Technical Report

Pattern Recognition and Image Processing Group
Institute of Computer Aided Automation
Vienna University of Technology
Favoritenstr. 9/1832
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801-18351
Fax: +43 (1) 58801-18392
E-mail: seitnerf@prip.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

PRIP-TR-100 December 21, 2005

Robust detection and tracking of objects

Florian Seitner
seitnerf@prip.tuwien.ac.at

Abstract

Due to the increasing availability of fast and cheap hardware in the past few years, today a wide range of complex visual tracking tasks is possible. Efficient mathematical methods can provide a high robustness which also makes visual tracking interesting for many industrial purposes. However, the high demands on quality and speed still provide a major challenge for each tracking application. In this thesis a tracking system is introduced, which tries to address both demands appropriately by using currently available algorithms to quickly track pedestrians in video streams. By combining these well-proved algorithms, a good solution regarding computational complexity, accuracy and stability is obtained.

To achieve this task, a fast object detector similar to the approach of Viola et al. [Viola2003] is used as one component in this tracking system. This detector uses Haar-like features which are very fast to compute and makes a quick pedestrian detection in a frame possible. Next to the detection system, an adaptive background model sub-divides each frame into foreground and background regions. As a compromise between complexity and robustness a single-mode parametric background model based on normal distributions and wrapped normal distributions is used. Both background model and detector are combined to provide the tracking system with locations of pedestrian-like regions and to sub-divide the body into three parts: head, upper body and lower body. After this segmentation into finer tracking units a set of colour and spatial features for further tracking is extracted from each part individually. Individual and spatially separated body parts also provide the possibility to use colour histograms in a spatial sense. Moreover, an appearance model provides accurate solutions and approximations when occlusions or missing detections occur.

Contents

List of Figures

Chapter 1

Introduction

The significant problems we face cannot be solved at the same level of thinking we were at when we created them.

Albert Einstein (1879 - 1955)

This chapter describes the main motivation behind this work and how it contributes to the field of computer vision. Moreover, it gives an overview of currently available tracking techniques and related work by describing their basic ideas and the theory behind them. A description of the structural design of this thesis closes this chapter.

1.1 Motivation

The task of tracking a complex object in a real world environment requires almost no effort for a human being. However for computers, simple object detection is already quite a difficult task and often complex mathematical methods on fast computers are employed to attempt to cope with this problem. The results are usually far from being as good as those achieved by human beings. The question arises, why computers are required for this task.

The simple answer lies on the one hand in the continuous automation by using technology and on the other hand in the huge amounts of image and video data which is produced and transfered daily. Today video data is available over a wide range of media and the increasing transfer rates over satellites or high-speed networks make fast access possible. Additionally, low-cost video cameras are widely available which makes them interesting for industrial applications. The processing of large amounts of data is always relatively time consuming or complex and therefore, for human beings, often mentally exhausting and subject to errors. Computers can, as in many industrial automation processes, support human operators in their daily work and improve the quality as well as the efficiency of a task.

Over the last years the demand for reliable tracking of objects in video streams has risen dramatically. There exist a large number of real-world applications where an automatic object tracking system could support a human operator in dangerous, extensive or time-consuming tasks and increase the efficiency, safety, as well as the security. A pedestrian is a good example for a complex object and is often used in tracking applications for demonstrating the functionality of a tracking method. The tracking system introduced in this work is designed to track pedestrians in video sequences but could be modified to track other objects (i.e. cars) as well.

The specific area of pedestrian tracking has numerous applications ranging from tasks in traditional video surveillance to human safety, marketing analysis and traffic systems. A system which reliably tracks pedestrians in video streams can — to a certain degree — address multiple security tasks such as to trace a robber after a bank robbery, to locate a lost child in a crowd or to decelerate a car automatically to avoid collisions with crossing pedestrians. Moreover, statistical applications based on the pedestrian frequency are imaginable. Examples such as to determine the utilization ratio of public transport and their capacity bottlenecks or to evaluate the value of a shop or an advertisement by the number of people passing by are only few of them. Fields like landscaping and urban planning provide an additional huge application field. Here an accurate planning according to the human need can be supported.

Next to the fact that many of these tasks rely on a high robustness, the throughput rate of a tracking system is also critical. Normally real-time processing of the arriving data is desired or necessary and large amounts of data — due to resolution, frame rate or multiple cameras — have to be processed quickly. The main motivation in this thesis was to design a tracking system for pedestrians which addresses the robustness as well as the speed of the system by combining well established methods of pattern recognition, machine learning and image analysis. Furthermore, the memory usage and efficiency should be addressed since large data amounts can quickly lead to the complete consumption of the available memory and requires careful consideration.

This tracking system focuses on robustness, efficiency and speed but since it represents a prototype which often requires changes, corrections and improvements, Matlab as a high-level programming language was chosen for the implementation. This language is specialized in scientific computation and provides a multitude of mathematical functions, operations and structures. Matlab also manages the memory allocation for data structures which allows the user to completely concentrate on the problem itself. Widely used, it is available for all major operating systems like Windows or Unix-like systems (e.g. Linux, Mac-OS) and supports compilation of the Matlab code into native machine code for all available platforms. Additionally, it can work together with a wide range of low-level interfaces and libraries and can therefore access many external ports and devices like serial or USB ports and web-cams. All these possibilities make Matlab a good choice for prototyping. Due to speed reasons, a manual transfer of the prototype to a low-level language afterwards is recommended, since the native code which is created by Matlab is quite fast but does not reach the speed of a native low-level language implementation.

The emphasis of this thesis lies in the design and implementation of a tracking system for pedestrians, which provides a usable degree of robustness and is — regarding its design — optimized to quickly process large amounts of video data. Well-known tracking problems like occlusions or missing detections are addressed.

1.2 Problem statement

Tracking of various kinds of object has been addressed in numerous works. An overview of the specific field of pedestrian tracking is given in Section 1.3 and various tracking approaches are described. However, every visual tracking algorithm has to deal with the same fundamental problems. The central challenge is to determine the location of a target object as it moves through a camera's field of view. This is normally done by matching multiple regions or features in successive frames of a video stream. This problem of feature matching is called the *temporal correspondence problem* and also arises in other related areas like motion estimation. The difficulty in solving the temporal correspondence problem is due to three main factors ([HB98]):

- Variations in the target pose or target deformation
- Variations in the illumination
- Partial or full occlusions of the target object

Ignoring only one of these factors would lead to inaccurate results and therefore a reliable tracking algorithm has to address all three kinds of variations. Furthermore, the algorithm has to do this with a certain computational efficiency since it often has to process the data — if not in real-time — at least in a reasonable amount of time.

In a video sequence where the observations are recorded with a moving camera, the complexity of the three problems listed above gets even higher. Therefore we made the assumption that each scene is observed with a static camera and designed the tracking system according to this.

The tracking system introduced in this thesis is designed to accurately cope with the three kinds of variations in a static camera environment and, at the same time, to provide an efficient way to quickly process the arriving video data.

1.3 Literature and related work

Tracking applications and algorithms have been used since many years and various approaches have been introduced. Strong variations between the different approaches exist, strongly depending on the available computation power and mathematical methods, the field of application or the requirements on efficiency and robustness. Alone in the field of pedestrian tracking the number of different approaches is high.

Most systems which try to detect or track objects in images work in the visual [WADP97, BBF+04] or infrared [KCT04, SKK00, BBBD05] spectrum of light and use one or multiple cameras [ZT00, BBDL05]. There exists a multitude of tracking approaches with strong differences regarding the concepts, the mathematical theory or the addressed environment (e.g. outdoor/indoor, static/dynamic camera). Each of these systems therefore has its very specific characteristics, strengths and problems. In this work we use

pedestrians as an example for a complex object tracking application and the overview given in this section will describe existing work done in this specialised area.

The concept or basic idea behind every tracking system is nearly always the same. The first step of most tracking algorithms is to locate regions of interest with a high probability of containing a pedestrian. Normally this is done by extracting low-level features like pixels or blobs from image areas. Various classification techniques used in pattern recognition are applied to determine if a pedestrian is present. For example, Oren et al. [OPS+98] use simple wavelet templates in combination with *Support Vector Machines* (SVMs) to classify pedestrians by their shape. Levi et al. [LW04] uses a similar shape-based approach based on edge orientation histograms (EOHs) for detecting dominant edge orientations and symmetry in faces which also could be used for pedestrian detection. A hierarchical approach for pedestrian shapes is introduced in [GG02] and uses a coarse-to-fine template matching technique based on distance transforms. Another statistical approach by Philomin et al. in [PDD00] uses a *Point Distribution Model* (PDM) to model the pedestrian shapes by a compact linear shape model. Afterwards, *Principle Component Analysis* (PCA) is used to extract a mean shape. A widely used approach which uses Haar-like feature templates and a cascade of boosted classifiers for pedestrian shape description is used in [JV03, LM02, LLK03]. A system for hand detection by contour comparison is introduced in [OB04]. Here the difference between two shapes or contours is calculated with a cost function and during detection a search for image areas with low cost values is performed. Approaches based on neural networks have also been used successfully for pedestrian detection in images [ZT00].

After the positive detection of a pedestrian, the system has to keep track of this object while it crosses the scene. It uses *tracking features* based on colour, contour, shape or texture to find the objects in the sequential frames again. It is therefore of great importance that the tracking features are – to a certain degree – time-invariant. For the case that a tracked object cannot be located in the current frame by using its tracking features, the tracking system must provide adequate estimations. Since each pedestrian can be regarded as a dynamic system in an unknown state, filters can be used to provide this state estimate. A multitude of different filters for dealing with this problem are used in current work. A widely used approach for state approximation is based on *Hidden Markov Models* (HMM). They are closely related to *Bayesian Networks* (BN) and regard the real state of a dynamic system as an unobserved (or hidden) *Markov process*. Each measurement is called an observed state of the HMM. The goal of this method is to find the sequence of hidden states which is most probable. This sequence is called the *Viterbi path.* An example of a tracking system for face contours based on HMMs is described by Chen et al. [CRH01].

The often used *Kalman filter* (KF) is based on HMMs. This efficient filter uses the last system state and the current (noisy) measurements to estimate the current state. The restrictions of the KF are that the measurement noise is regarded as Gaussian distributed and the filter is limited to linear assumptions. Most natural systems are non-linear and an extension of the Kalman filter – the Extended Kalman filter (EKF) which can also estimate non-linear dynamic systems – must be used here. However the EKF is still bound to Gaussian distributed noise. Bertozzi et al. [BBF+04] and Gavrila and Giebel [GG02] both detect pedestrians by using shape properties and use Kalman filters for estimating the position of the tracked person. An approach based on motion detection in gray-scale images and tracking of blobs with Kalman filters is introduced by Masoud et al. [MP01]. Here the pedestrian is modeled as a rectangle with a certain dynamic behavior and a Kalman filter is used for the pedestrian parameter estimation.

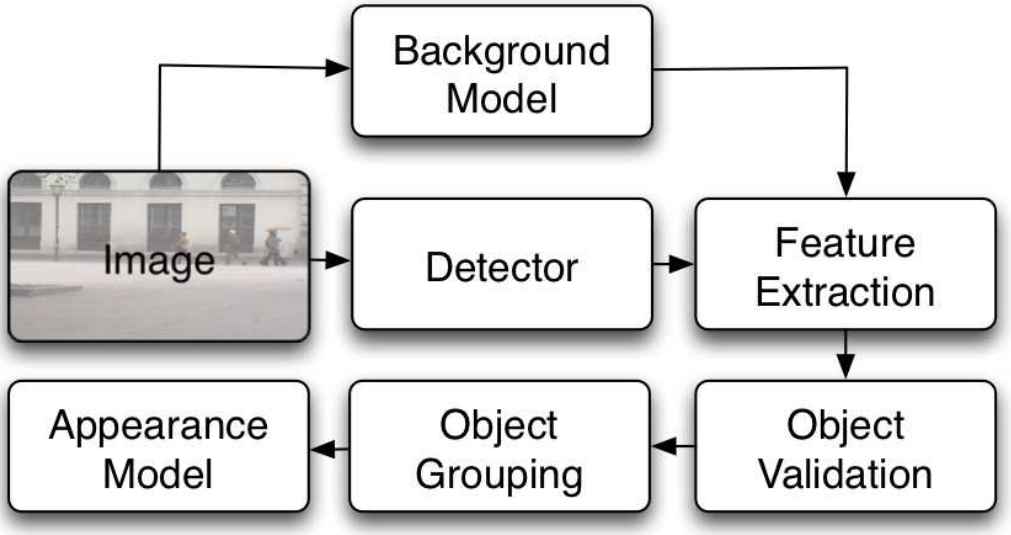

Figure 1.1: Structure of the tracking system

Another filter approach is based on *Sequential Monte Carlo methods* (SMC) or *Particle filters* (PF). They are often used for spatio-temporal estimation of shape and position and can – in contrast to the KF – also estimate dynamic systems with non-Gaussian noise or non-linear system behavior. This advantage has its cost and normally particle filters are computationally more expensive than Kalman filters. Isard and Blake [IB98] introduce the *Condensation* particle filter, which tracks the shape of faces, leaves or hands in complex video sequences. In [GGB+02] a general tracking system for aircraft based on particle filters is introduced. Shan et al. [SWTO04] use a Particle Filter in combination with a *Mean Shift* algorithm for hand tracking in real-time. Another tracking approach for human bodies based on motion and colour is described in [PT03]. This work uses background segmentation to extract dynamic image areas and tracks these regions by using colour information in combination with a *Mean Shift* algorithm.

1.4 Overview of the system

It is obvious that there is more than one way to implement a tracking system. In this work the prime focus is a high processing speed of the image frames and existing techniques should be combined to achieve near real-time performance.

Figure 1.1 visualizes the structure of the tracking system implemented and shows how the parts work together. Each frame of a video sequence is independently processed by a detector and a background model. The background model separates the raw image into foreground and background regions and creates a mask where each pixel is assigned to one of these two regions. In each image the detector part tries to locate pedestrians with various positions and sizes. It describes each detected pedestrian by a rectangle which determines the position and the size of a pedestrian in the image. Since this description of a pedestrian is not unique and does not support a reliable tracking process we extract additional features

of the pedestrian. This is done in the feature extraction part where the already available information about a pedestrian is extended with colour information and information about the single body parts of the pedestrian. We use a method which uses the available background model to gain this additional information. After we have the tracking features for each pedestrian detection we validate them for conclusiveness. This is done in the object validation part of the system where we decide if an object provides us with enough information for the tracking process. After validation the detections are grouped together to eliminate multiple detections of the same pedestrian. An appearance model collects the available information about each object during the whole video sequence and handles occlusions and wrong detection. No previous work based on such an approach could be found.

This thesis is organized in seven chapters and covers multiple aspects of object detection and tracking. Chapter 2 describes the background model. For better understanding how this model describes each pixel colour of the background a short overview of the theory of colour spaces is given and different colour models compared. Due to the fact that some colour representations use directional and linear values to describe a colour and its *colour components*, important terms of circular statistics for accurate processing of directional data are given in Section 2.2. Furthermore, Section 2.3 introduces different density functions for modelling the distributions of these colour components and Section 2.4 describes an adaptive model for static background modelling.

A detector for locating pedestrians in static image frames is presented in Chapter 3 and the required terminology explained. A fast detection algorithm based on Haar-like features ([VJ02, WAHL04]) was used for the pedestrian detection. The theory behind the detection system and the implementation is discussed in detail in Section 3.4. A method for quickly validating a detection by using the background model is proposed and the traditional definition of *boosted classifiers* (Section 3.3.2) is extended and optimized. This extended boosted classifier achieves the same accuracy regarding the classification results by using fewer computations than a normal boosted classifier.

Chapter 4 discusses the feature extraction. Therefore it describes some issues concerning the calculation of colour and spatial features for describing pedestrian-like objects and how they can be obtained from single images and video streams. After the features have been extracted, methods for validating the features according to logical and physical assumptions are introduced in Section 4.3. Furthermore, appropriate distance measures to support this validation task and to estimate the similarity between two objects are described and a brief introduction on the confidence of spatial and colour features is given.

The adaptive appearance model builds the highest layer of this tracking system and combines all parts together. It is the content of Chapter 5 which defines important terms like *model* and *object state* and describes how an object can be assigned to a detection. Moreover, this chapter shows how features of an object are updated and how their reliability changes during the tracking. In times when a feature cannot be determined exactly by using the current frame information, accurate methods to predict and to approximate its value must be provided. They are topic of Section 5.3.2.

The performance of the tracking system is discussed in detail in Chapter 6, where each component of the tracking system is analyzed. Moreover, the training and the test parameters are described and specific characteristics of the system like error rates and tracking speed analyzed.

Finally, in Chapter 7 a brief summary is given and conclusions are drawn. Moreover, some suggestions for future work based on the results of this thesis are made.

1.5 Contribution

In this thesis the following contributions to the field of pedestrian tracking are made

- A new combination of a background model and a fast Haar-like feature based detector for validating detections and for feature extraction is presented. The tracking system uses spatial and colour features and multiple individual body regions.
- Development of an accurate probability model for static background modelling.
- An extension of the traditional boosted classifier structure is introduced which makes a fast pre-rejection of negative samples already possible during the detection phase.
- A clipping algorithm to accelerate the classification process is developed.
- A method for weighting tracking features according to their reliability is introduced.
- An asymmetric FloatBoost version is introduced.

Chapter 2

Background model

See things as you would have them be instead as they are.

Robert Collier (1885-1950)

The proposed tracking system relies strongly on tracking features extracted from the foreground. To distinguish between foreground and background information and to assign frame pixels to one of these two domains, a background model is necessary, which is introduced in this chapter. Since the model operates on the HSV (hue, saturation and intensity) colour space, a short introduction to this colour representation and its advantages over the traditional RGB colour space is given in section 2.1. To address the circular nature of hue component in the HSV colour space, an accurate method for processing of directional data is described and important terms of circular statistics are introduced in Section 2.2. Section 2.3 proposes accurate density functions for modelling the colour components of a pixel colour and Section 2.4 describes how they are used in a parametric background model.

2.1 Colour spaces

For accurately describing natural colours we have to find a mathematical representation or model. Such colour models are called colour spaces and there are numerous of these spaces available. Each of them has its advantages and disadvantages and their usage normally depends strongly on the application and the requirements like quality, computation speed or required technology.

2.1.1 RGB colour space

The colour information of a video frame is normally available in the RGB colour space. This representation is based on an additive model which determines each colour by its combination of red R, green G and

blue B light and is widely used in the video electronics industry. This colour representation possesses properties which makes it easily usable for electronic devices like displays but on the other hand often makes the further processing and visualization of colour data difficult. Disadvantageous properties of the RGB colour space are:

- The high cross correlation between the colour components of $r_{BR} = 0.78$ (cross correlation between blue and red), $r_{RG} = 0.98$ and $r_{GB} = 0.94$ makes the processing of colour data more complicated [TT03].
- The RGB colour space is not very intuitive for humans regarding the visualization of a colour defined by its R, G and B values.
- The similarity or difference between two colours is — due to the human perception — not uniform. Linear changes of a colour signal are therefore not perceived proportionately (*Non-linearity of RGB colour space*).
- The RGB colour model is device dependent since it does not clearly define colours like e.g. "red".[1]

The HSV colour space does not eliminate all the disadvantages of the RGB colour space like device dependency or non-linearity but possesses a few properties which makes it highly suitable for this work. The next section gives a brief description of how the RGB colour space can be transformed to the HSV colour space and what makes the HSV model so suitable for further image processing.

2.1.2 HSV colour space

Already in 1915, the painter Albert Henry Munsell (1858 - 1918) tried to define an accurate system for describing colours by developing a collection of colour samples based on the same perceptual difference of colours. The idea of an intuitive colour space which is adjusted to the human perception is not new to the scientific community and the HSV (hue, saturation, value) colour space, which was introduced in 1978 by Alvy Ray Smith is only one of many [Smi78]. The HSV colour space[2] is a nonlinear transformation of the RGB colour space and defines each colour by three constituent components. It describes each colour by a hue component H, a saturation component S and an intensity component defined by the value V and can be visualised in the form of a colour cone (Figure 2.1). The transformation of a colour in the RGB colour space (R, G, B) defined by its red, green and blue component in the range of 0 and 1 to the HSV colour space is given by

$$H = \begin{cases} 60\left(0 + \frac{G-B}{max(R,G,B)-min(R,G,B)}\right) & if\ R = max(R,G,B) \\ 60\left(2 + \frac{B-R}{max(R,G,B)-min(R,G,B)}\right) & if\ G = max(R,G,B) \\ 60\left(4 + \frac{R-G}{max(R,G,B)-min(R,G,B)}\right) & if\ B = max(R,G,B) \end{cases} \tag{2.1}$$

[1] Different primaries are used for different devices. For devices that emit their own light (i.e. monitors or projectors) or collect light (i.e. scanners, cameras) normally the primaries red, green and blue are used to produce different colours. Devices such as printers where the output relies on reflected light normally use a combination of yellow, cyan and magenta to filter the light that hits the paper.

[2] Note that the HSV colour space is also called the HSB (hue, saturation, brightness) colour space.

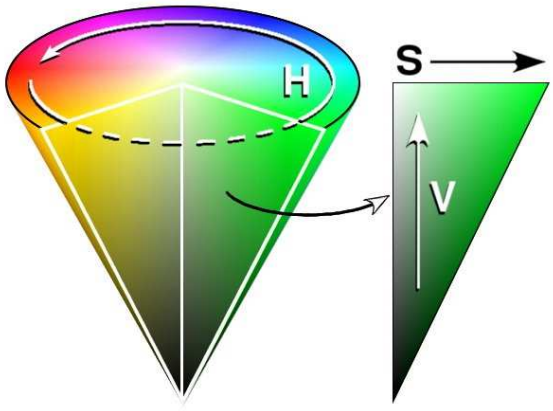

Figure 2.1: HSV colour space

$$S = \frac{max(R,G,B) - min(R,G,B)}{max(R,G,B)} \tag{2.2}$$

$$V = max(R,G,B). \tag{2.3}$$

Note that this leads to two undesirable properties of the HSV colour space: the saturation is undefined if $max(R,G,B) = 0$ and the hue is undefined for $max(R,G,B) = min(R,G,B)$. The hue which is defined in degrees in the range of 0° and 360°, represents the spectral component of a colour with the colour red aligned at 0°, green at 120° and blue at 240°. The value V and the saturation S are both linear values in the range between 0 and 1.

The saturation or *chroma* of a colour can be interpreted as its purity or non-whiteness and is the distance to the centred axis of the cone. Colours with a high saturation have mainly one spectral component and appear vivid whereas colours with a very low saturation are perceived as gray tones. These gray tones are mainly defined by their value component V which is the intensity or brightness of a colour. This intensity is determined by the vertical position of the colour in the HSV space. Colours with intensity $V = 0$ are without brightness and all represent black whereas $V = 1$ represents the brightest colours. Note that black has therefore no unique definition like in the RGB colour space but all colours with an intensity $V = 0$ are projected on black independently of their saturation and hue.

Specific properties of the HSV colour space makes it more suitable than the RGB colour space for processing colour information. As described in [FM99] the HSV colour model clearly separates the intensity information (V) from the chromatic components (H, S). Colour spaces like HSV which represent colour in an intuitive way that can be understood naturally by humans are also called *perceptual colour spaces*.

Next to the better human perception, the hue of the HSV colour space is invariant to certain types of highlights, shadings and shadows and makes shadow removal techniques easy to apply. Since the HSV colour space is a transformation of the RGB colour space, it is also device dependent.

Because of its more natural perceptual properties and clear separation of chroma and luminance the HSV colour model is used in this work. Disadvantages of this colour space [CJSW01] — the discontinuity of the hue component and the singularity and numerical instability for saturation values close to 0 — will be addressed in the Sections 2.2 and 2.3.

2.2 Circular statistics

The algebraic structure of the line and the circle are different and therefore methods of circular data analysis as discussed in [Mar72] must be used when working with directional data. In contrast to the linear domain only one operation, the addition modulo 2π is available in the circular domain. Due to the fact that the circle is a closed curve, its natural periodicity must be taken into account. In this section important definitions in circular statistics are given. Accurate distributions for working with directional data are introduced in Section 2.3.

