DIOGO CARBONERA LUVIZON

VEHICLE SPEED ESTIMATION BY LICENSE PLATE DETECTION AND TRACKING

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Curitiba PR
Agosto 2015

DIOGO CARBONERA LUVIZON

VEHICLE SPEED ESTIMATION BY LICENSE PLATE DETECTION AND TRACKING

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Área de concentração: *Engenharia de Sistemas Computacionais*

Orientador: Rodrigo Minetto
Co-orientador: Bogdan Tomoyuki Nassu

Curitiba PR
Agosto 2015

Dados Internacionais de Catalogação na Publicação

L976v 2015

Luvizon, Diogo Carbonera
Vehicle speed estimation by license plate detection and tracking / Diogo Carbonera Luvizon.-- 2015.
xxiv, 73 p.: il.; 30 cm

Texto em inglês, com resumo em português.
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada, Curitiba, 2015.
Bibliografia: p. 69-73.

1. Veículos - Velocidade. 2. Medição. 3. Detectores. 4. Placas de automóveis. 5. Processamento de imagens - Técnicas digitais. 6. Vídeo digital. 7. Reconhecimento ótico de caracteres. 8. Computação - Dissertações. I. Minetto, Rodrigo, orient. II. Nassu, Bogdan Tomoyuki, coorient. III. Universidade Tecnológica Federal do Paraná - Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD 22 -- 621.39

Biblioteca Central da UTFPR, Câmpus Curitiba

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 33

Aos 21 dias do mês de agosto de 2015 realizou-se na sala B-204 a sessão pública de Defesa da Dissertação de Mestrado intitulada "Medição de Velocidade de Veículos Automotores por Processamento Digital de Imagens", apresentada pelo aluno **Diogo Carbonera Luvizon** como requisito parcial para a obtenção do título de Mestre em Computação Aplicada, na área de concentração "Engenharia de Sistemas Computacionais", linha de pesquisa "Processamento Gráfico".

Constituição da Banca Examinadora:

Prof. Dr. Rodrigo Minetto, UTFPR - CT (Presidente) _____

Prof. Dr. Bogdan Tomoyuki Nassu, UTFPR – CT _____

Prof. Dr. Jorge Stolfi, UNICAMP _____

Profª. Drª. Leyza Baldo Dorini UTFPR - CT _____

Prof. Dr. Ricardo Dutra da Silva, UTFPR – CT _____

Em conformidade com os regulamentos do Programa de Pós-Graduação em Computação aplicada e da Universidade Tecnológica Federal do Paraná, o trabalho apresentado foi considerado _____ (aprovado/reprovado) pela banca examinadora. No caso de aprovação, a mesma está condicionada ao cumprimento integral das exigências da banca examinadora, registradas no verso desta ata, da entrega da versão final da dissertação em conformidade com as normas da UTFPR e da entrega da documentação necessária à elaboração do diploma, em até _____ dias desta data.

Ciente (assinatura do aluno): _____

(para uso da coordenação)

A Coordenação do PPGCA/UTFPR declara que foram cumpridos todos os requisitos exigidos pelo programa para a obtenção do título de Mestre.

Curitiba PR, _____/_____/_____ _____

"A Ata de Defesa original está arquivada na Secretaria do PPGCA".

A Gabriela, minha amada esposa.

Agradecimentos

Agradeço primeiramente a Gabriela, pelo amor, incentivo, paciência, e compreensão durantes as muitas horas em que me ausentei para me dedicar a este trabalho.

Agradeço muito ao meu orientador, Dr. Rodrigo Minetto, pela dedicação e pela ajuda constante, sem a qual não seria possível a realização deste trabalho. Agradeço também ao meu co-orientador, Dr. Bogdan T. Nassu, por toda ajuda e apoio.

Agradeço à minha mãe, Rosa Maria Carbonera, que nunca poupou esforços para oferecer a melhor educação para os filhos. Agradeço também ao meu pai, Osvaldo Luvizon, que me ensinou a ser uma pessoa honesta e honrada. Agradeço aos meus queridos irmãos, Aline e Rafael, por fazerem parte da construção do meu caráter.

Agradeço aos engenheiros Henrique Camargo, Oliver Miranda, Gabriel Panasco e Felipe Marcondes, meus amigos e colegas de trabalho, que participaram ativamente na concepção das primeiras ideias que resultaram neste trabalho.

Agradeço ao projeto Universal-CNPq, número do processo 444789/2014-6, pelo apoio financeiro.

Finally, I would like to thank Dr. Milan Mirković, Dr. Dubravko Ćulibrk, Dr. Dejan Vukobratovic, and Dr. Andraš Anderla, all from the University of Novi Sad, for accepting me in the QoSTREAM project as an exchange student.

Resumo

Sistemas de controle de velocidade são utilizados em vários países para fiscalizar o cumprimento dos limites de velocidade, prevenindo assim acidentes de trânsito. Muitos desses sistemas são baseados em tecnologias intrusivas que requerem processos de instalação e manutenção complexos, geralmente atrapalhando o trânsito. Neste projeto, propõe-se um sistema não intrusivo para estimativa da velocidade de veículos baseado em vídeo. O sistema proposto detecta veículos em movimento utilizando um detector de movimento otimizado. Aplicou-se um detector de texto especializado para localizar a placa dos veículos, a qual foi utilizada para seleção e rastreamento de pontos estáveis. Os pontos rastreados são então filtrados e retificados para remoção do efeito da perspectiva. A velocidade dos veículos é estimada comparando-se a trajetória dos pontos rastreados com dimensões conhecidas no mundo. Para os testes, utilizou-se aproximadamente cinco horas de vídeos em diferentes condições, capturados por uma câmera de baixo custo posicionada a 5,5 metros de altura. Os vídeos capturados contém mais de 8.000 veículos distribuídos em três pistas diferentes, com as velocidades reais para cada veículo obtidas a partir de um detector por laço indutivo. O detector de placas proposto foi comparado com três outros métodos no estado da arte e obteve os melhores resultados de performance para os nossos vídeos, com precisão (*precision*) de 0,93 e coeficiente de revocação (*recall*) de 0,87. A estimativa de velocidade dos veículos apresentou erro médio de -0,5 km/h, permanecendo dentro da margem de +2/-3 km/h, determinada por agências reguladoras em vários países, em 96,0% dos casos.

Palavras-chave: medição de velocidade de veículos, detecção de placa, rastreamento de características.

Abstract

Speed control systems are used in most countries to enforce speed limits and, consequently, to prevent accidents. Most of such systems are based on intrusive technologies which require complex installation and maintenance, usually causing traffic disturbance. In this work, we propose a non-intrusive video-based system for vehicle speed estimation. The proposed system detects moving vehicles using an optimized motion detector. We apply a specialized text detector to locate the vehicle's license plate region, in which stable features are selected for tracking. The tracked features are then filtered and rectified for perspective distortion. Vehicle speed is estimated by comparing the trajectory of the tracked features to known real world measures. For our tests, we used almost five hours of videos in different conditions, captured by a single low-cost camera positioned at 5.5 meters height. The recorded videos contain more than 8,000 vehicles, in three different road lanes, with associated ground truth speeds obtained from an inductive loop detector. We compared our license plate detector with three other state-of-the-art text detectors, and our approach has shown the best performance for our dataset, attaining a precision of 0.93 and a recall of 0.87. Vehicle speeds were estimated with an average error of -0.5 km/h, staying inside the +2/-3 km/h limit determined by regulatory authorities in several countries in over 96.0% of the cases.

Keywords: vehicle speed measurement, license plate detection, feature tracking.

Contents

List of Figures

List of Tables

List of Symbols

Chapter 1

$\mathbb{V}$	Video
$\mathfrak{D}$	Domain of video or images
M	Number of rows in video frames
N	Number of columns in video frames

Chapter 3

$\mathbb{I}$	Image
$\mathbb{H}$	Motion history image
$\mathbb{D},\mathbb{E}$	Binary images
Δt	Variation of time
ξ	Pixel threshold
τ	Number of frames interval
χ	Vertical projection profile without smoothing
φ	Conventional rolling mean smoothing function
η	Size of smoothing window
κ	Number of iterations
Ψ	Vertical projection profile
$\mathscr{R}$	Region of interest
ρ	Minimum percentage of inclination
R	Rising phase
F	Falling phase
S_a	Ascending slope
S_d	Descending slope
A	Ascending border
D	Descending border
s	Subsampling ratio

Chapter 4

$\mathbb{I}$	Image
$\mathbb{G}$	Grey scale edge image
$\mathbb{E}$	Binary edge image
$\mathbb{S}$	Binary dilated edge image
$\mathbb{V}$	Video
$\mathscr{R}$	Region of interest
γ	Edge extraction threshold
w, h	Width and height of a region
b	Structuring element for morphological operations
t_1, t_2, t_3	Character grouping parameters
υ	Threshold for a positive text region classification

Chapter 5

f	Number of tracked frames
$\mathbb{I}, \mathbb{J}$	Images
$\mathbb{T}$	Template region
$\mathbb{V}$	Video
$\mathfrak{D}$	Domain of video or images
Δt	Variation of time
λ	Minimum eigenvalue
E	Residual error
ℓ	Number of pyramidal levels
Ω	Window image
γ	Constant of window expansion
μ, σ	Mean and standard deviation of motion vectors

Chapter 6

p_w	Point in the world
$\hat{p}_w$	Point in the world plane
p_i	Point in the image plane
p_c	Point in the camera coordinate system
f	Camera focal length
H	Homograph matrix
Δt	Variation of time
c	Speed constant factor
s	Instantaneous vehicle speed
S_a	Average vehicle speed
S_f	Final vehicle speed

List of Abbreviations

2D	Two dimensional
3D	Three dimensional
AVC	Advanced Video Coding
CIE	*Commission Internationale de l'Éclairage*
CMOS	Complementary metal–oxide–semiconductor
EA	Event Area
GHz	Gigahertz
LASER	Light Amplification by Stimulated Emission of Radiation
MEI	Motion Energy Image
MHI	Motion History Image
NTSC	National Television System Committee
OCR	Optical Character Recognition
PPGCA	Programa de Pós-Graduação em Computação Aplicada
RAM	Random access memory
RGB	Color model (Red Green Blue)
ROI	Region of Interest
USA	United States of America
UTFPR	Universidade Tecnológica Federal do Paraná
XML	Extensible Markup Language

Chapter 1

Introduction

Speed control systems are used in most countries to enforce speed limits and, consequently, to prevent accidents. Most of such systems use specialized sensors to detect and to measure the vehicle's speed, and cameras to take photographs or to record videos when speed limit violations occur. Data produced by these systems may also be used by traffic control systems to determine the traffic intensity, length of vehicle queues, average speed, etc.

As described by Mathew [1], systems for vehicle detection and speed measurement are based on *intrusive* or *non-intrusive* technologies. Intrusive systems, usually based on inductive loop detector, are cheap, but have some application disadvantages, such as high installation cost, traffic disturbance during installation, and damage caused by wear and tear or asphalt maintenance. Non-intrusive sensors have been increasingly used for acquisition of traffic flow data, representing an emergent field of research, especially image based sensors, due to their capability to extract more information than other kinds of sensors. The advantages and disadvantages of some technologies are described below.

The main intrusive sensors are passive magnetometers, pneumatic tubes, inductive loop detectors, and piezoelectric sensors. Passive magnetometers are permanently mounted within holes in the road and can detect the variation of the Earth's magnetic field by the presence of a passing metal object. However, it cannot detect stopped vehicles. Pneumatic tubes are usually installed perpendicular to the traffic flow and produce an electrical signal generated by a switch triggered by air pressure. However, they are very fragile and can be easily damaged. Inductive loop detectors are coils buried in grooves in the road surface connected to an electronic oscillator circuit. The metal part of a moving vehicle changes the electrical properties of the circuit, triggering an event. It is the most common kind of sensor, but it can be damaged by street maintenance or humidity infiltration, and it is not suitable for metallic bridges and brick streets. Lastly, piezoelectric sensors detect a change in voltage caused by pressure exerted on the sensor. They can be used to detect weight-in-motion, but are less accurate than the previously presented sensors, and should be replaced once every three years.

Non-intrusive technologies are based on video image detection, passive or active infrared sensors, LASER detectors, ultrasonic Doppler meters or aerial photography. Such sensors can be roadside mounted, installed underside a gantry or a bridge, or can stay at ground level pointing perpendicular to the road. The advantage of video image detection is the possibility to extract a lot of information from vehicles, such as color, model, size, direction of motion, speed, and license plate identification. However, video image detectors are susceptible

to occlusions, as any other non-intrusive technology, and may suffer with bad weather conditions or low light environments. Infrared and LASER detectors work measuring the time of the light reflection. These systems can be used to count vehicles and estimate speed both in day and night conditions, as long as there is no haze or smoke. However, their cost is relatively high if compared with other non-intrusive solutions. Ultrasonic Doppler-based systems use the frequency shift of acoustic waves to calculate the speed of moving objects. The speed measurement is very accurate, but they cannot detect stationary vehicles, and are sensitive to spurious returns from adjacent objects [1].

1.1 Motivation

In many speed control systems, some specialized sensors (*e.g.* loop detectors, ultrasonic Doppler, etc.) are used to trigger a video camera to record the license plates of vehicles that exceed the speed limit. However, we can simplify these systems by extracting the speed information from the already available video frames. Thus, we can obtain a non-intrusive system with significantly reduced costs. As a byproduct of the uninterrupted video recording, in a future work, it will be possible to identify the license plates of the passing vehicles and to estimate the road flow volume and average speed.

1.2 Statement of the problem

The input data for the license plate detection and speed estimation problem is a digital video $\mathbb{V}$, which is defined as a sequence of n images (*frames*) $\mathbb{V}(0), \mathbb{V}(1), \ldots, \mathbb{V}(n-1)$ with a common domain $\mathfrak{D}$, equally spaced in time. The video is captured by a fixed overhead camera, which was installed at a position to properly record the rear license plates in three adjacent lanes. The experimental setup is shown in Figure 1.1. A sample image from this setup is shown in Figure 1.2.

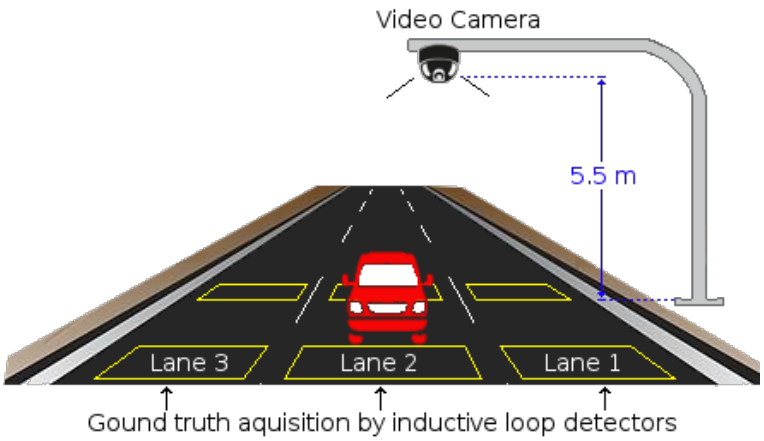
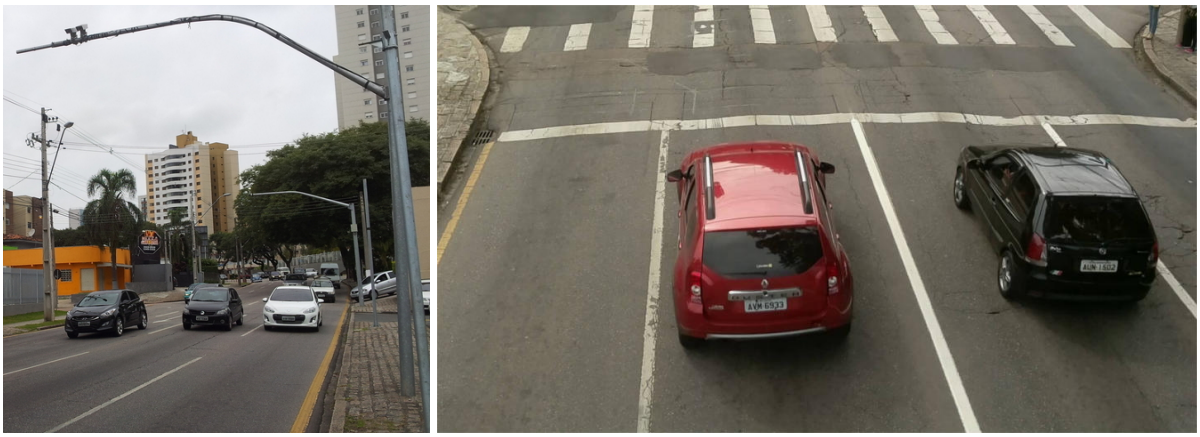


Figure 1.1: Experimental setup.

The outputs are: a set of license plate candidate regions, the tracked trajectories of these regions in successive frames, and the speed estimation for each vehicle.



(a) (b)

Figure 1.2: Sample images from our experimental setup: in (a) a picture of our camera and in (b) a sample image captured.

Some assumptions are made about the scene and the problem domain: each lane lies on a plane; the vehicles, in the region of speed measurement, move at a constant speed and with a straight trajectory from the lower to the upper part of the image; and the license plates have similar dimensions and are assumed to be roughly at the same distance from the ground.

In order to evaluate the system performance, a ground truth dataset was collected from a high precision speed meter based on inductive loop detectors, properly calibrated and approved by the Brazilian national metrology agency (Inmetro).

We assume that a *digital image* is a 2D array of color values (*pixels*). For monochromatic images, we assume that each pixel is a real number between 0 and 1, proportional to the light that falls on the corresponding sensor. For color images, a pixel consists of a separate measurement for each *color channel* (spectral band), *e.g.* red (R), green (G) and blue (B) channels. It may be convenient to convert RGB color images to monochromatic ones. For this purpose we use the CIE 601-1 luminance formula $0.299\,R + 0.587\,G + 0.114\,B$.

The *domain* of an image $\mathbb{I}$ with N columns and M rows is the axis-aligned rectangle with upper left corner at $(0,0)$ and lower right corner at (N,M). The frame (or image) *coordinates* (x,y) specify the position of points in the image domain. Note that the Y axis points down in the normal image view. For the sake of simplicity, we use $\mathbb{I}(u)$ and $\mathbb{I}(\vec{u})$ to refer to the pixel indexed by a two-dimensional point $u = (x,y)$ and vector $\vec{u} = (x,y)$, respectively.

1.3 System overview

The proposed method is divided into five main parts, as shown in Figure 1.3. The first step is an optimized motion detection algorithm that identifies and labels each region of interest, which supposedly contains a vehicle license plate. These regions are then fed to a license plate detector, which returns a set of axis-aligned rectangular sub-images. Features are then extracted from each rectangular sub-image [2], and tracked using the Kanade-Lucas-Tomasi (KLT) algorithm [3, 4]. To cope with large displacements, *e.g.* motion of a high-speed vehicle, we used for the initial motion estimation the Scale-Invariant Feature Transform [5]

algorithm. Finally, the vehicle speed is estimated by comparing the trajectory of the tracked features to known real world measures.

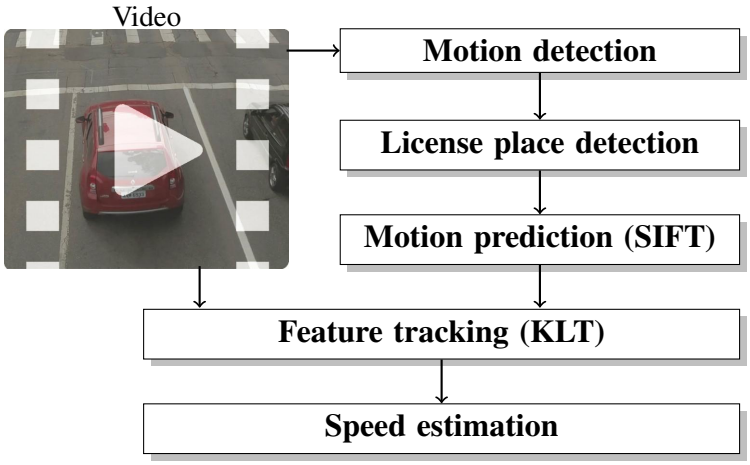


Figure 1.3: Overview of the proposed system.

1.4 Publication

A preliminary version of the system described in this thesis was presented at the International Conference on Acoustics, Speech and Signal Processing - ICASSP:

- *Vehicle speed estimation by license plate detection and tracking*
  Diogo C. Luvizon, Bogdan T. Nassu and Rodrigo Minetto.
  IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP), 2014.

1.5 Structure of the thesis

This master thesis is divided as follows. In Chapter 2, we discuss previous work on vehicle speed estimation and license plate detection. In Chapters 3, 4 and 5 we present our motion detection, license plate detection and vehicle tracking methods, respectively. The speed estimation method is presented in Chapter 6. The experimental evaluation and results are reported in Chapter 7. Finally, Chapter 8 concludes this document and gives directions for future work.

Chapter 2

Related work

There is an extensive literature on vehicle speed estimation [6–24]. Most approaches are based on specialized speed sensors [6], or are dedicated to specific contexts, such as speed estimation from vehicle's headlight in night scenes [7, 8], speed estimation using the signal from mobile telecommunication network [9], or speed estimation from a car-mounted camera to avoid collisions [10]. However, an exhaustive review of such methods is far beyond the scope of this thesis.

Comparatively little has been published about vehicle speed estimation from a digital video recorded by a ground fixed overhead camera (our primary interest in this master thesis). Such related approaches are reviewed in Section 2.1. Furthermore, as our system needs a vehicle license plate detector, we review some state-of-the-art methods in this context in Section 2.2.

2.1 Vehicle speed estimation

Vehicle speed estimation problem in digital videos is closely related to vehicle detection and tracking problems. However, solving the latter problems, in an outdoor scene context, is a challenging task. The reasons may include strong background cluttering, difficult illumination conditions, cast and cloud shadows, rainfall, poor image resolution, image noise and changes in weather conditions. Thus, we describe the methods in this section based on how they solve the stated problems.

In 2000, Dailey *et al.* [11] claimed that exact calibration is not necessary to estimate time-average traffic speed, once we have the geometric relationships inherently available in the images. Initially, the regions of interest are detected by using a simple frame difference and the Sobel operator to obtain an edge image with the object motion. This edge image is then enhanced by a morphological closing operation and fed to a convex hull algorithm [25] to create image blobs. These blobs are then tracked by enforcing collinearity between their centroids. The algorithm assumes that all vehicles have roughly the same length, thus the average dimension is used as a real world reference to compute the speed. For each blob at a given position, the algorithm compute the speed by dividing the mean inter-frame distance travelled by blobs by the frame interval. Similarly, Madasu and Hanmandlu [12] proposed a method to estimate speed of vehicles using uncalibrated camera based on the assumption that vehicles have roughly the same length. Differently from Dailey *et al.*, Madasu and Hanmandlu

proposed the use of the KLT algorithm [3,4] to track features selected by the criteria of Shi and Tomasi [2]. The disadvantage of both methods relies on the assumption of a constant length for all vehicles. For example, in our dataset, we have a wide range of vehicles with different lengths, like motorcycles, buses, trucks and other kinds of ordinary vehicles.

In 2001, Garibotto *et al.* [13] proposed to use high level features, such as clusters of matched license plate characters, to compute the instantaneous speed of vehicles. First, they recovered the 3D distance between two positions of the license plates in real world from the projected distance vector measured in the image plane. An OCR is then used to detect and to recognize the license plate characters in both images. The speed is estimated by using the distance between the center of mass of each cluster of recognized characters, the time interval between two consecutive frames and a mapping from the camera parameter system to the object coordinate system. As reported by the authors, the main drawback of this approach is the need of an extremely effective license plate recognition system.

In 2007, Zhiwei *et al.* [14] noted that the image background can be computed by a median filter applied in each image pixel across a sequence of images, but this process is space and time consuming. The authors suggested a background updating model that approximates a median filter by increasing or decreasing each image pixel by a constant value for each new frame. They employed a pinhole camera model to project the road plane into a rectified image, enabling the computation of vehicle displacements in real world and consequently the speed estimation. Similarly, in 2008, Maduro *et al.* [15] proposed a vehicle speed estimation method based on image rectification using two vanishing points. Vehicles are identified by background subtraction and the bounded blobs are tracked by using a Kalman filter. The speed is estimated by considering the displacement of the bottom margin of blobs. Due to their straightforward detection approaches, both methods are susceptible to bad weather or lighting change conditions.

In 2009, Palaio *et al.* [16] proposed to track vehicles using particle filtering. The vehicles are selected by a foreground/background segmentation process and are then represented by a descriptor based on window location, window color components, horizontal and vertical derivatives, and the Laplacian of the grey image. The region descriptors are then used to track vehicles using a particle filtering. Finally, the vehicle's positions are projected to a rectified image for speed estimation. The authors use the same rectification process employed by Maduro *et al.* [15].

In 2008, Lin *et al.* [17] noticed that for any fixed time interval, the displacement between vehicles in images is proportional to the amount of blur caused by the imaging process. Thus, if the parameters of the motion blur (*e.g.* , the motion length and the orientation), and the relative position between the camera and the object can be identified, the speed of vehicles can be estimated according to the imaging geometry. To compute the blur length in the horizontal direction, a subimage enclosing the detected vehicle is manually extracted, or obtained by a background subtraction, from the original motion blurred image taken by a side view stationary camera. Edge detection is then applied on this subimage to find the left and right blur regions. Ideally, there will be two ramps in the edge image with the same width. Thus, the blur length can be obtained by taking the average of those ramp widths. As reported by the authors, the major limitation of their method is the assumption that there are detectable sharp edges in the motion deblurred image. That is, the speed detection algorithm might not perform well if the environmental lighting is not sufficient (*e.g.*, for sunset or rainy days) or if the shadow of vehicles is significant.

