École Centrale de Nantes

Master's Thesis

Pedestrian Detection And Tracking

Author:
Izzatbek Mukhanov

Supervisor:
Philippe Martinet
Gaëtan Garcia
Salvador Domínguez Quijada

Robotics Department
IRRCYN

August 2014

ÉCOLE CENTRALE DE NANTES

Abstract

ARIA 2 - ROBA
IRRCYN

Master of Science

Pedestrian Detection And Tracking

by Izzatbek Mukhanov

This work gives an overview of currently existing methods of performing Pedestrian Detection and Tracking. Different detection techniques are compared. A working Pedestrian Detection and Tracking system is implemented using Integral Channel Features and AdaBoost classifier to detect pedestrians on the image. Sensor fusion of laser scanners and cameras is performed. Regions of interest are found using laser scanners to speed up the process and to improve the quality. Filtering is applied to track pedestrians in difficult situations and to reduce the number of false detections.

Contents

List of Figures

Abbreviations

ADAS	**A**dvanced **D**river **A**ssistance **S**ystem
BB	**B**ounding **B**ox
BOF	**B**ayesian **O**ccupation **F**ilter
CPU	**C**entral **P**rocessing **U**nit
FPPI	**F**alse **P**ositive **P**er **I**mage
FPPW	**F**alse **P**ositive **P**er **W**indow
FPS	**F**rames **P**er **S**econd
GPU	**G**raphics **P**rocessing **U**nit
HOG	**H**istogram of **O**riented **G**radients
IR	**I**nfra **R**ed
NMS	**N**on **M**aximum **S**uppression
OS	**O**perating **S**ystem
PDT	**P**edestrian **D**etection and **T**racking
ROI	**R**egion **O**f **I**nterest
ROS	**R**obot **O**perating **S**ystem
SVM	**S**ingular **V**alue **D**ecomposition
SVM	**S**upport **V**ector **M**achine

Chapter 1

Introduction

1.1 Motivation

Due to a large number of potential applications, Pedestrian Detection and Tracking (PDT) has become an extremely active area in computer vision research. The areas of application include Ambient Intelligence, advanced user interfaces, automated surveillance, image compression, etc. We will be particularly concerned with the PDT in the domain of vehicles.

In the last few years, there has been an increasing number of accidents on the street. Pedestrians constitute a large proportion of casualties amongst fatally injured road users (2 to 3 million each year [1]).

Both the scientific community and the automobile industry have contributed to the development of different types of protection systems in order to improve traffic safety.

Over the last decade, research has moved towards systems that react before actual collision situation to protect pedestrians. Those systems are called Active Pedestrian Systems. Whether it is a matter of Advanced Driver Assistance System (ADAS) or fully autonomous vehicle (figure 1.1), those systems may intervene in critical situations and prevent a possible collision or mitigate the severity of an inevitable collision (for example, by autonomous brake system).

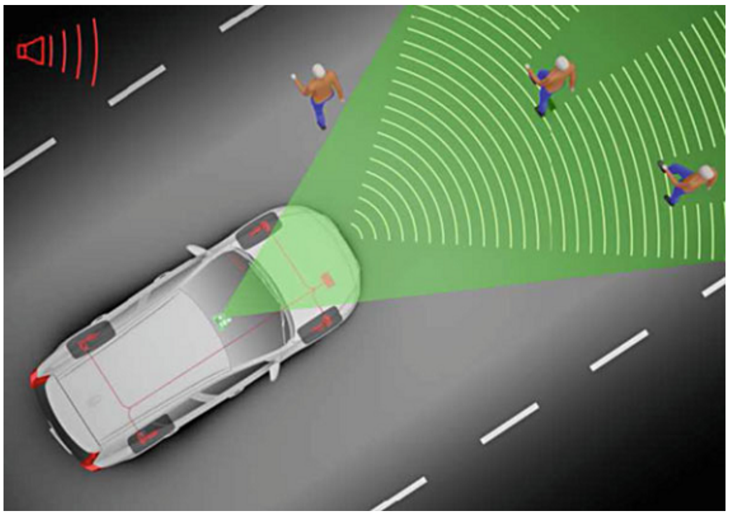

FIGURE 1.1: An example of Pedestrian detection system with full auto brake. The car is equipped with a radar and a camera [2].

Active safety measures consist of sensor system, functional algorithms and actuating elements (Figure 1.2). We will concentrate, particularly, on the second part. Our sensor system is detailed later (Section 1.3).

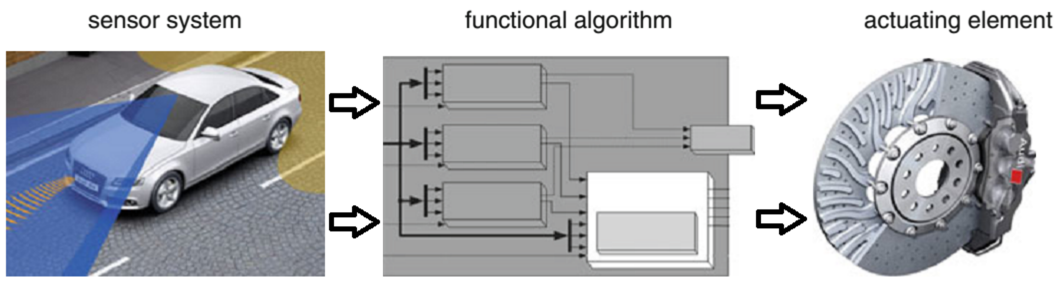

FIGURE 1.2: Schematic components of active safety systems [2].

Therefore, the main objective of the thesis is to develop a robust Pedestrian Detection and Tracking system implemented for use in smart vehicles to perform pedestrian protection and safety. This system should be able to detect dangerous situations involving pedestrians ahead of time in order to warn the driver or to automatically control the vehicle. Such systems are even more valuable for distracted or disabled people.

In addition, the system can be extended to be used to analyze and predict behavior of people. For example, by labeling what a pedestrian is doing (running, crossing a road, not moving, etc.) and predicting further movements.

The following steps are carried out during the project:

Outline the current state of the art The state of the art in PDT is discussed. Several techniques are reviewed. The thesis does not attempt to cover all aspects of PDT.

Implement working PDT First, detection was implemented using vision, then laser was added to improve results. Afterwards, tracking was implemented.

Analyze and compare different methods Comparison and analysis of different algorithms is carried out.

1.2 Challenges

While notable progress is done in PDT with the use of static cameras and simple background models, it is still very challenging to cope with the problem in highly dynamic, cluttered environments, especially when the observer itself is moving.

Pedestrian detection is a difficult task to perform from the computer vision perspective. The lack of explicit models of pedestrians leads to the machine learning based approaches which make use of provided examples to implicitly construct models.

The following challenges are generally encountered during PDT [3]:

Pedestrian appearance and pose Different local and global appearance is caused by various types of clothing. The non-rigid body of a pedestrian can undergo a large number of transformations.

Pedestrian position and orientation A pedestrian can be viewed at different orientations and distances.

Occlusion A pedestrian can be occluded by other people, the environment or carried things (bags, hats, briefcases).

Camera Field of View A pedestrian may be entering or exiting the scene.

Environmental Conditions Different weather and lighting conditions, reflections, shadows and weak variability in color intensity may cause difficulties.

Static or Moving Pedestrians A pedestrian may not move or move in an unpredictable, non-linear fashion, thus causing difficulties in tracking.

Splitting and Merging Pedestrians may be difficult to track when they are occluded by each other (for example, in crowded scenes).

A few examples are shown on figure 1.3.

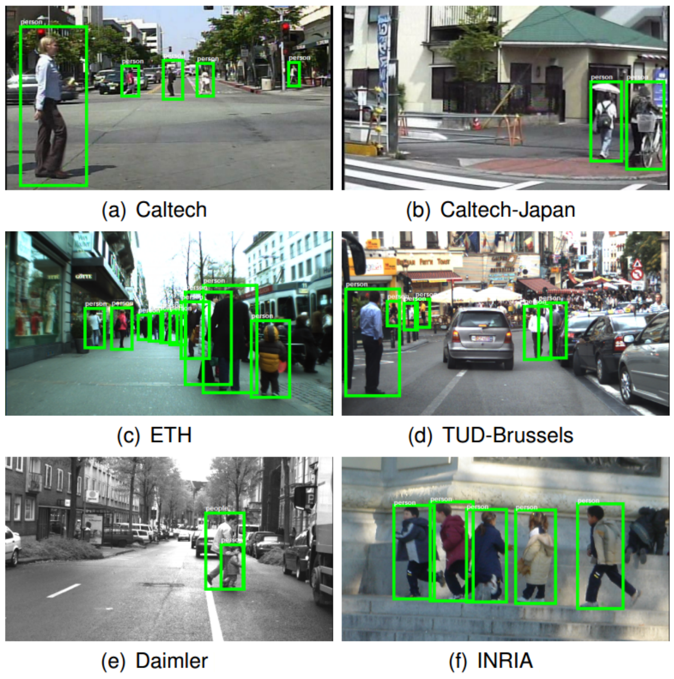

FIGURE 1.3: Example images (cropped) and annotations from six pedestrian detection datasets [4].

1.3 Equipment

The decision for triggering active pedestrian safety elements on the vehicle is made by algorithms which receive sensor signals as input data. This includes the state of the vehicle and the environment. Currently, there are many ways to perform detection of pedestrians [2]. The most popular ones are the followings:

Mono camera

With a single camera it is not possible to directly estimate the depth of an object without considering image sequences.

Stereo camera

Whereas, a pair of cameras can be used to estimate the depth of an object using *triangulation* provided transformation between cameras is known.

IR camera

Infrared cameras show a good performance during the night time, while having restricted performance in daylight. It picks up thermal radiation from objects within its field of vision and displays the temperature differences.

Radar sensor

This sensor gives a good estimation of the position and velocity in longitudinal direction, while having less speed and position resolution in lateral direction. It is suitable for active pedestrian protection systems to a limited extent.

Laser scanner

This system is much more stable with respect to ambient light. It uses light pulses to an object. It provides limited information about the height depending on the number of beams. It can be used to create local maps around the car in real-time. Generally, it has a large angle of aperture and long range.

Time-of-flight camera

This device gives a direct distance measurement for each pixel with fast 3D recording and high frame rate. It does not require ambient light. A laser is used to emit a pulse of light and the amount of time before the reflected light is seen by a detector is measured. The accuracy of a time-of-flight 3D laser scanner depends on how precisely time is measured. Typical time-of-flight 3D laser scanners can measure the distance of $10000 \sim 100000$ points every second.

Devices mentioned above can be combined to achieve better performance and accurate results. In our case, 2 laser scanners at the front of the car a stereo pair of cameras are installed (figure 1.4).

This choice of equipment is justified because of the following reasons:

- cheapness and popularity of the devices
- availability of research materials based on this equipment
- good performance of cameras and lasers simultaneously because of their complementarities

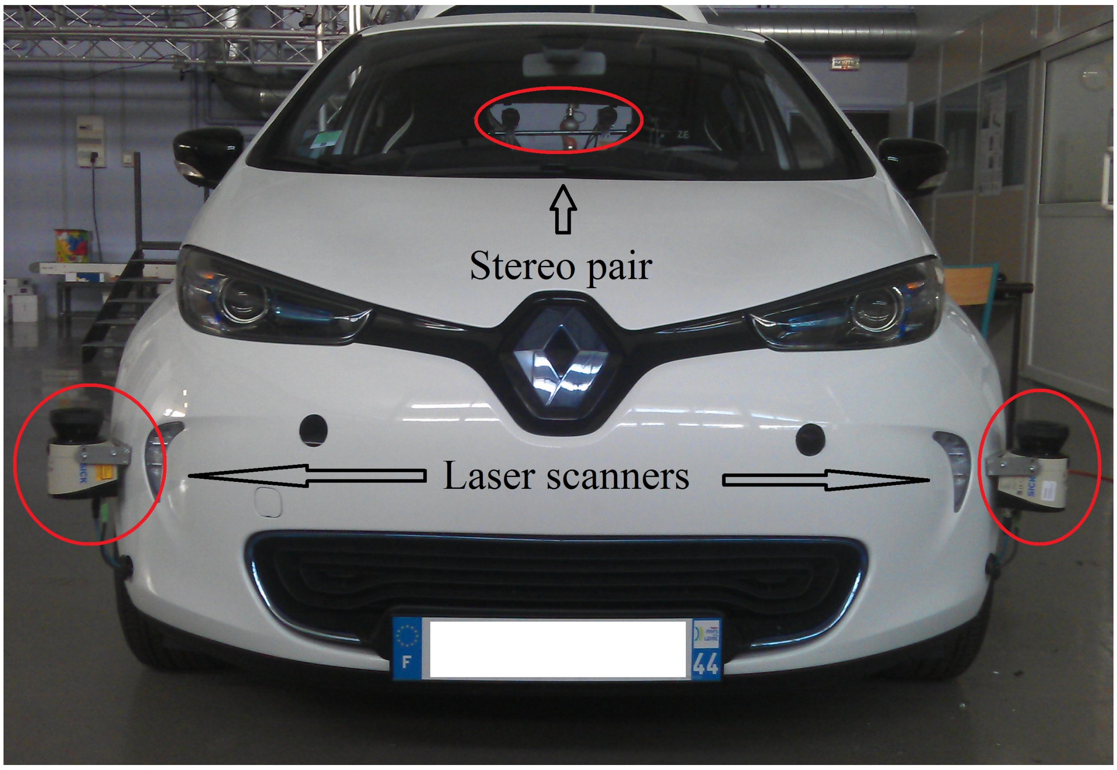

FIGURE 1.4: Renault ZOE equipped for pedestrian detection and tracking.

Our laser measurement system uses LMS151 laser sensors which detect objects up to 50 meters away with angular resolution 0.5°/0.25° (adjustable) and scanning angle $< 270°$ Baumer TXG12 high-resolution cameras are planned to be used.

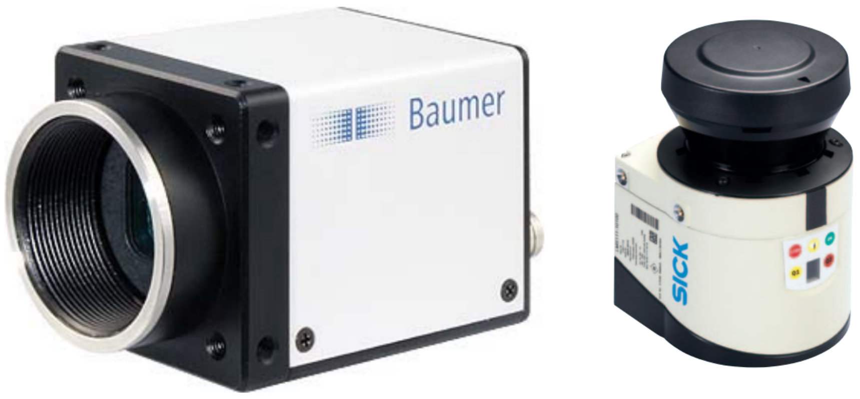

FIGURE 1.5: Camera and laser scanner installed into the car.

1.4 Software/Hardware

To carry out the project C++ programming language was used. Small scripts to analyze data, to draw plots or to calibrate the transformation matrices were written in Python.