It is obvious that the arithmetic mean is unsuitable for directional data since the result is very dependent on the choice of origin in the circular domain. Lovell [LKW91] describes a way to represent a set of N angular estimates by N unit phasors[3] with arguments equal to the corresponding angular estimates. The mean angle $\hat{\mu}_p$ is then given by the argument of the phasor sum and this direction is *independent* of the choice of origin. The magnitude of the phasor sum S gives a measure of the concentration about the mean direction and can be used as basis for a definition of the circular variance. If $S = N$ then all estimates are identically equal to the mean. For $S = 0$ no conclusion can be drawn. The general definition of circular mean and variance based on this phasor sum are of the following form.

Definition 1 *Circular Sample Mean and Sample Variance: Let* $\{\hat{\alpha}(k)\}, \hat{\alpha} : \mathbb{Z} \longmapsto \mathbb{R}$ *be a set of* N *observations of a random variable in the circular domain* $[0, 2\pi)$*. Then the circular sample mean* $\hat{\mu}$ *and the circular sample variance* $\hat{V}$ *are defined by*

$$\hat{\mu} = \left(\left(\arg \left[\sum_{k=0}^{N-1} e^{j\hat{\alpha}(k)} \right] \right) \right)_{2\pi} \tag{2.4}$$

and

$$\hat{V} = 1 - \frac{1}{N} \left| \sum_{k=0}^{N-1} e^{j\hat{\alpha}(k)} \right| \tag{2.5}$$

where $(( ))_{2\pi}$ *denotes reduction modulo* 2π *onto* $[0, 2\pi)$*.*

The circular variance $\hat{V}, \hat{V} \in [0, 1]$ cannot be compared directly with its linear equivalent σ^2 which lies in the domain $[0, \infty)$. However by using the relationship between the normal distribution on the circle (wrapped normal, [Mar72]) and the normal distribution on the line a circular standard deviation in the range $[0, \infty)$ can be defined as

$$\hat{\sigma} = \sqrt{-2log_n(1 - \hat{V})}. \tag{2.6}$$

[3] A phasor is a representation for complex numbers which is often used by engineers. A complex number of the form $x + iy$ is represented in the form $x + iy = |z|e^{i\phi}$ where $|z| = \sqrt{x^2 + y^2}$ is the complex modulus and $\phi = tan^{-1}(\frac{y}{x})$ the phase (or complex argument).

All statistical definitions in this work which are used in the context of hue values always refer to the above definitions from circular statistics.

2.3 Colour distributions

A colour model like the RGB or HSV colour space describes colours in an abstract mathematical way, typically by three (RGB, HSV, HSL) or four colour components (CMYK). A background model normally uses multiple sequential frames to determine which pixels are part of the background and what colour they possess. Since the colour values of a pixel in two sequential frames are never completely the same due to e.g. camera sensor noise, electromagnetic disturbance or lighting change, this definition of a colour value by a constant tuple of colours is not completely suitable. A common way to address this problem is to describe each component of a pixel colour by a uni-modal or multi-modal distribution. Using a uni-modal distribution, a colour component is determined by a mean value and a variance which indicates how far from the mean value its values typically are. The variance is also called the *second central moment* and is a measure of the statistical dispersion of a random variable.

Many distributions can be used for describing a colour component of a colour and this section will describe the density functions which are used in this system. Also alternative distributions are investigated and the reasons for not using them are described.

2.3.1 Normal distribution

Since intensity and saturation are linear variables, Gaussian distributions (Figure 2.2) characterized by a mean μ and a variance σ are used for modeling those two channels of a pixel. Note that colours are not Gaussian distributed [SL00] but as shown in numerous works can be well approximated by the standard normal distribution [FM99, WADP97]. The probability of observing a saturation value S at a pixel with a reference distribution $N(\mu_s, \sigma_s)$ for the saturation is therefore given by

$$P(S) = \frac{1}{\sqrt{2\pi}\sigma_s} e^{-\frac{(S-\mu_s)^2}{2\sigma_s^2}}. \tag{2.7}$$

The probability for the intensity is calculated identically.

As noted in [HS67] there exist many reasons why a normal distribution can provide a good description for an unknown random process. The main theoretical justification for the use of normal distributions is due to the central limits theorem. It states that the mean of a large number n of independent observations from any distribution, or even from up to n different distributions, with finite mean and variance approaches a normal distribution. In a completely static background the colour components of a pixel are theoretically constant and variations in the measurements only due to a multitude of small disturbances like camera noise or lighting changes. Therefore the noise should theoretically be normally distributed with mean at the real value of the colour component. However, Hahn states in [HS67] that normal distributions are sometimes not adequate as a physical model because of following reasons:

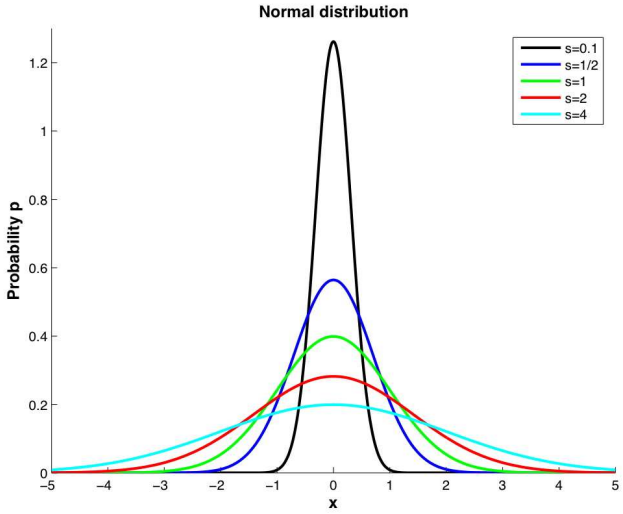

Figure 2.2: Normal distribution

1. It is clear that many variables cannot be reasonably regarded as the sum of many small effects and no reason exist for expecting these variables to be normally distributed.

2. The range of normally distributed variables is minus and plus infinity whereas physical variables often possess upper and lower limits. Errors due to this limited range can be negligible but also can lead to inaccurate results. In particular if the mean is relatively close to the physical boundary of a variable, a symmetrical distribution like the normal distribution is inadequate.

3. For some random variables the normal distribution provides a reasonable approximation in the center, but is not adequate at the tails of the distribution.

Other distributions like the Beta distribution (Figure 2.3) which are defined on a finite domain are probably better suited than the Gaussian for modelling finite colour ranges. In particular for values near the extremes the Beta distribution would probably provide a more accurate description than the Gaussian. By using the Beta distribution $B(\eta_s, \gamma_s)$, the probability for the saturation is

$$P(S) = \begin{cases} \frac{\Gamma(\eta_s+\gamma_s)}{\Gamma(\eta_s)\Gamma(\gamma_s)} S^{\gamma_s-1}(1-S)^{\eta_s-1} & ,0 \leq S \leq 1 \\ 0 & elsewhere, \end{cases} \tag{2.8}$$

where Γ is the gamma function and $\eta_s > 0, \gamma_s > 0$ are the parameters of the Beta distribution for the saturation. The natural finite domain $S \in [0, 1]$ of the Beta function is a significant advantage.

Nevertheless, most cameras have only a limited sensitivity range and do not cope well near the extremes. Since we are mainly interested in accurate modelling of pixels within this camera sensitivity range, the simpler Gaussian distribution is appropriate for saturation and value components. However, for the hue component the Gaussian distribution is inappropriate since circular data behaves quite differently from linear data. Here a von Mises distribution[4] which is the circular equivalent of the Gaussian distribution

[4]This distribution is named after the Austrian mathematician Richard von Mises.

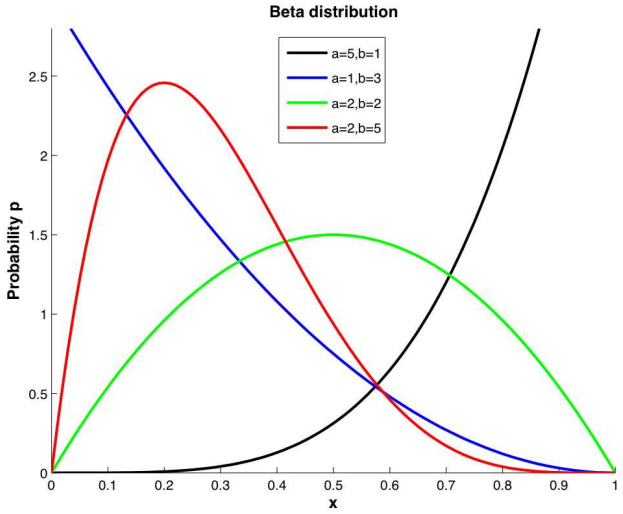

Figure 2.3: Beta distribution

is a suitable density function.

2.3.2 Von Mises distribution

The probability for a hue value H described by a von Mises distribution (Figure 2.4) can be written as

$$P(H) = \frac{1}{2\pi I_0(\kappa_h)} e^{\kappa_h \cos(H-\hat{\mu}_p)} \tag{2.9}$$

where the concentration κ_h and the mean direction $\hat{\mu}_p$ characterize the von Mises distribution. I_0 is the modified Bessel function of the first kind of order 0. Note that the von Mises distribution degenerates into a uniform distribution for $\kappa_h = 0$. Similar in shape to the von Mises distribution is the wrapped Gaussian or wrapped normal distribution [Mar72].

2.3.3 Wrapped normal distribution

The wrapped normal distribution of the form

$$P(H) = \frac{1}{\sqrt{2\pi}\sigma_h} \sum_{k=-\infty}^{\infty} e^{-\frac{(H+2\pi k)^2}{2\sigma_h^2}} \tag{2.10}$$

assumes H to be $N(0, \sigma_h)$ and reduces all observed angles to the interval between $-\pi$ and π. The resulting distribution is circular and wraps the ordinary normal distribution around a circle. This is done by adding all probabilities that fall into the same sector of the circle.

A comparison between the von Mises and the wrapped normal distribution is done in [Lov91, page 67]. It is shown that the wrapped normal distribution is a very accurate approximation to the von Mises

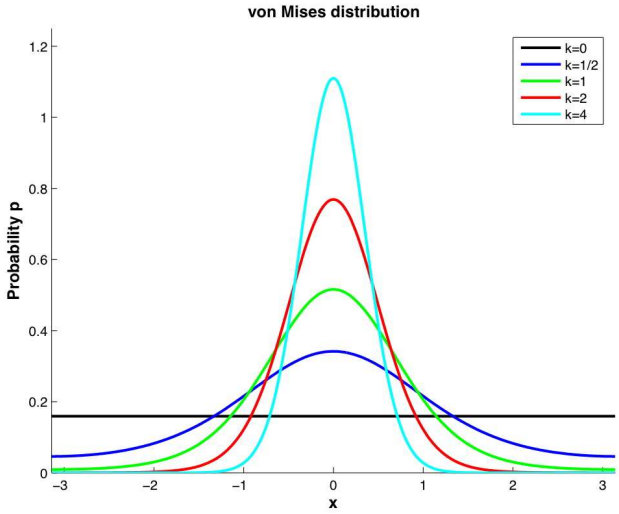

Figure 2.4: von Mises distribution

distribution for moderate SNR (signal-to-noise ratio). For a colour distribution described by a mean μ and a standard deviation σ, the SNR can be defined as

$$SNR = \frac{\mu}{\sigma}. \tag{2.11}$$

For a unimodal, static background we can assume that background changes occur slowly and signal distortions produced by, for example, camera sensors are in a moderate range. Therefore the wrapped normal distribution is an appropriate simpler alternative to the von Mises distribution for modelling the hue values and is used in this system.

Next to its simplicity there are other properties in favor of the wrapped normal distribution which are described in detail in [Mar72]:

- The wrapped normal approximation is just as accurate as the von Mises distribution approximation for moderate SNR (signal-to-noise ratio) [Lov91, page 67].
- The dispersion parameter of the wrapped normal distribution corresponds to the standard deviation of the familiar normal distribution on the line.
- The wrapped normal distribution possesses, in contrast to the von Mises distribution the reproductive property so that the sums and differences of two wrapped normally distributed random variables are also wrapped normally. Therefore, the distribution of the sum or difference of two von Mises distributed circular random variables is often calculated using the wrapped normal approximation [Mar72, page 67].
- The distribution of the mean of a large number of circular random variables approaches a wrapped normal distribution by the central limits theorem on the circle [Mar72, page 90]. Therefore, if the distribution of circular random variables is well described by the wrapped normal distribution, the

distribution of a linear combination of these random variables is even closer to a wrapped normal distribution[5].

Stephens has also shown in [Ste63] that a wrapped normal distribution $\tilde{N}(\mu, \sigma)$ can approximate the von Mises distribution $M(\mu, \kappa)$ over the entire range of the concentration parameter κ reasonably closely.

2.4 Background model

An adaptive background model is used for background subtruction and motion-based foreground selection. A parametric model as used by Francois et al. [FM99] for real-time segmentation of video streams is used. The model operates on the HSV colour space since it clearly separates chromatic and intensity information which makes it suitable for both intensity and colour measurements. Each colour channel of a background reference pixel is modelled as a single and separate distribution since we use a static camera sequence and assume that each pixel of the background can be represented as a single colour (single model). Parametric models based on mixtures of multiple distributions as used in [PT03] or a non-parametric model as introduced by Elgammal et al. [EDD01] can also cope with multi-model backgrounds but are computationally more expensive and would bring no advantage in a single model background.

As described in Section 2.3, each pixel of the background is described by two Gaussians $N(\mu_s, \sigma_s)$ and $N(\mu_v, \sigma_v)$ for saturation and intensity and one wrapped Gaussian $\tilde{N}(\mu_h, \sigma_h)$ for the hue. Initially, the means of all three colour channels of the reference distribution are set to the corresponding values of the pixels in the first frame. Normally each image is disturbed by noises (i.e. sensor noise, atmospheric noise) and therefore a natural colour variance in the colour of a pixel due to these disturbances is always present. To make the models robust against a basic amount of noise the variance for each background distribution is always set above a minimal variance value of $\sigma_{min} > 0$. For all sequences this minimal variance was set manually and a small value of $\sigma_{min} = 3 \cdot 10^{-3}$ provided good results.

After the initialization the model continually performs two main tasks. First the model determines if the pixels in the current frame can be described by the current reference distributions of the background model. If a pixel cannot accurately be described it is labeled as foreground, otherwise as background. Secondly, after the background mask has been generated, the distributions of our background model are updated.

2.4.1 Background generation

For describing the background each pixel has its own model which determines the pixel properties of the background at this position. The hue, the saturation and the intensity of a background pixel are determined by the mean values of the three colour channels $\mu = [\mu_h, \mu_s, \mu_v]'$ and their variances $\sigma =$

[5]The distribution of the sum of a large number of independent circular random variables approaches the uni-variant normal distribution.

$[\sigma_h, \sigma_s, \sigma_v]'$. We decide if a pixel $I = [H, S, V]'$ belongs to the background by comparing this pixel I with the model at the corresponding position in the background model. By thresholding the distance between the pixels and its model we can decide if the pixel belongs to the background or the foreground. For calculating this distance a distance measure δ_{hsv}, which uses all three colour channels independently, is defined:

$$\delta_{hsv}(I, \mu) = \begin{pmatrix} \delta_h \\ \delta_s \\ \delta_v \end{pmatrix} = \left| \begin{pmatrix} \measuredangle(H, \mu_h) \\ S - \mu_s \\ V - \mu_v \end{pmatrix} \right|. \tag{2.12}$$

The circular domain of the hue is taken into account when computing the hue difference δ_h. We make the simplified assumption that the colour channels are independent of each other to reduce computational complexity. Porikli observed in [PT03] that this assumption degrades the quality of the results only minimally. If, for a pixel at position x, the difference for one of the channels is larger than a foreground threshold $\lambda_{\{h,s,v\}}(x)$ the pixel is marked as foreground $B(x) = 0$, otherwise it is labelled as foreground $B(x) = 1$.

$$B(x) = \begin{cases} 1 & if\ \delta_{hsv}(I(x), \mu(x)) < \lambda(x) \\ 0 & otherwise. \end{cases} \tag{2.13}$$

The threshold $\lambda_{\{h,s,v\}}$ depends on the variance of the corresponding colour channel

$$\lambda_{\{h,s,v\}}(x) = 2\sigma_{\{h,s,v\}}(x). \tag{2.14}$$

The range of 2σ is equivalent to a 95.5% confidence interval for a standard normal distribution. Since colour information is not Gaussian distributed [SL00] we can still expect each colour value to lie in the interval $[\mu - 2\sigma, \mu + 2\sigma]$ with a confidence of at least 75% by applying Tchebychev's Inequality theorem. This theorem states that for *any distribution* with finite mean and variance, at least $[1 - (\frac{1}{k^2})]$ times 100 percent of the probability is in the range $\pm k\sigma$ around the mean [HS67, page 45].

For reliable computation of the hue difference we have to test if the saturation value $S(x)$ of the frame pixel or the mean $\mu_s(x)$ of the reference distribution is close or equal to 0. Pixels with saturation equal to 0 are in the achromatic range of the HSV colour space. In this range the pixel lies on the central line of gray values and its hue information is meaningless and not usable as a distance measure. We define a pixel as achromatic if its saturation lies below a saturation threshold $\lambda_{achr_s} = 0.2$. According to this we only use the reliable channels of the frame pixel and the reference distributions for comparison and distinguish between four cases:

1. if $S < \lambda_{achr_s}$ and $\mu_s < \lambda_{achr_s}$, check $\delta_v < \lambda_v$.
2. if $S > \lambda_{achr_s}$ and $\mu_s < \lambda_{achr_s}$, check $\delta_v < \lambda_v$ and $|S - \frac{\mu_s}{\mu_v}| < \lambda_s$.
3. if $S < \lambda_{achr_s}$ and $\mu_s > \lambda_{achr_s}$, check $\delta_v < \lambda_v$ and $|\frac{S}{V} - \mu_S| < \lambda_s$.
4. if $S > \lambda_{achr_s}$ and $\mu_s > \lambda_{achr_s}$, check $\delta_v < \lambda_v$, $\delta_h < \lambda_h$ and $|S\cos(\measuredangle(H, \mu_h))| < \lambda_s$.

In the first case no useful colour information is available and therefore only the intensity is used to measure the distance between the pixel and colour distribution. As shown in [FM99] the saturation can be scaled by the intensity to reflect the uncertainty of the colour information for lower intensity values (Case 2 and

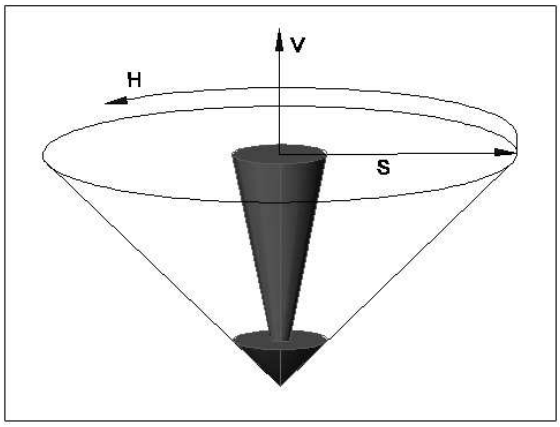

Figure 2.5: Separation of HSV colour space

3). In Case 4 the reference pixel as well as its reference distributions lie in the chromatic range and all channels can be used as distance measures. In this case the saturation is projected onto the mean hue direction. The allocation of each pixel to an achromatic or chromatic range is visualised in Figure 2.5. The gray area represents the achromatic part of the HSV colour space, the rest of the cone the chromatic range.

one of the four cases can be seen in .

For low brightness values the saturation component is unreliable and we use an additional threshold $\lambda_{achr_v} = 0.2$ for the intensity[6]. We add all pixels with an intensity $V < \lambda_{achr_v}$ to Case 1 since this intensity range of the HSV colour space represents the nearly black pixels with strong achromatic properties. The gray region represents the range where only the intensity is used. The rest of the cone is the colour range where also the saturation and the hue are used for comparison.

2.4.2 Update background model

After the pixels $I(t)$ in the current frame have been labelled as foreground or background, the colour distributions of all reference pixels are updated by

$$\mu(t) = [1 - \alpha]\mu(t-1) + \alpha I(t) \tag{2.15}$$

and

$$\sigma^2(t) = [1 - \alpha]\sigma^2(t-1) + \alpha[\mu(t) - I(t)]^2. \tag{2.16}$$

Here α is the learning rate which defines how quickly old frames are forgotten. A minimal standard derivation σ_{min} is introduced, which prevents the decreasing of the standard deviation $\sigma(t)$ below a minimal value. This is useful in a long period of time when the background remains constant. As in the background generation step, only those channels of a pixel which contain useful information to support

[6]Hanbury and Serra [HS02] shows that this brightness dependence can easily be removed by removing the brightness normalisation from the saturation expression (Equation 2.2). The saturation is computed as $S = max(R, G, B) - min(R, G, B)$.

Figure 2.6: Original frame

an update are updated. In the case where a reference pixel and a frame pixel are both in the achromatic range (Case 1) and no useful colour information is available, only its intensity distribution is updated.

2.5 Implementation

By using the background model described above, each pixel of a frame is assigned to one of the 4 cases described in Section 2.4.1. A frame before the background segmentation step is shown in Figure 2.6.

The frame is then separated into a group mask as shown in Figure 2.7. The red regions are pixels assigned to Case 1 where only intensity information can be used. Green pixels and blue pixels represent Case 2 and Case 3 respectively. In the white regions (Case 4) hue information is available in the background model as well as in the current frame and saturation and hue provide useful information. The intensity is used in all four cases. The large number of red pixels (Case 1) in the group mask is due to the large number of nearly gray pixels in the original image which do not provide good hue or saturation information. The reason for the large number of gray pixels is probably that this video sequence was recorded in winter time where less sunlight is present and at an urban location where the streets or concrete buildings normally have a characteristic gray colour. Additional reasons for achromatism in frames could be due to camera characteristics or filters in post-processing steps.

After the assignment of each pixel to one group, the background generation step now compares the colour of each pixel with the current background model and assigns it to background or foreground. The resulting foreground mask is shown in Figure 2.8.

The background model performed well on all test sequences. Problems occurred only when the frame resolution was too small or a high compression for the frames was used. Here – due to compression

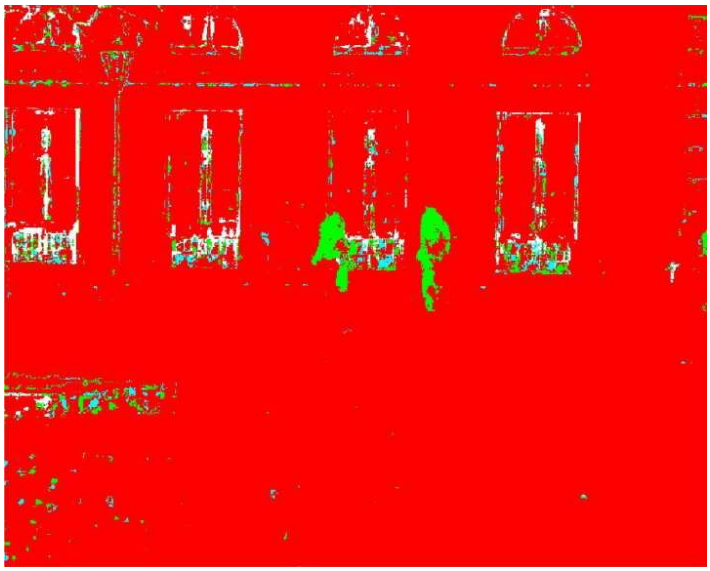

Figure 2.7: Four cases of colour reliability

Figure 2.8: Foreground mask

artifacts or the scaling filters – a lot of noise and disturbances were present in the image and the model could not explain the background distributions sufficiently.

Chapter 3

Pedestrian Detector

There are two ways of constructing a software design; one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

C. A. R. Hoare

This tracking system makes predictions about possible pedestrian locations in each video frame. These predictions are based on the available detections of pedestrians in the frames of the video sequence. This fundamental part of our system finds pedestrian-like patterns in an image – independent of their size and location. This chapter describes the theory behind this detector in detail and takes a closer look at its implementation. For better understanding, a brief overview in Section 3.1 describes important terms used in this area of image processing. Section 3.2 describes the features used for the pedestrian classification and how they can be computed quickly and with illumination correction. How we use these features for the classification process is described in Section 3.3. Section 3.4 takes a closer look at how these features can be applied at various scales and positions in a frame.