In 2010, Dogan *et al.* [18] proposed a speed estimation method for side view images taken by a single stationary uncalibrated camera. The authors first used image subtraction and thresholding to eliminate the static background. Then, distinctives features were selected within the vehicle region with the Lucas-Tomasi algorithm [4] and tracked with the Lucas-Kanade optical flow algorithm [3]. The speed estimation was computed by using the motion vectors obtained from tracking, the time interval between two consecutive frames, and a mapping from the video image coordinate system to the object coordinate system. A drawback of this approach is that all the motion vectors are supposed to belong to the same vehicle, thus, it can handle only a single vehicle at a time.

In 2010, Czajewski and Iwanowski [19] proposed a speed measurement method based on license plate recognition. They used an elementary license plate recognition algorithm based on an adaptive threshold of the input image, followed by a contour extraction and a character grouping step. If the license plate is detected and recognized in two consecutive frames, the method estimates the vehicle position for these two frames in real world and compute the speed by using the travelled distance in a frame interval. Similar to the approach presented by Garibotto *et al.* [13], the efficiency of this method relies on the effectiveness of the license plate recognition method.

There are also several other approaches to estimate the speed of vehicles in highways [20–24]. They assume that the camera is positioned far enough from vehicles, generating images with a wide range of view, so that it is easier to identify the entire vehicle. Thus, they are based on background subtraction and blob detection techniques, with the speed being estimated from the displacement of blobs (considering the centroid or part of the contour). However, in our experimental setup, the camera is closer to the vehicles. That makes blob analysis very sensitive to lighting variations and shadows. On the other hand, the closer view allows to detect and track distinctive features inside the license plate region. These differences make difficult a direct comparison between our system and the above mentioned work. Although, we tested a particle filter blob tracking algorithm in order to evaluate its performance in such a scenario.

2.2 License plate detection

License plate detection is an active field of research with many algorithms and extensive literature. The surveys of Anagnostopoulos *et al.* [26] and Du *et al.* [27] review state-of-the-art systems up to 2013. The difficulties in license plate detection mainly come from poor maintenance, occlusion, orientation and scale variations, complexity of the background, irregular illumination, low contrast, low image resolution, and motion blur.

According to Du *et al.* [27], most license plate detection algorithms can be categorized based on the type of feature extracted from the image: boundary or edge, texture, color, shape, and geometry. In this section, we review three state-of-the-art algorithms based on edges, texture, shape and geometry attributes. We consider the Zheng *et al.* algorithm [28], which was specifically designed for license plate detection, as well as the Stroke Width Transform [29] and SnooperText [30] algorithms, which were developed for urban scene text detection. These algorithms are compared in Chapter 7.

In 2005, Zheng *et al.* [28] proposed an algorithm for license plate detection based on edge detection. They first use a local pre-processing step to enhance image gradients in order to boost up the license plate region in low contrast scenarios. Then, they detect vertical edges

using the Sobel operator, and filter out edges that are too short, which probably correspond to noise, or too long, which probably correspond to the background or to some part of the vehicle. Finally, the authors use a sliding rectangular window with dimensions similar to those of a license plate, in order to identify regions with high edge density, which probably indicates the presence of a license plate.

In 2010, Epshtein *et al.* [29] proposed an algorithm, called Stroke Width Transform (SWT), to detect characters in images. They use the orientations of the gradients over edge pixels to determine a "local stroke width", and group pixels with similar stroke widths into candidate characters. The candidates are filtered based on the stroke variance and aspect ratio within the region. The remaining regions are then grouped into text regions. The Canny detector [31] is used to obtain image edges.

In 2014, Minetto *et al.* [30] proposed an algorithm, called SnooperText, which locates candidate characters by using morphological image segmentation and character/non-character classification based on shape descriptors. Candidate characters are grouped to form either candidate words or candidate text lines. These candidate regions are then validated by a text/non-text classifier called T-HOG [32], a HOG-based descriptor specifically tuned for single-line text regions. The algorithm is applied at multiple image scales in order to suppress irrelevant detail in character shapes and to avoid the use of overly large kernels in the segmentation.

The color attribute is also used by some authors for detection [33, 34]. This make sense when the license plates follow a distinctive color scheme. However, as stated by Anagnostopoulos *et al.* [26], color information is unstable when the lighting conditions change and are country-specific.

As described in Chapter 4, texture attributes, derived from the transition between the characters and the background — *e.g.* extracted from Gabor filters [35], Wavelet Transform [36], Histogram of Oriented Gradients (HOG) [32] and Scale Invariant Feature Transform (SIFT) [37] — are also important sources of information for license plate detection.

Chapter 3

Motion detection

In this chapter, we describe a fast algorithm for motion detection. The goal here is to efficiently narrow down the locations of the scene where motion is taking place (see Figure 3.1).

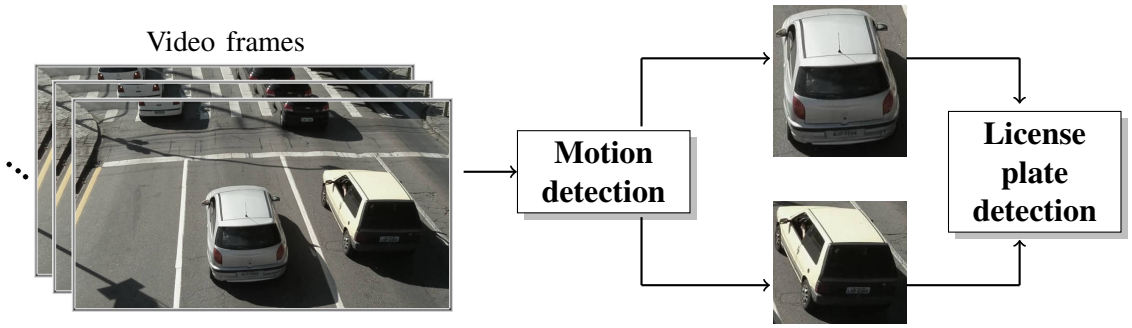


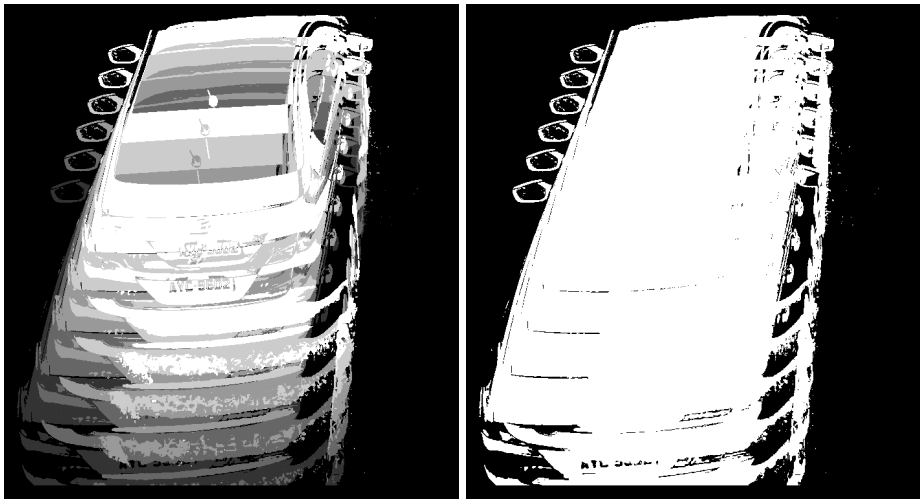
Figure 3.1: Motion detection: the input video frames are reduced to regions of interest that bound the moving vehicles.

The developed motion detector uses a sequence of successive frames to compute a binary image, where the foreground regions, in white color, represent moving vehicles, and the background region, in black color, represents the static scene. A key aspect to save time is the use of a regular sparse grid to compute the binary image. The foreground regions are then used to estimate the vehicles boundaries. These boundaries are essential to reduce the computational effort spent on the license plate detection, which is the most time-consuming part of our system.

The motion detector output is a set of *regions of interest* that ideally should contain the entire vehicle license plate. The following sections provide a detailed description of these steps.

3.1 Motion history and motion energy images

The *Motion History Image* (MHI) is a scalar-valued image where intensity is a function of recency of motion (Bobick and Davis [38]), that is, it represents the *object silhouette* as a grayscale image, where brighter pixels symbolize more recent changes. This straightforward technique is robust to represent motion, being used in applications related to activity recognition [39, 40]. Figure 3.2(a) shows an example of a MHI silhouette from a moving vehicle.



(a) (b)

Figure 3.2: The motion history image (MHI) (a) and the motion energy image (MEI) (b) from a moving vehicle, using a frame interval of $\Delta t = 1$, $\xi = 0.17$, and a sequence of $\tau = 6$ frames.

In order to build the MHI, we first compute the frame difference of two frames (not necessarily consecutive) by

$$\mathbb{D}(x,y,t) = \begin{cases} 1 & \text{if} \quad |\mathbb{I}(x,y,t) - \mathbb{I}(x,y,t-\Delta t)| \geq \xi \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

where the fixed parameters Δt and ξ are chosen to fit constraints of period of time and sensibility. The MHI silhouette $\mathbb{H}$ is then given by

$$\mathbb{H}(x,y,t) = \begin{cases} \tau & \text{if} \quad \mathbb{D}(x,y,t) = 1 \\ \max(0,\ \mathbb{H}(x,y,t-\Delta t) - 1) & \text{otherwise.} \end{cases} \tag{3.2}$$

where τ is a fixed parameter that represents the duration of the expected motion in frame units. Note that each pixel is a function of the temporal motion history at that point.

For simplicity, we build an image $\mathbb{E}(x,y,t)$ where the non-zero pixels of $\mathbb{H}(x,y,t)$ are set as foreground. This image, also known as *motion energy image* (MEI) [38], is given by the equation

$$\mathbb{E}(x,y,t) = \begin{cases} 1 & \text{if} \quad \mathbb{H}(x,y,t) > 0 \\ 0 & \text{otherwise.} \end{cases} \tag{3.3}$$

Although the MEI is simpler than the MHI, it is an useful binary representation of the motion in a frame sequence, as shown in Figure 3.2(b), and in addition, it can be used as a building block for more complex algorithms for motion detection.

3.2 Motion intensity by vertical projection profile

In the domain of our problem, the vehicles enter in the frames gradually in the lower part of the image. The detection becomes easier when the license plate is closer to the image bottom. For that purpose, we define a rectangular *event area* (EA), bounded by x_1 (left), y_1 (top), x_2 (right) and y_2 (bottom), as shown in Figure 3.3.

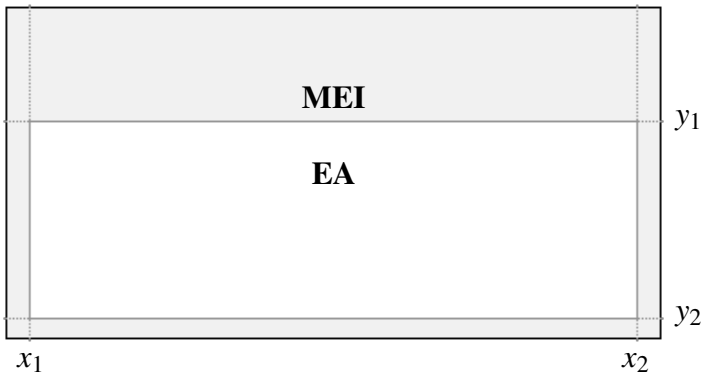


Figure 3.3: Event area for motion detection.

The binary MEI cannot be used by itself to exactly bound each moving vehicle. Holes, shadow or noise in the vehicle's silhouette may cause over or super segmentation. The spatial configuration of the motion intensity is then recovered by a technique known as *vertical projection profile* [41]. Specifically, let χ be the sum of all foreground pixels in a given row of the motion energy image (MEI) $\mathbb{E}$ bounded by the EA, that is

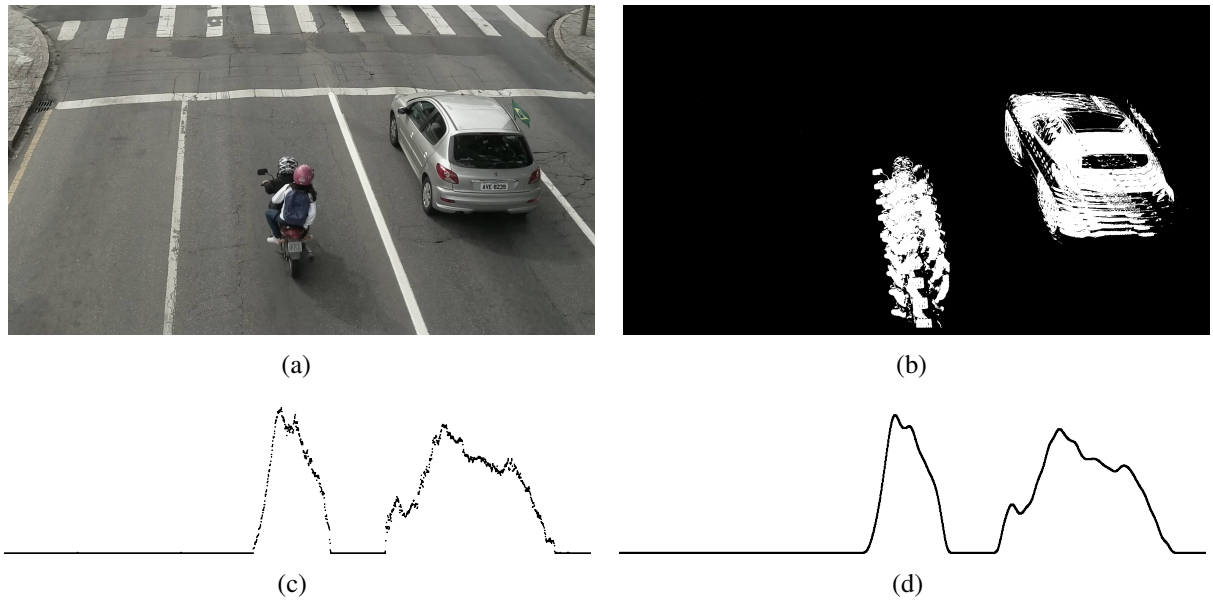
$$\chi(x,t) = \sum_{y=y_1}^{y_2} \mathbb{E}(x,y,t) \tag{3.4}$$

where x ranges from $[x_1, x_2]$. The vertical projection profile is given by

$$\Psi(x,t) = \varphi(\chi(x,t), \eta, \kappa) \tag{3.5}$$

where φ is a conventional rolling mean smoothing function, η is the size of the smoothing window, and κ is the number of iterations.

The smoothed curve Ψ is the projection onto an horizontal straight line of the scene motion. It provides important information about the number of vehicles aligned along the projection direction. This curve changes over time at each new frame. Figure 3.4 shows a sample video frame, its respective MEI and the vertical projection profile.



(a) (b) (c) (d)

Figure 3.4: Vertical projection profile: (a) a sample frame; (b) the binary motion energy image (MEI); (c) the vertical projection profile before smoothing; and (d) the vertical projection profile after smoothing.

3.3 Regions of interest

At this step, we aim to bound each region of interest as a rectangle $\mathscr{R}_i = (l, r, u, d)$, where the vertical projection profile is used to determine the left (l) and right (r) vehicle sides and the binary MEI is used to determine the up (u) and down (d) vehicle limits.

The algorithm to determine the left and right vehicle sides is outlined as Algorithm 1. It receives the vertical projection profile Ψ, and a threshold ρ that defines the minimum percentage of inclination. It returns a pair of lists $\{A, D\}$, such that each list element represents a slope ascending and descending border, respectively. For the sake of simplicity, in the following descriptions, we will omit the t argument of Ψ.

Algorithm 1 Routine to find slopes and borders in Ψ.

```
1:  function FIND-SLOPES (Ψ[1,...,n], ρ)
2:      {R,F} ← FIND-INCLINATION (Ψ, ρ);
3:      S_a ← SLOPE-ASCENDING (R, F);
4:      S_d ← SLOPE-DESCENDING (R, F);
5:      A ← { };   D ← { };                          ▷ set initialization
6:      for each x ∈ {1,...,n−1} do
7:          if (S_a[x] = 0) and (S_a[x+1] = 1) then
8:              A ← A ∪ x;                           ▷ slope ascending
9:          if (S_d[x] = 1) and (S_d[x+1] = 0) then
10:             D ← D ∪ x;                           ▷ slope descending
11:     end for
12:     return {A,D};
```

In step 2 of Algorithm 1, we call the routine FIND-INCLINATION, outlined as Algorithm 2. The purpose of this routine is to determine the *rising and falling phases* of the vertical projection profile Ψ (see Figure 3.5(a)). The threshold $\rho \in \{0.0, \ldots, 1.0\}$ was meant to prevent against false phases. For instance, if we are using $\rho = 0.1$ and the projection profile curve changes from 4.0 to 5.0, that is, a change of 20%, at some point x, then we set $R[x] = 1$, to state for a rising point. Although, if it changes from 400.0 to 410.0, nothing will happen, then we have $R[x] = 0$. Since the vehicle regions in the vertical projection profile are represented by high values, we aim with this step to prevent against accidental vehicles cut off.

Algorithm 2 Routine to find rising and falling phases in Ψ.

```
1:  function FIND-INCLINATION (Ψ[1,...,n], ρ)
2:      for each x ∈ {1,...,n} do
3:          R[x] ← F[x] ← 0;                          ▷ array initialization
4:      end for
5:      for each x ∈ {1,...,n−1} do
6:          δ ← (1 − (Ψ[x]/Ψ[x+1]));
7:          if δ > ρ then
8:              R[x] ← 1;                             ▷ rising phase
9:          if δ < −ρ then
10:             F[x] ← 1;                             ▷ falling phase
11:     end for
12:     return {R,F};
```

In step 3 of Algorithm 1, we call the SLOPE-ASCENDING routine, outlined as Algorithm 3. This function scans the R and F arrays looking for ascending regions. Ascending regions in R, that is transitions from 0 to 1, are flagged (set to 1) while a rising in F is set to 0. It means that an ascending region starts at the first rising phase and ends at the first falling phase. In regions without transition, we keep the last value. The SLOPE-DESCENDING routine, outlined as Algorithm 4, works in the same way, but with the order of the array elements reversed. In Figure 3.5(b,c), we shown the values for these functions over a sample projection profile.

Algorithm 3 Routine to compute the array of ascending slopes.

```
1:  function SLOPE-ASCENDING (R[1,...,n], F[1,...,n])
2:      for each x ∈ {1,...,n} do
3:          S_a[x] ← 0;                               ▷ array initialization
4:      end for
5:      for each x ∈ {2,...,n−1} do
6:          if (R[x] = 0) and (R[x+1] = 1) then
7:              S_a[x] ← 1;
8:          else if (F[x] = 0) and (F[x+1] = 1) then
9:              S_a[x] ← 0;
10:         else
11:             S_a[x] ← S_a[x−1];
12:     end for
13:     return S_a;
```

Algorithm 4 Routine to compute the array of descending slopes.

```
1: function SLOPE-DESCENDING (R[1,...,n], F[1,...,n])
2:     for each x ∈ {1,...,n} do
3:         S_d[x] ← 0;                                   ▷ array initialization
4:     end for
5:     for each x ∈ {n−1,...,2} do
6:         if (F[x] = 0) and (F[x−1] = 1) then
7:             S_d[x] ← 1;
8:         else if (R[x] = 0) and (R[x−1] = 1) then
9:             S_d[x] ← 0;
10:        else
11:            S_d[x] ← S_d[x+1];
12:    end for
13:    return S_d;
```

In steps 6-11 of Algorithm 1, we compare two consecutive samples of S_a and S_d to locate rising and falling borders, respectively, which indicates the left (l) and the right (r) boundary of a slope (see Figure 3.5(d)).

Finally, the procedure COMPUTE-ROI delimit each vehicle boundary. It requires the ascending (A) and descending (D) points and the binary motion energy image $\mathbb{E}$. It outputs the regions of interest as a set $\mathscr{R}$. Note that this procedure scans for each slope i — delimited by the

Algorithm 5 Routine to define the regions of interest.

```
1: function COMPUTE-ROI (A, D, 𝔼)
2:     ℛ ← { };                                          ▷ set initialization
3:     for i ∈ {A,D} do
4:         {l,r,u,d} ← {A[i], D[i], +∞, −∞};             ▷ vehicle boundary initialization
5:         for x ∈ {l,...,r} do
6:             for y ∈ {y1,...,y2} do                    ▷ y1, y2 are used to define the EA
7:                 if 𝔼(x,y) = 1 then
8:                     u ← min(u,y);
9:                     d ← max(u,y);
10:            end for
11:        end for
12:        ℛ ← ℛ ∪ {l,r,u,d};
13:    end for
14:    return ℛ;
```

ascending $A[i]$ and descending $D[i]$ interval — and for the minimum and maximum coordinates of $\mathbb{E}$ flagged as a foreground pixel.

An overview of the motion detection steps is shown in Figure 3.6.

When vehicles are gradually entering in the EA, it may occur that the license plate is not completely visible inside the ROI. To avoid an unnecessary license plate detection in such cases, we only consider regions of interest with the lower border (d) not coincident with the

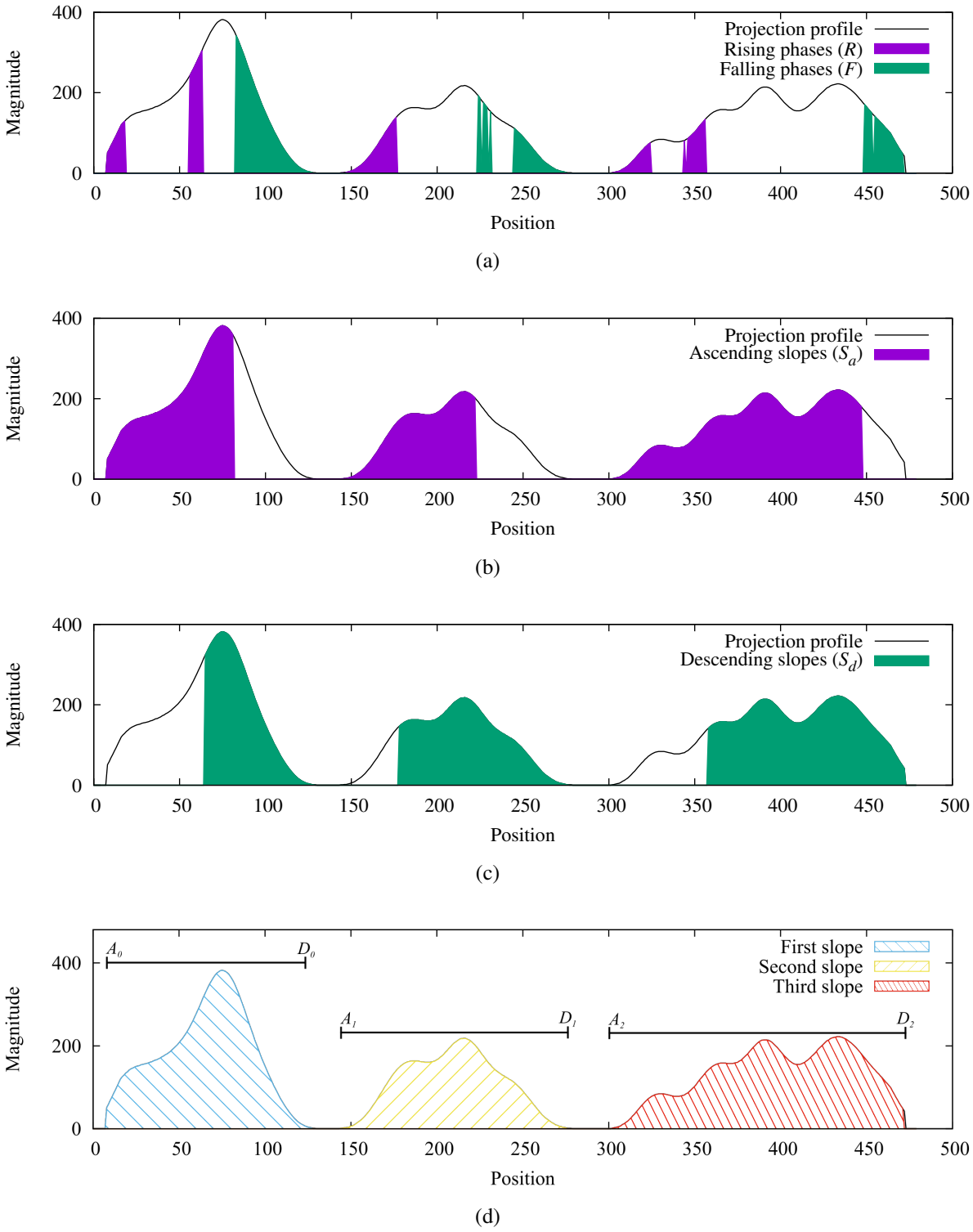


Figure 3.5: Internal steps of Algorithm 1: rising and falling phases according to a given threshold ρ (a), ascending slopes (b), descending slopes (c), and three slope regions delimited by the rising edge of ascending slopes and the falling edge of descending slopes (d).

lower event area limit (y_2), *i.e.* the rule $d < y_2$ must be satisfied to avoid an unnecessary license plate detection.