1.4.1 OpenCV

OpenCV is a popular open source computer vision library written in C/C++/Python. All of the primitive and many complex image processing algorithms and data structures are provided. OpenCV libraries are written in a highly efficient manner to avoid reinventing the wheel.

A CUDA-based GPU interface has been in progress since September 2010 and currently many of the simple image processing algorithms have GPU-versions, as well as some of the complex algorithms including Pedestrian Detection algorithm based on Histogram of Oriented Gradients. Many useful functions of OpenCV are used in this work, such as:

- Gradient computations
- Image resize, reshape, conversions
- AdaBoost classifier
- Clusterization
- Different feature extractors, descriptors
- Feature matchers

1.4.2 Robot Operating System

ROS (Robot Operating System) is rapidly becoming a de facto standard for writing interoperable and reusable robot software. It provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license. It has built-in components to handle the output of different types of sensors, such as cameras, lasers, actuators, etc.

ROS uses so-called *nodes* (which is no more than executable within a ROS package) to perform tasks. Different nodes can communicate with each other using a ROS client library. In order to have communications **roscore** must be running. It is a kind of "server" which allows to establish communications using *Topics*. A topic is published by a node and other nodes can subscribe to this topic and receive messages. *Messages* are simple data structures which comprise primitive types, like integer, floating point, boolean and others. Nodes can also provide or use a *Service*.

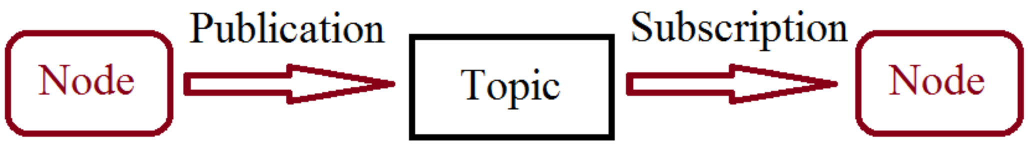

Figure 1.6: Simple ROS communication.

Another feature of ROS is the possibility of creating *.launch*-files that can run multiple ROS nodes, set their parameters in order to simplify tasks and merge logically related components. In the project, we usually used launch files to play *.bag*-files start our nodes, run visualization. *.bag*-files are recorded data collected in real-time that can be replayed later on simulating the same conditions. Playing a bag file is generally no different from having ROS nodes send the same data.

1.4.3 Effibox

The vehicle is equipped with a modern Computer with the Linux OS. To record the data in real time and replay the sequence with a proper timing and synchronization Effibox device is used. Effibox is more than a device, the Effibox software can be installed into any computer with Ubuntu-32. It has open C++ API and supports multi-threading. Effibox is compatible with many modern sensors, including cameras, range finders, GPS, etc. During the project several data acquisitions were made and afterwards corresponding *.bag*-files were recorded.

FIGURE 1.7: Effibox computer installed in the vehicle.

1.5 General overview of detectors

Almost all of the Pedestrian Detection algorithms use Machine Learning techniques to some extent. In our case, this is supervised learning, when labeled data is used to train a classifier with 2 classes of Pedestrian and Non-Pedestrian.

The sliding window method is used in almost all applications of Pedestrian Detection. This method is based on a window with fixed aspect ratio. The whole image is scanned by this window with a certain step and scale. A response is computed by a classifier and best fitting ones are selected to be pedestrians (Figure 1.8).

Thorough comparison and analysis of various detection algorithms is given in [4]. The Caltech Pedestrian Dataset is used to evaluate them. At the beginning of the project, the choices were made according to this article. In this article, 16 pedestrian detectors were evaluated. Modification or retraining of those detectors was avoided to ensure each method was optimized by its authors. Images have a resolution of 640×480. 350000 bounding boxes were annotated around 2300 unique pedestrians. Occluded pedestrians and large groups are annotated correspondingly.

The aspect ratio of detections was standardized to the value $= 0.41$, which represents the log-mean aspect ratio of the Caltech dataset.

The main performance evaluation criterion used in this paper is *log-average miss rate*. Different miss rates can be obtained by varying the threshold on detection

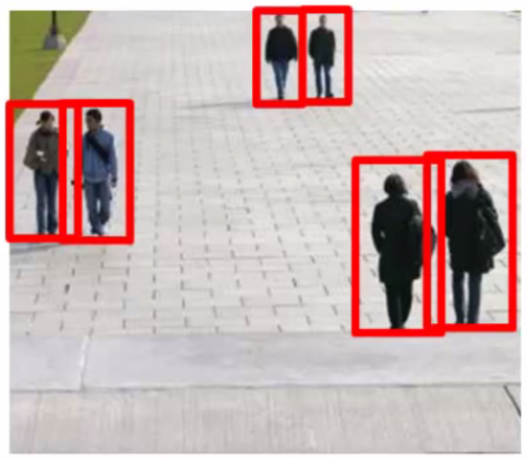

FIGURE 1.8: Detections with sliding window

confidence. Log-average miss rate is computed by averaging miss rate at nine False Positive Per Image (FPPI) rates evenly spaced in log-space in the range 10^{-2} to 10^{0}. This number is used to represent the entire curve with a single reference value and to estimate precision and recall. Overall performance is shown on figure 1.9.

The second criterion is the running time of a single detection, which is given by its inverse, in Frames Per Second (FPS) and varies from 0.004 up to 34.286.

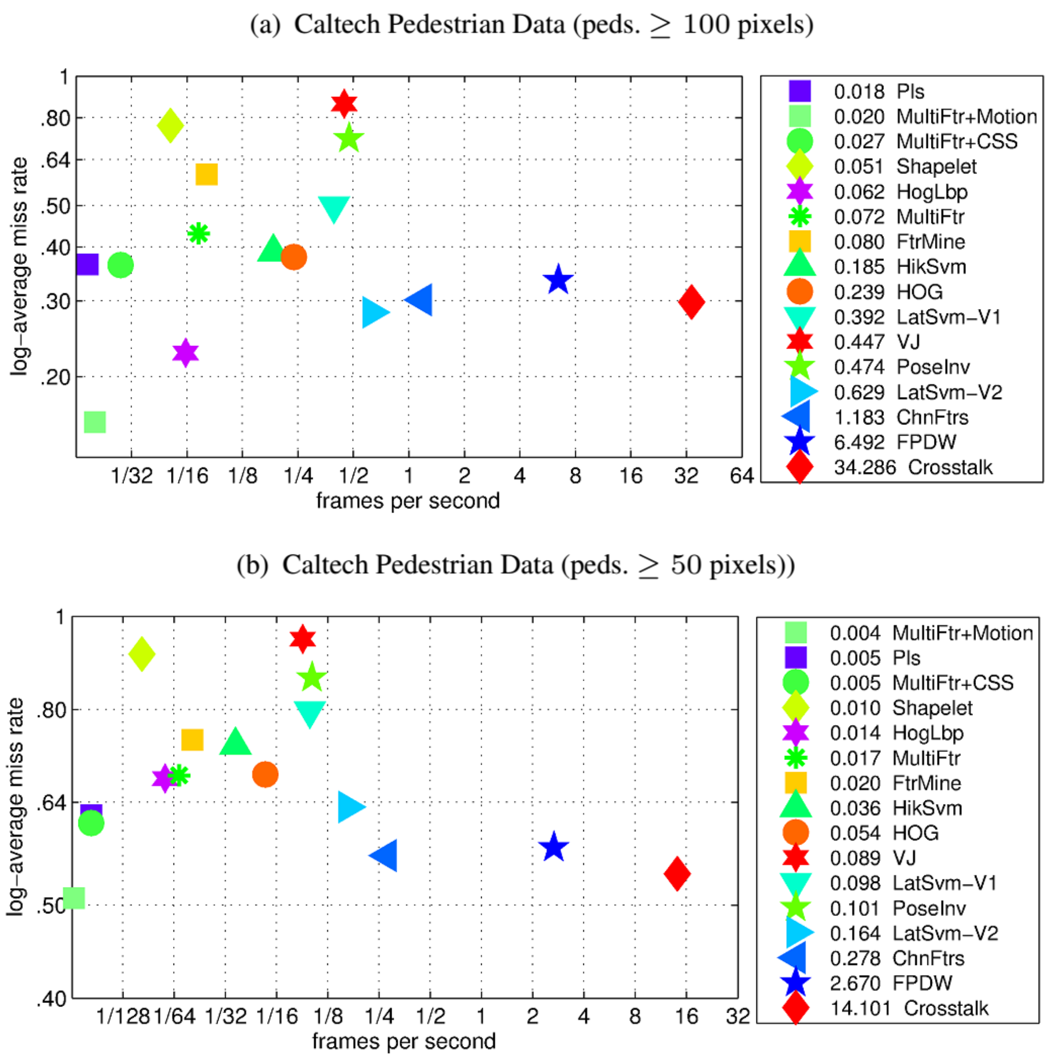

FIGURE 1.9: Log-average miss rate versus the runtime of each detector on 640×480 images from the Caltech Pedestrian Dataset. Run times of all detectors are normalized to the rate of a single modern machine, hence all times are directly comparable.(Note that the VJ implementation used did not utilize scale invariance and hence its slow speed). Legends are ordered by detector speed, reported in frames per second (fps) [4].

Chapter 2

Features

In this chapter we will discuss different feature types that were used in the project to perform the learning and detection.

Generally, there are different types of features that can be extracted:

- Gradient Histogram
- Gradients
- Intensity
- Color
- Texture
- Self-similarity
- Motion
- etc.

Many algorithms use one or several of these feature types. Nearly all modern detectors use gradient histograms to some extent. The best performing detectors combine different features to achieve better performance.

2.1 HAAR-like features

Haar-like features are used widely in object recognition. Haar-like features are based on Haar wavelets (figure 2.1) which use the idea of integral images in order to achieve constant time complexity [5]. Integral images are computed as follows:

$$I(x,y) = I(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$

Once an integral image is computed, any inner rectangle can be computed using 4 references into the integral image array.

A Haar-like feature is computed with respect to a specific location in a detection window with a specific mask, summing up the intensities in each region and calculating the difference between these sums. To further speed up the detection, a cascade of features may be used in order to discard non-pedestrians at each layer by introducing a degenerate decision tree of increasingly complex layers (Figure 2.2).

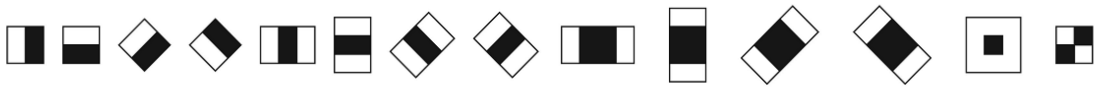

FIGURE 2.1: Haar wavelets. Black and white areas denote negative and positive weights, respectively [6].

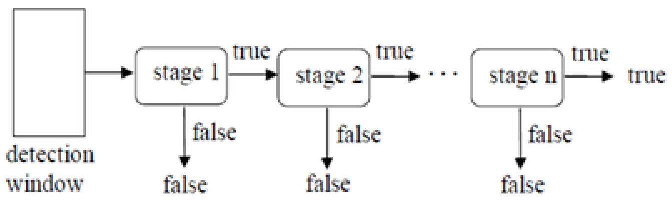

FIGURE 2.2: Classifier cascade

HAAR-like features are vastly used in Pedestrian Detection as weak classifiers, linear combinations of which, carefully selected and weighted by a learning algorithm (*e.g.* AdaBoost), can result in strong classifiers with robust performance [5]

2.2 Histogram of Oriented Gradients

Dalal and Triggs popularized HOG features by showing significant improvements over intensity based features [7]. The most popular detector is based on those features and SVM-classifier. Roughly, extraction of the HOG-descriptor consists of the following steps:

Gamma/Color Normalization These normalizations have only a modest effect on performance, because subsequent descriptor normalization achieves similar results.

Gradient Computation The simplest scheme turns out to be the best [7] with the masks $[-1, 0, 1]$ and $[-1, 0, 1]^T$.

Spatial/Orientation Binning Each pixel calculates a weighted vote for an edge orientation histogram channel based on the gradient centered on it, the votes are accumulated into bins which are evenly distributed over $0° - 180°$ ('unsigned' gradient). It turns out that for human, sign of the gradient does not play much of a role.

Normalization and Construction of Descriptor Blocks Due to local variations in illumination and foreground-background contrast, local contrast normalization is necessary. Descriptor blocks consist of $\varsigma \times \varsigma$ grids of $\eta \times \eta$ pixel cells each containing β orientation bins, where ς, η and β are usually 2, 8 and 9.

In [7], HOG-features were extracted for a sliding window and fed into a linear SVM (Section 3.1.1.2) classifier to achieve good performance (Figure 2.3). Using Gaussian kernel SVM increases performance by about 3%.

OpenCV has built-in version of this detector, pre-trained and easy-to-use. It is inside of the **Object Detection** module. The main function is called *detectMultiScale* which takes as input the following parameters:

- Input image
- Hit threshold for the distance between features and classifying plane.
- Window stride
- Padding
- Scale coefficient by which the image is multiplied.
- Grouping threshold to regulate the similarity.

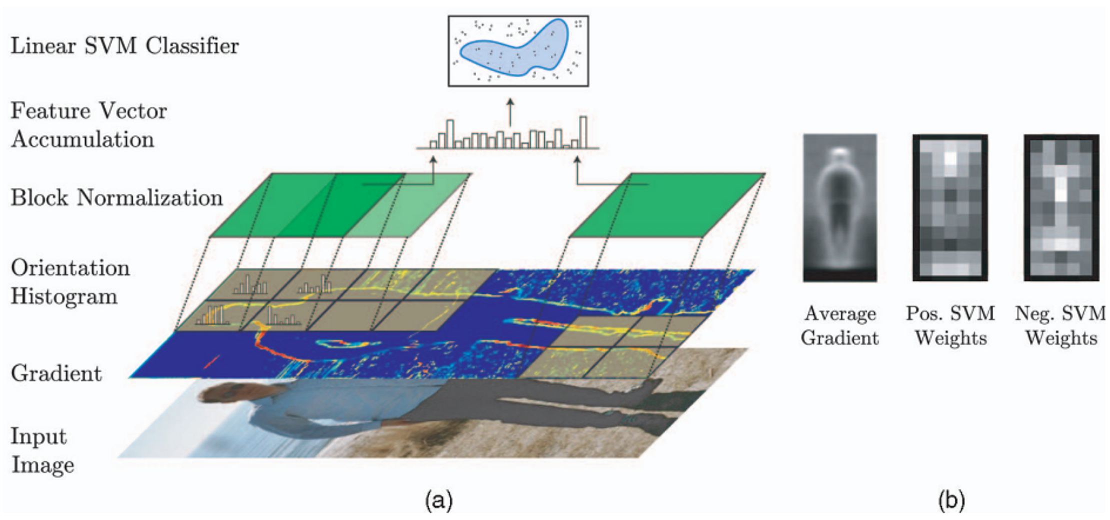

FIGURE 2.3: (a) Overview of HOG/linSVM architecture. (b) Average gradient image along with visualization of positive and negative SVM weights [6].

The function outputs the vector of all the possible locations (bounding boxes) according to input parameters. A bounding box has 4 degrees of freedom: x, y coordinates of the top left corner and its width and height. The main disadvantage is that the output confidence is not given meaning that it is binary. Either an area contains a pedestrian or not. There is no quantitative value indicating the probability. The main advantage is that the module contains GPU-versions of the functions and it can speed up the computations significantly. In the computer we used with Core i7 CPU and GeForce GTX 780 Ti the speed-up is about hundred times giving more than 50 FPS for an image resolution 640x480.

