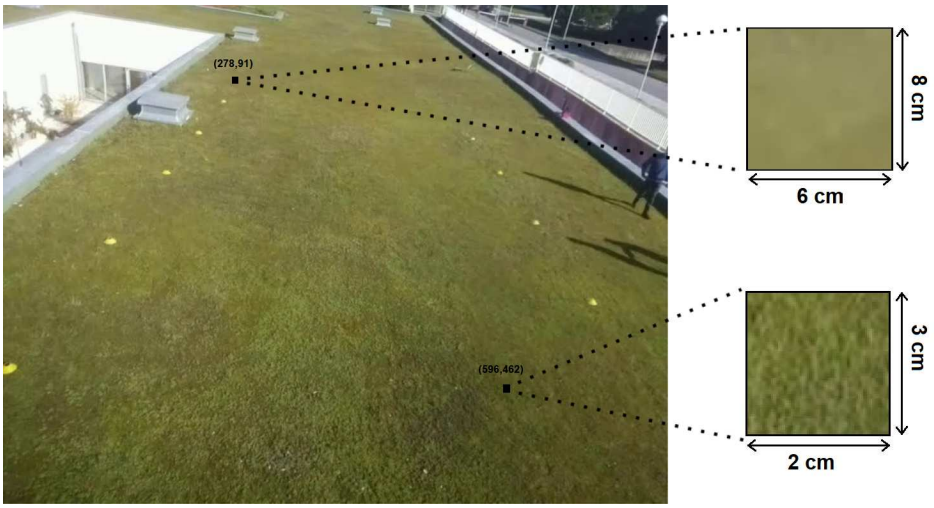


Figure 4.1: Example of sizing difference between pixels when compared to the actual distance they depict.

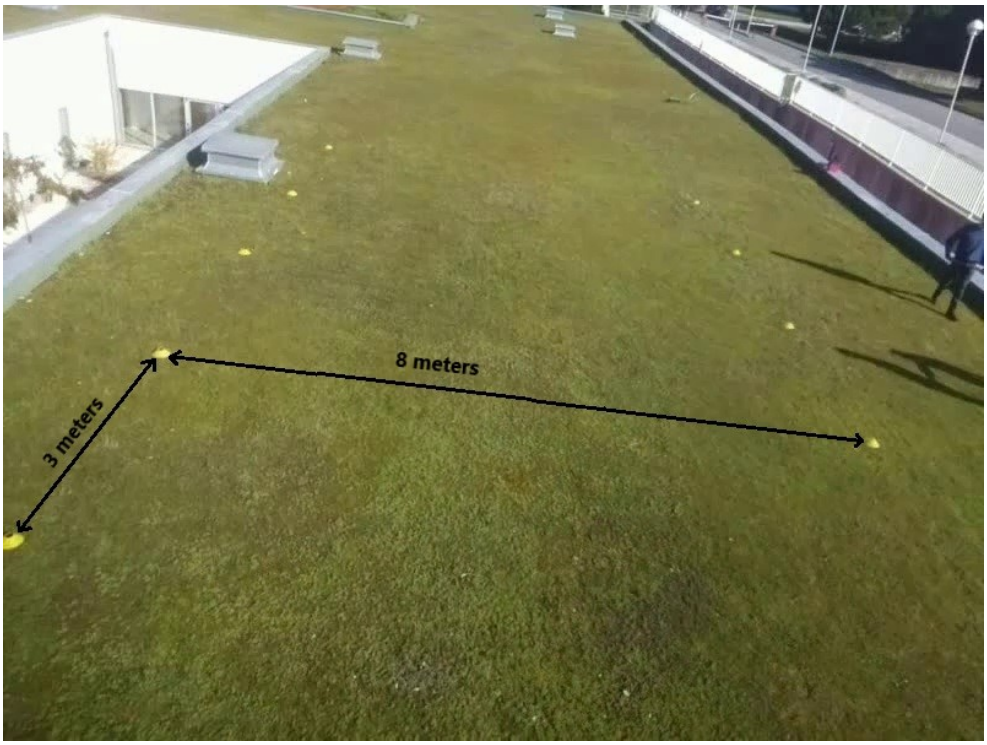


Figure 4.2: Relative pin placement used for distance estimation.

The UAV is positioned at the height required for carrying out the tracking mission: 5 meters above ground. This height was chosen as a good compromise between distance

to the target when centered inside the frame (between 5 and 8 meters when standing directly in front of the camera, optimal for Object detector performance as shown in Section 5.3.3), and maximizing the overview of the surrounding landscape area beneath the drone. Knowing the position of each pin placed on the ground relative to one another, we can obtain the following estimate values shown in Figures 4.4 and 4.3. Figure 4.3 shows a estimated distance in the horizontal direction perpendicular to the UAV's heading for each pixel in relation to its height value inside the frame.

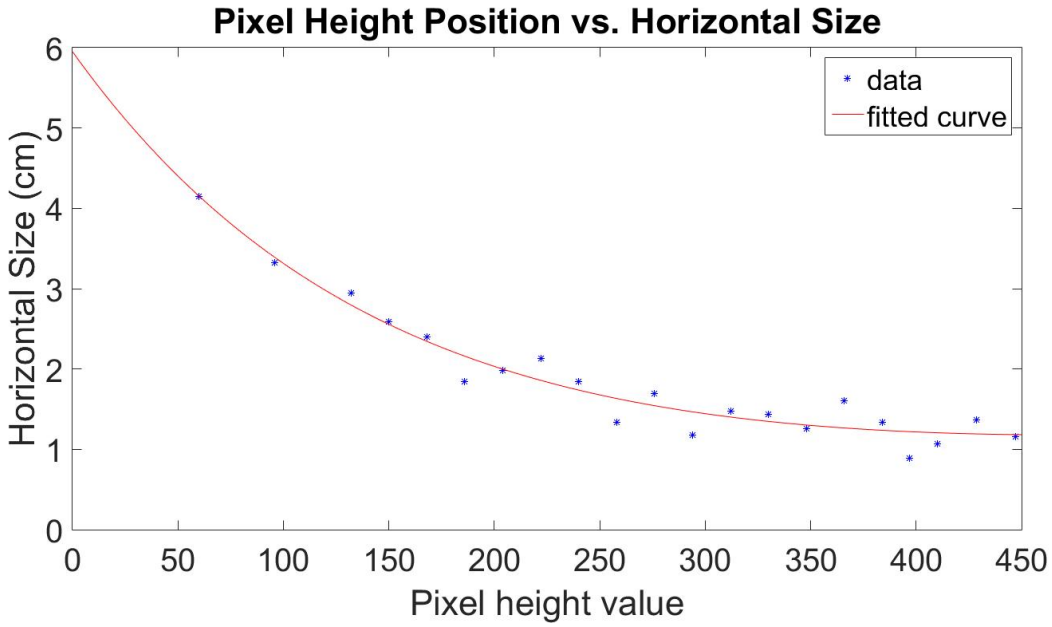


Figure 4.3: Horizontal distance estimation based on the pixel height value.

Figure 4.4 shows the estimated distance in the vertical direction parallel to the UAV's heading for each pixel in relation to its height value inside the frame. The values obtained show an exponential relationship between them that can be described by the following mathematical model:

$$y(x) = Ae^{bx} + Ce^{dx} \tag{4.1}$$

Table 4.1 shows the values of the coefficients obtained using a non linear least squares regression, where f1 and f2 correspond to the data shown in Figures 4.4 and 4.3 respectively.

Using these values, we can approximate dx, in cm, as a function of the pixels' height value using the equation:

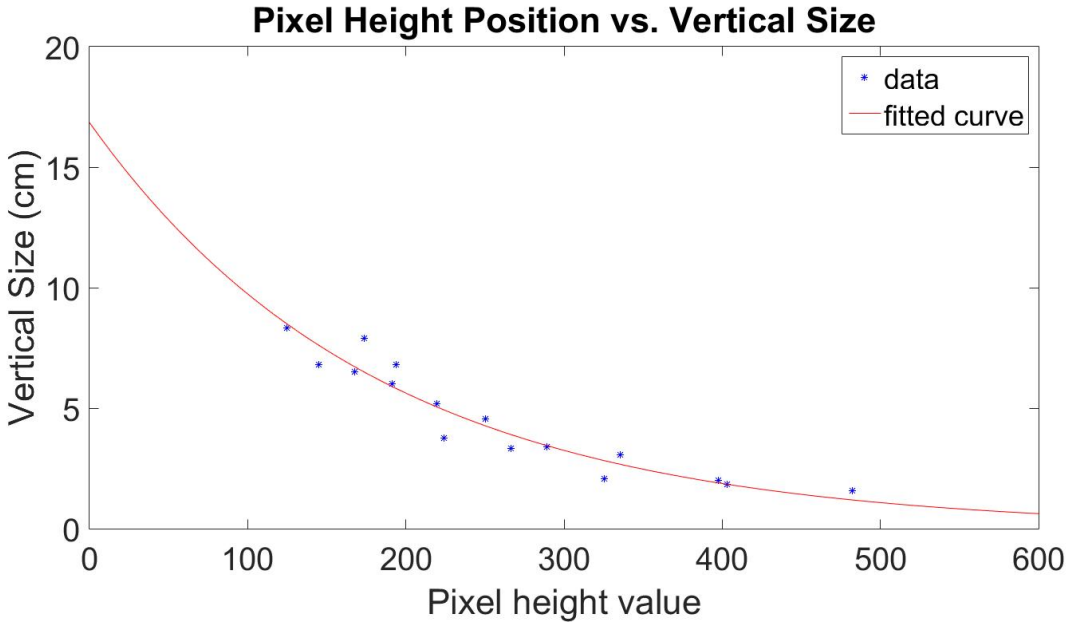


Figure 4.4: Vertical distance estimation based on the pixel height value.

	A	b	C	d
f1	4.2659×10^{4}	-5.2516×10^{-3}	-4.2642×10^{4}	-5.2515×10^{-3}
f2	5.4922	-6.8118×10^{-3}	4.5767×10^{-1}	1.5656×10^{-3}

Table 4.1: Coefficient values obtained.

$$dx(p\_height) = 5.4922e^{-0.0068118 \times p\_height} + 0.45767e^{0.0015656 \times p\_height} \tag{4.2}$$

In addition, we can also get an estimate of dy, in cm, as a function of the pixels' height value using the following equation:

$$dy(p\_height) = 42659e^{-0.0052516 \times p\_height} + 42642e^{-0.0052515 \times p\_height} \tag{4.3}$$

These estimations do not take into account variations in terrain inclination as well as drone altitude fluctuations. Fluctuations in these parameters will introduce errors in the estimated distance calculated, since the perspective of the camera is changed.

4.2 Noise Generated in the Image by a Stationary Drone

The camera used to create the video stream is directly mounted at the base of the UAV. Considering the fact that the UAV when airborne does not behave like a fixed object, the camera mounted on it will produce a video feed that has an inherent “wobble” due to different factors that the drones’ motors try to correct (e.g., wind, atmospheric pressure changes,etc.). This "wobble" will, consequently, introduce noise to the video feed in the form of a slight change in the perspective between each consecutive frame. An example of this is shown in Figure 4.5.

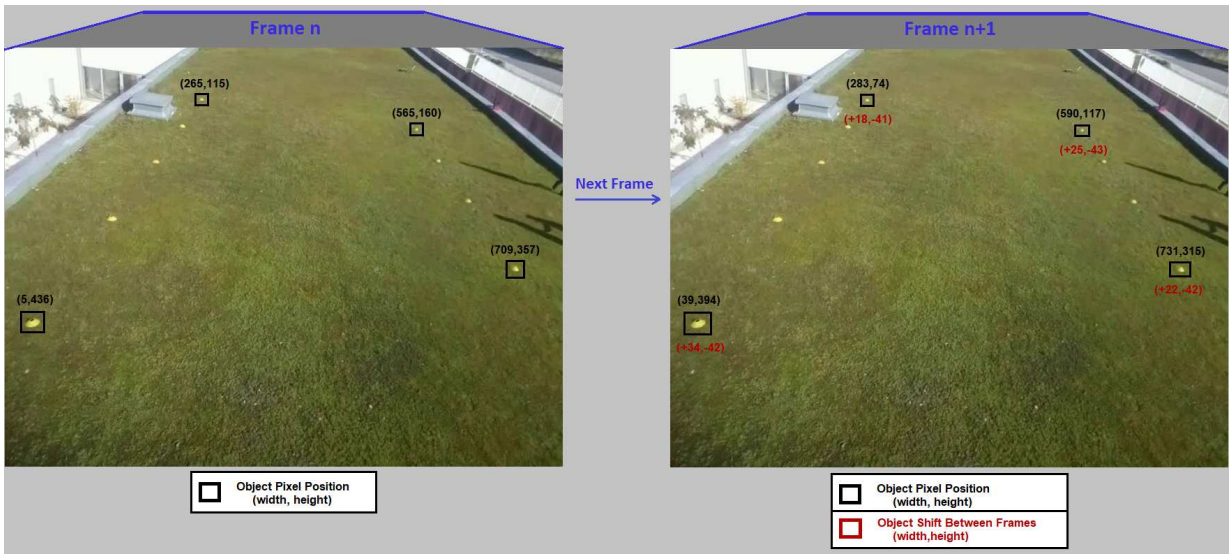


Figure 4.5: Object pixel position shift between consecutive frames.

