EURASIP Journal on Image and Video Processing
a SpringerOpen Journal

RESEARCH Open Access

Locating moving objects in car-driving sequences

Antonio Garcia-Dopico*, José Luis Pedraza, Manuel Nieto, Antonio Pérez, Santiago Rodríguez and Luis Osendi

Abstract

This paper presents a system for the search and detection of moving objects in a sequence of images previously captured by a camera installed in a conventional vehicle. The objective is the design and implementation of a software system based on optical flow analysis to detect and identify moving objects as perceived by a driver, taking into account that these objects could interfere with the driver's behavior, either through distraction or by posing an actual danger that may require an active response. The problem presents significant difficulties because the vehicle travels on conventional roads, open to normal traffic. Consequently, the scenes are recorded with natural lighting, i.e., under highly variable conditions (intensity, shadows, etc.). Furthermore, the use of a moving camera makes it difficult to properly identify static objects such as the road itself, signals, buildings, landscapes, and moving objects of the same speed, such as pedestrians or other vehicles. The proposed method consists of three stages. First, the optical flow is calculated for each image of the sequence, as a first estimate of the apparent motion. In a second step, two segmentation processes are addressed: the optical flow itself and the images of the sequence. Finally, in the last stage, the results of these two segmentation processes are combined to obtain the movement of the objects present in the sequence, identifying both their direction and magnitude. The quality of the results obtained with different sequences of real images makes this software suitable for systems to study driver behavior and to help detect danger situations, as various international traffic agencies consider in their research projects.

Keywords: Optical flow; Optical flow segmentation; Image segmentation; Real traffic; Driver behavior; Natural lighting

1 Introduction

This paper addresses the detection of moving objects that could interfere with driver behavior, either through distraction or by posing an actual danger. The scene is recorded as seen by the driver, i.e., with a moving camera. The optical flow obtained from these video sequences is computed to obtain an estimate of the apparent motion present in the scene. The camera is installed inside a conventional vehicle driving on public roads. The moving camera and the variable natural lighting pose a serious challenge for calculating optical flow. Taking the sequences while driving on actual roads also implies a large number of objects captured by the images (vehicles, vegetation, signs, buildings, etc.). This makes it difficult to correctly determine optical flow.

The main novelty present in our solution is the combined use of two independent and complementary segmentation processes, optical flow segmentation and raw image segmentation. Once both processes are finished, an additional computation stage is dedicated to find matches between both segmented image results. This stage also identifies objects with relatively high apparent motion to detect and point out any danger situation. All these processes require highly computing-intensive operations. Due to this fact, a parallel real-time version of this system has been developed and implemented as described in [1]. In this parallel version, all processing stages are carried out at a 45-frames per second (fps) rate when applied to 502×288 small images or at a 15-fps rate when high-resolution 720×576 images are used. These data come from a rather basic prototype, but we envisage running these processes on a powerful up-to-date multi-core quad processor PC to achieve 30 fps for full HD $1,920 \times 1,080$ images.

*Correspondence: dopico@fi.upm.es
DATSI, Facultad de Informática, Universidad Politécnica de Madrid, Boadilla del Monte 28660, Spain

Apparent motion identified in this way is similar to apparent motion as estimated by a human driver. Therefore, potential dangers that could be identified by a driver who is aware of his activity can be identified by a system based on the techniques described in this paper.

Once OF has been computed, all similar optical flow vectors are grouped together, because they probably belong to be the same object. The complexity of the process is increased by the fact that the camera is also moving, because all objects in the image sequence seem to move away from a given point, the focus of expansion (FOE), and there is no static object that can be used as a reference. In fact, depending on the movement of the camera relative to the rest of the objects, three situations can be distinguished:

1. Static camera with moving objects: this is the simplest case since the images have a fixed background, which helps differentiate the moving objects more clearly. An example of this case could be the sequences captured by roadside traffic cameras.
2. Moving camera with static objects: this scenario presents more difficulties, since there is no static background to easily determine the moving objects. However, the knowledge that the objects in the scene are static makes it easier to establish reference points for movement. Examples of this case could be the landscape sequences in commercial movies.
3. Moving camera with moving objects: this is the special case addressed in this paper, in which both the camera and the objects are moving in the scene. It is the most complex case since there is no available reference. The camera is installed on board a vehicle traveling at a regular speed, ranging from a few kilometers per hour to 120 km/h.

In the sequences taken with the moving camera, all of the pixels seem to recede from a given point which is known as the FOE. The FOE is the point on the horizon at which the camera is aimed. It seems to be static and, therefore, does not generate optical flow. As the distance of the pixels from the FOE increases, they create a higher optical flow vector, i.e., if an object is near the edge of the image, it appears to move faster than if it is close to the FOE. Even the direction of the vector is affected, since all objects appear to radially move away from the FOE. This means that vectors from the same object could present different magnitudes or directions, making the task of grouping them highly complex.

Moreover, due to the camera movement, optical flow can be rather high even when all the objects in the scene are static. This optical flow can be determined and the pattern of movement established, but it is not uniform or constant because the vehicle speed and direction change considerably. Depending on the speed of the camera and on the direction of its motion, the apparent pixel movement or movement pattern is obtained, determining the velocity of a static object relative to the camera. Furthermore, the images are taken outdoors with variable light conditions, so shadows and reflections constantly appear and disappear as the car moves, interfering with the optical flow calculation.

In addition to calculating and segmenting the optical flow, the image is also segmented in search of the objects composing it. Although a specific kind of image sequences is used, restrictions have not been applied, as the aim is to obtain general solutions applicable to any kind of sequences. The system tries to locate moving objects and to assign them vectors with its velocity with respect to the camera, based solely on the video sequences taken from the inside of a moving car traveling along a road.

Two approaches were used in analyzing the images: (1) static, i.e., the size and shape of the objects are identified; and (2) dynamic, i.e., analyzing the motion in the sequence to obtain velocity vectors. In a subsequent phase, these partial results are combined so that the objects are mapped to velocity vectors as shown in Figure 1.

By combining the best of each approach, satisfactory results can be obtained, since only the edges of uniform objects generate optical flow. Consequently, correct shapes of uniform objects can not be obtained using only

Figure 1 Joining of static and dynamic results to identify image objects.

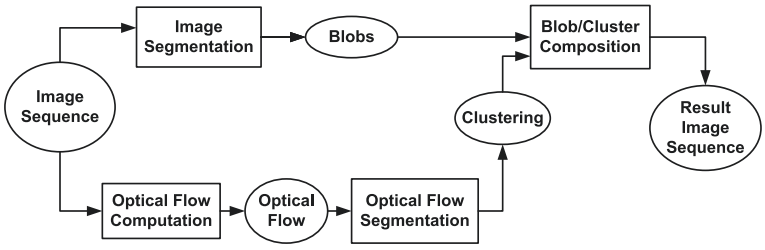

Figure 2 Image (static) and optical flow (dynamic) stages of the proposed algorithm.

velocity vectors. On the other hand, static images alone do not allow obtaining velocity vectors of moving objects.

The general approach followed is shown in Figure 2, in which the stages of the process are represented by rectangular shapes, and the results obtained by elliptical shapes. The images used for this work have been taken with a camera located inside the vehicle simulating the driver's point of view as shown in Figure 3.

2 Related work

This section briefly describes the basis on which the system analyzed in this paper is supported. As indicated earlier, the main novelty of our solution is the combined use of optical flow segmentation and raw image segmentation. Moreover, these processes should ideally be carried out in real time. Consequently, several of the application areas involved deserve to be placed in context, specifically, algorithms for optical flow computation and their parallelization, and segmentation of the optical flow results.

