Universidade de Aveiro Departamento de Engenharia Mecânica
2013

Pedro Batista e Silva

Visual Pedestrian Detection using Integral Channels for ADAS

Deteção Visual de Peões usando Canais Integrais para Ajuda à Condução

Universidade de Aveiro Departamento de Engenharia Mecânica
2013

Pedro Batista e Silva

Visual Pedestrian Detection using Integral Channels for ADAS

Deteção Visual de Peões usando Canais Integrais para Ajuda à Condução

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestrado em Engenharia Mecânica, realizada sob orientação científica de Vítor Manuel Ferreira dos Santos, Professor Associado do Departamento de Engenharia Mecânica da Universidade de Aveiro.

O júri / The jury

Presidente / President

Prof. Doutor Jorge Augusto Fernandes Ferreira
Professor Auxiliar da Universidade de Aveiro

Vogais / Committee

Doutor Cristiano Premebida
Investigador da Universidade de Coimbra

Prof. Doutor Vítor Manuel Ferreira dos Santos
Professor Associado da Universidade de Aveiro (orientador)

Agradecimentos / Acknowledgements

Em primeiro lugar dirijo um sentido agradecimento ao Professor Vitor Santos, que foi uma verdadeira fonte de motivação ao longo dos últimos anos.

Aos meus pais e irmão, cujo amor incondicional sempre foi e será um autêntico pilar na minha vida.

Aos meus verdadeiros amigos com quem partilhei tantos momentos e experiencias nestes últimos anos em Aveiro, esta página não seria suficiente para descrever o quão importantes são para mim.

A todos os colegas do LAR, especialmente ao Jorge que nunca deixou de parar o que quer que estivesse a fazer para ajudar.

To Piotr Dollár, for kindly answering my e-mails.

A nós os 7, sempre.

Keywords Pedestrian detection, Integral channels, Feature extraction, AdaBoost, Classification, Pedestrian dataset

Abstract

This work exploits a vision based pedestrian recognition technique to build a framework capable of performing detection in images of urban setting. This method takes advantage of the richness of information present in multiple channels of an image to assemble different detection techniques in a simple and generic infrastructure. The general idea behind this method is that pedestrians present specific visual properties that can be used to differentiate them apart from a scene. So, to exploit such properties, features are extracted from the channels in an optimized manner through the use of integral images and a machine learning engine based on AdaBoost is trained through positive and negative examples the relationship of those features with the presence of pedestrians on visual data. This framework then integrates a multi-scale, sliding window approach to perform visual pedestrian detection. To evaluate the algorithm, tests were carried out in two distinct datasets and results confirm its validity.

Palavras-chave Deteção de peões, Canais integrais, Descritores, AdaBoost, Classificação, Dataset de peões

Resumo

Este trabalho explora uma técnica de deteção visual de peões que visa estabelecer uma estrutura capaz de realizar detecção em imagens de ambiente urbano. O método tira partido da riqueza de informação presente em múltiplos canais da imagem para integrar diferentes técnicas de deteção numa estrutura simples e genérica. Esta técnica tem como base a idéia de que um peão apresenta propriedades visuais específicas que permitem a sua distinção do meio envolvente. Então, com vista a explorar essas propriedades, são recolhidos descritores dos múltiplos canais de uma forma otimizada com o uso da imagem integral, e, através de exemplos positivos e negativos, um algorítmo de aprendizagem baseado no AdaBoost é treinado para relacionar esses descritores com a presença de peões. Esta técnica integra uma estrutura de varrimento de imagem a múltiplas escalas para efectuar o reconhecimento visual de peões. O método é testado em dois conjuntos de dados diferentes e os resultados confirmam a sua validade.

Contents

Contents i

List of Tables iii

List of Figures v

1 Introduction 1
1.1 Motivation and Framework . . . 1
1.2 Description of the Problems . . . 2
1.3 State of the Art . . . 3
1.3.1 Sliding Window Detectors . . . 3
1.3.2 Multi-sensor Detectors . . . 7
1.4 Proposed Solution . . . 8
1.5 Development Tools . . . 9
1.5.1 Robotic Operation System . . . 9
1.5.2 OpenCV . . . 10
1.5.3 INRIA Dataset . . . 10
1.6 Experimental Setup . . . 11

2 Integral Channel Features 13
2.1 Channels . . . 13
2.1.1 Gradient Magnitude . . . 14
2.1.2 Gradient Histogram . . . 15
2.1.3 LUV color channels . . . 16
2.2 Integral Images . . . 18
2.3 Feature extraction . . . 18
2.4 Multi-scale Image Analysis . . . 20
2.5 Parametrization Summary . . . 21

3 Classification 23
3.1 AdaBoost . . . 24
3.2 Training Process . . . 26
3.2.1 Training Samples . . . 26
3.2.2 Classifier Parametrization . . . 26
3.3 Post Processing . . . 27
3.4 Computation Time . . . 28

4 Experiments and Results **31**
4.1 INRIA Dataset Experiments 31
4.1.1 Ground Truth 31
4.1.2 Methodology 32
4.1.3 Results 32
4.1.4 Commentary 32
4.2 *Atlas* Pedestrian Dataset Experiments 33
4.2.1 Ground Truth 33
4.2.2 Methodology 37
4.2.3 Results 38
4.2.4 Commentary 45

5 Conclusions and Future Work **47**

References **49**

A Usage Instructions **52**

List of Tables

2.1 Default parametrization summary 21

3.1 AdaBoost parameters 27

4.1 Atlas Pedestrian Dataset Results 38

List of Figures

1.1 *Atlas Car.* 2
1.2 Problems associated with pedestrain detection 3
1.3 schematic depiction of a detection cascade 4
1.4 Viola and Jones's face detection algorithm 4
1.5 Overview of HOG detection algorithm 5
1.6 Example results of the shape-based pedestrian detection method 5
1.7 Multi-feature detector performance 6
1.8 Examples of channels computed from an image 7
1.9 Schematic of a simple ROS communication architecture. 9
1.10 Examples of positive windows 11
1.11 Schematic representation of the experimental setup 12

2.1 Channel combination 14
2.2 Gradient magnitude computation 15
2.3 Detection performance of different numbers of bin orientations 15
2.4 Gradient histogram computation 16
2.5 Detection performance of different color channels 17
2.6 LUV color channels computation 17
2.7 Finding the sum of a rectangular region 18
2.8 Examples of random features 19
2.9 Dense image pyramid 20
2.10 Multi-scale image processing 21

3.1 Aggregation of *weak classifiers* 24
3.2 Rectangles generated by the detector 27
3.3 Merged rectangles 28
3.4 Difference between single boosted classifier and a cascade 29

4.1 Detector performance on the INRIA dataset 32
4.2 Ground truth example 1 34
4.3 Ground truth example 2 35
4.4 Ground truth example 3 35
4.5 Ground truth example 4 36
4.6 Ground truth example 5 36
4.7 Ground truth example 6 37
4.8 Example result 1 39
4.9 Example result 2 39

4.10 Example result 3 40
4.11 Example result 4 40
4.12 Example result 5 41
4.13 Example result 6 41
4.14 Example result 7 42
4.15 Example result 8 42
4.16 Example result 9 43
4.17 Example result 10 43
4.18 Example result 11 44
4.19 Example result 12 44

Chapter 1

Introduction

1.1 Motivation and Framework

Humans are unmistakably the most important components of a machine's environment. Whenever people are involved in a process with any associated risks, there is a great number of special security rules that must be followed in order to assure their safety. Visual detection of humans is a field with an extensive range of applications such as robotics, entertainment, surveillance, care for the elderly and disabled, road safety and others. In all of these, the knowledge of the presence of a person allows the equipment with whom it's interacting to act accordingly, be it sounding an alarm, stopping an operation, or any other action. The benefits of this become obvious when we think about a car being driven in an urban scenario. If the circumstances are such that the driver is always aware of the surrounding pedestrians, danger of accidents involving them would most likely decrease dramatically. One could even think of a safety mechanism that refrains a driver's action in a dangerous situation for a pedestrian.

In the European Union, 21% (European Comission, 2011 [19]) of all traffic fatalities are pedestrians, indicating that we're looking at a matter of great importance. Motivated by this, many researchers have devoted much work in developing algorithms for visual human detection, leading to extraordinary improvements in the past decade. Despite those significant improvements, there is still much room for progress due to the challenging nature of the problem. Issues like varying lighting conditions or uncertain pedestrian postures require robust solutions in order to overcome those difficulties.

In this work, an algorithm capable of visually detecting pedestrians achieving close to state-of-the-art detection rate is implemented and exploited. The objective was to build a base detector to be inserted in the *Atlas* project [6], of the Department of Mechanical Engineering of the University of Aveiro. This is an ongoing team project that started with the aim of participating in autonomous mobile robots competitions and has since then is grown into real road vehicles (Figure 1.1) with the goal of developing new Advanced Driver Assistance Systems (ADAS). Visual detection of pedestrians is an obvious and paramount feature that had yet to be incorporated in the project, making this work a significant contribution. The work presented in this document constitutes a base application that has still much room for improvement in the future, as this is a subject in constant development.

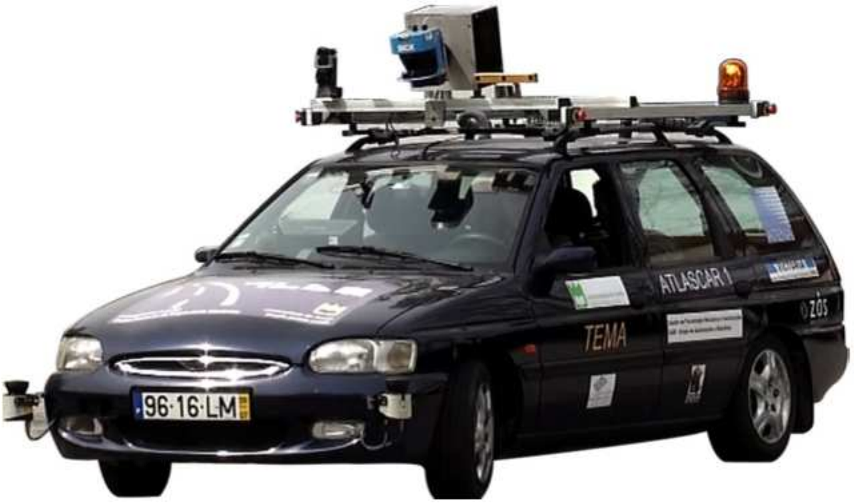

Figure 1.1: *Atlas Car.*

1.2 Description of the Problems

Visual pedestrian detection is a challenging task with a set of complex problems to overcome. In this section, an overview of some common problems associated with detecting pedestrians in individual monocular images will be presented.
Computer Vision (CV) is a technology that has grown in presence on many fields of society over the past two decades. In industry, product inspection systems have significantly improved with the aid of CV by allowing inspection of parts at a major scale, a fact that lead to considerable advancements in the process of finding defects. In such environments, a careful setup is planned in order to facilitate the processing of the images outputted by the camera, since controlling the lighting level, background color and other external parameters is of utmost importance for an easy object segmentation, meaning, separating the object of interest from the background.
On the contrary, it is virtually impossible to control the external factors of the images where pedestrians must be detected, precluding the possibility of segmentation and causing the need to process cluttered, random images with huge amounts of information. The unpredictability of the location where a pedestrian might come into sight also mandates the analysis of the whole scene. Moreover, the varying nature of the lighting conditions caused either by changes in the daylight, or different weather conditions further hampers the task. Another typical problem that leads to relatively high miss rates is that pedestrians often appear partially occluded by other objects in the scene, such as trees, traffic signs, bikes, and even by other pedestrians. The uncertainty of their posture also constitutes a problem, since it is obvious that an up-right pedestrian has different properties in an image than one sitting down or leaning into another object.
In sum, the mission is to detect pedestrians that might or not be partially occluded, in unpredictable locations, assuming different stances, on cluttered scenes with varying lighting conditions. Such conditions demand highly robust algorithms which are typically heavy and unable to run at the frame-rates that this task demands. Figure 1.2 attempts

to illustrate some of these problems.

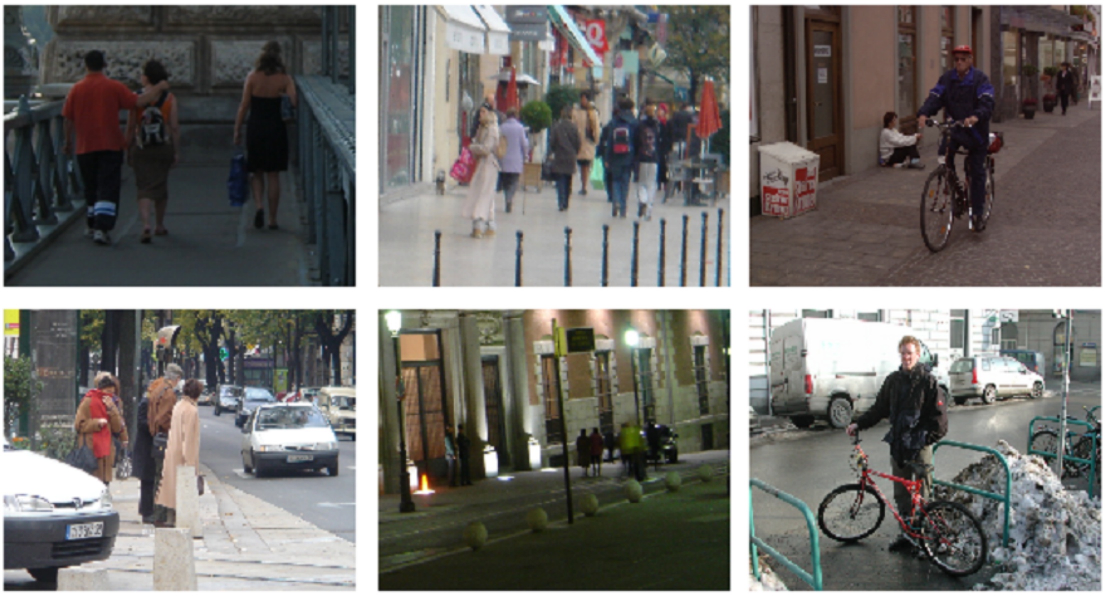

Figure 1.2: Varying lighting conditions, partial occlusion and different postures are some of the problems associated with pedestrian detection. Images taken from the INRIA dataset (Dalal and Triggs, 2005 [4]).

1.3 State of the Art

A great development has been made on the subject of visual pedestrian detection in the past two decades. In this section a compact description of some notorious contributions for this area will follow. This review will focus firstly on detectors with a sliding window approach, often seen as the most promising for low and medium resolution approaches. Secondly, some multi-sensor applications for ADAS will also be analyzed.

1.3.1 Sliding Window Detectors

One of the first sliding window visual object detector attempted to describe an object class in terms of an over-complete dictionary of local, oriented multi-scale intensity differences between adjacent regions; they are known as Haar Wavelets, and are applied to an example-based machine learning approach, where a model of an object class is derived implicitly from a training set of negative and positive examples (Papageorgiou et al., 2000 [2]). The specific learning engine used is a Support Vector Machine (Cortes and Vapnik, 1995 [3]) classifier, and results for car, faces and people detection tasks are shown. Before this work, visual human detection had not yet been successfully tackled, as they would typically assume a number of restrictive assumptions in order to produce results.

Building upon Papageorgiou's ideas, (Viola and Jones, 2001 [25]) (VJ) proposed a method that extracted Haar-like features with an highly optimized approach due to the use of integral images, which is an image transformation that allows for rectangular sums of pixels to be computed by fast arithmetic operations. In addition to this, a learning mechanism based on the AdaBoost algorithm (Freund and Schapire, 1999 [23]) was utilized in order to select the most relevant features to perform classification, and a decision structure in the form of a cascade was built for efficient decision-making. This cascade works by evaluating sets of features that grow in complexity as a sample advances in the structure, an idea that stands upon the notion that a positive instance in an image is an extremely rare event. By rejecting most negative samples in the earliest stages of the cascade, this method, applied to face detection, was able to run at 15 frames per second (FPS) with a high success detection rate. Figure 1.3 shows a schematic representation of the detection cascade.

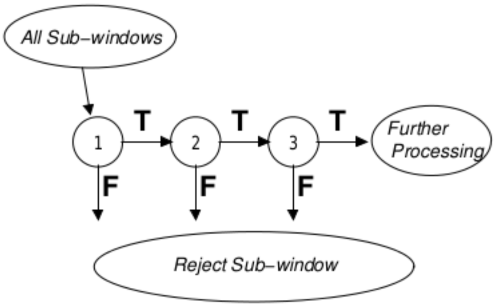

Figure 1.3: Schematic depiction of a detection cascade (Viola and Jones, 2001 [25]).

VJ's work is a popular and widely spread approach that still serves as foundation for many modern detectors, and full implementations of the method were made available in software development tools such as OpenCV and MatLab. In figure 1.4 some example results of the VJ algorithm are shown.

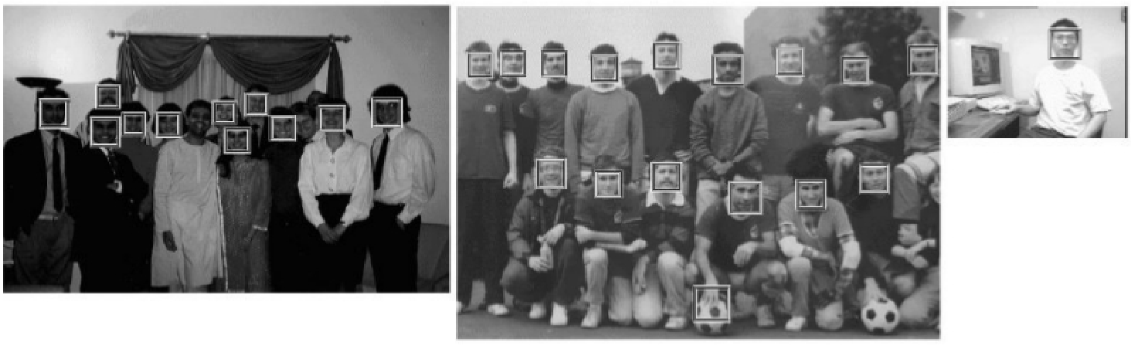

Figure 1.4: Viola and Jones's face detection algorithm (Viola and Jones, 2001 [25]).