As we can see, consecutive frames shift the perspective of the camera slightly leading to the objects being photographed, changing the position within each frame by a different amount. This will in turn have consequences when translating the object position inside the frame to real world coordinates in relation to the UAV. We can expect this noise, generated in the image by the UAV flight stabilization mechanisms reacting to the changing flight circumstances, to have a normal distribution (or bell shaped curve). The following results shown in Figure 4.6 show the different values of shift observed between frames when recording video (800x600) from the chosen UAV at 4 frames per second. The values are in terms of pixels for both shift along the width of the image as well as the height. Results were measured for each different frame quadrant.

Figure 4.7 shows one of the histogram obtained, in this case, for the first quadrant.

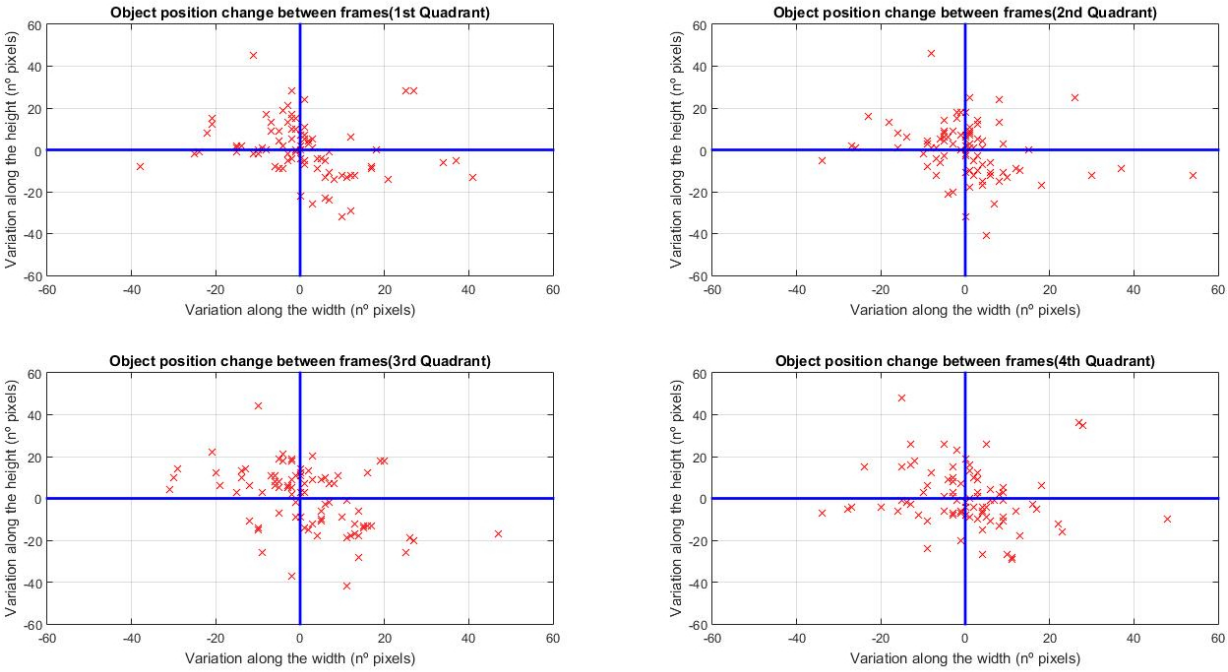


Figure 4.6: Scatter plot of the observed object shift between frames.

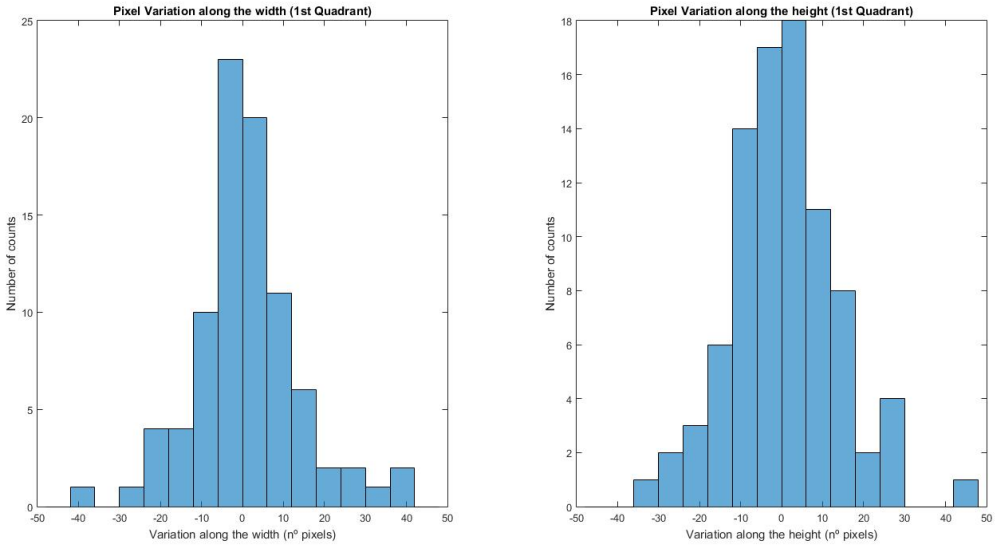


Figure 4.7: Histogram for the observed object shift between frames in the first quadrant.

As expected, the results show that the noise introduced is indeed similar to a normal distribution. Table 4.2 displays the values that characterize the noise introduced in each

	Horizontal (nr. pixels)		Vertical (nr. pixels)	
	σ	μ	σ	μ
1st Quadrant	12.84	0.43	13.27	0.26
2nd Quadrant	12.52	0.74	13.3	-0.26
3rd Quadrant	12.9	0.77	14.88	0.32
4th Quadrant	12.49	0.49	14.12	0.37

Table 4.2: Mean and standard deviation values of noise for each quadrant.

quadrant of the frame. This effect will have to be taken into account when performing the translation between the objects' pixel position and the distance from drone estimation. Also, when the object is detected in the first two quadrants, this noise will inevitably produce higher distance translation errors, since the pixels in these quadrants correspond to larger distances as shown in Section 4.1. For example, on a given frame, a shift in the height value of a detection from 163 to 150 (which is close to the value of the standard deviation found for the vertical noise introduced) will represent approximately a difference of 1 meter when performing the distance prediction described in Section 4.1. To minimize these errors and obtain a more reliable distance to target prediction, two strategies are used:

- Perform 3 consecutive positive detections and average out the resulting predicted bounding boxes.
- Consider the true bounding box to be within 2 standard deviation factors for a 95% certainty on the objects' position.

4.3 Control Algorithm

To perform the proposed tracking mission, the commands issued to the UAV will be based on the position of the detected target (person) inside the frames being streamed, so as to position the UAV in such a way that the target will be centered upon performing each command. The control logic implemented is shown in Figure 4.8.

After initializing the chosen Convolutional Neural Network (CNN) model characterized in Section 5.3, the Object Detection Endpoint proceeds to issue an Hypertext Transfer

Protocol (HTTP) GET request to the FFServer to acquire the video stream being published by the FFmpeg instance present on the drone side. Each frame received is then inputted to the already loaded CNN. The output from the CNN will either show a positive detection with the corresponding Object pixel coordinates predicted by the network or no valid detection. Multiple positive detections inside the same frame are considered as non detections by the control algorithm. The coordinates for a positive detection are fed into a data buffer called **Position_Stack**; this buffer is implemented as a First in First out (FIFO) queue with a maximum size of 12 integers (corresponding to 3 positive detections in 3 different frames, each represented by 4 pixel coordinates). In order to get to the final step where the drone command is calculated and sent, the **Position_Stack** must verify the following requirements:

Contents The data buffer is full.

Range Tolerance The 3 detections must be within a distance, in pixels, between each other of no more than 10% of the frame's width and height values. The reasoning behind choosing 3 different and consecutive valid detections arises from the fact that the airborne UAV cannot be considered a perfectly stable platform to gather video; even when simply hovering above the desired location there is always an inherent "wobble" in its position. This "wobble" will consequently be present in the video feed as well as the detection's coordinate values, details on this behaviour. The value of 10% was achieved through heuristic methods.

Expiration The coordinates corresponding to the head of the queue belong to a frame dating back no longer than 15 frames from the tail of the queue.

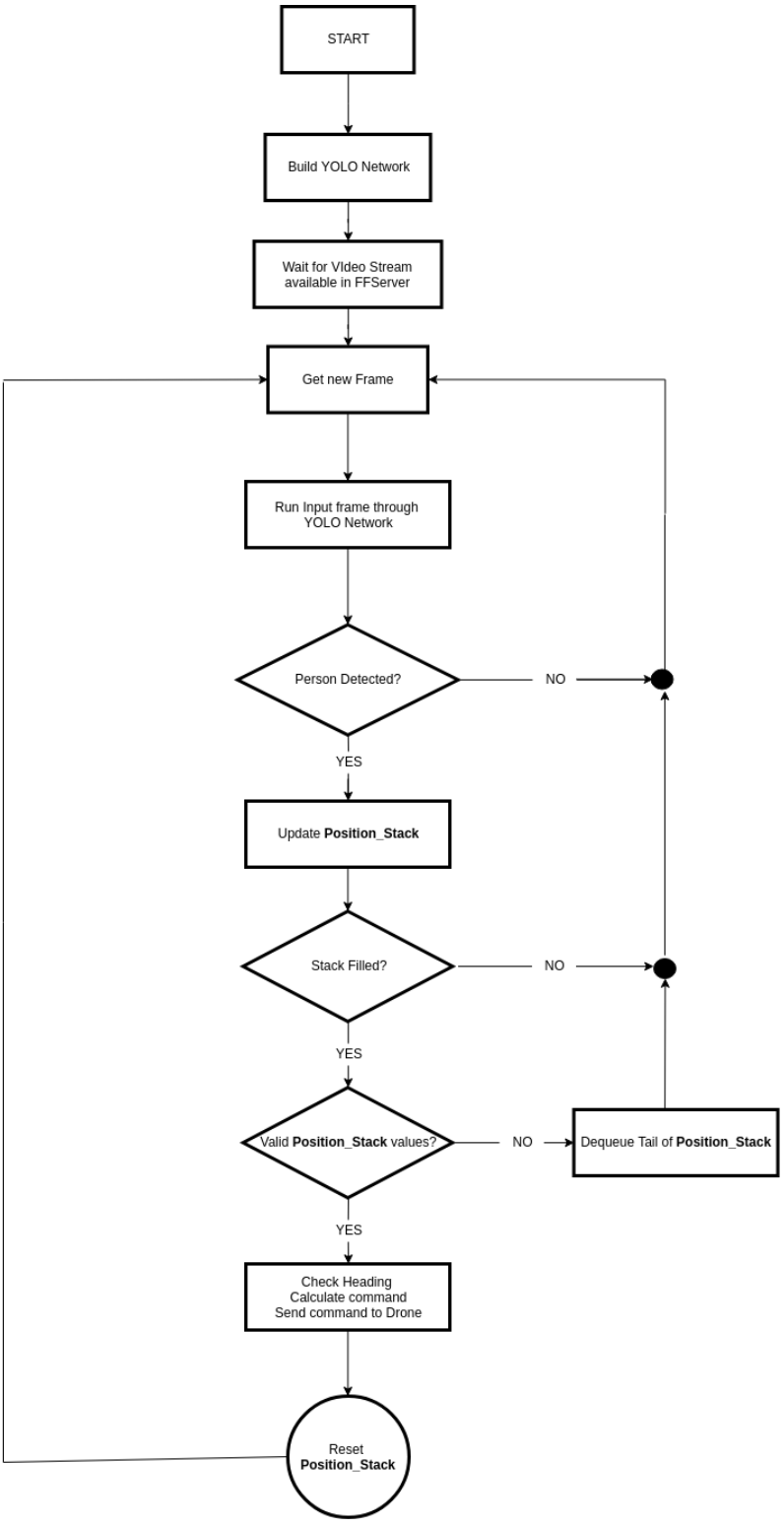


Figure 4.8: Diagram of the Control Algorithm.

4.3.1 Command Calculation

When calculating the command to be sent, two factors are taken into consideration:

Drone Heading This value corresponds to the direction that the drone faces in relation to the cardinal directions, and is aligned with the camera's perspective allowing for an estimation on the position of detected objects in relation to the drone. When airborne, the drone will experience constant slight changes in its heading. To account for this behaviour, the heading values are updated for every frame that has a positive detection and are obtained through the Drone Manager via HTTP request. The value obtained in response is a float value between [-180,180] and represents the current heading of the UAV in relation to the North direction, as shown in Figure 4.9.

Detection Pixel Position The detected position used for each command corresponds to the averaged out values of the detections' pixel position saved in the validated **Position_Stack**.

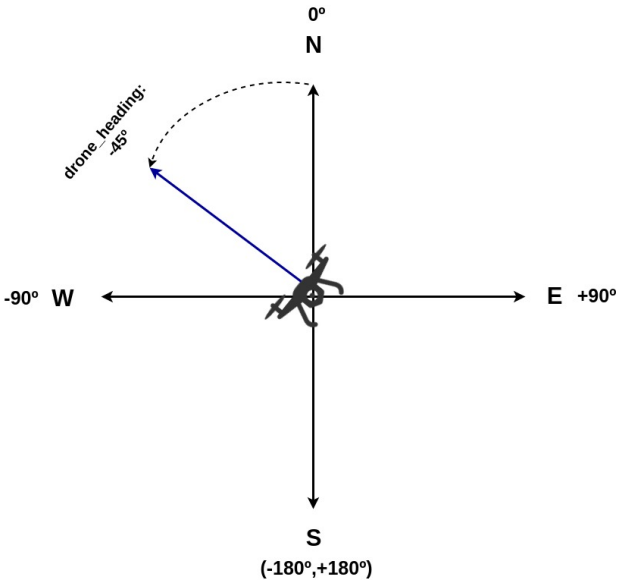


Figure 4.9: Possible UAV's heading values obtained using the Drone Manager.