2.1 Algorithms for optical flow computation

There is a variety of algorithms to perform the computation of the optical flow. Most of them are based on the classical and well-established algorithms analyzed in [2], which usually have an initial premise for their correct operation, the assumption that the illumination intensity is constant along the analyzed sequence.

Figure 3 Scene camera inside the vehicle. The picture is taken from the driving seat, and the camera is located between the driver and passenger seats.

Each algorithm shows some advantages and disadvantages; the main drawback of most of the algorithms is their high computational and memory costs. Some of them try to reduce these costs by sacrificing accuracy of results, i.e., they balance the cost of the algorithm against the level of accuracy.

Over the years, a lot of research has been carried out in the field of optical flow algorithms. It has been continuously improved, sometimes by concentrating on the algorithm itself [3-6], sometimes by combining two of them [7,8], and sometimes by combining with other techniques [9-11].

Although most optical flow algorithms were designed with the main objective of obtaining accurate results, the trade-offs between efficiency and accuracy in optical flow algorithms are highlighted in [12] as well as the importance of an efficient optical flow computation in many real world applications. They also analyze several classical algorithms under both criteria. However, a search of the literature did not identify any previous studies or comparisons of the efficiency of recent algorithms or of their potential for parallelization or real-time capabilities.

2.2 Parallelization of optical flow

Over the last decades, the computation of optical flow has always posed a challenge in terms of processor computing power. Alternative algorithms, designed for implementation on computers with multiple processors, have been proposed since the first steps of development of this technique.

There have been many alternatives, and they have evolved along with the technology. In some cases, single-instruction multiple data (SIMD) processor arrays with specific chips, either existing [13] or designed *ad hoc* for the computation of optical flow [14], have been used. General-purpose MIMD as the connection machine [15,16], networks of transputers [17], or cellular neural networks [18,19] were also used in the past.

In recent years, there have also been many implementations based on field-programmable gate array (FPGA) [20-22] and graphic processor units (GPU) [23-25]. The results of a comparative study of both technologies for real-time optical flow computation are presented in [26]. They conclude that both have similar performance, although their FPGA implementation took much longer to develop. A very thorough comparison of both technologies applied to real-time vision computing is done by Pauwels et al. [27]. They examine several algorithms common in vision computing under many aspects, such as speed, cost, accuracy, power consumption, design time, and their general behavior on specific computations such as Gabor filtering, optical flow, and warping. Although the more adequate technology for each studied aspect is suggested, they conclude that the GPU surpasses the FPGA in most of their comparisons and agree with [26] about the FPGA requiring much developing time than the GPU. Some of the methods mentioned previously for computing optical flow are based on the Lucas-Kanade [28,29] method used in this paper or their application appears to be similar to that described in this paper.

A system for driving assistance is presented in [14]. It detects vehicles approaching from behind and alerts the driver when performing lane change maneuvers. The system is based on images taken by a camera located in the rear of a vehicle circulating through cities and highways, i.e., under the same hostile conditions as those in our system. However, their model is simpler because it is limited to detecting large objects near the camera and moving in the same direction and sense. Their method is based on the determination of the vanishing point of flow from the lane mark lines and calculating the optical flow along straight lines drawn from the vanishing point. The optical flow is computed by a block matching method using sum of absolute differences (SAD). The entire system is based on a special-purpose SIMD processor called IMAP-CAR implemented in a single CMOS chip that includes an array of 1×128 8-bit VLIW RISC processing elements. It processes 256×240 pixel images at 30 fps. Their experimental results show 98% detection of overtaking vehicles, with no false positives, during a 30-min session circulating on a motorway in wet weather.

Another implementation of the Lucas-Kanade algorithm is presented in [21]; this time, based on FPGA. Their method is based on the use of high-performance cameras that capture high-speed video streams, e.g., 90 fps. Using this technology, they are able to reduce the motion of objects in successive frames. Additionally, variations in light conditions are smaller due to the high frame rate, thus moving closer to meeting the constant illumination condition. In summary, a high fps rate allows simplifying the optical flow computation model and allows obtaining accurate results in real time. The division of the Lucas-Kanade algorithm into tasks is similar to that used in our method, although in [21], the pipeline is implemented by using specific and reconfigurable FPGA hardware (Virtex II XC2V6000-4 Xilinx FPGA; Xilinx, Inc., San Jose, CA, USA). Each pipeline stage is subdivided into simpler substages, resulting in over 70 substages using fixed-point arithmetic for the most part. The throughput achieved is one pixel per clock cycle. Their system is capable of processing up to 170 fps with 800×600 pixel images and, although its real time performance should be measured relative to the acquisition frame rate, it appears to be significantly high for the current state of technology.

In recent years, cluster computing technology has spread to the extent of becoming the basic platform for parallel computing. In fact, today, most powerful supercomputers are based on cluster computing [30]. However, it is unusual to find references to parallel optical flow algorithms designed to exploit the possibilities offered by clusters of processors to suit the size of the problems. In [31-34], some solutions are presented based on clusters.

In [32] and [1], we present a parallelization of the Lucas-Kanade algorithm applied to the computation of optical flow on video sequences taken from a moving vehicle in real traffic. These types of images present several sources of optical flow: road objects (lines, trees, houses, panels, etc.), other vehicles, and also highly variable light conditions. The method described is based on splitting the Lucas-Kanade algorithm into several tasks that must be processed sequentially, each one using a different number of subimages from the video sequence. These tasks are distributed among cluster nodes, balancing the load of their processors, and establishing a data pipeline through which the images flow. The method is implemented in three different infrastructures, (shared, distributed memory, and hybrid) to show its conceptual flexibility and scalability. A significant improvement in performance is obtained in all three cases. The paper presents experimental results using a cluster of 8 dual-processor nodes, obtaining throughput values of 45 fps with 502×288 pixel images and 15 fps with 720×576 pixel images, reaching speedups of 8.41. This is the parallel implementation of the Lucas-Kanade algorithm that we use in the segmentation system described later in this paper.

In [34], the optical flow calculation with three-dimensional images by an extension of the Horn-Schunck model to 3D is used. They study three different multigrid discretization schemes and compare them with the Gauss-Seidel method. Their experimental results show that under the conditions of their application, the multigrid method based on Galerkin discretization very significantly improves the results obtained using Gauss-Seidel. They also perform a parallelization of the algorithm aimed at its execution in clusters and apply it to the calculation

of 3D motion of the human heart using sequences of two $256 \times 256 \times 256$ and $512 \times 512 \times 512$ images taken by C-arm computed tomography. Their method is based on subdividing the image into several 3D subsets and processing each one in a different processor. The analyzed method is well suited to the proposed application, because the image just includes a single object (heart), with highly localized relative movements of expansion and contraction. This fact, along with the uniformity of illumination, requires a very low communication overhead due to parallelization. The speedup[a] using 8, 12, and 16 processors is excellent: 7.8, 11.52, and 15.21, with an efficiency close to one, but it starts to decrease when reaching 32 processors: 28.46. The experiments were performed on an eight-node quad-processor cluster.

2.3 Segmentation of optical flow

Many authors have used the optical flow as a starting point for the segmentation of moving objects in many applications under different scenarios such as robotics, collision avoidance, navigation, video coding, and driving assistance. Our system is devoted to helping with driver behavior analysis [35]. Optical flow provides apparent motion information, so it is usually combined with other techniques to obtain accurate and useful results.