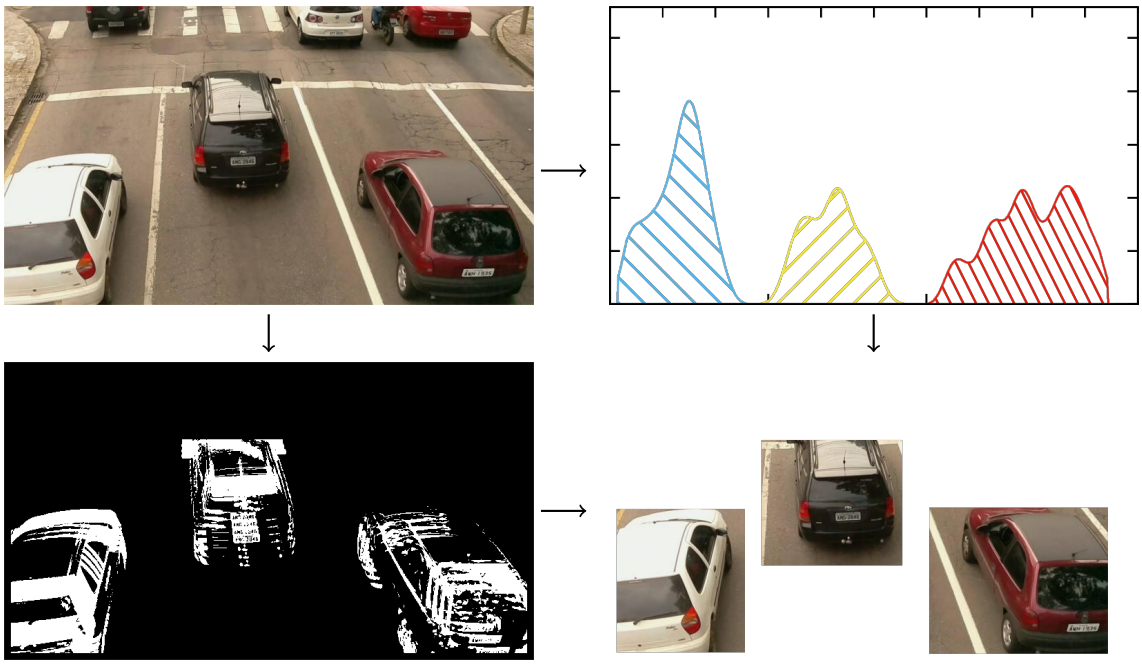


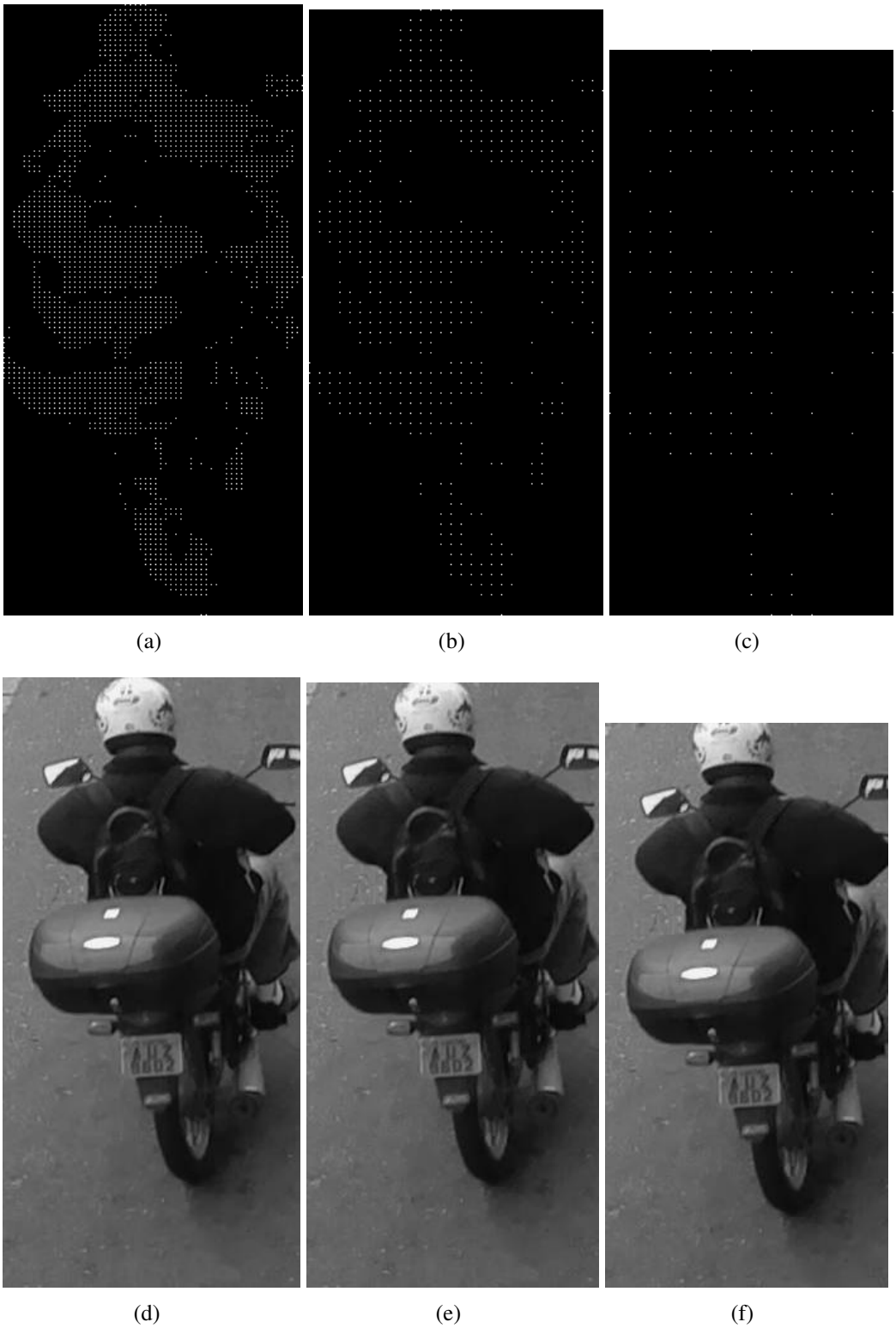
Figure 3.6: Detection of regions of interest.

3.4 Performance optimization by pixel subsampling

The regions of interest need not be accurately estimated, it is sufficient, that the vehicle license plate be entirely within each region. Based on this observation, we use a grid for pixel subsampling in order to speed up the search process.

Specifically, we subsample the MHI pixels, and as a consequence the MEI pixels, at specific coordinates determined by a regular sparsely grid, spanning the whole MHI (see Figure 3.7). Therefore, we significantly reduce the time spent updating the MHI by processing the image pixels according to a subsampling ratio of $1/s$ for each row and column. For example, by using a subsampling ratio of $s = 4$, only 1 pixel will be processed for every 16 pixels. To keep a constant magnitude of the projection profile for different pixel subsampling values, we changed Equation 3.4 by multiplying each position by s.

Note that this strategy may cause an error in region boundaries. However, the vehicle license plates are usually located far enough from the borders. This discussion is detailed in the experiments, in Section 7.3.2.



(a) (b) (c)

(d) (e) (f)

Figure 3.7: Pixel subsampling for motion detection: the motion energy image (MEI) and the regions of interest using a sparse grid with factor $s = 4$ (a,d); using $s = 8$ (b,e); and using $s = 16$ (c,f).

Chapter 4

License Plate Detection

The structure of the license plate detector is outlined in Figure 4.1. The input data for this problem is a cropped image $\mathbb{I} \subset \mathbb{V}(i)$ for a given frame i, which is delimited by a region of interest $\mathscr{R}_i$ as described in the motion detection Chapter 3. The output data is an axis-aligned rectangle, which is an approximate bounding box of the license plate region.

This detector follow the *hypothesis generation and validation paradigm* [30, 42]. Namely, in the hypothesis generation phase we use the *edge extraction*, *edge filtering*, *morphological dilation*, and *region grouping* modules to provide coarse candidate regions based on the edge attribute that make up the license plate. At this phase, we aim to isolate the license plate region and to prevent any true region loss, even at the cost of several false positives.

In the *hypothesis validation* phase, we use a *region classification* module to refine the candidates. For this classification we use the *Text HOG* (T-HOG) descriptor [32], which is based on the observation that a particular texture — the license plate textual information — can often be characterized by the distribution of the directions of the image gradients.

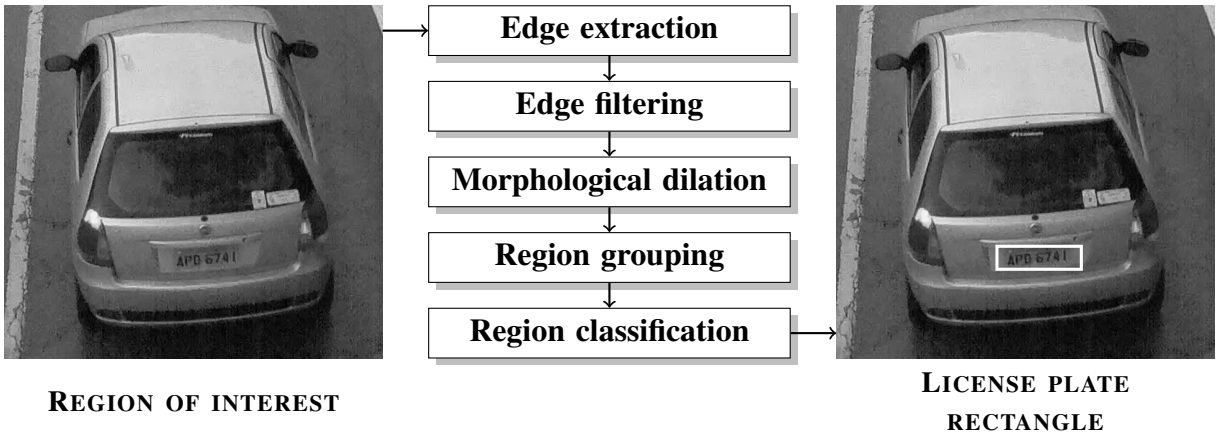


Figure 4.1: License plate detection scheme

The following sections provide a detailed description of each module.

4.1 Edge extraction

Zheng *et al.* [28] observed that background areas around the license plate region often have large horizontal edges and small random noise. Therefore, they proposed to extract only the vertical image edges in an attempt to isolate the license plate. The authors claimed that although some horizontal edges that delimit the license plate are lost, this information is not essential for license plate location. Thus, they use the Sobel operator to extract the vertical image edges. Specifically, the approximate derivative of image $\mathbb{I}$ in the x direction is given by

$$\mathbb{G}_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbb{I} \tag{4.1}$$

where the 3×3 matrix kernel denotes the Sobel operator and $*$ the two-dimensional convolution operation.

The mean μ over the absolute gradient values of $\mathbb{G}_x$ is given by

$$\mu = \frac{\sum_{x=0}^{M} \sum_{y=0}^{N} |\mathbb{G}_x(x,y)|}{MN} \tag{4.2}$$

The edge image $\mathbb{E}$ is then defined by

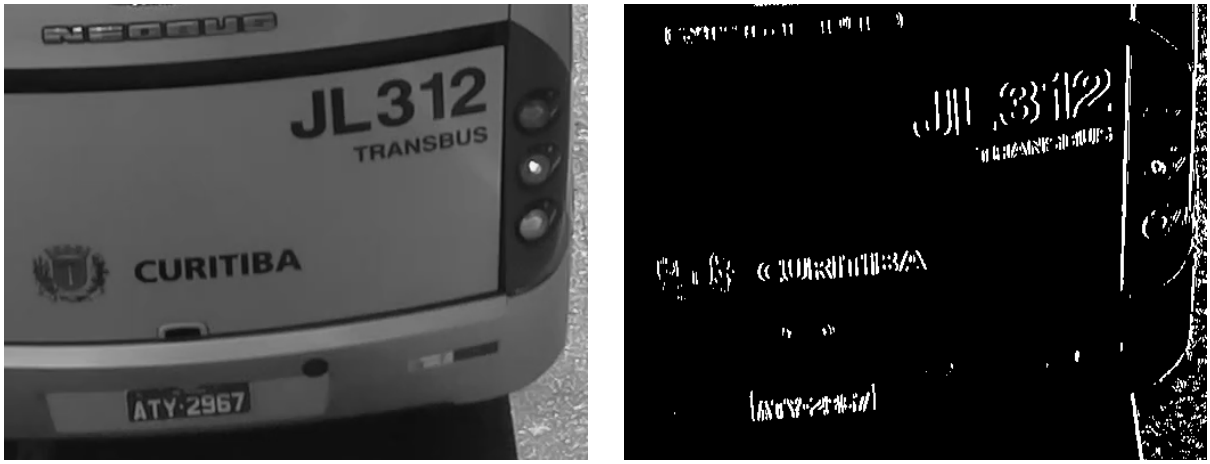
$$\mathbb{E}(x,y) = \begin{cases} 1 & \text{if} \quad |\mathbb{G}_x(x,y)| > \mu\gamma \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

for some value γ. An example of edge extraction is shown in Figure 4.2.

(a) (b)

Figure 4.2: Edge extraction: (a) region of interest as given by the motion detection module; and (b) extracted edges with a threshold $\gamma = 2$.

As shown in Figure 4.3, the advantage of this approach is that it is invariant to *distinctive license plate color schemes* — transitions between light and dark regions or vice versa — since in equation (4.3) it is considered only the gradient magnitude.



(a) (b)

Figure 4.3: Edge extraction, by using a threshold $\gamma = 2$, for a vehicle with white letters on dark background. In our dataset, this is the official color pattern of transportation vehicles like buses and taxis.

4.2 Edge filtering

The edge image $\mathbb{E}$ is then screened to identify plausible characters. For that purpose, we determine the 8-connected edge components by using a standard labeling algorithm. Then, we remove small components, or too tall or wide, *i.e.* those that do not satisfy the following constraints:

$$\begin{aligned} h_{\min} \leq \quad & h \quad \leq h_{\max} \\ w_{\min} \leq \quad & w \quad \leq w_{\max} \end{aligned} \tag{4.4}$$

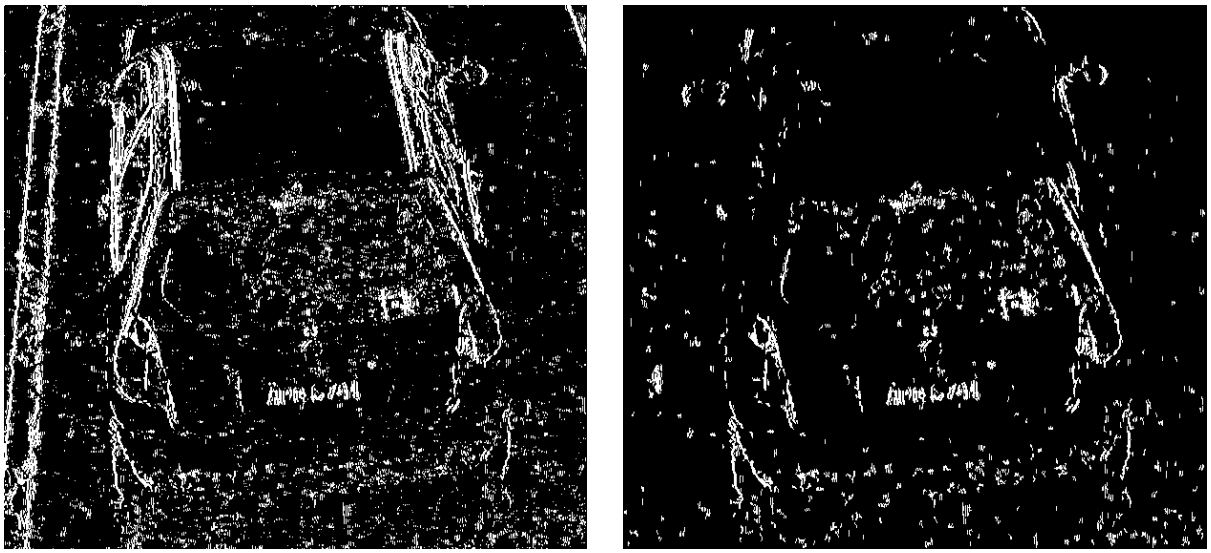
where w and h are the width and height of the bounding box enclosing each labeled component in the edge image $\mathbb{E}$, and $h_{\min}$, $h_{\max}$, $r_{\min}$ and $r_{\max}$ are fixed parameters. An example of edge filtering is shown in Figure 4.4.

4.3 Region grouping

The remaining vertical edges of image $\mathbb{E}$ are merged by a morphological dilation

$$\mathbb{S} = \mathbb{E} \oplus b \tag{4.5}$$

where b is a centered structuring element designed to join vertical neighboring edges in the x-direction into license plate candidates. See Figure 4.5.



(a) (b)

Figure 4.4: Edge filtering: (a) edges detected by the Sobel operator; and (b) edges filtered by the geometric constraint rules of equation (4.4), with $h_{\min} = w_{\min} = 4$ pixels and $h_{\max} = w_{\max} = 120$ pixels.

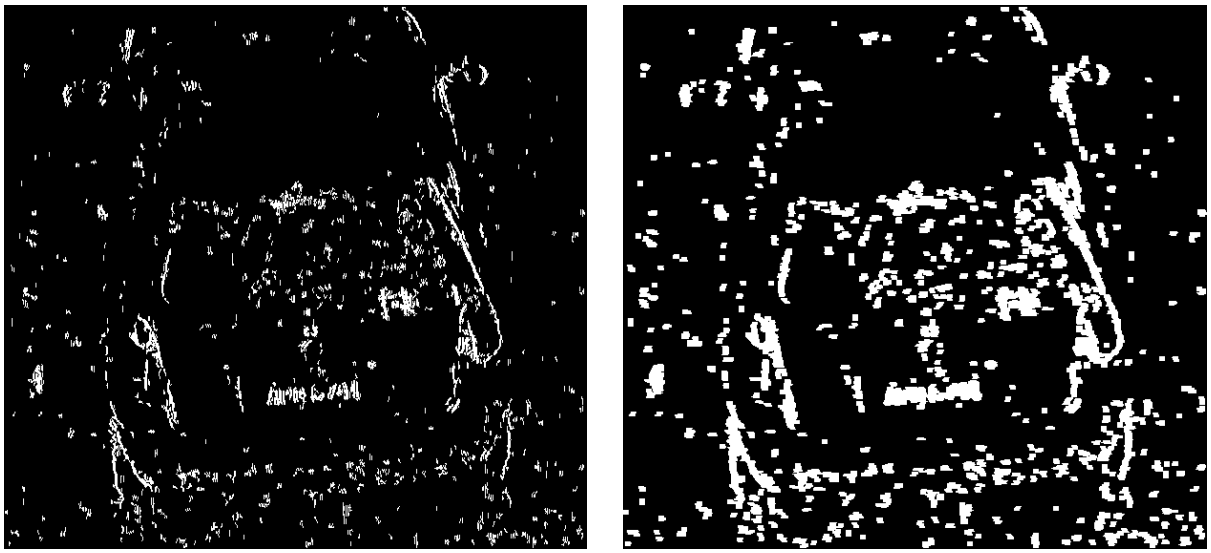


Figure 4.5: Horizontal morphological dilation with a centered 1×7 structuring element.

In order to avoid license plate super-segmentation we then group candidate regions according to the geometric criteria defined by Retornaz and Marcotegui [43]. These criteria take into account the heights h_1, h_2 and widths w_1, w_2 of two bounding boxes b_1 and b_2, as well as the coordinates (x_1, y_1) and (x_2, y_2) of their centers. Specifically, let $h = \min(h_1, h_2)$,

$d_x = |x_1 - x_2| - (w_1 + w_2)/2$ and $d_y = |y_1 - y_2|$. Then b_1 and b_2 are said to be *compatible* — that is, assumed to belong to the same object — if and only if

$$\begin{aligned} |h_1 - h_2| &< t_1\, h \\ d_x &< t_2\, h \\ d_y &< t_3\, h \end{aligned} \tag{4.6}$$

where t_1, t_2 and t_3 are fixed user parameters.

These criteria are applied to each isolated region by using the UNION-FIND data structure [25]. Specifically, the beginning of each region is a disjoint set created by the MAKE-SET routine, outlined in Algorithm 6. The output of this operation is shown in Figure 4.6 (a,b). The algorithm then tries to group all the *compatible* candidate regions by using the UNION-FIND routines, outlined in Algorithms 7 and 8. The output of this operation is shown in Figure 4.6 (c).

Algorithm 6 MAKE-SET routine (Cormen *et al.* [25]).

```
1: function MAKE-SET (b)
2:     father[b] = b;
```

Algorithm 7 UNION routine (Cormen *et al.* [25]).

```
1: function UNION (b1, b2)
2:     f1 ← FIND-SET(b1);          ▷ f1 is the father of b1
3:     f2 ← FIND-SET(b2);          ▷ f2 is the father of b2
4:     if (f1 ≠ f2) then
5:         if COMPATIBLE (b1, b2) (see equation 4.6) then
6:             father[f2] ← f1;
```

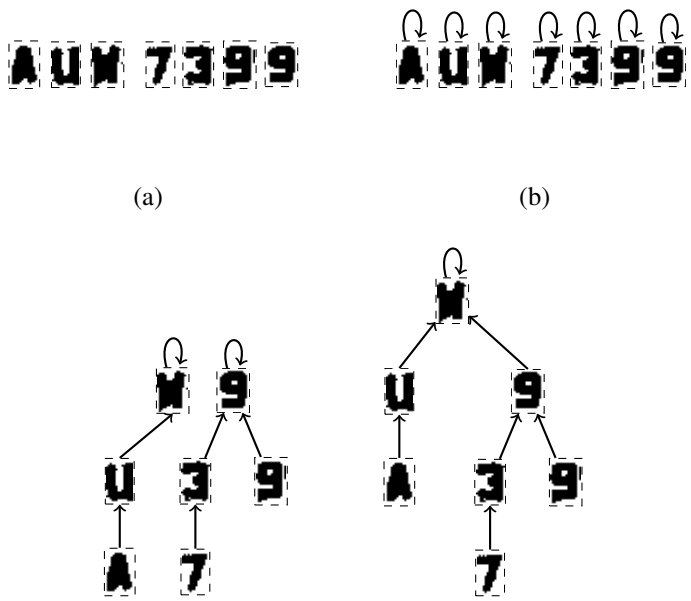
Algorithm 8 FIND-SET routine (Cormen *et al.* [25]).

```
1: function FIND-SET (b)
2:     if (father[b] = b) then
3:         return b;
4:     else
5:         return FIND-SET (father[b]);
```

The regions found by the grouping step (see Figure 4.7(a)) are then filtered using the geometric criteria defined in equation (4.4). However, we set the parameters $h_{\min}$, $h_{\max}$, $r_{\min}$ and $r_{\max}$ to get rid of regions — as opposed to characters as described in Section 4.2 — with dimensions not compatible with license plates, see Figure 4.7(b).

4.4 Region classification

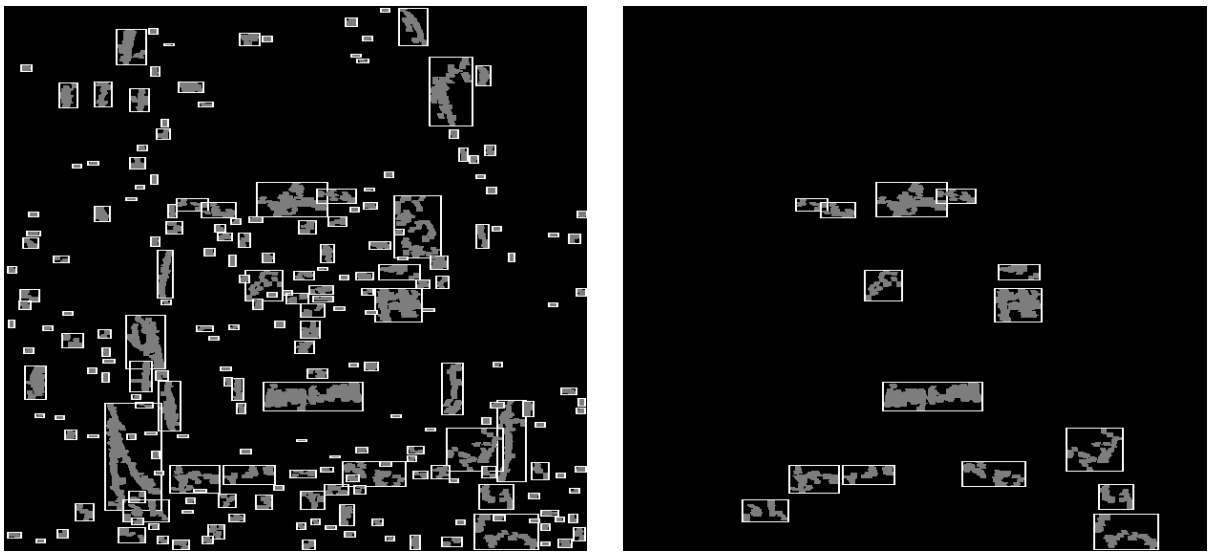
The region classification is essential to discard those regions that do not seem to contain any textual information but have dimensions and texture attributes similar to those of license



(a) (b)

(c)

Figure 4.6: Region grouping: (a) region bounding boxes of a sample license plate image; (b) MAKE-SET routine applied to all regions, the arrows indicate the node parent; (c) result of UNION(w,7). Figure adapted from Cormen *et al.* [25].