Using these two factors, the objective of the command issued to the UAV is to move the drone in the best possible direction and distance, so as to center as close as possible the target (person) in the video feed being gathered by the active UAV. Each movement command is sent to the Drone Manager via HTTP POST request and uses the parameter

command 'horizontalChange' present in the Drone Managers' Application Programming Interface (API): to move the drone in the North direction, the parameter 'north' is set as a positive value; while South bound movement corresponds to negative values. The same logic is applied to movement performed in the East-West direction, where East is set as positive values.

Example: A command issued to the drone with the aim of moving it 4.5 meters South and 3 meters East with the drone identifier: $REVOLUTION\_94E041$ can be achieved by executing the following POST request to $https : // < DroneManager\_address > /control$ with parameters:

command=horizontalChange&north=-4.5&east=3&droneID=$REVOLUTION\_94E041$.

4.3.2 Obtaining the command direction angle

The command sent to the drone can be seen as a 2D vector in terms of the geographic directions and is therefore characterized by an angle and length. The angle of the vector is derived from the vector formed between the center of the averaged out values present in the **Position_Stack** and the center of the frame in terms of pixels and the current value for the drone heading.

4.3.3 Obtaining the command values

Knowing the position of the detected person in pixel coordinates inside a given frame, it is possible to estimate the distance that is required to move parallel to the UAV's heading in order to center the person at the frame's height midpoint. This distance can be estimated as the area below the curve formed by Equation 4.3 between the height value of the detected person's boundary box and the midpoint of the frame's height, as shown in Figure 4.10.

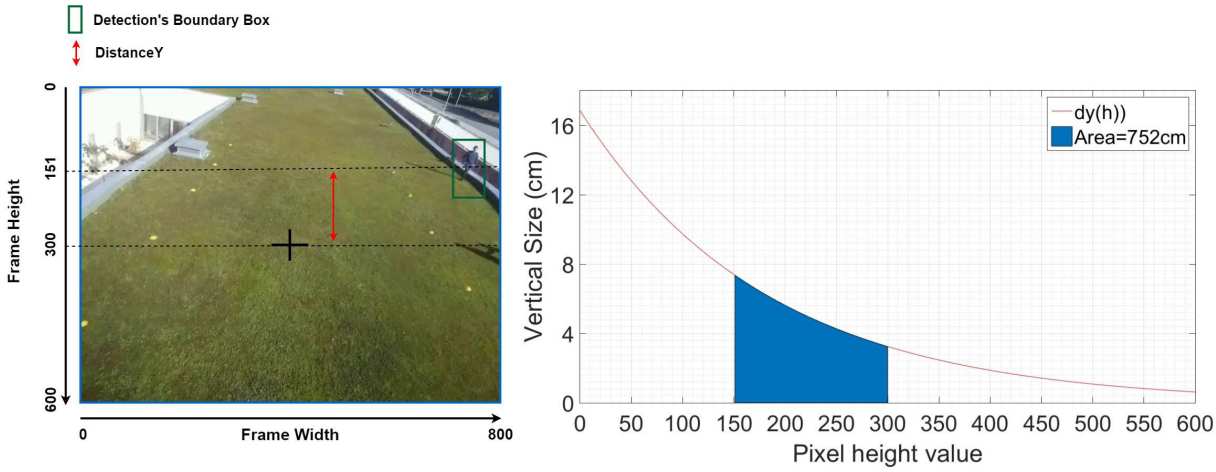


Figure 4.10: Estimation of vertical distance.

This area can be given by the integral of Equation 4.3 from $[h_b, \frac{Frame\_Height}{2}]$:

$$distanceY = \int_{h_b}^{\frac{Frame\_Height}{2}} (42659e^{-0.0052516y} + 42642e^{-0.0052515y})dy \tag{4.4}$$

Where $Frame\_Height$ represents the frame's pixel height and h_b represents the boundary box's height value.

Likewise, it is also possible to estimate the distance that is required to move perpendicular to the UAV's heading in order to center the person at the midpoint of the frame's width. This estimate is based on Equation 4.2 and is expressed by:

$$distanceX = (5.4922e^{-0.0068118\times h_b} + 0.45767e^{0.0015656\times h_b})(w_b - \frac{Frame\_Width}{2}) \tag{4.5}$$

Where $Frame\_Width$ represents the frame's pixel width and h_w represents the boundary box's width value.

Knowing the estimated distances $distanceX$ and $distanceY$, we need to convert these values into distances in the North-South and East-West bound direction in order to use the Drone Manager's API to issue drone movement commands. This can be achieved by using the transformation matrix T that maps out the vector formed by $distanceX$ and $distanceY$ to the geographical coordinates plane according to the value of the UAV's heading:

$$\begin{bmatrix} \text{WE} \\ \text{NS} \end{bmatrix} = \underbrace{\begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}}_{T} \times \begin{bmatrix} \text{distanceX} \\ \text{distanceY} \end{bmatrix} \tag{4.6}$$

Where θ is the value of the drone's heading and NS, WE are the the estimated distances to be issued in the North-South and East-West bound direction, respectively. Figure 4.11 shows an example of the final command values obtained knowing the drone's heading and the valid values for the **Position_Stack**.

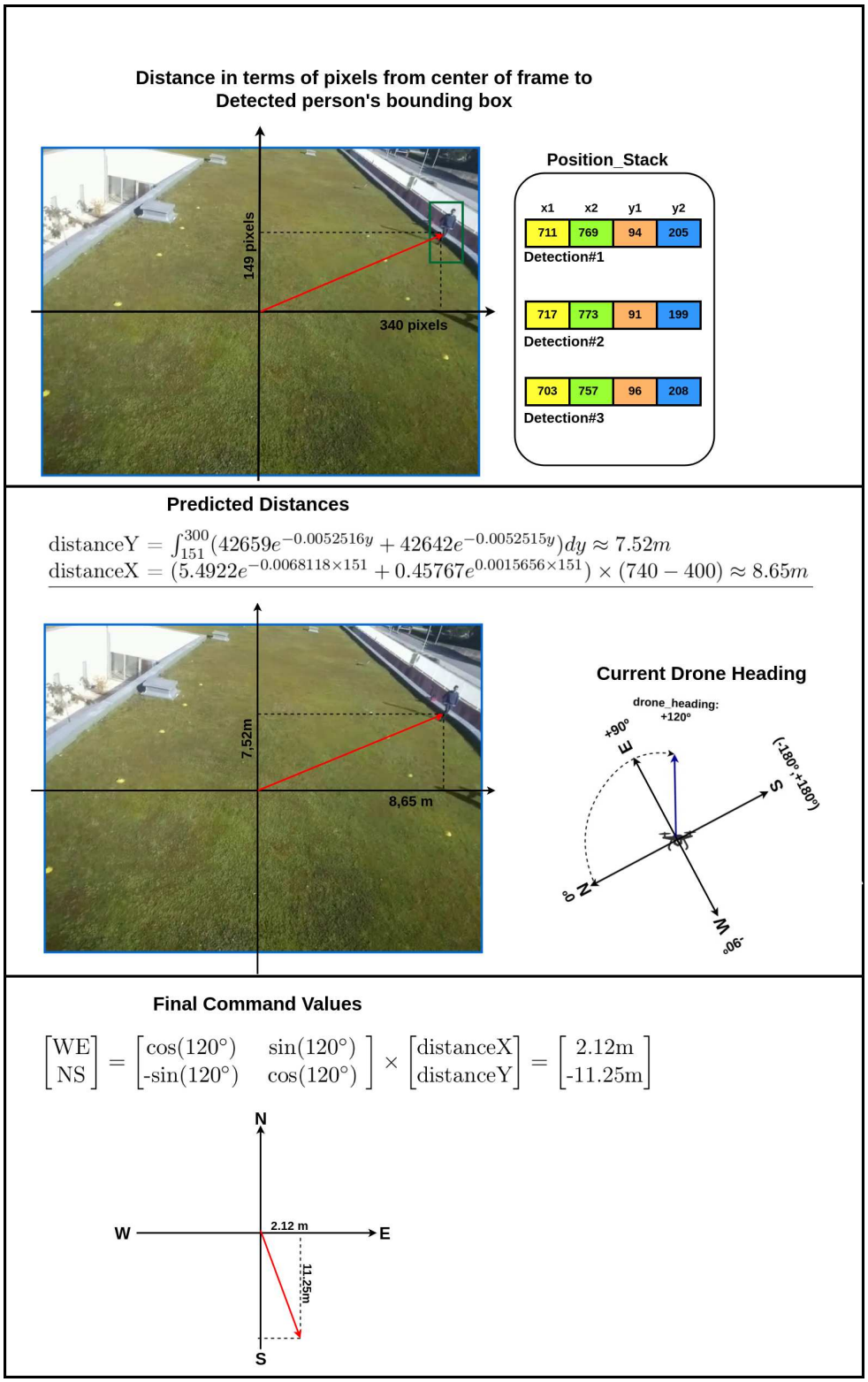


Figure 4.11: Example of an output command given the UAV's heading and 3 validated detections.

4.3.4 Logging

Each message traded between the Object Detection Endpoint and the Drone Manager is logged for debugging purposes for both successful and unsuccessful HTTP requests traded as shown in Figures 4.12 and 4.13.

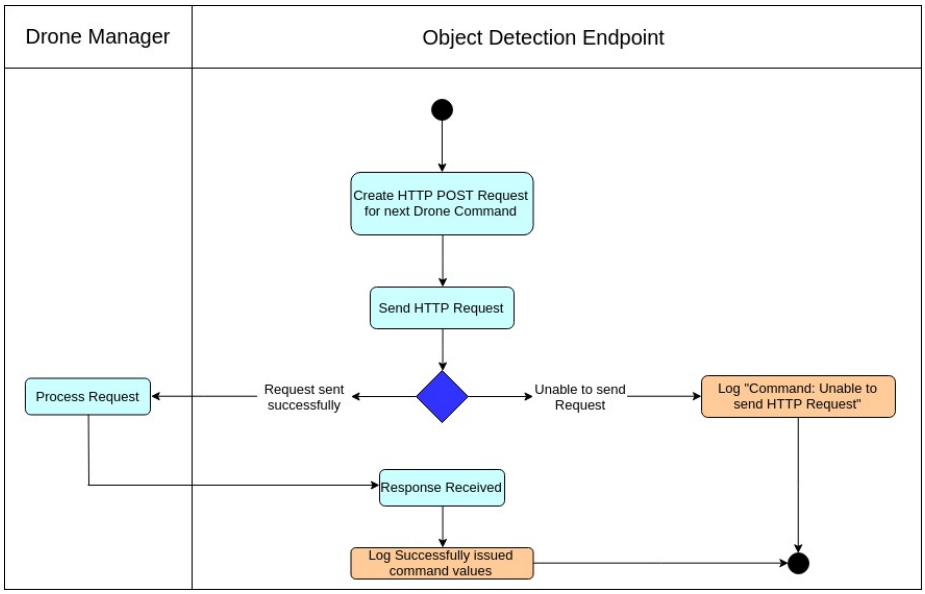


Figure 4.12: Activity diagram for HTTP POST requests.

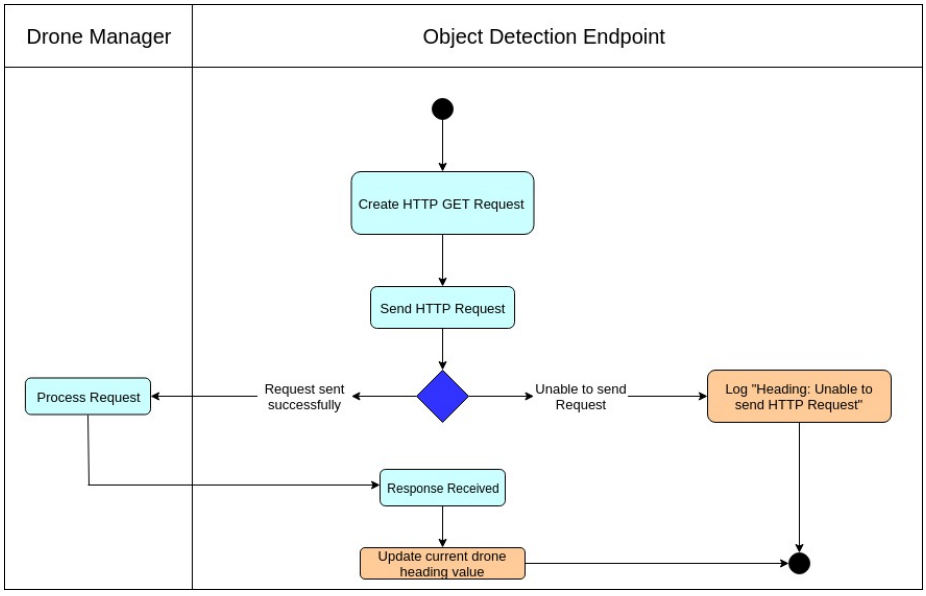


Figure 4.13: Activity diagram for HTTP GET requests.

4.4 Summary

This chapter, we described the implementation of the vision-based architecture for UAVs that fulfills the established requirements for person tracking. This includes the method used to obtain an estimation of the distance from the UAV to the person being tracked. Next, the uncertainty introduced in the live video feed by the airborne UAV was measured so as to establish its effect on the distance estimation. Finally, we described the control loop algorithm implemented, the logic behind validating a detection bounding box and the method used to convert the estimated distances to the geographical coordinates plane that is required in order to use the Drone Manager API for the drone control.