An early approach for detecting moving object segmenting the optical flow generated by moving cameras is presented by Adiv in [36]. The first processing step partitions the flow field into connected segments of flow vectors that are consistent with a rigid motion of a roughly planar surface. Each segment is assumed to correspond to a portion of only one rigid object. In the second processing step, segments which are consistent with the same 3D motion parameters are combined as induced by one rigid object, either because of its real movement or because of the moving camera. The results of its method are analyzed using synthetic and real 128×128 video images, according to the technology of the time.

Thompsom and Pong [37] analyzed four techniques for detecting moving objects based on optical flow with the help of additional knowledge about camera motion or scene structure. Each one being suitable for a specific situation, they suggested that a reliable detecting method will require combining several techniques that were appropriately selected.

In [38], Choia and Kim addressed optical flow segmentation using a region growing technique, in which the motion constraints are relaxed, applying a hierarchy of motion models. They perform multi-stage processing to detect uniform subregions, according to simple motion models, which are grouped in uniform regions with respect to a complex model. In a first stage, the optical flow is segmented in subregions with similar motion vectors, called 2D translational patch. Next, 2D translational patches consistent with planar rigid motion are grouped in a 3D planar patch. In the last stage, the 3D planar patches are grouped into homogeneous regions generated by roughly parabolic rigid objects with 3D motion (parabolic patches). A pre-processing stage is performed to detect the static background combining null and near-null optical flow fields with a change detection technique.

In [39], Chung et al. combined region-based and boundary-based techniques on optical flow to perform spatially coherent object tracking. Their approach uses feed-forward inside frame-processing steps (region-based information to boundary-based computations) and feedback between subsequent frames. The region-based technique is based on gradient-based motion constraints and intensity-consistency constraints. The boundary-based technique is based on the distance-transform active contour improved by feed-forwarded intensity consistency data. The motion constraints within the contour are fed backed to be used as initial motion estimates in the region-based module for the next frame. They report its method as suitable for accurate segmentation in sequences with a moving background or camera and multiple moving objects, taking 1 to 2 s of processing time per frame using MATLAB (The MathWorks, Inc., Natick, MA, USA) on a 2.6-GHz dual Xeon computer.

In [40], Klappstein et al. studied the detection of moving objects as a part of a driver assistant system using either a monocular or a stereoscopic system that captured real-world vehicle image sequences. Their approach tries to detect and distinguish between the 'static motion' generated by the motion of the camera, usually known as ego-motion, and the 'dynamic objects motion'. It is based on tracking feature points in sequential images and estimating depth information from 3D reconstructed images. Their method consists of five processing stages, in which different techniques are performed according to the vision system in use. The optical flow computation is the starting point of both processing sequences, although an initial 3D reconstruction is also performed in the stereoscopic system. In the second step, the ego-motion is estimated based on optical flow and 3D information, which is necessary to estimate or enhance the 3D reconstruction prior to motion detection and final object segmentation. The 3D stereo reconstruction is enhanced by fusing the optical flow and stereo information using a Kalman filter. The detection of moving objects is based on individual tracked feature points. In the monocular system, this is done by looking for inconsistencies of feature points in the 3D reconstructed image, according to several constraints that must be satisfied by a static 3D point. In the stereoscopic system, this is done by analyzing the velocity of feature points. The segmentation stage is based on a globally optimal graph-cut algorithm. The stereoscopic system is reported to offer similar but more accurate results than

the monocular system; however, it suffers from a problem of decalibration of the stereo cameras and has higher computational costs.

In [41], Pauwels et al. tried to reproduce the processes carried out by the human brain while segmenting moving objects. They identify six interdependent feature extraction stages and propose a GPU-based architecture that emulate the processing tasks and the information flow in the dorsal visual stream. The extracted features are Gabor pyramid, binocular disparity, optical flow, edge structure, egomotion and ego-flow, and independent flow segments. They perform reliable motion analysis of real-world driving video sequences in real time achieving 30 fps with 320×256 pixels and 21 fps with 640×512 pixels.

In [42], Samija et al. addressed the segmentation of dynamic objects in 360° panoramic image sequences from an omnidirectional camera. They improve the segmentation results projecting the optical flow vectors geometrically on a sphere centered in the camera projection center. The camera is on a mobile robot where the movement on the horizontal plane is known; hence, the ego-motion is also known. Their approach is based on the differences between the estimated optical flow generated by two subsequent images and the expected optical flow computed by applying the ego-motion to the first image.

In [43], Namdev et al. combined motion potentials from optical flow and from geometry in an incremental motion segmentation system for a vision-based simultaneous localization of moving objects and mapping of the environment (SLAM). A dense tracking of features from optical flow results in dense tracks for which multiview geometric constraints are calculated with the help of the ego-motion supplied by the VSLAM module. Then, motion potentials due to geometry are calculated using the geometric constraints. The motion segmentation is performed by a graph-based clustering algorithm that processes a graph structure created using the geometric motion potentials along with the optical flow motion potentials. They show the results obtained from several private and public datasets. A standard laptop running MATLAB was used, taking up to 7 min of processing time for each frame, which was mainly due to the time required by the optical flow computation.

3 The Lucas-Kanade algorithm

The Lucas-Kanade algorithm [28,29] takes a digital video as the only data source and computes the optical flow for the corresponding image sequence. The result is a sequence of 2D arrays of optical flow vectors, each array associated to an image of the original sequence and each vector associated to an image pixel. The algorithm analyzes the sequence frame by frame and performs several tasks. In some cases, a task may require a certain number of images preceding and following the image being processed; therefore, the optical flow is not computed for some of the images at the beginning and at the end of the sequence.

The Lucas-Kanade algorithm computes the optical flow using a gradient-based approach, i.e., it calculates the spatio-temporal derivatives of intensity of the images. This method assumes that image intensity remains constant between the frames of the sequence, a common assumption in many algorithms:

$$I(x, y, t) = I(x + u\Delta t, y + v\Delta t, t + \Delta t) \tag{1}$$

This expression, using Taylor series and assuming differentiability, can be expressed by the motion constraint equation:

$$I_x u\delta t + I_y v\delta t + I_t \delta t = \mathcal{O}\left(u^2\delta t^2, v^2\delta t^2\right) \tag{2}$$

In a more compact form, taking δt as the time unit,

$$\nabla I(\mathbf{x}, t) \cdot \mathbf{v} + I_t(\mathbf{x}, t) = \mathcal{O}\left(\mathbf{v}^2\right) \tag{3}$$

where $\nabla I(\mathbf{x}, t)$ and $I_t(\mathbf{x}, t)$ represent the spatial gradient and temporal derivative of image brightness, respectively, and $\mathcal{O}\left(\mathbf{v}^2\right)$ indicates second order and above terms of the Taylor series expansion.

In this method, the image sequence is first convolved with a spatio-temporal Gaussian operator to eliminate noise and to smooth high contrasts that could lead to poor estimates of image derivatives. Then, following from the implementation in [2], the spatio-temporal derivatives I_x, I_y, and I_t are computed with a four-point central difference.