Up until this moment, detection algorithms worked mostly on intensity images, a principle that changed when gradient-based features were introduced in the scope of pedestrian detection. Becoming widely known as Histogram of Oriented Gradient (HOG) (Dalal and Triggs, 2005 [4]), this method attempted to define a scene by dividing it into small spacial regions (cells), and accumulating for each one a local histogram of normalized gradient directions. These cells are combined over slightly larger and overlapping spacial regions (blocks), and each block is also locally normalized for better invariance to lighting conditions. The above-mentioned descriptors are then applied to a trained SVM based window classifier that identifies if a pedestrian is present in the scene or not. Figure 1.5 presents an overview of the HOG algorithm. This contribution resulted in large gains when compared to intensity based methods and, since its introduction, the number of variants of HOG has increased to the point that nearly all modern detectors use some form of these features.

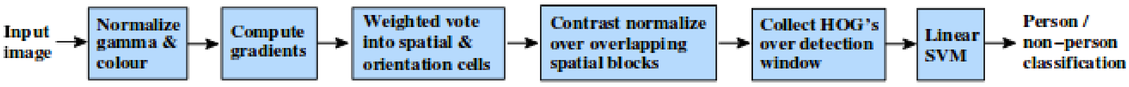
Input image
Normalize gamma & colour
Compute gradients
Weighted vote into spatial & orientation cells
Contrast normalize over overlapping spatial blocks
Collect HOG's over detection window
Linear SVM
Person / non−person classification

Figure 1.5: Overview of HOG detection algorithm (Dalal and Triggs, 2005 [4]).

Interpretation of shape is also an important cue to the subject. In general terms, shape-based methods work by generating templates of the desired object and finding matches for them in visual data. The work developed by (Gravila and Philomin, 1999 [13]) was one of the first to adopt this approach in the domain of pedestrian detection. It uses the Hausdorff distance transform and a template hierarchy to rapidly match image edges to a set of shape templates, and tests were made for pedestrian and traffic sign detection with satisfactory results, as shown in figure 1.6.

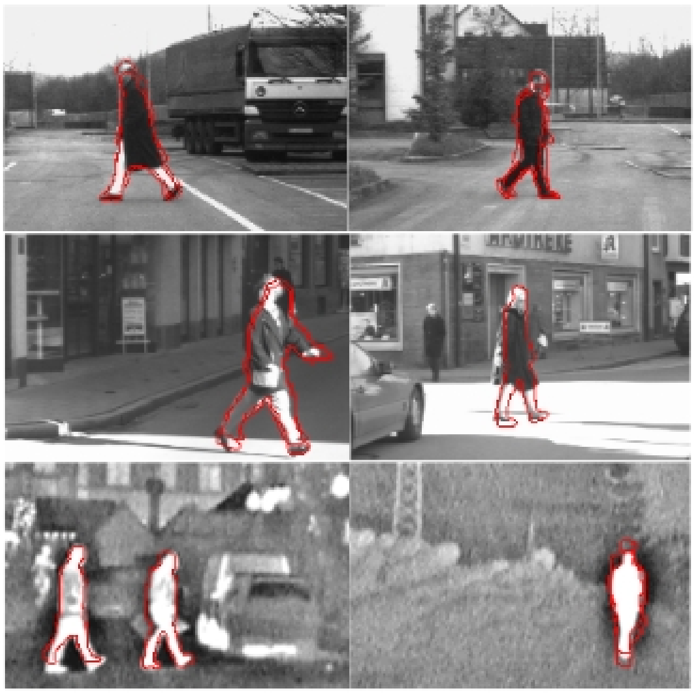

Figure 1.6: Example results of the shape-based pedestrian detection method (Gravila and Philomin, 1999) [13].

Still on this note, (Sabzmeydani and Mori, 2007 [22]) used gradient-based HOG-like features combined with an AdaBoost engine to learn head, torso, legs and full body shapes. In this approach two kinds of features are used for classification: the low-level features, which are simple and reminiscent to Haar-like features, and mid-level features

that are learned part models for template matching. This method is documented to outperform HOG by a considerable margin.
Some researchers have used motion features to further improve detection results. The basic idea is that in an usual situation people are in motion, rather then sitting still. Therefore it is natural to think that if the circumstances are such that detection of motion is achievable, important clues as to the possibility of the presence of pedestrians will be found. It is, however, a challenging task to incorporate motion features into detectors given a moving camera. Given a static camera, (Viola et al., 2005 [26]) proposed a similar approach to their previous work, but applied to the result of the difference of two sequential frames, resulting in large performance gains. For non-static imaging setups, camera motion has to be factored out, as did (Dalal et al., 2006 [5]) when they attempted to model motion statistics based on an optical flow's (Fleet and Weiss, 2006 [12]) internal differences, thereby compensating for uniform motion locally.
Although HOG has not been outperformed by any single feature, some researchers hypothesized that assembling multiple types of features could provide important complementary information. To prove this, (Wojek and Schiele, 2008 [28]) combined Haar-like features, shapelets, shape context, and HOG features to compare the resultant detector with each of the features performing on their own, demonstrating that the combo outperforms any single feature detector, as shown in figure 1.7. This framework was later extended to include the above-mentioned motion features in (Walk et al, 2010 [27]), further improving the detection results.

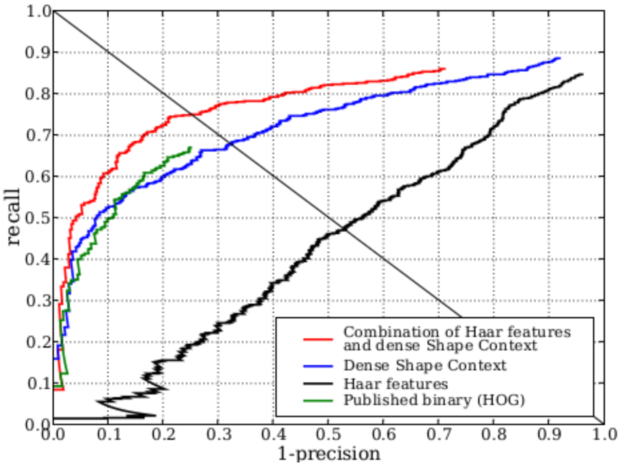

Figure 1.7: Multi-feature detector performance (Wojek and Schiele, 2008 [28]).

Using a different course of action, (Dollár et al., 2009 [9]) extracted Haar-like features from various channels, including gradient magnitude, LUV color channels and gradient magnitude quantized by orientation, providing a simple framework for integrating multiple feature types. Examples of possible channels are shown in image 1.8. In the author's approach, a large pool of features is extracted from random regions of the channels to guarantee a good characterization of the scene. A decision structure similar to the VJ's method is utilized with the purpose of selecting the most relevant features and performing efficient classification. This method became known as *Integral Channel Features*. A significant optimization was made to this algorithm when the authors hypothesized that features could be approximated at nearby scales with little sacrifice to results (Dollár et al., 2010 [8]). By eliminating the need to extract features at every scale, this algorithm is documented to perform multi-scale detection at 6 FPS and ranks among the best found in literature. A still better version of this framework was introduced in (Dollár et al., 2012 [7]), in which an even more efficient decision structure was proposed. In this brand new *Crosstalk Cascades* method, it is established that nearby decision windows have correlated responses. By creating a mean of communication between detector's responses, this method achieves similar detection rates as the *Integral Channel Features* while increasing speed by an order of magnitude.

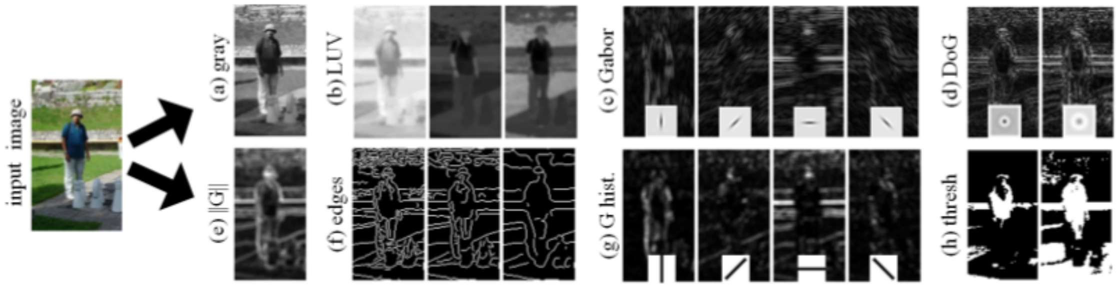

Figure 1.8: Examples of channels computed from an image (Dollár et al., 2009 [9]).

Some authors have made an effort to modify learning engines to improve their speed, as did (Tuzel et al., 2008 [24]) by utilizing covariance matrices computed locally over a large diversity of features as object descriptors. The learning data doesn't lie on the vector space, thus improving the learning/testing performance. Non-linear SVM applied to sliding window detection was made possible when (Maji et al., 2008 [16]) found that the use of the learning algorithm with an approximation to the histogram intersection kernel lead to substantial gains in terms of speed.

1.3.2 Multi-sensor Detectors

Visual data has great potential due to the richness of information it holds. However, it may be a challenging task to build a robust detector to be implemented in an ADAS relying solely on camera output as a result of the problems discussed in the previous section. To overcome those difficulties, some try to ally different type of sensor information to create robust and more reliable detectors.

Infra-red image and laser data were used by (Fardi et al. 2005 [11]) to generate regions of interest where pedestrians might be at sight. A first-step classification is obtained by evaluating a set of descriptors based on the Euclidean distance of Fourier between objects and reference sets on infra-red visual data, a classification that is later refined by motion features acquired using egomotion sensors and optical flow.

On a different approach, radar, color and infra-red information is fused in (Milch et al., 2005 [17]). In this work, hypothesis are generated through the evaluation of vision-based local histograms on edges, computed both on color and infra-red visual data. Neural Networks is the learning engine used for a preliminary classification, and its output undergoes further verification by a fusion between radar and tracking information.

Combining infra-red and visual spectra from the two camera types was proposed by (Bertozzi et al., 2006-2007 [1]). In the author's work, foreground segmentation is carried out by overlapping 2D and 3D information from both sensors and, finally, symmetry and template matching are used to classify, verify and refine final detections.

Laserscanner-based tracking of points was the strategy chosen by (Premebida et al. 2007 [20]) to generate candidate regions of interest for further analysis. Objects were defined by laser and visual features, and AdaBoost was utilized for generating responses.

These systems were built in scientific research environments where information is usually made accessible for anyone. That is not the case in industrial environments where, for commercial reasons, conducted research is kept in absolute secret, a circumstance that makes it hard to find reliable sources about the state of this technology in industry. It seems granted, however, that the on of first pedestrian detection system to be commercialized will be launched in 2014 by Mercedes and will be based on stereo camera images (Mercedes press information, 2013 [15]). A study made by Mercedes shows that their safety mechanism based on pedestrian detection could avoid 6% of pedestrian accidents and reduce the severity of a further 41%. Volvo was the first, and only to this date, to launch a real pedestrian detection application (Volvo Mobileye Pedestrian Collision Warning, 2012 [18]) to the market. Their mechanism uses monocular image and radar to perform detection, and issues a warning to the driver when there is risk of collision, giving him over two seconds to avoid it.

1.4 Proposed Solution

Although great advantages arise from fusing different type of sensor information, a multi-sensor approach also has its issues, such as difficulties in fusing and correlating different sensor data, higher registration demands, more complex system implementations, accumulation of errors generated from different sensors, and others. Despite these problems, it is obvious that any real and full-functional pedestrian detector will, in all likelihood, require the use of multiple sensors as a result of the extremely demanding nature of the task in hands.

However, creating such complex application is a large engineering effort that requires a great deal of know-how, expensive equipment and time, especially when the application is being built from scratch. Although much useful equipment already exists in the laboratory, as well as a staff with a comprehensive knowledge and set of skills, building a full-functional, multi-sensory system cannot the objective due to the short amount of time available to complete this work. The goal was to build a reliable, generic and simple vision-based pedestrian detection framework that leaves the possibility for future development and integration in more complex systems. It was decided then that the implementation of a sliding window algorithm was the way to go, since one can be modified to work with other sensors in future development.

In order to define a course of action, two premises were established: the implemented algorithm should rank among the best in terms of detection performance and should also leave space for future improvements. In respect of this, (Dollár et al, 2012 [10]) made an extensive survey of existing sliding window detectors, where 16 algorithms were compared against each other in a carefully designed evaluation platform. Out of all the evaluated algorithms, *Integral Channel Features* (*ChnFtr*) proved to be the most interesting for several reasons. Firstly, the only method that slightly outperforms it uses motion and gradient-based features, a computationally heavy approach that is documented to run

50 times slower than *ChnFtr*, rendering it uninteresting. On the contrary, the authors of *ChnFtr* have largely improved its performance in later work, to the point of enabling multi-scale detection at 30 FPS [8] [7]. Secondly, the method provides a relatively simple framework in terms of code implementation when compared to other approaches. Since this implementation fitted perfectly with the proposed goal, it was the chosen way to go for this project.
A detailed explanation of *ChnFtr* is presented in chapter 2.

1.5 Development Tools

It has become clear that this was mainly a software development project, and, as one would expect, most work involved designing, implementing and testing code. The programming language used was C++ under Linux platform, and, in addition to this, a set of indispensable development tools were utilized, all of which will be enumerated and described in this section.

1.5.1 Robotic Operation System

The Robot Operating System (ROS) (Quigley et al., 2009 [21]) provides a software development framework that is designed for the creation of robot software. This application has several built-in components prepared to handle the output of different types of sensors, such as cameras, lasers, actuators, contacts and other common elements in a robotic environment.
ROS also allows for an easy way to establish communication between different software modules (*nodes*), which permits the elaboration of an infrastructure that can communicate with any running process. This communication works in three steps: first, a *node* advertises a *ROS Topic*. Once that topic of communication is advertised, the same *node* is able to publish messages on that topic, and finally, those messages can be listened by any *node* that subscribes to the same topic. Such messages can be of any kind, from simple strings of characters to visual and laser data. Figure 1.9 tries to illustrate a very simple ROS communication architecture. It is easy to understand that this information exchanging structure has a great potential when applied to robotics, since an uniform and standardized communication between sensors facilitates the development of complex multi-sensor applications.

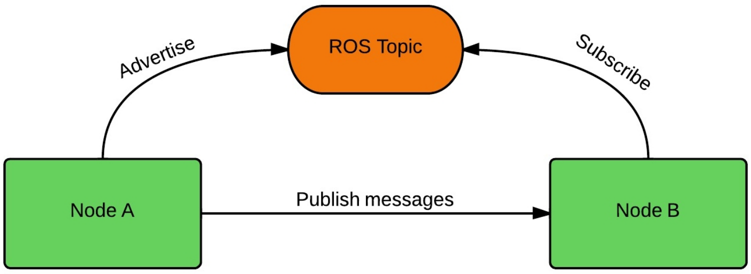

Figure 1.9: Schematic of a simple ROS communication architecture.

Another important feature that ROS provides is the possibility to record data logs

(*rosbags*) that can be replayed later. This allows for data to be collected in real scenery, to be later treated in a laboratory environment as if it was on the field.
Recently, to standardize the work developed for the *Atlas* project, an effort has been made to migrate every application to ROS environment and, for this reason, the work presented in this document was also developed in ROS.

1.5.2 OpenCV

OpenCV is a popular open source computer vision library written in C and C++ that was designed for high computational efficiency and with a strong focus on real-time image processing applications. It provides an infrastructure to help people build fairly sophisticated vision applications, and contains hundreds of functions that span many areas of application, like factory product inspection, security, medical imaging, robotics and many others. This is a well documented, very complete library with a huge and collaborative user community that is in constant growth. Such a healthy and knowledge-sharing environment is a positive aspect that largely contributed for the development of the project.
In this work many useful OpenCV functions are used for several purposes. Resizing images, computing gradients, converting images between colors spaces or selecting sub-windows from images are just a few examples of operations provided by OpenCV that are absolutely necessary for most visual pedestrian detection applications.

1.5.3 INRIA Dataset

In order to develop detection applications of objects in complex scenes, where segmentation and other similar approaches are not an option, the use of a learning machine algorithm is absolutely necessary. In a nutshell, these algorithms work by exhaustive learning of positive and negative instances of the problem in question, and once the learning process is finished, they are able to predict on new unseen data. So, in order to develop a detection application, the developer must possess a set of positive examples of the object to be detected, and, to ensure that the learning engine is correctly trained, the number of positive examples usually needs to be large. Acquiring hundreds, sometimes thousands, of positive instances of an object is obviously a slow and time-consuming task, even more when the data needs to be treated and labelled in laboratory. Fortunately, a handful of pedestrian datasets were made public by the scientific community, and anyone is free to use them.
The INRIA dataset was acquired by (Dalal and Triggs, 2005 [4]) with the objective of setting a challenging framework to test the HOG algorithm. Since then, this dataset has been used by most pedestrian detection researchers, as did the authors of *ChnFtr*.
This dataset provides a training set with 1218 images where no pedestrians appear (negative images), and 2416 positive training windows, meaning, pedestrian images cropped from the original scene in which they appear. The fact that the positive examples are already cropped out of the original images largely facilitated the training process. The dataset also provides a different set of images, with 1132 positive windows and 462 negative images to test the detector performance.
This dataset was of utmost importance for the development of this work, as it not only allowed for a facilitated training/testing process, but also for the implementation of a meaningful evaluation platform to compare results between the original algorithm and

the ones achieved in this work. Image 1.10 shows a set of example positive windows from the INRIA dataset.