Chapter 5

Results

This chapter's main goal is to test the feasibility of real-time object recognition through a video feed captured by an airborne Unmanned Aerial Vehicle (UAV). To provide person detection in the live video feed, two methods will be tested in terms of accuracy as well as time spent by each algorithm between detections (Support Vector Machine (SVM) human detector based on Histogram Oriented Gradients (HOG)s [19], followed by the Convolutional Neural Network (CNN) based architecture of You Only Look Once (YOLO) [11]).

Section 5.1 describes the image dataset created and used in the testing, and the consequent comparison of both methods.

Sections 5.2 and 5.3 will introduce the two different Object Detection methods. Both Sections will provide an overview of the methods and the architecture followed by their results when predicting images from the aforementioned dataset.

Section 5.3.3 will evaluate the accuracy of the chosen CNN when performing people detection only, for different distances from the camera.

Finally, Section 5.4 shows the final results obtained when carrying out the tracking mission.

5.1 Image Dataset Used

To test the Object Detection algorithms discussed in the next subsections (Section 5.2 and Section 5.3) when performing purely human detection on images taken from an airbourne UAV, an Image Dataset was created. This dataset will be used to characterize

both methods in terms of image processing time, as well as to obtain their Average Precision (AP) for these types of images. The results are meant to determine which method is most suited for the task. The dataset consists of 1100 images that depict one person standing in an open field at different distances and angles from the camera. To account for the inevitable intra-class variability inherent to human based detection, different poses and orientations were taken into account besides the basic front-view profile. Of the 1100 images, there are 72 in which no person is present to account for possible false detections. The images were taken at height of 5 meters using a Raspberry pi camera v2 with a resolution of 800x600 and using the H.264 codec (this codec was chosen to mimic the data compression introduced by the video feed). Samples of images from the described dataset are shown in Figure 5.1.

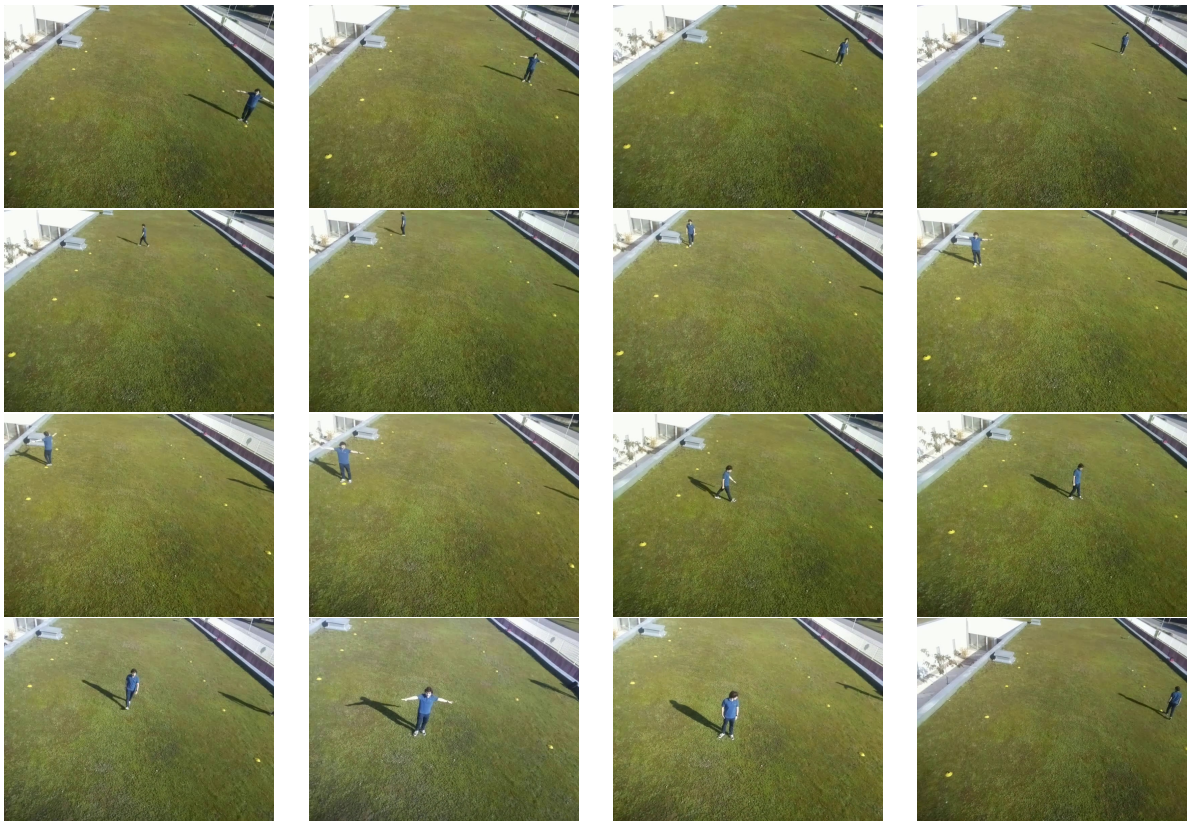


Figure 5.1: Examples of frames in the Testing Dataset.

Each frame is characterized by two labels that follow the format employed in the Pascal Voc Challenge [54], one uses the Extensible Markup Language (XML) data encoding while the other uses the JavaScript Object Notation (JSON) format. The labels contain

information for the corresponding frame in terms of:

- The frames': width, height and depth values;
- The class present (e.g., person, car, bicycle);
- The objects' bounding box pixel coordinates;
- The name of the Dataset;
- The filename;
- The file path.

An example showing the labels obtained for a given frame present in this Dataset is depicted in Figure 5.2.

5.2 Human detection through HOG based SVM

Presented in 2005 at the Computer Society Conference on Computer Vision and Pattern Recognition, a method was proposed using HOG along a grid to extract feature maps from images to train a linear SVM with the purpose of distinguishing between images showing a person/people and those that have none.

HOGs are a type of feature descriptor that can describe a given input image in terms of shape, color, texture or motion, among others. The purpose of using HOGs in this implementation is based on the assertion that different types of objects and shapes can often be characterized, to a certain degree of confidence, by the distribution of localized gradients or edge directions, even without precise knowledge of the corresponding gradient or edge positions [19].

The proposed detector is tested against two different datasets. The first one is the MIT pedestrian database [63] which contains mostly images of people in different urban scenarios. Samples from the dataset used are shown in Figure 5.3. This database consists mostly of people in an upright position, either walking or standing, thus limiting the range of different positions that a person can take. To account for this limitation a second dataset

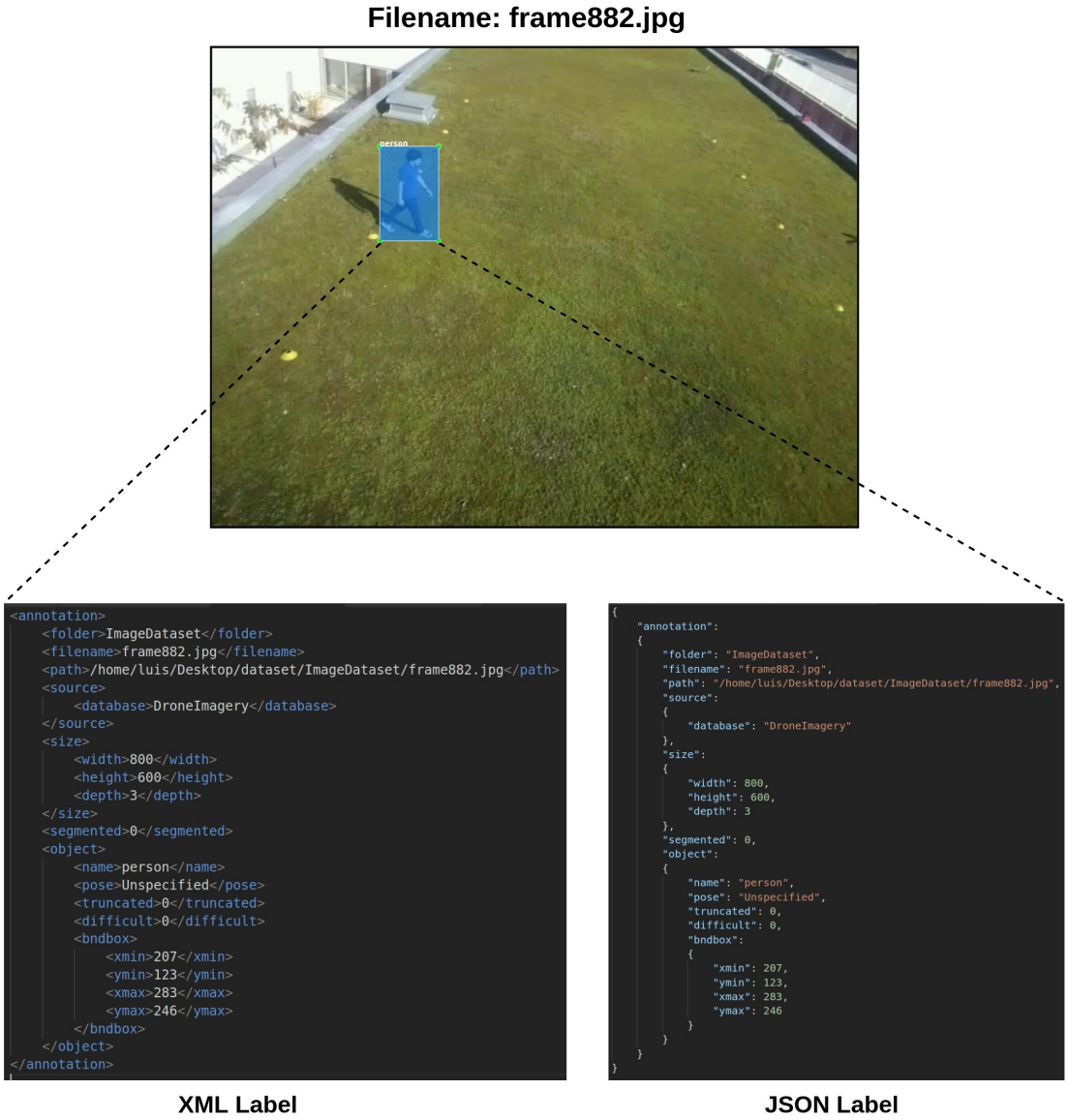


Figure 5.2: Example of labels in the XML and JSON format.

---

was added and used for training and testing purposes: "INRIA" [64]. This dataset includes different backgrounds including crowds as well as various types of poses taken by the person depicted.

Figure 5.3: Examples of images used in [19].

5.2.1 The SVM algorithm

An overview of the feature extraction and object detection chain used by this method can be seen in Figure 5.4.

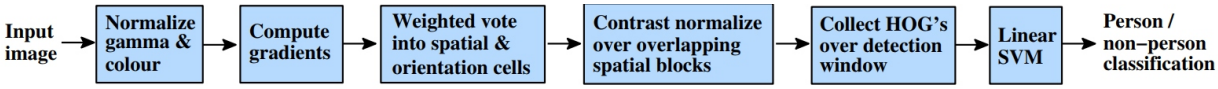


Figure 5.4: Overview of the feature extraction and object detection chain present in [19].

The first block is used to accomplish better invariance to illumination and shadowing on the input image [19], followed by the blocks responsible to extract the feature map used by the Linear SVM when performing object detection.

A visual representation of a possible feature map obtained when applying HOG to an image from the Testing Dataset is shown in Figure 5.5.

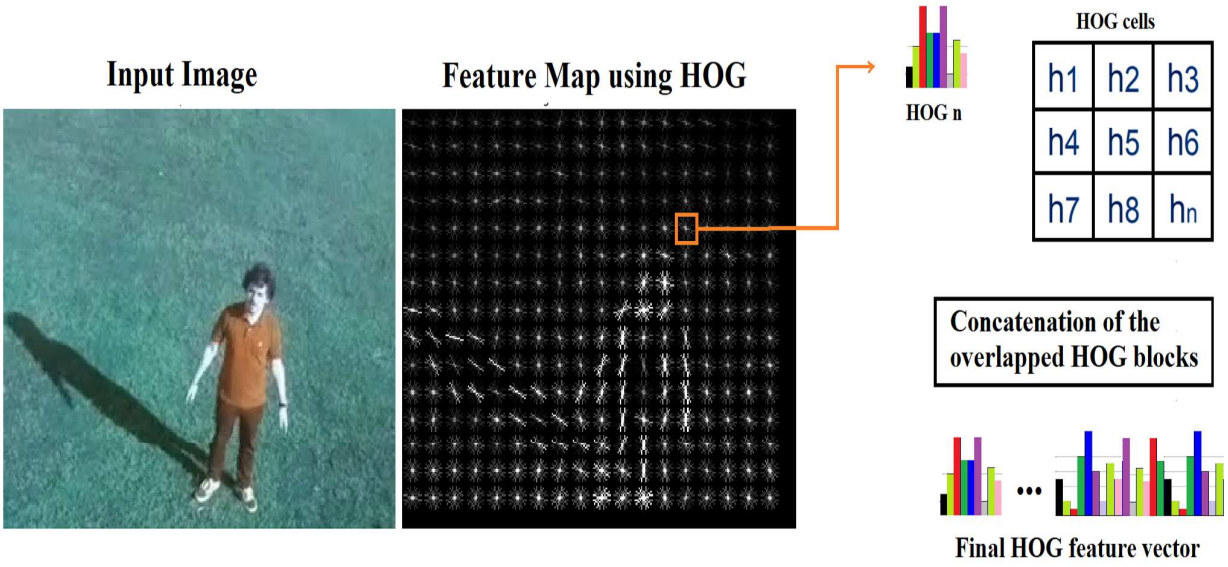


Figure 5.5: Example of an HOG feature map.