Finally, the two velocity components, $\mathbf{v} = \left(v_x, v_y\right)$, are obtained by a weighted least squares fit with local first-order constraints, assuming a constant model for $\mathbf{v}$ in each spatial neighborhood $\mathcal{N}$ and by minimizing

$$\sum_{x \in \mathcal{N}} \mathbf{W}^2(\mathbf{x}) \left[\nabla I(\mathbf{x}, t) \cdot \mathbf{v} + I_t(\mathbf{x}, t)\right]^2 \tag{4}$$

where $\mathbf{W}(\mathbf{x})$ denotes a window function that assigns more weight to the center. The resulting solution is

$$\mathbf{v} = (\mathbf{A}^T\mathbf{W}^2\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}^2\mathbf{b} \tag{5}$$

where for n points, $\mathbf{x}_i \in \mathcal{N}$ at a single time t

- $\mathbf{A} = \left[\nabla I(\mathbf{x}_1), \ldots, \nabla I(\mathbf{x}_n)\right]^T$
- $\mathbf{W} = \operatorname{diag}\left[\mathbf{W}(\mathbf{x}_1), \ldots, \mathbf{W}(\mathbf{x}_n)\right]$
- $\mathbf{b} = -\left(I_t(\mathbf{x}_1), \ldots, I_t(\mathbf{x}_n)\right)^T$

The product $\mathbf{A}^T\mathbf{W}^2\mathbf{A}$ is a 2×2 matrix given by

$$\mathbf{A}^T\mathbf{W}^2\mathbf{A} = \begin{bmatrix} \sum \mathbf{W}^2(\mathbf{x}) I_x^2(\mathbf{x}) & \sum \mathbf{W}^2(\mathbf{x}) I_x(\mathbf{x}) I_y(\mathbf{x}) \\ \sum \mathbf{W}^2(\mathbf{x}) I_y(\mathbf{x}) I_x(\mathbf{x}) & \sum \mathbf{W}^2(\mathbf{x}) I_y^2(\mathbf{x}) \end{bmatrix} \tag{6}$$

where all the sums used are points in the neighborhood $\mathcal{N}$.

3.1 Implementation

In this section, the implementation of the Lucas-Kanade algorithm proposed by Correia [44,45] is described because this implementation has been used to compute the optical flow prior to being segmented. All the parameters used in this section are obtained from the original sequential implementation by Correia presented in [44,45].

This implementation starts by smoothing the image sequence with a spatio-temporal Gaussian filter to attenuate temporal and spatial aliasing, as shown in [2]. It applies a smoothing Gaussian filter:

$$\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{x^2}{2\sigma^2}} \tag{7}$$

In this implementation σ is 3.2, and therefore, 25 pixels are required: the central one and 4σ [45] pixels at each side. This one-dimensional (1D) symmetrical Gaussian filter is applied three times, first on the temporal 't' dimension, then on the spatial 'X' dimension, and finally on the spatial 'Y' dimension.

The result of applying the Gaussian smoothing filter to an image can be seen in Figure 4, which shows the original image (Figure 4a) and three steps: the result for the temporal filter (Figure 4b), for the spatial filters (Figure 4c), and the final result (Figure 4d).

After smoothing, the next step of the Lucas-Kanade algorithm is to compute the spatio-temporal derivatives for the three dimensions: t, x, and y (I_t, I_x, I_y). Using the previously computed image, smoothed on t, X and Y, and applying a numerical approximation, the derivatives (I_t, I_x, I_y) are separately computed making use of the five-point central finite differences method, used to compute the first order of derivative with fourth order of accuracy on one-dimensional grid, based on central finite differences [46]:

$$f'(x_3) = \frac{f(x_1) - 8f(x_2) + 8f(x_4) - f(x_5)}{12h} \tag{8}$$

Taking $h = 1$ because the distance between two consecutive pixels is one, the one-dimensional array to be used as the convolution coefficient mask in the computation of the partial derivatives is obtained as follows:

$$\left[\frac{1}{12}, \frac{-8}{12}, 0, \frac{8}{12}, \frac{-1}{12}\right] \tag{9}$$

The results of the convolutions are the estimates of the partial derivatives, which are shown in Figure 5, and represent the temporal (Figure 5a), horizontal (Figure 5b), vertical (Figure 5c), and combined intensity changes (Figure 5d).

Finally, the velocity vectors associated to each pixel of the image are computed from the spatio-temporal partial derivatives previously computed. This is done by using a spatial neighborhood matrix of 5×5 pixels, centered on each pixel and a one-dimensional weight kernel with the following coefficients: (0.0625, 0.25, 0.375, 0.25, 0.0625)

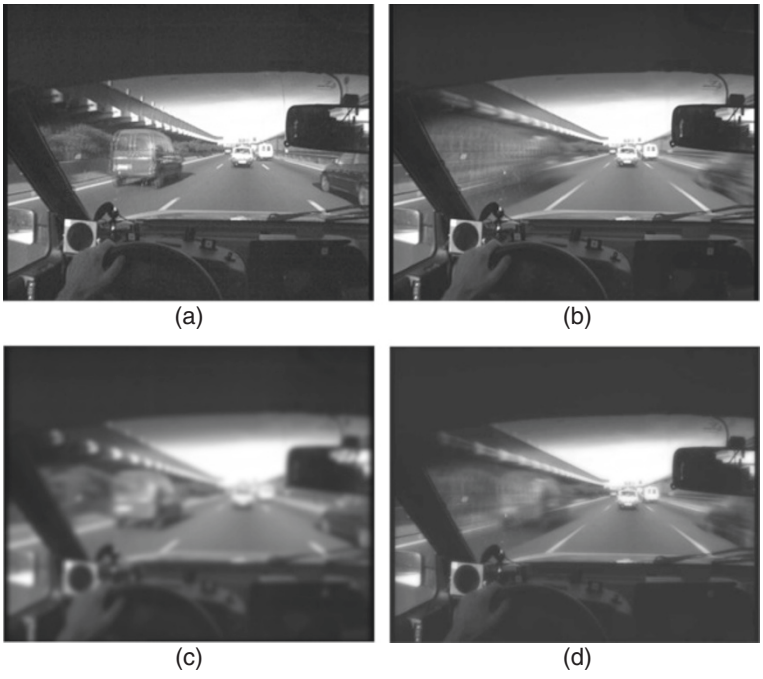

Figure 4 Image smoothing in the *t*, *x*, and *y* dimensions. (a) Original image. **(b)** Smoothing in *t*. **(c)** Smoothing in *x* and *y*. **(d)** Smoothing in *t*, *x*, and *y*.

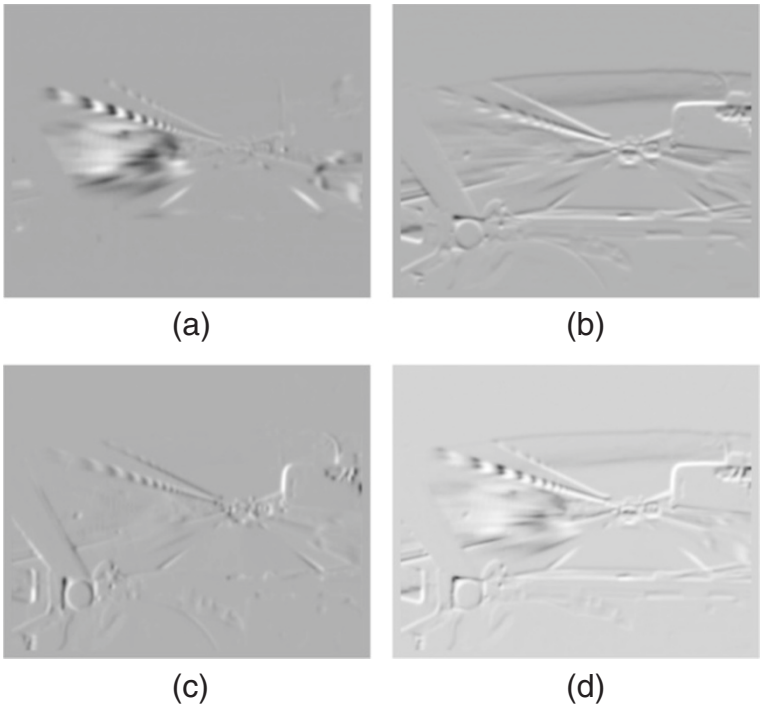

Figure 5 Partial derivatives of an image in the *t*, *x*, and *y* dimensions. (a) Derivative in *t*. **(b)** Derivative in *x*. **(c)** Derivative in *y*. **(d)** Derivatives in *t*, *x*, and y.