2.3 Integral channel features

Integral channel features were introduced in 2009 [8] by P.Dollár et al. They were specifically designed to detect pedestrians. The main advantage of those features is that they are computed efficiently using integral images and channel representation. Those features are computed using linear and non-linear transformations of the input image. They not only outperform other features such as HoG, but also naturally integrate different types of information, have fewer parameters to tune, allow for more accurate spatial localization and can be used efficiently with cascade classifiers. Here are the most common features used in the literature:

- Grayscale images

- LUV color channels
- Gradient magnitude
- Gradient magnitude quantized by orientation

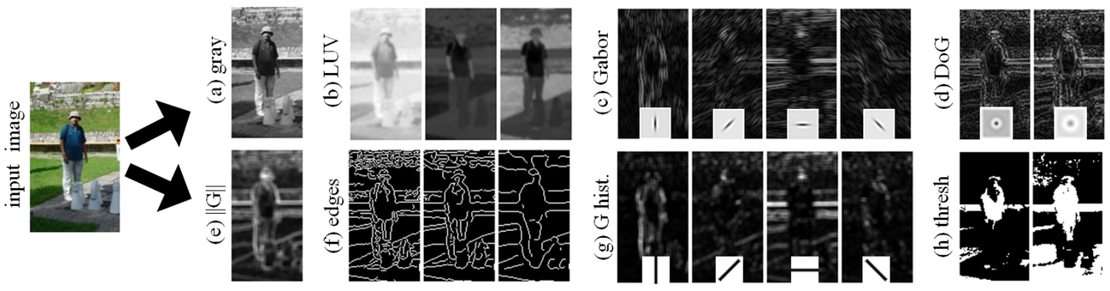

FIGURE 2.4: Multiple registered image channels are computed using various transformations of the input image [8].

2.3.1 Channels

The notion of channels emerged with the early days of computer vision. The orthogonal spatial derivative kernels can give a response, which combined nonlinearly to obtain a measure of edge strength and orientation, could be thought as the ur-channels. Viola and Jones proposed an object detection with boosting using HAAR-wavelets and integral images. With the increasing computing power it became less expensive to compute integral images on top of channels, thus naturally extending the representation.

For a given input image I a corresponding channel is a registered map of the original image, where the output pixels are computed from corresponding patches of input pixels thus preserving overall image layout. A trivial channel could be $C = I$ for a grayscale image. Or, each color channel could be considered as a channel. If we denote channel computation function as Ω such as $C = \Omega(I)$, then in the case of using integral channels we can say that channels are translationally invariant, such that for given I and I', $C = \Omega(I)$ and $C' = \Omega(I')$ must be related by the same translation.

There are many different types of channels that can be used in detection. Various filters, thresholding operations, different orders of derivatives are some of the examples of those channels. Types of channels that give best detection results may be counterintuitive. Fortunately, this problem was carefully analyzed.

In the original paper [8] authors evaluated different combinations of channels and the best performing combination turned out to be LUV-color channels, magnitude of the gradient of an image and 6 channels representing orientation of the gradient binned into different images according to an orientation.

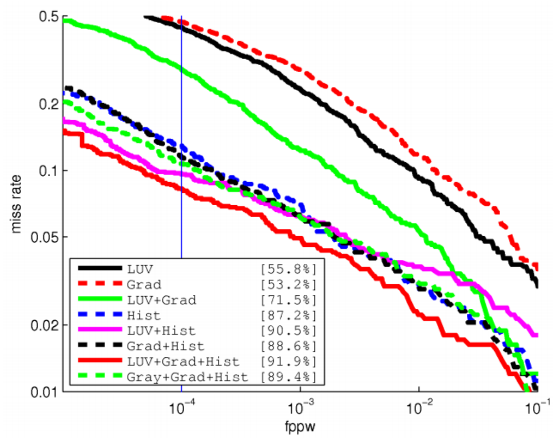

FIGURE 2.5: Evaluation of different combinations of channel features

However, in our case the combination Gray + Grad + Hist is used because of the black-and-white cameras. The detection rate is slightly less than LUV + Grad + Hist combination. This combination can easily be modified in the code later if required. Most channel types can be computed with a few lines of code given standard image processing tools. An example of the channels image is displayed on the fig. 2.6.

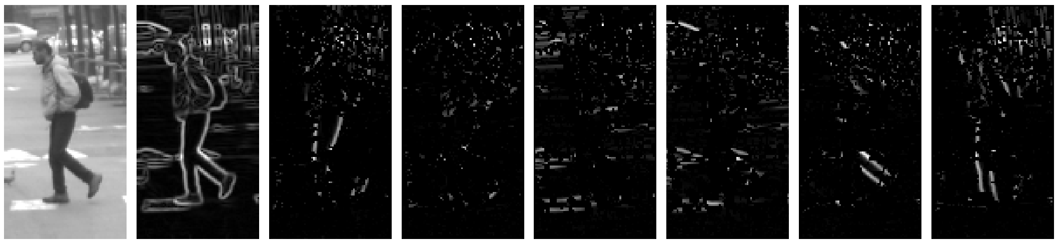

FIGURE 2.6: 8 channels of a given image

2.3.1.1 Gray image

The first channel type is just a gray image. Pre-smoothing with $\sigma = sqrt((2r+1)/4)$ is applied to eliminate high-frequency noise, where $r = 1$ is a radius parameter.

2.3.1.2 Gradient magnitude

First, Sobel derivatives are computed for horizontal and vertical directions. Then, those values squared and summed up. Finally, the magnitude is square root of the obtained value.

2.3.1.3 6 gradient orientation channels

An orientation is binned according to a given angle. It turns out that 6 unsigned orientation bins are sufficient to achieve good performance in the detection of objects (fig. 2.7). Further performance improvement is not worth increasing the number of channels. Each bin, in this case is a separate channel. An index of the channel (bin) is computed as follows:

$$\alpha = atan2(\frac{\partial I}{\partial y}, \frac{\partial I}{\partial x}) + \pi$$

$$index = int(\frac{N_{or}\alpha}{\pi})modN_{or}$$

Where α is computed orientation from 0 to 2π. $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial x}$ - derivatives, computed applying Sobel-operators. N_{or} - number of orientations (in our case, 6) Therefore, index can take values from 0 to 5.

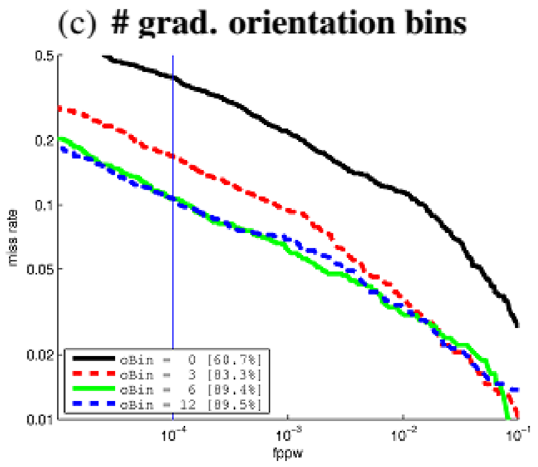

FIGURE 2.7: Detection performance according to number of orientation bins

2.3.2 Integral images

Having computed channels of an image, in order to speed up feature extraction it is necessary to compute integral image for each channel. This step makes easier to evaluate local sums given a rectangular box. Integral image is an image any point of which contains sum of all values of the points above and left to it.

$$I(x,y) = \sum_{\substack{x' \leq x \\ y' \leq y}} i(x', y')$$

It can be constructed efficiently in a single pass over the image. Once it is computed, any rectangle can be accomplished in constant time with just for array references. Specifically, having A = (x_0, y_1), B = (x_1, y_1), C = (x_1, y_0), D = (x_0, y_0), the sum of the rectangle spanned by A, B, C and D is just (fig. 2.8)

$$\sum_{\substack{x_0 < x \leq x_1 \\ y_0 < y \leq y_1}} i(x,y) = I(C) + I(A) - I(B) - I(D)$$

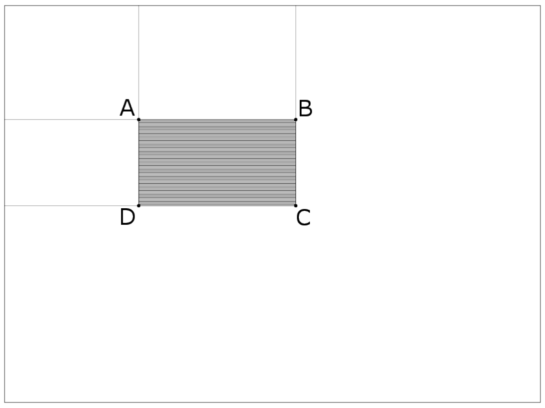

Figure 2.8: Finding the sum of a rectangular area

2.3.3 Pool of features

Once integral channel images are computed, features can be extracted. This is done by scanning the whole image with a certain step and scale with a bounding box. In our case, the image is rescaled and bounding box with a fixed dimensions $64x128$ is used to extract features. For a given BB predefined pool of features is extracted. This pool of features is generated randomly once. A certain weight is assigned to a feature according to its response during the training process. In this case, a feature has 5 characteristics:

- Index of the channel
- x-coordinate the rectangle
- y-coordinate the rectangle
- Width of the rectangle
- Height of the rectangle

Rectangle characteristics are generated with respect to the bounding box. For each generated feature the local sum is computed and stored in a feature vector, which then is fed into a classifier. An example of a possible reduced pool of features is represented on the fig. 2.9.

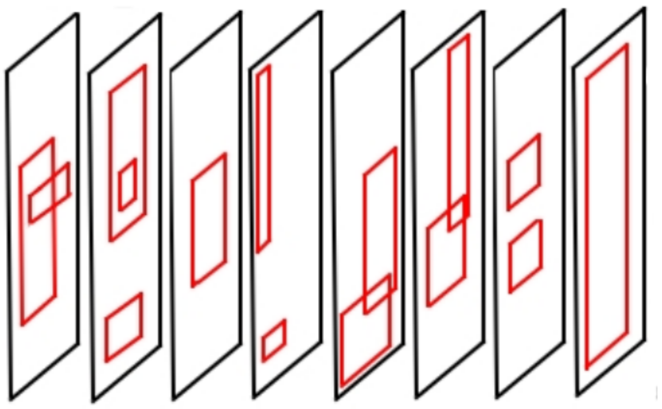

FIGURE 2.9: Example of a reduced pool of rectangle features

2.3.4 Comparison with HOG+SVM

The authors claim that the integral channels detector combined with AdaBoost gives better results than HOG combined with Support Vector Machines. Detection performance of the two classifiers are given on figure 2.10.

This might seem surprising at the beginning. Because HOG is a sophisticated algorithm and both HOG and ChnFtrs use the idea of gradient orientations and magnitude. However, carefully examined, the key difference can be noticed. Whereas

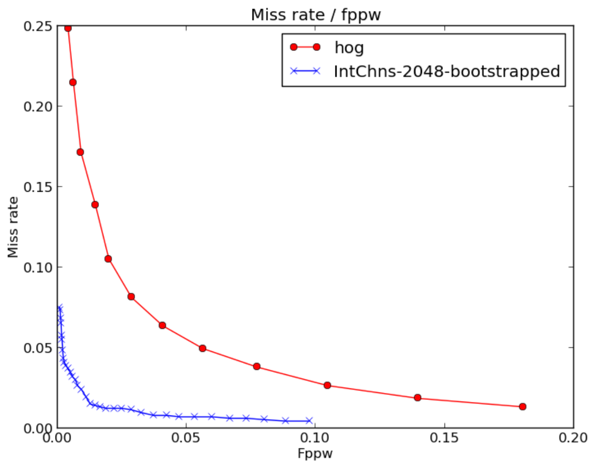

FIGURE 2.10: Detection performance of HOG based classifier and Integral Channels classifier with 2048 features.

Dalal and Triggs [7] place the HOG cells uniformly, the ChnFtrs detector learns where to put features in order to maximize the discrimination.

2.4 Principal component features

During the bibliographical research a preliminary choice was made in favor of integral channel features. Those features are relatively easy to implement. Good results are achieved by generating quite a large pool of features. However, those features are generated randomly, not by computing carefully. That is why, during the research, another type of features was designed in order to compare the performance.

The main idea was to compute features of pedestrians by taking low frequency information of the pedestrian set. These low level components are called principal components. Here are the main steps performed to generate features on the dataset of positive pedestrian images:

- All pixels of an image are put into one vector, which is then normalized according to mean and standard deviation of its values as follows:

$$x_i = \frac{x_i - \mu}{\sigma}$$

- The covariance matrix is obtained as $x \cdot x^T$

- All those matrices are summed up to one accumulator matrix and values normalized dividing by the number of vectors.
- Singular Value Decomposition (SVD) is performed on this matrix.
- Σ-matrix is analyzed to retain a certain proportion of variance. In our case, 95 % was taken. In order to do that, a certain number of components k must be chosen as principal. For given k:

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.05$$

  Where S_{ii} is a corresponding element of Σ-matrix.

- When, the number of principal components is determined, we can reduce the U-matrix:

$$U_{reduced} = U(:, 1 : k)$$

- Finally, $U_{reduced}$ is used to obtain a feature vector z:

$$z = U'_{reduced} \cdot x$$

To verify that obtained feature matrix is meaningful, we can visualize the columns of this matrix and see the "eigen-pedestrians". To do that, a vector of the matrix must be reshaped to have original image dimensions ($64x128$). The results are shown on the figure 2.11.

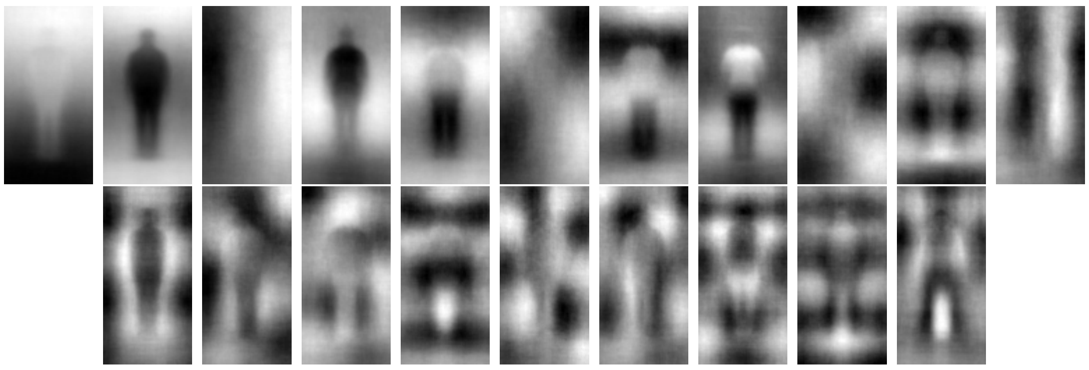

FIGURE 2.11: First 20 pedestrian eigenvectors

We can see that obtained eigenvectors resemble different shapes of pedestrians. The main hypothesis was to have a basis of vectors which can separate two classes of a pedestrian and non-pedestrian while decomposing original image onto this basis. However, obtained results were not as good as we would like to have. A few different tries were done. In many cases, the detector could not separate classes well, tending to overfit the data for the testing set. Generalization was not achieved. One possible explanation may be the fact that we performed an SVD on raw image, without extracting any other type of information, like gradients, edges, etc. Another explanation might be the fact that AdaBoost classifier is not very suitable for this type of training. In order not to spend more time for the features, it was decided to use the solution which was already well explained. Therefore, HOG and IntChns were the choices to explore.