In the example shown in Figure 5.5, the input image is divided into cells of 16x16 pixels, each cell then has a local histogram of gradient directions with 9 bins (for 9 possible orientations divided evenly between [0,180]°), where the intensity of each gradient direction shown represents the relative frequency found in the corresponding block. These features can then be used to obtain a prediction of what the input image contains. To achieve this, Navneet Dalal and Bill Triggs [19], use the extracted features and concatenate them into a single vector for each image. This vector is then inputted into a linear SVM to perform the classification task.

To obtain the feature vector necessary as input for the linear SVM, the algorithm proposed scans the image using a sliding window of 64x128 pixels. The sliding window is divided in 16x16 pixel blocks (with 50% overlap or, in other words, 4 pixel stride), and each block consists of 2x2 cells of size 8x8. From each cell, a HOG is extracted with the concatenation of all computed cells's HOGs forming the final feature vector. To allow this method to detect objects found in images at different scales, this sliding window is applied at all different positions with different scales on the original image, as depicted in Figure 5.6.

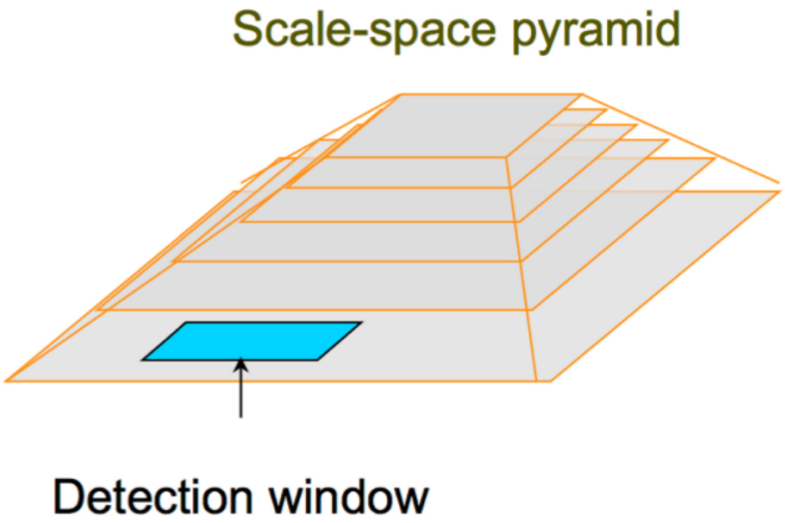


Figure 5.6: Image pyramid formed when applying an SVM detector [20].

The number of sliding windows applied for each image depends, as consequence, on two main factors: the stride and scaling factors used. This will impact on the final resulting time needed to process a given image as well as the level of accuracy. Smaller increments used in the scaling of the image pyramid will result in more fine tuned detection for objects at different scales in respect to the input image, with the disadvantage of requiring more overall sliding windows and consequent processing time. Similarly, smaller values of stride will have the same effect.

5.2.2 Evaluation

To find out the effectiveness and suitability of this implementation when applied to the case study conducted in this dissertation, two main characteristics will be evaluated: the level of accuracy when performing the detection on people being filmed from an airborne UAV and processing time between frames. These values are highly correlated with the stride and scaling factor used. Knowing this the following results, shown in Figure 5.7, are obtained using the image dataset discussed in Section 5.1 and considering only true positive detections that have higher than 50% Intersection over Union (IoU) using Equation 2.7.

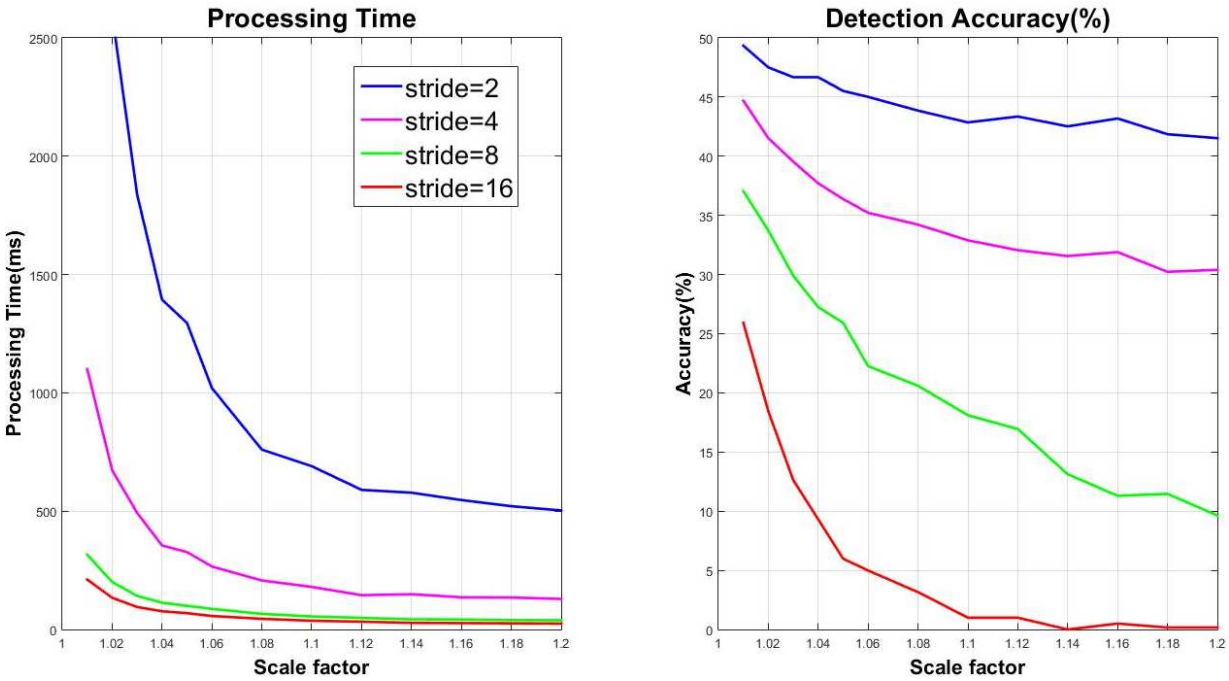


Figure 5.7: Results when applying different values of stride and Scaling factors to the SVM Detector.

As expected, smaller values for the stride and scaling factor result in higher overall accuracy, with the downside of larger required detection processing times. The best value for accuracy achieved resulted only in nearly 50% accuracy, and required an average processing time of 4162 ms. These results are far from the requirements needed to implement real-time detection, rendering this method unsuitable with the hardware used.

5.3 Real-Time Object Detection with CNN

The next method proposed to tackle the task of person detection uses a pre-trained CNN for Real-Time Object Detection. This state of the art object detection CNN, YOLO, is open source and attempts to predict the position of different types of classes of objects present in an image (80 classes for YOLOv2) with people being one of them. Previous work done on the field of object detection used classifiers to perform detection of a specific class of objects, and evaluated it at various locations and scales in the test image [11].

One of the major drawbacks in these types of implementations resides in the fact that the context of each different image being analyzed is not taken in to account, leading to

problems, such as, background patches being mistaken for an object. Also, due to the need to run the classifier in different regions of the image with different scales makes these methods very inefficient in terms the the allocation of computational resources.

YOLO uses a single CNN to predict spatially separated bounding boxes, with associated class probabilities. The input image goes through the network only once to obtain the desired object predictions, hence the name YOLO: You Only Look Once. This network is trained using the ImageNet 1000-class competition dataset [65]. When using the PASCAL VOC 2007 dataset [66], it was able to get a 63,4% Mean Average Precision (mAP) [11]. The first iteration of YOLO used an architecture comprised of 24 convolutional layers followed by 2 Fully Connected (FC) layers and is shown in Figure 5.8.

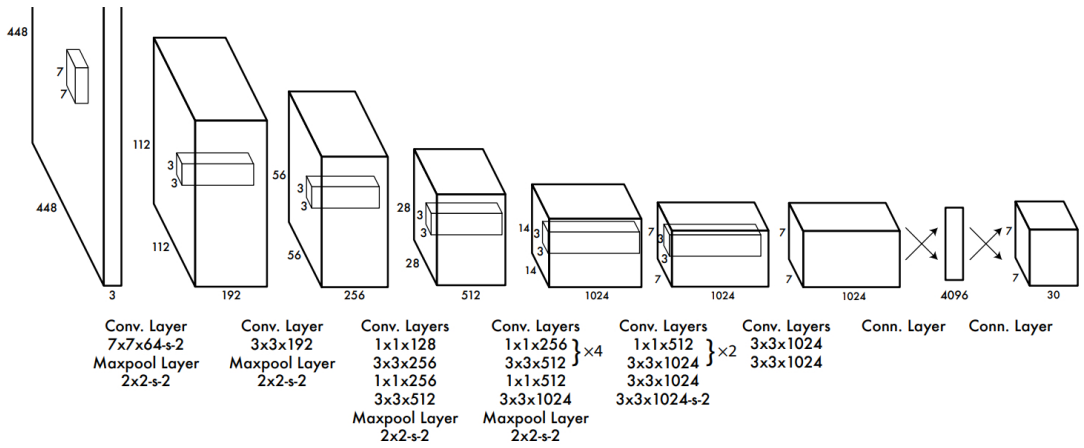


Figure 5.8: YOLO version 1 CNN architecture (from [11]).

In 2017 the version 2 of YOLO was published, improving on the previous performance achieved by version 1, achieving a 78,6% mAP [21] on the PASCAL VOC 2007 image dataset [66]. This updated version of YOLO was chosen to perform the object detection task due to the flexibility in regards to the sizing of the Convolutional Network. This version uses only convolutional and pooling layers, removing the last 2 FC layers from the previous architecture. Removing these last layers makes resizing of the network possible, the only constraint being that the size of the input images must be divisible by 32 (this is due to the 5 pooling layers present in the architecture that reshape previous layers to half their original size). The filters used remain the same (mostly 3x3), and so do the hyperparameters between each layer. An example of a possible CNN compiled using

YOLOv2 is shown in Figure 5.9 where the size of the input chosen is 608x608x3.

```
Building net ...
Source | Train? | Layer description                | Output size
-------+--------+----------------------------------+---------------
       |        | input                            | (?, 608, 608, 3)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 608, 608, 32)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 304, 304, 32)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 304, 304, 64)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 152, 152, 64)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 152, 152, 128)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 152, 152, 64)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 152, 152, 128)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 76, 76, 128)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 76, 76, 256)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 76, 76, 128)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 76, 76, 256)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 38, 38, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 256)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 19, 19, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 19, 19, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 19, 19, 512)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | concat [16]                      | (?, 38, 38, 512)
 Load  |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 64)
 Load  |  Yep!  | local flatten 2x2                | (?, 19, 19, 256)
 Load  |  Yep!  | concat [27, 24]                  | (?, 19, 19, 1280)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
 Load  |  Yep!  | conv 1x1p0_1    linear           | (?, 19, 19, 425)
-------+--------+----------------------------------+---------------
```

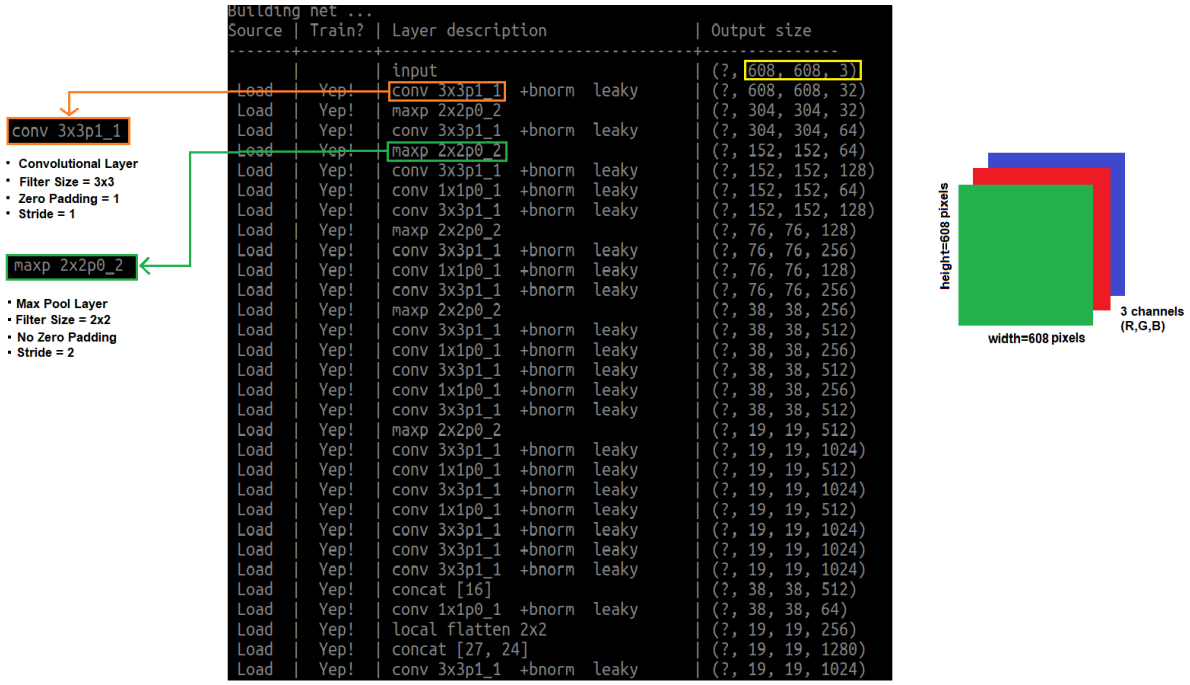
Figure 5.9: Example Architecture of YOLO with input size 608x608x3.