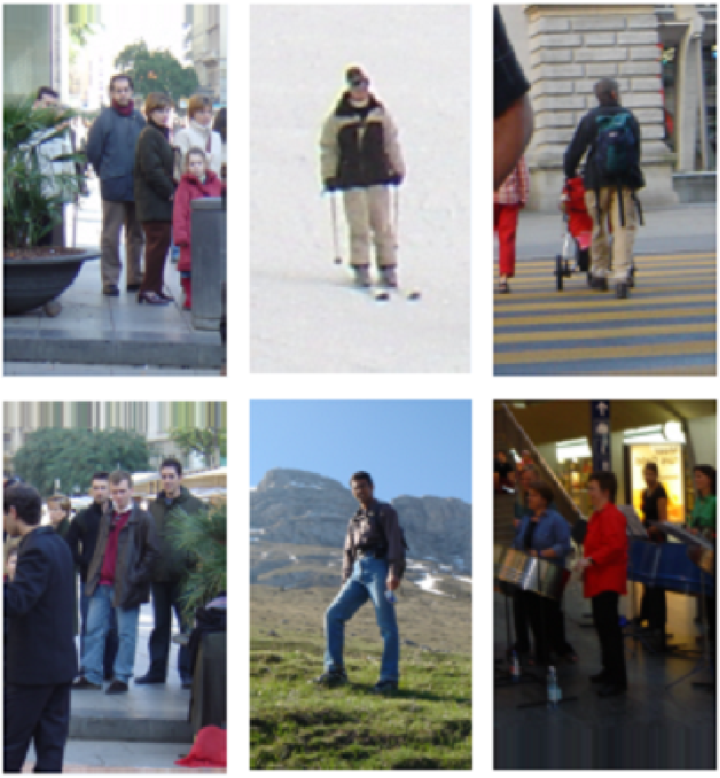

Figure 1.10: Examples of positive windows used for training a classifier. In this dataset people appear on a wide variety of backgrounds.

1.6 Experimental Setup

The tools described on the previous section integrate an experimental setup that was the base for all the development. A schematic representation of the experimental setup is shown in figure 1.11.

Having in mind that the end goal of this project was to build an application to run on the *Atlas Car*, it was an important pre-requisite for the setup to be generic enough to allow for full development in laboratory environment, and also be easily set to run on the field.

To fulfill that important pre-requisite, two *nodes* were created. The image server is responsible for advertising an *image transport* topic, loading images from file and publishing those images on the topic. The image client subscribes to that topic and, the event of an image being sent over triggers a callback function that processes and analyses the image using OpenCV.

To test the application on another setting, be it with camera output or *rosbag* replay, one just needs to set the image client to subscribe to the topic in which those instances are publishing.

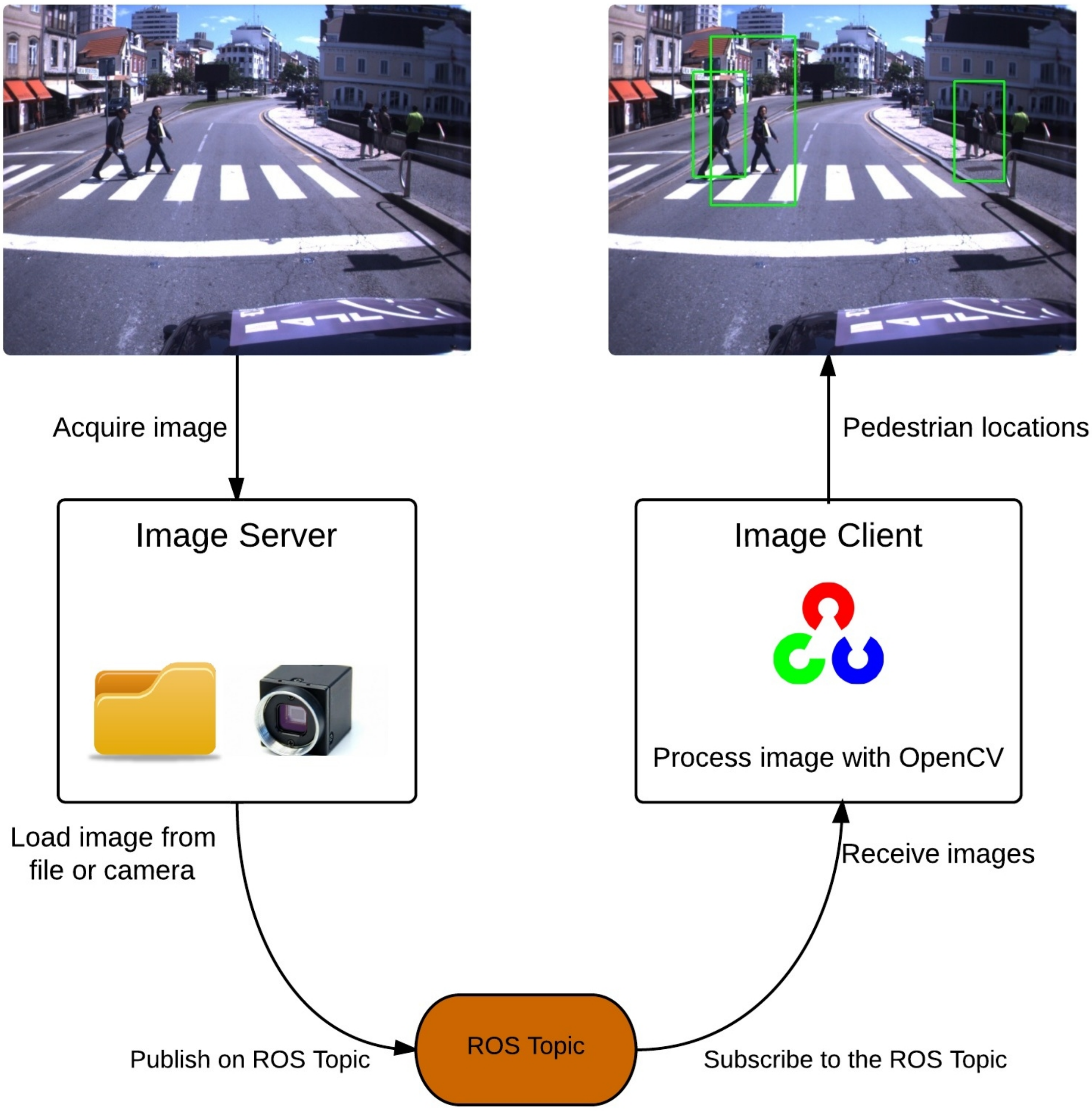

Figure 1.11: Schematic representation of the experimental setup.

Chapter 2

Integral Channel Features

To solve a complex vision detection problem it is necessary to perform exhaustive descriptions of the objects in order to find a set of descriptors that present significant variability in their presence. The proposed solution takes advantage of the richness of information present in multiple channels of an image, using it to assemble different detection techniques in a simple framework. By evaluating a very large number of simple features from multiple channels, this algorithm integrates normal, HOG-like, shape-based and other feature types in an indirect manner and, not only that, those features can be extracted in an optimized manner with the use of the integral images. These are the reasons as to why *Integral Channel Features* (*ChnFtr*) has so much potential, since it allies multi-feature analysis and speed in a relatively simple infrastructure. *ChnFtr* is divided into three main parts:

1. Computation of channels
2. Feature extraction
3. Classification

In this chapter items 1 and 2 will be discussed, leaving the topic of classification for the next chapter.

2.1 Channels

In the context of this work, a channel of an image is a representation of the original, where the output pixels are obtained by using linear or non-linear transformations on the input ones, thus preserving the overall image layout. A trivial channel is the grayscale representation of an image, and, likewise, the different color channels of an image can also serve as channels in this context.

There are countless transformations that can be applied to an image that will result in new channels; application of filters, binary thresholds and edge detectors are just a few examples of simple ways to obtain numerous channels. However, our human perception only goes so far when it comes to realize which channels will contribute with more relevant information for detection. The only way to complete such a task is to test various channels for performance and check which are more informative by evaluating results.

Fortunately, that information is already available in the original publication, where the detection performance of different channels were put against each other. As shown in figure 2.1, the conclusions that were drawn from that study were that the channels that lead to the best results are the gradient magnitude, gradient histogram (labelled *Hist* in figure 2.1) and the LUV color channels. For this obvious reason, these channels were the ones used, and will be described in the following sub-sections.

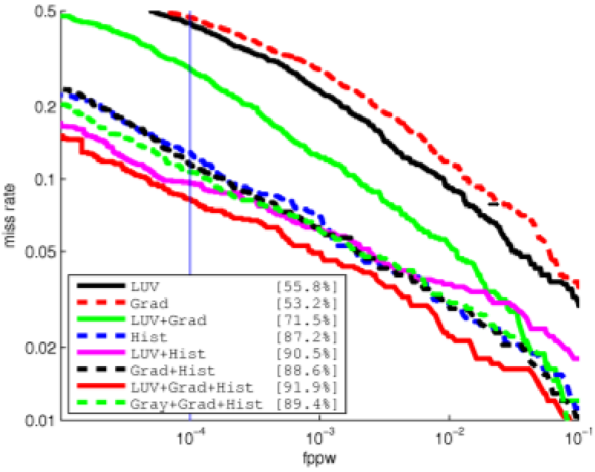

Figure 2.1: Detection performance of different channel combinations (Dollár et al., 2009 [9]).

2.1.1 Gradient Magnitude

Given an input image *I*, the gradient magnitude (GM) of I is a representation of the strength of its edges. The channel is computed as shown in equation 2.1, in which $\frac{\delta I}{\delta x}$ and $\frac{\delta I}{\delta y}$ are respectively the horizontal and vertical gradients of *I*.

$$GM(x,y) = \sqrt{\left(\frac{\delta I}{\delta x}(x,y)\right)^2 + \left(\frac{\delta I}{\delta y}(x,y)\right)^2} \tag{2.1}$$

In terms of code implementation, the horizontal and vertical gradients are computed by applying a simple *Sobel* operator. The gradients are then squared, summed and finally square rooted for the final result. Figure 2.2 describes how this is processed in the code.

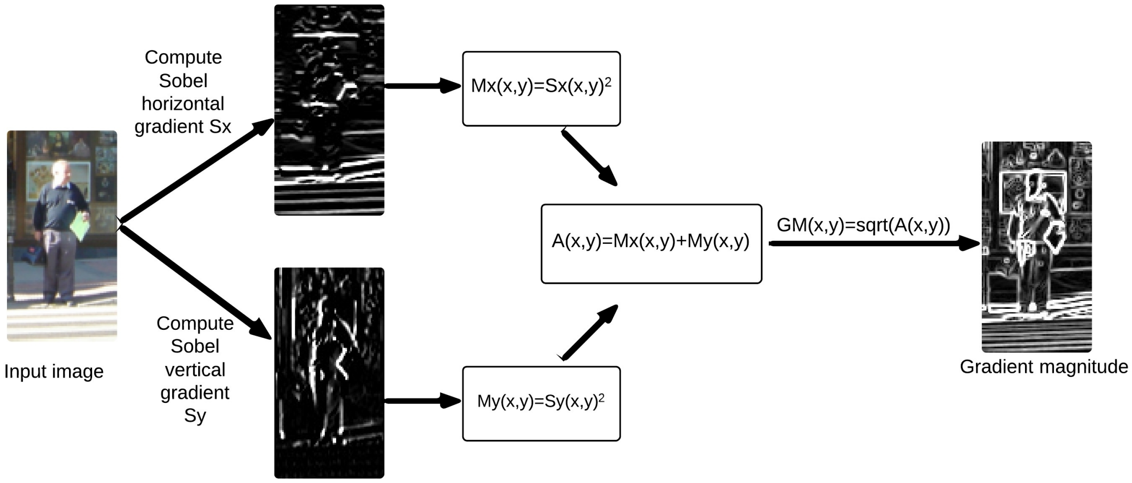

Figure 2.2: Gradient magnitude computation.

2.1.2 Gradient Histogram

A gradient histogram is a weighted histogram where bin index is determined by gradient angle and weighted by gradient magnitude, providing important information about edge strength under different pixel orientations. The only parameter for computing gradient histogram is the number of bin orientations, knowing that each bin generates a separate channel. This parameter is set to 6, which is the number from which the results stop improving, as shown in picture 2.3.

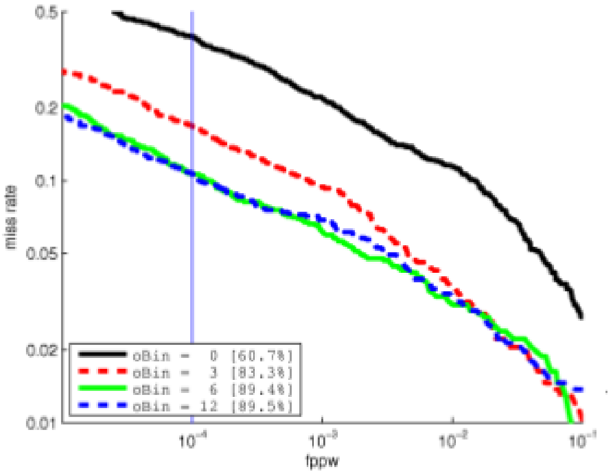

Figure 2.3: Detection performance of different numbers of bin orientations (Dollár et al., 2009 [9]).

The orientation at each pixel is calculated according to equation 2.2.

$$\theta(x,y) = arctan\left(\frac{\frac{\delta I}{\delta y}(x,y)}{\frac{\delta I}{\delta x}(x,y)}\right) \tag{2.2}$$

Concerning the code implementation, the horizontal and vertical gradients obtained before are used to compute quotients between the gradients, which are stored in a new data structure. Then, the orientation for each pixel is calculated and a heuristic structure is built to accumulate the previously computed gradient magnitude values on different bins depending on the value of θ. A schematic description of how gradient histogram bins are processed in the code is shown on figure 2.4.

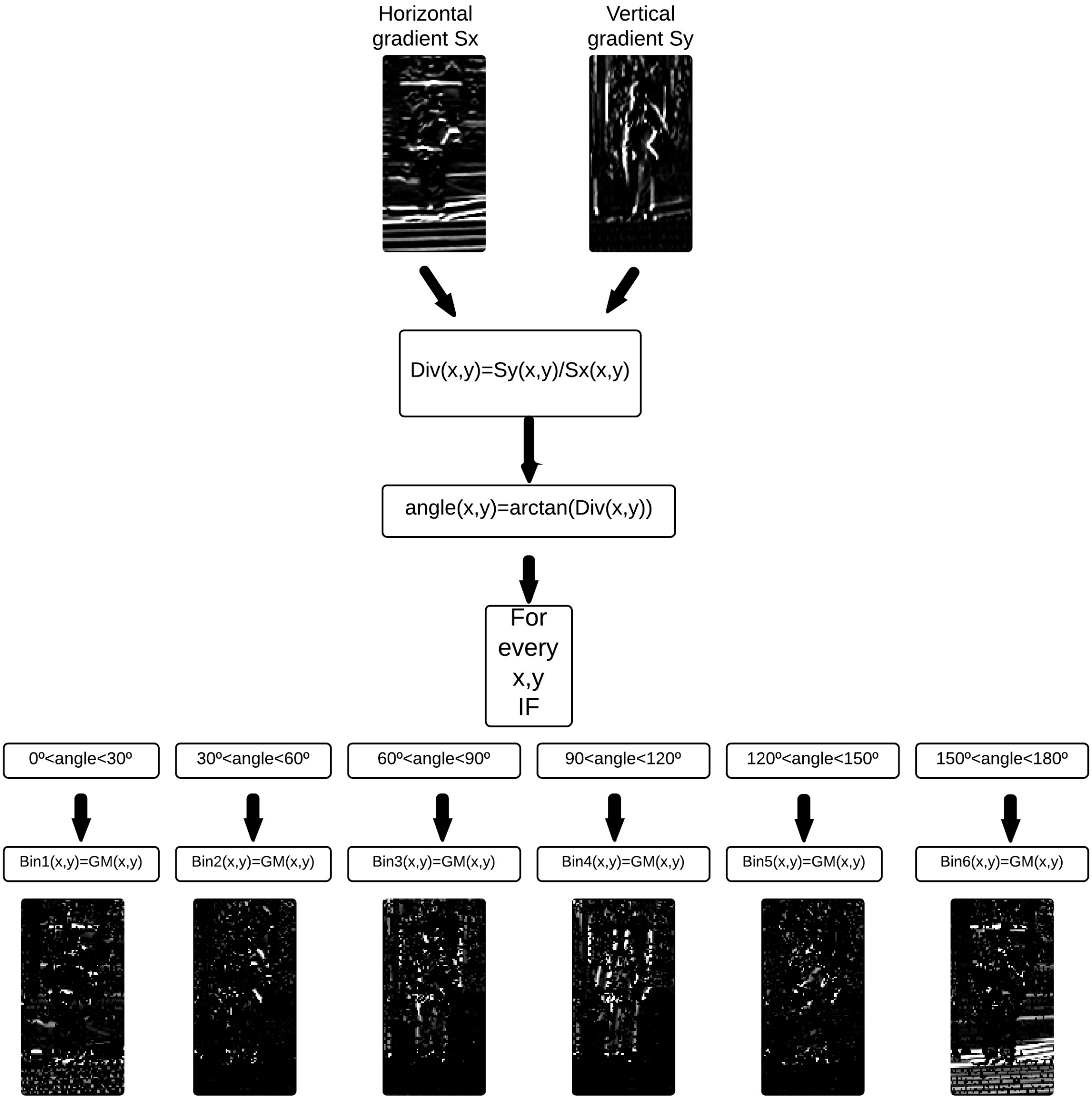

Figure 2.4: Gradient histogram computation.