[2]. Noise parameters are $\sigma_1 = 0.08$, $\sigma_2 = 1.0$, and $\sigma_p = 2.0$ [47]. The estimated velocity vectors whose highest eigenvalue of $A^T W^2 A$ is less than 0.05 are considered unreliable (noise) and are discarded [2].

3.2 Results of the optical flow algorithm

Figure 6 shows the optical flow computed for the image of Figure 4a. The processing steps have been analyzed and are shown in Figures 4 and 5. The original image corresponds to a three-lane highway. The vehicle carrying the camera is overtaking the vehicle on the right while it is being overtaken (quite fast) by the vehicle on the left. This introduces some noise in the results, since it would require a higher temporal resolution to correctly handle the movement of objects at such speed.

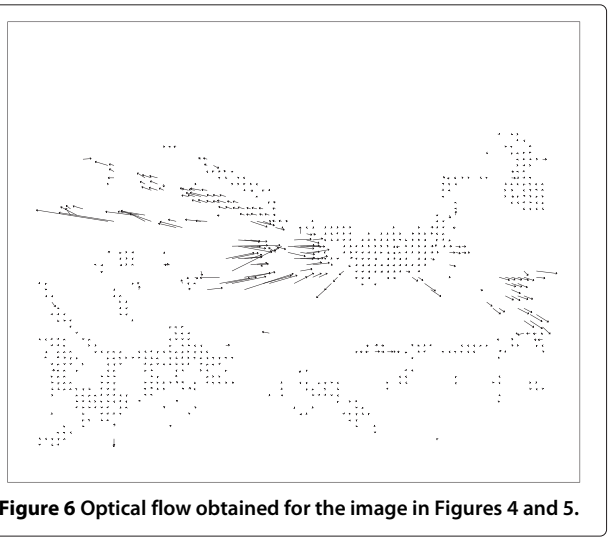

Figure 6 Optical flow obtained for the image in Figures 4 and 5.

Figure 7 shows two images of a video sequence that has been processed with this algorithm and also the corresponding optical flow. In this sequence, a vehicle can be observed on the right going slower than the vehicle carrying the camera, and a second vehicle on the left is changing lanes. Also visible is a traffic information panel on the upper-right corner of the image. The optical flow generated by these three objects, road markings, and other elements in the image is also shown in Figure 7.

4 Optical flow segmentation

Segmentation is performed once the optical flow has been calculated and assuming that a speed vector has been associated with each pixel of the image. During segmentation of the optical flow, nearby vectors with similar speeds are combined in clusters. A cluster is a group of vectors with similar properties which collects information on its position within the image, its area, the average vector,

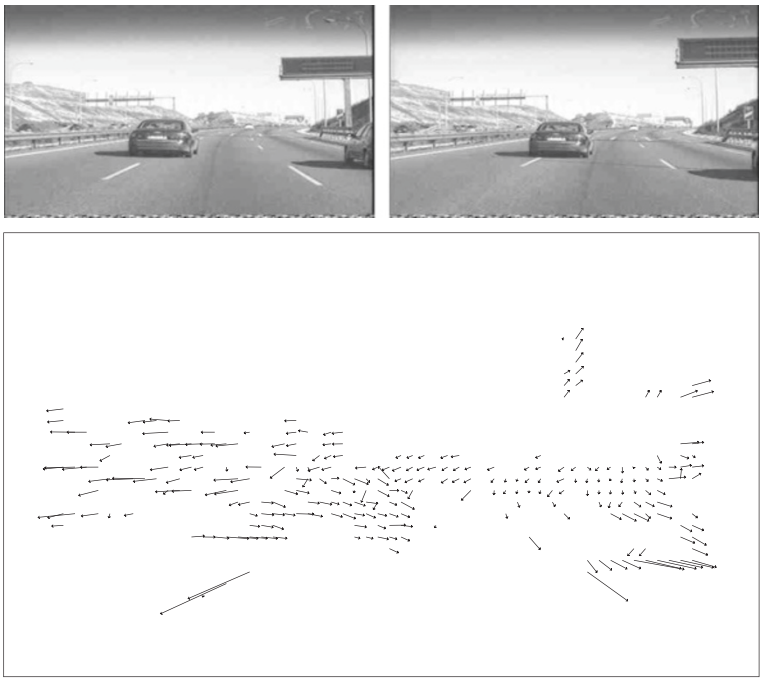

Figure 7 Frames for example session and optical flow. Frames 10 (upper left) and 20 (upper right) of an example session and the optical flow for frame 15 (bottom center).

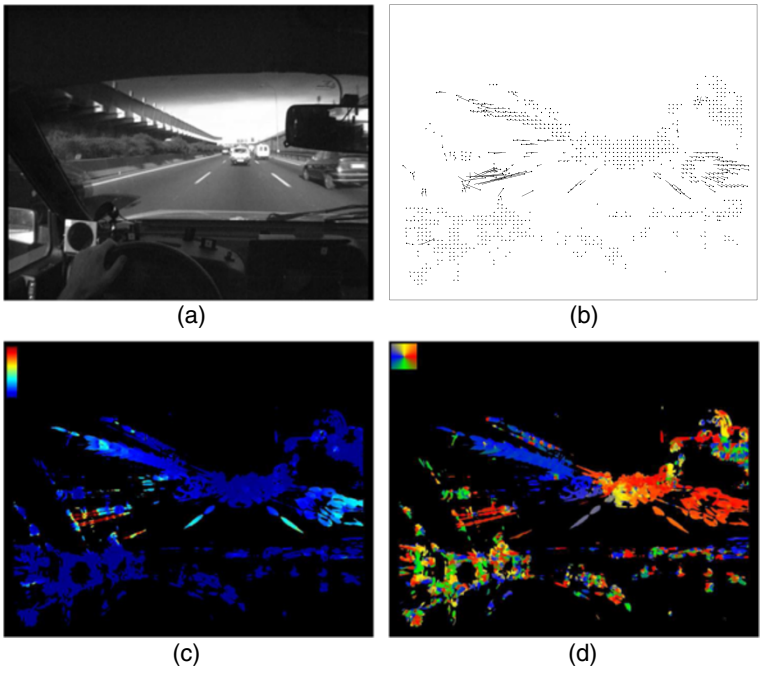

Figure 8 Modulus and the angle of the optical flow vector. Image from an example session **(a)** and its optical flow result **(b)**. Optical flow components **(c and d)**. **(a)** Image, **(b)** optical flow, **(c)** modulus, and **(d)** angle.

etc. The underlying assumption is that when a set of similar and closely grouped vectors is found, they should correspond to the same object.