The task of extracting features between layers is computed by every individual convolutional layer, each with a different number and set of filters. To account for nonlinearities between layers, every convolutional layer is preceded by a leaky rectified linear activation expressed by:

$$\phi(x) = \begin{cases} x, & \text{if} \quad x > 0 \\ 0.1x, & \text{otherwise} \end{cases} \tag{5.1}$$

The subsampling between layers is performed by the maxpooling layers. Finally, layers 26 and 29 are used to concatenate the higher resolution features with lower resolution ones.

Figure 5.10 shows in more detail the information outputted to the terminal when building the YOLO version 2 CNN.




```
Building net ...
Source | Train? | Layer description                | Output size
-------+--------+----------------------------------+---------------
       |        | input                            | (?, 608, 608, 3)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 608, 608, 32)
Load   |  Yep!  | maxp 2x2p0_2                     | (?, 304, 304, 32)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 304, 304, 64)
Load   |  Yep!  | maxp 2x2p0_2                     | (?, 152, 152, 64)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 152, 152, 128)
Load   |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 152, 152, 64)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 152, 152, 128)
Load   |  Yep!  | maxp 2x2p0_2                     | (?, 76, 76, 128)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 76, 76, 256)
Load   |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 76, 76, 128)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 76, 76, 256)
Load   |  Yep!  | maxp 2x2p0_2                     | (?, 38, 38, 256)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
Load   |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 256)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
Load   |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 256)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 38, 38, 512)
Load   |  Yep!  | maxp 2x2p0_2                     | (?, 19, 19, 512)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
Load   |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 19, 19, 512)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
Load   |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 19, 19, 512)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
Load   |  Yep!  | concat [16]                      | (?, 38, 38, 512)
Load   |  Yep!  | conv 1x1p0_1  +bnorm  leaky      | (?, 38, 38, 64)
Load   |  Yep!  | local flatten 2x2                | (?, 19, 19, 256)
Load   |  Yep!  | concat [27, 24]                  | (?, 19, 19, 1280)
Load   |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 19, 19, 1024)
```

Figure 5.10: Terminal output for the YOLOv2 CNN of input size 608x608.

Figure 5.11 shows a visual representation of the first 2 layers of the network (Convolutional layer followed by a Pooling layer).

```
Building net ...
Source | Train? | Layer description                | Output size
-------+--------+----------------------------------+---------------
       |        | input                            | (?, 608, 608, 3)
 Load  |  Yep!  | conv 3x3p1_1  +bnorm  leaky      | (?, 608, 608, 32)
 Load  |  Yep!  | maxp 2x2p0_2                     | (?, 304, 304, 32)
```

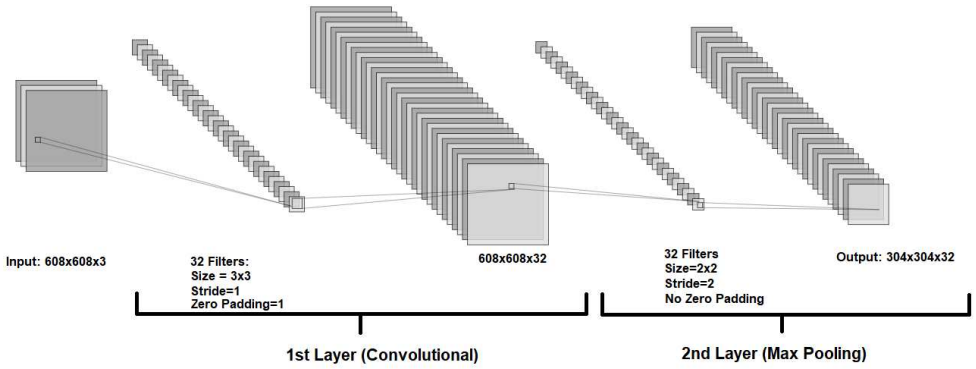


Figure 5.11: First 2 Layers.

The last convolutional layer has a 1x1 filter whose function is to reduce the data to the shape of 19x19x425. This output tensor divides the input image's height and width to a 19x19 grid as shown in Figure 5.12, the corresponding 425 channels contain the data for the bounding boxes associated with each grid cell. Each grid cell has 5 bounding boxes that try to predict an object that is centered around the corresponding grid cell. The bounding box attributes are as follows:

$\mathbf{t_x}, \mathbf{t_y}$ Represent the coordinates for the center of the bounding box relative to the bounds of the respective grid cell.

$\mathbf{t_w}, \mathbf{t_h}$ Represent the width and height of the bounding box relative to the whole image.

$\mathbf{p_o}$ Represents how likely the box is to contain an object (Objectness Score).

$\mathbf{p_x}$ Represents the conditional class probability, meaning, the probability that the detected object belongs to a specific class (e.g., person, dog, chair, bird...).

Figure 5.12 shows a visual representation of the output tensor pertaining to the input image. There are 80 classes being evaluated in the input image, each bounding box has 85

parameters. Since there are 5 bounding boxes per grid, we obtain the previously mentioned output tensor of size 19x19x425.

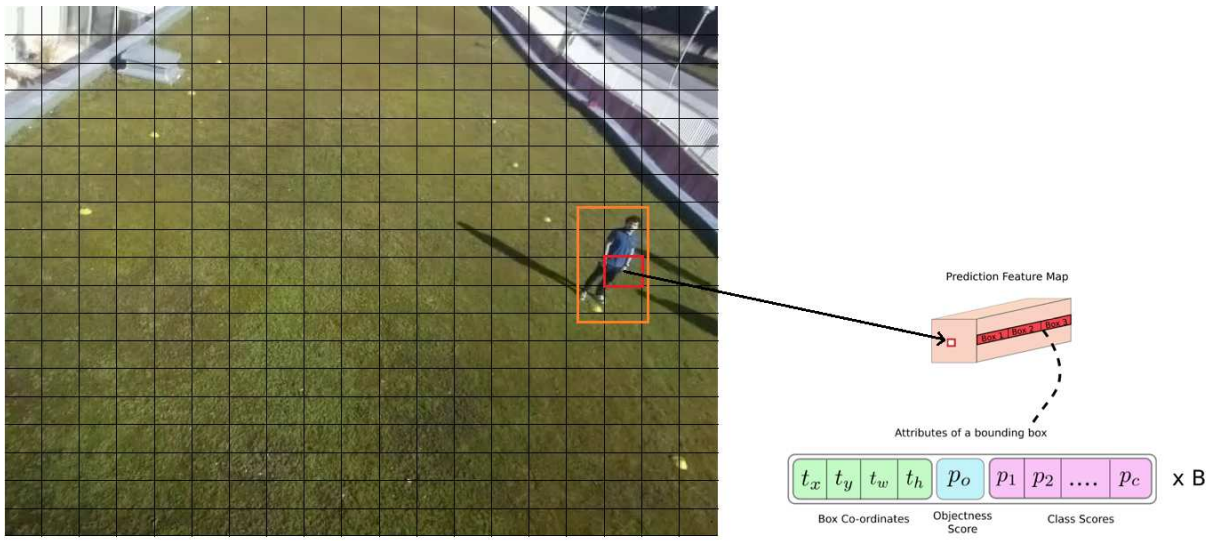


Figure 5.12: Relationship between the output tensor and input image.

If we instead build a network with an input size of, for example 320x320x3, the final output tensor produced will be a 10x10x425. This change in the input size will reduce the number of convolutions per image, leading to faster detection times at the cost of possibly reducing the network's overall accuracy. This behaviour can be seen in Figure 5.13, where different sizes at the input of the network result in differences in both accuracy and rate of detection.

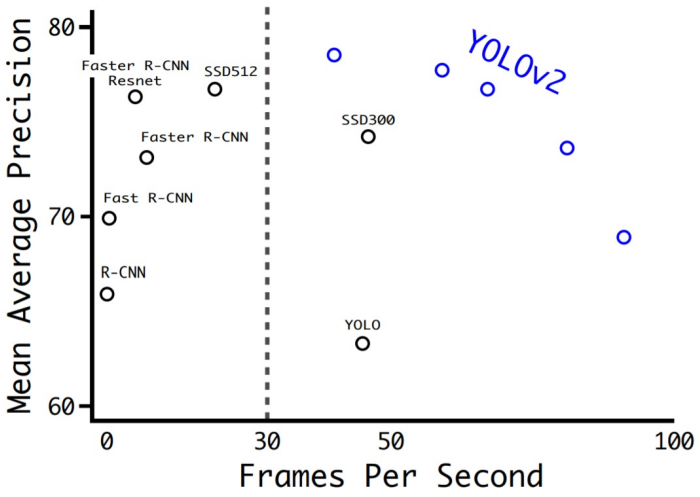


Figure 5.13: Accuracy vs Frames per Second in YOLOv2 (from [21]).

As expected, there is a tradeoff between the accuracy of the predictions and the frame rate achieved when changing the size of the network. The results shown in Figures 5.13 and 5.14 are based on images from the PASCAL VOC 2007 database [53]. This database is made up of 20 different classes, meaning that the mAP value of 78% (using the YOLOv2 544x544 Network Size) is related to all 20 classes evaluated. Since the purpose of using this CNN in our architecture is only to perform detections on people, this value might not be true and skewed either in favor or against the actual value when this method is used only as a human detector.

Detection Frameworks	Train	mAP	FPS
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

Figure 5.14: YOLOv2 results in the PASCAL VOC 2007 using diferent network sizes (from [21]).

5.3.1 Input Net Sizing

Using the dataset described in Section 5.1, the following results, shown in Figure 5.15 are obtained for the average time needed to perform a single detection, and the overall accuracy for different sizes at the input of the Network. In this test we consider a true positive to have at least an IoU of over 50%.

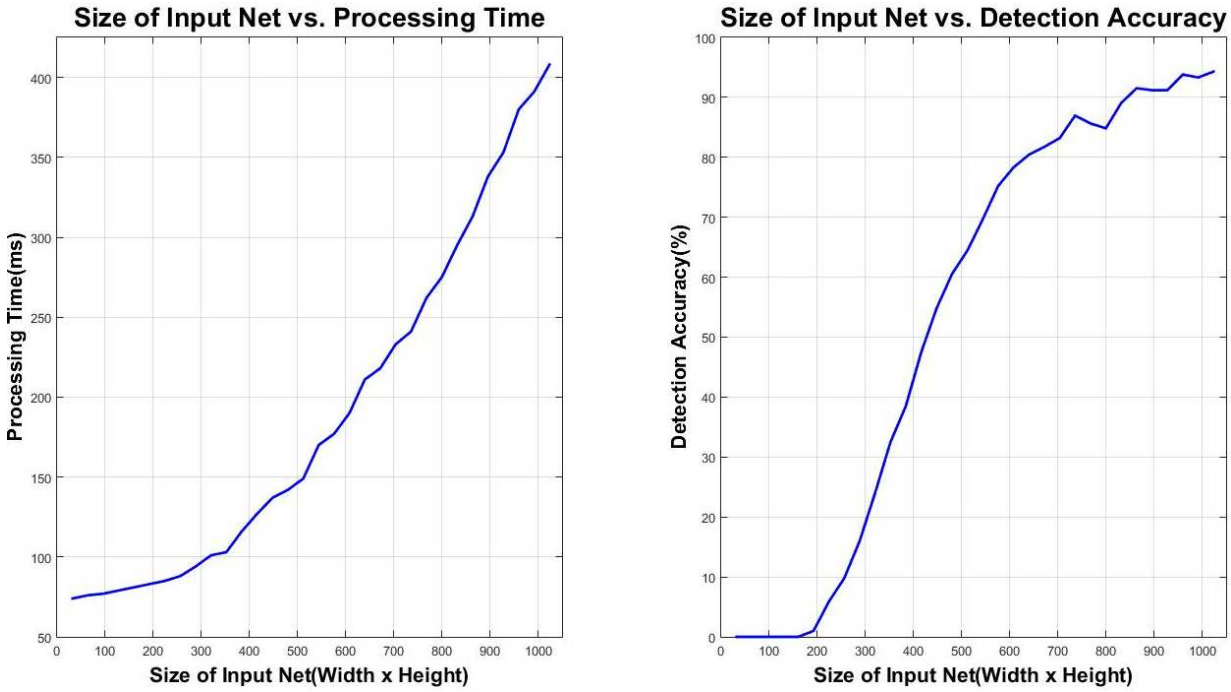


Figure 5.15: Processing Time and Overall detection accuracy for different sizes at the input of the YOLO CNN.

The results obtained demonstrate a large improvement when compared to the ones obtained for the previous object detector, making this method the preferred one to perform the proposed task. As the size of the input net increases, so does the time to perform a detection at a linear rate. This is due to the increase in the number of convolutional operations performed by the Graphics Processing Unit (GPU) as the size of the layers increases accordingly. However, the overall detection accuracy does not behave the same way as we increase the input Net size, leading to diminishing returns in performance as the input Net size increases. Since real-time detection is the main goal, there needs to be a compromise between both accuracy and average required processing time. With a goal of achieving a final value of 3-4 frames per second, the input size of 672x672 was chosen.

This sizing choice corresponds to an average of 218 ms of processing time required and overall accuracy values of 81,7%.

5.3.2 Average Precision Results

To further determine the viability of the choices made in regards to optimizing object detection and described in the previous Sections, another metric that is widely used in computer vision to assess object detectors is the Average Precision (AP). This metric measures the methods' precision at different recall values resulting in a plot called: Precision-Recall curve. The resulting data points in the curve are obtained for a specific ranking method used by the object detector to determine the predicted detection's hierarchy. The results depicted in Figure 5.16 show the precision-recall curves obtained for 3 different levels of IoU thresholds. The ranking method used when computing each curve and corresponding AP result is based on YOLO's confidence threshold level.