Chapter 3

Detection

After the discussion of the different types of features in this chapter we will talk about the classifier, training and detection algorithms.

3.1 Training

In this section we will discuss which classifier is used and why. How the training process is done.

3.1.1 The choice of a classifier

The main types of modern classifiers are the followings:

- Neural Networks
- SVM
- AdaBoost

3.1.1.1 Neural Networks

One of the detectors used in modern algorithms is Neural Networks. The main advantage of this type of classifier is that it can fit highly non-linear functions.

However, it should be tuned carefully (number of layers, neurons, etc.) and trained more time compared to others. Very often, raw data is used without computation of features. Neural networks can be designed with 2 output neurons which give the probability that the current image contains a pedestrian or non-pedestrian (Figure 3.1).

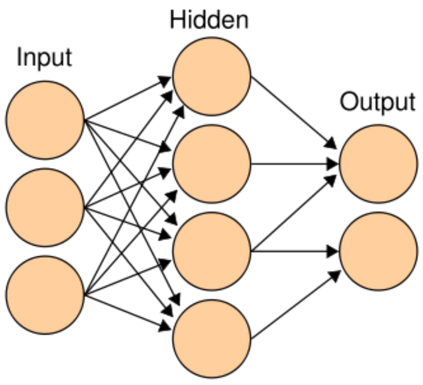

FIGURE 3.1: Neural networks

3.1.1.2 Support Vector Machines

SVMs have evolved into a powerful tool to solve pattern classification problems. Neural networks minimize an artificial error, whereas SVMs maximize the margin of linear boundaries (hyperplanes) (Figure 3.2). Thus, maximum separation between object classes is achieved. SVMs can have nonlinear kernels, polynomial or radial, as implicit mapping of the samples into a higher dimensional space.

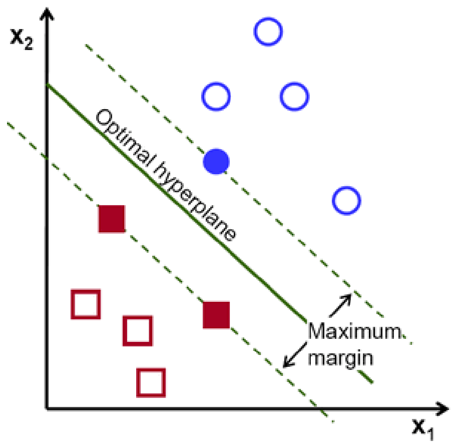

FIGURE 3.2: Optimal hyperplane

3.1.1.3 AdaBoost

AdaBoost is well suited to our problem, especially to ChnFtrs features because it can handle thousands of features per sample, it is fairly resistant to over-fitting and can accurately select features. AdaBoost constructs strong classifiers iteratively by linearly combining weak ones. It is often used with image intensities. It can be improved by constructing cascades of classifiers to get rid of non-pedestrians at on early stage of detection. AdaBoost can be tuned with a small set of parameters. It can be combined with other classifiers as a pre-classifier. It can be used with different features, but is mainly used with Haar-like or combined features, which is our case. Most of the fast algorithms are based on this classifier.

A short algorithm of the AdaBoost can be formulated as follows [9]:

- Given (x_1, y_1), ... ,(x_m, y_m) where $x_i \in X, y_i \in \{-1, +1\}$
  Initialize $D_1(i) = 1/m$
- For $t = 1, ..., T$:
  - Train weak learner using Distribution D_t.
  - Compute error of a hypothesis h_t:

    $$\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$$

    .
  - Choose $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$.
  - Update
    $$D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$
    where Z_t is a normalization factor for D_{t+1}
- Output the final hypothesis:

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

In this algorithm, α measures the importance that is assigned to h_t. It gets larger, as ϵ gets smaller. At each iteration, the weights of misclassified examples are increased and the weights of correctly classified ones are decreased. Thus, the weights tend to concentrate on ”hard” examples. The *final hypothesis* H is a weighted combination of the T weak classifiers.

3.1.2 Learning

The training procedure consists of two parts:

Feature extraction and output of training samples A big sample matrix is constructed, where columns represent features and rows represent samples. One column is dedicated for the class of a sample. In our case, ”Pedestrian” or ”Non-pedestrian”. For simplicity and convenience, this matrix is written into the *.csv*-file.

Training the classifier using these data The file is read by OpenCV AdaBoost classifier. At the end of the training, the final classifier is output, which is usually saved into the *.xml*-file. One of the key parameters of the classifier is the number of *weak classifiers.* A good choice is to set this number to be twice as small as the number of features.

3.1.2.1 Dataset

Pedestrians appear in very complex scenarios that are difficult or impossible to define with a specific model. Therefore, they are learned implicitly. In order for computer to learn how to discriminate pedestrians from non-pedestrians, it is necessary to collect correctly labeled data (ground truth) and train a classifier. In order to have a robust classifier the data must contain many different positive and negative samples in various conditions. Collecting the data and labeling it in the same manner is very exhausting task. Fortunately, there are many publicly available datasets.

One of the popular Pedestrian datasets is INRIA Pedestrian Dataset. This dataset was initially acquired [7] to train the SVM with HOG features. Currently, many other detectors use this dataset.

This dataset is divided into 2 parts: Training and Testing parts. Each of the sets contain cropped positive pedestrian images (2416 + 1132) and images with no pedestrians (1218 + 462). From the latter, many negative samples can be cropped

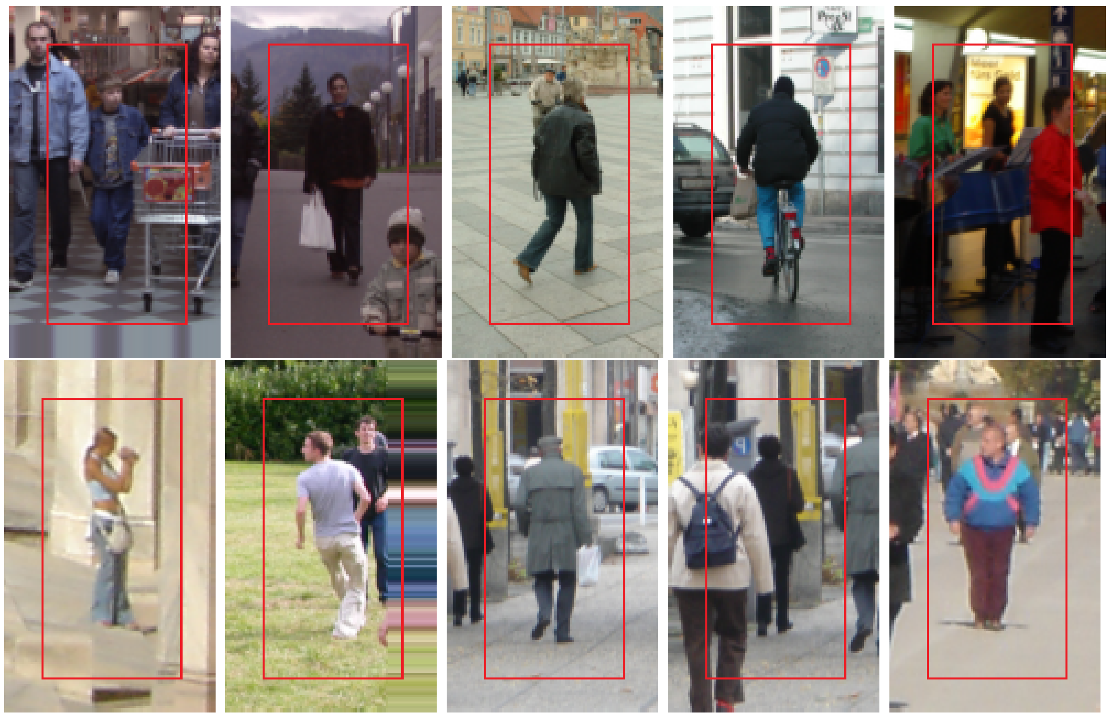
FIGURE 3.3: Pedestrian images from INRIA training Dataset. A margin of 16 pixels around each side is removed.

FIGURE 3.4: Examples of images without pedestrians.

with a certain step and window size. Therefore, the number of positive samples is significantly fewer than the number of negative ones. The training methodology was as follows:

Training The training of the classifier is performed on 2500 positive samples and 15000 negative samples. Usually this step takes 30-180 minutes depending of the number of features.

Bootstrapping The second step is to run the trained classifier. Usually, at this step, there are many false positives. Then, to improve its robustness, a classifier is run on negative samples which are generated in a different way from the same images (not the same negative samples as in the first step). The ones which gave false positives are written into the file, containing initial training matrix (usually around 4000 samples more). After retraining the classifier, usually the result is much better, false positive rate decreases.

Testing The last point is running the classifier on the testing set to validate the training. At this step, precision and recall of the classifier is analyzed.

3.2 Classification

In this section we will talk what is done after the training process, assuming that we already have a ready-to-use classifier.

3.2.1 Detection on a single image

The first thing that was implemented is the detection of pedestrians on a single image. Firstly, the straightforward approach was implemented which uses fixed steps and scales. Roughly, the whole algorithm can be described as follows:

- **For a certain scale do**

  Pre-smoothing An input image is smoothed, applying a Gaussian kernel.

  Integral image computation Integral channels are computed for the whole image.

  Step Scan image with a certain step with a Bounding Box and determine if there is a pedestrian.

  NMS Perform Non-Maximum suppression to find the best BB of a pedestrian in some neighborhood.

- Perform clusterization of different scales to avoid the detection of the same pedestrian (figure 3.5).

- Output all pedestrian rectangles.

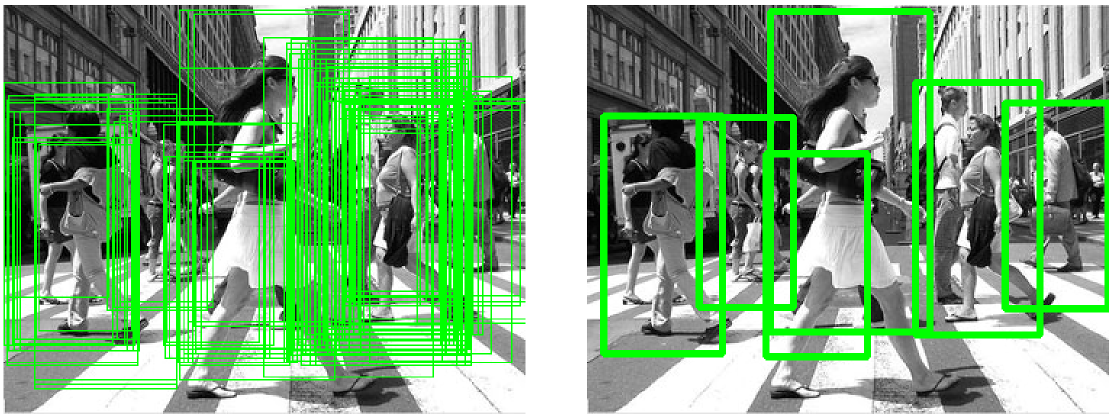

FIGURE 3.5: The same pedestrian can be detected at different scales and spaces. The result of clustering.

In order to speed up the detection, the idea of variable step scanning was introduced. The main idea is to use 2 different thresholds to start scanning with a small (which supposedly is the same as in the case of fixed step) or large step. These thresholds can be tuned in a way that there is no loss in the detection quality, but there is a big gain in the detection speed.

Figure 3.6 illustrates how the algorithm works. Here, for the sake of simplicity, we assume that a cell represents a scan with a Bounding Box and neighbor cells highly correlate because they have common area. Thus, one cell represents a smallest scan step. By default, an image is scanned with the step of dark cells. Then when the response is computed there might be 3 cases:

R > Excitation threshold If the response is high, such that there is a high probability that there might be a pedestrian, a neighborhood of this cell is scanned to find the maximum response to localize the pedestrian accurately. Even, in this case, not all the neighborhood is scanned if possible. For example, if a cell to the left to the dark cell gives lower response, there is no need to evaluate 2 other left corner cells.

R < Inhibition threshold If the response is low enough, the next dark cell is skipped, so in this case the step is equal to 6 cells.

Otherwise The step (3 cells) is not changed, the next dark cell is evaluated.

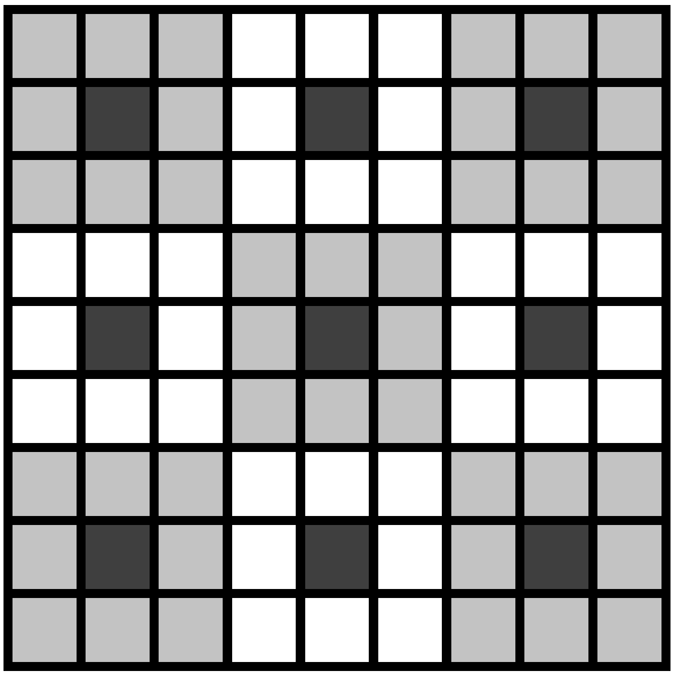

FIGURE 3.6: Dark cells are scanned by default and depending on the response, the step may be changed to be one cell, 3 cells or 6 cells.

3.2.2 Detection on a stereo pair

With a single camera it is difficult to estimate the depth of objects. However, when a basic detection algorithm on a single image is ready to use it can be combined with the corresponding image of the second camera to estimate the depth. This is done as follows:

- All 2D rectangles of pedestrians on the left image are found.
- Features are extracted from these rectangles and the second image.
- Stereo matching is performed for each pedestrian image
- Disparities are computed for matched keypoints. A simple ransac-algorithm is run to find the best disparity which corresponds to most of the keypoints, small disparities are discarded as outliers.
- Finally, average disparity is computed from inliers and then its corresponding depth is calculated by stereo triangulation.

On the figure 3.7 pedestrian features are matched with features on the right image. Assuming that we have a good matching and sufficient number of features, 3D-position of a pedestrian can be estimated with the precision of tens of centimeters.