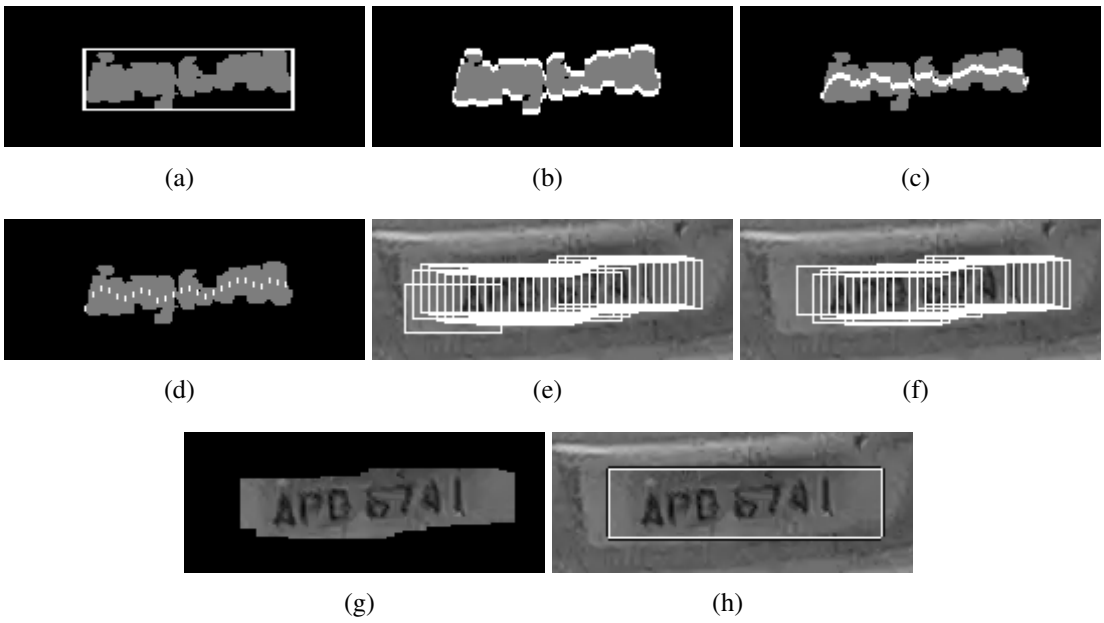
(a) (b)

Figure 4.7: In (a) the region grouping and in (b) the region filtering with the geometric criteria defined in equation (4.4) ($h_{\min} = 10$ pixels, $w_{\min} = 32$ pixels and with $h < w$).

plates. We use for this task the T-HOG text descriptor [32] which is a texture classifier specialized to capture the gradient distribution characteristic of character strokes in occidental-like scripts.

The T-HOG classification is the most time consuming step of the detection. In order to prevent nearby classifications that will consume to much time to produce a similar outcome,

we sampled at a regular interval specific point locations within a candidate region. More specifically, we compute the column center of each candidate image region on image $\mathbb{S}$ (dilated image after region filtering) by using the *region baselines* — the coordinates of the 'top' and 'bottom' region column, see Figure 4.8(b). This approximate center line, see Figure 4.8(c), is then used to guide the classification. Namely, we centered a fixed-size window at some point locations over that center line, see Figure 4.8(d,e), computed the T-HOG descriptor and fed it to an SVM classifier, whose output is thresholded to give a binary text/non-text (positive/negative) region classification. The positive classifications (text regions) are shown in Figure 4.8(f,g).



(a) (b) (c)

(d) (e) (f)

(g) (h)

Figure 4.8: Region baselines for T-HOG/SVM classification: (a) the candidate region; (b) the region baselines in white color; (c) the region center line in white color; (d) sampled points to guide the classification in white color (step of 4 pixels); (e) windows selected for textual classification; (f,g) regions classified as text; (h) the text region bounding box.

The region most likely to have a license plate is that one with at least υ positive text region classification. Note that vehicles with textual advertisements may have several regions that respect this condition. However, the direction of the vehicle flow in our video dataset is from the lower to the upper part of the image. Therefore, if we have more than one candidate respecting the above condition, we keep the one closest to the image bottom. The approximate license plate bounding box is an axis-aligned rectangle that encloses all positives text region classification, see Figure 4.8(h). The T-HOG/SVM classifications for the whole image is shown in Figure 4.9.

4.5 Discussion

In our dataset, the license plates of motorcycles differ from other vehicles, see Figure 4.10(a). They are nearly squared with two closer lines, the top line with three letters and the bottom with four numbers. The strategy to detect this kind of license plate remains the same,

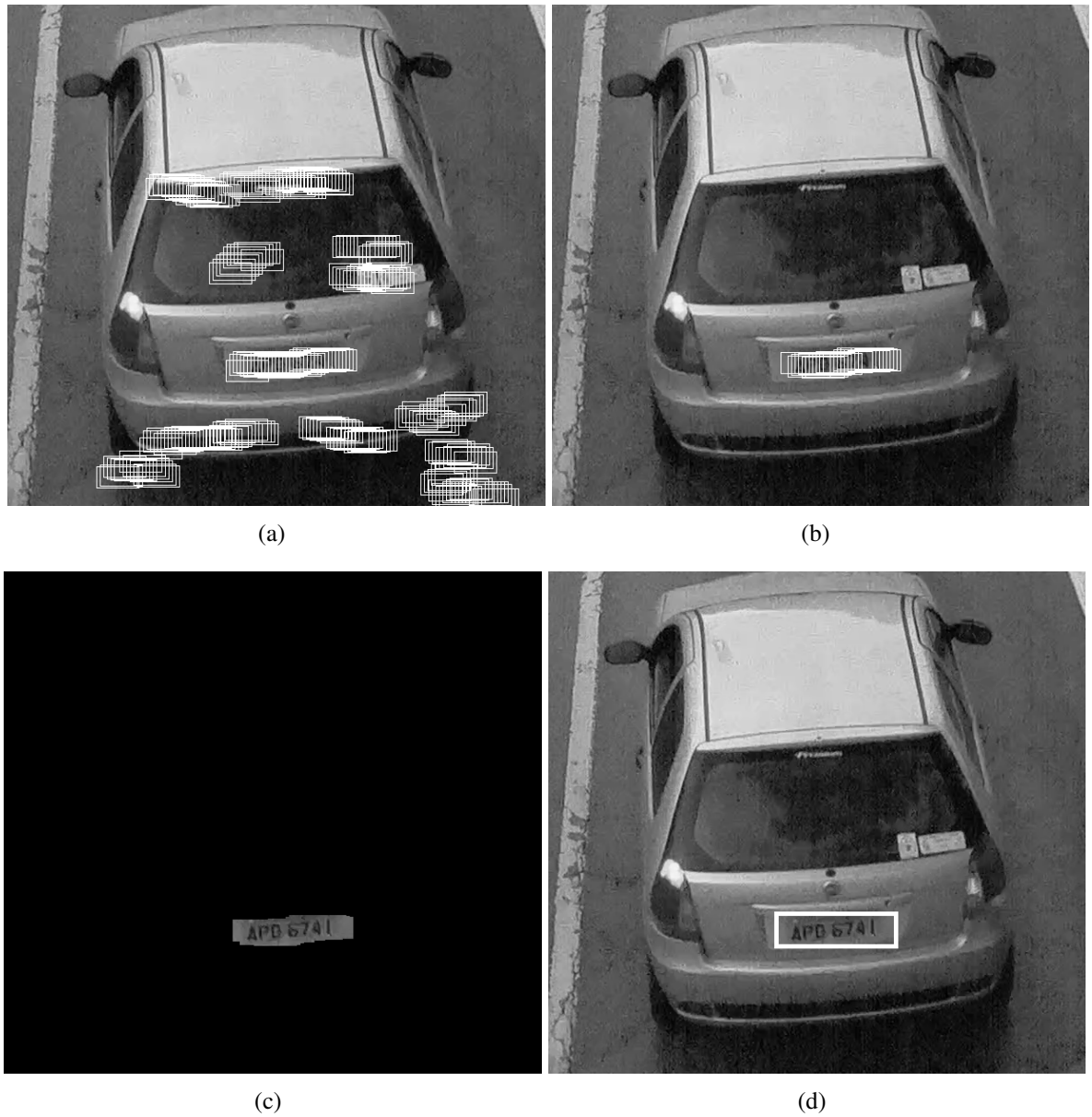


Figure 4.9: T-HOG/SVM classification: (a) the windows selected for textual classification; (b,c) the regions classified as text; (d) the license plate bounding box.

however some thresholds must be set up accordingly to take into account the new dimensions. Moreover, we did not compute the center line of each candidate region, as described in Section 4.4, because the edges of the letters (on top) and numbers (at bottom) are usually merged by the edge extraction module (see Figure 4.10(b)). Instead, we simply use T-HOG/SVM classification within each region returned by the grouping module. As an example, a detection output is shown in Figure 4.10(c).

Finally, the selection of distinctive features for tracking does not require that the detected bounding box rectangle be accurately adjusted to the license plate dimensions, since these features will probably be those that make up the letters of the license plate or the plate corners, as discussed in Chapter 5.

(a) (b) (c)

Figure 4.10: Motorcycle license plate detection: (a) the region of interest as given by the motion detection module; (b) the edge extraction by using a threshold $\gamma = 4$; and (c) the license plate bounding box.

Chapter 5

Feature Selection and Tracking

The selection and tracking of a set of distinctive features - such as corners, contours and texture - from the license plate region is the basis of our vehicle speed estimation scheme. The input data of this module are a license plate region, detected at video frame $\mathbb{V}(i)$ (see Chapter 4), and a sequence of f consecutive frames, $\mathbb{V}(i)$, $\mathbb{V}(i+1)$,...,$\mathbb{V}(i+f-1)$, starting from index i, so that f is the last frame where the features being tracked are visible. The output is a list of motion vectors for each tracked feature in a pair of consecutive video frames.

The structure of our tracking scheme is outlined in Figure 5.1. We select distinctive features, only once for each vehicle, according to the criteria suggested by Shi and Tomasi [2]. The selected features are tracked along the video frames using the Kanade-Lucas-Tomasi (KLT) [3, 4] algorithm. To cope with large feature displacements, *e.g.* from a vehicle moving at a high speed, the movement is initially estimated by matching SIFT (Scale-Invariant Feature Transform) [5] features extracted from the first two frames in the sequence. Incorrect motion vectors obtained by KLT and the SIFT matching are filtered out by an outlier rejection routine.

5.1 Feature selection

Tomasi and Kanade [4] observed that it is hard or even impossible to track a single pixel. The reason is simple: the value of that pixel may change in each frame due to noise or small distortions, or be confused with another adjacent pixel with similar properties. As a consequence, feature selection and tracking are usually done by using a *window of pixels*, here represented by the symbol Ω, with dimension $(2\omega+1)\times(2\omega+1)$. Unfortunately, even a window of pixels may not contain sufficient information to be correctly matched to its corresponding window in the next video frame. This ambiguity is known as the *aperture problem* [44] (see Figure 5.2).

As an attempt to avoid the aperture problem, some authors proposed to use different criteria to select *trackable windows*: regions containing zero crossings of the Laplacian of the image intensity [45]; image corners [46]; or high spacial frequency [47]. However, most of these methods assume that good features can be selected independently of the tracking algorithm.

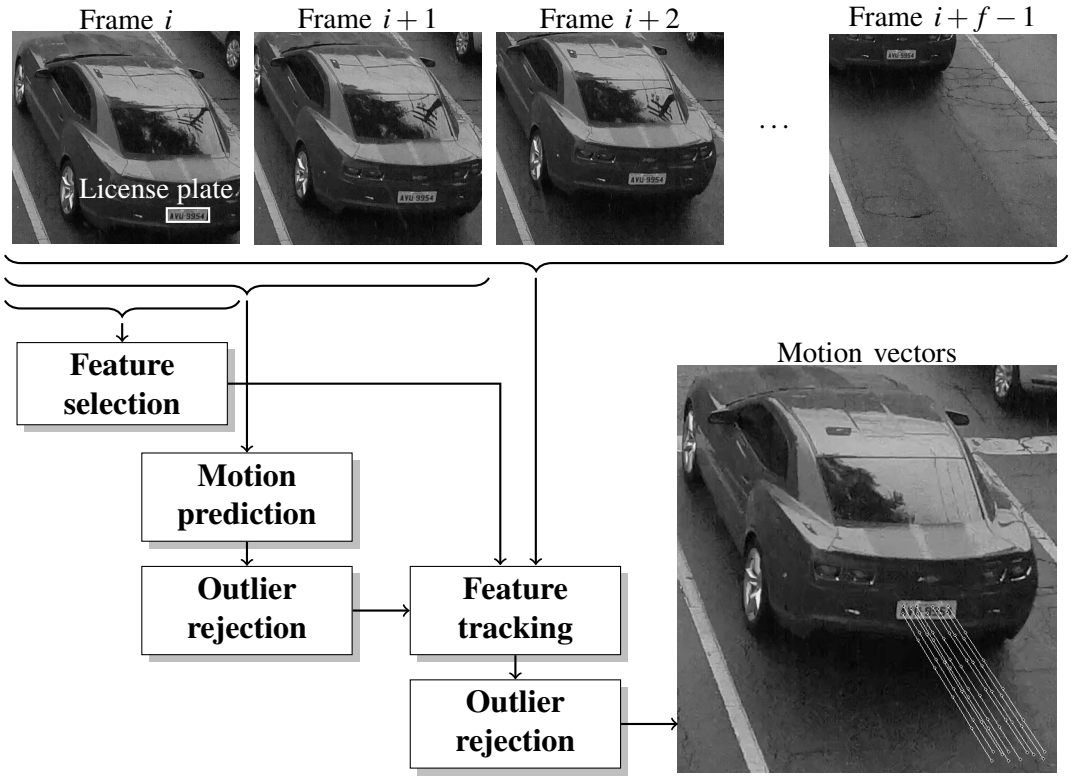


Figure 5.1: Overview of the proposed feature tracking method.

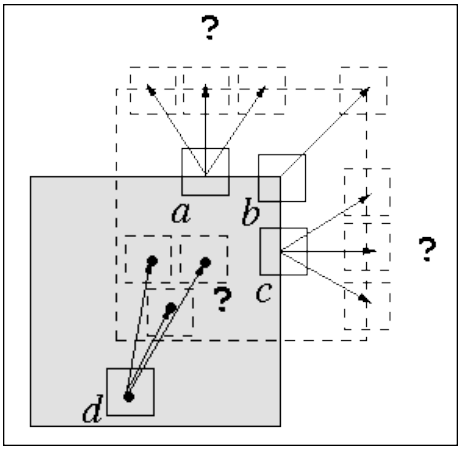


Figure 5.2: Aperture problem: if a moving object is viewed through a minimal window, or if the window does not contain sufficient information, the direction of motion of a local feature may be ambiguous.

Shi and Tomasi [2] proposed a selection criterion that is optimal by construction: "a good window is one that can be tracked well", that is, the notion of a good window is based on the method used for tracking. Namely, let

$$\nabla \mathbb{I} = [\mathbb{I}_x \, \mathbb{I}_y] = \left[\frac{\partial \mathbb{I}}{\partial x} \, \frac{\partial \mathbb{I}}{\partial y} \right] \tag{5.1}$$

be the image derivatives in the x and y directions of a given image $\mathbb{I}$, and let

$$\mathrm{Z} = \sum_{\Omega} \begin{bmatrix} \mathbb{I}_x{}^2 & \mathbb{I}_x\mathbb{I}_y \\ \mathbb{I}_x\mathbb{I}_y & \mathbb{I}_y{}^2 \end{bmatrix} \tag{5.2}$$

be the 2×2 gradient matrix in a given window Ω. According to Shi and Tomasi, if λ_1 and λ_2 are the two eigenvalues of Z, then Ω is selected as a trackable window if

$$\min(\lambda_1, \lambda_2) > \lambda \tag{5.3}$$

for some predefined threshold λ. Precisely, a good feature is a region with high intensity variation in more than one direction, such as corners and highly textured regions. We shown in Figure 5.3, the eigenvalues for five windows selected on different locations of a license plate region.

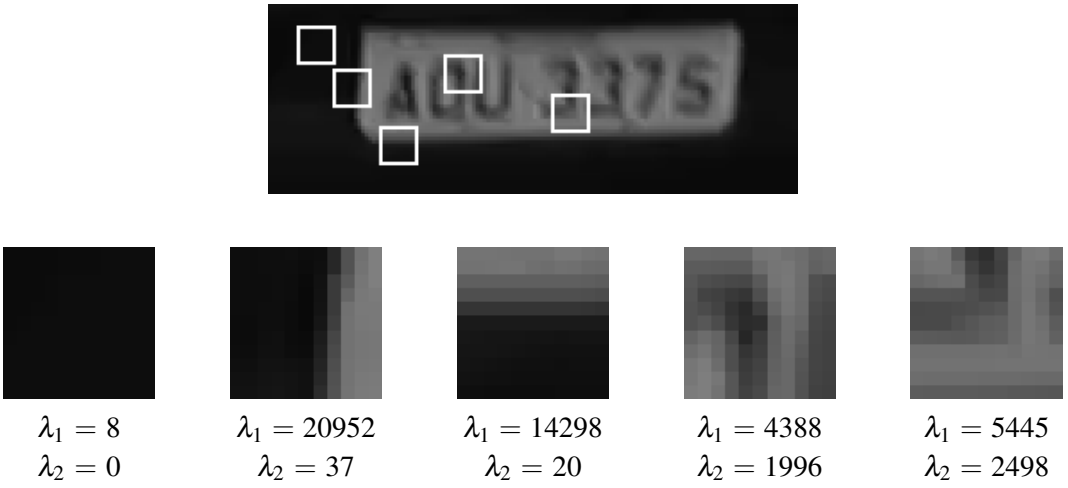


Figure 5.3: Tracking confidence: from left to right, five selected windows with their respective eigenvalues.

5.2 Feature tracking

The computation of 2D image velocities, or *optical flow*, is based on the fact that the intensities of pixels in a small region in two consecutive frames remain constant but the position may change, that is

$$\mathbb{I}(u,t) \approx \mathbb{J}(u + \vec{d}, t + \Delta t) \tag{5.4}$$

where $\vec{d} = (x_d, y_d)$ denotes the displacement of a point $u = (x_u, y_u)$, and Δt is the time interval to the next frame in sequence. In this context, the goal of the Lucas-Kanade algorithm [3] is to minimize the sum of the squared error between these two images, for a small image region Ω, that is

$$E = \sum_{\Omega} \left[\mathbb{I}(u) - \mathbb{J}(u + \vec{d}) \right]^2 \tag{5.5}$$

where E is the residual error to be minimized. Note that the parameter t was removed from $\mathbb{I}$ and $\mathbb{J}$ for simplicity.

A simple method to find the correct displacement vector $\vec{d}$ is to perform an exhaustive search for all possible values of $\vec{d}$ in the domain $\mathfrak{D}\mathbb{J}$, but this is an expensive brute force approach that requires high computational effort. To optimize the expression in Equation (5.5), the Lucas-Kanade algorithm assumes that a current estimate of $\vec{d}$ is known and then iteractively solves for increments $\vec{\Delta}d$, namely

$$E = \sum_{\Omega} \left[\mathbb{I}(u) - \mathbb{J}(u + (\vec{d} + \vec{\Delta}d)) \right]^2 \tag{5.6}$$

updating the parameter $\vec{d}$ at each iteration

$$\vec{d} = \vec{d} + \vec{\Delta}d. \tag{5.7}$$

These steps are interacted, with subpixel accuracy, until the convergence of parameter $\vec{d}$. The derivation of the Lucas-Kanade algorithm is out of the scope of this dissertation, but it uses the image derivatives of $\mathbb{I}$, as shown in Equation 5.2, the difference of images $\mathbb{I}$ and $\mathbb{J}$, and an iterative solution of a Newton-Raphson procedure to find a good match.

An assumption of the LK algorithm is that the motion is small (order of one pixel between the two images). To overcome this limitation, Bouguet [48] described a pyramidal feature tracking, also known as KLT. More precisely, let $\mathbb{I}^0, \mathbb{I}^1, \ldots, \mathbb{I}^\ell$ be a multi-scale *image pyramid*. The base $\mathbb{I}^0$ of the pyramid is the highest resolution image $\mathbb{I}$, and each subsequent image (*level*) $\mathbb{I}^k$ is a copy of the preceding one $\mathbb{I}^{k-1}$, reduced in width and height by a constant factor, usually of $1/2$. Therefore level $\mathbb{I}^k$ has $1/2^{2k}$ as many pixels as level $\mathbb{I}^0$. The maximum level ℓ depends on the size of the original image and on the maximum allowed displacement.

The optical flow is computed at the deepest pyramid level $\mathbb{I}^\ell$. The result of that computation is propagated to the upper level $\mathbb{I}^{\ell-1}$ in the form of a guess for the pixel displacement. Then, the refined optical flow is computed at level $\mathbb{I}^{\ell-1}$, and the result is propagated to level $\mathbb{I}^{\ell-2}$, and so on up to level 0 (the original image).

As depicted in Figure 5.4, in our system the KLT algorithm is used to align a set of n template regions $\mathbb{T}_{\{1,\ldots,n\}} \in \mathbb{I}$, over the license plate region, to an input image $\mathbb{J}$, where the point u_i is the center of the image template $\mathbb{T}_i$.

5.3 Motion prediction

As observed by Bouguet [48], the maximum pixel displacement d that the pyramidal KLT algorithm can handle is given by $d = (2^{\ell+1} - 1)\delta$, where ℓ is the maximum number of pyramid levels and δ is the pixel motion allowed by elementary optical flow computation. That is, for $\ell = 3$ the maximum displacement allowed is about 15 pixels. Increasing the number of levels may allow for larger displacements, but the window Ω at a higher pyramid level will include data from a larger neighborhood of $\mathbb{I}$, making the similarity property of Equation 5.4 harder to hold, and the system more susceptible to incorrect matches.

As the KLT algorithm can use a point's previous motion to estimate its position in the next frame, it can handle large displacements well, as long as the previous results were fairly accurate. However, for the first frame in a sequence (*i.e.* the moment a license plate is detected), there is no known motion. Let $\mathbb{I} = \mathbb{V}(i)$ be the image where the license plate was

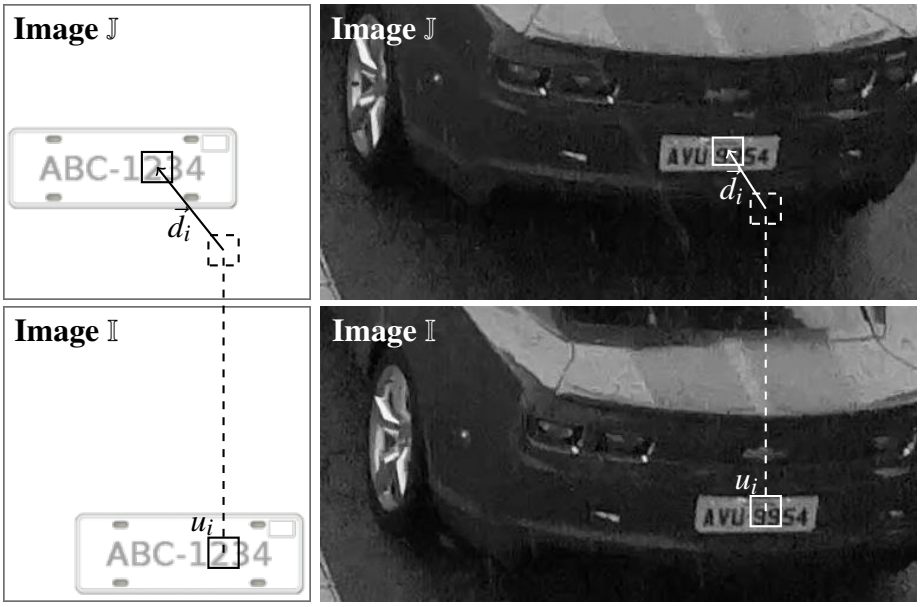


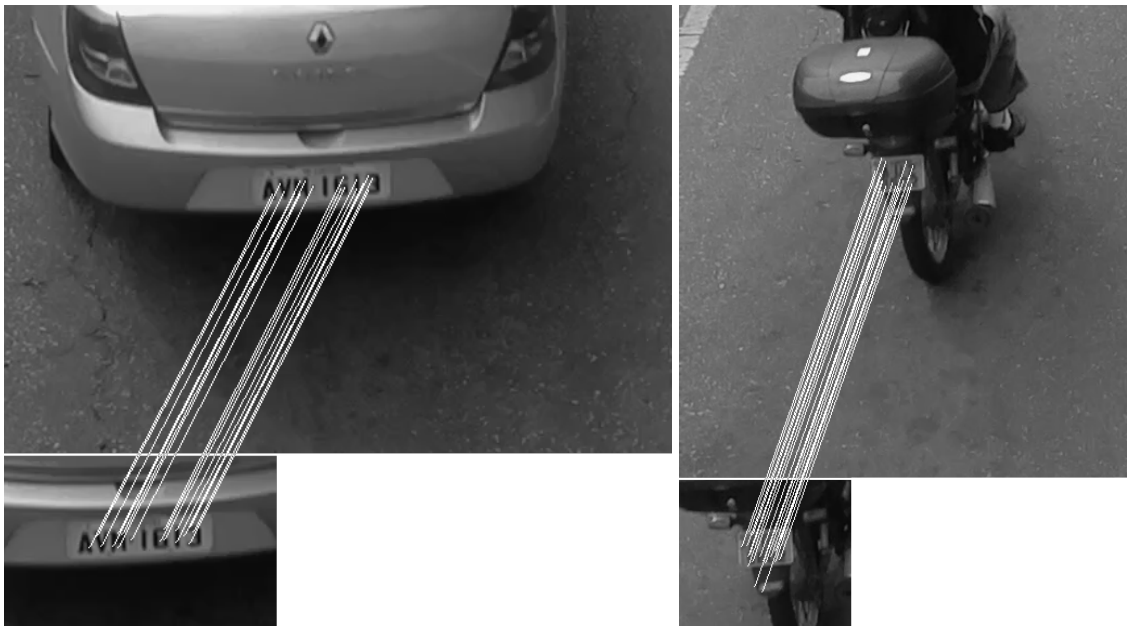
Figure 5.4: Template region alignment: the displacement vector $\vec{d}_i$ should be determined to minimize the difference between the image template, *e.g.* the window Ω centered at $\mathbb{I}(u_i)$, and the window Ω centered at $\mathbb{J}(u_i + \vec{d}_i)$.

detected, and $\mathbb{J} = \mathbb{V}(i+1)$ be the next image from video sequence. If a vehicle is moving fast, the displacement of the tracked points between $\mathbb{I}$ and $\mathbb{J}$ may be too large for us to use the common assumption that $\vec{d} = (0,0)$ in Equation 5.6 when the displacement is unknown.