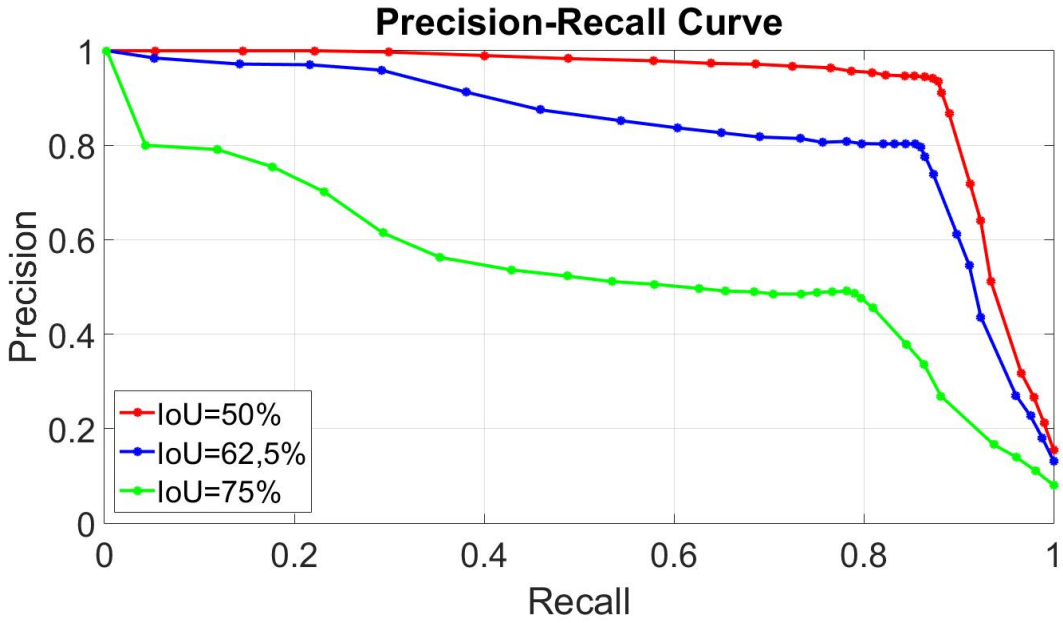


Figure 5.16: Precision Recall curve obtained with the image dataset created for the YOLO CNN at an IoU of 0,75.

Using Equation 2.8, we obtain the results shown in Table 5.1.

	IoU@50%	IoU@62,5%	IoU@75%
Average Precision	88,4%	78,7%	52,59%

Table 5.1: Average Precision results obtained at different IoU levels.

5.3.3 Person Detection Accuracy vs Distance From Drone

Using the same database from Section 5.1, it is also possible to compute the average accuracy of the implemented CNN when detecting people at different distances from the UAV.

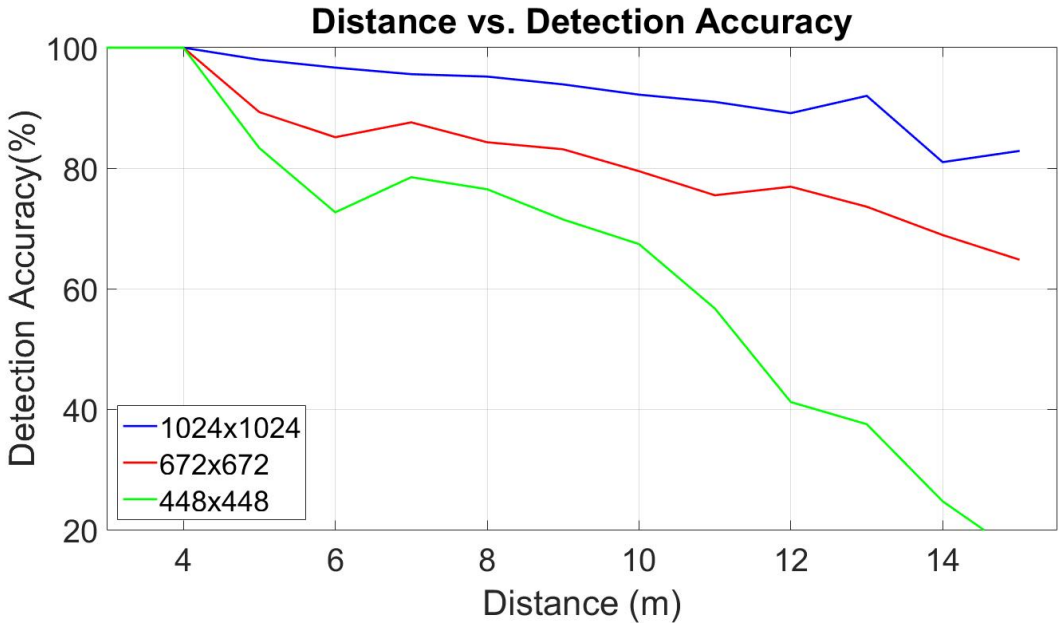


Figure 5.17: Detection Accuracy as a function of distance to object (person) for YOLOv2 using our Image Dataset.

Figure 5.17 shows the results obtained for the detection accuracy when the target distance ranges from 3 to 15 meters from the UAV. This distance corresponds to the upper and lower limits of distance to the target aimed to be maintained when the UAV is performing a tracking mission. The accuracy is measured for 3 different sizes at the input of the CNN.

5.4 Tracking Experiment

For the experimental evaluation, a wide open area located inside the University of Aveiro campus was chosen. For each tracking mission test, a drone starts to fly to a designated starting position. Once the drone reaches the desired location and hovering at 5 meters above the ground, the video feed is initiated. Next, a person who is the experiment's target proceeds to get in range of the drone's camera field of view, triggering the first movement command issued to the drone. An already planned path is then traversed by the target. The path taken by both the drone and the person being tracked in one of the experiments is shown in Figure 5.18. Each drone position depicted is labelled with the drone's current orientation (the values shown follow the coordinate system described previously in Section 4.3.1).

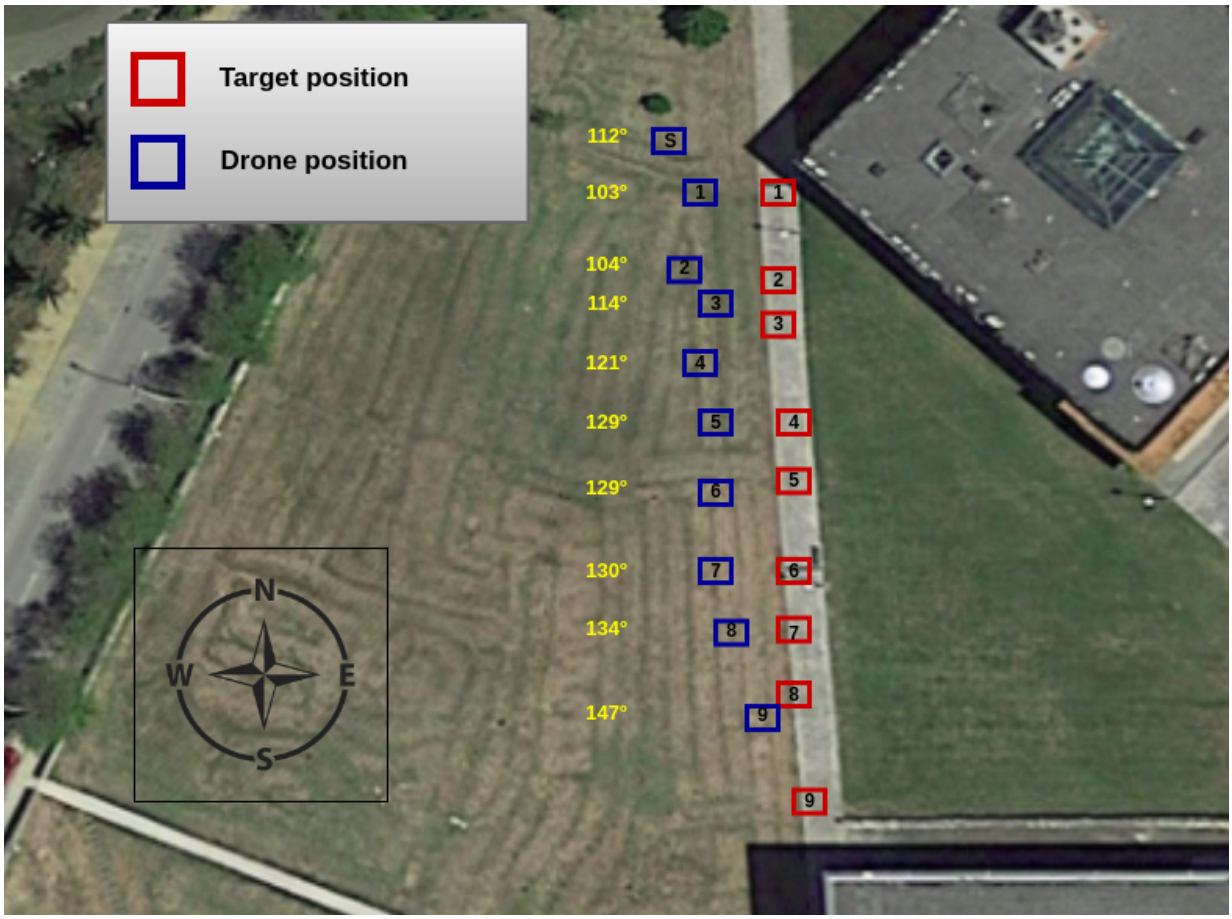


Figure 5.18: Path followed by UAV and target person during tracking experiment.

The heading of the drone is constantly changing its value as it moves from point to point. This can be perceived in Figure 5.18 with the changes in the position the drone takes relative to the person, as it tracks the person during this experiment. Table 5.2 shows in more detail the telemetry data obtained during the experiment. The data shown characterizes the drone's GPS position and heading values, as each movement command is issued by the object detection endpoint, as well as the time elapsed since the start of the experiment.

	GPS Position	**Command Issued**	**Time Elapsed**	**Heading**
S	40°63'41.99"N 8°66'04.32"W	**NS**=-6.5m **WE**=2.1m	0:02s	112°
1	40°63'41.41"N 8°66'04.13"W	**NS**=-5.7m **WE**=-2.2m	0:08s	103°
2	40°63'40.89"N 8°66'04.33"W	**NS**=-4.3m **WE**=3.8m	0:12s	104°
3	40°63'40.50"N 8°66'03.99"W	**NS**=-4.8m **WE**=-1.5m	0:17s	114°
4	40°63'40.07"N 8°66'04.13"W	**NS**=-4.2m **WE**=1.5m	0:23s	121°
5	40°63'39.69"N 8°66'04.00"W	**NS**=-5.5m **WE**=-1.8m	0:29s	129°
6	40°63'39.20"N 8°66'04.16"W	**NS**=-6.5m **WE**=1m	0:37s	129°
7	40°63'38.62"N 8°66'04.07"W	**NS**=-5.1m **WE**=1.0m	0:42s	130°
8	40°63'38.16"N 8°66'03.98"W	**NS**=-7.9m **WE**=3.0m	0:49s	134°
9	40°63'37.45"N 8°66'03.71"W		0:55s	147º

Table 5.2: Telemetry data from the conducted tracking experiment.

The average speed by the target obtained for this tracking experiment is approximately 3,4Km/s, which corresponds to a normal walking speed for the average person. The maxi-

mum speed allowed for tracking a given target with this implementation is mainly dependent on the rate at which the movement command is issued to the drone. This rate is given by the number of frames required to fill the **Position_Stack** (described in Chapter 4.3) with valid detection inputs. In order to obtain these valid inputs, the video feed must be gathered with the drone in stationary hovering position. This means that, while the drone is performing the issued movement command and has not reached the next stationary position, there will be a significant amount of frames with no validated **Position_Stack**. After a stationary position is reached by the drone, new valid detection inputs will be added until the **Position_Stack** is filled with valid detections triggering the next command to be sent to the drone. The relative weight that both these factors have on the rate at which new movement commands are issued is depicted in Figure 5.19.

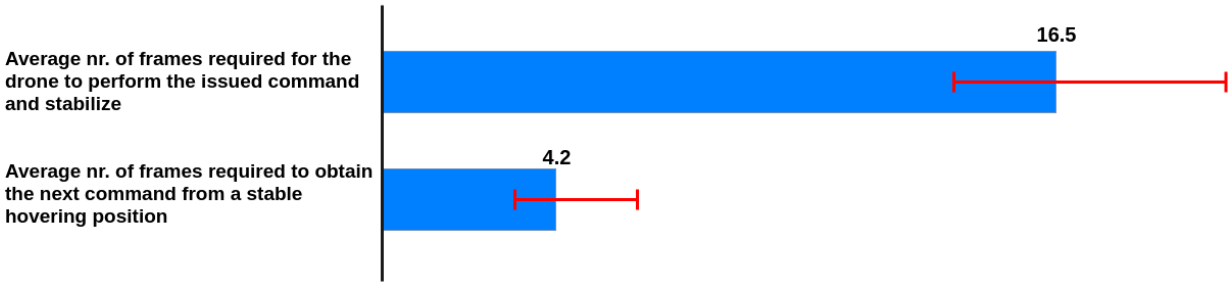


Figure 5.19: Average frames required between each stage of drone flight.

Figure 5.19 clearly shows that the major limiting factor on the tracking speed cap is due to the time required to obtain a stable video feed from the drone between each command. Adding camera stabilization in a future implementation would increase the rate of issued commands, and therefore, the maximum speed that a target can be tracked at.