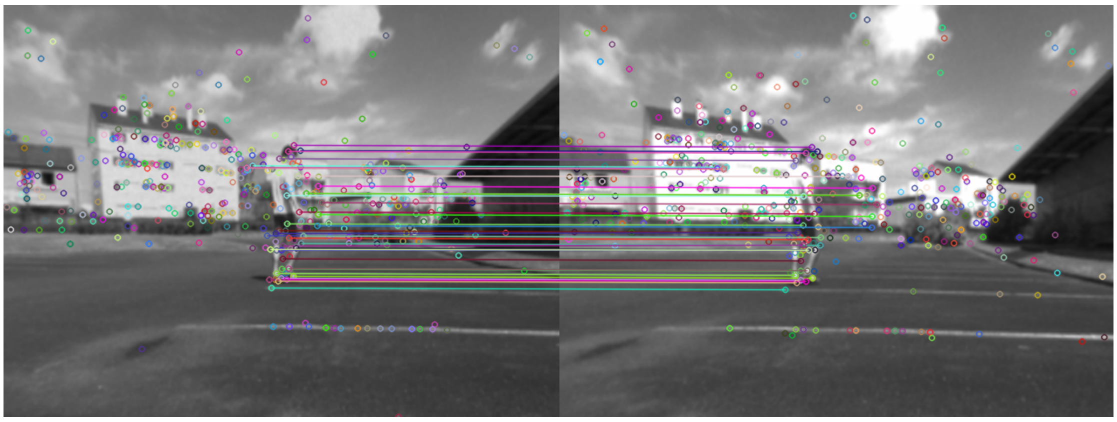

FIGURE 3.7: Computed depth is equal to 5.08 meters.

Generally, results are quite satisfactory. Sometimes, there are jumps of distances in consecutive frames.

3.2.3 Detection with the use of laser scanner

Using only vision to detect pedestrians is a hard task to perform, especially when it comes to precision and time constraints simultaneously. This is why, many modern applications use a combination of sensors to perform an object detection. In our case, we use laser scanners. Characteristics of our laser scanners:

- Scanning range of 50 m.
- Scanning angle: 270°
- Angular resolution: 0.5°

So, one laser scanner generate 540-vector of ranges for each 0.5°. In our vehicle two laser scanners can be combined to have a wide view of the environment.

3.2.3.1 Laser

The detection of pedestrians is possible using only laser sensors with the use of motion information filtering.

The detection of pedestrians using range images consists of the following steps [10]:

Segmentation

Ego-motion-estimation It is possible to compute relative velocities of the objects with respect to the vehicle. However, if the vehicle motion itself is known, absolute velocities of the objects can be obtained.

Object-Tracking Comparing the segment parameters of the current scan with the predicted one it is possible to track objects (for example, using Kalman filtering). Different association techniques can be used, such as *nearest neighbor*.

Object-Classification An object is classified as pedestrian only if it is moving or has moved previously during the tracking. Otherwise, it is classified as non-pedestrian. Even in this case, the information is useful, because if this object is on the predicted path of the vehicle it is rated as dangerous.

The range of certain possible velocities can be set to identify objects more robustly. However, to achieve very good classification and cope with the diversity of outdoor situations, integration with other types of sensors is necessary.

3.2.3.2 Sensor Fusion

There are different types of sensor fusion available:

1. Sensors are integrated independently with identically distributed assumptions
2. Laser scanner segmentation method is used to find the most likely ROIs
3. Semantic Fusion

3.2.3.3 Semantic Fusion

In 2010 [14], Semantic Fusion method was proposed as an alternative to classical approaches which could show better performance in time and detection miss rate (Figure 3.8).

The main ideas of this method are the following:

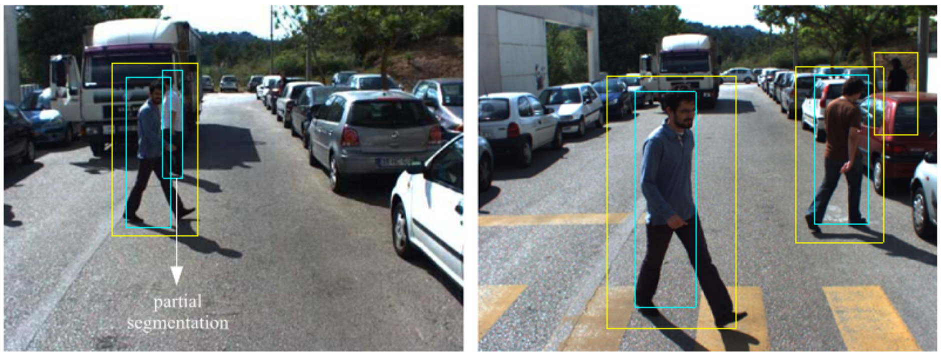

FIGURE 3.8: Two scenarios where traditional laser and vision fusion systems could fail. The light bounding box (yellow) is the result of an image classifier; the darker ones (blue) are the result of laser segmentation. [15].

Coarse-to-fine segmentation Laser segmentation and labeling is performed to obtain higher level representation of the detected points with 4 different classes: 'torso', 'arm', 'potential occlusion' and 'noise'. Each label is initially given by measuring the width of the segments. Confidence scores are assigned to those labels.

Searching objects in laser sensing space Two planes parallel to the laser plane $\pi_1 = (0, -1, 0, dist_1)^T$ and $\pi_2 = (0, 1, 0, dist_2)^T$ representing the ground and a plane which passes throughout the camera, respectively, are considered in such a way that a sliding window in laser reference frame is slid in the view-port formed by π_1 and π_2 as the top and bottom bounding planes (Figure 3.9).

Occlusion in laser and image spaces Occlusion in laser and image spaces is modeled and, depending on the distance, segmentation, labeling and confidence score, fusion is performed differently and a final decision is made.

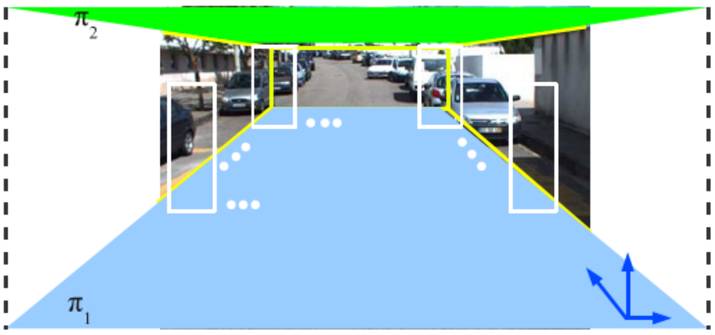

FIGURE 3.9: Sliding window in laser reference frame [14].

This method decreases false alarm rate. However, in [14], real-time working system was not provided. It can be optimized by parallelization or code optimization.

3.2.3.4 Fast fusion with ROI

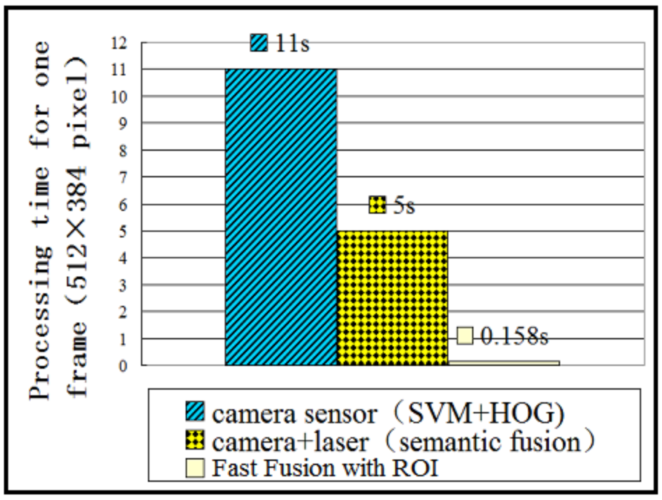

FIGURE 3.10: Average processing time of the PDS with and without the sensor fusion [15].

In [15], fast detection with laser and image fusion was proposed. Authors compare it with semantic fusion (Section 3.2.3.3) and obtain significantly faster results (figure 3.10). Therefore, this method was chosen to be implemented. Our version is implemented not in the same manner as in [15], however, it works in real time for an image with much larger resolution (966×1296). Details are given further in this chapter. The laser scanner and camera must be calibrated and then relative transformation matrix must be computed in order to project the ROI onto image-space.

3.2.3.5 Calibration

In order to perform a sensor fusion it is necessary to obtain transformation matrices between sensors. In our case, we have to compute 2 main transformation matrices: between laser sensors and between a laser scanner and a stereo pair.

3.2.3.6 Lasers transformation

The first task after the installation of laser scanners was to compute the transformation matrix between them. There are three variables that must be found in order to have a full transformation matrix assuming that laser sensor planes are the same 3.11. In order to compute it, the calibration board in front of the car was used and least squares iterative refinement was applied. Positions of the left and right points of the calibration board are found from the data in each of the frames of lasers.

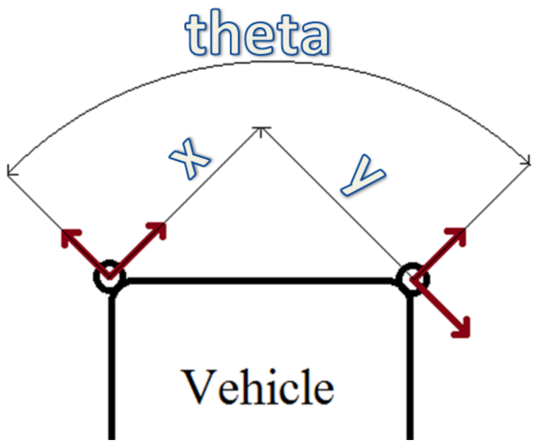

Figure 3.11: Laser scanners in front of the car.

In order to write a least square problem, positions of those points in one frame must be written with respect to another one (fig 3.12).

$$P = \begin{bmatrix} x + d_2 \cdot cos(\theta + \alpha_2) \\ y + d_2 \cdot sin(\theta + \alpha_2) \end{bmatrix} = \begin{bmatrix} d_1 \cdot cos(\alpha_1) \\ d_1 \cdot sin(\alpha_1) \end{bmatrix}$$

Here, we have 2 equations, but 3 unknowns, that's why computation is done for 2 points having 4 equations. The error between the expressions for the same point must be zero:

$$\Delta = \begin{bmatrix} x + d_2 \cdot cos(\theta + \alpha_2) \\ y + d_2 \cdot sin(\theta + \alpha_2) \end{bmatrix} - \begin{bmatrix} d_1 \cdot cos(\alpha_1) \\ d_1 \cdot sin(\alpha_1) \end{bmatrix} = 0$$

Therefore, iteratively, we compute:

$$\delta\xi = J^{+} \cdot \Delta$$

$$\xi = \xi + \delta\xi$$

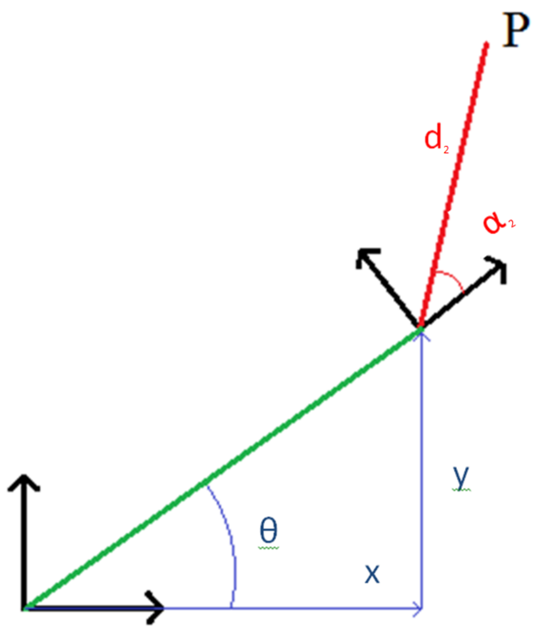

FIGURE 3.12: Position of a point P with respect to another frame.

Where $\xi = [x, y, \theta]^T$, and J is the Jacobian of the points P^1 and P^2 with respect to the ξ.

$$J = \begin{bmatrix} 1 & 0 & -d_2^1 \cdot sin(\alpha_2^1 + \theta) \\ 0 & 1 & d_2^1 \cdot cos(\alpha_2^1 + \theta) \\ 1 & 0 & -d_2^2 \cdot sin(\alpha_2^2 + \theta) \\ 0 & 1 & d_2^2 \cdot cos(\alpha_2^2 + \theta) \end{bmatrix}$$

3.2.3.7 Laser-to-camera transformation

The transformation matrix between the laser frame and the camera frame is computed as follows. The same calibration board is installed in front of the car.

$$^{laser}T_{camera} = {}^{laser}T_{leftcorner}\,{}^{leftcorner}T_{board}\,{}^{board}T_{camera}$$

Figure 3.13 illustrates the relations of those transformation matrices

$^{laser}T_{leftcorner}$ can be directly obtained using the left point of the calibration board detected on te left laser scanner.

$^{leftcorner}T_{board}$ incudes a translation along the board and two rotations around vertical axis and horizontal axis lying in the board plane. The left and right points of the board detected in the laser frame are used to compute the rotation

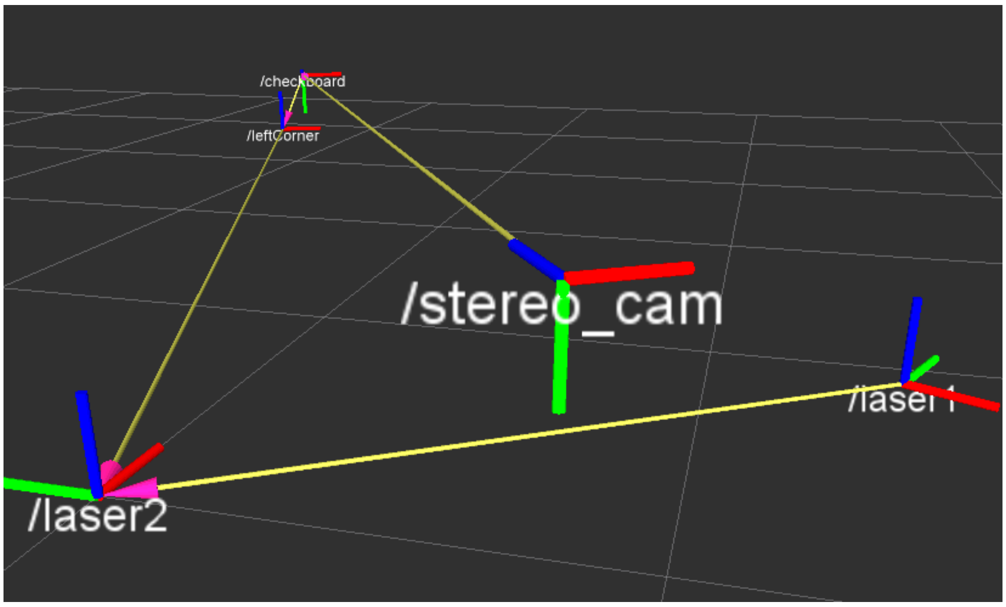

FIGURE 3.13: Calibrated transformation matrices

around vertical axis. The translation is measured from the board. And another rotation is just 90° to change the direction of z-axis (from camera).