2.1.3 LUV color channels

Color is an important cue for detection. In respect of this, different color spaces were compared against in order to find out which is more informative; this a comparison lead to the conclusion that the LUV color channels are the ones that provide the best results, as show in figure 2.5.

Obtaining these channels is done with few lines of code, since OpenCV provides a straightforward way to convert images between color spaces. Figure 2.6 illustrates how the channels are computed code-wise.

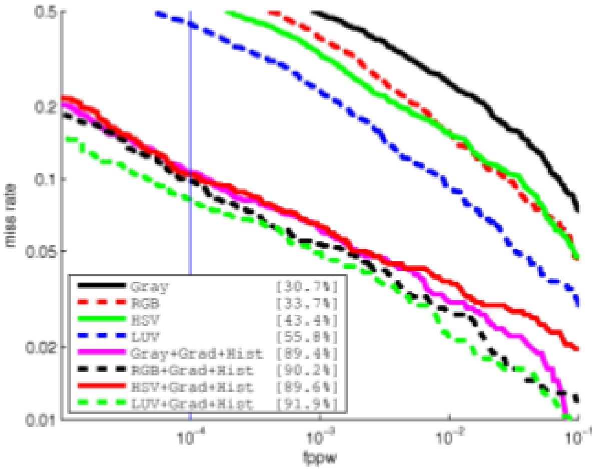

Figure 2.5: Detection performance of different color channels (Dollár et al., 2009 [9]).

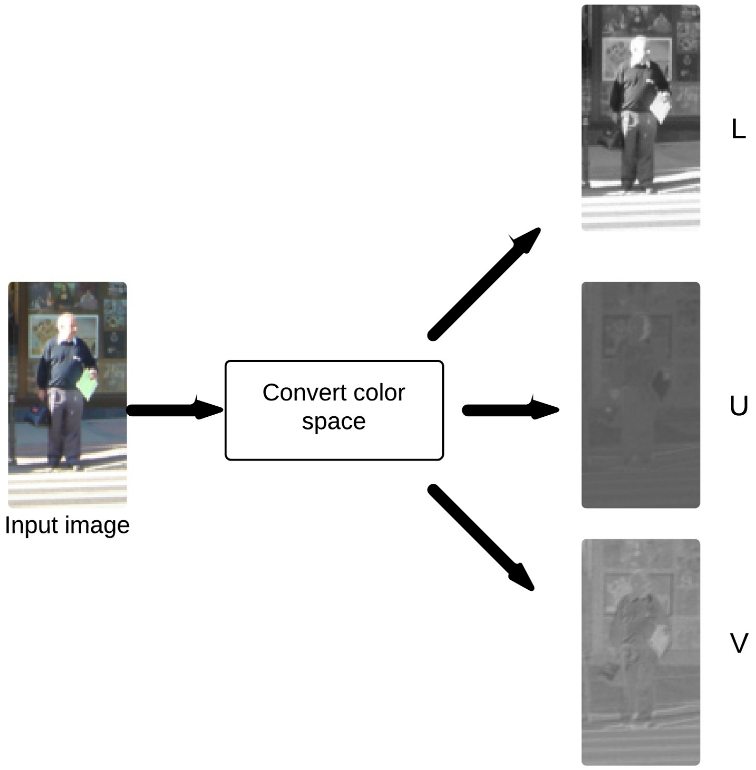

Figure 2.6: LUV color channels computation.

2.2 Integral Images

There is a middle step between computation of channels and feature extraction. Since a scene description is done through the computation of great amounts of rectangular sums of pixels, it is convenient to use a optimized way for computing local sums. This is done with the use of the integral image, and is a key point of the algorithm.
The integral image is an image transformation that allows efficient generation of sums of pixel values in a rectangular subset of an image. Given an input image *I*, the value at any point (x,y) in the integral image *Int* is just the sum of all pixels above and to the left of (x,y), as illustrated in equation 2.3.

$$Int(x,y) = \sum_{\substack{x' \leq x \\ y' \leq y}} I(x',y') \tag{2.3}$$

Then, the sum of any rectangular region of the image can be calculated by a simple arithmetic operation, as illustrated in figure 2.7.

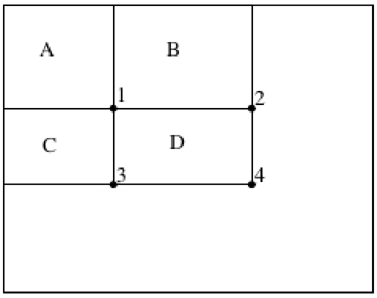

Figure 2.7: Finding the sum of a rectangular region (Viola and Jones, 2001 [25]). In this example, the value of the summed pixels in area *D* is *Int(4)+Int(1)-Int(2)-Int(3).*

OpenCV already provides a way for computing integral images automatically, making it unnecessary to implement this operator by hand. The 10 channels (1 gradient magnitude, 6 bins of orientation and 3 color channels) are transformed into integral images, and the resulting transformations are referred as *Integral Channels.*

2.3 Feature extraction

Sliding window detection is performed by applying a Detection Window (DW) over the image, evaluating a set of features, and then sliding it to an adjacent place to repeat the process. The DW has constant dimensions (64x128 as in most sliding window detection methods), so, in order to find pedestrians with different sizes the image has to be rescaled and re-analysed multiple times. This compact description of a generic multi-scale sliding window detector shows that defining how a DW is analysed is a major key point for detection. This section will focus on feature extraction from DWs.
There are distinct approaches for describing a DW. Some researchers opt to generate a fine tuned pool of features that are subject to tests until they achieve good results. This is the case of the HOG algorithm, where features are made of local sums calculated over a dense overlapping grid in the DW. On the contrary, rather then carefully designing

a feature space, *ChnFtrs* generates random features from the channels, knowing that a good characterization of the DW is most likely granted if the feature pool is large enough. In the context of this work, a feature is no more than a rectangular sum of pixels, and each feature has the following random parameters: the channel where it is calculated, the starting position of the rectangle and its dimensions. To make this more clear, an illustration of 20 possible random features is shown in figure 2.8. One is inclined to think that thousands of random features are likely to lead to a strong and robust characterization of a DW.

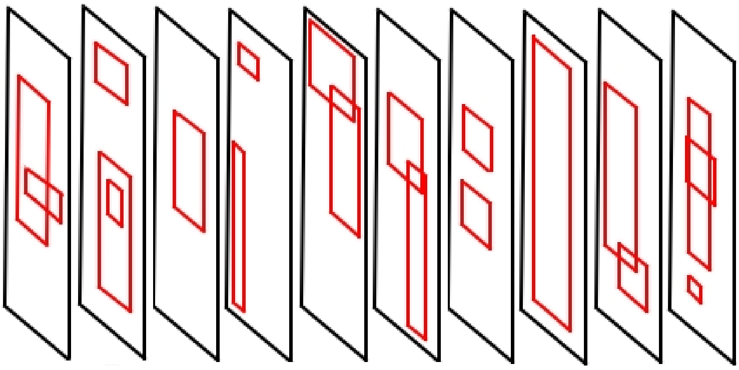

Figure 2.8: Examples of random features: in this illustration, the black rectangles represent the DW over the 10 channels, and the red rectangles represent examples of random rectangles over which local sums are calculated.

To extract any number of random features, it is necessary to generate their parameters first. In terms of code implementation, a vector data structure was created in which each element has information about the 5 random parameters necessary to define a rectangle:

- Channel index
- Width
- Height
- X coordinate of the upper-left corner of the rectangle
- Y coordinate of the upper-left corner of the rectangle

Then, a function to initialize the vector with N random parameters was created. Keeping in mind that even if the parameters are random, it is vital that the program is able to reproduce the same parameter vector any number of times. To achieve this, OpenCV provides an *RNG* (Random Number Generator) class that is initialized with a seed. If the seed is the same, the random parameters generated will also be the same.
For each element of the vector, the following random parameters are generated:

- Random(0,9) - channel index
- Random(5,DW width) - rectangle width
- Random(5,DW height) - rectangle height

- Random(0,DW width-rectangle width) - X coordinate
- Random(0,DW height-rectangle height) - Y coordinate

This process occurs once when the program is initiated, and the resulting parameter vector's elements are accessed when a DW is processed.
The integral channels and feature parameter information is all that is necessary to compute features over a DW. In what concerns the code implementation, for every random parameter, a rectangle is defined and the local sum is computed and stored in a feature vector. The resulting vector characterizes the DW feature-wise.

2.4 Multi-scale Image Analysis

Once feature extraction from DWs is set, the next step is to build an architecture for full image analysis. As it was mentioned before, it is essential to analyse not only the entire image, but the same image at multiple scales to find pedestrians with multiple sizes. It is then necessary to build a dense image pyramid (figure 2.9) for analysis.

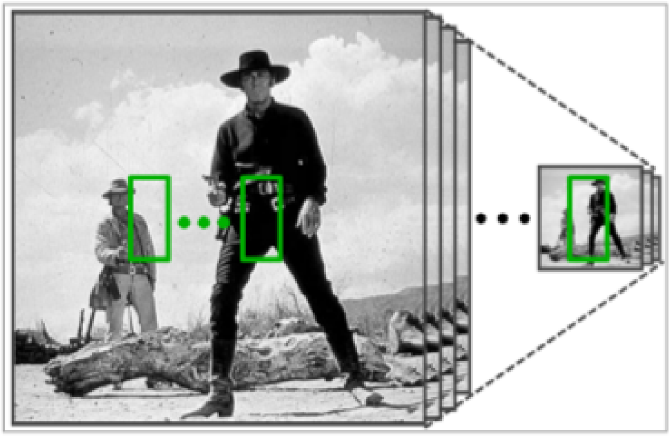

Figure 2.9: Dense image pyramid (Dollár et al., 2010 [8]). The green rectangle represents the DW of constant dimensions.

There are a few parameters for building this pyramid, all of which are easily changeable.
At each step the image is rescaled depending on the number of scales per octave (*nPerOct*), and its default value is set to 8. An octave is the necessary scaling to rescale the image by a factor of 2. So, each downsampled image is rescaled according to equation 2.4, and for upsampling the logic is the same (equation 2.5).

$$DownScale = 2^{\frac{-1}{nPerOct}} \tag{2.4}$$

$$UpScale = 2^{\frac{1}{nPerOct}} \tag{2.5}$$

For 640x480 images, upsampling becomes a computationally heavy operation that is performed when detection of far scale pedestrians is a requisite, so, it is turned off by default. Downsampling goes on until the image reaches a parametrizable minimum size, which default value is set to the same of the DW (64x128).

All that remains is to slide a DW through all the images in the pyramid and store a feature vector for every DW analysed. The default value for the step by which the DW slides through an image is set to 4 (in both directions), which is the step used in the original implementation.

With the default parameters described in this section, each 640x480 image has ~60000 DWs and takes ~0.8 seconds to be processed if 1000 features are extracted per DW, in an Intel core i7 processor. The processing time grows fairly linearly with the number of features computed, taking ~1.6 seconds for 2000 features and so on.

Regarding the code implementation, a cyclic architecture described by figure 2.10 was built.

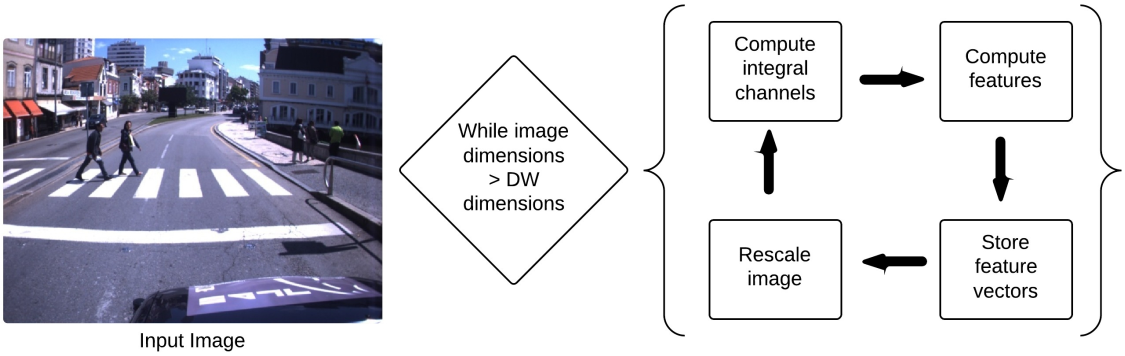
Input Image
While image dimensions > DW dimensions
Compute integral channels
Compute features
Store feature vectors
Rescale image

Figure 2.10: Multi-scale image processing.

2.5 Parametrization Summary

For an easy overview of the work so far, this section provides a table that summarizes the parametrization of the algorithm.

Integral Channel Features	
Channel types	Gradient magnitude Gradient histogram LUV color
DW size (w x h)	64 x 128
min. feature size (w x h)	5 x 5
max. feature size (w x h)	64 x 128
Number of scales per octave	8
max. scaled image size (w x h)	640 x 480
min. scaled image size (w x h)	64 x 128
Sliding DW step	4

Table 2.1: Default parametrization summary

These parameters are default values that are defined in a header file. The program is prepared to run cleanly for any parametrization. So, for running the application with different parameters, one just needs to change the header file definitions.

Chapter 3

Classification

Searching for an object in a scene demands for a method capable of evaluating the feature vectors that describe it. This can be done through the implementation of a Machine Learning (ML) mechanism. The purpose of ML is to turn data into information, a process that becomes fundamental when the information has to be inferred from large amounts of data, much like the case of the problem in hands. Typically, these methods attempt to model an object class from a training set of examples to then make predictions on new and unseen data, or, in other words, perform classification.
There are multiple ML methods, each with its specifications and applications, so, one must choose a method that correctly fits the problem. The adopted method needs to classify between two classes, *Pedestrian* and *Not Pedestrian*, needs to handle thousands of features per sample and should also be fairly resistant to over-fitting. The method that best fits these requirements is AdaBoost, and a compact description will be carried out in the following section. The basic idea behind AdaBoost, short for Adaptive Boosting, is that it is possible to generate a very accurate prediction rule, or *strong classifier*, through the aggregation of rough and moderately inaccurate linearc rules, *weak classifiers*, provided that they perform just slightly better than a random classifier would. A graphic illustration of this is shown in figure 3.1.

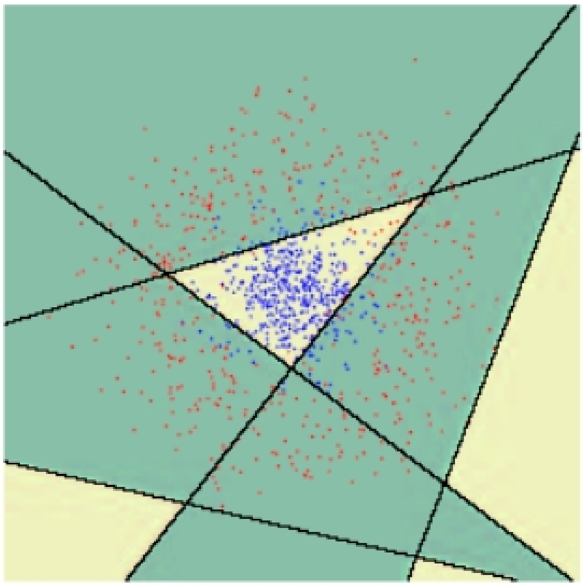

Figure 3.1: Aggregation of *weak classifiers*: in this example, each black line represents a *weak classifier* that attempts to distinguish red from blue dots. It is easy to understand that each *weak classifier* doesn't perform well on its own, whereas a combination of the 5 *weak classifiers* will correctly label most samples.

3.1 AdaBoost

This algorithm takes as input a training set *(x_1,y_1),..., (x_m,y_m)*, where x_i belongs to the domain X, and refers to a feature vector; y_i refers to the label of the corresponding sample, and belongs to *Y={-1, +1}*. AdaBoost calls a *weak learning* algorithm over and over again in a series of rounds *t= 1,...,T*. One of the key points of this method is to maintain a distribution set of weights over a training set, which are set uniformly in the first iteration. The weight of this distribution on a training example i on round t is denoted *D_t(i)*. At the starting point, all weights are set uniformly but, on each iteration, the weights of incorrectly classified examples are increased in an attempt to force the algorithm to give them a special attention, and this is why the method is called Adpative Boosting, since it adapts iteratively to focus on hard examples. The *weak learner's* function is to find a *weak classifier*: h_t*: X* $\longrightarrow$ *{-1,+1 }*, appropriate for the weight distribution D$_t$. A *weak classifier* is evaluated by its error, calculated according the equation 3.1.

$$\epsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i) \tag{3.1}$$

Once an hypothesis h$_t$ has been set, AdaBoost chooses a parameter α_t, which is a measure of the importance of the learned *weak classifier*, and is calculated according to the equation 3.2.

$$\alpha_t = \frac{1}{2} ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right) \tag{3.2}$$

Note that $\alpha_t > 0$ if ϵ_t<1/2, meaning, a *weak classifier* is only attributed with importance if it gets atleast half of the training examples right. It is also intuitive that a *weak classifier* is as more important as lower its error is. The distribution D$_t$ is then updated according to the rule illustrated by equation 3.3, in which the denominator is a normalization factor used to ensure that D$_{t+1}$ is a probability distribution.

$$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{\sum_i D_t(i)exp(-\alpha_t y_i h_t(x_i))} \tag{3.3}$$

This rule assures that the misclassified examples are attributed with more weight so that in the next iteration they can be resolved.