To overcome the limitation described above, an initial value for $\vec{d}$ is estimated by using a different method for matching features extracted from $\mathbb{I}$ and $\mathbb{J}$. We have used SIFT (Scale-Invariant Feature Transform) features, which were proposed by Lowe [5]. SIFT is a popular method for detecting and describing image keypoints, being invariant to scale and rotation, as well as robust to illumination variations, and to affine and perspective distortions up to a certain extent. SIFT was chosen for our system mostly because it is a popular and tested method, with implementations readily available. Other similar descriptors might have been used for the same purpose, such as SURF [49] or GLOH [50]. Internal details of SIFT are outside the scope of this thesis, and can be found in [5].

We define two windows $\Omega_1 \in \mathbb{I}$ and $\Omega_2 \in \mathbb{J}$, which are obtained by expanding the license plate boundaries respectively by γ_1 and γ_2 pixels. Here, γ_1 is a constant used only to allow SIFT features to be properly extracted from the license plate region, since SIFT discards regions too close to the image borders. The value of γ_2 was selected so that the license plate still appears inside Ω_2, even for fast moving vehicles. SIFT features extracted from Ω_1 are matched to features extracted from Ω_2, as shown in Figure 5.5. We have used the "nearest neighbor distance ratio" matching strategy described by Mikolajczyk [50], in which a match occurs if the ration between the distance from a feature to its nearest neighbor and the distance to its second nearest neighbor is below a given threshold (0.6, in our experiments).

The obtained feature matches can be used to compute the displacement of each SIFT feature between frames $\mathbb{I}$ and $\mathbb{J}$. The average feature displacement is used as the initial value for $\vec{d}$ in Equation 5.6. Note that this process occurs only once for each detected license plate —



(a) (b)

Figure 5.5: Sample images of SIFT keypoints matching. The lower two images correspond to samples of window Ω_1 and the upper two images correspond to samples of window Ω_2.

after the motion is roughly predicted, the system relies on the KLT algorithm, which allows for a faster and more accurate computation for the optical flow.

5.4 Outlier rejection

In statistics, an *outlier* is a data point which appears to be inconsistent with the rest of the data [51]. For our system, an outlier is a motion vector that differs significantly from the other computed motion vectors, due to a mismatch. This kind of mismatch, in our system, may compromise the initial guess, resulting in a tracking failure, or may compromise the speed estimation. To prevent against such failures, we used an *outlier rejection* routine, which is applied for the motion vectors obtained by the KLT algorithm, as well as those obtained by matching SIFT features.

The motion vectors $\vec{d_i} = (x_i, y_i)$ are initially estimated by the motion prediction module (see Section 5.3) and refined with sub-pixel accuracy by the KLT feature tracking module (see Section 5.2), where $i = \{1, 2, \ldots, n\}$ is the index of a set of individual features tracked or keypoints matched. In order to discard outliers, we compute the mean (Equation 5.8) and the standard deviation (Equation 5.9) of the displacements in the x and y axes. Namely,

$$\mu_x = \frac{\sum_{i=1}^{n} d_i(x)}{n} \qquad \mu_y = \frac{\sum_{i=1}^{n} d_i(y)}{n} \tag{5.8}$$

$$\sigma_x = \frac{\sum_{i=1}^{n} (d_i(x) - \mu_x)^2}{n-1} \qquad \sigma_y = \frac{\sum_{i=1}^{n} (d_i(y) - \mu_y)^2}{n-1} \tag{5.9}$$

Then, we discard the motion vectors outside the three-sigma deviation in any direction. This procedure is repeated until the standard deviation in both x and y axes become smaller than 0.5 pixel.

The outlier rejection step applied to KLT tracked features and to SIFT keypoint matches is exemplified in Figures 5.6 and 5.7, respectively. In our experiments, the KLT tracking was extremely consistent, rarely requiring more than one iteration, even in situations with rain or for noisy images.

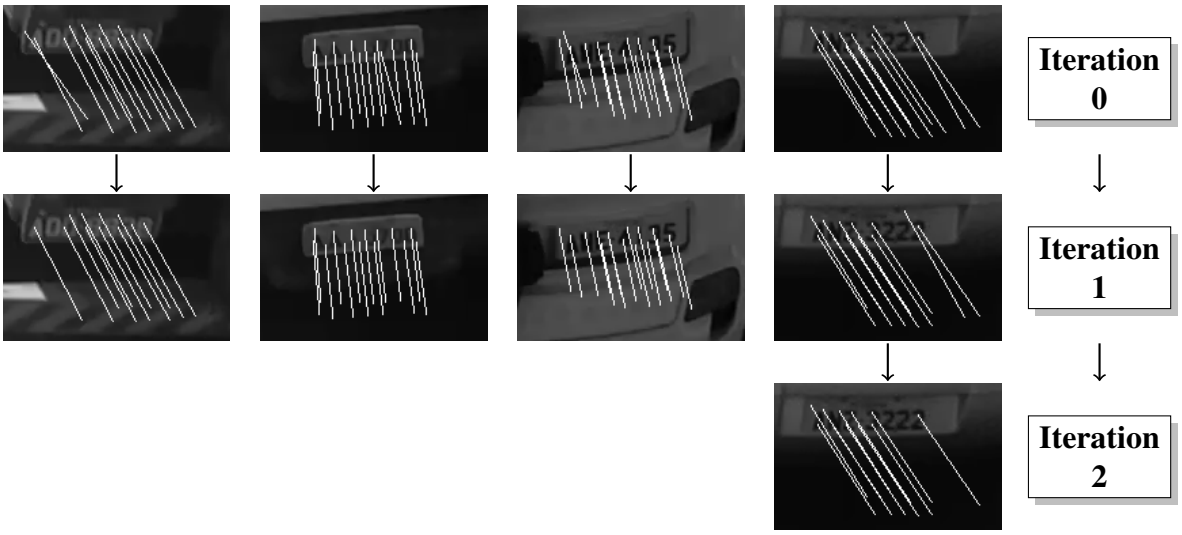


Figure 5.6: Outlier rejection for KLT tracked features: image samples with all motion vectors (iteration zero), and first and second iterations of the outlier rejection step.

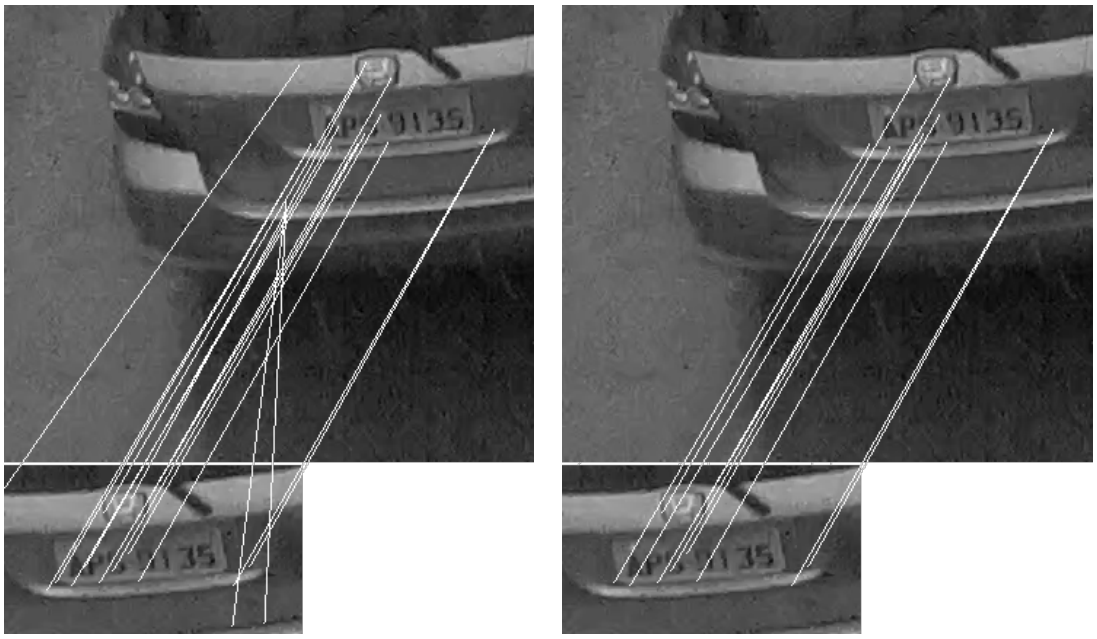


Figure 5.7: Outlier rejection for SIFT keypoints matches: the image on the left show all matching keypoints and the image on the right show the remaining keypoints after the outlier removal step. For this sample, 3 iterations were required.

Chapter 6

Speed Measurement

Our system estimates vehicle speed based on the motion vectors obtained by the feature selection and tracking method (see Chapter 5). Each motion vector $\vec{d}_i$ can be associated with a measurement of a vehicle's instantaneous speed in the image plane at a particular time, given in pixels per frame. To estimate the vehicle speed, these measurements are averaged and converted to meters per second (m/s) or kilometers per hour (km/h), in the ground plane. To convert the displacements from pixels to meters, we employ a perspective rectification method based on known real-world distances, which produces a displacement $\vec{v}_i$ in the ground plane from each motion vector $\vec{d}_i$ (see Figure 6.1). To convert the time unit from frames to seconds, we use the camera frame rate, which is supposed to be constant. These steps are detailed in the following sections.

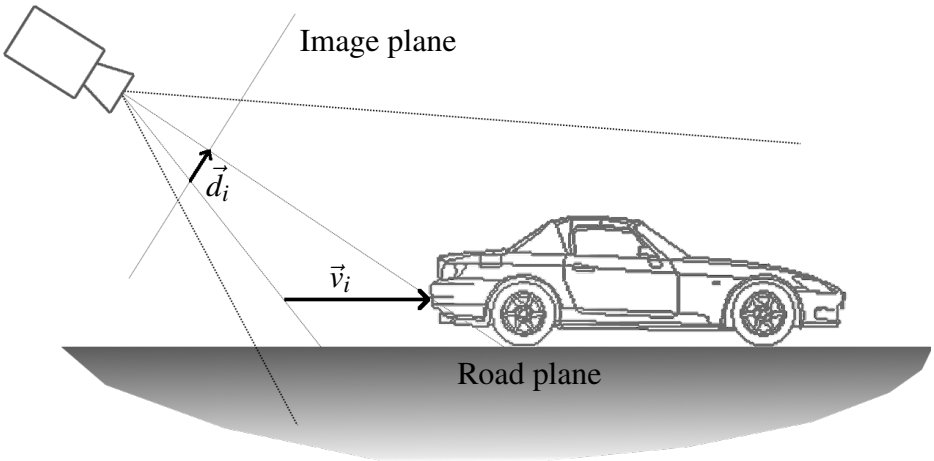


Figure 6.1: Vehicle speed estimation scheme.

6.1 Camera model

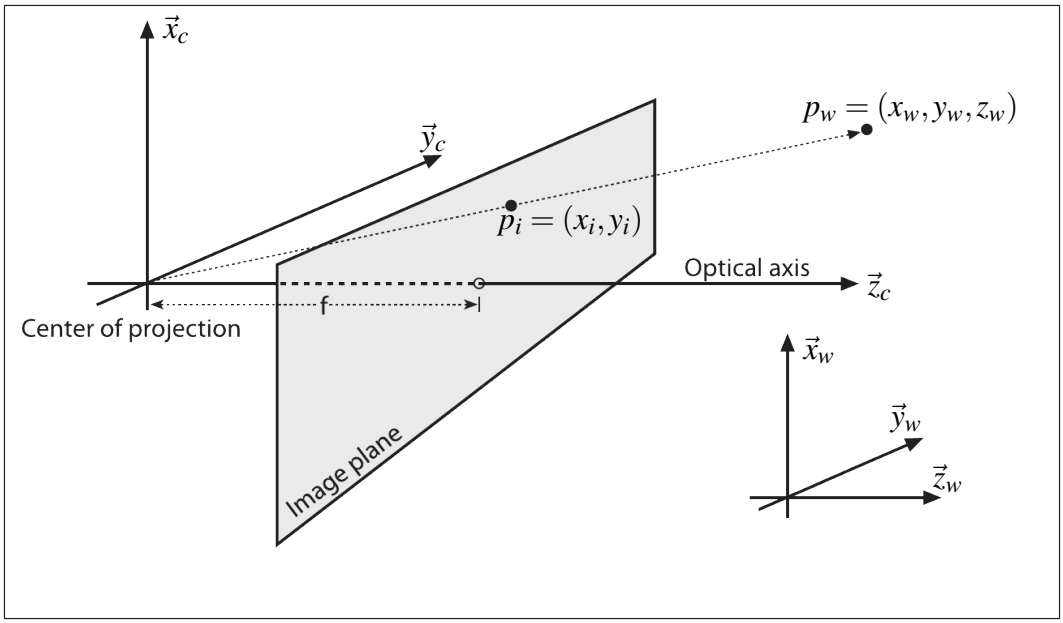


Figure 6.2: The pinhole camera model (adapted from Bradski and Kaehler [52]).

A *camera model* describes the mapping between points in a 3D space and pixels in the 2D image plane. The *pinhole camera*, depicted in Figure 6.2 is the simplest model, and provides a good approximation to the behavior of most real cameras [53]. In the pinhole camera model, light reflected or emitted from a point $p_w = (x_w, y_w, z_w)^{\mathrm{T}}$ in the world enters the camera through an infinitely small aperture (the *center of projection*), being projected into an image plane inside the camera. Let $p_c = (x_c, y_c, z_c)^{\mathrm{T}}$ be a point in the camera coordinate system — a coordinate system where the x and y axes are aligned to the image plane, and the origin is located at the center of projection. According to the pinhole projection model, point p_c projects in the image plane at image coordinates $p_i = (x_i, y_i)^{\mathrm{T}}$, given by

$$x_i = \frac{f \cdot x_c}{z_c} + c_x \qquad y_i = \frac{f \cdot y_c}{z_c} + c_y \tag{6.1}$$

where f is the camera's *focal length*, or the distance between the image plane and the center of projection; and (c_x, c_y) are the coordinates of the principal point (in pixels, *e.g.* $c_x = N/2$ and $c_y = M/2$ indicate the principal point is at the center of the image). For the sake of simplicity, we consider a single value f, in pixels, for the focal length — in practice, the value is converted from physical units to pixels, and if the pixels are rectangular rather than square, different values are used for the x and y axes. Equation 6.1 can be expressed using homogeneous coordinates, with the transformation being given by a 3×3 camera *intrinsic matrix*, that is

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} z_c \cdot x_i \\ z_c \cdot y_i \\ z_c \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{6.2}$$

In the equations above, point $p_c = (x_c, y_c, z_c)^{\mathrm{T}}$ is given in the camera coordinate system. To relate points in the world coordinate system to the camera coordinate system, a 4×4 *extrinsic matrix* is used:

$$\begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{yz} & t_y \\ r_{zx} & r_{zy} & r_{zz} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{6.3}$$

The upper right 3×3 sub-matrix of the extrinsic matrix is the orthonormal (rotation) matrix, that determines the orientation of the camera axes relative to the scene axes. The column vector $T = (t_x, t_y, t_z)^{\mathrm{T}}$ contains the camera coordinates of the world system's origin.

Equations 6.2 and 6.3 can be combined into a single linear equation by

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} z_c \cdot x_i \\ z_c \cdot y_i \\ z_c \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} & t_x \\ r_{yx} & r_{yy} & r_{xz} & t_y \\ r_{zx} & r_{zy} & r_{xz} & t_z \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{6.4}$$

A particular case of the above transformation occurs when all the 3D points lie on the same plane. In that case, we can set $z_w = 0$, simplifying Equation 6.4 to

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} z_c \cdot x_i \\ z_c \cdot y_i \\ z_c \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{xx} & r_{xy} & t_x \\ r_{yx} & r_{yy} & t_y \\ r_{zx} & r_{zy} & t_z \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ 1 \end{bmatrix} \tag{6.5}$$

The intrinsic and extrinsic matrices can be joined in a single 3×3 matrix H, that is

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{xx} & r_{xy} & t_x \\ r_{yx} & r_{yy} & t_y \\ r_{zx} & r_{zy} & t_z \end{bmatrix} \tag{6.6}$$

Equation 6.6 shows that, when all the 3D points lie on the same plane, the camera model is completely specified once the 3×3 matrix H is specified [54]. That means points in the same world plane can be mapped to the image plane by a *homography* — a plane-to-plane projective transformation. Our system benefits from this property by assuming that each street lane lies on a plane. That allows us to obtain (plane) world coordinates from image coordinates based on a single view of the scene, a process sometimes referred to as *inverse perspective mapping* [55]. Section 6.2 details the perspective rectification procedure.

A final note about the considered camera model is that it is an approximation that will produce incorrect results if there is a significant amount of radial distortion or sensor inclination. Initial tests performed using a camera calibration algorithm [52] showed that the camera used in our experiments had negligible sensor inclination and radial distortion, especially in the measurement area. For that reason, we opted for a simpler scheme, which does not attempt to explicitly recover camera and distortion parameters. That makes the camera calibration procedure simpler, and less prone to errors introduced by noisy measurements.

6.2 Perspective rectification

The purpose of perspective rectification is obtaining the coordinates of a point $\hat{p}_w = (x_w, y_w, 1)^{\mathrm{T}}$ in the world plane from its position p_i in the image. The homography is an invertible transformation [56], and the inverse transformation is also a homography, namely

$$p_i = H\hat{p}_w \tag{6.7}$$

$$\hat{p}_w = H^{-1}p_i \tag{6.8}$$

The homography matrix H (or its inverse, H^{-1}) may be obtained by associating points in the image to known coordinates in the world plane. Suppose image point $p_i = (x_i, y_i)^{\mathrm{T}}$ is associated with a known position $\hat{p}_w = (x_w, y_w, 1)^{\mathrm{T}}$ in the world plane. From Equations (6.5), (6.6) and (6.7) this correspondence provides

$$x_i = \frac{h_{11}\cdot x_w + h_{12}\cdot y_w + h_{13}}{h_{31}\cdot x_w + h_{32}\cdot y_w + h_{33}} \tag{6.9}$$

$$y_i = \frac{h_{21}\cdot x_w + h_{22}\cdot y_w + h_{23}}{h_{31}\cdot x_w + h_{32}\cdot y_w + h_{33}} \tag{6.10}$$

These equations can be rearranged, resulting in

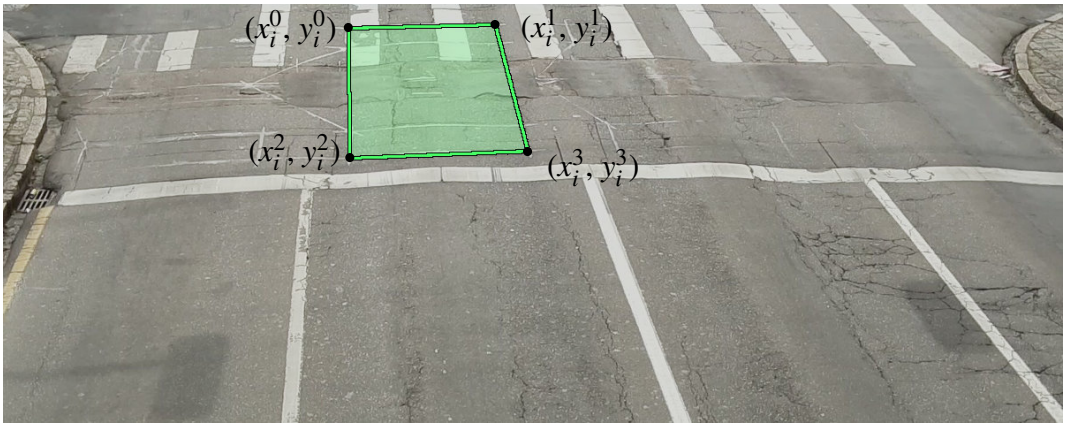
$$h_{11}\cdot x_w + h_{12}\cdot y_w + h_{13} = h_{31}\cdot x_w\cdot x_i + h_{32}\cdot y_w\cdot x_i + h_{33}\cdot x_i \tag{6.11}$$

$$h_{21}\cdot x_w + h_{22}\cdot y_w + h_{23} = h_{31}\cdot x_w\cdot y_i + h_{32}\cdot y_w\cdot y_i + h_{33}\cdot y_i \tag{6.12}$$

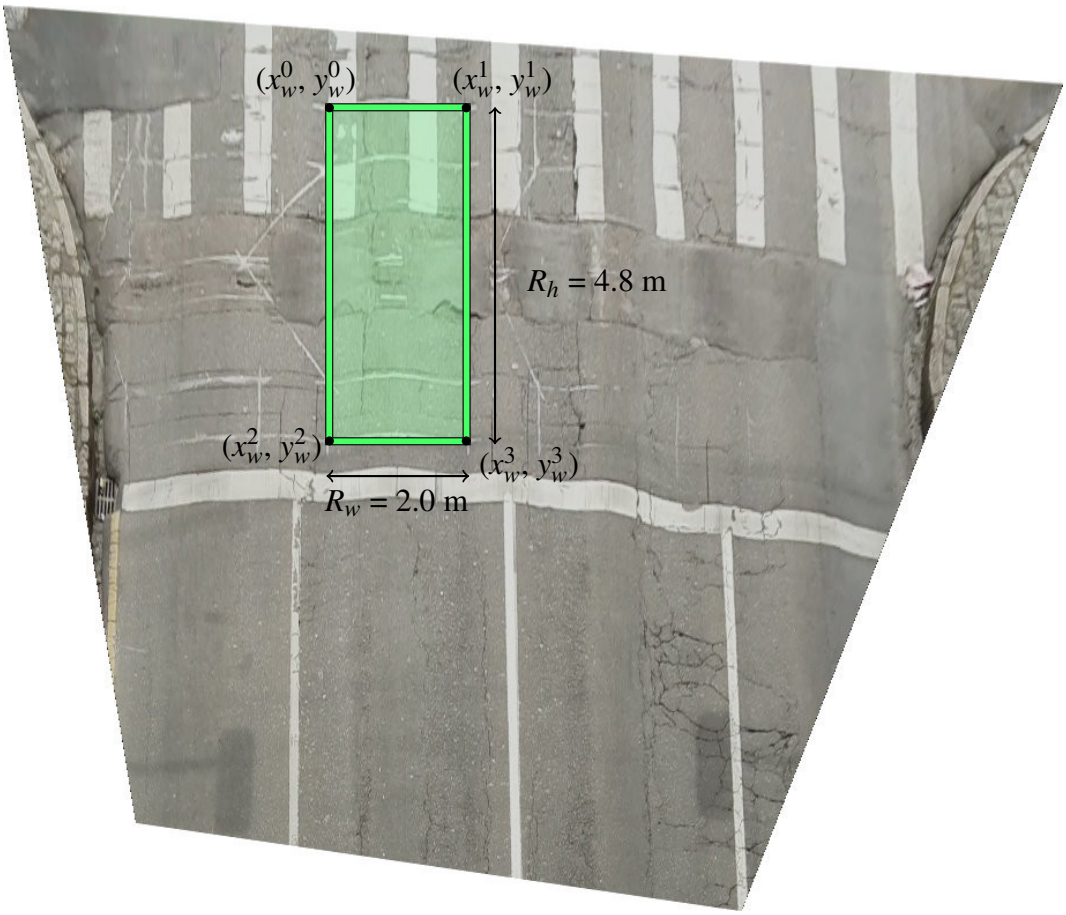
Suppose we have 4 image points $p_i^0, p_i^1, p_i^2, p_i^3$, as well as their respective know coordinates in the world plane $\hat{p}_w^0, \hat{p}_w^1, \hat{p}_w^2, \hat{p}_w^3$. Assuming no three points are collinear and they all lie on the same world plane, we have 9 unknowns and 8 equations. We usually set h_{33} to 1, and arrange all the 8 equations as a linear system:

$$\begin{bmatrix}
x_w^0 & y_w^0 & 1 & 0 & 0 & 0 & -x_w^0\cdot x_i^0 & -y_w^0\cdot x_i^0 \\
x_w^1 & y_w^1 & 1 & 0 & 0 & 0 & -x_w^1\cdot x_i^1 & -y_w^1\cdot x_i^1 \\
x_w^2 & y_w^2 & 1 & 0 & 0 & 0 & -x_w^2\cdot x_i^2 & -y_w^2\cdot x_i^2 \\
x_w^3 & y_w^3 & 1 & 0 & 0 & 0 & -x_w^3\cdot x_i^3 & -y_w^3\cdot x_i^3 \\
0 & 0 & 0 & x_w^0 & y_w^0 & 1 & -x_w^0\cdot y_i^0 & -y_w^0\cdot y_i^0 \\
0 & 0 & 0 & x_w^1 & y_w^1 & 1 & -x_w^1\cdot y_i^1 & -y_w^1\cdot y_i^1 \\
0 & 0 & 0 & x_w^2 & y_w^2 & 1 & -x_w^2\cdot y_i^2 & -y_w^2\cdot y_i^2 \\
0 & 0 & 0 & x_w^3 & y_w^3 & 1 & -x_w^3\cdot y_i^3 & -y_w^3\cdot y_i^3
\end{bmatrix}
\cdot
\begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix}
=
\begin{bmatrix} x_i^0 \\ x_i^1 \\ x_i^2 \\ x_i^3 \\ y_i^0 \\ y_i^1 \\ y_i^2 \\ y_i^3 \end{bmatrix} \tag{6.13}$$

By solving Equation (6.13), we may obtain the homography matrix H. Note that the same approach may be used to obtain H^{-1}, which is also a homography matrix.