The purpose of grouping similar vectors is the ability to assign a cluster to every object with independent movement and associate the cluster's average velocity vector to that object. This makes it easier to study the optical flow, since no multiple one-to-one vector comparisons are needed. Instead, representative vectors from different clusters are compared.

Similar, in this context, means similar in both magnitude and angle. Figure 8 shows the modulus and the angle of the optical flow vector, thereby illustrating the complexity of the problem. In Figure 8c, colors are applied according to the color bar in the upper-left corner, with modulus ranging from zero in black to its maximum value in red. In Figure 8d, the vector angle is colored according to the color circle in the upper-left corner, with blue, yellow, red, and green corresponding to the directions left, top, right, and bottom, respectively.

- Even within the same object, the vector modulus varies according to the distance from the camera, i.e., the closer a pixel, the greater the modulus. This is as expected, because considering similar velocity objects, motion perception increases as the object approaches the camera (i.e., as the depth Z coordinate decreases).
- Large differences can be found between different optical flow vector moduli. This is due to large actual differences in velocity between static objects and vehicles traveling in the same or in opposite directions.
- In general, the vector diverges by forming a conical shape, in such a way that its focus is the FOE, i.e., the imaginary point where the vectors originate and which the camera is focusing on. Consequently, static objects in the scene emerge from the FOE, and they move away towards the image edges.
- The closer a vector is to the image edges, the higher its modulus.
- Some very close vectors have rather different moduli and angles, as is the case with most of the vectors located inside the vehicle, i.e., close to the camera. These vectors represent noise and should be discarded.

4.1 Segmentation method implemented

The segmentation method implemented is based on an iterative algorithm operating on the optical flow vector matrix. This matrix consists of as many elements as there are pixels in the image matrix. Each of these elements is a vector which originates from a single pixel. Many matrix elements will be null due to the absence of an optical flow vector starting from the corresponding pixel. The algorithm tries to identify optical flow vector clusters by selecting those with similar properties and lying close to each other. To this end, the array of optical flow vectors is traversed from left to right and from top to bottom, i.e., in the storage order, so that for each non-zero vector, the distance - or similarity - to each of the previously identified clusters is obtained and the current vector is associated to the closest cluster. If none is found with similar characteristics, the current vector is used as the first element of a new cluster. To perform this task, the similarity function described below has been defined:

$$\text{similVect}\left(\overrightarrow{u}, \overrightarrow{v}\right).$$

4.1.1 Similarity of vectors

To calculate the distance or similarity between a vector and an optical flow vector cluster, every cluster is represented by its average vector. The similarity is then obtained by taking vector pairs, each pair being constituted by the vector candidate to be assigned to a cluster and the vector representing the average of each cluster previously created. Calculating the similarity between vector pairs is based on the three magnitudes described below. The general principle may be summarized as follows: the smaller the differences between these values, the shorter the distance between vectors (and the higher the similarity). They are as follows:

- Modulus difference.
  $m = \left|\left|\overrightarrow{u}\right| - \left|\overrightarrow{v}\right|\right| = \left|\sqrt{u_x^2 + u_y^2} - \sqrt{v_x^2 + v_y^2}\right|$
- Minimum angle between the two vectors, given by its scalar product.
  $\overrightarrow{u} \cdot \overrightarrow{v} = \left|\overrightarrow{u}\right| \cdot \left|\overrightarrow{v}\right| \cdot cos\alpha$ and hence
  $\alpha = \arccos\left(\frac{\overrightarrow{u}\cdot\overrightarrow{v}}{|\overrightarrow{u}|\cdot|\overrightarrow{v}|}\right) = \arccos\left(\frac{u_x\cdot v_x + u_y\cdot v_y}{\sqrt{u_x^2+u_y^2}\cdot\sqrt{v_x^2+v_y^2}}\right)$
- Euclidean distance between the positions of the two vector origins in the image. The weight of this magnitude in the calculation of similarity between vectors should ideally be small. In fact, this magnitude is considered as a rule to facilitate discarding vectors located far from the cluster.
  $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Given two vectors $\overrightarrow{u}$ and $\overrightarrow{v}$, the $\text{similVect}\left(\overrightarrow{u}, \overrightarrow{v}\right)$ function returns a scalar value that is used as a similarity measure. Like any other distance, it is zero when both vectors are identical and grows as they become less similar. A MaxGAP value is experimentally determined as the maximum return value for this function. Vectors are not considered as belonging to the same cluster if the similVect function returns the MaxGAP value.

$$\text{similVect}\left(\overrightarrow{u}, \overrightarrow{v}\right) = \begin{cases} \text{if} & (d > \text{thresholdDist} \vee \alpha > \text{thresholdAng} \vee m > \text{thresholdMag}) \\ \text{then} & = (\text{MaxGAP}) \\ \text{else} & = (\alpha \cdot k\text{Ang} + m \cdot k\text{Mag} + d \cdot k\text{Dist}) \end{cases}$$

4.1.2 Algorithm

The algorithm used for building the vector clusters is as follows:

1. The starting point is an optical flow vector matrix $m\text{OpticalFlow}(i, j)$ and an empty cluster list $l\text{Cluster} = \Phi$.
2. For each non-empty optical flow vector $\overrightarrow{v}$ belonging to the $m\text{OpticalFlow}(i, j)$ matrix, reading it from left to right (j) and from top to bottom (i), the following steps are performed:
   (a) The function similVect $\left(\overrightarrow{u}, \overrightarrow{v}\right)$ is called for each of the clusters of the list, so as to obtain the cluster which is most similar to the current vector. similVect is called with the following parameters:

   $\overrightarrow{v}$: current optical flow vector
   $l\text{Cluster}(k).\overrightarrow{\text{avgVector}}$: average vector of the cluster

   (b) If the previous step does not return any value smaller than MaxGAP, there are no matching clusters for the current vector. This may be because the vector is far from all clusters or due to the lack of correspondence between the two vectors in terms of their modulus or angle. In this event, a new cluster $\overrightarrow{v}$ is created containing only one vector. The new cluster is registered on the cluster list.
   (c) If similar clusters are found, the current vector is associated to the maximum similarity cluster, i.e., the cluster with the smallest return value in similVect. Once associated, the average vector and other characteristic properties of this cluster are recalculated.

To minimize the time spent on calculating the similarity function, the algorithm avoids calculating the distance of each vector $\overrightarrow{v}$ from all the other vectors within each cluster. Instead, a cluster bounding box containing all the vectors in the same cluster is defined and an initial filtering step performed. If the vector distance from any of the bounding box polygons is greater than a predefined value peakDist, then MaxGAP is returned.

Not only peakDist but also the maximum thresholds peakAng and peakMag must be defined. If the comparison returns values above these thresholds, the vectors are not considered similar. It is also necessary to determine the values of certain constant parameters to weight the angle (kAng), magnitude difference (kMag), and distance (kDist) between vectors.

Determining all these values formed part of an experimental process considering different kinds of video sequences. From these experiments, the values obtained were as follows: the maximum distance or peakDist was fixed as 10 pixels. This means that optical flow vectors belonging to the same cluster are separated less than 10 pixels. A higher value for this parameter would lead to interpret several clusters as being the same. A lower value for the parameter would artificially lead to a cluster being interpreted as several independent clusters.

The maximum value for the angle differences peakAng was set to $\frac{\Pi}{3} = 60°$. Consequently, no optical flow vector can be associated to a cluster if its angle difference from the average cluster vector is greater than 60°. This peak value may seem to be excessive, but it is derived from the actual use of a moving camera. The camera movement can generate optical flow vectors with fast changing directions, especially if the observed objects are close to the optical axis of the camera.