The final hypothesis H is a majority weighted vote of the T *weak classifiers*, as show in equation 3.4.

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{3.4}$$

Or equation 3.5 to obtain a measure of the confidence of the detector

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{3.5}$$

This whole process is summed up in algorithm 1.

```
Algorithm 1 The boosting algorithm AdaBoost
Given (x$_1$,y$_1$),..., (x$_m$,y$_m$) where x$_i$ $\in$ X, y$_i$ $\in$ Y = {-1,1 }
Initialize D$_1$(i)=$\frac{1}{m}$
For t=1,...,T:

    • Train weak learner using distribution D$_t$

    • Get weak hypothesis with error:

        $$\epsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$$

    • Choose $\alpha_t = \frac{1}{2}ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

    • Update:

        $$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{\sum_i D_t(i)exp(-\alpha_t y_i h_t(x_i))}$$

Output of the final hypotheses:

    $$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$
```

3.2 Training Process

OpenCV provides an implementation of AdaBoost, so there was no need to build one from scratch. The most relevant aspects of the training process of the final classifier are discussed in this section.

3.2.1 Training Samples

OpenCV's AdaBoost training method uses a matrix containing the training data as input, and outputs a classifier that can later be used to perform detection in new samples. The most convenient way to prepare the data is to write it to a text file and only later convert it into a matrix. So, simple changes had to be made to the infrastructure to enable the writing of feature vectors in file.

Selecting the training samples of the final classifier is done in three stages. The first stage classifier is trained with the 2416 positive training windows available on the INRIA dataset and ∼5000 negative windows generated randomly from a wide variety of negative images. The resulting classifier is then tested on the same negative images, and this time ∼5000 false positives are added to the training data as negative examples, forcing the classifier to learn the hard examples, a process that is known as *bootstrapping*. Finally, a final round of *bootstrapping* is performed where more ∼5000 hard negatives are added to the training data. In sum, The final classifier has 2416 positives examples and ∼15000 negative examples, ∼10000 of which are samples that were incorrectly classified as positive by the first 2 stage classifiers.

3.2.2 Classifier Parametrization

There are a few additional parameters important for training the classifier. The number of features to be computed per window is a key factor that has a major effect on the detector's performance. To understand how the size of the feature pool affects results, two classifiers were trained, one with 10000 features and another with 15000. Note that a classifier trained with a given number of features has to be fed with that same number of features in order to produce a response during test time.

The number of *weak classifiers* is also an important parameter. As explained before, each *weak classifier* will contribute with a weighted vote for each decision. This parameter is set to 2000, which means that 2000 *weak classifiers* are derived from the feature pool.

The final parameter is the depth of the classifier, which is set to 2. This parameters defines the number of features that constitute each *weak classifier*. Depth 1 uses one feature per *weak classifier* while depth 2 uses three features per *weak classifier*.

In practical terms, the OpenCV boost training method will go through the training data for 2000 iterations, and in each iteration will select the combination of 3 features that best classifies the training data. Table 3.1 gives a summary of the classifier's parameters.

Adaboost Classifier	
Positive class	Pedestrian
Negative class	Non pedestrian
Feature pool size	15000 10000
Positive samples	2416
Negative random samples	∼5000
Negative *bootstrapped* samples	∼10000
Weak classifiers	2000
Depth	2

Table 3.1: AdaBoost parameters

3.3 Post Processing

In full image analysis, it is necessary to keep track of the windows that are classified as positive in order to draw rectangles around detected pedestrians. Since feature extraction and classification are processes that occur on par, it was necessary to create a data structure to accumulate the image coordinates of the positive windows, as well as the scale at which they were obtained in order to transform them to the original scale. As one would expect, in a multi-scale detection platform an object is often detected multiple times, generating multiple rectangles, as shown in figure 3.2. It was then important to find a way to group rectangles that belong to the same object together. Fortunately, since this is a common detection problem, OpenCV provides a way to merge rectangles with similar sizes and locations, and the resulting output is illustrated by figure 3.3.

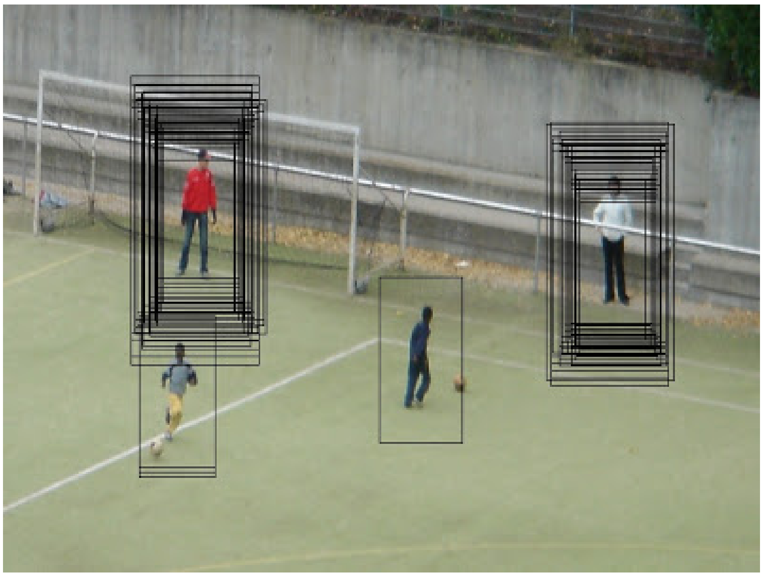

Figure 3.2: Rectangles generated by the detector.

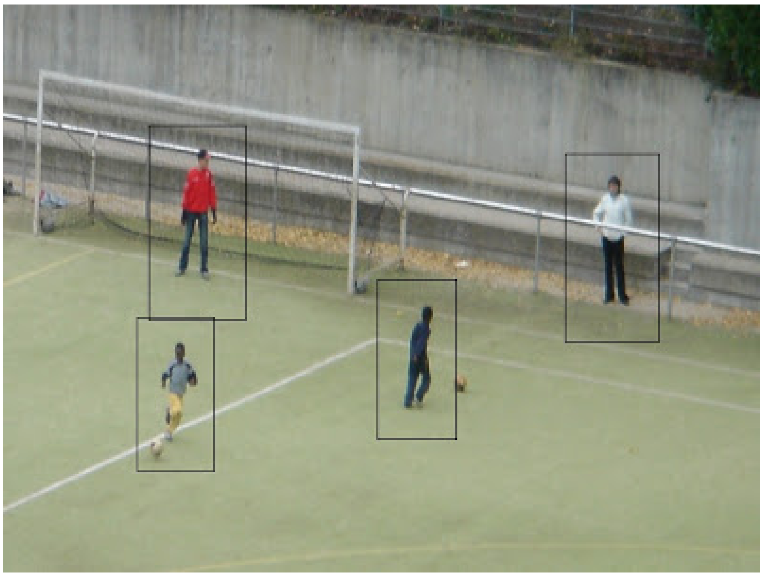

Figure 3.3: Merged rectangles.

3.4 Computation Time

In the original implementation of *ChnFtrs*, classification is done through a cascade of boosted classifiers. Each stage of the cascade grows in complexity and evaluates more features. So, if at any point in the cascade a sample is rejected, the detector stops evaluating features and goes on to the next window. In one 640x480 image there are ∼60000 windows to evaluate, and, on average, less then 0.1% of those windows will be positive for pedestrian, so, it follows that if most negative windows can be rejected by evaluating a small set of features, the algorithm will speed up by multiple orders of magnitude. Indeed, the cascade of boosted classifiers is the major key point for the speed of the detector, however, a great deal of complexity lies in training one and there are no standard tools that offer a ready to use implementation. Due to the short amount of time available to develop this project, it was not possible to implement a cascade. Instead, one big boosted classifier is utilized, largely sacrificing speed since the detector evaluates the whole set of features for every single detection window. In this setting, each image takes ∼20 seconds to be fully processed and classified if 15000 features are extracted per window.

However, given that the results in terms of detection rate do not vary much between both approaches, the validation of the method is not in question. Figure 3.4 illustrates the differences between a single boosted classifier and a cascade of boosted classifiers.

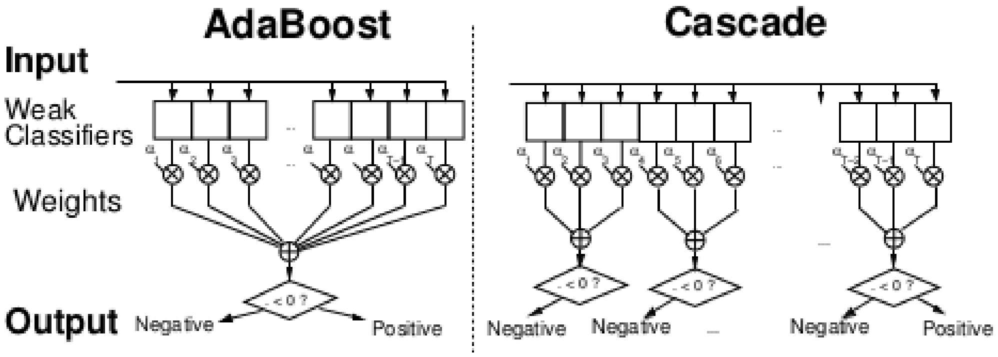

Figure 3.4: Difference between single boosted classifier and a cascade (Grossmann, 2004 [14]).

Chapter 4

Experiments and Results

There are two main aspects important to evaluate a detector's performance: true positive rate and false positive rate. These parameters are not independent, since a very permissive detector may achieve almost perfect results in terms of true positives, but with prohibitive amounts of false positives as a consequence. The conclusion is that these parameters must be assessed simultaneously in order to achieve a meaningful performance evaluation. This can be done through the use of ROC (Receiving Operating Characteristic) curves, which draw the relationship between true positive and false positive rates.
To test the detector's performance, experiments were carried out in two distinct datasets. The INRIA dataset provides a test set that allows for a straightforward quantification of results and makes them directly comparable to other detectors that were tested in the same manner, conferring objectiveness and significance to the obtained results.
The detector was also tested in a dataset acquired aboard the *Atlas Car* in Aveiro. Achieving a significant quantification of results on the *Atlas Pedestrian Dataset* demands the establishment of a ground truth that reflects the necessities of the application, so, rather than aiming for pedestrian individualization, situations representing any potential risk to pedestrians were the considered targets.
All the relevant aspects related to the attainment of results are discussed in this chapter.

4.1 INRIA Dataset Experiments

The INRIA Dataset provides a test-bed that largely facilitates the process of generating ROC curves. The adopted methodology, ground truth rules and results are discussed in this section.

4.1.1 Ground Truth

To produce the data to draw the curves, the algorithm was tested on 1132 positive windows and around 10 million negative windows taken from 452 different images. In these circumstances the ground truth is established in a straightforward manner, since it is known before-hand that each one of the 1132 positive windows contain a pedestrian, and none of the 452 images from which the negative windows originate contain a single pedestrian.

4.1.2 Methodology

For a given threshold, the positive windows are evaluated and every negative response from the detector means a missed pedestrian. Then the negative windows are tested in the same fashion, only now any positive response from the detector is a false positive. This process is repeated several times for different thresholds and the curves are drawn by connecting the various data points. To understand the effect of the feature pool size on the detector performance, two detectors are tested in the same way and compared, one trained with 15K features and the other with 10K.

4.1.3 Results

Figure 4.1 shows the ROC curves for both detectors. The x-axis measures in false positives per window and is log-scaled for a better visualization. The y-axis measures detection rate in percentage, which is the quotient between the true positives and the total number of pedestrian windows tested.

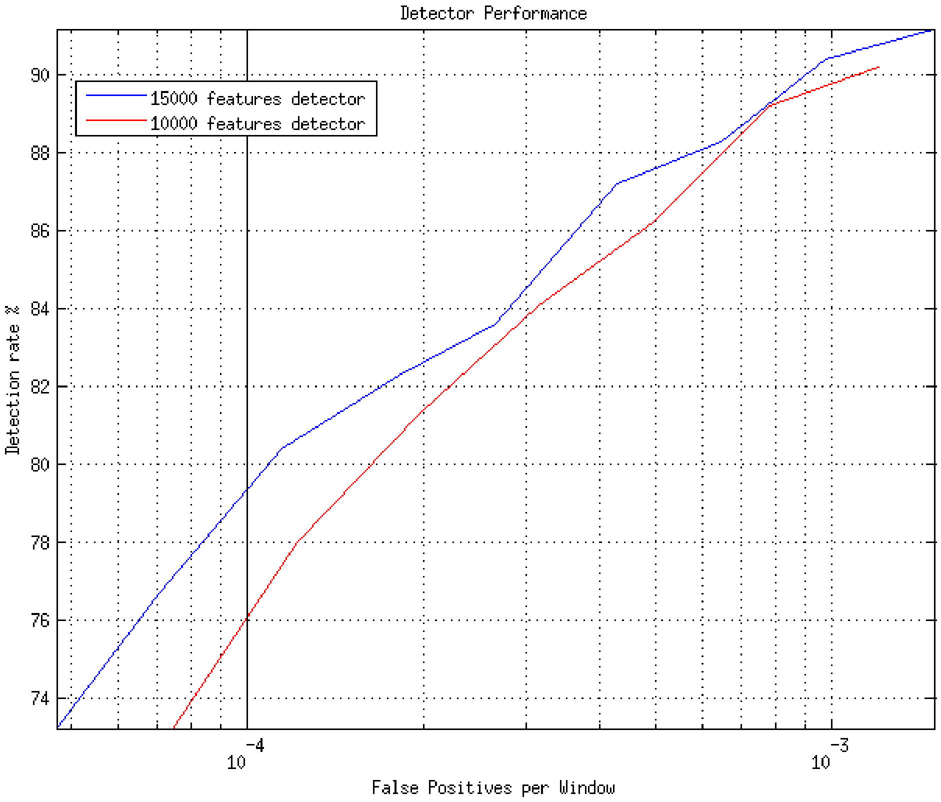

Figure 4.1: Detector performance on the INRIA dataset.

4.1.4 Commentary

On the reference value of 10^{-4} false positives per window, the algorithm trained with 15K features achieves 79.5% detection rate, and, although this is not a bad figure at all, the original implementation documents a detection rate of 91%, over 10 points above the presented implementation.

It was hypothesised that the explanation for this difference in performance could lie in the fact that the original classifier was trained with 30K features, rather than 15K. To back this hypothesis, a similar classifier was trained but now with only 10K features; if the number of features could be causing this under-performance, an even lower detection rate was expected, and, in fact, that is what the results show, with the 10K detector performing ∼3.5% points below the 15K detector. That is not to say that if this detector was trained with 30K features the results would be exactly the same as in the original, they would probably still be lower for different reasons. The method may be the same, but the implementations are unique, and this algorithm has still much room for tweaks and improvements.

Tests weren't made for 30K features because OpenCV's data structures wouldn't even allow for such big feature spaces. So, instead, it was decided that 15K and 10K feature detectors were to be trained and tested for performance in order to establish a coherent relation with the results shown in the original document.

4.2 *Atlas* Pedestrian Dataset Experiments

One of the main goals for this project was the implementation of a pedestrian detection infrastructure on the *Atlas Car*, making the evaluation of the detector on data recorded by the car's cameras a paramount procedure. This dataset was obtained in Aveiro on some areas known for being usually busy, such as the Avenida Lourenço Peixinho and the area around the university. Because it is important to test the detector on a wide variety of urban environments, some areas with less pedestrian activity were also recorded and tested.

Quantifying results isn't nearly as straightforward on the *Atlas* Pedestrian Dataset as it was on the INRIA Dataset, since labelling all the acquired data frames was an impossible to complete task in the available time to complete the project. It was then necessary to adopt a different methodology for obtaining significant results.

In this section ground truth rules, methodology and results will be discussed.

4.2.1 Ground Truth

A close analysis of the *Atlas* Pedestrian Dataset leads to the conclusion that in order to achieve any meaningful result out of real road data, a careful ground truth rule that takes into account the intended application has to be established. One can easily think that the results of an application for counting pedestrians have to be perceived differently then the ones for driving assistance, where detecting all pedestrians in the environment is not only unnecessary, but undesirable, since many appear in situations that do not pose any risk. Having this in mind, only pedestrians that appear at a close to medium range on the image are taken into account, since these are the ones that may be at risk, and pedestrians that appear either in the far opposite side of the street, or very far into the horizon are discarded. Also, rather than individualizing pedestrians, it is important to detect situations that may represent any potential risk. It was then assumed that two or more pedestrians that stand relatively close to each other represent one target. Figures 4.2 through 4.7 illustrate some situations where pedestrians were considered valid targets, and others where they were not. For easier visualization, green rectangles were manually drawn around valid targets, and red rectangles around not valid ones.