3.1 Terminology

In image processing the same words are often used to describe different facts and properties. For a better understanding a short description of the terminology used in this work is provided in this section.

The term *detection rate D* always refers to the percentage of positives which are classified correctly by the detector from the total number of positives. The *false positive rate F* is the percentage of negatives,

which are incorrectly classified as positives. Good detection rate D or low false positive rate F alone do not necessarily indicate a good detector performance. If the detection rate D and the false positive rate F of a detector are both very high or very low this means that the detector cannot distinguish between positive and negative patterns very well. In the first case it will classify nearly everything as positive whereas in the second case, the detector will not classify as many negatives as positives but neither will it recognize positives very well. Therefore only both terms together can describe the *performance* or *effectiveness* of a detector in a meaningful way.

When we speak of an *image* or *frame* we always refer to a complete and unresized frame in the video sequence. If we want to access a smaller part of the image we refer to *sub-images* in the frame within a rectangular-shaped *window*. This sub-image is a standard image but represents only a smaller part or sub-region of the frame. Each window is simply determined by an upper-left and lower-right corner. When we work with a window it should be clear from the context if we want to access the geometric information of the window or the colour information of the referenced sub-image pixels.

The *detector* or *pedestrian detector* described in this chapter refers to a classifier which only works on images and classifies them either as pedestrian or as non-pedestrian. Also sub-images can be classified by the detector since they are standard images inside a larger image. If we write that the detector classifies a window it means that the detector indirectly classifies the referenced sub-image. Another important term is a *pedestrian detection* or *detection*. These terms always refer to windows in the current frame whose referenced sub-images were classified as pedestrian by the detector.

3.2 Features

Every classifier working on an image has to use some kind of feature to determine the class of this image. The feature can simply be a single pixel but often more complex features like edges, pixels regions or shapes are used. In this detector we use Haar-like features (Section 6.3.3) for describing a pedestrian in an image. By using integral images (Section 3.2.2) we can quickly compute these features independent of their size and can apply an illumination normalization (Section 3.2.3) on the feature output. A feature calculation is always done on an image and therefore if applied on a window it indirectly uses the referenced sub-image.

3.2.1 Haar-like features

In this detector, Haar-like features [VJ01a] are used for the classification. Each feature uses a set of two or more pixel filters r where each filter is defined by a rectangular shaped area $\Im((p_1, q_1), (p_2, q_2))$. Here (p_1, q_1) and (p_2, q_2) are the upper-left corner and the lower-right corner of the rectangular area. The response $r(I)$ of a filter r for an image I is simply the sum of pixel intensity values in the area $\Im$:

$$r(I) = \sum_{x=p_1}^{p_2} \sum_{y=q_1}^{q_2} I(x, y) \tag{3.1}$$

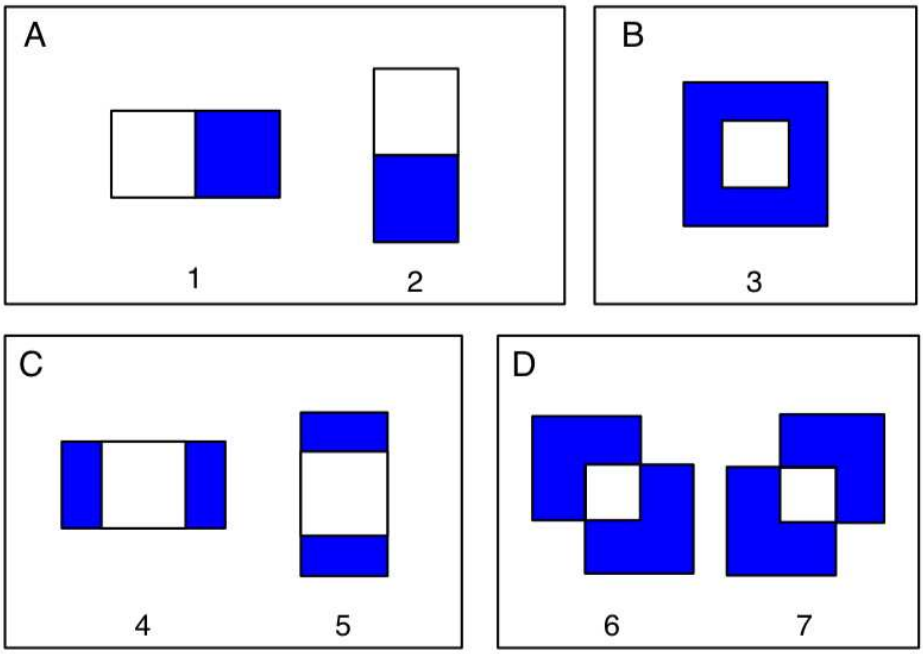

Figure 3.1: Feature types

Since all pixel intensity values are larger than or equal to zero, the filter response r is also always larger than or equal to zero. Section 3.2.2 describes how those rectangle area sums can be computed very quickly using integral images.

Each Haar-like feature ρ consists of a set of N_ρ pixel filters $r_{1..N_\rho}$ with corresponding filter responses $r_{1..N_\rho}(I)$ and orientations $s_{1..N_\rho} \in \{-1, 1\}$. The orientation or sign s_i works as a signum function and turns the positive filter response r_i into a positive or negative output. Typically each Haar-like feature has at least one filter with a contrary orientation to the rest of its filters. The output $\rho(I)$ of the feature ρ for an image I is a linear combination of weighted filter responses

$$\rho(I) = \sum_{k=1}^{N_\rho} s_k r_k(I),\ s_k \in \{-1, 1\} \tag{3.2}$$

In this detector, seven static feature types are used (Figure 3.1). Motion based features as used in [CD99, VJS03] were not used in this implementation because of their higher computational complexity compared to static features.

In Figure 3.1, the filled and the white regions of the features represent rectangles with different orientation s. Therefore, the sum of the responses of all filled rectangles has the opposite sign to the sum of the white rectangle responses. The features in box A are sensitive to horizontal and vertical edges. Feature type 3 focuses on local context comparing a center and its surrounding information. The types 4 and 5 are good for describing horizontal and vertical line features and are often used for the relation between

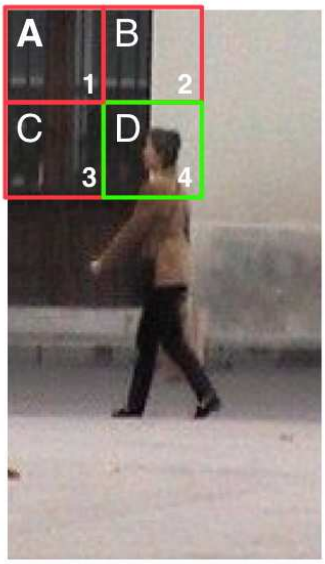

Figure 3.2: Integral Image

eyes and nose or head and background. Instead of the additional use of rotated features as proposed by Lienhart [LM02, LKP03] the feature types in box D for non-horizontal and non-vertical lines are introduced which should also cover a similar range to rotated features. In contrast to rotated Haar-like features no additional computation of the rotated integral image is necessary for the feature types 6 and 7. Both features focus on non-horizontal and non-vertical edges.

The coordinates and size values of all feature filters are always normalized to the range $[0, 1]$ and in relation to the windows where the features are applied. This feature normalization is useful when the features are used at different scales as is done in the image pyramid (Section 3.4.1). In this image pyramid we work with windows of different sizes and their referenced sub-images. Before a feature is applied it is scaled to the size of the window. A feature rectangle with a width of 0.5 means that the scaled feature covers half of the window width.

3.2.2 Integral Image

Since the filters described in Section 6.3.3 build the basic elements of the detector, it is important to compute them very quickly and effectively. A fast way to compute rectangle sums independent of the size of the rectangle are integral images [VJ01b]. Since the filters in this detector use rectangle sums, integral images can be used for computing the filter responses.

An integral image II of an image I can be computed as

$$II(p,q) = \sum_{x=1}^{p} \sum_{y=1}^{q} I(x,y) \tag{3.3}$$

where $I(x, y)$ is the pixel intensity value in image I at position (x, y). Each value $II(p, q)$ in the integral image II at position (p, q) represents the sum of the pixel intensity values in the rectangular area between the upper-left corner and the position (p, q) in the image I.

The intensity sum (i.e. the filter response $r(I)$) of a rectangular area $\Im((p_1, q_1), (p_2, q_2))$ therefore can be computed independently of the rectangle size with 3 arithmetic operations

$$r(I) = II(p_2, q_2) - II(p_1, q_2) - II(p_2, q_1) + II(p_1, q_1) \tag{3.4}$$

An example is shown in Figure 3.2 in which we aim to get the sum of the green rectangle D. The 4 rectangle sums are labelled with 1 to 4 where the rectangle sum 2 includes the sum of grayvalues of rectangle A and B. Rectangle sum 3 is the sum of rectangle A and C and rectangle sum 4 contains all pixel values of the rectangles A to D. First rectangle 2 and 3 are subtracted from rectangle 4. Area 1 was subtracted twice and has to be added again to get the sum of rectangle D.

The integral image II of an image I can be computed efficiently by first cumulatively summing the columns / rows and then cumulatively summing the row sums / column sums. A more detailed description of computing integral images can be found in [LM02].

3.2.3 Illumination correction

To minimize the effect of different lighting conditions, all images are variance normalized before the image is passed to the detector cascade for classification. As demonstrated in [LM02, VJ01a, VJ01b], the mean μ

$$\mu = \frac{\sum I(x, y)}{size(I)} \tag{3.5}$$

of all pixel intensity values in an image I can be calculated very quickly by using the integral images again. Here $size(I)$ is the number of pixels of image I. The variance σ is determined as

$$\sigma^2 = \mu^2 - \frac{1}{size(I)} \sum I(x, y)^2. \tag{3.6}$$

For calculating the sum of the squared pixel intensity values $I(x, y)^2$ in image I, a second integral image II_2 of image I squared is calculated (e.g. 2 integral images of the original image are used in the whole process):

$$II_2(p, q) = \sum_{x=1}^{p} \sum_{y=1}^{q} I(x, y)^2 \tag{3.7}$$

By using the integral image and the squared integral image we can compute the mean and the variance of an image independent of its size. As also shown in [VJ01a] it is possible to apply the illumination

correction as a post processing step after calculating the filter responses of a feature. Instead of variance normalizing each single intensity value in an image I the normalization takes place after the computation of the filter responses which requires only a few arithmetic operations. For an image I with mean μ and variance σ we can normalize a filter with response

$$\rho(I) = \sum_{k=1}^{K} s_k r_k(I),\ s_k \in \{-1, 1\} \tag{3.8}$$

as

$$\begin{aligned} \rho(I) = & \sum_{k=1}^{K} s_k \frac{|r_k(I) - size(r_k)\mu|}{\sigma} \\ = & \frac{1}{\sigma} \sum_{k=1}^{K} s_k \left| r_k(I) - size(r_k)\mu \right| \end{aligned} \tag{3.9}$$

For fast computation we use only features which consist of two rectangle filters. The normalized output of a feature is of the form

$$\rho(I) = \frac{1}{\sigma} \sum_{k=1}^{2} \frac{\omega_k}{\sum \omega_k} s_k \left| r_k(I) - size(r_k)\mu \right|. \tag{3.10}$$

where ω_k is a weight which is assigned to each filter to make its output size independent. This is necessary to make a comparison between differently sized filters as used by feature type 3 (Figure 3.1) possible. In our implementation the weights and orientations for the features with two filters r_1 and r_2 are set to

$$\omega_1 = 1, \quad \omega_2 = \frac{size(r_1)}{size(r_2)}$$

and

$$s_1 = 1,\ s_2 = -1.$$

Here ω_i and s_i are the weight and the orientation of filter r_i. This leads to the final equation for the variance normalized feature response of

$$\begin{aligned} \rho(I) = & \frac{1}{\sum \omega_k \sigma} \sum_{k=1}^{2} \omega_k s_k \left| r_k(I) - size(r_k)\mu \right| \\ = & \frac{1}{2\sigma} \left[\left| r_1(I) - size(r_1)\mu \right| - \left| r_2(I) \frac{size(r_1)}{size(r_2)} - size(r_1)\mu \right| \right] \end{aligned} \tag{3.11}$$

which can be computed with a few arithmetic operations.

3.3 Classification

The features described in the last section simply provide a feature output in form of a floating point number. A classifier now has to make the class assignment according to this feature output. A weak classifier is the simplest form of classifier used in this work and described in Section 3.3.1. More complex and powerful classifiers are designed in Sections 3.3.2 to 3.3.5.

3.3.1 Weak classifier

A classifier c uses a single Haar-like feature ρ for the classification. Each classifier c has a threshold λ_c which is used as a bias for the classification. A classifier c determines the class $c(I)$ of an image I as

$$c(I) = \begin{cases} 1 & \text{if } \rho(I) \geq \lambda_c \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

Here $\rho(I)$ is the response of feature ρ for image I. If $\rho(I)$ is smaller than the threshold λ_c, classifier c labels the image I as negative (i.e. non-pedestrian, class 0). Otherwise it is classified as positive (i.e. pedestrian, class 1).

The threshold λ_c of a classifier c with feature ρ is defined in such a way that the classifier has the minimal classification error ϵ_c on the training set. This training set consists of N_T training samples $x_{1..N_T}$ with corresponding class $y_{1..N_T} \in \{0, 1\}$. Since AdaBoost (see Section 1) does not consider all training samples equally but assigns image weights $w_{1..N_T}$ to them, a simple mean of the feature responses on the training set does not suffice to estimate the threshold λ_c. Therefore, for calculation of the threshold λ_c, univariate quadratic discriminant analysis as proposed in [DHS00] is used. For calculating the threshold λ_c for classifier c, first a negative threshold λ_{neg} for all negative training samples x_i with $y_i = 0$ and image weight w_i is calculated.

$$\lambda_{neg} = \frac{\sum w_i \rho(x_i)}{\sum w_i} \tag{3.13}$$

The same computation step is done for the positive threshold λ_{pos} with the positive samples x_j where $y_j = 1$. Finally the overall threshold λ_c of a weak classifier c is calculated as the mean of the negative and positive threshold λ_{neg} and λ_{pos}.

$$\lambda_c = \frac{\lambda_{neg} + \lambda_{pos}}{2} \tag{3.14}$$

We now have a quite simple classifier based on rectangle filters which – because of their simplicity – are also called *weak classifiers*. Since a pedestrian is a quite complex object it is reasonable that a single weak

classifier will not be powerful enough to describe it. Therefore we combine many of these weak classifiers into one "super-classifier" by using a technique called *boosting* which is described in the next section.

3.3.2 Boosting

As in the detector of Viola and Jones [VJ01a], a boosting algorithm is used in this implementation. Boosting [DHS00] is widely used in the field of pattern classification. It is the idea of letting multiple and simple (or *weak*) classifiers decide a classification task by a majority vote. For boosting, any kind of learning algorithm like SVMs, neural networks or decision trees can be used. This implementation uses weak classifiers based on Haar-like features for building *boosted classifiers*.

A boosted classifier C uses a set of N_C weak classifiers $c_{1..N_C}$ (Section 3.3.1). The output of each weak classifier $c_i(x)$ is weighted with a weak classifier weight α_i where $\sum_{i=1}^{N_C}\alpha_i = 1$ and $\sum_{i=1}^{N_C}\alpha_i c_i(x) \in [0,1]$. For deciding on the class $C(x)$ of a sample x the sum of all weighted outputs is used. If this sum is above the threshold λ_C of the boosted classifier C, the sample x is classified as positive, otherwise as negative.

$$C(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^{N_C}\alpha_i c_i(x) \geq \lambda_C \\ 0 & \text{otherwise} \end{cases} \tag{3.15}$$

In this implementation a threshold $\lambda_C = \frac{1}{2}$ was taken during the training, which means that if all classifier weights are equal, a sample has to be classified by at least 50% of the weak classifiers as positive to belong to class 1.

A boosted classifier already achieves good results but for difficult classification problems a way for combining multiple boosted classifiers exists which is described in the next section. It splits the main problem into smaller problems and trains one boosted classifier for each of these sub-problems.

3.3.3 Adaptive boosting

The common boosting process described in the last section leads to a good classifier, which consists of a set of weak classifiers. This boosted classifier normally provides a better detection rate as well as a lower false positive rate than each of its weak classifiers. A major problem in many classification tasks is that with increasing detection rate, the false positive rate normally also increases. For hard classification problems a boosted classifier hardly achieves a high detection rate together with a low false positive rate. Adaptive boosting (AdaBoost) tries to overcome this problem by specially training and combining several boosted classifiers.

The idea behind adaptive boosting is to create N_{BC} boosted classifiers $C_{1..N_{BC}}$, where each of them concentrates on reaching a high detection rate ($D_{1..N_{BC}} \approx 1$) and only an average false positive rate ($F_{1..N_{BC}} \leq$

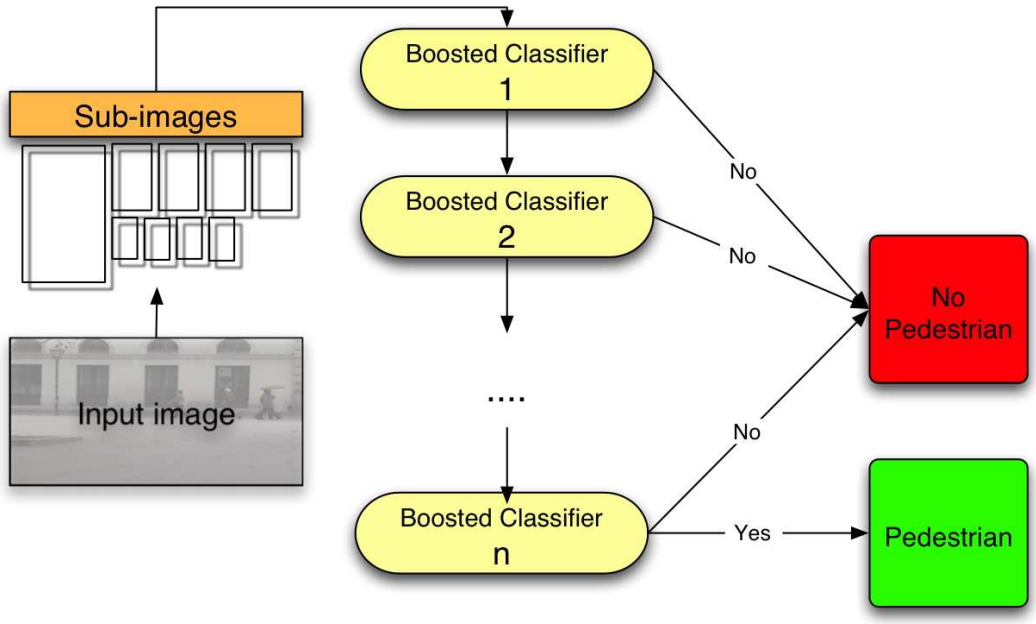

Figure 3.3: Adaptive Boosting with a cascade of classifiers

0.5) on a training set. We do not try to address both problems (low false positive rate + high detection rate) at the same time but only one problem – a high detection rate for each boosted classifier. The N_{BC} boosted classifiers are then connected together in a *classifier cascade* (see Figure 3.3). This cascade contains N_{BC} *classifier stages* with one boosted classifier per stage. Since each of the classifiers $C_{1..N_{BC}}$ in this cascade has a detection rate $D_{1..N_{BC}} \geq 0.99$ and a false positive rate $F_{1..N_{BC}} \leq 0.5$, the classifier cascade with N_{BC} stages (=N_S boosted classifiers) has a final detection rate D and a false positive rate F of

$$D = \prod_{i=1}^{N_{BC}} D_i \tag{3.16}$$

$$F = \prod_{i=1}^{N_{BC}} F_i \tag{3.17}$$

With each additional boosted classifier in the cascade the detection rate as well as the false positive rate decrease. Since in this structured classification process the detection rate decreases much more slowly then the false positive rate we can achieve a good detection rate as well as a low false positive rate. The term *classifier cascade* in this work can be understood as one super-classifier which has the function to decide if an image is a pedestrian or a non-pedestrian.

At the beginning of a classification process, as shown in Figure 3.3, the detector divides an image into multiple sub-images (see Section 3.4.1). Each sub-image is classified by all boosted classifiers in the classifier cascade and if rejected by one of them, classified as negative or non-pedestrian. If a sub-image passes through all stages (i.e. accepted as a pedestrian by all classifiers), it is labelled as positive.

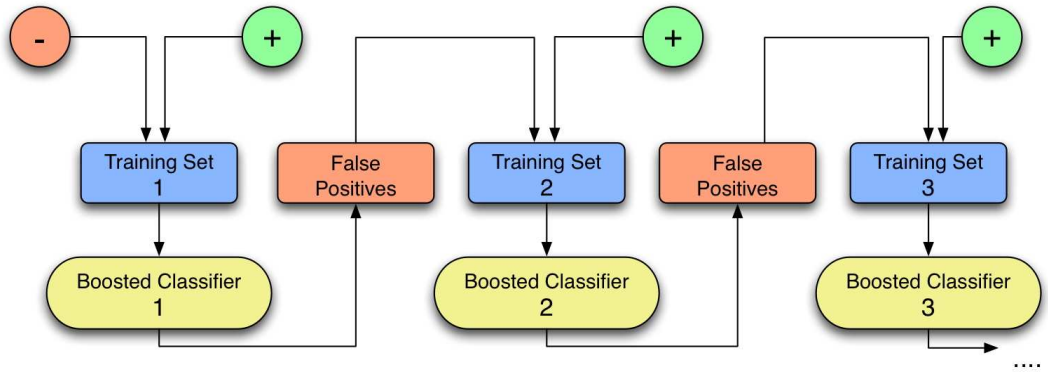

Figure 3.4: Each classifier is trained with the same set of positives and with negatives wrongly classified by previous classifiers

After the complete training, every boosted classifier C_i of the classifier cascade should have a high detection rate and will therefore classify nearly all positives correctly. A problem occurs if multiple classifiers reject an equal set of samples as negatives. Since each additional classifier in the cascade has the function of rejecting additional negatives, it does not make sense to use multiple classifiers in the cascade for rejecting the same samples. They would only complete the same classification twice without improving the false positive rate but only decreasing the detection rate. If all cascade classifiers are (as in the worst case) completely equal, each classifier will reject and pass on the same samples and the cascade will bring no improvement at all but will only increase computation time and lower the detection rate. To avoid the scenario above, the AdaBoost algorithm tries to create a cascade, where each of the cascade classifiers rejects different negatives to its previous classifiers. This is done by training each new classifier in the cascade with negatives which were classified incorrectly by the previous classifiers (Figure 3.4).

For the creation of the negative samples bootstrapping [DHS00] is used. The *Bootstrap* algorithm randomly selects subsets of a given negative training set, and uses them as new training patterns. Randomly chosen areas of these images serve as new negative samples and provide a nearly unlimited number of negatives for the training process.

The first boosted classifier C_1 is trained on all positive examples and on a set of negatives until it achieves a detection rate $D_1 \approx 1$ and a false positive rate $F_1 \leq 0.5$. Each additional classifier in the cascade is trained with the same set of positives but with negatives not classified correctly by all previous boosted classifiers. Therefore before training a new classifier C_{i+1}, negatives are created with bootstrapping and classified by the existing classifiers $C_{1..i}$. If a negative is classified as positive by all classifiers $C_{1..i}$ it is added to the negative training set of classifier C_{i+1}.

3.3.4 Boosting algorithm

The creation of a good boosted classifier C is a quite complex task and done by a boosting algorithm. This boosting algorithm can be regarded as a kind of feature selection algorithm, since it selects the

features for the weak classifiers $c_{1..N_C}$ of a boosted classifier. Additionally it assigns the weak classifier weights $\alpha_{1..N_C}$ to each weak classifier which defines the influence of this weak classifier in the boosted classifier.

There are many versions of the boosting algorithm like Discrete AdaBoost [VJ01a], GentleBoost [FHT01, LKP03], LogitBoost [FHT01] or Real AdaBoost [FHT01]. They all differ in the way they select features and weights for the boosted classifiers and therefore have strong differences in computational complexity and training time as well as in size and efficiency of the resulting (boosted) classifier. Instead of the symmetric Discrete AdaBoost algorithm (see Algorithm 1) used by Viola and Jones [VJ01a, VJS03, VJ01b], an *asymmetric* version of this algorithm was used and combined with the FloatSearch algorithm used in FloatBoost [LZSZ02].