(a)



(b)

Figure 6.3: Perspective rectification. The input frame is rectified for illustrative purposes only, the actual system manipulates a sparse set of points.

To solve Equation 6.13, we need four image points, as well as their coordinates in the world plane. A common choice is selecting four points that form a quadrilateral in the image and a rectangle with size $R_w \times R_h$ meters in the world plane, as shown in Figure 6.3. Based on

the size of the rectangle, we may establish a direct relation between meters and pixels. If we set $\hat{p}_w^0$ as the world plane origin, we have

$$\hat{p}_w^0 = (0,0) \tag{6.14}$$

$$\hat{p}_w^1 = (R_w,0) \tag{6.15}$$

$$\hat{p}_w^2 = (0,R_h) \tag{6.16}$$

$$\hat{p}_w^3 = (R_w,R_h) \tag{6.17}$$

The procedure to obtain the homography matrix H then consists in selecting $\hat{p}_w^0$, $\hat{p}_w^1$, $\hat{p}_w^2$ and $\hat{p}_w^3$, finding correspondences for these points in an image captured by the camera, and solving Equation 6.13. The rectangle is usually obtained by placing a known planar object on the desired plane (one of the road lanes, for our problem). For our system, we have used as references the markings left on the asphalt by the inductive loop detectors. A different homography matrix was obtained for each road lane, *i.e.* instead of a single ground plane, we assume each lane lies on a different plane. Note that there are more sophisticated methods, such as RANSAC [52], which obtain the homography based on more than 4 correspondences, but these methods rely on combining results produced by the same reasoning discussed above.

6.3 Vehicle speed estimation

The feature selection and tracking method (see Chapter 5) produces, for each vehicle and each pair of frames, a set of motion vectors $\vec{d_i} = u_i(t) - u_i(t-\Delta t)$, where $u_i(t)$ is the feature position in the current video frame, $u_i(t-\Delta t)$ is the feature position in the previous video frame, Δt is the frame interval, and $i = \{1,2,\ldots,n\}$ is a sequence of tracked features. In order to estimate the vehicle speed, we need to compute the feature displacements in the real world, denoted by $\vec{v}_i$ (see Figure 6.1). Assuming that all the motion vectors for a vehicle lie on the same plane, we may use the perspective rectification approach to obtain $\vec{v}_i$ from Equation 6.8, namely

$$\vec{v}_i = H^{-1}u_i(t) - H^{-1}u_i(t-\Delta t) \tag{6.18}$$

The displacement in meters between frames $t-\Delta t$ and t for a motion vector $\vec{u}_i$ can be obtained by the Euclidean norm of $\vec{v}_i$, *i.e.* $\|\vec{v}_i\|$. Each displacement vector can be associated with a measurement of the vehicle's instantaneous speed, given by

$$s_i = \frac{\|\vec{v}_i\|}{\Delta t} \tag{6.19}$$

where Δt is the time, in seconds, between two frames. This time is supposed to be constant, and is the inverse of the frame rate — *e.g.* for a frame rate of 30 frames per second, $\Delta t = 1/30$. The instantaneous vehicle speed is estimated by averaging the values of s_i for a set of tracked features, namely

$$s = \frac{\sum_{i=1}^{n} s_i}{n} \tag{6.20}$$

where n is the number of tracked motion vectors.

The assumption that all the motion vectors for a vehicle lie on the same plane is a simplification, used so that the actual 3D position of the tracked features does not have to be discovered. As the actual license plates are always higher than the road level, the computed speeds will be greater than the actual vehicle speeds. To mitigate the effects of these erroneous measurements we multiply each vehicle's measured speed by a constant factor c, which is smaller than 1, namely

$$s' = s \cdot c \tag{6.21}$$

where s' is the instantaneous speed in m/s, computed for each interval of frames.

As shown in Chapter 7, the use of the c factor is simple but effective, as long as the tracked features have approximately the same distance from the ground.

In order to provide a fair comparison between the estimated speed and the ground truth speed, we define a *speed measurement region* close to the ground truth loop detectors, as shown in Figure 6.4. We average the instantaneous speed s' across frames for a given vehicle while it is located in the speed measurement region, resulting in the final speed S_f.

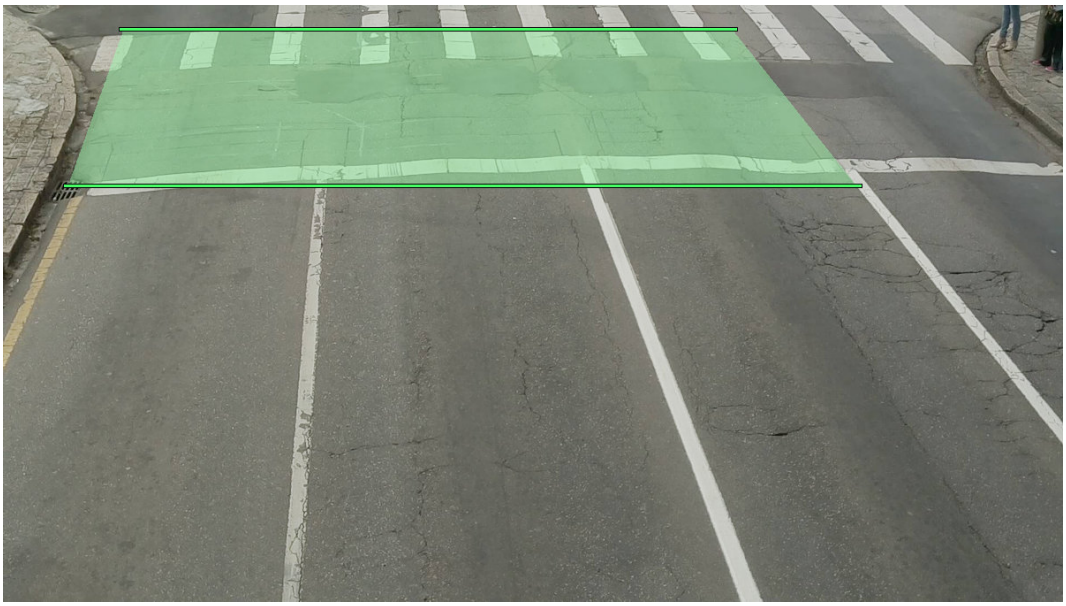


Figure 6.4: Speed measurement region.

Chapter 7

Experiments and Results

In this chapter, we present the experiments performed to evaluate our system, and a discussion about the obtained results. We first describe the dataset created for this purpose, as well as the considered metrics. We then present the parameters selected for the tested methods. Finally, we evaluate our system and compare it with other approaches.

7.1 Dataset

For our tests, we used 20 videos divided in 5 subsets according to weather and recording conditions, with frame resolution of 1920×1080 pixels and recorded at 30.15 frames per second. They are summarized in Table 7.1. In addition, we created for each video a *ground truth file*, in a simple XML format, containing the bounding boxes with the first license plate occurrence of each vehicle, and the corresponding ground truth speed. The ground truth for the vehicle license plates was obtained by human inspection of the images. The ground truth speeds were obtained from a high precision speed meter based on inductive loop detectors, properly calibrated and approved by the Brazilian national metrology agency (Inmetro).

Subset	Duration [min]	No. videos	No. vehicles	No. plates	No. speed	No. valid	Quality
01	34	4	1,146	1,128	1,033	1,019	[h]
02	169	11	4,829	4,713	4,345	4,241	[l]
03	26	2	960	936	876	855	[n]
04	41	2	1,045	1,034	928	917	[n,r]
05	20	1	869	800	795	734	[l,b]
Total	291	20	8,849	8,611	7,977	7,766	

Table 7.1: Characteristics of the videos in our dataset. The quality options are: [h] high-quality, [n] frames affected by natural or artificial noise, [l] frames affected by severe lighting conditions, [b] motion blur, and [r] rain.

The Venn diagram in Figure 7.1 shows the relation between the sets “vehicles” (*No. vehicles*), “vehicles with license plate” (*No. plates*), “vehicles with speed ground truth” (*No. speed*) and “vehicles with license plate **and** speed ground truth” (*No. valid*).

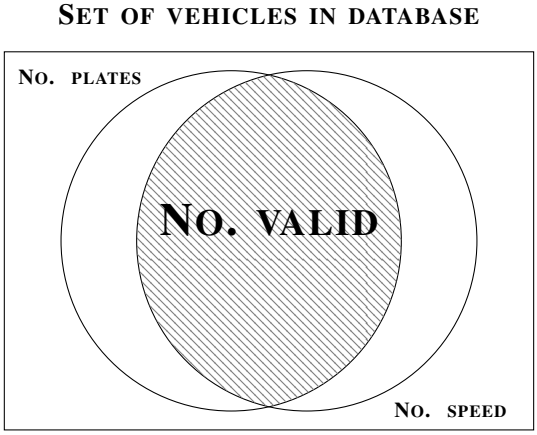


Figure 7.1: Set of vehicles considered for each problem: the rectangle depicts the set of vehicles used to evaluate the motion detection module; the left circle depicts the set of vehicles used to evaluate the license plate detection module; and the dashed area depicts the set of vehicles used to evaluate the entire system, that is, vehicles with license plates and with ground truth speed.

The histograms of vehicles per lane and vehicles per speed range, considering the entire dataset, are shown in Figures 7.2 and 7.3. Note, there is a concentration of vehicles in the [40-59] km/h speed range, this is due to the speed limit of this roadway, which is 60 km/h.

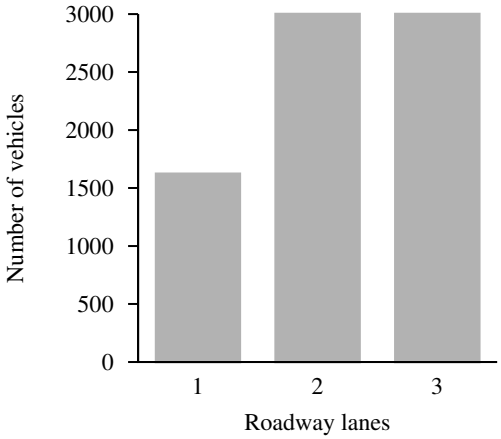


Figure 7.2: Vehicles per lane.

In our manual ground truth annotation, we include a flag for motorcycles and non-motorcycles (ordinary vehicles). From that information, we noted that only 4.5% of the total number of vehicles are motorcycles. We also noted that the inductive loop detector was able to measure the speed of 43% of all motorcycles and 92% of all ordinary vehicles. Indeed, the motorcycles represent only 2.1% of the *No. valid* dataset.

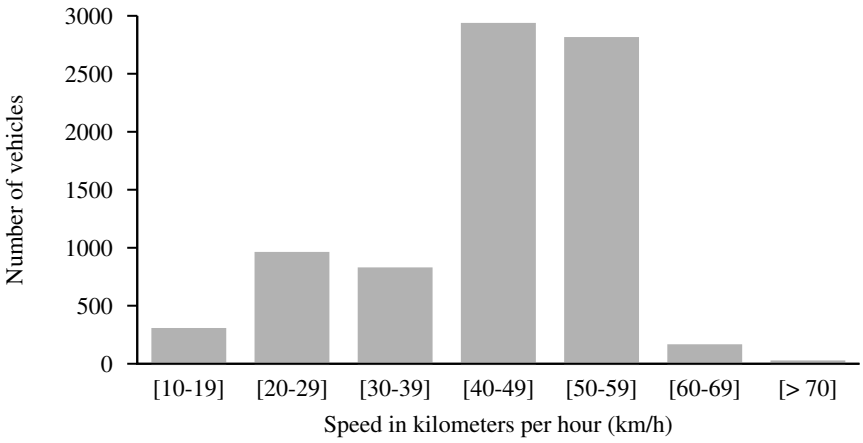


Figure 7.3: Vehicles speed histogram.

7.2 Metrics

To evaluate the motion detection performance, we used a metric to compare two rectangles r and s, which takes into account the disproportional size between a region of interest, which bounds the whole vehicle, and a license plate region. This metric is defined as

$$m(r,s) = \begin{cases} 1 & \text{if} \quad \dfrac{\text{area}(r \cap s)}{\min(\text{area}(r), \text{area}(s))} > \lambda \\ 0 & \text{otherwise} \end{cases} \tag{7.1}$$

for some threshold λ.

To evaluate the license plate detection performance, we used a metric to compare two rectangles r and s — which are supposedly to have nearly the same size — defined by the PASCAL Visual Object Detection Challenge (VOC) [57], defined as

$$m(r,s) = \begin{cases} 1 & \text{if} \quad \dfrac{\text{area}(r \cap s)}{\text{area}(r \cup s)} > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{7.2}$$

The threshold of 0.5 (50%), used in the PASCAL/VOC challenge, was meant to account for inaccuracies in the ground truth bounding boxes.

The extension of both metrics m to a set of rectangles $S = \{s_1, s_2, \ldots, s_{|S|}\}$ is given by the formula

$$m(r,S) = \max_{i=\{0,\ldots,|S|\}} m(r,s_i) \tag{7.3}$$

The *precision* p and *recall* r scores, as described by Wolf *et al.* [58], are given by

$$p = \frac{\sum_{i=1}^{|D|} m(d_i, G)}{|D|} \qquad r = \frac{\sum_{i=1}^{|G|} m(g_i, D)}{|G|} \tag{7.4}$$

where $G = \{g_1, g_2, \ldots, g_{|G|}\}$ is the set of ground truth license plate regions, and $D = \{d_1, d_2, \ldots, d_{|D|}\}$ is the set of license plates reported by the detector. For ranking purposes,

the ICDAR 2005 committee used the *f measure* [59] which is the harmonic mean of precision and recall,

$$f = \frac{2}{1/p + 1/r}. \tag{7.5}$$

7.3 Settings

In this section we describe the parameter selection for the compared methods and for the experimental setup. In order to properly select the fixed parameters of each method we built a "training set" with 5 videos (25% of the original dataset size), one from each different subset.

7.3.1 Camera settings

In our tests, we used a 5-megapixel CMOS image sensor, connected to a single-board computer called Raspberry PI for video compression and storage. The sensor was configured to capture images with 1920×1080 pixels at 30.15 frames per seconds, using a fixed *shutter* (time of exposure for each frame) of 1.8 milliseconds (except for subset 05, in which was used a shutter of 3.2 milliseconds). The videos were compressed with the standard H.264/AVC, configured with high-level quality options, in order to reduce the artifacts caused by video compression. The video camera was installed at 5.5 meters high. The videos were recorded during the day due to the lack of artificial illumination, and due to the low-cost sensor and lens of our camera, which has low light sensibility.

7.3.2 Motion detection settings

Since vehicles usually appear in the video sequence with relative high speed, we configure our binary energy image (MEI) to show fast movements, setting the pixel threshold to $\xi = 0.17$, and the expected motion to $\tau = 1$ frame interval (see Equations 3.1 and 3.2).

We optimize the motion detection by using a subsampling factor of $s = 4$, chosen according to our tests presented in Section 7.4.1. The pixel subsampling factor is described in Section 3.4. The vertical projection profile parameters of Equation 3.5 were configured with the smoothing window size to $\eta = 67/s + 1$ (*i.e.* $\eta = 17$), and the number of iterations to $\kappa = 3$.

The event area (EA) — see Figure 3.3 — was selected as a rectangle mainly at the bottom of the image, using $x1 = 24$, $x2 = 1896$, $y1 = 280$, and $y2 = 1056$ (in pixels coordinates).

7.3.3 License plate detection settings

The geometric parameters for the license plate detectors were chosen based on the manually annotated license plate dimensions in the ground truth (see Figure 7.4).

In the evaluation Section 7.4, we compare the proposed license plate detector to the following algorithms: the SnooperText [30], the Zheng *et al.* [28] algorithm, and the Stroke Width Transform (SWT) [29]. The geometric parameters were chosen according to the mean license plate width and height for our database. In order to choose the best non-geometric parameters for each evaluated method, we plotted figures containing the precision-recall and the f-measure, varying each parameter individually, as follows.

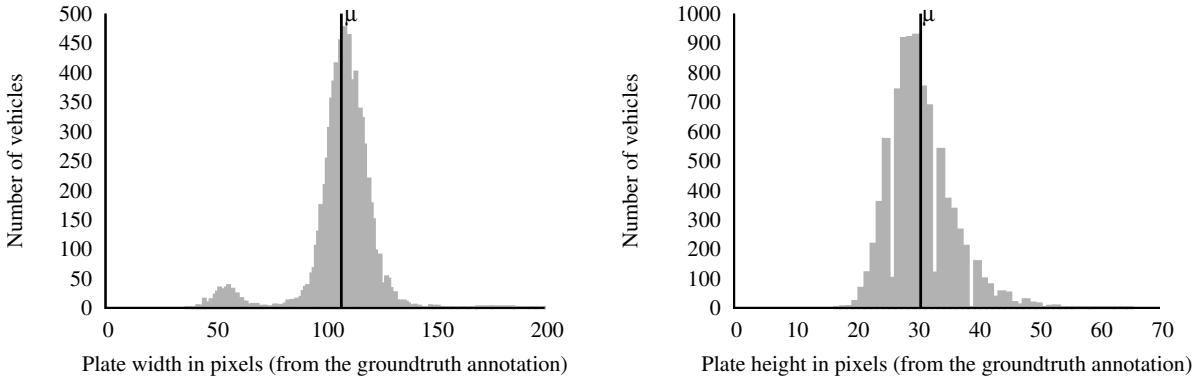


Figure 7.4: The license plate width and height histogram for the whole database. The mean width and height were 107 pixels and 30 pixels respectively.

Proposed license plate detector

The threshold value used to compute the vertical edges (see Section 4.1) for ordinary vehicles was chosen as $\gamma = 2$, as shown in Figure 7.5. The license plate of motorcycles was treated as a special case, and we find that $\gamma = 4$ is the best value for this kind of vehicles. The edge filtering stage (see Section 4.2) was configured with $w_{\min} = 4$, $h_{\min} = 4$, $w_{\max} = 300$, and $h_{\max} = 120$. In the region grouping process, we find that a structuring element $b = 1 \times 7$ is the best option for our training dataset, as shown in Figure 7.6. The minimum number of positive region classifications by T-HOG (see Section 4.4) was set up to $\upsilon = 8$, according to Figure 7.7.

As described by Minetto *et al.* [30], the region grouping module parameters t_1 (max relative

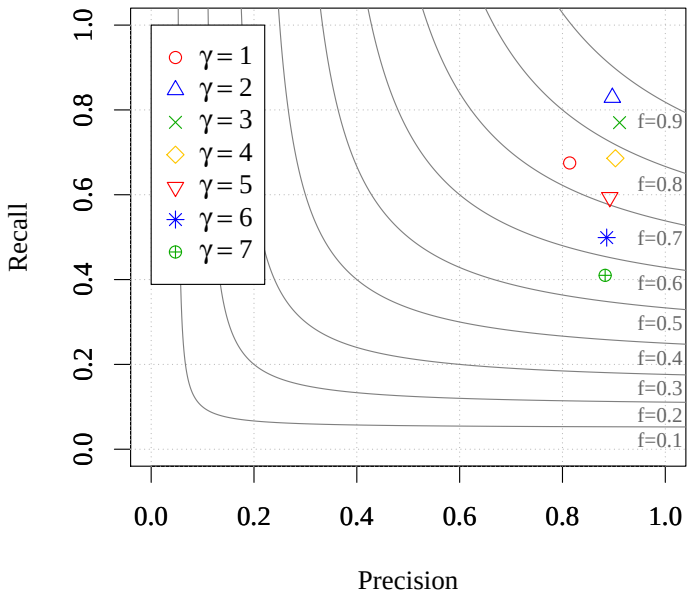


Figure 7.5: Performance of the proposed detector according to the edge threshold value γ.

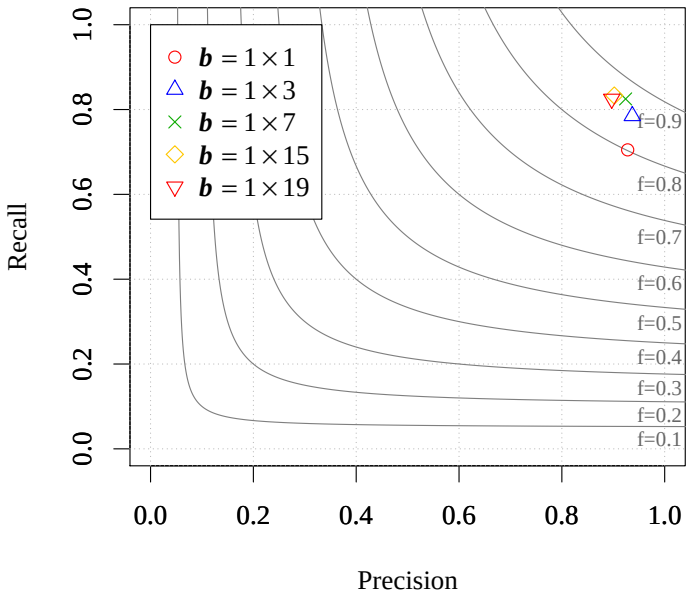


Figure 7.6: Performance of the proposed detector according to the structuring element size b.

height), t_2 (maximum relative letter spacing) and t_3 (max relative y offset) in Equation 4.6 of Section 4.3 were set up as 0.7, 1.1, and 0.4, respectively.

SnooperText

The fixed parameters of the SnooperText algorithm were set up as follows: we selected a window size of 19×19 pixels, as shown in Figure 7.8, for the toggle segmentation (m_s parameter); a minimum contrast ($c_m = 18$), see Figure 7.9; and the percentage of foreground/background classification ($p_{fb} = 86$), see Figure 7.10. These parameters are described in details in Minetto *et al.* [30]

Zheng *et al.* algorithm

In order to find the optimal values for the Zheng *et al.* algorithm, we vary the following parameters: the edge strength (e_s), the parameters T_{short} and T_{long}, to filter image edges that are too short or too long, and the percentage of license plate edges inside a rectangular region T_{plate}. These parameters are described in details in Zheng *et al.* [28]. As shown in Figures 7.11-7.13, the best values were found to be $e_s = 3$, $T_{\text{short}} = 14$, $T_{\text{long}} = 76$ and $T_{\text{plate}} = 0.25$.

Stroke Width Transform (SWT)

For the SWT detector, we vary the low and high threshold values of the Canny edge detector (T_{low} and T_{high}), as shown in Figure 7.14. Among those tested, we found that $T_{low} = 75$ and $T_{high} = 150$ were the best parameters.

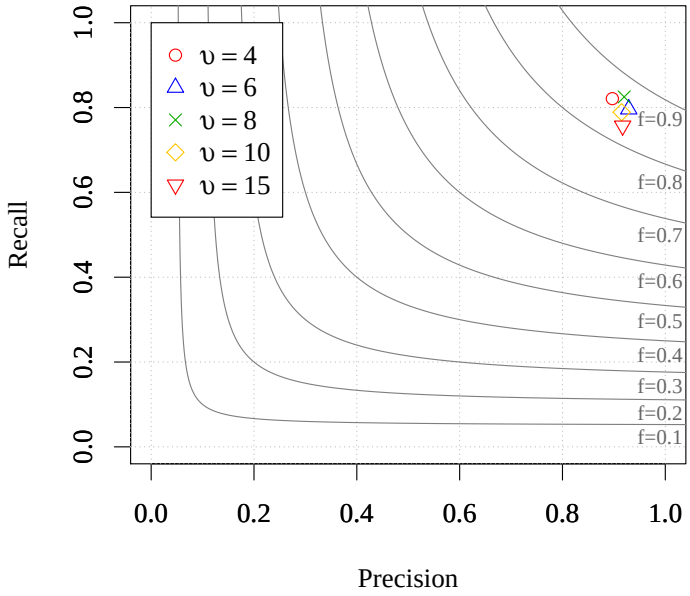


Figure 7.7: Performance of the proposed detector according to the minimum number of positive text regions υ.