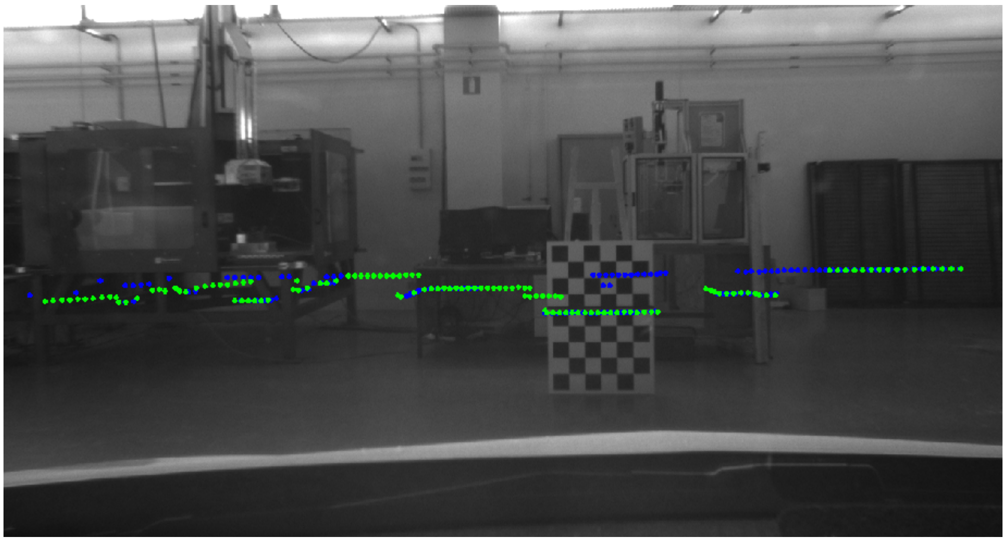

FIGURE 3.14: Projecting laser points to the image space.

${}^{board}T_{camera}$ can be obtained by using the calibration board and the stereo pair.

The result of projecting laser points onto image is shown on the figure 3.14. Green points correspond to the left laser scanner and blue ones to the right laser scanner.

3.2.3.8 Clusterization

Different methods of clusterization were tried. The simplest was to run OpenCV clusterization on 2d rectangles, where user can specify minimum possible number of rectangles, a difference threshold. However, this works only on the image, 3d information is not used, and grouping of the rectangles was not as good as expected.

Therefore, another method had to implemented.

- For each point of the laser, we have a bounding box with possible pedestrian in it. All such boxes with positive responses are extracted, mapping those responses onto corresponding 3D-points.
- They are sorted according to their response in descending order.
- Each bounding box is considered as containing a new pedestrian, if the distance between them is more than a certain threshold (for example, half a meter).

In such a way we can extract pedestrians from the image, retaining their 3D-information. In addition, this way is much easier to apply filtering.

3.2.3.9 Implementation

Since the installation and calibration of the equipment took a while, only the simplest method of sensor fusion was implemented. It uses laser scanners to detect objects and to determine regions of interest in the image space. Even with this simple approach, significant improvements were done to the work which used only vision in terms of the quality and time of the detection. Roughly, the following steps are performed:

- Detected points from the right laser sensor are transformed into points in the frame of the left sensor
- Points in laser plane are transformed into the 3D-points in camera frame.

- Those points are projected onto 2D-image and depending on the distance to the points, a bounding box of a certain size is put onto those points so that if there is a pedestrian, it is the center of this rectangle.
- Each of the rectangles is rescaled to the size of classifier bounding box size $(64x128)$.
- Clusterization is applied.
- The results are computed.

Since the laser scanners give precise points, we can accurately localize boxes on 2D-image and there is no need to scan the whole image. This simple method works in realtime with high-resolution images.

Chapter 4

Tracking

Visual tracking is a very challenging task to perform. If detection rate is high enough, one can argue that tracking is not required and methods that integrate detector outputs may suffice. However, tracking is justified in a few cases such as reducing ROI, recognizing people in movement and partial occlusion, simplification of detection, etc.

Most visual tracking methods consist of image input, feature description, context information integration, decision and modal update [11].

A visual tracking system can be classified into 2 main categories: *Target Representation Localization* and *Filtering and Data Association* [12]. The first method is very efficient for rigid object tracking (*e.g.* Contour-based tracker). The estimation of the dynamic characteristics of the traffic participants is basically a *multi-target tracking* problem. Observations are collected, potential obstacles are computed at each time step. One of the tracking methods is to maintain a list of *tracks*. The main difficulty is *Data Association*. Observation-to-track association is used to decide whether a new sensor observation corresponds to an existing track or not. Then new tracks should be added or old ones maintained. The traditional Data Association is a challenge in situations involving numerous appearances, disappearances and occlusions.

Filtering, on the other hand, can deal with complex objects along with multi-object interactions.

To simplify the tracking often a grid representation of the environment is used (usually 2-dimensional).

4.1 Kalman Filter

In order to track pedestrians Extended Kalman Filter was used. It is an efficient recursive filter that estimates the state vector of a dynamic system with partial and noisy measurements. The purpose of using Kalman Filter is to predict velocities of pedestrians, directions of movement and filter detection outputs. The first version of Kalman Filter was implemented on the image. As a state vector we had the following:

$$x = \begin{bmatrix} x_{top} & y_{top} & w & h & \dot{x}_{top} & \dot{y}_{top} & \dot{w} & \dot{h} \end{bmatrix}^T$$

Where x_{top}, y_{top}, w and h are the coordinates of top left corner, and width and height of the bounding box of a pedestrian respectively. $\dot{x}_{top}$, $\dot{y}_{top}$, $\dot{w}$, $\dot{h}$ are their corresponding time derivatives.

This filter had 4 degrees of freedom, but no 3d-information about pedestrian. It helped in very few cases as like when a pedestrian leaves a scene or is occluded for a while.

As soon as laser scanners were installed, it was redesigned. The new state vector in this case is:

$$x = \begin{bmatrix} x & z & \dot{x} & \dot{z} & \end{bmatrix}^T$$

Where x, z are the coordinates of a pedestrian in the camera frame. y is not taken into account, because it is constant with respect to the lasers. Measurements are done by laser scanners and from all the points only those are chosen which give maximum response during the detection applied by vision. $\dot{x}$ and $\dot{z}$ are the time derivatives, which are not measured. Therefore, the observation matrix is as follows:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

The state transition matrix is written as follows:

$$F = \begin{bmatrix} I_2 & \Delta t \, I_2 \\ 0_2 & I_2 \end{bmatrix}$$

Since we cannot control pedestrians, motion vector $u = 0_2$.

The following equations are applied to update the state vector:

- Prediction

  This step is performed at each iteration.

$$x = F \cdot x + u \tag{4.1}$$

$$P = F \cdot P \cdot F^T + Q_\alpha \tag{4.2}$$

  Here, P is uncertainty covariance matrix, which is initialized with a small initial uncertainty for the position part and much larger initial uncertainty for the velocity part, since laser scanners can give precise estimate of the position and no estimate of the velocity. Q_α is state uncertainty matrix which is used to add uncertainty at the prediction step.

- Measurement

  This step is performed only when the detector gives an output.

$$y = z - H \cdot x \tag{4.3}$$

$$S = H \cdot P \cdot H^T + R \tag{4.4}$$

$$K = P \cdot H^T \cdot S^{-1} \tag{4.5}$$

$$x = x + K \cdot y \tag{4.6}$$

$$P = (I - K \cdot H) \cdot P \tag{4.7}$$

  Where, y is innovation (measurement residual), computed as a difference of a new measurement and the current position state, S is innovation covariance, R is a matrix that characterizes measurement noise. K is the *optimal* Kalman gain. Equations 4.6 and 4.7 are posteriori state estimate and estimate covariance.

4.1.1 Filter matching

For each pedestrian in the scene a corresponding filter is used. If a pedestrian is just appeared, a new filter is initialized, and after a few frames it appears in the scene. If the same pedestrian is not detected for a few frames the associated filter is destroyed.

The usage of multiple filters leads to the process of matching. In our case, the Mahalanobis distance is used to match the current state of each filter with a new state. It is a unitless and scale-invariant, and takes into account the correlations of the data set.

$$d(x, y) = \sqrt{(x - y)^T S^{-1} (x - y)}$$

Where x and y are state vectors, S is the covariance matrix. A single pedestrian is matched with a single filter if Mahalanobis distance is within a certain threshold. The minimum possible distance is chosen. The threshold is usually chosen according to the number of degrees of freedom of the system and the value at which the false measurements may occur.

$X = chi2inv(P, V)$ computes the inverse of the chi-square cumulative distribution function with degrees of freedom specified by V for the corresponding probabilities in P. However, P is not as obvious as V. Therefore, the Mahalanobis threshold was determined by plotting all those distances (figure 4.1). This plot was done for 2 pedestrian filters during the matching procedure. Usually the matching is done for the first pedestrian, when the best match is found, it is deleted from the list of candidates and the matching is continued for the next one. That is why, many of the points on the figure lie within the threshold, because, in this case, the second pedestrian is usually matched with its correct candidate.

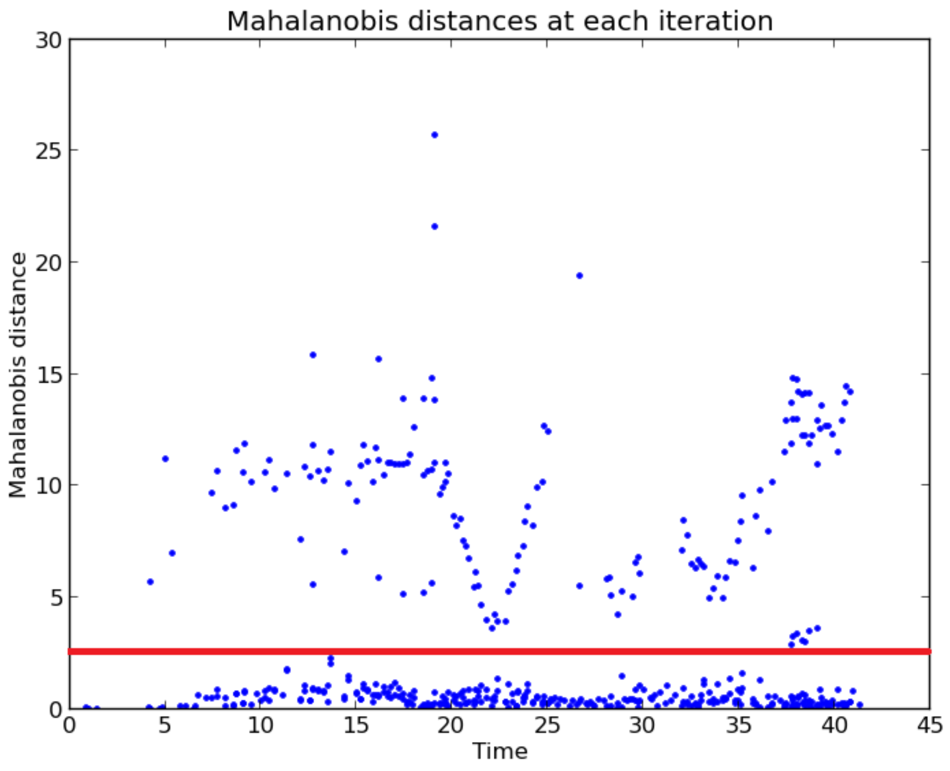

FIGURE 4.1: Mahalanobis threshold = 2.5 is chosen. Most of the correctly matched pedestrians lie within this threshold.

Chapter 5

Experiments and Results

In this chapter we will discuss obtained results. Many different algorithms were used, especially in the detection part. Relatively good detection rate was achieved by using just vision, however the solution was not working with real time constraints. There was a need for speed-ups.

Initially, GPUs were planned to be used. However, many things were not implemented completely to start GPU programming, not because it is hard to implement, but because it requires quite a bit of time for technical issues, like dealing with compiler and specific syntax. In addition, the GPU version of HOG classifier was working 50 fps. Moreover, closer to the end of the project, usage of either GPU-speedup or speedup by laser scanners was planned. As soon as laser scanners were installed, they were calibrated and simple sensor fusion was implemented. This significantly improved performance of the detection.

5.1 Detection

For Integral channels detector the key parameter is the number of features. In our case, many different numbers were tried. A good compromise between the quality and the time of detection was found at the value equaling to 2000 features. Different versions of classifiers were trained. Table 5.1 compares the result of 2 different classifiers.

i	Training	Samples	Features	Weak cl.	FPPW
1	Initial	18500	2048	1000	0.0047
2	Initial	18500	1024	500	0.0053
3	Bootstrapped	22000	2048	1000	0.0034
4	Bootstrapped	22000	1024	500	0.0045

TABLE 5.1: Different classifiers at the same miss rate = 0.0415.

We can see that with the increase of the number of features and weak classifiers False Positive Per Window (FPPW) rate decreases. Bootstrapped version of the classifier improves the detection.

On the figure 5.1 we can see that the detection performance of classifiers with 1024 and 2048 features with initial training.

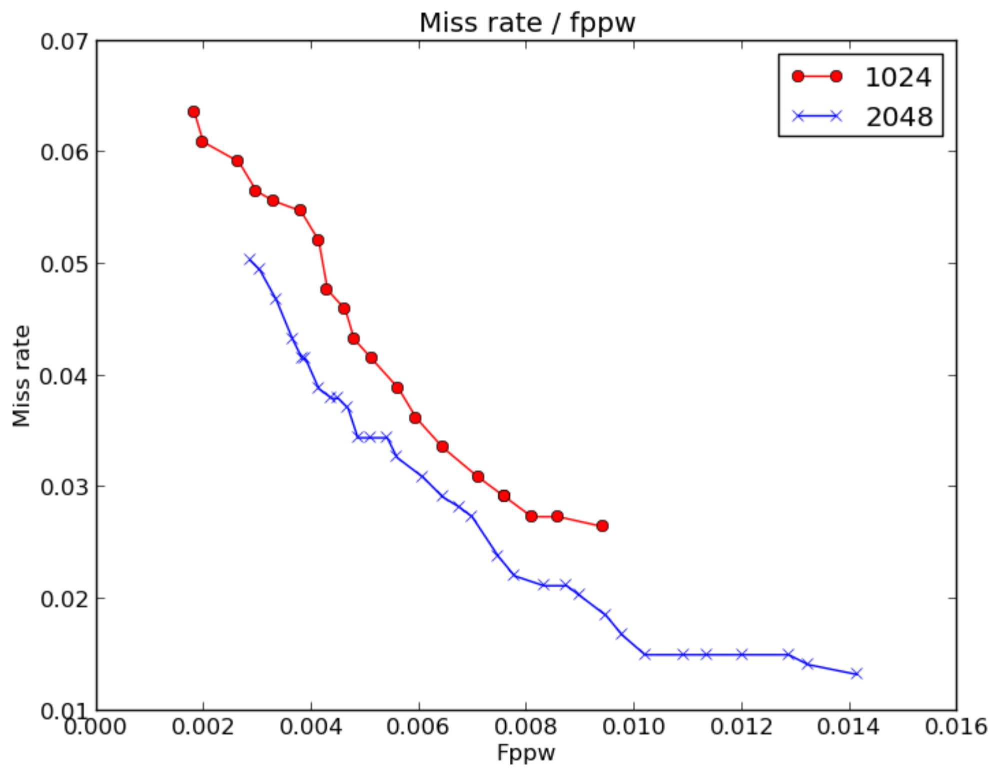

FIGURE 5.1: Detection performance of classifiers with 1024 and 2048 features.

5.1.1 Detection on a single image

The first version of the detector was implemented for a single image. A fixed scan step version run around 6-8 seconds for a 640 × 480 image. An improved version with a variable step run around 1-2 seconds.