All AdaBoost versions have in common that they assign image weights $w_{1..N_T}$ to the training samples $x_{1..N_T}$ at the beginning of the boosting process (Algorithm 1). The weights are used to determine the error of the weak classifiers on the training set and to select the weak classifier c_i with the smallest error ϵ_i. During the training the sample weights are re-weighted according to the error of the selected weak classifier on the training set. If a sample is classified correctly, its weights (and therefore its influence on the training error) is lowered, otherwise increased. The algorithm constantly tries to eliminate samples which are highly weighted and therefore badly classified. The weak classifier weight α_i of a weak classifier c_i is chosen according to its error on the training set and determines its influence in the boosted classifier.

The Discrete AdaBoost [FS96, VJ01a] algorithm uses a symmetric error function to compute the error or the fitness of a weak classifier. In this context the term symmetric means that positive and negative samples are regarded as equally important during the training. To use a symmetric function and to minimize the error on the complete training set makes sense if one boosted classifier alone decides on the class of a sample. If the error is minimized, the false positive rate F and the detection rate D are also optimized.

In the system described in this thesis, multiple boosted classifiers are used within an adaptive boosting scheme (see Section 3.3.3). For this structure of combined boosted classifiers an average false positive rate $F_C \approx 0.5$ and a high detection rate $D_C \approx 1$ are more accurate (Viola [VJ02], Wu [WRM04]) and an asymmetric error loss function results in a smaller and more efficient boosted classifier within less training time. The asymmetric error function in the boosting algorithm tries to create boosted classifiers with only an average false positive rate. The reason for this is that if one boosted classifier does not reach $D \approx 1$ the detection rate of the classifier cascade will decrease significantly. If the detector misses the false positive rate of $F \leq 0.5$ the consequence will be a less selective classifier. AdaBoost compensates for this less selective classifier, since it adds boosted classifiers to the cascade until it reaches a final goal for detection rate and false positive rate. The asymmetric version of AdaBoost therefore focuses primarily on achieving the high detection rate and then on the other hand on decreasing the false positive rate. For this reason we substituted the error loss function from line 3*b* in Algorithm 1 by an asymmetric cost function:

$$\epsilon_j = \omega_P \frac{\sum_{i=1}^{N_{T,pos}} w_i \mid c_j(x_i) - y_i \mid}{\sum_{i=1}^{P} w_i} + (1 - \omega_P) \frac{\sum_{k=1}^{N_{T,neg}} w_k \mid c_j(x_k) - y_k \mid}{\sum_{k=1}^{N} w_k} \tag{3.18}$$

Algorithm 1 Discrete AdaBoost [VJ01a]

1. Given a training set $(x_1, y_1), ..., (x_{N_T}, y_{N_T})$ where x_i are the training samples and $y_i = 0, 1$ the corresponding class (0 = negative, 1 = positive).
2. Initialize weights $w_{1,i} = \frac{1}{2N_{T.pos}}, \frac{1}{2N_{T.neg}}$ for $y_i = 0, 1$ respectively, where $N_{T.pos}$ and $N_{T.neg}$ are the number of negatives and positives respectively.
3. For $m = 1, ..., N_C$
   (a) Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{N_T} w_{t-1,j}}$, so that w_t is a probability distribution.
   (b) For each feature, ρ_j, train a weak classifier c_j which is restricted on using a single feature. The error is evaluated with respect to the weights $w_{1..N_T}, \epsilon_j = \sum_{i=1}^{N_T} w_i \left| c_j(x_i) - y_i \right|$.
   (c) Add the classifier c_t with lowest error ϵ_t to the strong classifier C.
   (d) Update the weights, $w_{t+1,i} = w_{t,i}\beta_t^{1-\epsilon_i}$, where $\epsilon_i = 0$, if sample x_i is classified correctly and $\epsilon_i = 1$ otherwise, the weight correction is $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
   (e) If $D_C \geq 0.99$ and $F_C \leq 0.5$ than stop boosting (classifier C is good enough). Otherwise continue. D_C and F_C are the detection and the false positive rate of classifier C.
4. The final strong classifier is

$$C(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{M} \alpha_t c_t(x) \geq \lambda_C \sum_{t=1}^{M} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$

   where $\alpha_t = log\frac{1}{\beta_t}$ is the weight and $\lambda_C = \frac{1}{2}$ is the threshold of classifier C.

This AdaBoost version is called the *Asymmetric AdaBoost* algorithm. In our implementation an error ratio of $\omega_P = \frac{2}{3}$ for positive samples is used. Misclassifying a positive sample increases the error ϵ_j more than misclassifying a negative one with the same weight. This error function prefers a fast increasing detection rate to a slow decreasing false positive rate. Note that the asymmetric error function still uses the sample weights of the training samples but penalizes a false classification of positives stronger than one of negatives.

After each iteration of the Asymmetric AdaBoost algorithm a backtrack mechanism (Float Search) tries to minimize the classification error directly and not – as during one iteration of the Discrete AdaBoost – by optimizing an exponential function of the margin. This decreases the number of weak classifiers but leads to a higher training time. The resulting *Asymmetric FloatBoost* algorithm used during the training of our detector can be seen in Algorithm 2.

Algorithm 2 Asymmetric FloatBoost

1. Given example images $(x_1, y_1), ..., (x_{N_T}, y_{N_T})$ where x_i are the training samples and $y_i = 0, 1$ the corresponding class (0 = negative, 1 = positive).
2. Initialize weights $w_{1,i} = \frac{1}{2N_{T.pos}}, \frac{1}{2N_{T.neg}}$ for $y_i = 0, 1$ respectively, where $N_{T.pos}$ and $N_{T.neg}$ are the number of negatives and positives respectively.
3. For $m = 1, ..., N_C$
   (a) Normalize the weights, $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{N} w_{t-1,j}}$, so that w_t is a probability distribution.
   (b) For each feature, ρ_j, train a weak classifier c_j which is restricted on using a single feature. The error is evaluated with respect to w_t,
   $$\epsilon_j = \omega_p \frac{\sum_{i=1}^{N_{T.pos}} w_i \mid c_j(x_i) - y_i \mid}{\sum_{i=1}^{P} w_i} + \omega_n \frac{\sum_{k=1}^{N_{T.neg}} w_k \mid c_j(x_k) - y_k \mid}{\sum_{k=1}^{N} w_k}.$$
   (c) Add the weak classifier c_t with lowest error ϵ_t to the strong classifier C.
   (d) Find weak classifiers $c_i \in C, i \in \{1..m-1\}$ where $error(C \backslash c_i) = min$. C is the current boosted classifier and m the current number of weak classifiers in C. $C \backslash f_i$ is the classifier C without the weak classifier c_i.
   (e) If $error(C \backslash c_i) < error(C)$ erase c_i from C, decrease m by 1 and go back to step $3d$. Otherwise continue with step $3f$.
   (f) Update the weights, $w_{t+1,i} = w_{t,i}\beta_t^{1-\epsilon_i}$, where $\epsilon_i = 0$, if sample x_i is classified correctly and $\epsilon_i = 1$ otherwise, the weight correction is $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$.
   (g) If $D_C \geq 0.99$ and $F_C \leq 0.5$ than stop boosting (classifier C good enough). Otherwise continue.
4. The final strong classifier is
   $$C(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{M} \alpha_t c_t(x) \geq \lambda_C \sum_{t=1}^{M} \alpha_t \\ 0 & \text{otherwise} \end{cases}$$
   where $\alpha_t = log\frac{1}{\beta_t}$ is the weight and $\lambda_C = \frac{1}{2}$ is the threshold of classifier C.

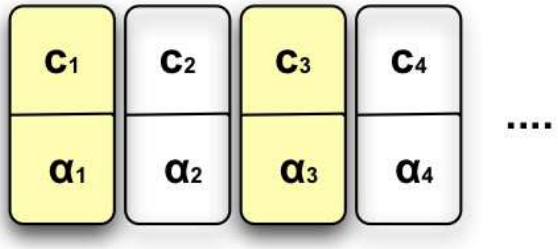

Figure 3.5: Normal boosted classifier

3.3.5 Extended Boosted Classifier

After the training of the boosted classifiers and the classifier cascade we can classify an image with this "super-classifier". In this work an extension to the traditional boosted classifier is introduced which reduces the amount of necessary computation during the classification process noticeably. The "normal" boosted classifier described in Section 3.3.1 can be visualised as shown in Figure 3.5. Here c_i are the weak classifiers and α_i their corresponding weights. A boosted classifier C has N_C weak classifiers.

In a traditional boosted classifier normally all weak classifiers are evaluated on the sample image. If the weighted sum of all these outputs is above a certain threshold the image is classified as positive – otherwise as negative. The fundamental idea behind extended boosting is to test already during the evaluation of the boosted classifier (e.g. after 50% of the weak classifiers) if it makes sense to evaluate the remaining weak classifiers. In particular in boosted classifiers with a large number of weak classifiers this would result in a performance increase.

We can easily pre-classify an image as pedestrian if after evaluating k weak classifiers the sum of their votes $\sum_{i=1}^{k} \alpha_i c_i(I)$ is above the pedestrian threshold λ_C of this boosted classifier C:

$$\underbrace{\sum_{i=1}^{k} \alpha_i c_i(I)}_{current} > \underbrace{\frac{1}{2} \sum_{m=1}^{N_C} \alpha_m}_{\lambda_C} \Rightarrow C(I) = 1 \tag{3.19}$$

If this is the case we can skip the remaining $N_c - k$ weak classifiers and the boosted classifier can classify the image immediately as pedestrian.

This *positive pre-classification* is implementable without changing the structure of the boosted classifiers. Another idea is if a *negative pre-classification* is possible where we can reject an image as non-pedestrian before we have evaluated all weak classifiers of a boosted classifier. The test at a weak classifier c_k determines if it makes sense to evaluate the remaining and unevaluated weak classifiers by

$$\underbrace{\sum_{i=1}^{k} \alpha_i c_i(I)}_{current} + \underbrace{\sum_{j=k+1}^{N_C} \alpha_j}_{possible} < \underbrace{\frac{1}{2} \sum_{m=1}^{N_C} \alpha_m}_{\lambda_C} . \tag{3.20}$$

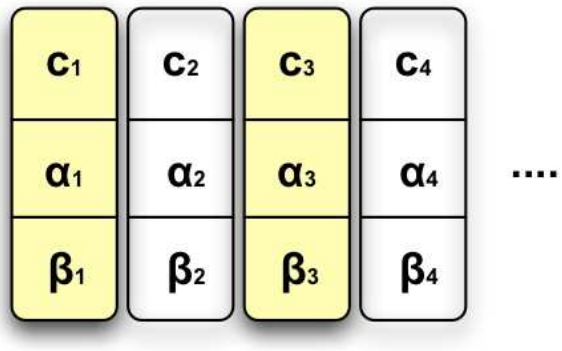

Figure 3.6: Extended boosted classifier

Here the term $\sum_{i=1}^{k} \alpha_i c_i(I)$ represents the summed classification results of the already evaluated classifiers. If all remaining classifiers vote for the class pedestrian ($class = 1$) then the maximal possible increase could be $\sum_{j=k+1}^{N_C} \alpha_j$. An image I could be a pedestrian if the current sum $\sum_{i=1}^{k} \alpha_i c_i(I)$ and the maximal possible rest $\sum_{j=k+1}^{N_C} \alpha_j$ can reach the classification threshold $\frac{1}{2}\sum_{m=1}^{N_C} \alpha_m$. If they cannot reach it we can immediately classify the image as non-pedestrian and abort the classification. If we bring the term $\sum_{j=k+1}^{N_C} \alpha_j$ to the right side of the equation, we now have a test which can be performed at each weak classifier c_i:

$$\underbrace{\sum_{i=1}^{k} \alpha_i c_i(I)}_{current} \quad < \quad \underbrace{\frac{1}{2}\sum_{m=1}^{N_C} \alpha_m - \sum_{j=k+1}^{N_C} \alpha_j}_{\beta_k} \quad \Rightarrow \quad C(I) = 0 \tag{3.21}$$

Since both terms on the right side are now constant we can calculate them already offline as

$$\beta_k = \frac{1}{2}\sum_{m=1}^{N_C} \alpha_m - \sum_{j=k+1}^{N_C} \alpha_j = const. \tag{3.22}$$

Therefore the traditional boosted classifier structure is extended to an *extended boosted classifier* (Figure 3.6) which contains, next to the weights $\alpha_{1..N_C}$, additional test thresholds $\beta_{1..N_C}$ for each of its features $f_{1..N_C}$. If the current sum is below the current test threshold β_k the classification can be aborted immediately and the image labelled as non-pedestrian, i.e.

$$\sum_{i=1}^{k} \alpha_i c_i(I) < \beta_k \Rightarrow C(I) = 0 \tag{3.23}$$

For positive and negative pre-classification it makes sense to sort the weak classifiers c_i according to their weights α_i so that

$$\alpha_1 \geq \alpha_2 \geq ... \geq \alpha_{N_C}. \tag{3.24}$$

The weak classifiers are evaluated according to their importance since classifiers with a higher weight have a stronger influence on the final result.

For complex objects like pedestrians normally many weak classifiers must be evaluated until the image can be pre-classified by the tests of the Equations 3.19 and 3.23. Therefore we test only after 50%, 66%, 75% and 85% of the weak classifiers if the image could be a pedestrian.[1]

3.4 Detecting pedestrians

The detector searches for pedestrians in the current video frame at various sizes and locations. This is done by selecting sub-images in the frame and classifying them. For selecting a sub-image we do not copy this part of the frame into a new memory location, but reference with a window to this sub-image. The detector then uses this reference for classifying the correct sub-region of the frame. The windows used for classification are uniformly distributed over the frame and the image pyramid described in Section 3.4.1 handles the scaling and the translation of the windows. In Section 3.4.2 we introduce a clipping algorithm which reduces the constant number of windows from the pyramid by regarding the foreground motion in a sub-image. The detector then classifies this reduced number of windows.

3.4.1 Image pyramid

The image pyramid creates a set of uniformly distributed windows (with referenced sub-image) to find pedestrians in an image I at multiple scales and locations. A more common approach for image pyramids is to resize the image I to the scale of each pyramid level and to apply the unresized classifiers in this scaled image at uniformly distributed positions. This may make sense in a case where it is computationally too complex or inaccurate to scale a filter or use it at different sizes. For our filters the scaling of the filter makes sense because they can be applied at any scale with the same costs.

The computation time for a feature ρ is independent of its size (due to the integral images) and it is therefore faster to resize the complete classifier cascade with all its boosted classifiers to each pyramid level and to use it in the unresized image I at various (uniformly distributed) positions by using the windows. The detector classifies the sub-image which is referenced by the window. The image pyramid used in this detector consists of 13 different pyramid levels. Each level contains the complete classifier cascade which is resized for this level scale and a set of uniformly distributed windows where the classifier cascade is applied in image I.

Since all our video frames have a fixed resolution which does not change during one video sequence, the pyramid has to be computed only once at the beginning and can be used for all frames without additional

[1] Note that both pre-classification methods save the evaluation of weak classifiers and increase the speed of the classification process. They neither change the classification result of a boosted classifier nor the detection or false positive rate.

computation. Note that the resizing of the classifier filters to multiple scales also takes place only once at the initialization of the detector and can be used for all frames. If we would use an image pyramid with multiple resolutions in each frame, computationally expensive scaling for each video frame would be necessary.

At a scale factor $S = 1.2$ between each pyramid level good results were obtained. Each frame was divided into $N_W = 2684$ windows $W_{1..N_W}$ with a window width A_l in pyramid level l of

$$A_l = A_o^{S^l} \tag{3.25}$$

Here A_0 is the initial window width at pyramid level $l = 0$. For the window height twice the window width was used $B_l = \frac{A_l}{R}$. The horizontal and vertical equal step size U was chosen relative to the window width A_l at level l with

$$U_l = \frac{3}{10} A_l \tag{3.26}$$

Therefore the step size changes with each pyramid level. The complete set of windows from all pyramid levels build a set of windows $W = \{W_1, .., W_{N_W}\}$ which is passed to the clipping algorithm in the next section.

Another interesting possibility for future work would be to generate randomly distributed and randomly scaled sub-windows without the uniform distributed pyramid described above. This method would be similar to particle filters [GGB+02, AMGC02] and could concentrate on areas with a high probability for locating a pedestrian e.g. in locations where pedestrians were detected in the frame and at the borders of the frame.

3.4.2 Background clipping

A possibility to decrease the computations is to limit the number of images which are given to the classification cascade. This section describes a fast clipping algorithm which uses the background model for filtering sub-images which cannot be used by the tracking system. The tracking algorithm in our system, which is described in more detail in Chapter 4, relies completely on the foreground information of a pedestrian which is provided by the background model. If no or only little foreground information is present in a sub-image then it does not matter if there is a pedestrian in this window or not since it is impossible to find any correspondence between this detection and other objects. A logical implication is that it does not make sense to give sub-windows without foreground information to the classifier cascade. The clipping algorithm introduced in this section is based on this implication.

After the background segmentation (Chapter 2) we compute the fraction of selected foreground pixels in a sub-image referenced by a window W :

$$frac_{W.fg} = \frac{N_{W.fg}}{size(W)}. \tag{3.27}$$

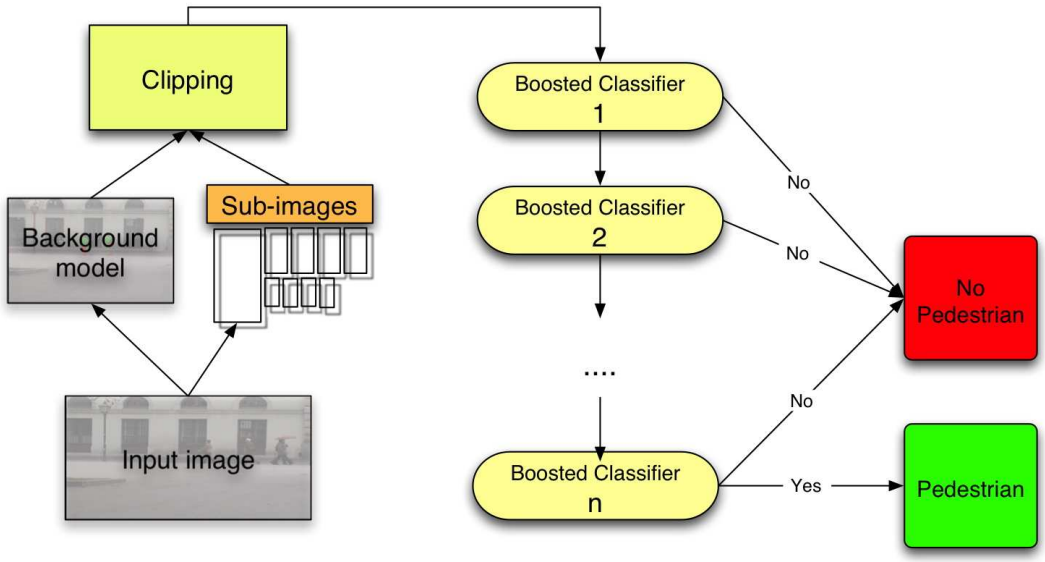

Figure 3.7: Adaptive Boosting with clipping

Here $N_{W.fg}$ is the number of foreground pixels in the area of the background mask referenced by window W and $size(W)$ the size of this area. If the fraction $frac_{fg}$ is above a threshold $\lambda_{clip} = 0.07$ then the sub-image of window W is given to the classification cascade, otherwise rejected already before the classification. The new structure of the detector with clipping is shown in Figure 3.7.

In Figure 3.8 we can see the results of this clipping algorithm for a set of example windows. The red windows $3 - 6$ are clipped because they do not contain enough foreground information. The fraction of foreground pixels in window 1 and 2 is higher than λ_{clip} and both windows are passed onto the pedestrian classifier.

Long-term static pedestrians are adapted into the background and the windows which contain these pedestrians are clipped. This is legitimate since for these windows – even if they contain a pedestrian – no useful tracking information is provided .

In video sequences with a lot of background noise where the background model does not provide a good separation between foreground and background, this clipping algorithm will not increase the performance significantly but maybe lead to a higher number of computations. Since our system addresses only static camera environments with moderate background noise a performance increase was achieved (Section 6.3.6).

After we have clipped all pyramid windows we classify them with the detector. All windows W_i which are classified as pedestrians by the classifier cascade are named *pedestrian detections* $\dot{d}_{1..N_{\dot{d}}}, \dot{d}_i \in W$ where $N_{\dot{d}}$ is the number of detections and $N_{\dot{d}} \leq N_W$.

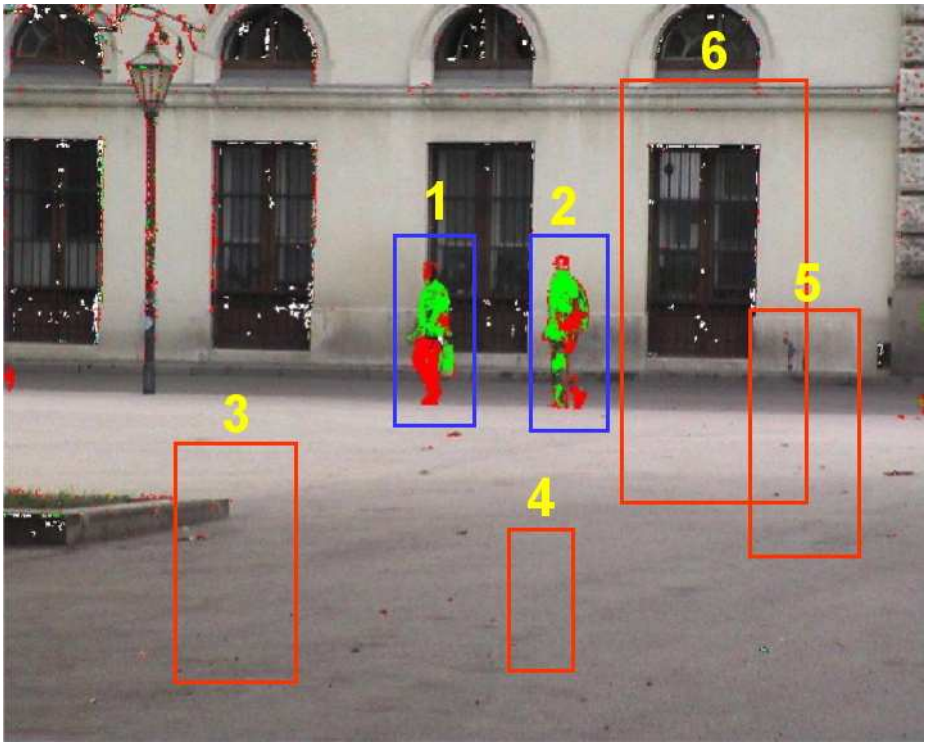

Figure 3.8: Background clipped windows

Chapter 4

Tracking features

Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth.

Sherlock Holmes
by Sir Arthur Conan Doyle
(1859-1930)

This chapter describes the structure which is used by the tracking system to reliably track a pedestrian and introduces a fast method for splitting a pedestrian-like detection up into three individual regions. After the splitting the spatial and colour features are extracted, tested for their reliability and weighted. Moreover, for comparing feature vectors with each other, appropriate distance measures are introduced. These features are experimentally evaluated in Section 6.4.