The peak difference in terms of magnitude, peakMag, is also experimentally set at a value of 2. This is justified by the small differences observed when comparing the optical flow vectors generated from different pixels of the same object.

Finally, the weights used for the expression in Section 4.1.1 have been assigned the following values: $k\text{Ang} = 0.8$, $k\text{Mag} = 0.15$, and $k\text{Dist} = 0.05$. This means that a much greater importance is assigned to the difference between angles, while the magnitudes are considered significantly less important and the distances within the image almost negligible. The overall similarity measure is thus established in the system as $\alpha \cdot 0.8 + m \cdot 0.15 + d \cdot 0.05$.

4.1.3 Filtering

Once all of the optical flow vectors present in the original matrix are grouped into clusters, the classification using the algorithm can be considered completed. However, using this approach, without a post-filtering stage can generate many clusters that are irrelevant to the study of stationary or moving objects in the field of view of the moving camera. These clusters are usually derived from image noise, caused by shadows or differences in lighting. This noise can also be due to the detection of almost

completely static clusters, i.e., clusters moving with the same speed and direction as the camera used, so presumably are part of the vehicle itself, such as the dashboard and mirror.

Based on these considerations, the segmentation method includes a final step for cluster filtering. In this step, clusters consisting of a small number of vectors are interpreted as noise and consequently removed. Those clusters whose average vector modulus is less than a preset value are also removed, as they are interpreted as being part of the vehicle in which the camera is installed.

Figure 9 shows the result of applying cluster filtering with the following parameters:

- minimum number of vectors in a cluster = 180
- minimum value for the modulus of the average vector = 0.2

Figure 9a shows the result prior to filtering, in which 2,149 clusters are identified. However, a significant proportion of irrelevant clusters can be observed. By contrast, Figure 9b shows the 31 clusters obtained after filtering.

4.2 Optical flow segmentation results

This section presents the results of applying the optical flow segmentation method to multiple images from different video sequences, all of them taken with the camera installed on board a moving car. Ideally, the best result would be a single cluster for each object in the scene, regardless of its shape or whether the cluster completely covers the object. This is because the geometrical shape of the object is derived from the segmentation of the image itself, not from the segmentation of its optical flow. But, this is not always possible, and several clusters for each object with slight variations in direction or in the modulus are usually found. Furthermore, since the camera is in motion, the same single object can provide optical flow vectors with different directions, depending on their relative positions with respect to the FOE. For example, on a straight road segment, a traffic sign in front of the camera can take up the whole width of the scene, generating optical flow in all directions, though always moving away from the FOE.

4.2.1 Highway with traffic in both directions

In this sequence, the vehicle is driving along a four-lane highway in which other vehicles driving in both directions can be seen. For vehicles moving in the same direction, the closest vehicle to the camera is slower than the rest, so it is overtaken in the initial images of the sequence. The remaining vehicles move at a very similar pace as the vehicle with the camera. For vehicles moving in the opposite direction, a large truck and several small vehicles are approaching the camera. The result of the segmentation of the optical flow for this sequence is shown in Figure 10.

In this experiment, clusters are detected for the oncoming truck and for the vehicle moving in the same direction. Clusters can not be obtained for the remaining vehicles, or only intermittently. However, some clusters appear in some frames, such as for the vehicles in the background which are driving in the same direction as the camera. For these vehicles, a cluster has been found as shown in Figure 10a. This is mainly because the vehicle group moves at a similar speed to the vehicle with the camera, so perceived motion is negligible. In the last frame, the vehicles behind the truck and driving in the opposite direction are detected. This is because their distance from the camera is shorter than those in the previous frames. However, the vehicle group is treated as a single cluster. Furthermore, certain clusters can be discontinuously observed near the edges of the windshield throughout the sequence. These clusters are clearly related to noise and are removed during the cluster-object aggregation stage as they are usually clearly separate from any objects.

4.2.2 Highway with traffic in both directions (2)

Once again, the vehicle is driving along a four-lane highway. However, this case is characterized by the presence

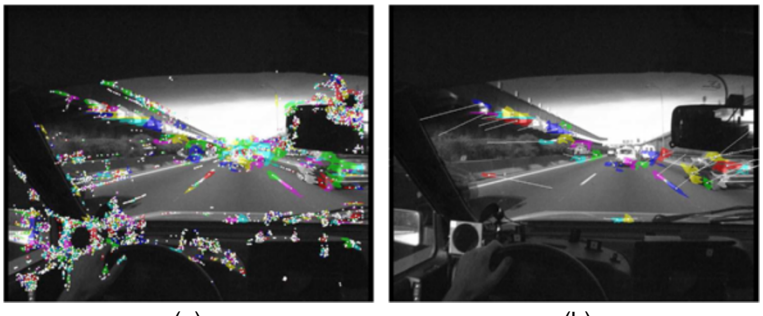

Figure 9 Filtering stage to eliminate clusters consisting of a small number of vectors. (a) Clusters found without filtering. **(b)** Clusters after filtering.

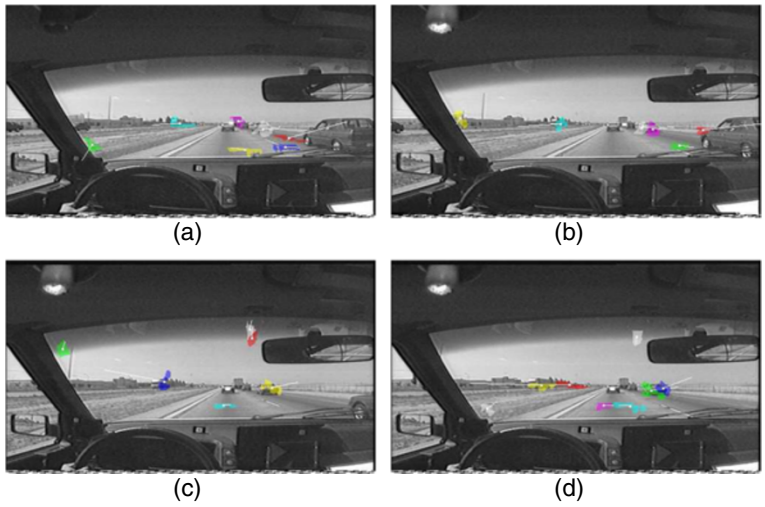

Figure 10 Optical flow segmentation of the first highway sequence example. (a) Image 1. **(b)** Image 2. **(c)** Image 3. **(d)** Image 4.

of a truck at close distance filling a great proportion of the image. Both sides of the road are lined with trees, and a bridge can be seen at a distance.

The bridge over the road can help us examine the impact of a large object on the optical flow sequence. The result of the optical flow segmentation for this sequence is shown in Figure 11.

The first thing to note is that, as expected, the truck is not interpreted as a single cluster, instead there are several, up to eight in the first image. This is mainly because the truck consists of several parts, and each one generates a slightly different optical flow vector, especially in terms of its modulus. Moreover, given that optical flow vectors are obtained only for the contours (or, more specifically, for changes in the image intensity) of the parts that comprise the truck and the differences in the direction of the optical flow vectors due to camera movement, it is reasonable to obtain several clusters for a single large vehicle.

Distant vehicles moving in the same direction as the camera do not generate any cluster, precisely due to their distance, their low relative speeds, and their small apparent size. For vehicles approaching in the opposite direction, clusters are continuously obtained throughout the sequence, sometimes resulting in misinterpretations of the bridge over the road. However, this should not cause any problem once the clusters are combined with the objects.