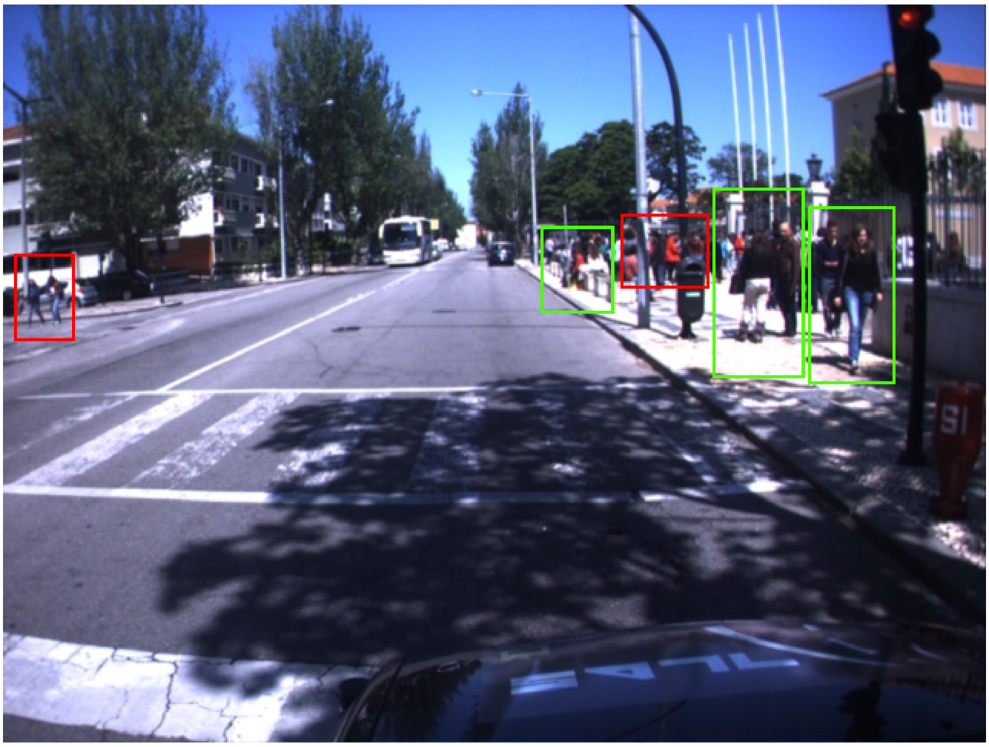

Figure 4.2: Ground truth example 1 - Close and medium range pedestrians that stand in the sidewalk are targets, whereas far range pedestrians in the opposite side of the road or far in the sidewalk are not.

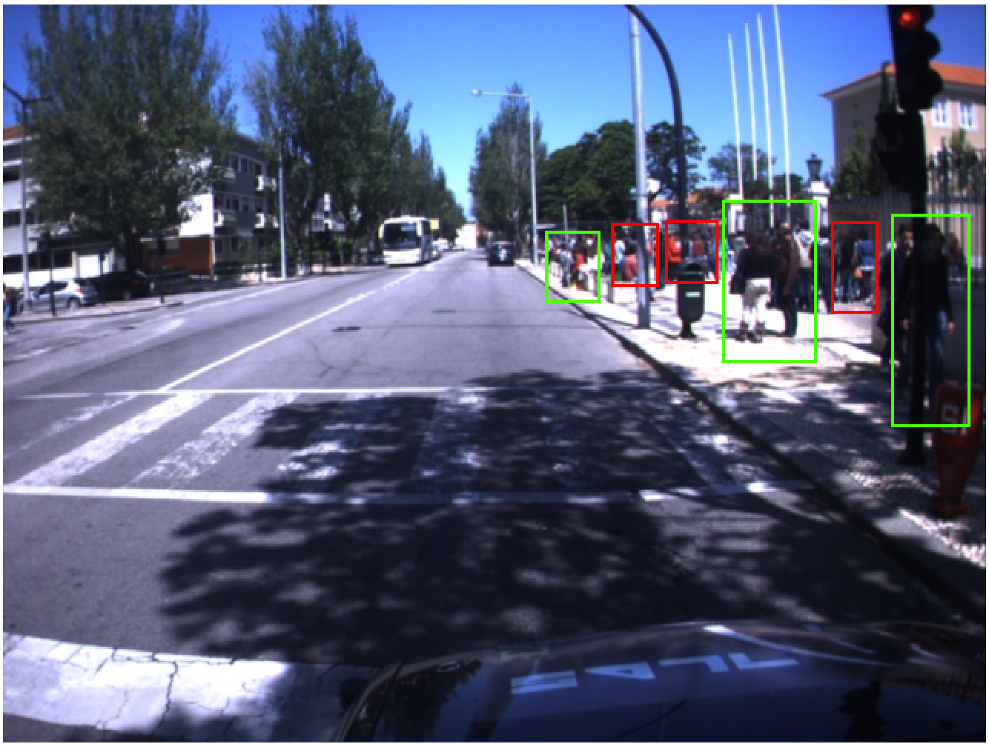

Figure 4.3: Ground truth example 2 - Occluded pedestrians that stand in close or medium range are also valid targets.

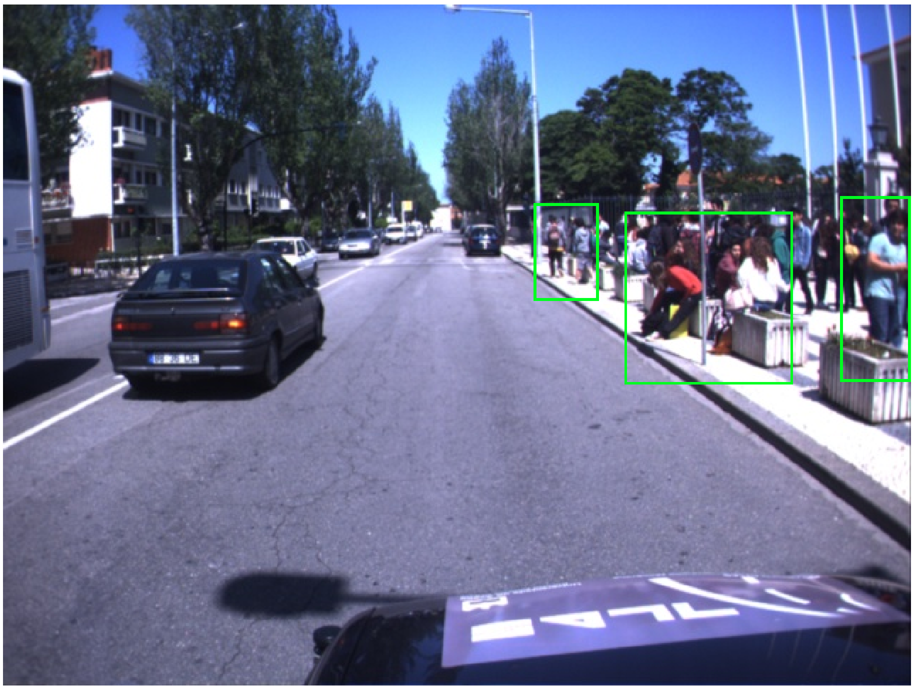

Figure 4.4: Ground truth example 3 - Groups of pedestrians standing close to each other are assumed to be only one target.

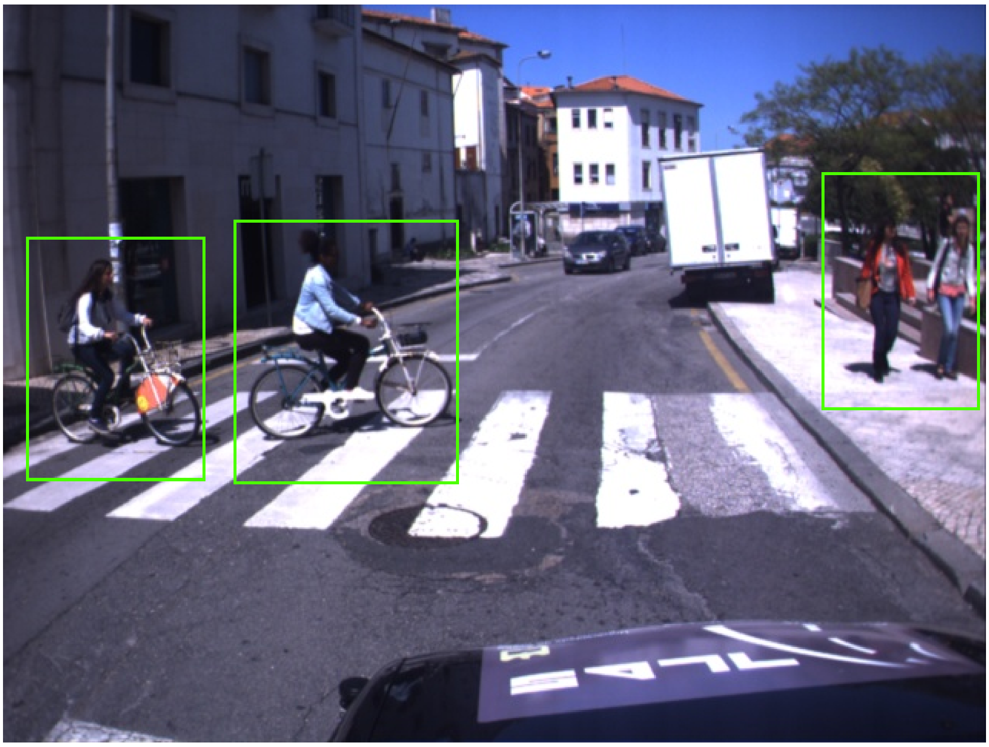

Figure 4.5: Cyclists are considered valid targets.

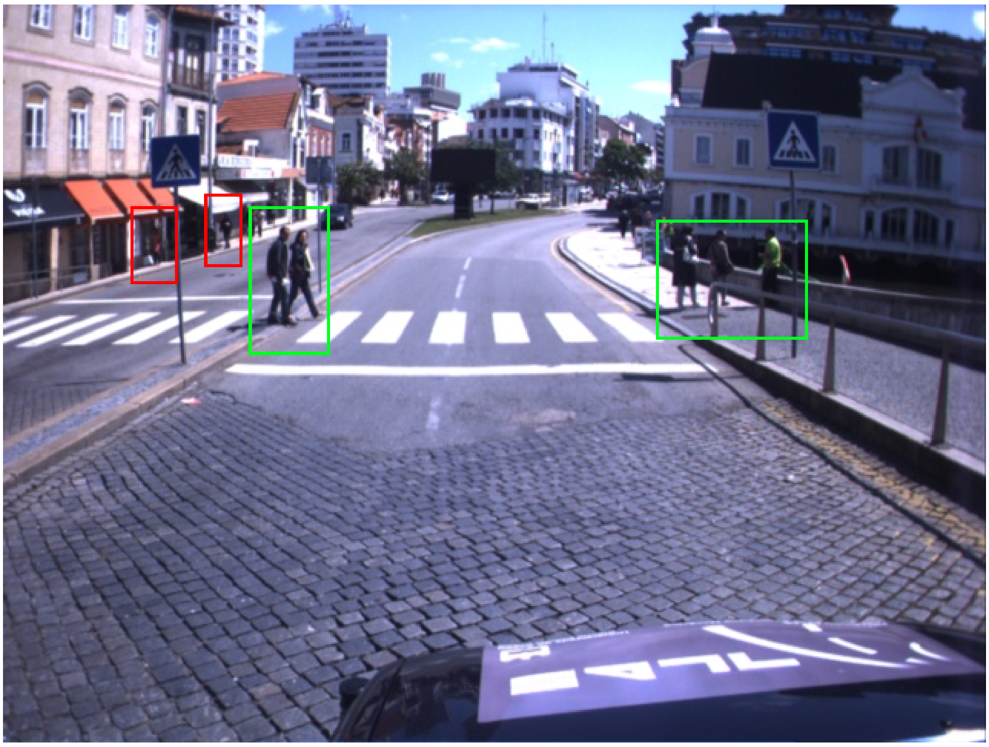

Figure 4.6: Ground truth example 5 - Two pedestrians crossing the street together are considered to be one target.

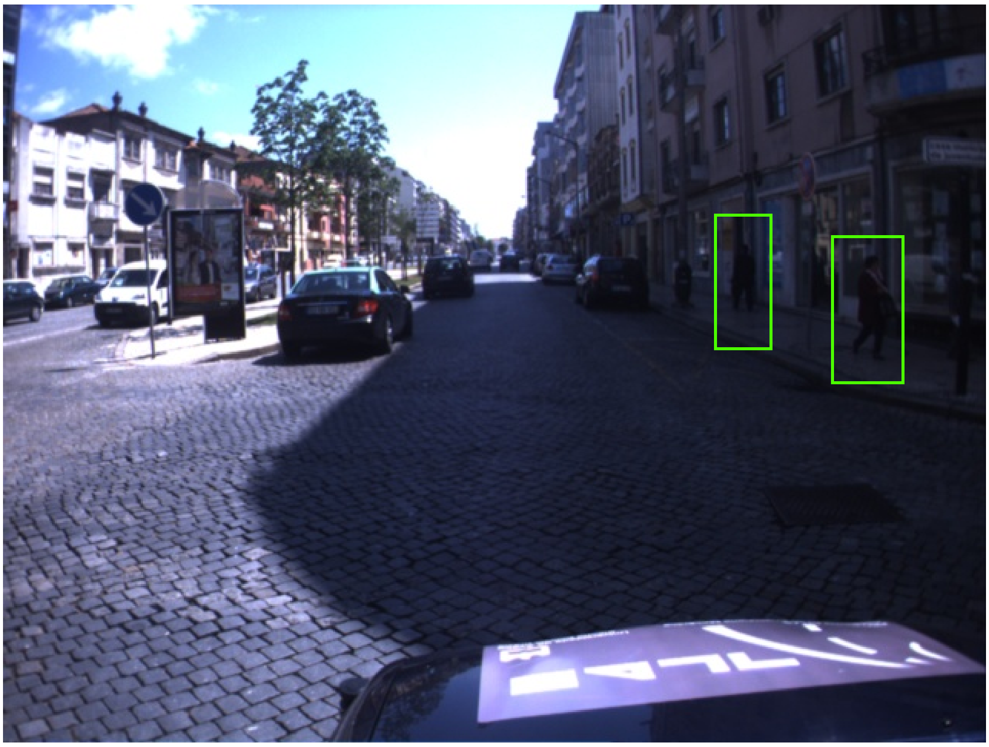

Figure 4.7: Ground truth example 6 - Difficult examples such as shadowed pedestrians are valid targets.

4.2.2 Methodology

To assess the detector's performance on the *Atlas* Pedestrian Dataset, the program is set to draw green rectangles around positive detections on the images, and save them to disk. Afterwards, the images are analysed one by one to identify the risk situations involving pedestrians, and whether they are detected or not.

A preliminary assessment of the detector's behaviour makes it clear that most of the false detections occur on vertical-shaped, pedestrian-like objects. To objectively understand the behaviour of the detector on false detections, the concept of *Pseudo-False Positives* (PFP) was used. These are false detections that instead of behaving randomly, follow a predictable pattern that can be easily understood. In this work, false detections on objects of vertical expression, such as trees, traffic signs or mail boxes are labelled as PFP. So, in an attempt to lower the PFP rate and determine whether the nature of the training data could influence the performance, a third round of bootstrapping was carried out on top of the previous INRIA data. Using just a few selected images with no pedestrians, ~750 hard negative windows were added to the training data and a new detector was trained. The detector trained solely with INRIA data is referred as *INRIA Detector* and the one that was modified to include some *Atlas* examples is referred as *Atlas Bootstrapped Detector*. They are compared in terms of detection rate over a given number of False Positives per Image (FPPI).

4.2.3 Results

The detectors described in the previous section were both tested in the same data logs using the *rosbag* feature provided by ROS, which allows for data to be replayed as it was recorded in the field.
It is important to mention that to perform these experiments, the re-player had to be set to run 16 times slower than the original frame-rate and, since experiments were not carried out at the same time for both detectors, the frames that were analysed were not exactly the same. Because of that, there is a slight difference in the number of frames tested for each detector and consequently a varying number of targets. This situation is also explained by the fact that in an experiment a pedestrian might happen to be visible for multiple frames and not on the other. However, rather than invalidating the evaluation approach, this circumstance further attests to the method's capacity for generalization.
The evaluated parameters displayed on table 4.1 are the following:

- True Positives: correctly labelled targets.
- False Negatives: not correctly labelled targets.
- False Positives: wrong positive classifications of non-targets.
- Pseudo-false Positives: wrong positive classifications of pedestrian-shaped non-targets.
- Detection Rate: ratio between the correctly labelled targets and the total number of them.
- False Positives per Image: ratio between total number of both types of false positives and the total number of frames tested.

Atlas Pedestrian Dataset Results		
Parameter	**INRIA Detector**	**Atlas Bootstrapped Detector**
Total frames analysed	529	540
Total number of targets	189	207
True Positives	81	101
False Negatives	108	106
False Positives	102	89
Pseudo-false Positives	142	88
Detection Tate	42.8%	48.7%
False Positives per Image	0.45	0.33

Table 4.1: Atlas Pedestrian Dataset Results

To provide a better understanding of how the detection works, some example results are shown in figures 4.8 through 4.19, in which green rectangles were automatically drawn by the program. Some of the most common urban pedestrian detection situations are covered by the example images that follow.