4.1 Pedestrian structure

The effectiveness of the tracking process depends strongly on the choice of the tracking features. Until the current stage in the system, each video frame was broken up into multiple sub-images by using the image pyramid and a detector (Chapter 3) classified these sub-images as pedestrians or non-pedestrians. After the classification, a set of pedestrian detections $\dot{d}_k, k \in \{0..N_{\dot{d}}\}$ in the current frame is provided. These detections build a subset of the pyramid windows $W_{1..N_W}$ and are also defined by size and position. Since this system tracks each body part of a pedestrian individually, a separation of the detection into the three parts: head, upper body and lower body has to be done. Normally a detection window containing a pedestrian is vertically as well as horizontally larger than the pedestrian itself. For a more accurate height estimation of the pedestrian, we use a reduction algorithm described below and reduce the window vertically until it contains mainly foreground pixels. After we have the vertically resized

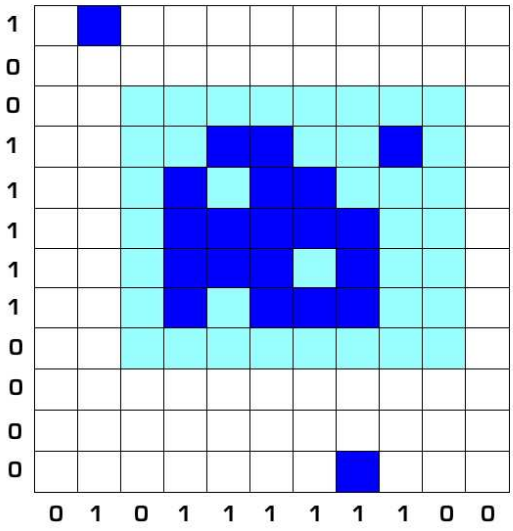

Figure 4.1: Body part reduction with shrinking ratio $R_{shrink} = \frac{4}{5}$

window, we divide each detection window into the three individual body zones by using a fixed height ratio $R_{height} = (R_{head}, R_{ub}, R_{lb})' = [\frac{1}{4}, \frac{3}{8}, \frac{3}{8}]'$. This leads to three body part windows with equal width and fixed height ratio. Since the widths of the parts are normally not equal we now reduce each body part individually in the horizontal direction. This leads to three individually sized body parts.

The reduction mentioned above can quickly be done by searching for a certain number of successive rows and columns which contain at least one foreground pixel. An example is shown in Figure 4.1. Here the blue pixels represent foreground and white pixels represent background. In this example a shrinking ratio of $R_{shrink} = 4/5$ is used, which means that a region is reduced in every direction until 4 of 5 rows or columns contain at least one foreground pixel. After some empirical testing a relative ratio of $R_{shrink} = 11/16$ was chosen for the implementation and resized according to the size of the body part. The reason for using a ratio unequal to one is to be tolerant against wrongly selected or missing foreground pixels and moderate background noise.

Since all parts of a pedestrian body are created by dividing and reducing the original detection window, it is ensured that all three body parts always stay inside the original detection window and therefore in an accurate distance with respect to each other. In Figure 4.2 the complete structured pedestrian after the reduction process is shown.

After the splitting of the detection into individual parts and the reduction to best-fitting rectangular regions, the system extracts the tracking features.

4.2 Feature extraction

Each part of the structured pedestrian is tracked individually and therefore appropriate features must be extracted. In this system a mixture between colour and spatial features for pedestrian tracking is used.

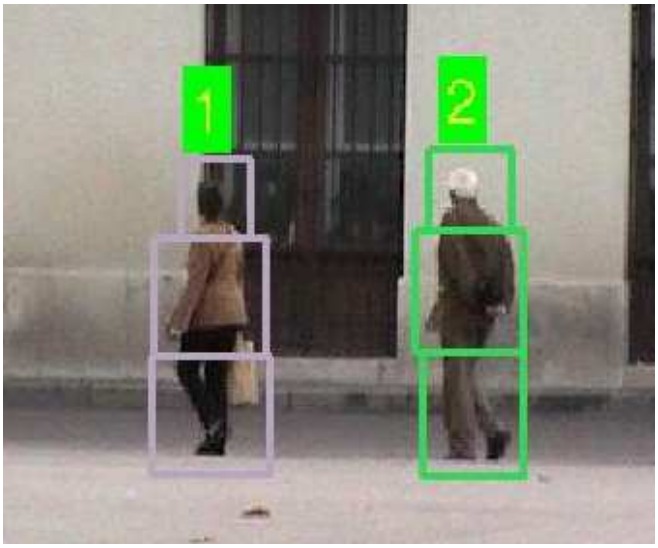

Figure 4.2: Body parts after reduction

In addition to the already available information about location and size of a body part, also the colour histograms and means and variances of the foreground pixels of each part are extracted.

Colour histograms describe the colour characteristics of a set of pixels and are used to compare images in many applications like image retrieval or object recognition. The reason for using two very distinctive feature types is that colour features, in contrast to spatial features, are very robust against partial occlusions, object deformation and scaling. Additionally they are very efficient to compute and — up to a certain degree — insensitive to changes in the camera viewpoint.

However there are also some limitations of colour histograms. They do not use any spatial information and therefore merely describe the quantity of each colour value occurring in the image. Therefore two images with different appearances can have similar histograms due to similar overall colour distributions. In addition colour histograms are very sensitive to changes of the overall image brightness and to compression artifacts.

The feature vector for each detection $\dot{d}_k$ has the form

$$f_k = \begin{pmatrix} P_k \\ \dot{P}_k \\ S_k \\ H_k \\ \mu_k \\ \sigma_k \end{pmatrix} \tag{4.1}$$

where all six elements are vectors. P_k contains the absolute and relative position information of the pedestrian detection and the body parts in the detection window. $\dot{P}_k$ and S_k respectively describe the velocity and size information. They are of the following form:

$$P_k = \begin{bmatrix} X_{k.abs} \\ X_{k.head} \\ X_{k.ubody} \\ X_{k.lbody} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} x_{k.abs} \\ y_{k.abs} \end{pmatrix} \\ \begin{pmatrix} x_{k.head} \\ y_{k.head} \end{pmatrix} \\ \begin{pmatrix} x_{k.ubody} \\ y_{k.ubody} \end{pmatrix} \\ \begin{pmatrix} x_{k.lbody} \\ y_{k.lbody} \end{pmatrix} \end{bmatrix}, \dot{P}_k = \begin{bmatrix} \dot{X}_{k.abs} \\ \dot{X}_{k.head} \\ \dot{X}_{k.ubody} \\ \dot{X}_{k.lbody} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} \dot{x}_{k.abs} \\ \dot{y}_{k.abs} \end{pmatrix} \\ \begin{pmatrix} \dot{x}_{k.head} \\ \dot{y}_{k.head} \end{pmatrix} \\ \begin{pmatrix} \dot{x}_{k.ubody} \\ y_{k.ubody} \end{pmatrix} \\ \begin{pmatrix} \dot{x}_{k.lbody} \\ \dot{y}_{k.lbody} \end{pmatrix} \end{bmatrix} \tag{4.2}$$

$$S_k = \begin{bmatrix} S_{k.abs} \\ S_{k.head} \\ S_{k.ubody} \\ S_{k.lbody} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} A_{k.abs} \\ B_{k.abs} \end{pmatrix} \\ \begin{pmatrix} w_{k.head} \\ h_{k.head} \end{pmatrix} \\ \begin{pmatrix} w_{k.ubody} \\ h_{k.ubody} \end{pmatrix} \\ \begin{pmatrix} w_{k.lbody} \\ h_{k.lbody} \end{pmatrix} \end{bmatrix} \tag{4.3}$$

Here $X_{k.*} \in \mathbb{R}^2$, $\dot{X}_{k.*} \in \mathbb{R}^2$ and $S_{k.*} \in \mathbb{R}^2$ are the absolute and relative positions, velocity and sizes of a pedestrian and its body parts. The positions are in respect to the centres of the pedestrian and the body parts. The velocity vector $\dot{P}_k$ is set to 0 at the initialization. The absolute size $S_{k.abs}$ and the absolute position $X_{k.abs}$ are determined by the detection and the absolute size and position of the each single body part is calculated by the reduction algorithm.

Vectors H_k, μ_k and σ_k contain the histograms, means and variances for all three colour channels for all three body parts.

$$H_k = \begin{bmatrix} H_{k.hue.head} \\ H_{k.hue.ubody} \\ H_{k.hue.lbody} \\ H_{k.sat.head} \\ H_{k.sat.ubody} \\ H_{k.sat.lbody} \\ H_{k.val.head} \\ H_{k.val.ubody} \\ H_{k.val.lbody} \end{bmatrix}, \mu_k = \begin{bmatrix} \mu_{k.hue.head} \\ \mu_{k.hue.ubody} \\ \mu_{k.hue.lbody} \\ \mu_{k.sat.head} \\ \mu_{k.sat.ubody} \\ \mu_{k.sat.lbody} \\ \mu_{k.val.head} \\ \mu_{k.val.ubody} \\ \mu_{k.val.lbody} \end{bmatrix}, \sigma_k = \begin{bmatrix} \sigma_{k.hue.head} \\ \sigma_{k.hue.ubody} \\ \sigma_{k.hue.lbody} \\ \sigma_{k.sat.head} \\ \sigma_{k.sat.ubody} \\ \sigma_{k.sat.lbody} \\ \sigma_{k.val.head} \\ \sigma_{k.val.ubody} \\ \sigma_{k.val.lbody} \end{bmatrix}. \tag{4.4}$$

The normalized range of $[0, 1]$ of each colour component (hue, saturation and value) is divided into N_{HB} equally sized intervals and each colour histogram contains N_{HB} bins which express the relative quantity of the colour measurements for each of these intervals.

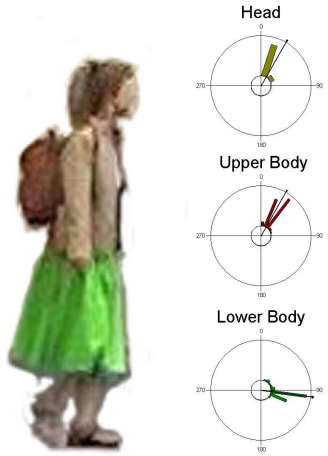

Figure 4.3: Hue histograms and hue means.

$$H_{k.*} = \begin{bmatrix} B_1 \\ B_2 \\ B_3 \\ \vdots \\ B_{N_{HB}} \end{bmatrix}, B_i \in [0,1]\, with \sum B_i = 1 \tag{4.5}$$

For generating the hue histogram all hue values are additionally weighted by their corresponding saturation.

In Figure 4.3 the statistical hue information $H_{hue.*}$ for the head, for the upper body and for the lower body is visualised in a circular histogram. For each body part the circular histogram and the mean of the hue information is shown. Here we can see that the histograms of the three parts represent the colours of each body part i.e the histogram of the lower body shows the strong amount of green pixels for this regions. The histogram maxima is aligned around 100° and close to the hue of the primary green with 120°.

4.3 Feature validation

Each component of the feature vector is validated after the extraction. If a component does not fit certain criteria it is regarded as not sufficiently reliable and is not included in similarity measurements between different pedestrian detections. In this implementation, criteria for feature components are given as follows

1. The absolute position $X_{k.abs}$ in the feature vector of a pedestrian (determined by the detection) is always used as reliable feature.

2. The size S_k of a body or one of its parts must be greater than a minimal size S_{min} (in pixels) to provide reliable position and size information for a pedestrian. If an element of S_k is too small we assume that the background segmentation provides too little information for the reduction of this body part. In this implementation a minimal width/height of 6 pixels was used for the body parts. This leads to a minimal pedestrian height of 18 pixels and a minimal width of 6 pixels.
3. At least $N_{minHist} = 25$ observation pixels must be used for creating a meaningful histogram.
4. At least $N_{minHist} = 25$ observation pixels must be used for calculating the means $\mu_{\{h,s,v\}}$ and variances $\sigma_{\{h,s,v\}}$.

According to these criteria, a reliability vector R_k for each feature vector f_k of the form

$$R_k = \begin{bmatrix} R_{k.position} \\ R_{k.size} \\ R_{k.hist} \\ R_{k.mean} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} r_1 \\ \vdots \\ r_4 \end{pmatrix} \\ \begin{pmatrix} r_5 \\ \vdots \\ r_8 \end{pmatrix} \\ \begin{pmatrix} r_9 \\ \vdots \\ r_{17} \end{pmatrix} \\ \begin{pmatrix} r_{18} \\ \vdots \\ r_{26} \end{pmatrix} \end{bmatrix}, r_i = \begin{cases} 1 & \text{if feature } i \text{ is usable} \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

is defined. An element $r_i \in R_k$ is 1 for all elements of the feature vector which can be used for comparison with other feature vectors – otherwise it is set to 0 . The absolute position is always regarded as reliable and r_1 is always 1. The velocity is left out in the reliability vector since it is only used for the state estimation (Section 5.3.2) but not as a tracking feature.

4.4 Distance measures

We now have a set of pedestrian detections – described by their feature and reliability vectors – and want to express the similarity between them by their distance to each other. A *distance* or *metric* is a function which assigns a distance to a set of elements. Pedestrian detections with equal or similar feature values should have a smaller distance than strongly different pedestrians.

4.4.1 Desirable properties of distance measures

An ideal distance $\delta(x, y)$, where x and y are two observations has the following properties [GM76]:

1. $\delta(x, y) = \delta(y, x)$
2. $\delta(x, y) > 0 \; if \; x \neq y$
3. $\delta(x, y) = 0 \; if \; x = y$
4. $\delta(x, y) \leq \delta(x, z) + \delta(z, y)$

Property 1 is necessary for symmetry and properties 2 and 3 are required for positive-definiteness. Property 4 is called the *triangle inequality*. Examples of distance measures which satisfy all four properties are e.g. the *Euclidean distance* or the *Mahalanobis distance*.

4.4.2 Distance functions

After we have validated which elements of the feature vector can be used for comparison with other feature vectors, we want to compute the distances between two reliable feature vectors. Since our feature vector contains spatial as well as colour information, different distance measures are used to calculate the distances between two of its elements.

For measuring a spatial difference δ_{spat} between two vectors $X_i, X_j \in \mathbb{R}^2$, a normalized *Euclidean distance* is used which satisfies all four properties of Section 4.4.1:

$$\delta_{spat}(X_i, X_j) = \frac{\|(X_i, X_j)\|_2}{\Delta_{spatmax}}, \tag{4.7}$$

where $\Delta_{spatmax}$ is the maximal spatial distance which is possible between two detections of the same object and which normalizes δ_{spat} to the interval $[0, 1]$. The maximal spatial distance $\Delta_{spatmax}$ is set relatively to the absolute widths in the feature vectors. We calculate the maximal spatial distance for the absolute positions of two detections d_i and d_j with feature vectors f_i and f_j as

$$\Delta_{spatmax} = \frac{A_{i.abs} + A_{j.abs}}{2} \tag{4.8}$$

and the distance between the absolute positions by

$$\delta_{spat}\left(X_{i.abs}, X_{j.abs}\right) = \frac{\|(X_{i.abs}, X_{j.abs})\|_2}{\Delta_{spatmax}} = \frac{2\,\|(X_{i.abs}, X_{j.abs})\|_2}{A_{i.abs} + A_{j.abs}}. \tag{4.9}$$

If $\|(X_{i.abs}, X_{j.abs})\|_2 > \Delta_{spatmax}$ the two detections with feature vectors f_i and f_j are regarded as spatially to far apart to be similar and the distance between them is $\delta_{spat}(f_i, f_j) = \infty$. The distance between two size vectors S_i and S_j is computed equally. For the body parts we use the relative size of each part to determine the maximal spatial distance.

The difference $\delta_{hist} \in [0, 1]$ between two colour histograms H_i and H_j is computed by using the *Bhattacharyya distance*

$$\delta_{hist}(H_i, H_j) = 1 - \sum_{k=1}^{B} \sqrt{H_i(k) H_j(k)} \tag{4.10}$$

where B is the number of histogram bins. In our work it was set to $B = 10$. Note that the Bhattacharyya distance only satisfies the properties 1 to 3 of Section 4.4.1 but not the triangle inequality as noted in [Kai67]. This fact is important for later processing of distance values and will therefore be investigated in more detail later in this section.

As distance δ_{dist}, $\delta_{dist} \in [0, 1]$ between a reference distribution $A(\mu_1, \sigma_1)$ and other distribution $B(\mu_2, \sigma_2)$ a modified *Mahalanobis distance*

$$\delta_{dist}(\mu_1, \sigma_1, \mu_2, \sigma_2) = min\left(\frac{|\mu_1 - \mu_2|}{2min(\sigma_1, \sigma_2)}, 1\right) \tag{4.11}$$

is used. Here the distance is normalized by the factor 2σ (which is the border of the 95% confidence interval) and bound to the range $[0, 1]$ by the minimum function. The distance between a distribution $A(\mu_1, \sigma_1)$ and a value x is computed as

$$\delta_{dist}(\mu_1, \sigma_1, x) = min\left(\frac{|\mu_1 - x|}{2\sigma_1}, 1\right) \tag{4.12}$$

Note that the normal Mahalanobis distance $D_{Mahalanobis} = \frac{|\mu - x|}{\sigma}$ is the same as the Euclidean distance for the case $\sigma = 1$:

$$\delta_{dist}(\mu, 1, x) = |\mu - x| = \sqrt{(\mu - x)^2} = \|(\mu, x)\|_2 . \tag{4.13}$$

After the distances for all elements of the feature vectors are computed independently with the chosen distance functions the unnormalized distance vector $\delta, \delta \in \mathbb{R}^{26}$ contains spatial (position / size), distribution and histogram distances between two feature vectors f_i and f_j.

$$\delta(f_i, f_j) = \begin{bmatrix} \delta_{spat}\left(X_{i.abs}, X_{j.abs}\right) \\ \vdots \\ \delta_{spat}\left(X_{i.lbody}, X_{j.lbody}\right) \\ \delta_{spat}\left(S_{i.abs}, S_{j.abs}\right) \\ \vdots \\ \delta_{spat}\left(S_{i.lbody}, S_{j.lbody}\right) \\ \delta_{hist}\left(H_{i.hue.head}, H_{j.hue.head}\right) \\ \vdots \\ \delta_{hist}\left(H_{i.val.lbody}, H_{j.val.lbody}\right) \\ \delta_{dist}\left(\mu_{i.hue.head}, \mu_{j.hue.head}, \sigma\right) \\ \vdots \\ \delta_{dist}\left(\mu_{i.val.lbody}, \mu_{j.val.lbody}\right) \end{bmatrix} . \tag{4.14}$$

Note that the velocity features are not used for the distance calculations between two feature vectors. They are required in the dynamic system described in Section 5.3.2 for estimating the current position of an object.

The unnormalized distance vector δ is then element-wise multiplied by a weight vector Ω and by the reliability vector R and summed to an overall value $\delta_{norm} \in [0, 1]$. The weight vector Ω is set according to size of the body part and the amount of available information and reflects the influence of each feature measurement. It has the form

$$\Omega = \begin{bmatrix} \Omega_{position} \\ \Omega_{size} \\ \Omega_{hist} \\ \Omega_{mean} \end{bmatrix} = \begin{bmatrix} \begin{pmatrix} \omega_1 \\ \vdots \\ \omega_4 \end{pmatrix} \\ \begin{pmatrix} \omega_5 \\ \vdots \\ \omega_8 \end{pmatrix} \\ \begin{pmatrix} \omega_9 \\ \vdots \\ \omega_{17} \end{pmatrix} \\ \begin{pmatrix} \omega_{18} \\ \vdots \\ \omega_{26} \end{pmatrix} \end{bmatrix}. \tag{4.15}$$

The weights were set manually and according to the size of each body part. The feature values of larger upper and lower body parts had stronger weights than the head features ($head : ub : lb = \frac{1}{5} : \frac{2}{5} : \frac{2}{5}$).

The final normalized distance δ_{norm} is a linear combination of weighted distances and is computed as

$$\delta_{norm}(f_i, f_j) = \frac{(\delta(f_i, f_j) \cdot \Omega)^T R}{\Omega^T R}, \delta_{norm} \in [0, 1] \cup \infty \tag{4.16}$$

where δ is the unnormalized distance vector, R the reliability vector and Ω the feature weight vector. The reliability vector R is a logical OR combination of the two reliability vectors R_i and R_j

$$R = R_i \wedge R_j. \tag{4.17}$$

Therefore in Equation 4.16 only those elements of the feature vectors are used for the distance computation which are reliable in both feature vectors.

Because δ includes a metric which does not satisfy the triangle inequality, also for δ and for our normalized distance function δ_{norm} this requirement does not hold. However, we can limit the influence of the distances for histogram features with the weight vector Ω. With a boundary condition

$$\sum \Omega_{hist} = \Omega_9 + \Omega_{10} + .. + \Omega_{17} = const \tag{4.18}$$

we limit the influence of the histogram features to $100 \cdot \sum \Omega_{hist}$ percent of the total distance. The spatial and the distribution distances provide the remaining $100 \cdot (1 - \sum \Omega_{hist})\%$.

Also with the distance limitation to a constant factor, our normalized distance δ_{norm} still does not satisfy the triangle inequality but a triangle condition based only on spatial and distribution features is possible:

$$\delta_{norm}(x,z) - \sum \Omega_{hist} \leq \delta_{norm}(x,y) + \delta_{norm}(y,z). \tag{4.19}$$

On the left side we subtract the value $\sum \Omega_{hist}$ which is the maximal increase of the distance function δ_{norm} due to the histogram part.

4.5 Object merging

The tracking features are for assigning detections to objects in the appearance model but are also used by the detector to find multiple detections of the same object in a frame. All sub-windows which were passed to the detector and classified as pedestrian windows $\dot{d}_k, k \in \{1..N_{\dot{d}}\}$, are tested for their similarity to each other. A similarity threshold λ_{merge} and the normalized distance function δ_{norm} decide if multiple detection windows represent the same object:

$$\delta_{norm}(\dot{d}_i, \dot{d}_j) < \lambda_{merge}, i,j \in \{1..N_{\dot{d}}\} \Rightarrow \dot{d}_i = \dot{d}_j. \tag{4.20}$$

Similar detections are grouped together to merged pedestrian detections $d_k, k \in \{d_1..N_d\}$ with $N_d \leq N_{\dot{d}}$. If a pedestrian is detected and confirmed by multiple detection windows we assume that the detector is more certain about this classification. We therefore remember this number M_k of merged detections for each d_k and use it in the appearance model described in the next chapter as an estimate for the certainty of a detection.

Chapter 5

Appearance model

It is only shallow people who do not judge by appearances.

Oscar Wilde (1854-1900)

This chapter describes an appearance model – the controlling part of the tracking system – which collects the information about all detections occurring in the video sequence and merges this information together. In this chapter the term *pedestrian detection* or *detection* refers to a window in a single frame which was classified as pedestrian. The term object always refers to an object structure which exists during a longer period of multiple frames and describes a real world object i.e. a pedestrian while it moves through the scene. Each object is handled by the appearance model and has the same structure as a detection and an additional life value α. This life value α describes the state of the object. Since the structure of objects and detections are completely equal except for this life value α we can also compare the feature vectors of detections and objects with each other.

The adaptive appearance model (AAM) used in this work has to address multiple tasks. A major problem the AAM should cope with is the stable handling of occlusions. In the case where an object is partly or completely occluded the AAM should be able to predict its current position and size. This is done by using the information about the velocity, direction, size and position of the occluded object, which was collected during the previous frames. Since a missing object detection – due to the detector – can be regarded as a complete occlusion, this special case can also be addressed by the appearance model. Additionally, non-pedestrian objects which are incorrectly classified as persons by the detector can be filtered by the AAM. They normally occur only briefly or stay stationary for long periods of time and therefore can be distinguished from "real" pedestrians.

The tasks of the AAM are:

- to convert a detection d to an object structure o_i when it appears in the scene for the first time. This new object o_i with feature vector f_i and life value α_i is included in the object pool O of the appearance model and described by a feature vector f_i.

- to keep track of all objects $o_i \in O = \{o_1, .., o_{N_O}\}$ while they move through the scene.
- to recognise when an object leaves the scene and to delete it from the object pool.
- to filter wrong detections of non-pedestrians.
- to estimate the feature vectors of all objects where currently no update information is available.
- to recognise occlusions between multiple objects.

Use of the described appearance model is experimentally evaluated in Section 6.5.

5.1 Object assignment

This appearance model uses an object pool O where all objects known to the model are included. An important property of the appearance model is its ability to decide if a pedestrian detection $d_k, k \in \{1..N_d\}$ is similar to an already existing object o_i in the current object pool $O = \{o_1, .., o_{N_O}\}$ where N_O is the number of objects in the pool.