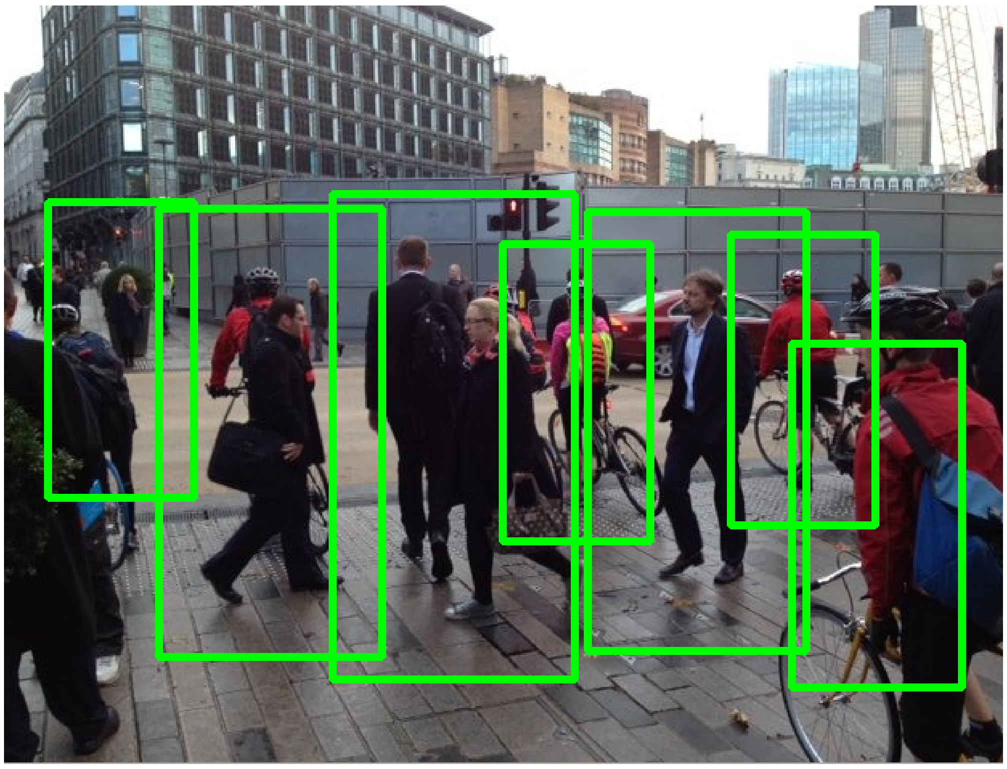

FIGURE 5.2: An example of the detection on a single image.

As one can expect from the machine learning algorithm, there are false negatives and false positives results (figures 5.3 and 5.4). All of those examples are from the INRIA Pedestrian Dataset.

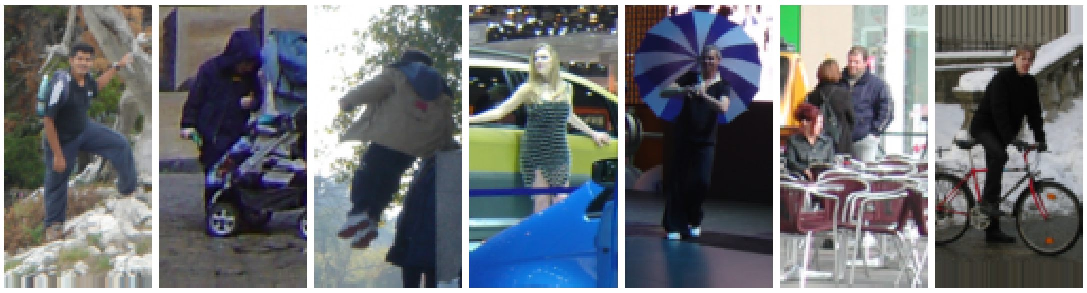

FIGURE 5.3: False negative images .

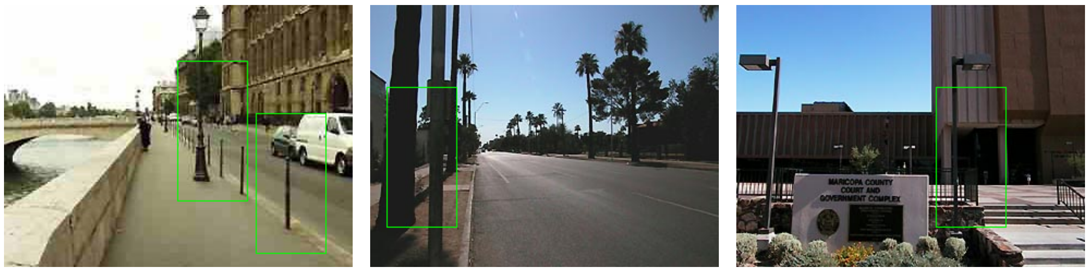

FIGURE 5.4: False positive images.

5.1.2 Detection with lasers and vision

The points from the laser scanner are projected onto the image (figure 5.5). Not all of the points are projected because of the limited field of view of the camera, whereas a laser scanner has very wide view, which allows to see objects on the sides of the vehicle. All the projected points are converted into corresponding bounding boxes (figure 5.7).

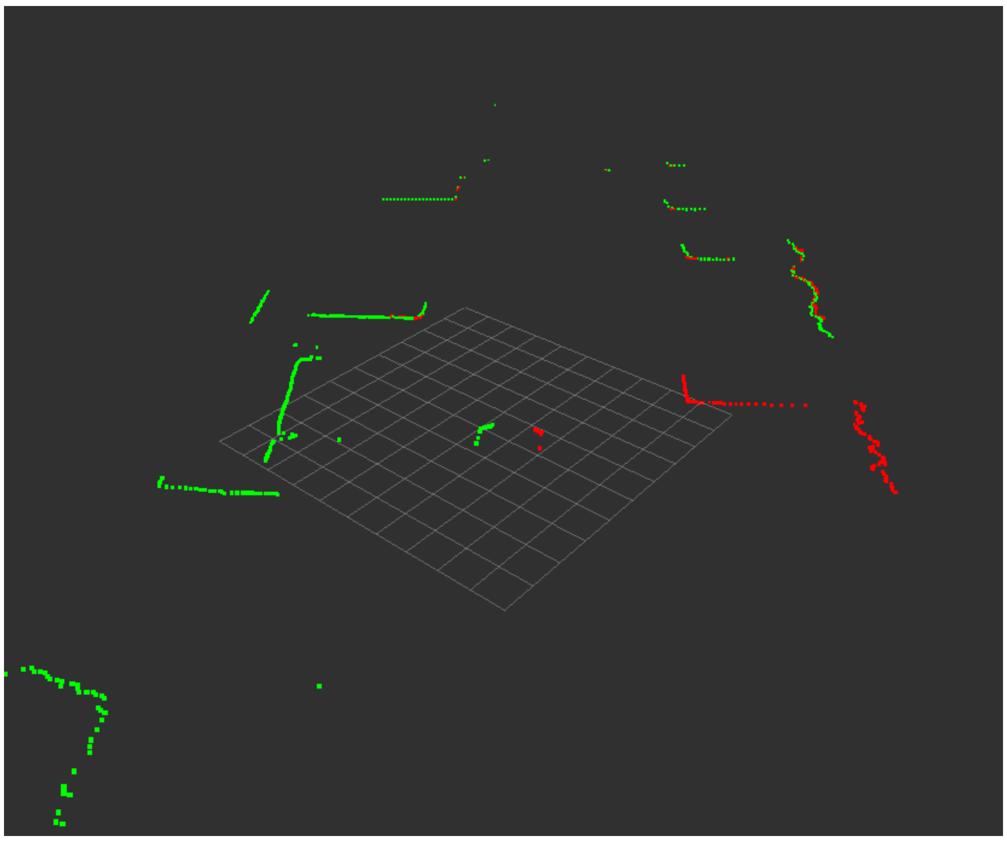

FIGURE 5.5: All objects detected by 2 laser scanners.

Entire algorithm was run on the 45 seconds length video. In this video, there were very few false positives, around 5 10 from time to time, and almost the time, these false positives are the same pedestrians detected on different scales (figure 5.8).

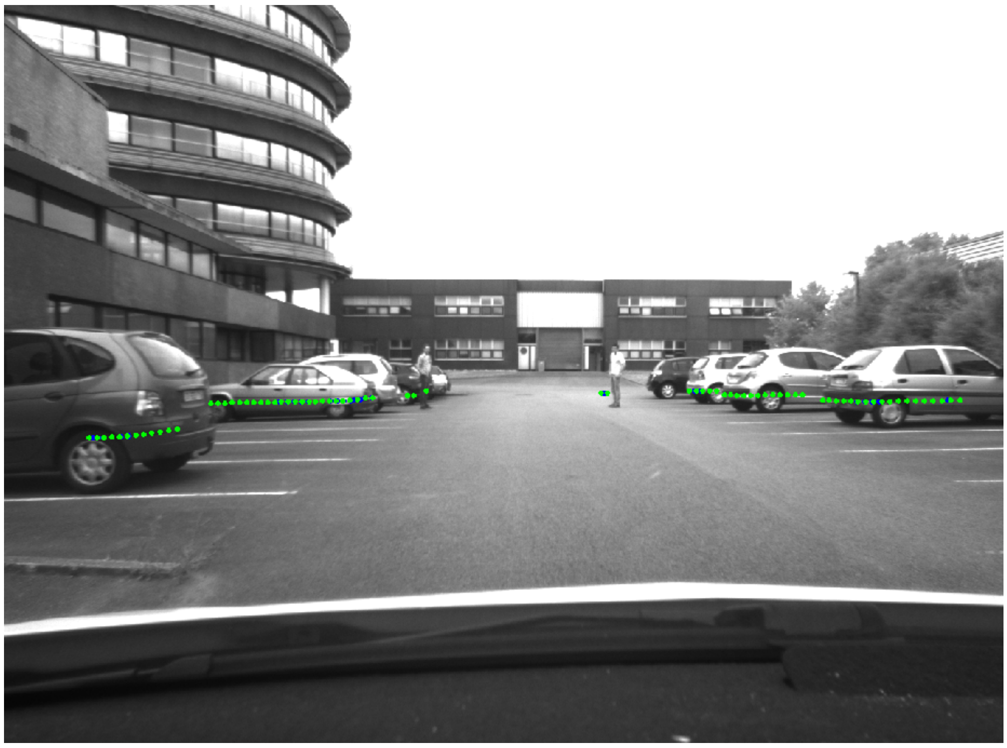

Figure 5.6: Laser candidates projected onto the image.

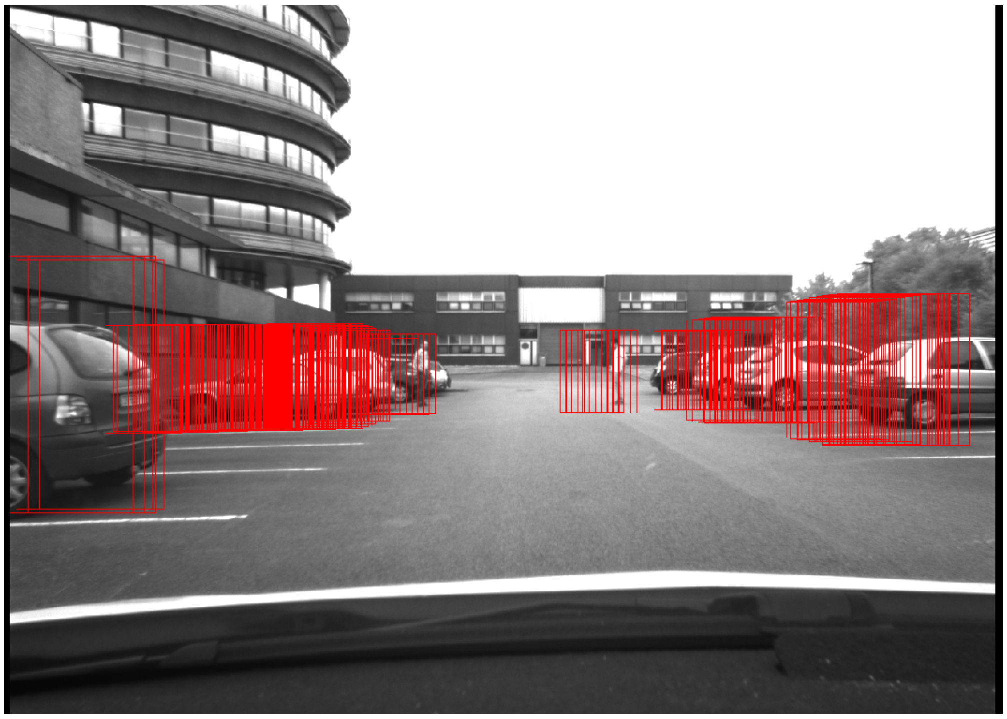

Figure 5.7: Projected points converted into the corresponding bounding boxes.

This happens because there are objects detected by a laser scanner and therefore candidate bounding boxes of different sizes are created for pedestrian points and objects behind them and classifier gives positive response for different scales of the same pedestrian, while clustering algorithm outputs two different results, because the distance between those objects is far enough. This problem can be avoided performing another type of clustering, possibly for the laser points. However, in our algorithm those false positives are filtered out by filtering.

5.2 Filtering

Filtering is very helpful not only in the cases when there is a false positive, but in cases when there is a false negative (when a pedestrian is not detected by the classifier), in such cases it helps to predict the state of a pedestrian and for a few frames it can work without any response from the classifier. For the same video, 40-55 times there was no pedestrian detected for the corresponding filter, those cases were filtered out because they appear at different times and for a different pedestrian.

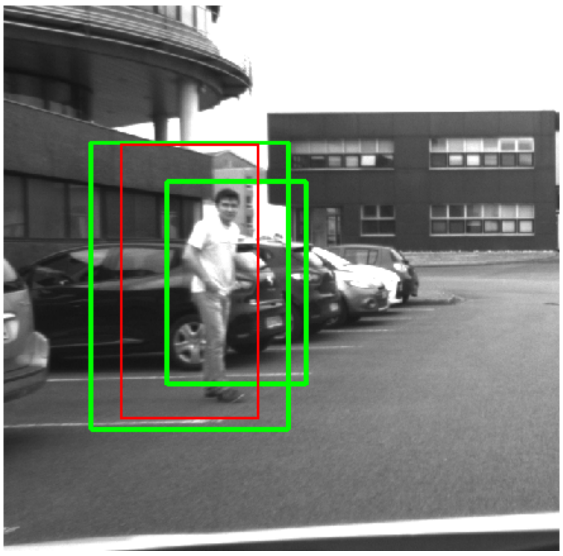

FIGURE 5.8: The same pedestrian is detected at different scales. Large rectangles are the detector outputs and the small one is filter output.

The figures 5.9 and 5.10 show the evolution of 3 filters for 2 pedestrians, because the second pedestrian leaves the scene and appears again. The first pedestrian is being detected from 22 meters distance, whereas the second one is detected from the distance 14 meters.

On those figures (5.9 and 5.10) measurements are taken after applying clustering of the points, and only with the closest Mahalanobis distances. Others are ignored.