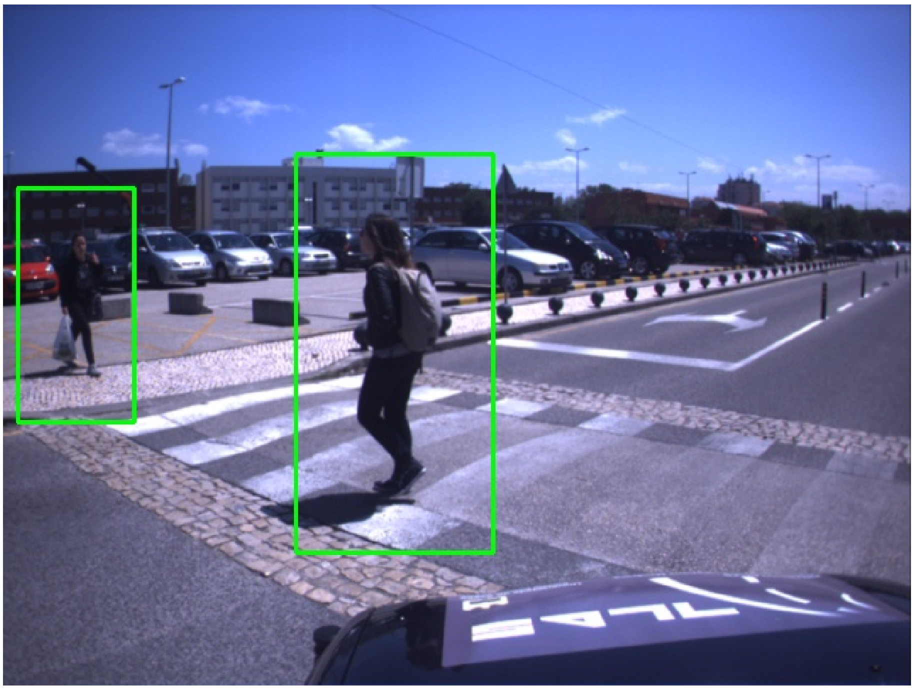

Figure 4.8: Example result 1 - Close range pedestrians crossing the street don't go undetected.

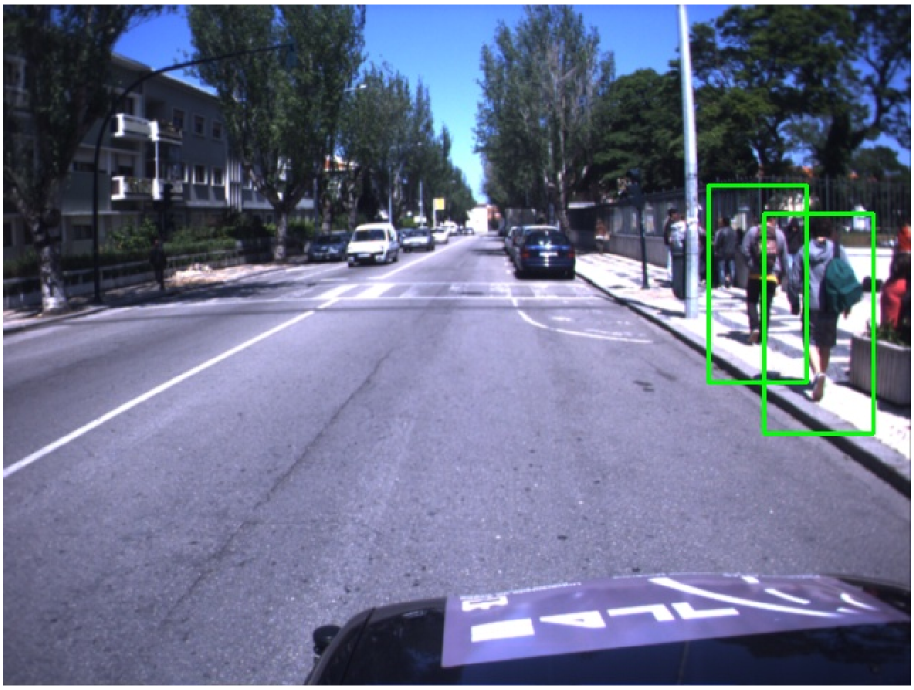

Figure 4.9: Example result 2 - In bright environments, close range pedestrians walking on the sidewalk are usually detected.

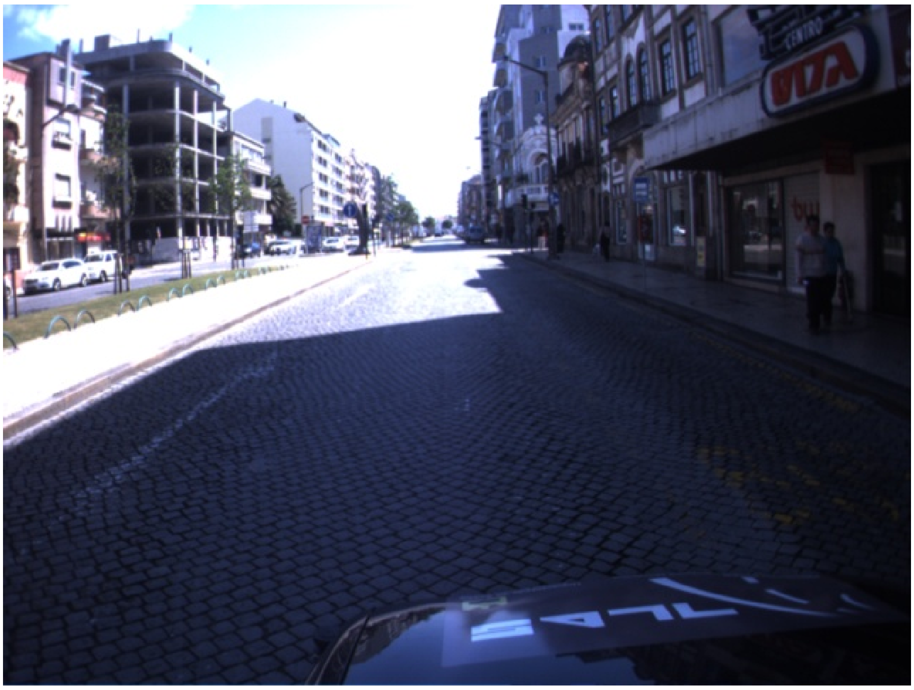

Figure 4.10: Example result 3 - Shadowed pedestrians highly contribute to the miss rate.

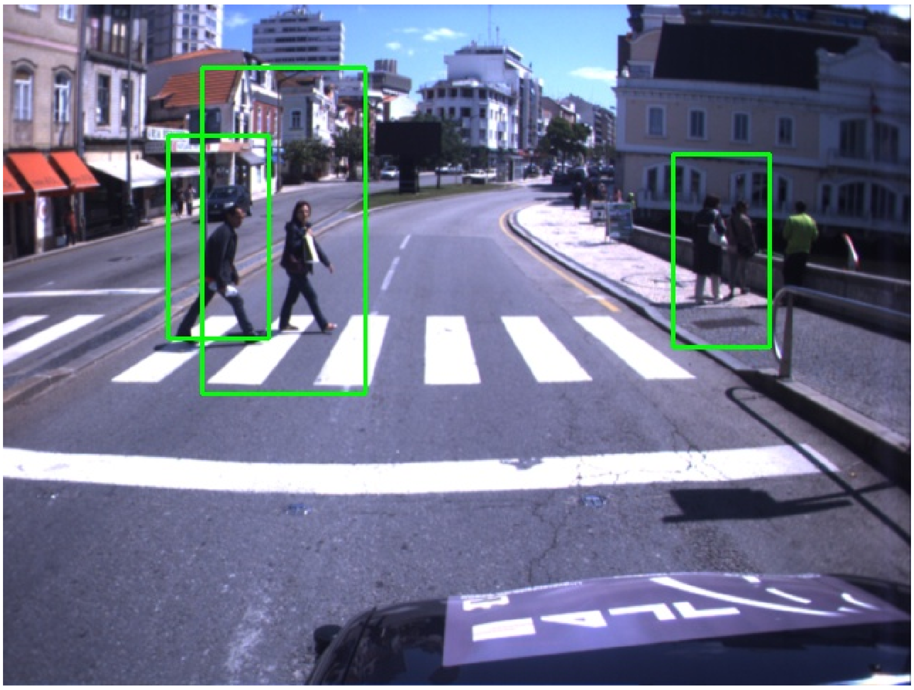

Figure 4.11: Example result 4 - Groups of pedestrians are often detected as one target, rather then being individualized. Also, partially occluded pedestrians are often missed.

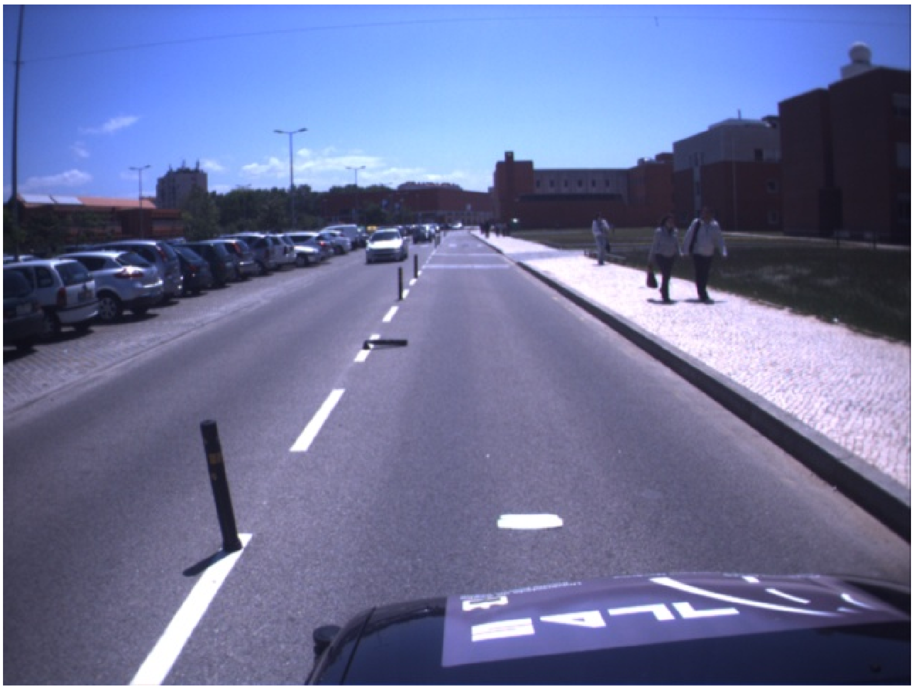

Figure 4.12: Example result 5 - Medium and far range pedestrians are sometimes missed even in bright environments.

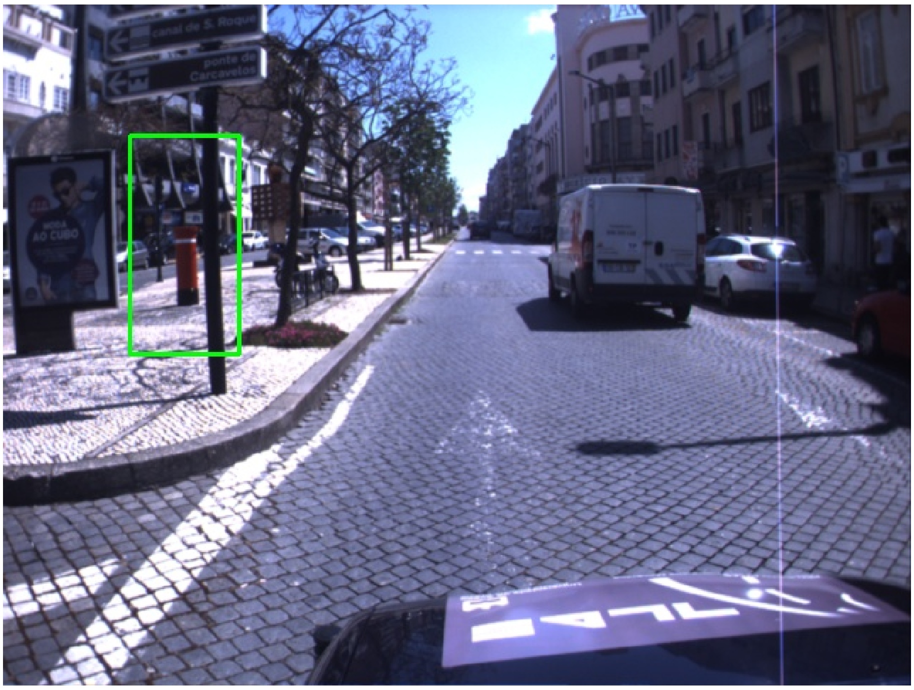

Figure 4.13: Example result 6 - Example of a common pseudo-false positive.

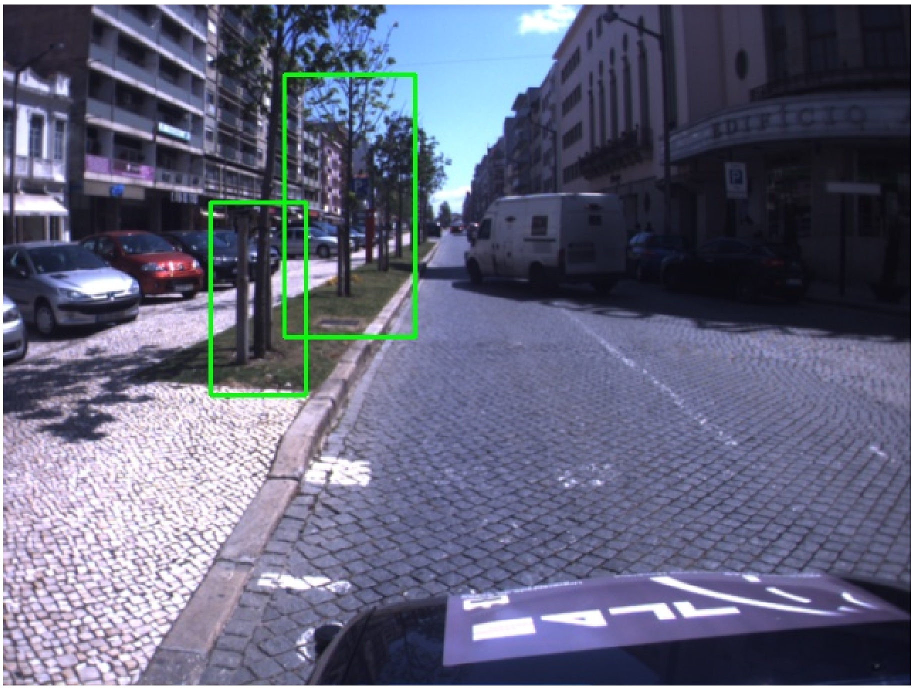

Figure 4.14: Example result 7 - Trees are also commonly detected as pedestrians.

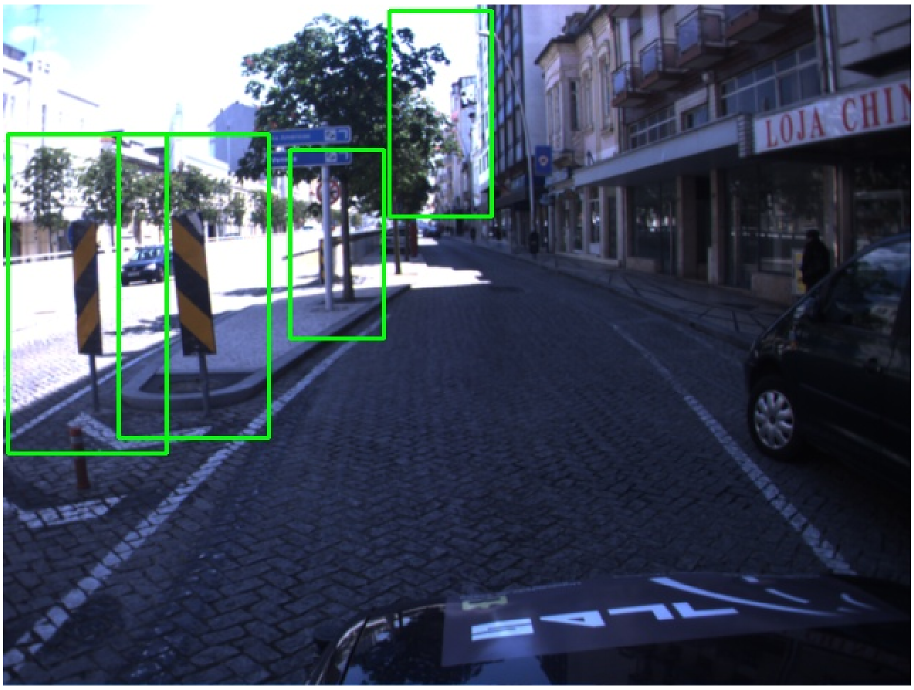

Figure 4.15: Example result 8 - Traffic signs are also confused by the detector rather often. On the right, another example showing that occlusion and shadowed environments are major issues in pedestrian detection.

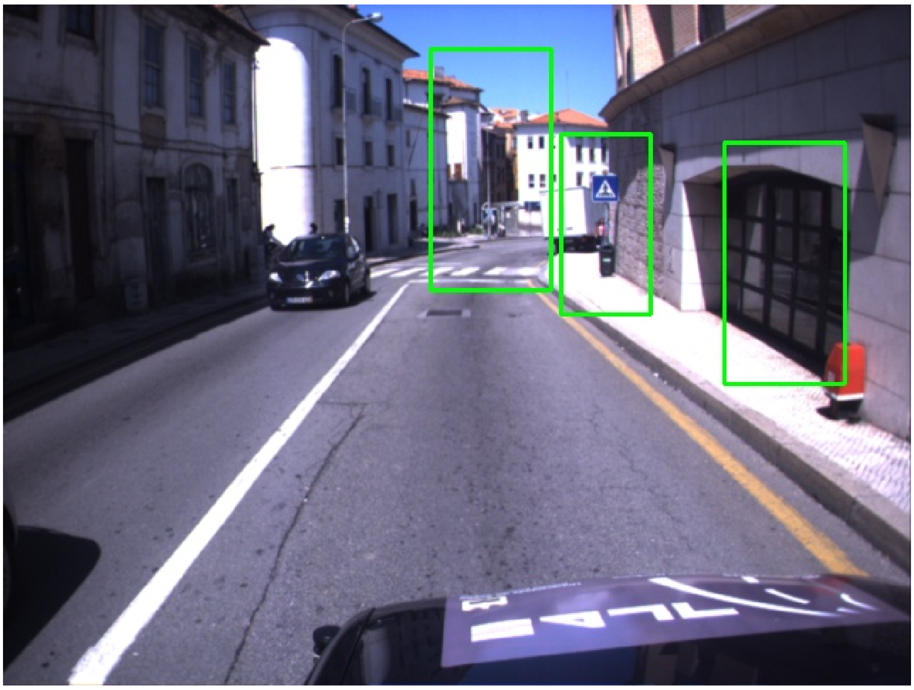

Figure 4.16: Example result 9 - The detector occasionally makes random false detections.