Since detections and objects have equally structured feature vectors we can use the same distance function as for the object merging process (Section 4.5) for finding similarities between two feature vectors and for the assignment of a detection to an object. A detection d_k with feature vector f_k is assigned to an object o_m with feature vector f_m if

$$assign(f_m, f_k) = \begin{cases} 1 & \text{if } \delta_{norm}(f_m, f_k) < \lambda_{assign} \\ 0 & otherwise. \end{cases} , m \in \{1..N_O\}, i \in \{1..N_d\}. \tag{5.1}$$

where λ_{assign} is an assignment threshold and defined as

$$\lambda_{assign} = \frac{\lambda_{merge} - \sum \Omega_{hist}}{2}. \tag{5.2}$$

The assignment threshold λ_{assign} and the weak triangle inequality of Equation 4.19 guarantee that each (merged) detection window is only assigned to a single object in the object pool. After the merging process the minimal distance between two merged detections with feature vectors f_i and f_j is at least

$$\delta_{norm}(f_i, f_j) \geq \lambda_{merge} = 2\lambda_{assign} + \sum \Omega_{hist} \geq 2\lambda_{assign}, i, j \in \{1..N_d\}. \tag{5.3}$$

If a pedestrian detection d_i is assigned to an object o_m then no other detection can be closer than λ_{assign} to the same object window.

The whole process of merging and assigning is visualised in Figure 5.1. Here the windows of the image pyramid which are classified as pedestrians are merged together by similarity to avoid that multiple windows detect the same object. During the object assignment we try to assign each detection to an existing object in the object pool. Each detection can only be assigned to one object.

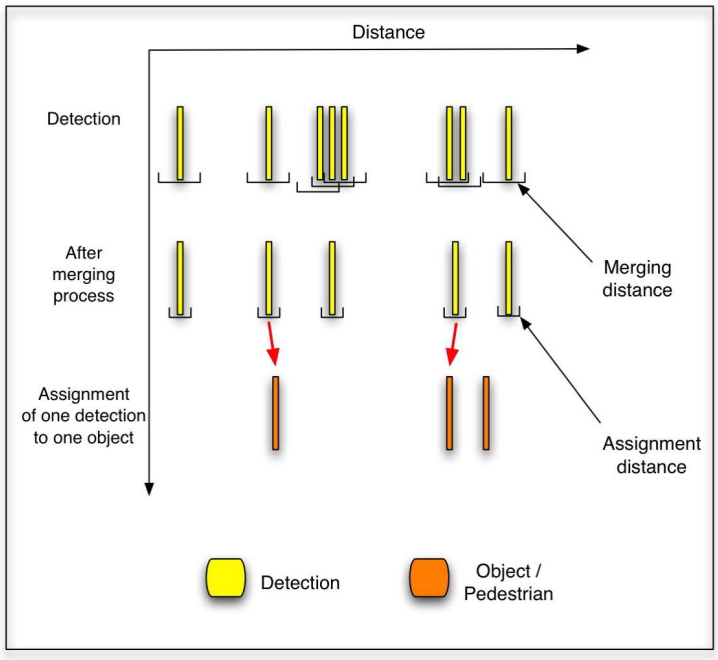

Figure 5.1: Object assignment

5.2 Object states

One fundamental job of the appearance model is the handling of all objects in its object pool. It provides each object with a life cycle and an age. For each detection which cannot be assigned to an existing object in the object pool a new object is created. This creation is the beginning of the object's life cycle. During its life an object can stay a normal object or become a pedestrian. After the object cannot be found in the scene anymore it is destroy and its life cycle ends. The appearance model has to support the object during its life cycle and therefore handles the life update which describes the state of an object during its life cycle.

This model uses a similar state model as used in [BCT04] for describing the state of an object by a life value α. According to the current life value α, an object in the object pool of the appearance model can be in one of two possible object states. For an unassigned detection d_k a new object is created and assigned to the state *Object*. In this state it is regarded as a possible candidate for the *Pedestrian* state but before it can reach this state it has to prove that it is no short-lived detection but stays permanently also in the subsequent frames. At this creation point the new object o gets an initial life value α_0 and is regarded as a possible pedestrian candidate. The initial life value α_0 is set as

$$\alpha_0 = \alpha_{init} + min(\alpha_{max}, \alpha_{bonus} M_k). \tag{5.4}$$

Here α_{max} is the maximal life bonus an object can receive per frame and α_{init} the initial life bonus. An object receives an additional life bonus α_{bonus} which depends on the detection which was used for the

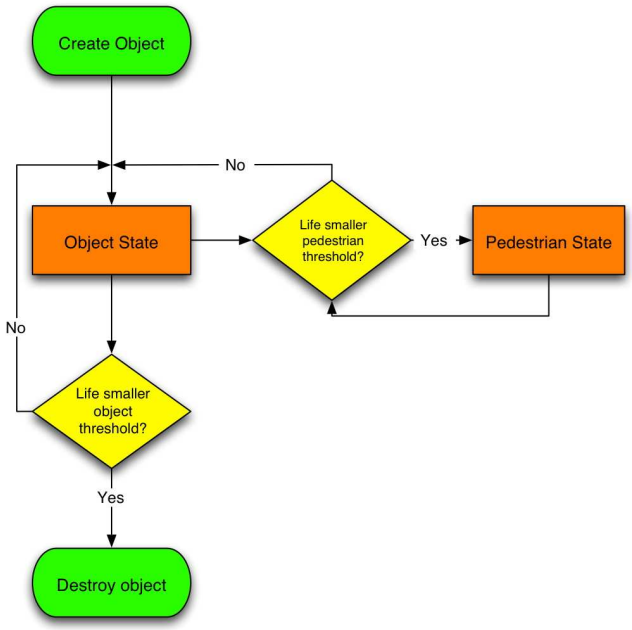

Figure 5.2: Object states

object creation. Since each detection represents one or more merged detections (see Section 4.5) we use the number M_k of merged detections for calculating the initial life bonus. An object which is confirmed by many detections starts with a higher life value than an object with a single detection and is regarded as having a higher probability to be a pedestrian.

If another detection d_m is assigned to an already existing object o, the current life value α_{t-1} of the object is increased by a life bonus α_{bonus}:

$$\alpha_t = \alpha_{t-1} + min(\alpha_{max}, \alpha_{bonus} M_m) \tag{5.5}$$

where α_t is the new object life value. The term M_m is the number of merged detections of the single detection d_m. For each frame an object without an assigned detection receives a penalty regarding the life value

$$\alpha_j = \alpha_j - \alpha_{penalty}. \tag{5.6}$$

This is always the case if an object leaves the scene, if it is not detected or if it gets occluded by another object.

If the life value α_j of an object rises above a threshold λ_{acc} the object reaches the *Pedestrian* state and the object can be post-labelled as pedestrian in all the previous frames. If the life span decreases below the threshold λ_{acc} it is set into Object state again. If it decreases below a threshold $\lambda_{destroy}$ it is removed from the object pool of the appearance model. In Figure 5.2 the possible states of an object during its life cycle are demonstrated.

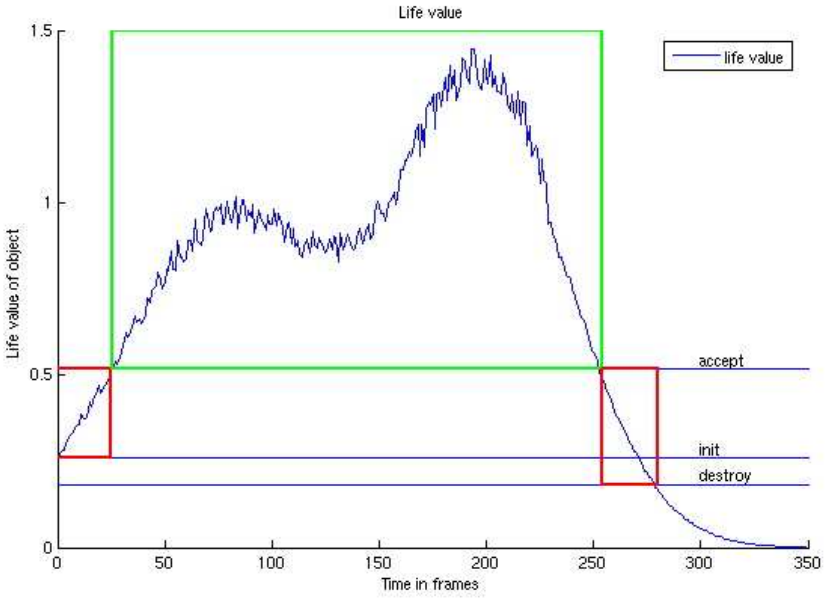

Figure 5.3: Life cycle of an object

In Figure 5.3 the different phases in the lifetime of an object are shown. The red rectangle on the left side represents the phase of the object creation until it is accepted as a pedestrian. The green rectangle shows the phase where the object is regarded as a pedestrian. Note that the objects in all frames of the first rectangle are post-labelled as pedestrians. The red rectangle on the right side represents the time when the tracker loses the object because it has left the field of view or is occluded for too long a time. Finally the object is removed from the object pool.

This two-state model works like a filter for wrong detections since these wrong detection windows are normally short-lived and stay for only 1 or 2 frames. At their first appearance they are included into the *Object* pool but never stay long enough to rise into the *Pedestrian* state.

The model thresholds λ_{acc} and $\lambda_{destroy}$ depend on the model parameters α_{max}, α_{bonus} and $\alpha_{penalty}$. By setting these parameters we can decide how fast an object is recognised as a pedestrian or removed from the object pool. They are all set relatively to the initial life bonus α_{init} which therefore has no influence on the model behaviour.

5.3 Object updates

Each object in the object pool tries to describe a real object which moves through the scene. It gets its feature vector from the initial detection but since the appearance of a real world object changes over time also the feature vectors of the object in the object pool have to be updated. This update is described in this section.

5.3.1 Feature update

Since each moving object changes its visual properties during its life time, the object feature vector also changes. If we want to compare a detection in the current frame to an object in the appearance model we have to update the features of the object to adapt it to the current situation. The appearance model therefore overwrites the old values in the object feature vector f_i with the new and reliable information (Section 4.3) from the assigned detection feature vector f_j.

Often the feature vectors of the new detection and the assigned object do not have the same set of reliable features. This is due to different temporal or spatial conditions during the lifetime of an object where the reliable features of an object normally change. For example if the background model does not provide a good separation between the current background and the lower body of a pedestrian, the information or features of the other body parts maybe become more important. It is also clear that the number of reliable features normally changes continuously. During subsequent frames more information about the object becomes available and the reliable feature number in the object feature vector rises.

If the detection cannot provide values for some of the reliable features of the object, we must estimate them as described in Section 5.3.2.

5.3.2 Feature estimates

The feature vectors of an object in the object pool and its assigned pedestrian detection in the current frame often do not provide the same *reliable features* (Section 4). Since we cannot use an unreliable feature of the detection to update the corresponding object feature we have two choices for handling this problem. The first one is to leave the feature value unchanged and to hope that they do not change until the next availability of this feature. We do this for all features which rely on the colour information like histograms and distributions.

In contrast to colour information the position and size of a moving object normally change and these values must be estimated. In this system a linear model estimates the current position of an object by linear interpolation. It uses the last object position and the last available velocity to determine the position in the current frame.

$$X_t = X_{t-1} + \dot{X}_{last}. \tag{5.7}$$

The velocity $\dot{X}_{last}$ is interpolated between two available position observations X_{t-k} and X_t. Here the velocity is calculated as

$$\dot{X}_{last} = \frac{X_t - X_{t-k}}{k} \tag{5.8}$$

This interpolation is done for each body part individually. A better estimate would probably be provided by a Kalman filter and is planned for future work.

If an assigned detection provides reliable features which are unreliable in the object feature vector we can use these values to gain additional information about an object and to supplement its feature vector.

5.3.3 Collision group

Areas where multiple objects are too close together to be separated clearly enough from each other have to be specially addressed during the feature update. In these areas no clear extraction of the detection feature vector is possible and object assignment as well as feature update must be done very carefully.

A collision group in our system is a set of objects where each object in this group collides with at least another one of this group. Two objects o_i and o_j are colliding if

$$\|(P_i, P_j)\|_2 < (A_i + A_j)\,\lambda_{collision}. \tag{5.9}$$

Here P_i and P_j are the centers of the rectangular formed object windows and A_i, A_j the widths of both object windows. $\lambda_{collision}$ is the collision distance threshold which was set to $\lambda_{collision} = 0.55$.

Before the assignment process each object of the appearance model is tested for collisions and collision areas are created. If an object is in a collision area, it is regarded as temporarily read-only. No update of the colour or histogram features is done but the position and size are estimated as in Section 5.3.2. Figure 5.4 visualizes the assignment of an object to different collision states.

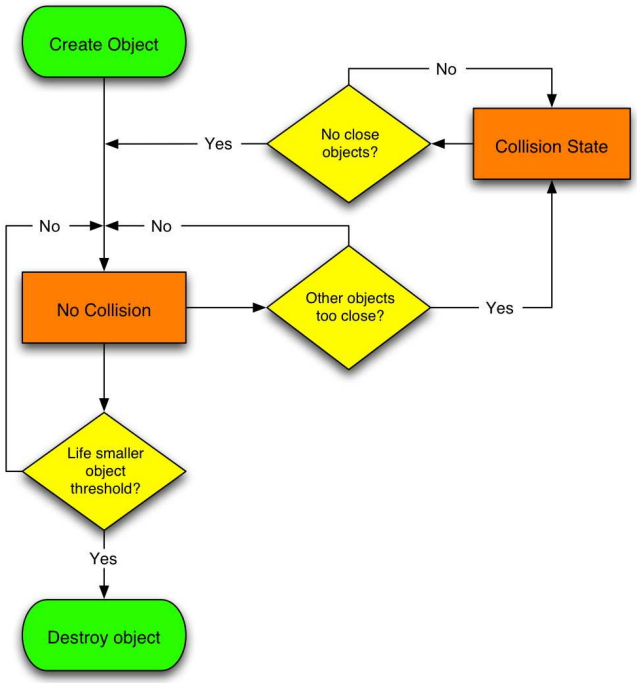

Figure 5.4: Collision area

Chapter 6

Results

In theory, there is no difference between theory and practice. But, in practice, there is.

Jan L.A. van de Snepscheut

This chapter investigates the efficiency of the single parts of the tracking systems and how they perform on the test sequences. The requirements on speed and robustness will be tested and shortcomings of our system described. Individual parts of the system are evaluated separately: the detector in Section 6.3, the features in Section 6.4 and the appearance model in Section 6.5. An example of the full system working is given in Section 6.6.

6.1 Specification of the training and test system

For the training of the detector a dual Pentium 3 with two 1.1 GHz processors and 2 Gbyte RAM running Linux was used. Because of the large number of users running programs on this computer simultaneously, the operating system simulates four virtual processors to better distribute the performance between the users. Due to the implementation in Matlab 7 which uses only a single processor, no advantage of the multi-processor system is taken and only a single processor – or 25% of the available computation power – is effectively used. The main advantage of running the training on this computer was its large amount of memory. During the training a large number of images is constantly needed. The hard disk is the bottleneck of most computers as it has only a limited data transfer rate. It is therefore often faster to keep the data in the working memory to have them always available for fast access.

Additionally Matlab is very memory consuming when running on a Unix machine. This is because Unix (and Linux), in contrast to Windows, frees the memory of a process after the process terminates and not after the memory is not used anymore and freed by the process. If a data structure does not fit into the

allocated and unused memory space of an application, new memory is allocated. Similar to file systems on hard disks, in the dynamic memory a fragmentation also takes place, which leads to continuous allocation of new memory if the code is not written carefully. Working with very large data structures and a long process running time, it was very important to take this into account to avoid a fast consumption of the available memory.

The implementation of the tracking system was done partly under Microsoft Windows and MacOS X and shows the platform independence of Matlab. No alteration of the source was necessary between the different operating systems. As system for testing the performance of the tracking system, a power book G4 (Power-PC 7450) with 1.5 GHz and 512 Mbyte RAM was used.

6.2 Test sequences

For the training of our detector we used a training set of 1500 manually extracted pedestrians from several video sequences. For the tests described in this section two video sequences – different to the training sequences – were used. The tests show how the tracking system handles sequences with multiple persons and partial or full occlusions. All pedestrians occurring in test sequence 1 and 2 were manually labelled and the ground truth information used for validating the results of the tracking system. Table 6.1 shows an overview of the properties of each test sequence. A detailed description of each sequence is given below.

	Test sequence 1	Test sequence 2
# of persons	4	3
# of frames	225	300
Partial occlusions	yes	yes
Full occlusions	no	yes
Ground truth	yes	yes

Table 6.1: Properties of test sequences

6.2.1 Test sequence 1

The first sequence (9 seconds, 225 frames) shows four persons where two of them are moving from the right side of the camera view to the left and the other two persons leave the image after two seconds on the right side. One person is partially occluded by a street lamp for part of the trajectory. This basic test sequence should primarily test if the feature vectors extracted are usable to distinguish between the pedestrians. Since both pedestrians have nearly the same size, primarily the colour and position information is usable. The partial occlusion of the pedestrian shows the effect of occlusions and how they are handled by the tracking system. Figure 6.1 shows a frame of test sequence 1.

Figure 6.1: Frame #1 of test sequence 1

6.2.2 Test sequence 2

In this sequence with a length of 12 seconds or 300 frames, two persons are moving next to each other from the right to the left part of the scene. Another person enters from the left and is first partially occluded by the street lamp and then occludes the two other pedestrians completely when he continues to go to the right. This sequence tests the system behavior on partial and full occlusions. Additionally it shows how it handles a group of multiple persons which are moving simultaneously and next to each other. One frame of test sequence 2 is shown in Figure 6.2.

6.3 Detector

This section describes the implementation of the detector in more detail and investigates the performance of the final detector on the two test sequences. Furthermore the results obtained are compared to existing implementations with respect to the detection rate and the false positive rate.

6.3.1 Training time

To see how strongly the Asymmetry and the Float Search algorithm influences the training time and the classifier size, we trained our system three times – once with the traditional AdaBoost algorithm (Algorithm 1), once with the asymmetric AdaBoost algorithm and once with the asymmetric FloatBoost algorithm (see Algorithm 2). The strong differences in the training time can be seen in Figure 6.3.

Figure 6.2: Frame #80 of test sequence 2

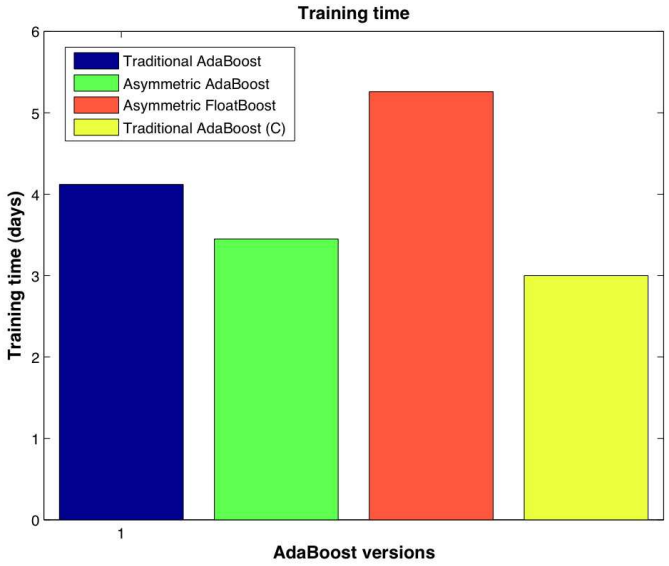

Figure 6.3: Training time

Training the classifier cascade with the traditional AdaBoost algorithm and the training parameters from Section 3.4 takes about 4 days. By using the asymmetric AdaBoost version a shorter training time of 3.5 days is achieved. The longest training time was needed by the asymmetric version with the Float Search algorithm. The training time is much higher than the other implementations, but the resulting boosted classifiers have less weak classifiers as described in Section 6.3.2. These smaller boosted classifiers are faster to evaluate which results in a faster classification process.

To provide a comparison between our Matlab implementation and a low-level implementation, the training time of a detector for face detection at the University of Queensland, Australia which uses a traditional AdaBoost implementation in C from the OpenCV library[1] is available. The training of this low-level implementation with a similar number of training images to our system takes takes 3 days for the training. The main factors for the relatively small difference in training time is that Matlab internally compiles the code instead of interpreting it and the use of the asymmetric AdaBoost algorithm leads to a faster convergence. However, an implementation of this algorithm in a low-level language would probably lead to even faster results.

An interesting point is that at the beginning of the training process the feature selection is computationally more expensive compared to the creation of the positive and negative training sets. In later stages the creation of the negative examples for the classifier training with bootstrapping becomes more time consuming since the false positive rate decreases in each stage and it gets harder to produce the necessary set of negatives.

Another interesting optimisation which could decrease the training time was proposed by Friedman et al. [FHT01] but is not used yet. They propose that training samples with a low weight, because of their small influence in the error loss function, can be neglected dynamically. Therefore during each round of the boosting algorithm less but more influential samples are used for the training, which results in less computations.

6.3.2 Stages

A final classifier cascade consists of 11 boosted classifiers. The classifiers C_i are sorted by their numbers of weak classifiers f so each one has equal or more weak classifiers than its previous stage. Since the computational complexity of each classifier rises with its number of features, the images are first classified by classifiers with a small size (Figure 6.4). In this figure the efficiency of the FloatBoost algorithm is demonstrated. Most of the boosted classifiers in the FloatBoost-trained cascade are smaller than their corresponding classifier trained with the traditional AdaBoost version and the total sum of weak classifiers for the FloatBoost cascade is nearly 10% smaller.

Due to the arranging of the boosted classifiers according to their size, only a reduced number of samples reaches higher classifiers in the cascade, which are more expensive to compute. If we test the detector with

[1] URL: *http://sourceforge.net/projects/opencvlibrary/*

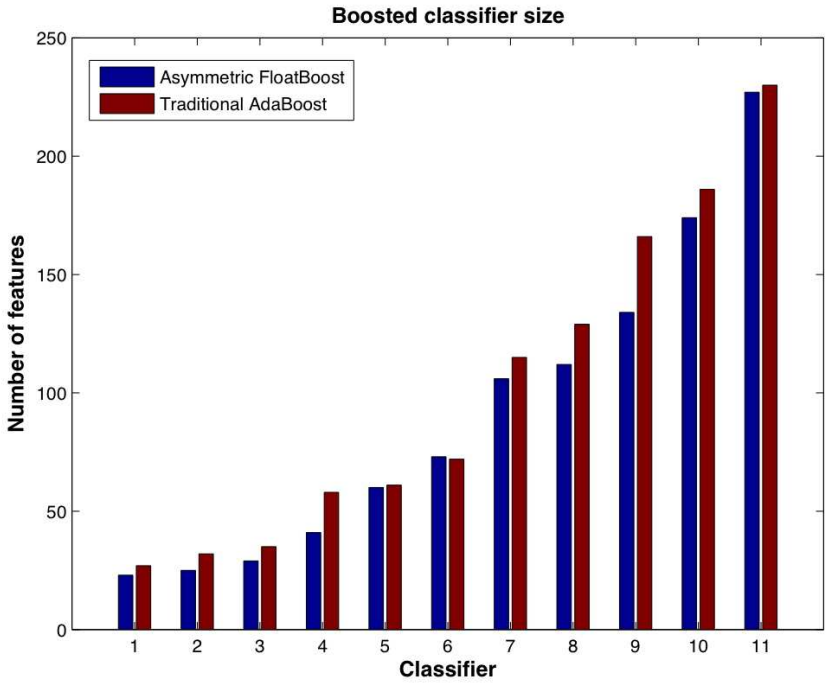

Figure 6.4: FloatBoost against AdaBoost

a set of only negative samples, then the percentage of negatives dropped per boosted classifier gives us information about the similarity between the boosted classifiers and the redundancy of a boosted classifier compared to its previous stages. This gives us the possibility to evaluate the efficiency of the AdaBoost algorithm. If a boosted classifier eliminates none or only a very small percentage of the negative samples which reach its stage, it means that it represents nearly redundant information. All previous classifiers together represent the same information as this classifier. In the AdaBoost algorithm this should never be the case because each classifier C_{i+1} should learn to eliminate new negatives which have not been classified correctly by the previous boosted classifiers $C_{1..i}$.

So if a single boosted classifier C_i reaches the target detection rate D_i (in our implementation $D \geq 0.99$) and the target false positive rate $F \leq 0.5$ then it represents new information regarding the classifier cascade.