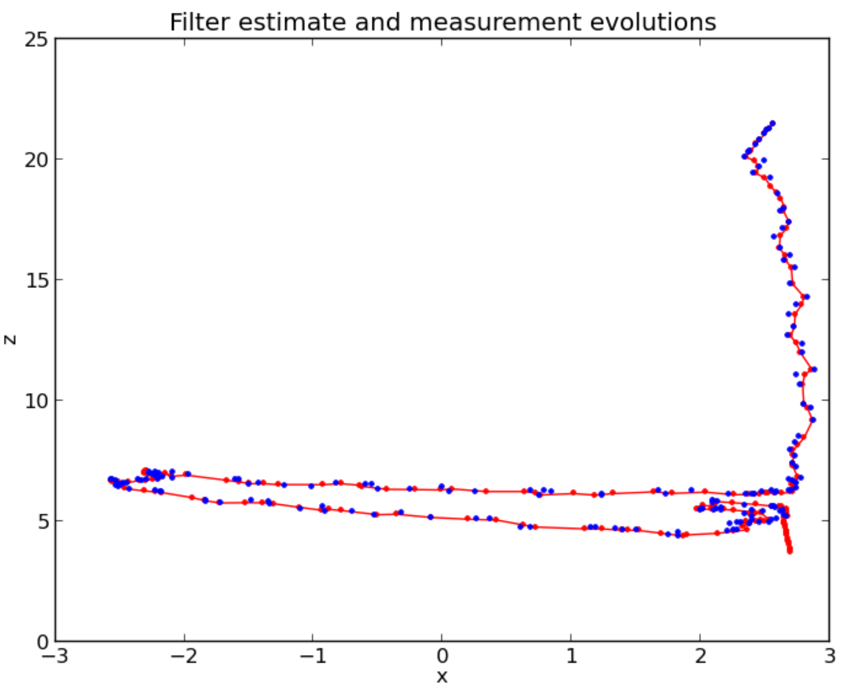

Figure 5.9: Filter estimate and measurements for the first pedestrian on the video. Red points represent filter estimate and blue ones measurements.

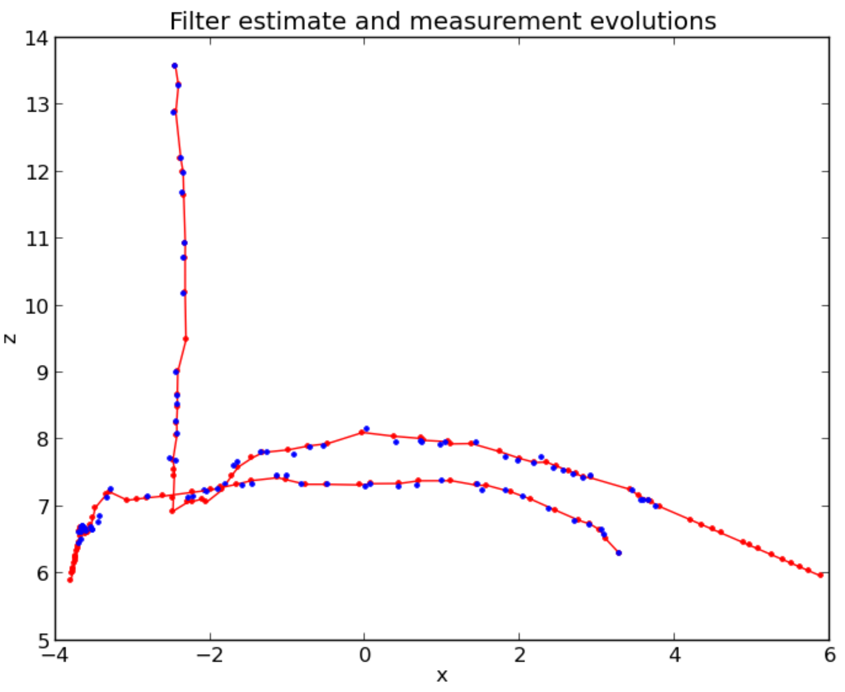

Figure 5.10: Filter estimate and measurements for the second pedestrian on the video. Red points represent filter estimate and blue ones measurements.

Another informative plot (figure 5.11) shows the all the detected points which gave positive response without applying clustering.

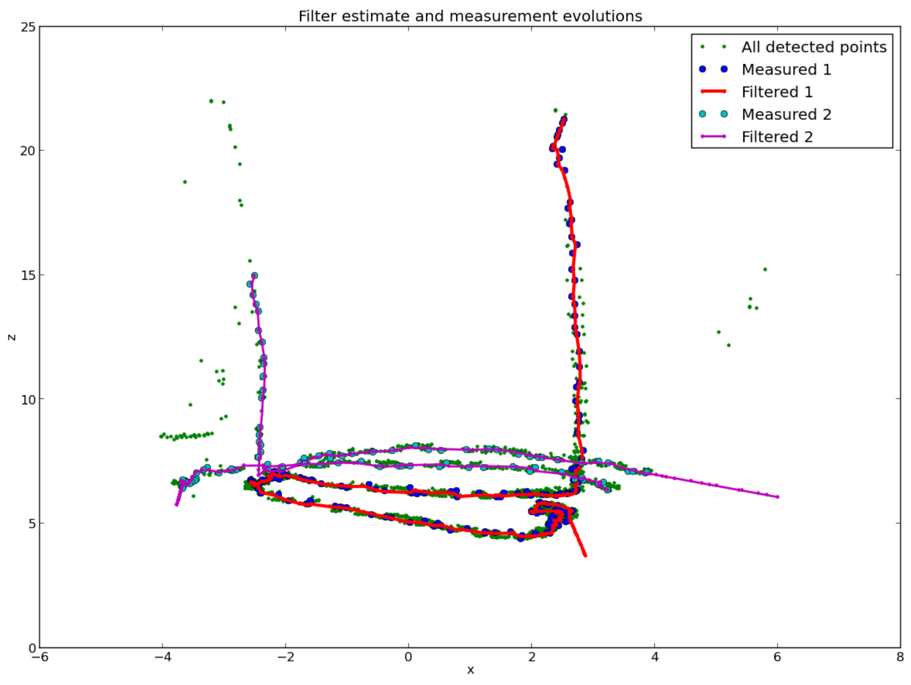

FIGURE 5.11: All the points at which pedestrians were detected. Lines represent paths of 2 pedestrians. The circles are the measurements after application of clustering.

Figure 5.12 shows the standard deviation values taken from the covariance matrix of a filter for one pedestrian. We can see that they make sense. They are small enough when there are measurements and getting larger when there is no measurement.

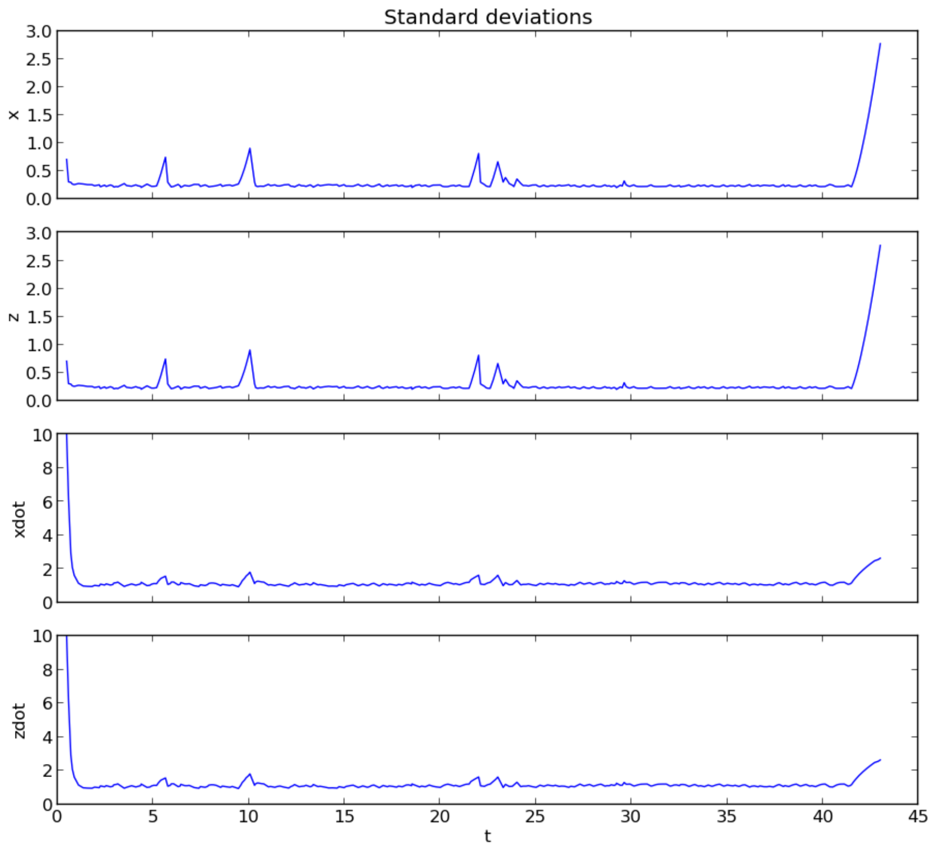

FIGURE 5.12: Standard deviations of the state variables for one pedestrian filter.

Chapter 6

Conclusion

Intelligent vehicles represent a key technology for reducing the number of accidents between pedestrians and vehicles. Pedestrian Detection and Tracking plays an indispensable role in building such vehicles. PDT is a relatively new area of research and by no means an easy task. Numerous papers address the challenge. In this thesis, we reviewed some of the existing algorithms and techniques to perform PDT using laser and vision systems. Some of them were implemented and good results were obtained.

For the visual detection part we used Integral channel features (section 2.3) and AdaBoost classifier (section 3.1.1.3) which gave good results. Training and testing of the classifier was done using INRIA Pedestrian Dataset. To achieve performance closer to real-time and improve precision and recall, laser scanners were used (section 3.2.3.4). This was done by selecting regions of interest on the image. Clusterization based on the response value and distance to the next candidate is performed. Kalman filtering (section 4.1) was used to filter out cases when pedestrian is occluded or when the false detections occur. In addition, filtering can be used to predict velocity of a pedestrian.

Implementation was done on Linux computer, the main programming language was C++, the main library used to process images was OpenCV and Robot Operating System was used to work with the vehicle and the hardware.

6.1 Future Work

The implemented system is ready-to-use and can be integrated into the vehicle. However, many things can be improved and extended. If time allowed, each part of the algorithm could be analyzed much thoroughly, for example, for the detection part we could use more features and more training samples, translate codes to run on GPUs. Other detectors could be implemented and compared. The sensor fusion could be implemented in a much smarter way, for example assigning different weights depending on the situation. Clusterization of the laser points could be done to remove inappropriate objects from the candidate list. Matching of the filters could be done using other criteria, rather than just Mahalanobis distance. Furthermore, different tracking methods could be implemented, for example particle filtering to compare it with the current one.

Yet another idea was to use part based pedestrian detection, where different body parts are detected separately and then fused together with certain weights according to detection confidence and location of the part in order to make a final decision.

Overall, the system could be extended to label pedestrians depending on what they are doing and where they are going. Moreover, other classes of objects could be detected by the classifier.

Bibliography

[1] David Geronimo; Antonio M. Lopez; Angel D. Sappa; Thorsten Graf. Survey of pedestrian detection for advanced driver assistance systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1239–1258, July 2010.

[2] Stefan Schramm; Franz Roth; Johann Stoll; Ulrich Widmann. Proactive pedestrian protection. *Handbook of Intelligent Vehicles*, 31:787–826, 2012.

[3] Philip Kelly. Pedestrian detection and tracking using stereo vision techniques. *PhD Thesis*, 2007.

[4] Piotr Dollár; Christian Wojek; Bernt Schiele; Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(4):743–761, April 2012. URL https://s3-us-west-2.amazonaws.com/mlsurveys/97.pdf.

[5] Paul Viola; Michael J. Jones; Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, February 2005. URL http://luthuli.cs.uiuc.edu/~daf/courses/AppCV/Papers-2/t61k38u53j53134.pdf.

[6] Markus Enzweiler; Dariu M. Gavrila. Monocular pedestrian detection: Survey and experiments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2179–2195, December 2009. URL http://gavrila.net/pami09.pdf.

[7] N. Dalal; B. Triggs. Histogram of oriented gradients for human detection. *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition*, pages 886–893, 2005. URL http://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf.

[8] Piotr Dollár; Zhuowen Tu; Pietro Perona; Serge Belongie. Integral channel features. 2009.

[9] Yoav Freund; Robert E.Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, pages 771–780, September 1999. URL http://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf.

[10] Kay Ch. Fuerstenberg; Ulrich Lages. Pedestrian detection and classification by laserscanners. 2003.

[11] Hanxuan Yang; Ling Shao; Feng Zheng; Liang Wang; Zhan Song. Recent advances and trends in visual tracking: A review. *Neurocomputing*, (74):3823–3831, July 2011. URL http://lshao.staff.shef.ac.uk/pub/Tracking_NeuCom2011.pdf.

[12] Nguyen Xuan Tuong; Thomas Muller; Alois Knoll. Robust pedestrian detection and tracking from a moving vehicle. 7878, January 2011. URL http://www6.in.tum.de/Main/Publications/Mueller2011c.pdf.

[13] M.K Tay; K. Mekhnacha; M. Yguel; C. Coué; C. Pradalier; C. Laugier; Th. Fraichard and P. Bessière. The bayesian occupation filter. *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*, pages 77–98, 2008. URL http://hal.archives-ouvertes.fr/docs/00/29/50/84/PDF/08-star-tay-etal.pdf.

[14] Luciano Oliveira; Urbano Nunes; Paulo Peixoto; Marco Silva; Fernando Moita. Semantic fusion of laser and vision in pedestrian detection. *Pattern Recognition*, 43(10):3648–3659, October 2010. URL http://www.sciencedirect.com/science/article/pii/S0031320310002347.

[15] Bo Wu; Jixiang Liang; Qixiang Ye; Zhenjun Han; Jianbin Jiao. Fast pedestrian detection with laser and image data fusion. *International Conference on Image and Graphics*, 2011.

[16] Rodrigo Benenson; Markus Mathias; Radu Timofte; Luc Van Gool. Pedestrian detection at 100 frames per second. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2903 – 2910, June 2012. URL http://rodrigob.github.io/documents/2012_cvpr_pedestrian_detection_at_100_frames_per_second.pdf.

[17] Piotr Dollár; Serge Belongie; Pietro Perona. The fastest pedestrian detector in the west. pages 68.1–68.11, 2010. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.310.8578&rep=rep1&type=pdf.

[18] Rodrigo Benenson; Radu Timofte; Luc Van Goo. Stixels estimation without depth map computation. *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference*, pages 2010 – 2017, November 2011. URL http://vpa2.sabanciuniv.edu/conferences/ICCV2011/ICCV2011_workshops/material/ws23_008.pdf.

[19] Antonio Prioletti; Paolo Grisleri; Mohan M. Trivedi; Alberto Broggi. Design and implementation of a high performance pedestrian detection. *IEEE Intelligent Vehicles Symposium*, (IV):23–26, June 2013. URL http://www.ce.unipr.it/people/bertozzi/pap/cr/iv2013.pd.pdf.

[20] Piotr Dollár; Ron Appel; Wolf Kienzle. Crosstalk cascades for frame-rate pedestrian detection. *12th European Conference on Computer Vision*, pages 645–659, October 2012. URL http://vision.ucsd.edu/~pdollar/files/papers/DollarECCV12crosstalkCascades.pdf.