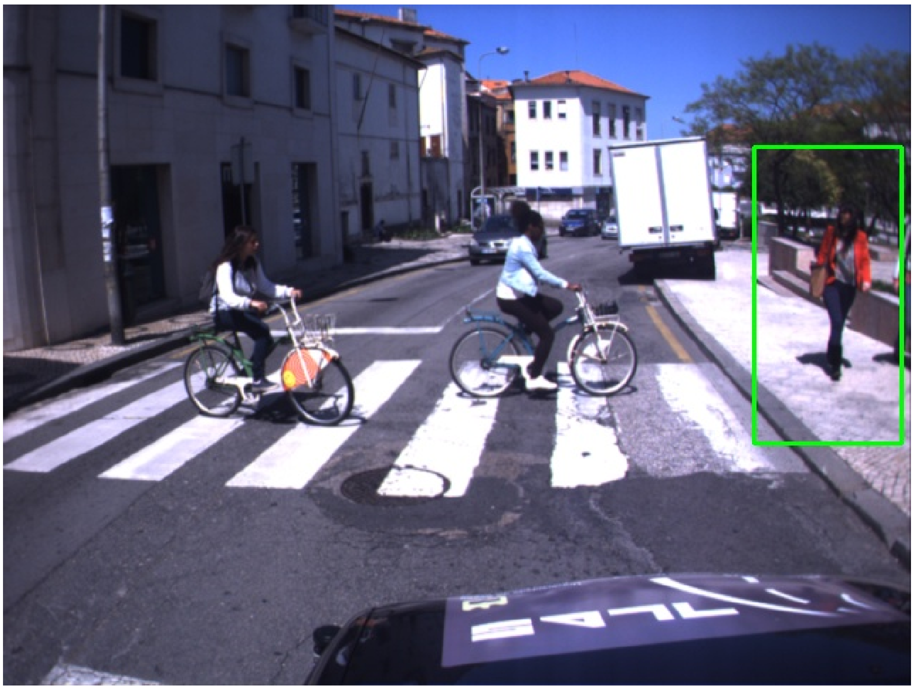

Figure 4.17: Example result 10 - Cyclists are often missed when they lack the vertical expression of a pedestrian.

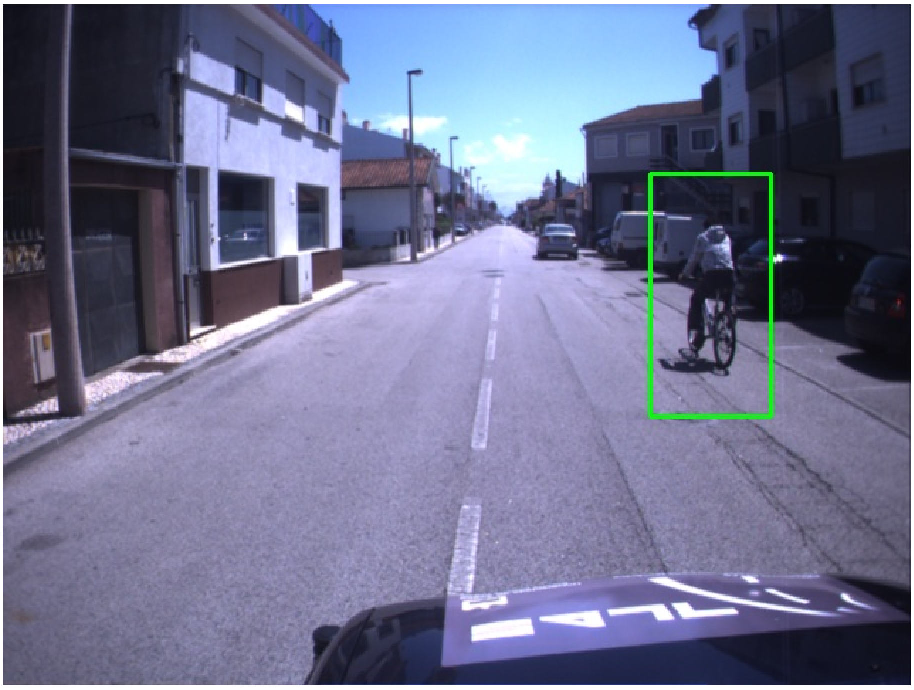

Figure 4.18: Example result 11 - Cyclists are detected if they appear on frontal or back perspectives.

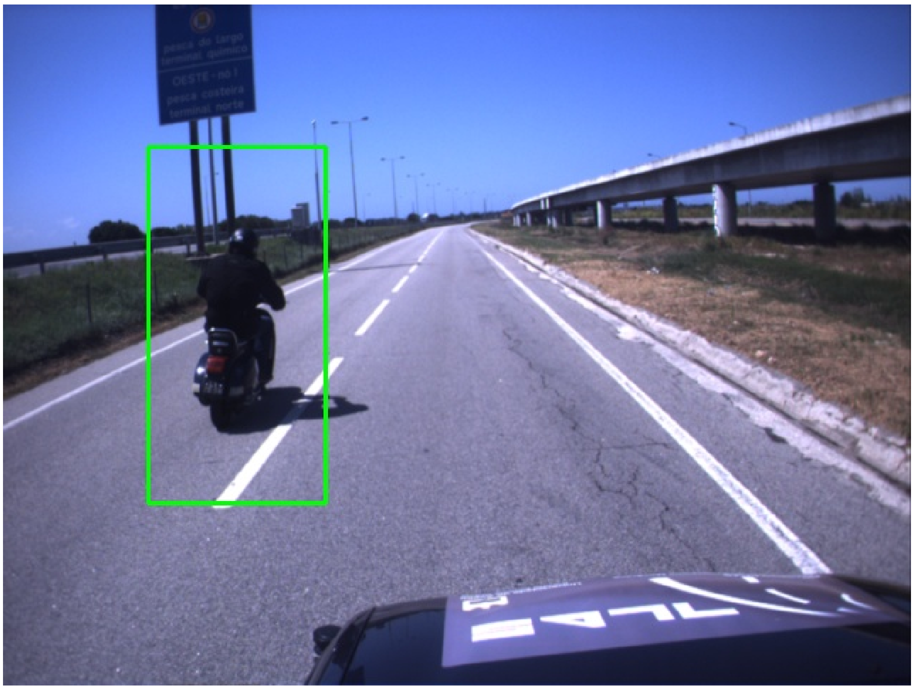

Figure 4.19: Example result 12 - A result obtained on the highway, where a motorcyclist was wrongly identified as a target.

4.2.4 Commentary

Although a quantification of the results on this dataset was obtained, it is important to have in mind that these are preliminary results, and they might not assess with a great degree of confidence to the performance of the detector. Obtaining more confident results would require more experiments and a larger testing data. It is, however, a preliminary result that gives a fair idea of the potential of this approach when applied to driving assistance.

At a first glance, the achieved detection rate seems to be a low one, but, to make a correct interpretation of the results, it is crucial to have in mind that the primary detector that was used in the *Atlas* data was solely trained with images from the INRIA dataset, which are of completely different nature than the *Atlas* ones, as most aren't even of urban environment, and the sensors used to acquire the data are also different. This provides a hint that the adopted method generalizes well for images of different nature. However, the results improved significantly just by adding a few hundred examples from the *Atlas* dataset to the training data. The improvement of the results is best observed in the lowering of the false positive rate, which accounts for the fact that only negative examples were added to the training data. The lower occurrence of false positives made possible for a less demanding threshold to be used in the *Atlas Bootstrapped Detector* and, because of that, the detection rate went from 42.8% to 48.7% with an even lower rate of False Positives per Image. One is inclined to think that if the detector was trained solely or mainly with *Atlas* data, the results would improve even further.

Chapter 5

Conclusions and Future Work

An application for visual recognition of pedestrians was successfully implemented in this work. *Integral Channel Features* uses a simple and generic framework that takes advantage of the richness of the information present in various channels of an image to produce a number of features capable of distinguishing between the presence and absence of pedestrians through the use of a trained classifier based on AdaBoost.

To validate the approach, experiments were carried out in two distinct datasets. The INRIA dataset served as an important test-bed because of the objectiveness and significance it conferred to the obtained results. In this dataset, the detector achieves a detection rate of almost 80% on the reference value of 10^{-4} false positives per window, which is in line with many detectors found on literature. It is important to mention, however, that in the publication on which this work was based, a detection rate of 91% on the same reference value was documented. The explanation for this difference in performance may lie in the fact that the original detector was trained with twice the features that were used in the presented implementation, and it was shown that performance grows rapidly with the number of features utilized. It is safe to assume that if twice the features had been used, a significantly higher detection rate would have been achieved.

Tests were also performed on visual data captured on the *Atlas Car*. A preliminary quantification of the results was done by analysing more than 500 frames acquired in Aveiro. A detector trained solely with INRIA data achieved a detection rate of 42.8% at the False Positive per Image rate of 0.45, and this detection rate was amplified to 48.7% just by adding a few hundred hard negative samples from the *Atlas* Pedestrian Dataset to the training data, at the even lower rate of False Positives per Image of 0.33.

In a more qualitative analysis, it was observed that every single pedestrian that was captured in close-range was detected, showing that the method generalizes very well on images of different nature than the ones used for training. The detector also behaves well on mid-range pedestrians given a bright environment. Far-range pedestrians, however, are often missed, which could be partially explained by the fact that the pedestrian windows used for training only contain close and mid-range pedestrians. This problem would probably be minimized if the image was subject to an upscaling prior to analysis, with sacrifices to the computation time. On crowded areas the detector makes usually several hits, but often not individualizing targets, giving more confirmation that the detector is obviously prone to make positive classifications where pedestrians appear.

As one would expect, shadowing and partial occlusion are responsible for most missed pedestrians, as this factors are known to be hard to overcome. Pedestrians on the sidewalk are detected less often than pedestrians that are crossing the street, even if they appear in the same range. This odd result can be explained by a slight distortion that occurs on the *Atlas* images but not on the INRIA's, certainly causing the image to have different properties on the borders and more missed pedestrians as a consequence.

The detector also makes a fair amount of false detections, mainly on areas with great density of objects, but it was surprising to find that the vast majority were not random detections at all, but instead followed a consistent and predictable pattern. Most false detections occur on vertical-shaped, pedestrian-like objects, such as trees, traffic signs, lighting poles, mail boxes and other similar objects. This situation not only attests for the validity of the method, since it is obvious that for the most part pedestrian-shaped objects are being detected, but also opens the possibility of the implementation of a post-processing method that could further differentiate these false positives from actual pedestrians.

In fact, there is a wide range of possibilities to improve the framework. Improving the detector speed can be done through the use of a decision structure based on a cascade of boosted classifiers, rather than a single big boosted classifier. This implementation alone would improve the detector speed by multiple orders of magnitude without sacrifice to results (in fact they could be even better). Not only that, a boosted cascade would allow for more features to be evaluated with almost no speed loss, improving detection rate significantly. Still working on the algorithm itself, multi-scale detection over a dense pyramid can be avoided since (Dóllar et al., 2010 [8]) demonstrated that features can be approximated at nearby scales, thus improving speed even further. Moreover, this algorithm can be even further developed to take advantage of the correlation between detector responses at nearby window positions, a concept introduced in (Dóllar et al., 2012 [7]) which documents multi-scale detection at 30 FPS.

Other directions can be taken to develop this framework. Using other sensor information such as laser or thermal image to generate possible candidates prior to classification would eliminate the need to analyse the full image, and computation time would decrease exponentially. In fact, multi-sensor detection systems are unavoidable when it comes to robust and reliable detectors, and the work developed in this project is open to that possibility.

As it was shown on by the *Atlas Bootstrapped Detector* results, the data used for training could also be better focused on the application, since the INRIA dataset is rather poor on urban scenarios. So, gathering a more complete dataset highly focused on road situations and use it for training would most certainly greatly improve the detection results on urban environments.

Although this work is highly focused on ADAS, it is obvious that an infrastructure for visual human detection has a wider range of applications. In fact, the work developed in this project is generic enough to be applicable in any field, and one could imagine the usefulness of detecting humans for security, robotics and many other applications.

The greatest contribution of this project was the creation of a framework for pedestrian detection to be incorporated in the *Atlas Car*, a feature that had not yet been developed in the scope of the project. It can and most certainly will be improved in the future, and, although this is a personal opinion, I think that a very robust, fast and reliable detector can be achieved if some of the proposed improvements were to be developed and integrated in the present framework.

References

[1] M. Bertozzi, A. Broggi, C.H. Gomez, R. I. Fedriga, G. Vezzoni, and M. Del Rose. Pedestrian detection in far infrared images based on the use of probabilistic templates. In *Intelligent Vehicles Symposium, 2007 IEEE*, pages 327–332, 2007.

[2] Tomaso Poggio Constantine Papageorgiou. A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.

[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Mach. Learn.*, 20(3):273–297, September 1995.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893 vol. 1, 2005.

[5] Navneet Dalal, Bill Triggs, and Cordelia Schmid. Human detection using oriented histograms of flow and appearance. In *European Conference on Computer Vision*, 2006.

[6] Departamento de Engenharia Mecânica da Universidade de Aveiro. http://atlas.web.ua.pt, 2013.

[7] P. Dollár, R. Appel, and W. Kienzle. Crosstalk cascades for frame-rate pedestrian detection. In *ECCV*, 2012.

[8] P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *BMVC*, 2010.

[9] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, 2009.

[10] P. Dollar, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: An evaluation of the state of the art. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(4):743–761, 2012.

[11] B. Fardi, U. Schuenert, and G. Wanielik. Shape and motion-based pedestrian detection in infrared images: a multi sensor approach. In *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pages 18–23, 2005.

[12] David J. Fleet and Yair Weiss. Optical flow estimation, 2005.

[13] D.M. Gavrila and V. Philomin. Real-time object detection for ldquo;smart rdquo; vehicles. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 87–93 vol.1, 1999.

[14] Etienne Grossmann. Automatic design of cascaded classifiers. In Ana Fred, TerryM. Caelli, RobertP.W. Duin, AurélioC. Campilho, and Dick de Ridder, editors, *Structural, Syntactic, and Statistical Pattern Recognition*, volume 3138 of *Lecture Notes in Computer Science*, pages 983–991. Springer Berlin Heidelberg, 2004.

[15] Mercedes Press Information. Extended pre-safe protection, 2013.

[16] S. Maji, A.C. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8, 2008.

[17] S. Milch and M. Behrens. Pedestrian detection with radar and computer vision, 2001.

[18] Mobileye. Pedestrian collision warning, 2012.

[19] European Road Safety Observatory. Traffic basic facts, 2011.

[20] C. Premebida, G. Monteiro, U. Nunes, and P. Peixoto. A lidar and vision-based approach for pedestrian and vehicle detection and tracking. In *Intelligent Transportation Systems Conference, 2007. ITSC 2007. IEEE*, pages 1044–1049, 2007.

[21] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[22] P. Sabzmeydani and G. Mori. Detecting pedestrians by learning shapelet features. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, 2007.

[23] Robert E. Schapire. A brief introduction to boosting. In *Proceedings of the 16th international joint conference on Artificial intelligence - Volume 2*, IJCAI'99, pages 1401–1406, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[24] O. Tuzel, F. Porikli, and P. Meer. Pedestrian detection via classification on riemannian manifolds. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(10):1713–1727, 2008.

[25] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–511–I–518 vol.1, 2001.

[26] P. Viola, M.J. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 734–741 vol.2, 2003.

[27] S. Walk, N. Majer, K. Schindler, and B. Schiele. New features and insights for pedestrian detection. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1030–1037, 2010.

[28] Christian Wojek and Bernt Schiele. A performance evaluation of single and multi-feature people detection. In *DAGM-Symposium*, pages 82–91, 2008.

Appendix A

Usage Instructions

A launch file was created to launch the necessary ROS nodes for the program to run. One of the launch files is used to process images from file. It launches a server that checks a folder path for image files and sends them one by one over a ROS topic. If the client is subscribed to that same topic it will receive and process each received image. This launch file also launches a image view model, which is a ROS node that subscribes to image messages and displays them on a window.
Simple changes like choosing folder path or changing the frame rate are done by manually changing the source code and re-compile the program.
The next box shows the code for the launch file described above.

```
<launch>
       <node name="Client" pkg="PedestrianDetect"
       type="client" output="screen"/>

       <node name="Server" pkg="PedestrianDetect"
       type="server" output="screen"/>

       <node name="image_view" pkg="image_view" type="image_view">
          <remap from="image" to="Image_Out"/>
          </node>
</launch>
```

Provided access to the LAR tool kit, follow the next steps to run the program:

1. Open a terminal on Ubuntu.

2. Enter "roscd PedestrianDetect".

3. Too access and change the source code enter in the terminal "kate src/serverimg2.cpp" and "kate src/clientimg4.cpp".

4. Compile by entering "make" on the PedestrianDetect path or "rosmake PedestrianDetect" anywhere.

5. Run the program by entering "roslaunch PedestrianDetect peddetect.launch".

If the goal is to run the program on a *rosbag* or output from a camera, there is no need to launch the server. A very similar launch file was created that launches only the client and the image view. To use it follow the same steps described above but substitute "peddetect.launch" for "bagplayer.launch".