In Figure 6.5 we can see the average fraction of examples eliminated in each classifier stage which was determined empirically. Here we can see, that each classifier eliminates at least 10% of the negatives reaching its stage. So each classifier represents different information. We can see that the classifier at the end of the cascade eliminates only 12% of the negatives in its stage which means that the false positive rate is quite high and that it only presents a small amount of additional information for the whole cascade. This should not be the case since it is the classifier with the highest feature number and therefore should provide the best description of a pedestrian. It is obvious that for each boosted classifier it becomes more difficult to explain how a typical pedestrian looks without using the same representation as the boosted classifiers before it. Since the number of significant features for describing a pedestrian is also limited often no further improvement of the detector is possible. In our case the training was aborted after the

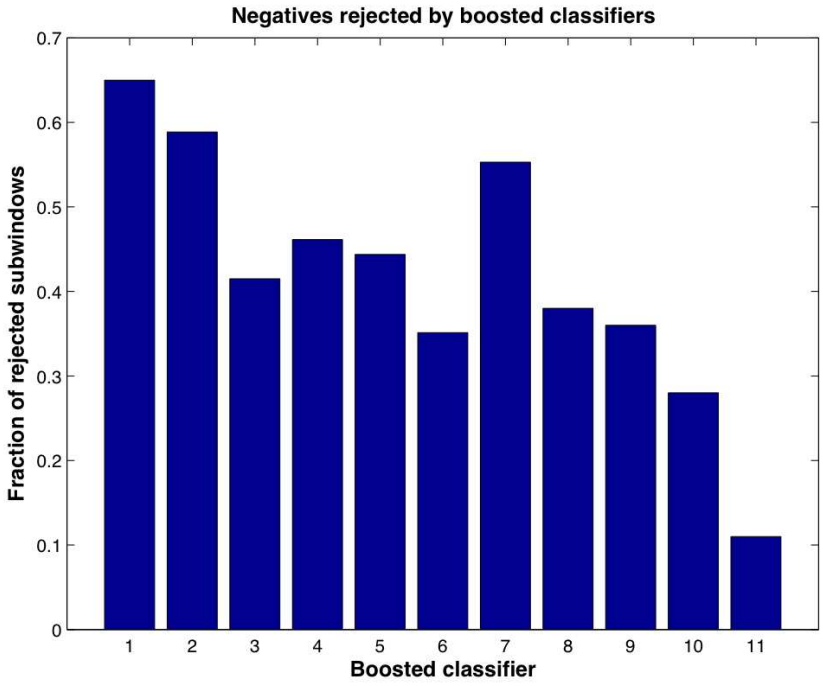

Figure 6.5: Rejected negatives

11th boosted classifier because no further improvement could be achieved.

6.3.3 Features

An interesting question is which features are used for the pedestrian classification. In Figure 6.6 we see which features in the boosted classifiers describe pedestrians. It is surprising that all seven features are nearly equally often used.

Figure 6.7 shows some examples of features which are used by the detector for a pedestrian description. It is interesting that most classifiers in our system compare a part or a characteristic feature of the pedestrian with the background. Features for finding relations between two pedestrian features are not used very often. In many face detectors based on the same kind of Haar-like features, often relations between two characteristic features of the object are used (e.g. the spatial relation between the two eyes or between the face and the nose). The reason why our detector sets mostly the background in relation with pedestrian features could be that our system – in contrast to most face detectors like [JV03] – uses images of different angles of pedestrians during the training. In face detection normally only the frontal view of faces is used for the training. Therefore, most face detectors only detect a face if it looks nearly frontally into the camera and does not find it if it is rotated by more than 15 degrees. In our system the relation between the pedestrian and background is most likely more constant than the relation between the parts of the pedestrian.

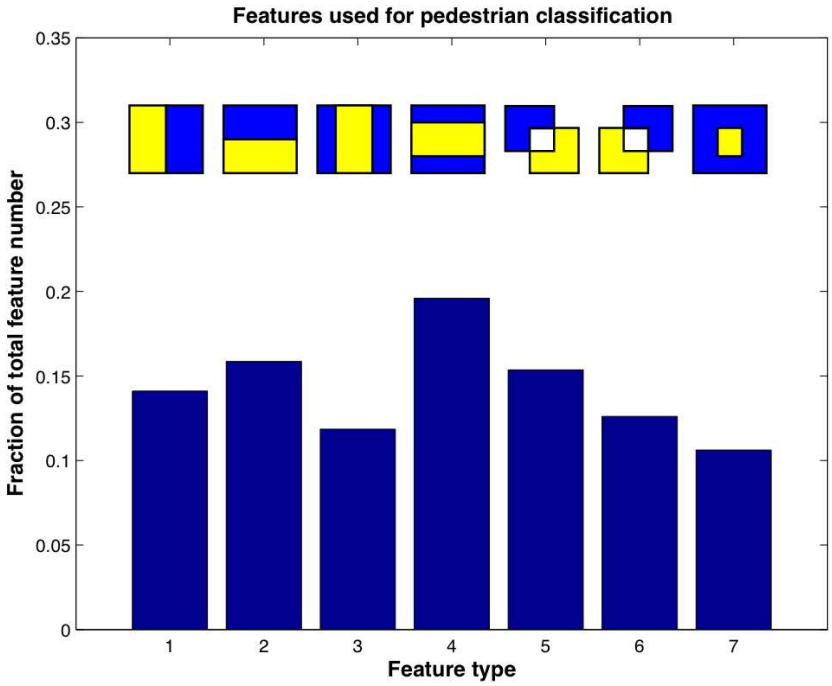

Figure 6.6: Features used in the cascade

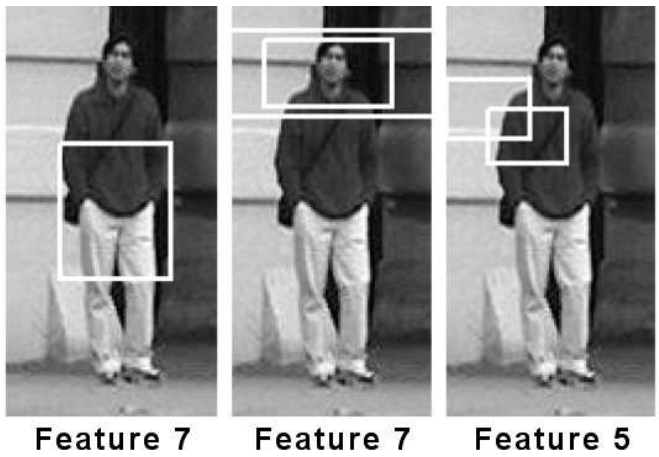

Figure 6.7: Feature examples

6.3.4 Classifier performance

At the beginning of the performance evaluation a static version of a detector introduced by Viola et al. [VJS03] as reference detector to our detector was chosen. The reason for using this detector as a test reference was because of its great similarity to our detector, e.g. it also uses Haar-like features and a cascade structure for the classification. Our system differs only minimally in the form of the static features (feature type 6 and 7) and the used training algorithm (Algorithm 2). As can be seen in [VJS03] the performance of the reference detector is very dependent on the test sequence. On one test scene the referenced detector reached a detection rate $D \geq 0.95$ for false positive rates greater than $F \geq 5 \times 10^{-5}$. For the second test sequence the detector only reached a detection rate of $D \geq 0.7$ at the same false positive rates.

Also if the detectors are tested on the same test sequence we normally have to use the same test method for the performance evaluation and the same test parameters. Viola et al. [VJS03] create the ROC curve for a test sequence by adjusting the threshold of each boosted classifier one at a time and taking the resulting rates of this truncated classifier cascade. Since we neither have the same number of classifier stages nor use independent and adjustable thresholds for each boosted classifier a direct comparison is not possible.

Therefore we did the performance evaluation of our detector with the test sequence 1 and 2 which are described above and only compared the order of magnitude of the resulting rates with the reference detector for plausibility. To build the ROC curve we slowly lowered or raised the constant threshold λ_C for the boosted classifiers. The decreasing or increasing of the threshold λ_C leads to a rising or descent of the detection and the false positive rate of the detector which is used for creating the ROC curve. This global lowering of the threshold avoids the adjusting of each boosted classifier and always regards the performance of the complete classifier cascade. During the generation of the ROC curve the clipping algorithm was deactivated to avoid a performance deterioration of the detector. The influence of clipping on the detector performance is investigated in Section 6.3.6.

We can see in Figure 6.8 that for both test sequences the detection rate is around 80% at a false positive rate of 10^{-5}. At a higher false positive rates of $F \geq 2 \times 10^{-5}$ the detector reaches a detection rate of 88% in the first and 84% in the second test sequence. In the second sequence this was the highest detection rate which could be achieved. In test sequence 1 a detection rate $D \approx 0.91$ at a false positive rate of $F \geq 7 \times 10^{-5}$ was achievable. Compared to the reference detector the rates of our detector are in a similar order of magnitude.

The average false detection rate for a boosted classifier threshold $\lambda_C = \frac{1}{2}$ on both test sequences was around $\tilde{F} = 3 \times 10^{-5}$ which is equivalent to one wrongly classified window in $\frac{1}{0.3 \times 10^{-4}} = 3333$ windows and an average detection rate of $\tilde{D}_{seq1} = 0.88$ for test sequence 1 and $\tilde{D}_{seq2} = 0.84$ for test sequence 2. These rates should provide a good base for our tracking system.

6.3.5 Pre-classification

The positive pre-classification with the extended boosted classifier (Section 3.3.5) was tested with a random set of negatives which was created with bootstrapping. We compared the percentage of windows

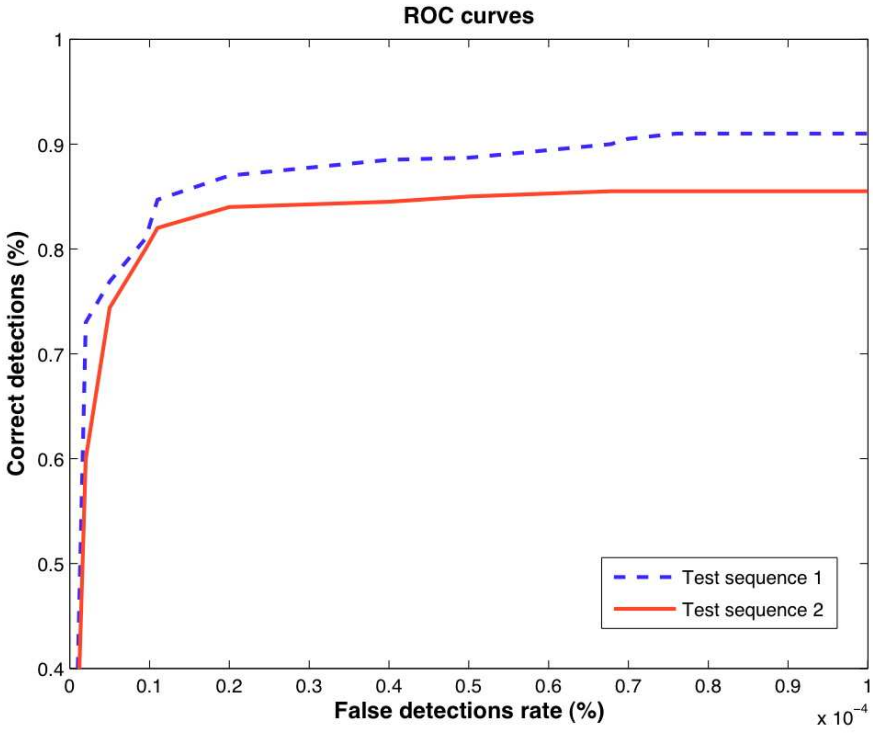

Figure 6.8: ROC curves of our detector.

rejected at one of the defined pre-classification points at 50%, 66%, 75% and 85% of the total feature number. Additionally the window percentage which required more than 85% of the weak classifiers was counted.

The left side in Figure 6.9 shows the relative fraction of rejected windows at each of the positive pre-classification points and on the right side – for better visualisation – the cumulative sums of these rates. We can see that only around 4% of the windows can be rejected after evaluating 50% of the weak classifiers. After 66% already 30% of the windows can pre-classified as a pedestrian and we can classify more than the half of the windows with only 75% of the weak classifiers. The other half of images is harder to classify and requires at least 85%.

For the negative pre-classification we can see in Figure 6.10 that only a relatively small amount of 5% of the windows can already be rejected after the evaluation of only one half of the weak classifiers. After 66% already around 15% of the negative samples can be rejected. After evaluating 75% of the weak classifiers already 40% of the negative samples can be classified as negative. A large ratio of more than two thirds of all negatives is rejected after 85% of the weak classifiers and only a rest of 26% requires the complete evaluation of the boosted classifier.

The theoretic velocity gain of the two pre-classification step seems quite high but in practice the effective velocity gain is much smaller. The negative pre-rejection only saves computations in the boosted classifier which rejects the negative sample. Since we do not know in advance which of the boosted classifiers in our cascade will reject the negative, we have to apply pre-classification in every boosted classifier which

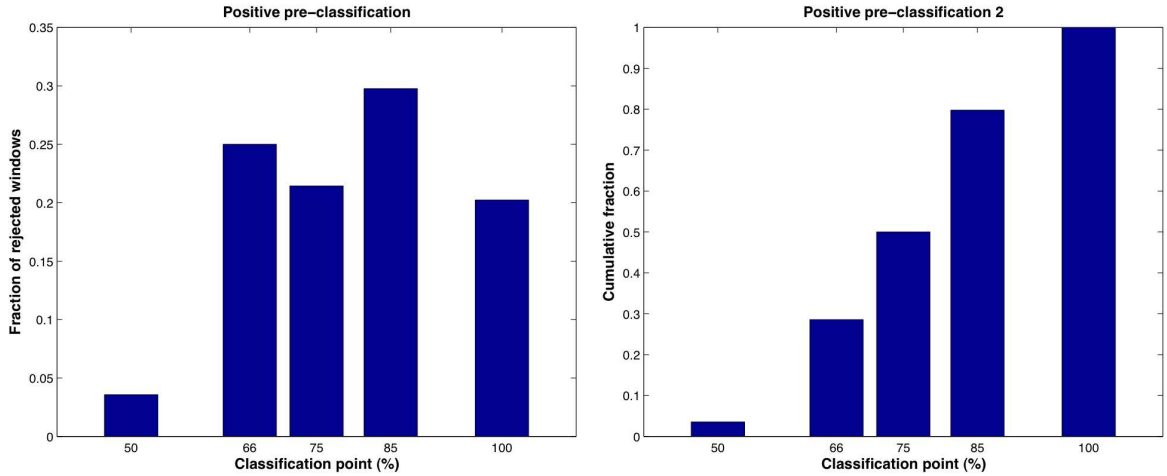

Figure 6.9: Positive pre-classification

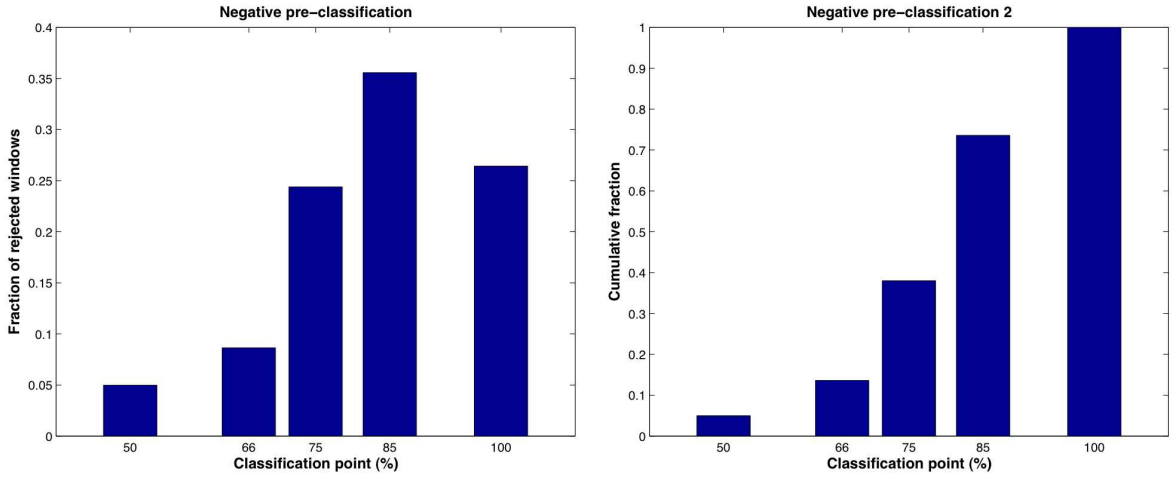

Figure 6.10: Negative pre-classification

leads to 4 additional comparisons per boosted classifier. The highest velocity gain for negative pre-classification would be possible for complex boosted classifiers which contain many weak classifiers and are located at the end of the cascade. Due to the cascade structure only few negatives normally reach these complex classifiers and therefore the efficiency of negative pre-classification is limited. The positive pre-classification saves in contrast to the negative pre-classification the evaluation of weak classifiers in each boosted classifier and is more efficient. Since the number of positive windows (i.e. pedestrians) normally is much smaller than the number of negative windows only for these few positives the full velocity gain for each boosted classifier is effectively used. However, also for negatives which are hard to classify and which reach higher stages a velocity gain is achieved since they are classified as positives in all previous boosted classifiers before the rejecting one.

As already mentioned in Section 3.3.5 negative as well as positive pre-classification does not have any influence on the result of the boosted classifier classification but only on the required evaluation time.

6.3.6 Clipping

The evaluation of the clipping algorithm (Section 3.4.2) is tested on the two test sequences. In Figure 6.11 and Figure 6.12 we can see the efficiency of this algorithm. Due to a clear background segmentation it is possible to reduce the numbers of detection windows constantly to around 10% in both video sequences. This relatively small percentage is then classified by the classifier cascade.

In Figure 6.11 one can see, that at the beginning when the background model is adapting itself to the scene the number of clipped windows is lower than at the end of the scene. Additionally, at the beginning four people are in the scene where two of them leave the scene after a few seconds. After an initialization phase of the background model and after the two people have left the scene, the number of remaining windows stays quite constant in the range between 13 and 18%. This constant behavior was predictable since the remaining two pedestrians do not change their distance to the camera and the background activity stays nearly constant.

For the second test sequence in Figure 6.12 one can also see the initial adaption of the background model where the number of clipped windows rises. Around frame 150 one pedestrian in the scene occludes the other one and a decreased foreground area leads to a lower number of clipped windows. The number of clipped windows rises again at the end when one of the pedestrians leaves the scene.

The activated clipping does not deteriorate the performance of the detector in both test sequences. This is probably due to the constant movement of the pedestrians in both test scenes. If a pedestrian would not move and remain still for a longer period, he/she would be adapted into the background and therefore clipped in the next detection step. This is intended because the pedestrian then does not provide any tracking information. For a video sequence where the background cannot be separated clearly enough from the foreground, this clipping algorithm would neither result in a performance boost nor deteriorate the detection rate.

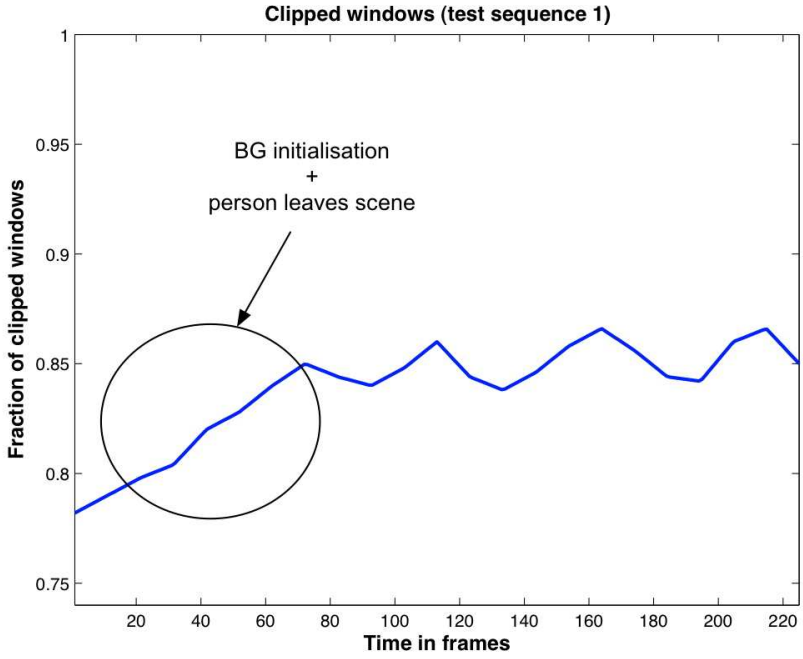

Figure 6.11: Clipping in test sequence 1.

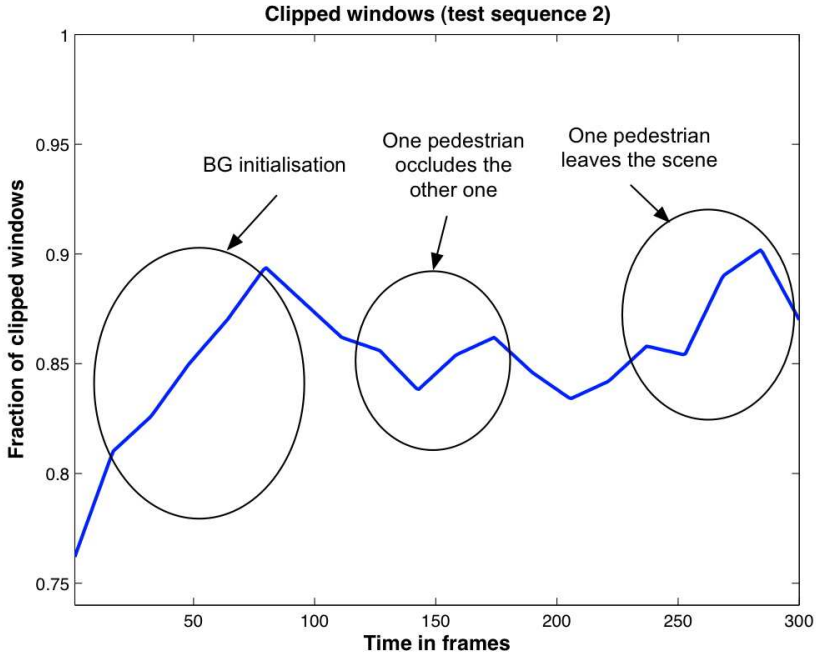

Figure 6.12: Clipping in test sequence 2.

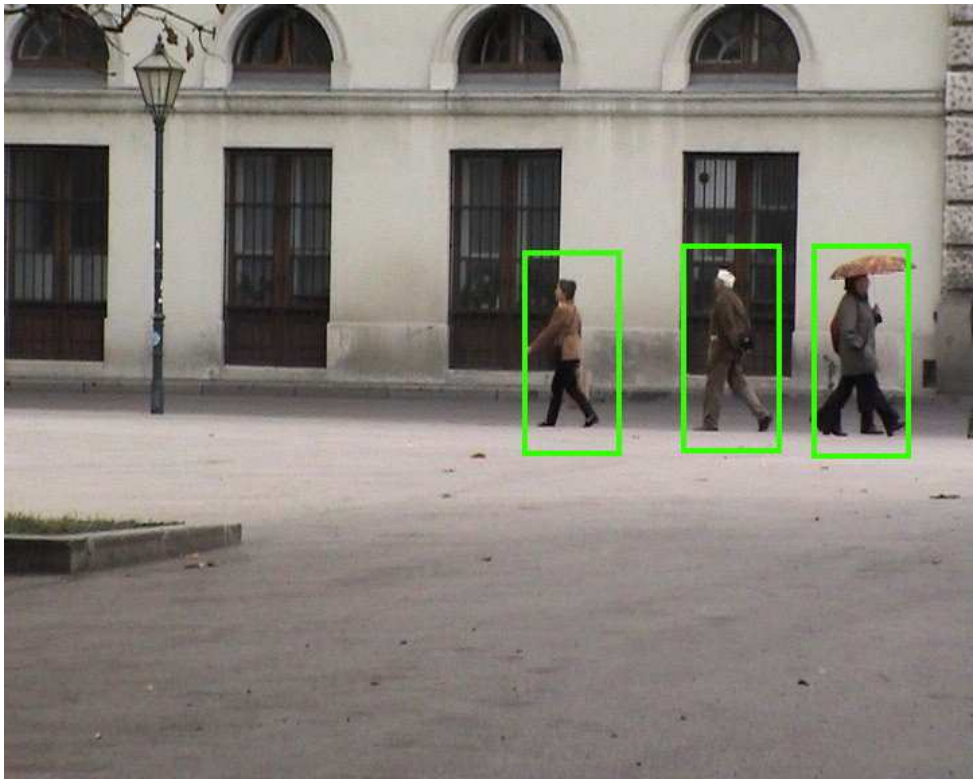

Figure 6.13: The detector in progress.