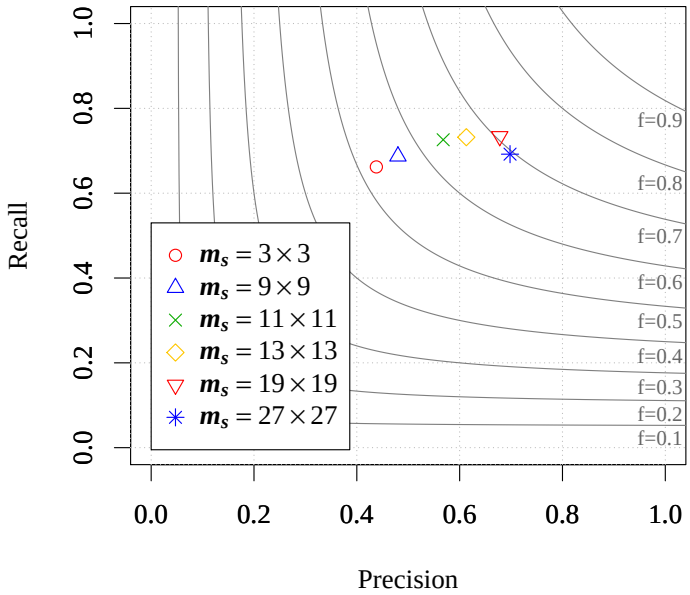


Figure 7.8: Performance of SnooperText according to the mask size parameter m_s.

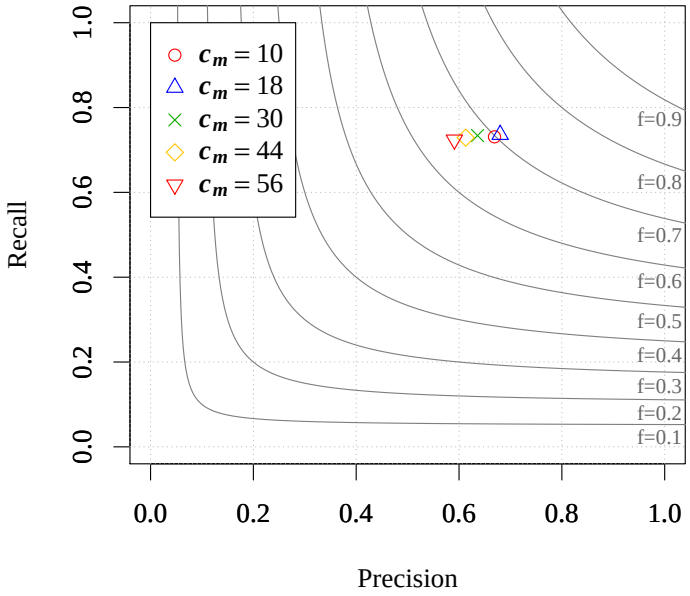


Figure 7.9: Performance of SnooperText according to the contrast parameter c_m.

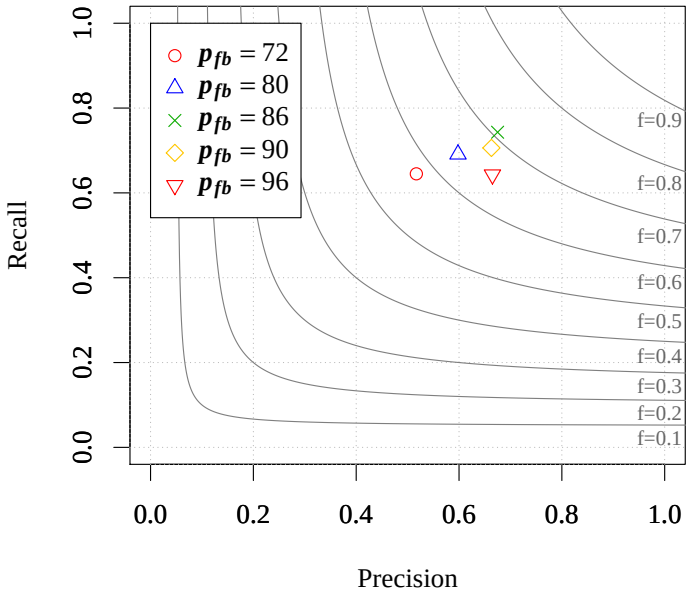


Figure 7.10: Performance of SnooperText according to the percentage of foreground/background selection parameter p_{fb}.

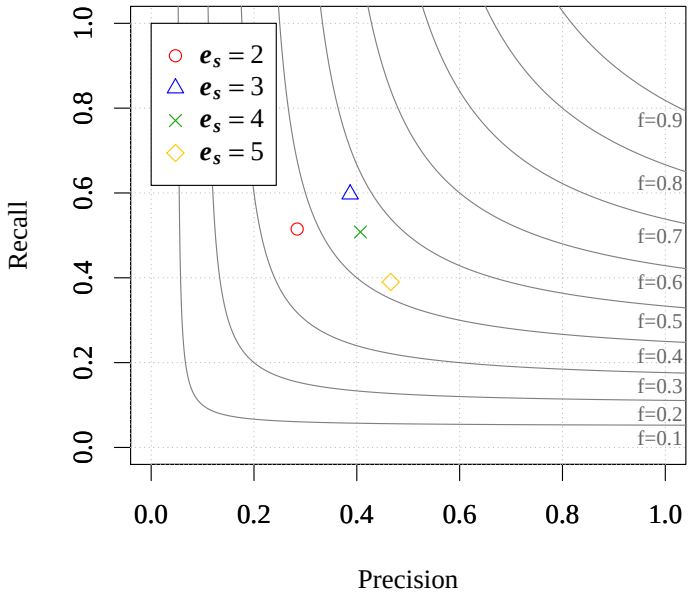


Figure 7.11: Performance of the Zheng *et al.* algorithm according to the edge strength parameter e_s.

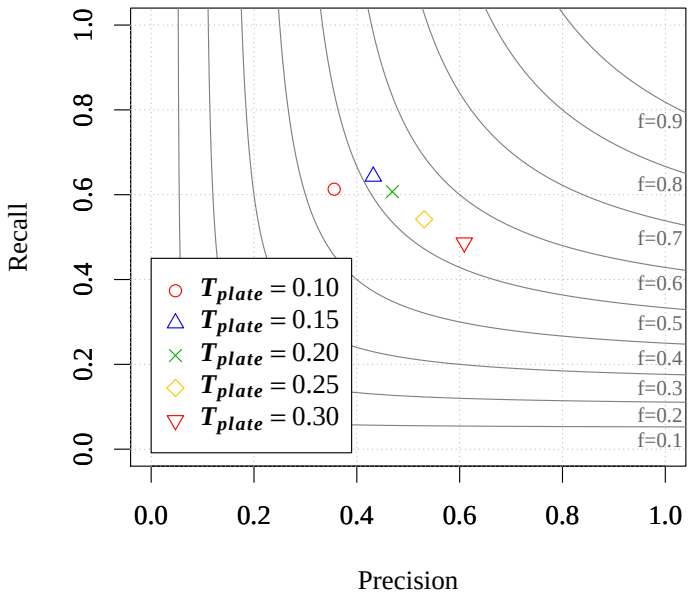


Figure 7.12: Performance of the Zheng *et al.* algorithm according to the percentage of license plate edges inside a rectangular region (T_{plate}).

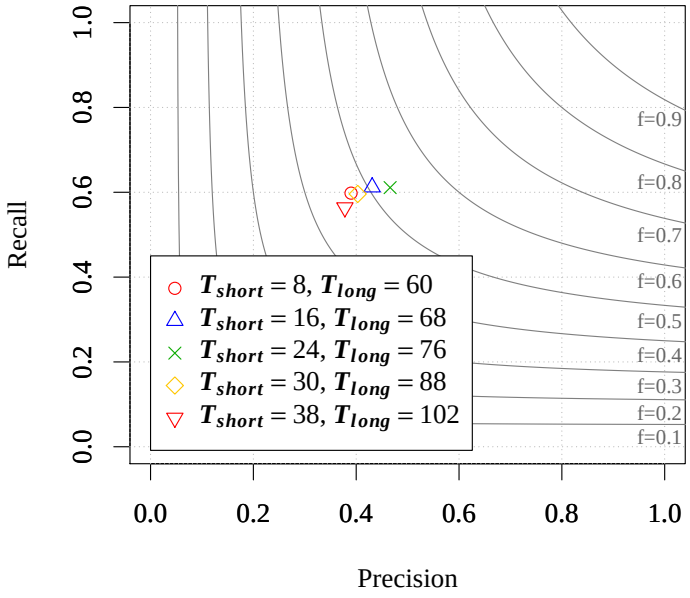


Figure 7.13: Performance of the Zheng *et al.* algorithm for the edge filtering parameters T_{short} and T_{long}.

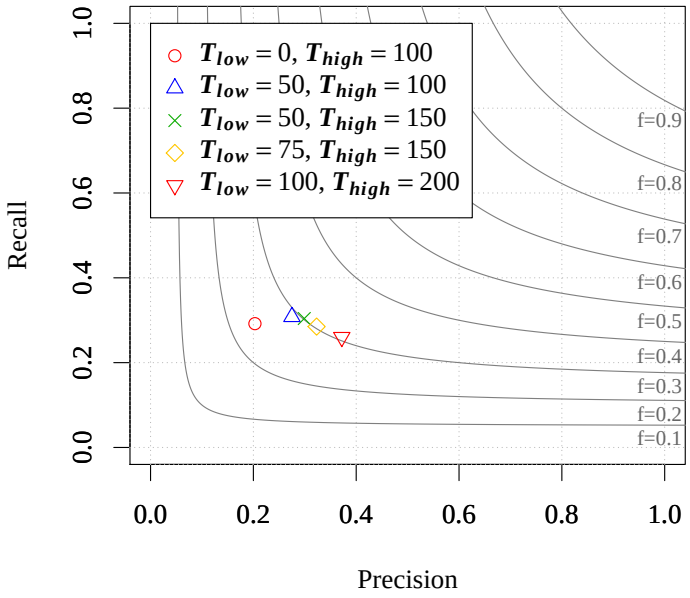


Figure 7.14: Performance of SWT varying the Canny low (T_{low}) and high (T_{high}) thresholds.

7.3.4 Speed estimation settings

The perspective rectification (see Section 6.2) was applied to each road lane individually. For each lane, four points were picked manually from the ground, near to the corners of the inductive loop detector. The rectified reference points are the vertices of a world rectangle with 2 meters wide and 4.8 meters high (see Figure 6.3).

The values used for perspective rectification and speed estimation for all road lanes are shown in Table 7.2. The resulting homography transformation matrices ($H_{\{1,2,3\}}$) are presented in Equation 7.6 — the values were truncated to four decimal places.

Road lane	(x_i^0, y_i^0)	(x_i^1, y_i^1)	(x_i^2, y_i^2)	(x_i^3, y_i^3)	R_w	R_h	c
Lane 1	(278, 52)	(525, 44)	(180, 293)	(494, 272)	2.00	4.80	0.972
Lane 2	(649, 43)	(896, 37)	(650, 266)	(954, 254)	2.00	4.80	0.933
Lane 3	(1020, 39)	(1266, 43)	(1108, 250)	(1406, 251)	2.00	4.80	0.901

Table 7.2: Perspective rectification and speed estimation setup for each road lane.

$$
\begin{aligned}
H_1 &= \begin{bmatrix} 0.4575 & 0.4551 & 95.8097 \\ 0.0402 & 1.6626 & -5.5362 \\ -0.0001 & 0.0011 & 1.0000 \end{bmatrix}, \\
H_2 &= \begin{bmatrix} 0.4589 & 0.7075 & 337.2005 \\ 0.0358 & 1.7245 & -7.7187 \\ -0.0000 & 0.0011 & 1.0000 \end{bmatrix}, \\
H_3 &= \begin{bmatrix} 0.3720 & 0.9234 & 589.9877 \\ -0.0363 & 1.7035 & 54.5783 \\ -0.0001 & 0.0011 & 1.0000 \end{bmatrix}
\end{aligned}
\tag{7.6}
$$

The feature tracker was configured to select 10 features without replacement, *i.e.* the features are selected only in the first frame where the license plate was detected.

7.4 Evaluation

In this section we used the full dataset (training-set + testing-set) for evaluation. The tests were carried out on an Intel Core i7 machine (2.2 GHz) with 12 GB of RAM running Linux and C++. In the next sections we evaluate each part of our system individually and, at the end of Section 7.4.3, we present the overall performance for the proposed system with all modules together.

7.4.1 Motion detection evaluation

The motion detection module uses a regular sparse grid for pixel sub-sampling — spanning the whole motion history image — in order to speed up the search process for vehicle motion. Note, however, that this time optimization may cause a super segmentation (region splitting) or loss in some parts of the object (regions without a license plate).

The sub-sampling ratio parameter was defined as $1/s$ for each row and column (see Section 3.4). For the sake of simplicity, let $s \times s$ be the image sub-sampling step for rows and columns, respectively. The trade-off between running time and precision-recall for several s values is shown in Table 7.3.

The performance was evaluated by using the precision and recall metrics of Equation 7.4, with the metric m of Equation 7.1. We performed tests with two different thresholds $\lambda = 1.0$ and $\lambda = 0.5$. The former consider only those license plates fully within some region of interest, while the latter consider those with at least 50% inside some region of interest.

Subsampling	1×1		2×2		4×4		8×8		16×16		32×32	
	p	r	p	r	p	r	p	r	p	r	p	r
$\lambda = 1.0$	0.86	0.99	0.86	0.99	0.86	0.99	0.81	0.99	0.69	0.99	0.58	0.99
$\lambda = 0.5$	0.87	0.99	0.88	0.99	0.87	1.00	0.84	1.00	0.74	1.00	0.65	1.00
Average time [ms]	21.50		7.97		3.54		1.36		0.61		0.20	

Table 7.3: ROI (with license plates) detection performance: the precision p, recall r and average time (in milliseconds, for each frame) for seven subsampling configurations and two overlapping thresholds.

Note that, the higher is the subsampling factor, the lower is the processing time. However, the number of ROIs without any license plate was also increased, as can be verified by the precision parameter, resulting in many unnecessary license plate detection calls. We chose a subsampling factor of 4×4 in our system.

7.4.2 License plate detection evaluation

We compared our license plate detector against three text and license plate detectors described in the literature (see Section 2.2). Specifically, we compared it with SnooperText [30], the Zheng *et al.* [28] algorithm, and the Stroke Width Transform (SWT) [29]. The results for each one of the five video subsets are shown in Table 7.4. As we can see, our detector significantly outperformed the other approaches in all tests.

Samples of license plate detection from the proposed method are shown in Figure 7.15. Note that the license plates were identified correctly even in situations with severe image noise or motion blur.

	PROPOSED DETECTOR			SNOOPERTEXT			ZHENG *et al.*			SWT		
	p	*r*	*f*	*p*	*r*	*f*	*p*	*r*	*f*	*p*	*r*	*f*
Subset01	**0.96**	**0.94**	**0.95**	0.81	0.88	0.84	0.92	0.88	0.90	0.76	0.61	0.68
Subset02	**0.92**	**0.84**	**0.88**	0.86	0.81	0.83	0.45	0.29	0.35	0.28	0.23	0.25
Subset03	**0.94**	**0.94**	**0.94**	0.56	0.79	0.66	0.87	0.90	0.88	0.66	0.62	0.64
Subset04	**0.94**	**0.92**	**0.93**	0.44	0.71	0.54	0.91	0.88	0.89	0.79	0.58	0.67
Subset05	**0.88**	**0.82**	**0.85**	0.76	0.72	0.74	0.48	0.48	0.48	0.15	0.15	0.15
Total	**0.93**	**0.87**	**0.90**	0.73	0.80	0.76	0.65	0.52	0.58	0.44	0.37	0.40

Table 7.4: License plate detection performance evaluation, where *p* means precision, *r* recall and *f* is the harmonic mean of precision and recall. The boldface values are the maxima in each row among the proposed detector, SnooperText, Zheng *et al.*, and SWT.

The detection errors were caused mainly in the hypothesis generation phase, where true license plate regions were eliminated by some filtering criteria due to the fact of being connected with a background region. Samples of detection errors are shown in Figure 7.16.

The average execution time to process each region of interest (ROI) was: 58 milliseconds for SnooperText; 918 milliseconds for Zheng *et al.*; 402 milliseconds for SWT; and 195 milliseconds for our detector. In Figure 7.17 are shown examples of images from the segmentation step from each evaluated license plate detector.

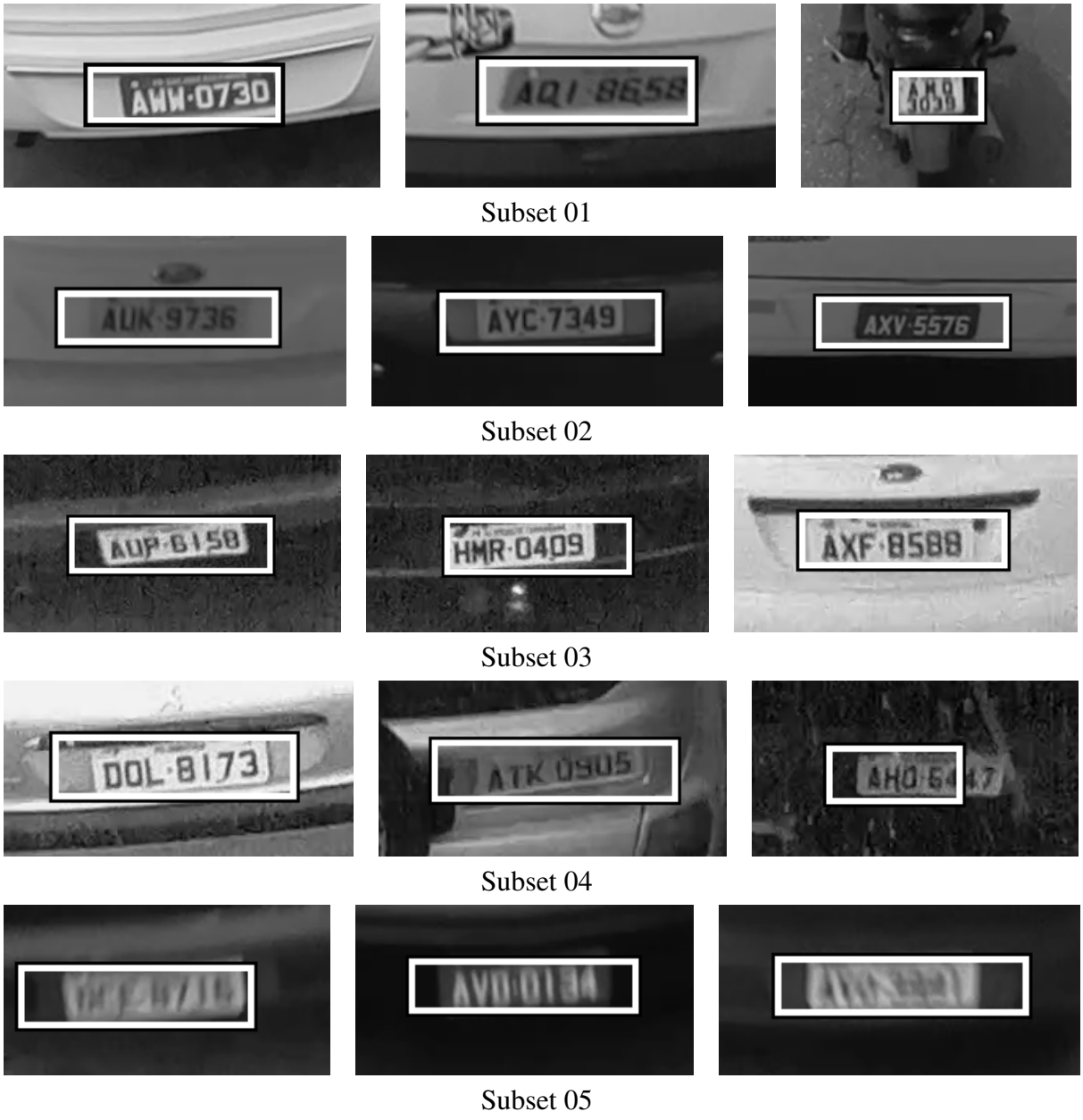
7.4.3 Vehicle speed estimation evaluation

The speed performance was determined by comparing the estimated speed, returned by a given system, with the available ground truth speed, obtained from the inductive loop detector. According to the standard adopted in the USA [60], an acceptable estimative of speed must be within the [-3 km/h, $+2$ km/h] error interval. These limits are more restrictive than those adopted in Brazil (of [-3 km/h, $+3$ km/h]) [61]. Therefore, we adopted in this section the USA standard, however some results for the Brazilian standard are shown in Table 7.5.

The speed error distribution for our complete system, considering the *No. valid* vehicles set, is shown in Figure 7.18. The maximum nominal speed error values for the whole dataset were -4.68 km/h and $+6.00$ km/h, with a standard deviation of 1.36 km/h.

Examples of vehicle speed estimation using the complete system are shown in Figures 7.19 and 7.20. The feature tracker, set up to select the best 10 features in the license plate region, achieved a percentage of 99.2% of vehicles being correctly tracked from the bottom part of the images until the region of measurement. The averaged number of features tracked with success, until the region of measurement, was of 5.7. On average, our tracking module spent 49.8 milliseconds per frame.

In order to verify the assumption that distinctive features from a license plate region are good features to track and to estimate the vehicle speed, we compared our system against a



Subset 01

Subset 02

Subset 03

Subset 04

Subset 05

Figure 7.15: Examples of license plate detection, returned by our system, for representative samples of each subset.

similar approach but allowing "*a free feature selection*", that is, a feature selection spread over the whole vehicle region. The results are shown in Figure 7.21. For the same number of tracked features, 10 in both experiments, our system achieved 96.0% of vehicles inside the speed limit, while with a free selection of features only 73.4% were inside that limit. The averaged number of features tracked with success, until the region of measurement, was of 4.6.

We also tested our system with an *ideal license plate detector*. That is, we used the manually annotated license plate regions to verify the speed performance of our system in the presence of very poor license plate regions, hard to be identified even by an human observer. See Figure 7.22. As shown in Table 7.5, we achieved in this scenario 96.1% of vehicles inside the speed limit.

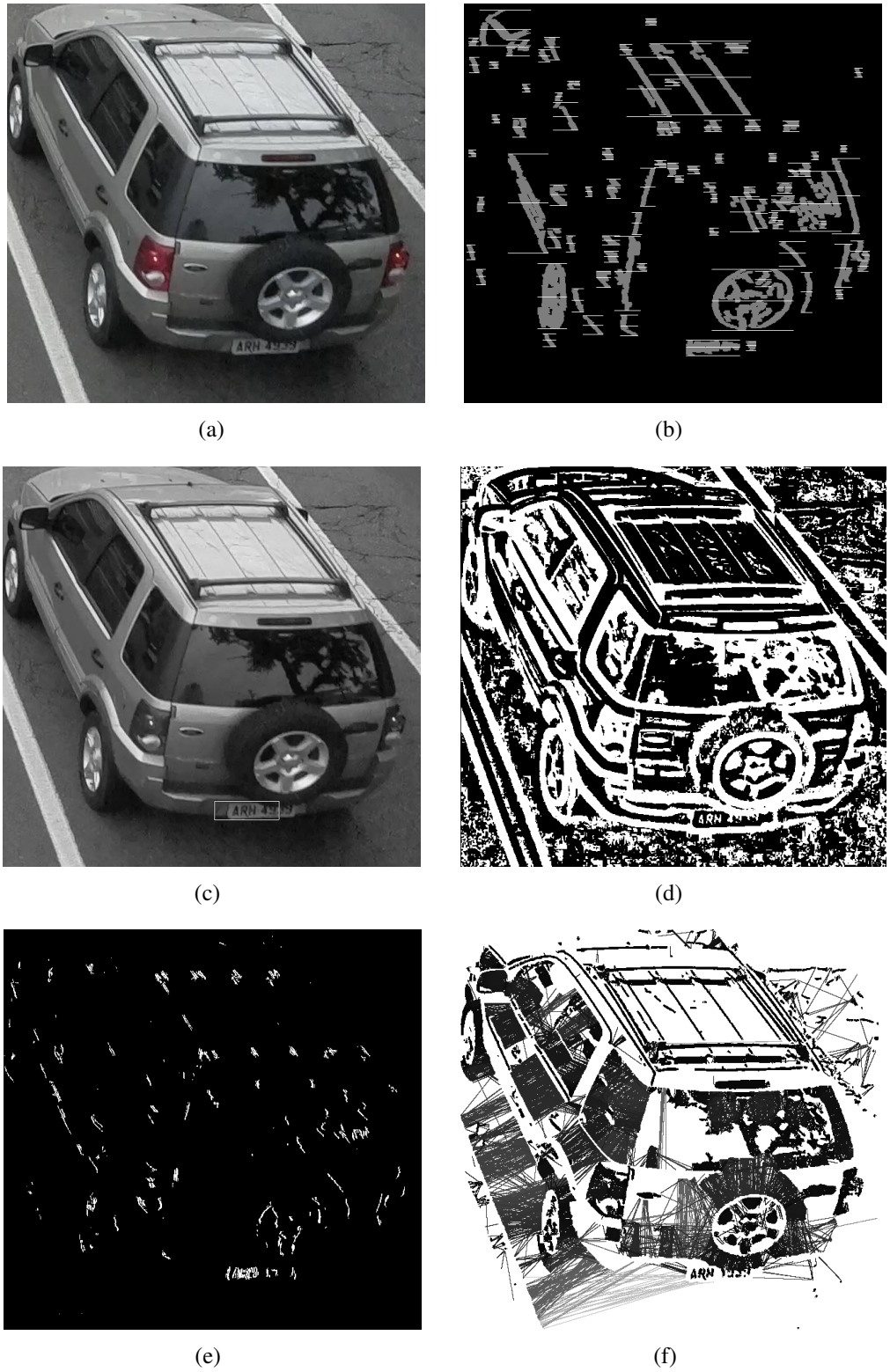
Figure 7.16: Samples of license plate detection errors returned by our system.