5.5 Summary

This chapter tested and evaluated two different machine learning algorithms used for object detection on images in order to determine which one would be more suitable for real-

time human detection. The method using CNNs was proven to be the superior approach in both terms of: time between detections and overall accuracy on the predictions. Finally it was presented the evaluation scenario used to assess the overall performance of real-time person tracking relative to the proposed implementation. The results obtained demonstrate that real-time person tracking is achievable for average walking speeds, with the largest constraint experienced when trying to achieve better performance in the system is the usage of an on-board camera without stabilization.

Chapter 6

Conclusions and Future Work

This thesis presents the implementation of a system architecture capable of performing real-time tracking of people using Unmanned Aerial Vehicle (UAV)s. To achieve this goal, the UAV platform described in [24] is used to accomplish drone control and flight data acquisition.

The method used to detect in-frame objects that belong to the person class uses a Convolutional Neural Network (CNN) architecture and is described in [11]. This method enabled a detection rate of approximately 4,6 frames per second (or 218 ms per detection) at a resolution of 800x600 pixels with an Average Precision (AP) of 88,4%, 78,7%, 52,6% at 0.5, 0.625 and 0.75 Intersection over Union (IoU), respectively, for the person class, and thus, legitimizing the possibility of performing real-time object detection on a video feed for tracking purposes. This result was obtained for an image dataset created specifically for this dissertation's purpose. The dataset created is comprised solely of imagery taken from the perspective of a hoovering UAV at an open terrain and featuring persons at different distances from the drone displaying different types of poses. To obtain these object detection performance results, we had to both introduce some modifications on the "vanilla" version of the You Only Look Once (YOLO) CNN architecture, namely the size at the input net and detection threshold values, as well as optimizing the hardware used for convolutional matrices operations.

In order to achieve the desired tracking behaviour with a person as the target subject, a control logic based around the output obtained from the object detector used was developed. The implemented architecture is comprised of several modules connected in such a way as to form a feedback loop between the video being gathered on the airborne

UAV and the positioning commands issued over to it. These commands are in turn based on the pixel position of the target obtained via the employed object detector, and aim to continuously position the UAV in such a way so as to center the target being tracked at the center of the video feed.

Finally, the tracking experiments conducted showed that the developed system is capable of performing tracking of an individual at normal walking speeds; further performance boosts requiring a method of stabilizing the video feed collected at the UAV.

6.1 Future Work

The work conducted during this dissertation implements and interconnects the basic building blocks required to achieve the desired objective of performing real-time tracking of people using UAVs.

Although the aim of this dissertation was achieved, there are some elements that can be improved to increase the overall performance obtained. Examples of possible improvements are the following:

Gimbal Camera Support A gimbal is an hinged support that enables the rotation of an object about a single axis. The addition of this component on the drone to work as the camera's support would enable stabilized video feeds to be acquired by an active UAV and, thus, increase both the performance of the employed object detector as well as allowing for less time required to issue each drone positioning command.

Object Detection Performance The hardware used throughout this dissertation to implement the YOLO CNN is far from what current technology is capable of performance wise. Upgrading the hardware used would lead to the consequent increase in object detection performance, would enable higher resolutions for the UAV's gathered video feed, which would in turn lead to a more refined object detection allowing for objects standing at a greater distance to be detected.

Camera Calibration and Distance to Target Estimation The method used to obtain estimated distances from UAV to target, although effective, as shown in the tracking experiment conducted, is limited in its usage. This method constrains the UAV's allowed elevation from the ground to 5 meters while also accruing errors in

the distance estimation as the terrain inclination increases or decreases drastically. A more accurate and flexible method would lift these constrains and allow for expansion of possible tracking scenarios.

Expand list of trackable object types The object detector used, YOLO, allows for the detection of many different types of objects (200 possible classes for YOLOv2), meaning, the system implemented is already capable of tracking objects on the ground. This assumption is only valid for classes of objects that would have similar levels of AP as the person class when photographed from the airborne UAV. Additionally, since there are 200 possible types of objects able to be detected, means there is a whole new range of applications possible to implement besides the object tracking ones.

Mobile Access Point The airborne UAV can be used to provide Internet access to the person on the ground for either personal use or the retrieval of information from possible sensors in his possession.

Bibliography

[1] R. Szeliski, *Computer Vision: Algorithms and Applications*, 1st ed. Berlin, Heidelberg: Springer-Verlag, 2010.

[2] H. G. Jung, D. S. Kim, P. J. Yoon, and J. Kim, "Structure analysis based parking slot marking recognition for semi-automatic parking system," in *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 2006, pp. 384–393.

[3] S. Pereira, A. Pinto, V. Alves, and C. A. Silva, "Brain tumor segmentation using convolutional neural networks in mri images," *IEEE Transactions on Medical Imaging*, vol. 35, no. 5, pp. 1240–1251, May 2016.

[4] Redmon, Joseph, "Yolo - you only look once," 2018, [Online; accessed November, 2018]. [Online]. Available: https://pjreddie.com/darknet/yolo/

[5] G. Chen, "Object detection with large intra-class variation," Ph.D. dissertation, University of Missouri–Columbia, 2011.

[6] J. Vojt, "Deep neural networks and their implementation." M.S. thesis, Charles University, Faculty of Mathematics and Physics, Prague, CZ, 2016.

[7] Y.-M. Chiang, L.-C. Chang, and F.-J. Chang, "Comparison of static-feedforward and dynamic-feedback neural networks for rainfall–runoff modeling," *Journal of hydrology*, vol. 290, no. 3-4, pp. 297–311, 2004.

[8] S. Olavi, "Master thesis: Object detection from images using convolutional neural networks neural networks." M.S. thesis, Aalto University, School of Science, Otaniemi, FI, 2017.

[9] U. Matej, "Methods for increasing robustness of deep convolutional neural networks." M.S. thesis, Halmstad University, Halmstad, SE, 2015.

[10] J. Redmon, "Yolo: Real-time object detection," 2016. [Online]. Available: https://pjreddie.com/darknet/yolov2/

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

[12] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun 2010. [Online]. Available: https://doi.org/10.1007/s11263-009-0275-4

[13] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. New York, NY, USA: Cambridge University Press, 2007.

[14] A. C. Lorena and A. C. de Carvalho, "Uma introdução às support vector machines," *Revista de Informática Teórica e Aplicada*, vol. 14, no. 2, pp. 43–67, 2007.

[15] A. Rosebrock, "Intersection over union (iou) for object detection - pyimagesearch," 2016. [Online]. Available: https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[16] DJI, ""flame wheel arf kit"," 2018. [Online]. Available: https://www.dji.com/pt/flame-wheel-arf

[17] R. P. Foundation, "Raspberry pi 2 model b." [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

[18] "Camera module v2 - raspberry pi," 2017. [Online]. Available: https://www.raspberrypi.org/products/camera-module-v2/

[19] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, June 2005, pp. 886–893 vol. 1.

[20] S. Fidler, "Csc420: Intro to image understanding," University Lecture, University of Toronto, 2012.

[21] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525, 2017.

[22] A. Chapman, ""types of drones: Multi-rotor vs fixed-wing vs single rotor vs hybrid vtol"," June, 2016. [Online]. Available: https://www.auav.com.au/articles/drone-types/

[23] "Stream formats supported by ffserver," 2018. [Online]. Available: https://www.ffmpeg.org/sample.html

[24] B. Areias, N. Humberto, L. Guardalben, J. M. Fernandes, and S. Sargento, "Towards an automated flying drones platform," 01 2018, pp. 529–536.

[25] J. F. Keane and S. S. Carr, "A brief history of early unmanned aircraft," *Johns Hopkins APL Technical Digest*, vol. 32, no. 3, pp. 558–571, 2013.

[26] D. Gonzalez-Aguilera and P. Rodriguez-Gonzalvez, "Drones—an open access journal," *Drones*, vol. 1, no. 1, 2017. [Online]. Available: http://www.mdpi.com/2504-446X/1/1/1

[27] Y. A. Gunchenko, S. A. Shvorov, V. I. Zagrebnyuk, V. U. Kumysh, and E. S. Lenkov, "Using uav for unmanned agricultural harvesting equipment route planning and harvest volume measuring," in *2017 IEEE 4th International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*, Oct 2017, pp. 262–265.

[28] Y. A. Pederi and H. S. Cheporniuk, "Unmanned aerial vehicles and new technological methods of monitoring and crop protection in precision agriculture," in *2015 IEEE International Conference Actual Problems of Unmanned Aerial Vehicles Developments (APUAVD)*, Oct 2015, pp. 298–301.

[29] J. A. Paredes, J. González, C. Saito, and A. Flores, "Multispectral imaging system with uav integration capabilities for crop analysis," in *2017 First IEEE International Symposium of Geoscience and Remote Sensing (GRSS-CHILE)*, June 2017, pp. 1–4.

[30] A. M. Goncalves-Coelho, L. C. Veloso, and V. J. A. S. Lobo, "Tests of a light uav for naval surveillance," in *OCEANS 2007 - Europe*, June 2007, pp. 1–4.

[31] T. L. Craft, C. F. Cahill, and G. W. Walker, "Using an unmanned aircraft to observe black carbon aerosols during a prescribed fire at the rxcadre campaign," in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2014, pp. 77–82.

[32] M. C. L. Patterson, D. Osbrink, A. Brescia, D. Downer, J. Etro, and J. Cione, "Atmospheric and ocean boundary layer profiling with unmanned air platforms," in *2014 Oceans - St. John's*, Sep. 2014, pp. 1–7.

[33] H. G. Jung, D. S. Kim, P. J. Yoon, and J. Kim, "Structure analysis based parking slot marking recognition for semi-automatic parking system," in *Structural, Syntactic, and Statistical Pattern Recognition*, D.-Y. Yeung, J. T. Kwok, A. Fred, F. Roli, and D. de Ridder, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 384–393.

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[36] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, Aug 2017, pp. 1–6.

[37] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154. [Online]. Available: https://physoc.onlinelibrary.wiley.com/doi/abs/10.1113/jphysiol.1962.sp006837

[38] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[39] N. Huang, J. He, and N. Zhu, "A novel method for detecting image forgery based on convolutional neural network," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, Aug 2018, pp. 1702–1705.

[40] J. Rathod, V. Wazhmode, A. Sodha, and P. Bhavathankar, "Diagnosis of skin diseases using convolutional neural networks," in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, March 2018, pp. 1048–1051.

[41] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 2146–2153.

[42] M. Betke and N. C. Makris, "Fast object recognition in noisy images using simulated annealing," in *Computer Vision, 1995. Proceedings., Fifth International Conference on.* IEEE, 1995, pp. 523–530.

[43] A. L. Yuille, P. W. Hallinan, and D. S. Cohen, "Feature extraction from faces using deformable templates," *International journal of computer vision*, vol. 8, no. 2, pp. 99–111, 1992.

[44] C. P. Papageorgiou, M. Oren, and T. Poggio, "A general framework for object detection," in *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, Jan 1998, pp. 555–562.

[45] S. Boughorbel, J.-P. Tarel, and F. Fleuret, "Non-mercer kernels for svm object recognition." in *BMVC*, 2004, pp. 1–10.

[46] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT '92. New York, NY, USA: ACM, 1992, pp. 144–152. [Online]. Available: http://doi.acm.org/10.1145/130385.130401

[47] S. Balasundaram and N. Kapil, "Application of lagrangian twin support vector machines for classification," in *2010 Second International Conference on Machine Learning and Computing*, Feb 2010, pp. 193–197.

[48] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[49] K.-R. Muller, S. Mika, G. Ratsch, K. Tsuda, and B. Scholkopf, "An introduction to kernel-based learning algorithms," *IEEE transactions on neural networks*, vol. 12, no. 2, pp. 181–201, 2001.

[50] M. Schmidt and H. Gish, "Speaker identification via support vector classifiers," in *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, vol. 1. IEEE, 1996, pp. 105–108.

[51] Y. Li, S. Gong, and H. Liddell, "Support vector regression and classification based multi-view face detection and recognition," in *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition (Cat. No. PR00580)*. IEEE, 2000, pp. 300–305.

[52] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *European conference on machine learning*. Springer, 1998, pp. 137–142.

[53] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International Journal of Computer Vision*, vol. 111, no. 1, pp. 98–136, Jan. 2015.

[54] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[55] G. Salton and M. J. McGill, *Introduction to modern information retrieval*. McGraw-Hill, Inc., 1986.

[56] F. Bellard, "Ffmpeg," 2012. [Online]. Available: https://ffmpeg.org/

[57] L. Xiaohua, J. Xiuhua, and W. Caihong, "Design and implementation of a real-time video stream analysis system based on ffmpeg," in *Software Engineering (WCSE), 2013 Fourth World Congress on*. IEEE, 2013, pp. 212–216.

[58] Thtrieu, "Darkflow," 2017. [Online]. Available: https://github.com/thtrieu/darkflow

[59] Google, "Tensorflow," 2018. [Online]. Available: https://www.tensorflow.org/api_guides/python/upgrade

[60] I. Corporation, "Opencv," 2018. [Online]. Available: https://opencv.org/opencv-3-0.html

[61] N. Corp., "Cuda toolkit 9.0," 2017. [Online]. Available: https://developer.nvidia.com/cuda-90-download-archive

[62] Nvidia, "cudnn," 2018. [Online]. Available: https://developer.nvidia.com/cudnn

[63] C. Papageorgiou and T. Poggio, "A trainable system for object detection," *International journal of computer vision*, vol. 38, no. 1, pp. 15–33, 2000.

[64] E. Maggiori, Y. Tarabalka, G. Charpiat, and P. Alliez, "Can semantic labeling methods generalize to any city? the inria aerial image labeling benchmark," in *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2017.

[65] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[66] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results," http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.