6.3.7 Detector in progress

Figure 6.13 shows the detector in progress. The green boxes show pedestrian detections where each of them represents an already merged set of one or multiple detection windows. The windows are merged as described in Section 4.5. At this step in the tracking system no tracking features are available but only the raw position and size information for each pedestrian.

6.4 Tracking features

This section presents test of the quality of the tracking features. Since this feature quality is not easily measurable we had to find a way to analyse the tracking features regarding their expressiveness and robustness.

An idea for testing the features was to lower the classification threshold for the boosted classifiers of the detector progressively[2]. By doing so the detector classifies more sub-windows as pedestrians which leads to a higher number of competing detection windows and where we can test if the features can distinguish the feature vectors well enough and support an undisturbed tracking process. The dependence between

[2]The clipping algorithm was deactivated during this test.

the decreasing threshold λ_C and the increasing number of windows classified as pedestrians gives us a method of testing the quality of the tracking features.

The tracker performed well until we reached the threshold level resulting in a false detection rate of 15 wrong detections per frame. No wrong pedestrian was accepted or real pedestrian lost above this threshold. Below this threshold, incorrectly accepted pedestrians increased rapidly and in the first test sequences two wrong pedestrians were accepted. At 17 wrong detections per frame one pedestrian was lost and not found again and at 24 wrong detections no continuous tracking was possible.

In the second test sequence three wrong pedestrians were accepted and furthermore one of three pedestrians was lost at the beginning, found again later, and tracked as a different pedestrian. The lost pedestrian is partly due to the large number of competing detection windows, but also because of the background model which needs a few frames at the beginning to initialize the background distributions correctly. At a threshold level resulting in 26 wrong detections no object could be tracked for longer than a few frames.

For both test sequences the system stays robust for a moderate number of false detections. For a higher number of false positives the system cannot separate the objects clearly enough and loses track of them or accepts non-pedestrians as pedestrians. The critical limit of 15 wrong detections in one frame in test sequence 1 is equivalent to a false positive rate of $F = 5.6 \times 10^{-3}$ for 2684 windows. Since our detector normally has a much lower false positive rate we will not have to deal with such a high number of competing windows. The results of this test indicate that the used tracking features are suitable for our application.

The object validation according to the evaluation criteria for tracking features (Section 4.3) proves to be highly effective. In both test sequences many wrong detections which contain only little tracking information are automatically rejected during the feature extraction. In the test above the object validation rejected about one third of the false detections.

6.5 Appearance model

This section describes the results of the appearance model evaluation. Here test sequence 2 was used for testing the behavior of the tracking system during occlusions and wrong detections. In this sequence the detector incorrectly labels multiple wrong windows as pedestrians and the filtering by the appearance model can be investigated. Further we have partial as well as full occlusions in this sequence which provide a way to test the state estimation of our system.

6.5.1 Occlusions

In the ground truth information of test sequence 1 and 2 all occlusions occurring were manually marked. By using this information we can evaluate how the appearance model handles these situations. In Figure 6.14 we can see the the ground truth x-coordinates of the three pedestrians while the persons are

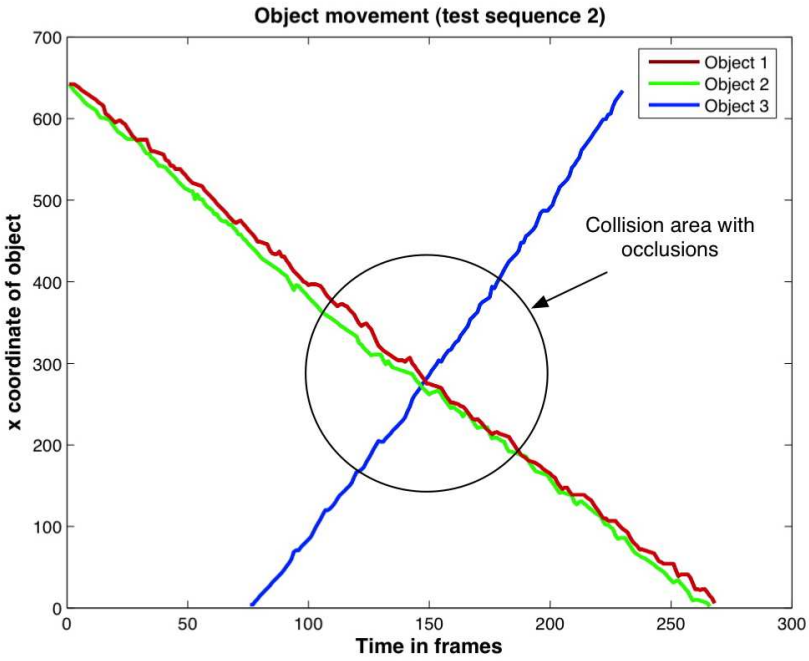

Figure 6.14: Object movements in test sequence 2.

moving through the scene. Pedestrian 1 and 2 are moving from the right side of the camera view to the left and pedestrian 3 is coming from the left. As we can see there is a point at around 140 frames when all three pedestrians are too close together and a collision occurs. This collision area is marked with a circle and should be recognized by the appearance model.

For clearness we will take a closer look at pedestrian 2. This person is coming from the right side, is occluded after around 140 frames by pedestrian 3 and after around 225 frames by a street lamp. Figure 6.15 shows the two collision areas and the manually extracted position of this object. For better visualisation we show the tracking of this object by our tracking system in a separate figure (Figure 6.16). In this figure one can see how the appearance model handles the collision with another pedestrian and with a street lamp. During the collision with the second pedestrian the positions are interpolated until both pedestrians are outside of the collision area again. Since no clear detection or extraction of the features is available in the collision areas, we will never get exactly the real object path but only an estimate. Figure 6.16 shows that the appearance model estimates the position of the pedestrian and handles this complete occlusion robustly. After the pedestrian has left the collision area, it is recognized again and its position corrected with the new detection information[3].

Since the system has no information about collisions with non-pedestrians, no collision areas are created for these regions. For the case of the street lamp no estimations are done but this partial occlusion with one quarter of the pedestrian width is quite small and has nearly no noticeable influence on the tracking

[3]Note that also outside of collision areas a deviation from the manually labelled object path is present since the reduction algorithm only provides an position estimate and depends on the quality of the current background prediction.

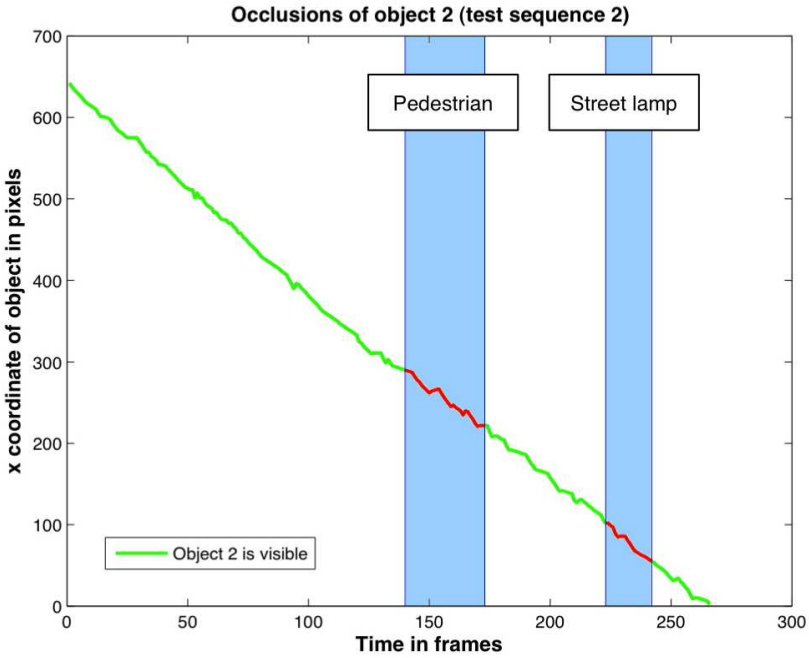

Figure 6.15: Object occlusions in test sequence 2.

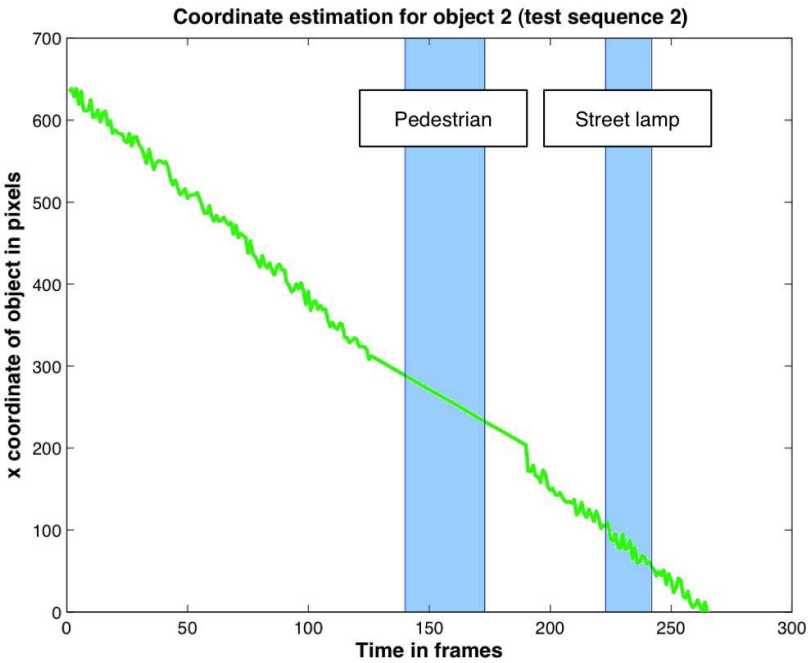

Figure 6.16: Coordinate estimation by the appearance model.

results. The detector seems to be robust against small partial occlusions since no estimation is used and the detector also finds the partially occluded pedestrian. For larger partial occlusions by a non-pedestrian, the detector cannot find the pedestrian and the appearance model has to estimate his/her state.

The estimate for the single body parts is normally determined by the detection window of the pedestrian. During occlusions, no exact window is provided and also the position and size of the body parts is estimated. In contrast to the absolute position of the pedestrian we do not achieve the same quality for the estimates of the body positions. The head, the upper body and the lower body have a periodic up-down movement which cannot be described by a linear function. Here an interpolation with a function of an angle (i.e. the sine function) could provide better estimates.

6.5.2 Filtering of wrong detections

In Section 6.3.4 the average false detection rate in both test sequences was given with one wrongly classified window in $\frac{1}{3\times10^{-5}} = 3333$ windows. In an image pyramid with 2684 windows this results on average in one wrong detection every three frames. Each wrong detection is also included in the object pool of our appearance model, but not immediately regarded as a pedestrian. The appearance model works as a filter for these wrong detections. Tested on both test sequences this filtering process proved to be very useful since not even one false detection window was accepted as a pedestrian and the false positive rate was decreased to zero. The test done in Section **??** indicates that the appearance model cannot filter all negatives if the number of wrong detections exceeds a certain limit.

6.6 Tracker in progress

The final tracking system in progress can be seen in Figure 6.17. This figure shows the tracking process in test sequence 2 which includes partial and full occlusions. In the beginning at frame 0 the system does not show any pedestrian but already in this phase pedestrians are detected and objects created by the appearance model. These objects have not reached the pedestrian state yet but have to be recognized again in subsequent frames. The final tracker needs about two seconds per frame on an Apple Powerbook G4 with 1.5 GHz and 512 MB memory. An implementation in a lower level language like C would increase this rate significantly.

Figure 6.17: The tracker in progress for test sequence 2. The frame number is shown in each image.

Chapter 7

Summary & Conclusion

I may not have gone where I intended to go, but I think I have ended up where I needed to be.

Douglas Adams

The research done for this thesis mainly focused on the developing of a fast and robust tracking system for pedestrians. For this purpose already existing algorithms were investigated and combined in this work. The approach in this system which combines a pedestrian detector and a background model for the feature extraction showed its successful operation in multiple test sequences.

The detector robustly detects the pedestrians in a video frame and also seems to be robust against partial occlusions. The cascade structure of this detector makes a fast classification process possible and the asymmetric FloatBoost algorithm results in small and efficient boosted classifiers. The pre-classification extension which was introduced in this work leads to an increase in the classification speed without influencing the classification results. In addition the proposed clipping algorithm based on the background model decreases the number of classification windows of the detector significantly and provides a very effective possibility to validate pedestrian detections after the classification process.

The background model provides a good estimate for the background and foreground in the test sequences. This can indirectly be seen in the high number of clipped windows and the feature extraction. The separation into three body parts and the extracted features reliably distinguish between the objects in our test sequences and support the tracking process. Here the separation in reliable and unreliable feature values improves the information which is used for the tracking. For partial and full occlusions when no clear features can be extracted the appearance model provides good estimates for the object states.

The resulting tracking system of this work seems to accomplish the requirements on robustness and speed. However, there are still some possibilities for improvement. The estimates done by the appearance model

could be improved by the use of a better interpolation or filter i.e. a Kalman filter. All parts of the tracking system are strongly dependent on the background model which only achieves a moderate result for images with a low resolution or a high compression rate. The obtained results in the test sequences are still suitable for our application but since the tracking system strongly relies on the quality of the background model a future goal should be to improve the model for example by the use of a better colour model or with shadow detection. Additional performance tests on standard test sequences should be done to be able to better compare the tracking results.

Also a low-level implementation to increase the speed of the tracking system provides possibilities for future work. Since most implementations are written in a low-level language this would also provide a good possibility to compare the velocity of our system with others and to evaluate the performance increase due to the introduced speed optimisations.

Acknowledgements

I would maintain that thanks are the highest form of thought, and that gratitude is happiness doubled by wonder.

G. K. Chesterton

I would like to gratefully and honestly thank all persons who supported me during the intensive work on this thesis. In particular, I would like to express deepest gratitude to my advisor, Allan Hanbury, for allowing me to work in a very interesting research area, for his full support and patience. Furthermore, I would like to sincerely thank Brian Lovell for his support and interesting discussions during my time at his department.

For their valuable disscussions, friendship and numerous coffee breakes I would like to thank Remy Mevel, Daniel Wilhelm, Reinhard Klein and Amin Abbosh who furthermore made my stay in Australia unforgettable.

Finally, I would like to thank my family. I am greatly indebted to my mother who provided me support, friendship, great freedom during my studies, delicious dinners and tons of chocolate. Above all, I cannot express my full gratitude to my partner, Julia, who constantly supported me through this entire thesis.

Bibliography

[AMGC02] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, Feb 2002.

[BBBD05] Massimo Bertozzi, Emanuele Binelli, Alberto Broggi, and Michael Del Rose. Stereo vision-based approaches for pedestrian detection. In *Procs. Intl. IEEE Workshop on Object Tracking and Classification in and Beyond the Visible Spectrum*, San Diego, USA, June 2005.

[BBDL05] Massimo Bertozzi, Alberto Broggi, Michael Del Rose, and Andrea Lasagni. Infrared stereo vision-based human shape detection. In *Proc. IEEE Intelligent Vehicles Symposium 2005*, pages 23–28, Las Vegas, USA, June 2005.

[BBF+04] M. Bertozzi, A. Broggi, A. Fascioli, A. Tibaldi, and R. Chapuis F. Chausse. Pedestrian localization and tracking system with Kalman filtering. In *IEEE Intelligent Vehicles Symposium 2004*, pages 584 – 589, 2004.

[BCT04] Tilo Burghardt, Janko Calic, and Barry Thomas. Tracking animals in wildlife videos using face detection. In *European Workshop on the Integration of Knowledge, Semantics and Digital Media Technology*, October 2004. London, U.K.

[CD99] Ross Cutler and Larry Davis. Real-time periodic motion detection, analysis and application. In *Proc. of IEEE Conference on Computer and Pattern Recognition*, pages 326–331, 1999. Fort Collins, USA, 1999.

[CJSW01] H.D. Cheng, X.H. Jiang, Y. Sun, and J. Wang. Color image segmentation: advances and prospects. *Pattern Recognition*, 34(6):2259–2281, 2001.

[CRH01] Y. Chen, Y. Rui, and T. Huang. Jpdaf based HMM for real-time contour tracking. In *Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition*, 2001.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2nd edition, 2000.

[EDD01] Ahmed M. Elgammal, Ramani Duraiswami, and Larry S. Davis. Efficient non-parametric adaptive color modeling using fast Gauss transform. In *CVPR (2)*, pages 563–570, 2001.

[FHT01] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. In *The Annals of Statistics*, volume 28, pages 337–374, October 2001.

[FM99] A. R. Francois and G. G. Medioni. Adaptive color background modeling for real-time segmentation of video streams. In *Proceedings of the International Conference on Imaging Science, Systems, and Technology*, pages 227–232, 1999.

[FS96] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *13th International Conference on Machine Learning*, pages 148–156, 1996.

[GG02] D. M. Gavrila and J. Giebel. Shape-based pedestrian detection and tracking. In *IEEE Intelligent Vehicle Symposium, 2002.*, volume 1, pages 8 – 14, 17-21 June 2002.

[GGB+02] F. Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P. Nordlund. Particle filters for positioning, navigation, and tracking. In *IEEE Transactions on Signal Processing*, volume 50, pages 425–435, Feb 2002.

[GM76] A. H. Gray and J. D. Markel. Distance measures for speech processing. In *IEEE Trans. Acoust., Speech, Signal Processing, ASSP-24*, pages 380–391, 1976.

[HB98] Gregory D. Hager and Peter N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1025–1039, 1998.

[HS67] Gerald J. Hahn and Samuel S. Shapiro. *Statistical Models in Engineering.* John Wiley & Sons, 1967.

[HS02] Allan Hanbury and Jean Serra. A 3d-polar coordinate colour representation suitable for image analysis. Technical Report PRIP-TR-077, PRIP, TU Vienna, Vienna, 2002.

[IB98] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. In *International Journal of Computer Vision*, volume 29(1), pages 5–28, 1998.

[JV03] Michael Jones and Paul Viola. Fast multi-view face detection. Technical Report TR2003-096, Mitsubishi Electric Research Laboratories (MERL), June 2003.

[Kai67] T. Kailath. The divergence and Bhattacharyya distance measures in signal selection. In *IEEE Trans. Comm. Technology*, volume 15, pages 52–60, Feb. 1967.

[KCT04] S. J. Krotosky, S. Y. Cheng, and M. M. Trivedi. Face detection and head tracking using stereo and thermal infrared cameras for 'smart' airbags: A comparative analysis. In *7th IEEE Conf. on Intelligent Transportation Systems*, October 2004.

[LKP03] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Pattern Recognition, 25th DAGM Symposium, Magdeburg, Germany, September 10-12, 2003*, pages 297–304, 2003.

[LKW91] Brian C. Lovell, Peter J. Kootsookos, and R. C. Williamson. The circular nature of discrete-time frequency estimates. In *IEEE International Conference on ASSP*, pages 3369–3372, Toronto, May 1991.

[LLK03] Rainer Lienhart, Luhong Liang, and Alexander Kuranov. A detector tree of boosted classifiers for real-time object detection and tracking. In *IEEE ICME2003*, volume 2, pages 277–280, July 2003.

[LM02] Rainer Lienhart and Jochen Maydt. An extended set of Haar-like features for rapid object detection. In *IEEE ICIP2002*, volume 1, pages 900–903, Sept. 2002.

[Lov91] B. C. Lovell. *Techniques for Non-Stationary Spectral Analysis*. PhD thesis, University of Queensland, 1991. Brisbane.

[LW04] K. Levi and Y. Weiss. Learning object detection from a small number of examples: the importance of good features. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition 2004*, pages 53–60, 2004. Washington DC.

[LZSZ02] S. Li, Z. Zhang, H. Shum, and H. Zhang. Floatboost learning for classification. In *Proceedings of The 16-th Annual Conference on Neural Information Processing Systems (NIPS)*, pages 993–1000. MIT Press, Dec 2002. Vancouver, Canada.

[Mar72] K. V. Mardia. *Statistics of directional data*. Academic Press, London, 1972.

[MP01] O. Masoud and N.P. Papanikolopoulos. A novel method for tracking and counting pedestrians in real-time using a single camera. In *IEEE Transactions on Vehicular Technology*, volume 50 of *5*, pages 1267–1278, September 2001.

[OB04] Eng-Jon Ong and Richard Bowden. A boosted classifier tree for hand shape detection. In *Sixth IEEE International Conference on Automatic Face and Gesture Recognition (FGR 2004), May 17-19, 2004, Seoul, Korea*, pages 889–894, 2004.

[OPS+98] M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio. Pedestrian detection using wavelet templates. In *IEEE ICCV*, pages 555–562, 1998. Bombay, India.

[PDD00] V. Philomin, R. Duraiswami, and L. Davis. Pedestrian tracking from a moving vehicle. In *Intelligent Vehicles Symposium, 2000. IV 2000.*, pages 350–355, 2000.

[PT03] Fatih Porikli and Oncel Tuzel. Human body tracking by adaptive background models and mean-shift analysis. In *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, March 2003.

[SKK00] Yoichi Sato, Yoshinori Kobayashi, and Hideki Koike. Fast tracking of hands and fingertips in infrared images for augmented desk interface. In *Proc. of IEEE Automatic Face and Gesture Recognition (FG2000)*, pages pp.462–467, 2000.

[SL00] Nicu Sebe and Michael S. Lew. A maximum likelihood investigation into color indexing. In *Proceedings Visual Interface 2000*, pages 101–106, 2000.

[Smi78] Alvy Ray Smith. Color gamut transform pairs. *SIGGRAPH Comput. Graph.*, 12(3):12–19, 1978.

[Ste63] M. A. Stephens. Random walk on a circle. In *Biometrika*, 50, pages 385–390, 1963.

[SWTO04] Caifeng Shan, Yucheng Wei, Tieniu Tan, and Frédéric Ojardias. Real time hand tracking by combining particle filtering and mean shift. In *Sixth IEEE International Conference on Automatic Face and Gesture Recognition (FGR 2004), May 17-19, 2004, Seoul, Korea*, pages 669–674, 2004.

[TT03] Marko Tkalcic and Jurij Tasic. Colour spaces - perceptual, historical and applicational background. In Baldomir Zajc and Marko Tkalcic, editors, *Eurocon 2003 Proceedings*. IEEE Region 8, September 2003.

[VJ01a] Paul Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 511–518, December 12-14 2001. Kauai, Hawaii.

[VJ01b] Paul A. Viola and Michael J. Jones. Robust real-time object detection. In *Second International Workshop on Statistical Learning and Computational Theories of Vision-Modelling, Learning, Computing and Sampling*, July 2001.

[VJ02] P. Viola and M. Jones. Fast and robust classification using asymmetric Adaboost and a detector cascade. In *NIPS2002*, 2002.

[VJS03] Paul A. Viola, Michael J. Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. In *Proc. of IEEE International Conference on Computer Vision ICCV03*, volume 2, pages 734–741, 2003.

[WADP97] Christopher Richard Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.

[WAHL04] Bo Wu, Haizhou Ai, Chang Huang, and Shihong Lao. Fast rotation invariant multi-view face detection based on Real Adaboost. In *Sixth IEEE International Conference on Automatic Face and Gesture Recognition (FGR 2004), May 17-19, 2004, Seoul, Korea*, pages 79–84, 2004.

[WRM04] J. Wu, J. Rehg, and M. Mullin. Learning a rare event detection cascade by direct feature selection. In *Advances in Neural Information Processing Systems 16 (2004)*, 2004.

[ZT00] L. Zhao and C.E. Thorpe. Stereo- and neural network-based pedestrian detection. *ITS*, 1(3):148–154, September 2000.