We also compared our system against a blob-based tracker. In this experiment, we used a particle filter algorithm [62, 63] to track a region of interest, which was found by our motion detection module and through an extensive set of experiments to determine the best filter parameters and the ROI dimension. However, this approach was found to be unstable for our application. We suspect that the reason is that our camera is installed very close to the vehicles, in a way that the probabilistic search cannot precisely define the position of the vehicle in all frames. The speed estimation by using this approach is shown in Figure 7.23.

	PROPOSED SYSTEM			PARTICLE FILTER			IDEAL DETECTOR			FREE FEATURE SELECTION		
	Low	Ideal	High	Low	Ideal	High	Low	Ideal	High	Low	Ideal	High
USA standard [−3/+2 km/h]	1.1%	96.0%	2.8%	20.3%	22.1%	57.6%	0.9%	96.1%	3.0%	11.3%	73.4%	15.3%
Brazilian standard [±3 km/h]	1.1%	98.2%	0.7%	20.3%	30.9%	48.8%	0.9%	98.5%	0.6%	11.3%	79.9%	8.8%

Table 7.5: Speed estimation results considering four scenarios: the proposed system, the particle filter, the ideal detector (using the license plates from the ground truth), and the free selection of features in the vehicle. The columns “Low”, “Ideal”, and “High” represent speed errors below, above and within the acceptable limits, respectively.

Finally, we suspect that the assumption that all the vehicle license plates are located nearly at the same height from the ground is the main cause of speed estimation errors. We show in Figure 7.24(a) a sample from a negative error, where the license plate seems to be close to the road plane, and in Figure 7.24(b) a sample from a positive error, where the license plate is located above the expected height for a plate in an ordinary vehicle.



(a) (b) (c) (d) (e) (f)

Figure 7.17: Examples of images from the segmentation step for all detectors. In (a) the original image. In (b), the segmentation of the proposed detector and (c) the license plate detected, followed by the segmentation image of (d) SnooperText, (e) the Zheng *et al.* algorithm, and (f) SWT.

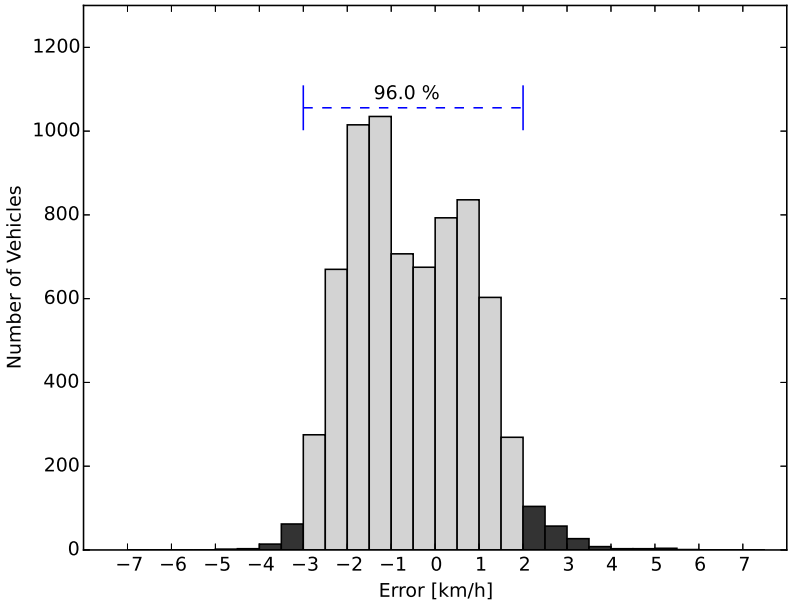


Figure 7.18: Speed error distribution for the whole database using the proposed approach to track features inside the license plate.

(a) 33.8 km/h (real 35.0 km/h)

(b) 48.0 km/h (real 47.5 km/h)

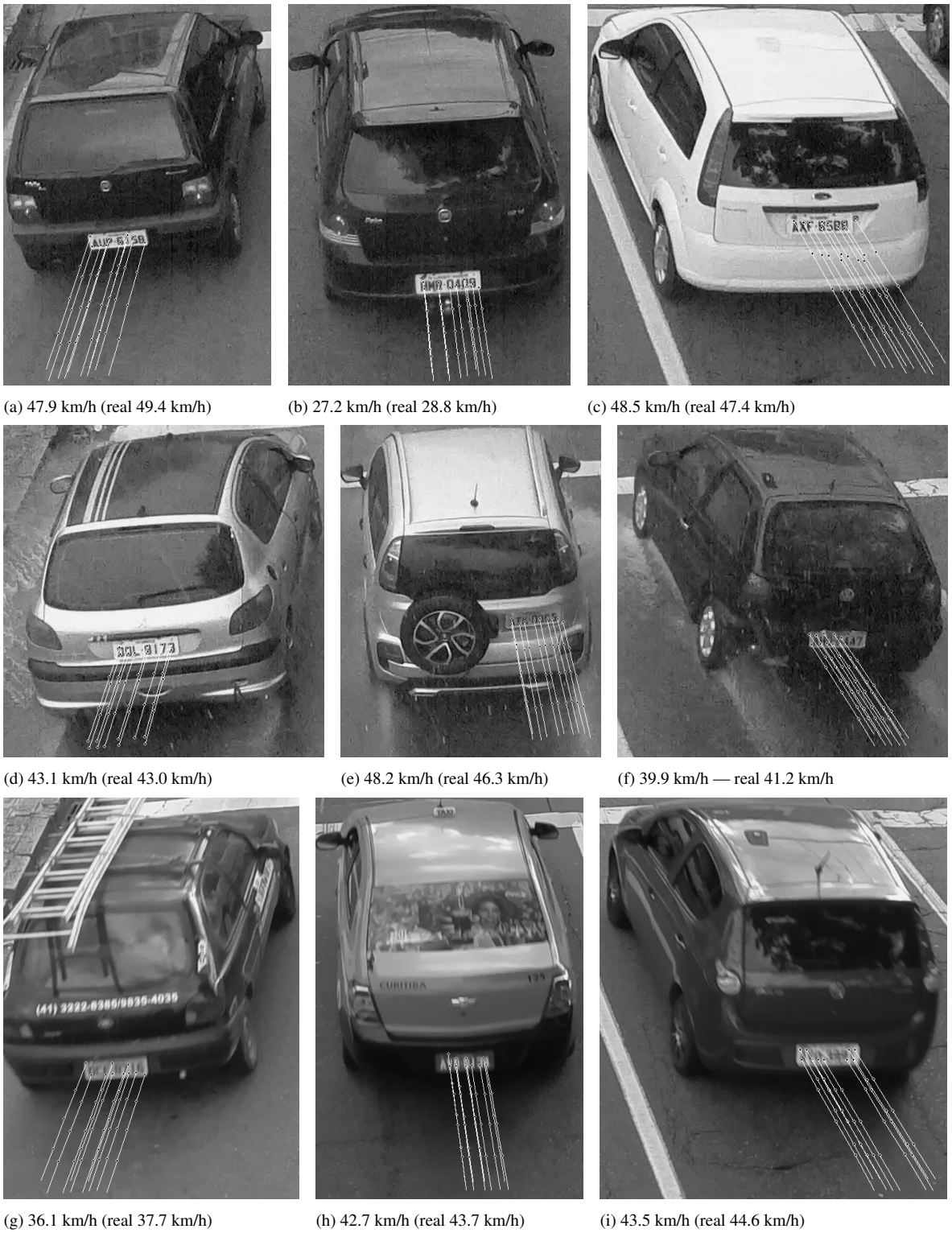
(c) 54.7 km/h (real 54.6 km/h)

(d) 45.7 km/h (real 46.1 km/h)

(e) 46.2 km/h (real 47.7 km/h)

(f) 45.5 km/h — real 48.0 km/h

Figure 7.19: Examples of vehicle speed estimation for those license plates shown in Figure 7.15. Samples from subset 01 (a,b,c) and subset 02 (d,e,f).



(a) 47.9 km/h (real 49.4 km/h)
(b) 27.2 km/h (real 28.8 km/h)
(c) 48.5 km/h (real 47.4 km/h)
(d) 43.1 km/h (real 43.0 km/h)
(e) 48.2 km/h (real 46.3 km/h)
(f) 39.9 km/h — real 41.2 km/h
(g) 36.1 km/h (real 37.7 km/h)
(h) 42.7 km/h (real 43.7 km/h)
(i) 43.5 km/h (real 44.6 km/h)

Figure 7.20: Examples of vehicle speed estimation for those license plates shown in Figure 7.15. Samples from subset 03 (a,b,c), subset 04 (d,e,f) and subset 05 (g,h,i).

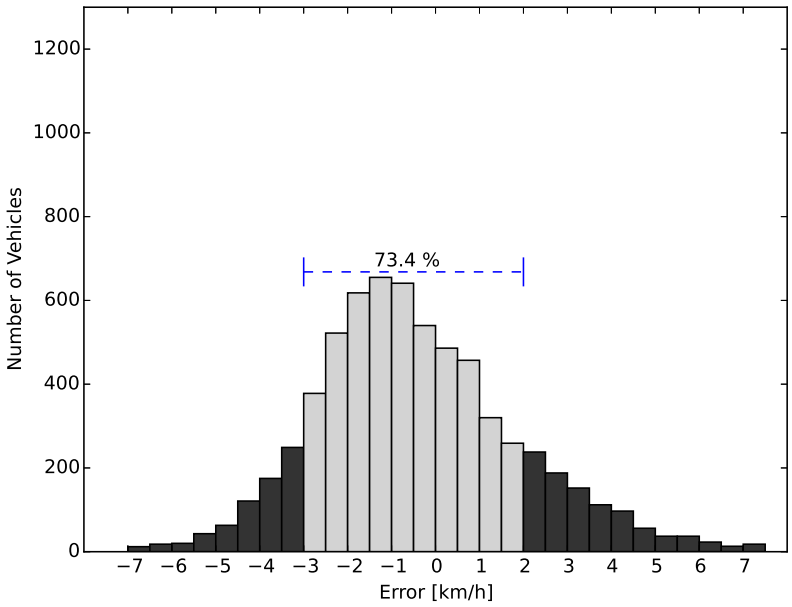


Figure 7.21: Speed error distribution for free feature selection (feature selection spread over the whole vehicle region): 73.4% of samples (light grey region) were inside the error interval of +2/-3 km/h (maximum speed error allowed by USA regulatory authorities).

Figure 7.22: Examples of manually annotated license plates in poor conditions.

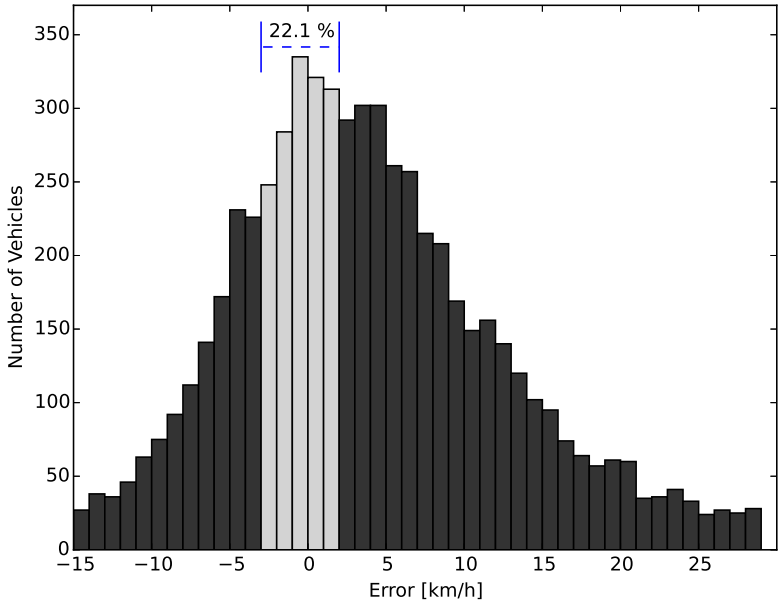


Figure 7.23: Speed error distribution for a particle filter tracker: only 22.1% of samples (light grey region) were inside the error interval of +2/-3 km/h (maximum speed error allowed by USA regulatory authorities).

(a) 67.3 km/h (real 72.0 km/h)

(b) 61.8 km/h (real 56.1 km/h)

Figure 7.24: Samples of speed estimation errors returned by our system.

Chapter 8

Conclusions

In this master thesis, we were primarily concerned with the vehicle speed estimation problem. For that purpose, we collected almost five hours of full-hd quality videos (in a total of 36 gigabytes), with associated ground truth speed (obtained from a high precision inductive loop detector). The videos, with more than 8,000 vehicles from three different road lanes, were recorded with a low cost camera, from a urban roadway of Curitiba, a city of Brazil, under different weather and daylight conditions.

The proposed system is based on the selection and tracking of distinctive features within each vehicle for speed estimation. Therefore, in order to reduce the complexity in handling such amount of data, we developed a motion detection method, with a real-time response, to efficiently narrow down the regions of the scene with vehicle motion. Then, we suppose that distinctive features from the license plate region, such as letters corners, were good candidates to estimate the speed, since they are in accordance with well-established criteria used to select good features to track and are nearly at the same height from the ground.

We tested a well-known license plate detector and two state-of-the-art urban scene text detectors for the task of license plate detection. However, they performed poorly in our problem domain. To overcome this limitation, we proposed a novel license plate detector which uses a texture classifier specialized to capture the gradient distribution characteristics of character strokes that make the license plate letters. Then, to evaluate the performance of this part we compiled from our videos, a set of ground truth files, in a simple XML format, containing the bounding boxes of the license plate occurrences for each vehicle. We have shown that our license plate detector achieved a 93% of precision and 87% of recall on this benchmark, outperforming the other approaches.

We also have shown that the selective extraction of distinctive features from the license plate region outperformed an approach based on a free feature selection spread over the whole vehicle, and also outperformed a particle feature tracking approach. In our experiments, the speed estimation had an average error of -0.5 km/h, staying inside the +2/-3 km/h limit, determined by the regulatory authorities in several countries, in over 96.0% of the cases.

As future work, we intend to verify if the distance of the license plate to the ground can improve the speed estimation and we aim to apply an OCR on the detected license plates in order to create a traffic speed control system with integrated surveillance tools, *e.g.* to compute the traffic flow, to identify stolen vehicles or that have not paid the taxes, etc.

Bibliography

[1] Tom V. Mathew. Intrusive and Non-Intrusive Technologies. Technical report, 2014.

[2] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

[3] Bruce D. Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. *Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[4] Carlo Tomasi and Takeo Kanade. Detection and Tracking of Point Features. Technical report, 1991.

[5] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004.

[6] C. Sun and S. Ritchie. Individual vehicle speed estimation using single loop inductive waveforms. *Journal of Transportation Engineering*, 125(6):531–538, 1999.

[7] I. Sina, A. Wibisono, A. Nurhadiyatna, B. Hardjono, W. Jatmiko, and P. Mursanto. Vehicle counting and speed measurement using headlight detection. In *International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 149–154, Sept 2013.

[8] Wei Zhang, Q.M.J. Wu, Guanghui Wang, and Xinge You. Tracking and pairing vehicle headlight in night scenes. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):140–153, March 2012.

[9] Ren-Huang Liou, Yi-Bing Lin, Yu-Long Chang, and Ming-Feng Chang. Deriving the vehicle speeds from mobile telecommunications network. In *International Conference on ITS Telecommunications (ITST)*, pages 429–432, Nov 2012.

[10] K. Osamura, A. Yumoto, and O. Nakayama. Vehicle speed estimation using video data and acceleration information of a drive recorder. In *International Conference on ITS Telecommunications (ITST)*, pages 157–162, Nov 2013.

[11] D.J. Dailey, F.W. Cathey, and S. Pumrin. An algorithm to estimate mean traffic speed using uncalibrated cameras. *IEEE Transactions on Intelligent Transportation Systems*, 1(2):98–107, Jun 2000.

[12] V.K. Madasu and M. Hanmandlu. Estimation of vehicle speed by motion tracking on image sequences. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 185–190, June 2010.

[13] G. Garibotto, P. Castello, E. Del Ninno, P. Pedrazzi, and G. Zan. Speed-vision: speed measurement by license plate reading and tracking. In *IEEE Intelligent Transportation System*, pages 585–590, 2001.

[14] H. Zhiwei, L. Yuanyuan, and Y. Xueyi. Models of Vehicle Speeds Measurement with a Single Camera. *International Conference on Computational Intelligence and Security Workshops (CISW)*, pages 283–286, 2007.

[15] C. Maduro, K. Batista, P. Peixoto, and J. Batista. Estimation of Vehicle Velocity and Traffic Intensity Using Rectified Images. *IEEE International Conference on Image Processing (ICIP)*, pages 777–780, 2008.

[16] H. Palaio, C. Maduro, K. Batista, and J. Batista. Ground plane velocity estimation embedding rectification on a particle filter multi-target tracking. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 825–830, May 2009.

[17] Huei-Yung Lin, Kun-Jhih Li, and Chia-Hong Chang. Vehicle speed detection from a single motion blurred image. *Image and Vision Computing (IVC) - Elsevier*, 26(10):1327–1337, 2008.

[18] Sedat Dogan, Mahir Serhan Temiz, and Sitki Kulur. Real time speed estimation of moving vehicles from side view images from an uncalibrated video camera. *Sensors*, 10(5):4805–4824, 2010.

[19] Witold Czajewski and Marcin Iwanowski. Vision-based vehicle speed measurement method. In *International Conference on Computer Vision and Graphics: Part I (ICCVG)*, 10, pages 308–315, 2010.

[20] C. H. Xiao and N. H. C. Yung. A Novel Algorithm for Estimating Vehicle Speed from Two Consecutive Images. *IEEE Workshop on Applications of Computer Vision (WACV)*, page 12, 2007.

[21] H. A. Rahim, U. U. Sheikh, R. B. Ahman, and A. S. M. Zain. Vehicle Velocity Estimation for Traffic Survillance System. *World Academy of Science, Engineering and Technology (WASET)*, page 772, 2010.

[22] T.N. Schoepflin and D.J. Dailey. Dynamic Camera Calibration of Roadside Traffic Management Cameras for Vehicle Speed Estimation. *Intelligent Transportation Systems (ITS)*, 2003.

[23] L. Grammatikopoulos, G. Karras, and E. Petsa. Automatic Estimation of Vehicle Speed from Uncalibrated Video Sequences. *Modern Technologies, Education and Professional Practice in Geodesy and Related Fields*, pages 332–338, 2005.

[24] Ilkwang Lee, Hanseok Ko, and D. K. Han. Multiple Vehicle Tracking Based on Regional Estimation in Nighttime CCD Images. *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 4:IV–3712–IV–3715, 2002.

[25] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[26] Christos-Nikolaos E. Anagnostopoulos, Ioannis E. Anagnostopoulos, Ioannis D. Psoroulas, Vassili Loumos, and Eleftherios Kayafas. License plate recognition from still images and video sequences: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 9(3):377–391, 2008.

[27] Shan Du, Mahmoud Ibrahim, Mohamed Shehata, and Wael Badawy. Automatic license plate recognition (ALPR): A state-of-the-art review. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(2):311–325, 2013.

[28] Danian Zheng, Yannan Zhao, and Jiaxin Wang. An efficient method of license plate location. *Pattern Recognition Letters (PRL)*, 26(15):2431–2438, 2005.

[29] Boris Epshtein, Eyal Ofek, and Yonatan Wexler. Detecting text in natural scenes with stroke width transform. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 886–893. IEEE Computer Society, 2010.

[30] Rodrigo Minetto, Nicolas Thome, Matthieu Cord, Neucimar J. Leite, and Jorge Stolfi. SnooperText: A text detection system for automatic indexing of urban scenes. *Computer Vision and Image Understanding*, 122(0):92–104, 2014.

[31] John Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, PAMI-8(6):679–698, 1986.

[32] Rodrigo Minetto, Nicolas Thome, Matthieu Cord, Neucimar J. Leite, and Jorge Stolfi. T-HOG: An effective gradient-based descriptor for single line text regions. *Pattern Recognition (PR), Elsevier,*, 46(3):1078–1090, 2013.

[33] K. Deb and Kang-Hyun Jo. HSI color based vehicle license plate detection. In *International Conference on Control, Automation and Systems (ICCAS)*, pages 687–691, Oct 2008.

[34] Vahid Abolghasemi and Alireza Ahmadyfard. An edge-based color-aided method for license plate detection. *Image and Vision Computing (IVC) - Elsevier*, 27(8):1134–1142, 2009.

[35] J. Ilonen, J.-K. Kamarainen, P. Paalanen, M. Hamouz, J. Kittler, and H. Kalviainen. Image feature localization by multiple hypothesis testing of gabor features. *IEEE Transactions on Image Processing*, 17(3):311–325, March 2008.

[36] Huiping Li, David Doermann, and Omid Kia. Automatic text detection and tracking in digital video. *IEEE Transactions on Image Processing (TIP)*, 9(1):147–156, 2000.

[37] W. Zhou, Houqiang Li, Yijuan Lu, and Qi Tian. Principal visual word discovery for automatic license plate detection. *IEEE Transactions on Image Processing (TIP)*, 21(9):4269–4279, Sept 2012.

[38] A.F. Bobick and J.W. Davis. The recognition of human movement using temporal templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 23(3):257–267, Mar 2001.

[39] R. Venkatesh Babu and K. R. Ramakrishnan. Recognition of human actions using motion history information extracted from the compressed video. *Image Vision Computing (IVC) - Elsevier*, 22(8):597–607, 2004.

[40] Md. Atiqur Rahman Ahad, J. K. Tan, H. Kim, and S. Ishikawa. Motion History Image: Its Variants and Applications. *Machine Vision Applications (MVA) - Springer*, 23(2):255–281, Mar 2012.

[41] Jaekyu Ha, R.M. Haralick, and I.T. Phillips. Document page decomposition by the bounding-box project. In *International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, pages 1119–1122 vol.2, Aug 1995.

[42] Rodrigo Minetto, Nicolas Thome, Matthieu Cord, Jonathan Fabrizio, and Beatrice Marcotegui. Snoopertext: A multiresolution system for text detection in complex visual scenes. In *IEEE International Conference on Image Processing (ICIP)*, pages 3861–3864, 2010.

[43] Thomas Retornaz and Beatriz Marcotegui. Scene text localization based on the ultimate opening. In *International Symposium on Mathematical Morphology (ISMM)*, volume 1, pages 177–188, 2007.

[44] Barton L. Anderson and Pawan Sinha. Reciprocal interactions between occlusion and motion computations. In *Proceedings of the National Academy of Sciences (1997)*, volume 94, pages 3477–3480, Apr 1997.

[45] David Marr, Tomaso Poggio, and Shimon Ullman. Bandpass channels, zero-crossings, and early visual information processing. *Journal of the Optical Society of America*, 69(6):914–916, 1979.

[46] Les Kitchen and Azriel Rosenfeld. Gray-level corner detection. *Pattern Recognition Letters (PRL) - Elsevier*, 1(2):95–102, 1982.

[47] Hans Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. In *Technical report CMU-RI-TR-80-03, Carnegie Mellon University*, number CMU-RI-TR-80-03. September 1980.

[48] Jean-Yves Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker. *Intel Corporation, Microprocessor Research Labs*, 2000.

[49] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European Conference on Computer Vision (ECCV)*, pages 404–417, 2006.

[50] Krystian Mikolajczyk and Cordelia Schmid. A Performance Evaluation of Local Descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 27(10):1615–1630, 2005.

[51] F. E. Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11:1–21, 1969.

[52] Dr. Gary Rost Bradski and Adrian Kaehler. *Learning OpenCV, - 1st Edition*. O'Reilly Media, Inc., first edition, 2008.

[53] George Vogiatzis, Philip H. S. Torr, Steven M. Seitz, and Roberto Cipolla. Reconstructing relief surfaces. *Image and Vision Computing (IVC) - Elsevier*, 26(3):397–404, 2008.

[54] Antonio Criminisi, Ian Reid, and Andrew Zisserman. A plane measuring device. 1997.

[55] Hua Li, Mingyue Feng, and Xiao Wang. Inverse perspective mapping based urban road markings detection. In *IEEE International Conference on Cloud Computing and Intelligent Systems (CCIS)*, volume 03, Oct 2012.

[56] Guanghui Wang, Zhanyi Hu, Fuchao Wu, and Hung-Tat Tsui. Single view metrology from scene constraints. *Image and Vision Computing (IVC) - Elsevier*, 23(9):831–840, 2005.

[57] Mark Everingham, Luc Van Gool, C. K. I. Williams, J. Winn, and Andrew Zisserman. The PASCAL Visual Object Classes (VOC) challenge, 2009.

[58] C. Wolf and J.-M. Jolion. Object count/area graphs for the evaluation of object detection and segmentation algorithms. *International Journal on Document Analysis and Recognition (IJAR)*, 8(4):280–296, 2006.

[59] S.M. Lucas. Icdar 2005 text locating competition results. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 80–84 Vol. 1, Aug 2005.

[60] U.S. Departament of Transportation. Speed-measuring Device Performance Specifications: Across-the Road Radar Module, 2007.

[61] Portaria Inmetro n 544, de 12 de dezembro de 2014. `http://www.inmetro.gov.br/legislacao/rtac/pdf/RTAC002192.pdf`. Accessed: 2015-07-16.

[62] P. Perez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In *European Conference on Computer Vision (ECCV)*, pages 661–675, 2002.

[63] Particle Filter Object Tracking. `https://web.engr.oregonstate.edu/~hess/downloads/track.tar.gz`. Accessed: 2015-07-16.