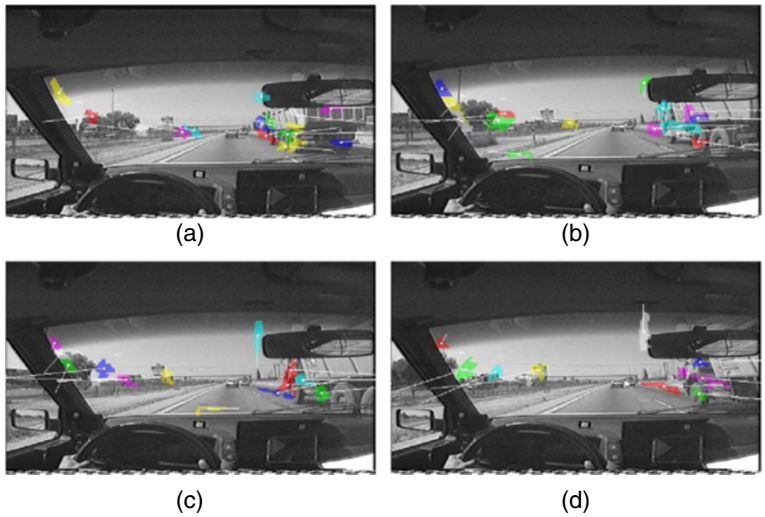

Figure 11 Optical flow segmentation of the second highway sequence example. (a) Image 1. **(b)** Image 2. **(c)** Image 3. **(d)** Image 4.

Finally, it can be noted that some clusters are obtained near the windshield edges of the vehicle in which the camera is installed. These clusters clearly represent noise due to the high contrast between the vehicle and the road. There are also clusters obtained for the trees located on one side of the road. However, this is a correct detection because even though they are static, they present an apparent motion with respect to the camera and, hence, generate optical flow.

5 Image segmentation

In addition to segmenting the optical flow, the images are also segmented since the optical flow does not preserve the shape and size of objects, so this information has to be retrieved from the original images. An edge-based segmentation scheme using a modified Canny algorithm [48] has been chosen with the following characteristics:

- It minimizes edge detection errors: this is important to avoid false detections.
- Good edge location: minimizes distance between the detected edge and the actual edge.
- A single answer for each edge: the transition that conforms the edge may be large, but this scheme seeks the maximum gradient to render a clearly defined edge.

The segmentation scheme consists of six steps, as shown in Figure 12. It takes the source image as input and generates a list of blobs as output. A blob represents a uniform region of the image, containing information such as its size in pixels, the smallest rectangle containing it, and its average intensity. Segmentation is divided into the following steps:

- Smoothing: the image is smoothed using a Gaussian filter to remove noise.
- Edge extraction: a Sobel operator was chosen because it has low computational cost and low noise sensitivity (very important for the type of images used).

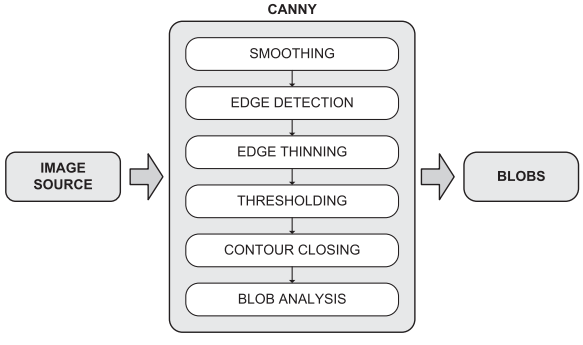

Figure 12 Canny scheme used in this system for segmenting images.

- Thinning edges: the edges obtained have to be homogenized by thinning them to a pixel's width, choosing only those with maximum intensity gradient.
- Thresholding: in this step, the image is binarized, and most of the edges generated by noise are eliminated (although a few real edges are also eliminated). A segmentation algorithm based on the k-means with $k = 2$ is used. This algorithm, unlike the Bayesian ones, does not need *a priori* information of the classes, it only needs the number of classes (k) into which to divide the histogram (bottom and edges).
- Closing contours: this avoids discontinuities in the edges and facilitates subsequent detection of objects. The Deriche and Cocquerez algorithm [49] is used, which is based on the assumption that an open-edge section can be closed following the direction of maximum gradient. However, the search for the next edge pixel is limited so that the maximum angle between two consecutive edge pixels is 45°.
- Blob analysis: this last step identifies the objects. The following recursive algorithm is used:

1. It starts from the binarized image and a mask for each pixel indicating whether it has been processed (initially, the mask values for the pixels are all false).
2. The image is explored from left to right and top to bottom.
   (a) If the pixel has not been processed and does not correspond to an edge, a blob is created, initialized, and appended to the list of blobs.
   (b) A recursive search is started from the pixel indicated.
      (i) The pixel is marked as processed and added to the blob.
      (ii) The blob parameters are updated, i.e., number of pixels, maximum and minimum coordinates, center of gravity, etc.
      (iii) For each unprocessed pixel in the neighborhood that is not an edge, the search is repeated recursively, i.e., it returns to step (b).
3. The result is a list of blobs.

Filtering the blobs based on their size is the next step. Any blobs above or below certain size limits are discarded, since small blobs are associated with noise, and very large blobs are associated with background details. As a consequence, some quite far away

vehicles can be discarded, but due to their distance, they are not relevant for the driver.

5.1 Results of image segmentation

This section presents the results of applying the algorithm to real images in different environments. The aim is to obtain at least one blob for each vehicle in the image. If a vehicle gives rise to several blobs, the latter are merged at the stage that combines optical flow with blobs since both present similar velocity vectors and are located in close proximity. Undesired blobs (for example, inside the car, in the sky, or on the sides of the road) do not present a problem as many of these are later removed for having no associated velocity vector.

5.1.1 Highway with two-way traffic

This sequence shows four vehicles traveling in the same direction as the car camera (our vehicle) and an oncoming truck. One of the vehicles (furthest to the right) travels in the same direction at a medium distance, and the rest are far away. Figure 13 shows the original image (a), the result after applying the smoothing filter (b), the edges once the contours have been closed (c), and finally, the blobs obtained after the search for connected components and the required filtering have been performed (d).

A single blob is obtained for the truck coming from the opposite direction, containing everything but the cabin. In the same lane, another blob is obtained that merges various vehicles and some of the trees in the area. Three blobs are obtained in our lane, one for the nearest vehicle, one for the intermediate one, and one that merges a truck and another vehicle that are close together. Another blob is also obtained on the road, which merges part of the shoulder with a white line. There are also many blobs found inside the car and to the left of the mirror, but they are clearly invalid and must be removed.

5.1.2 Highway with heavy traffic in one direction

This sequence presents higher traffic density at different distances as well as diverse vegetation on the sides, and a bridge in the background, than the sequence described in Section 5.1.1. It is not an easy sequence to segment, although it also has the advantage that the vehicles closer to the camera have colors that strongly contrast with the background, making their contours easier to identify.

The segmentation result is shown in Figure 14, which shows that at least one blob has been obtained for each of the nearby vehicles, but in some cases, up to three blobs have been found. A blob has been obtained corresponding to a portion of the road's left shoulder, another for a house, and another for vegetation on the right side. As in the previous sequence, blobs have been obtained inside the vehicle and in the rearview mirror, which must be removed.

6 Combination of clusters and blobs

To obtain the final results, the partial results obtained from both the segmentation of the optical flow (clusters) and the image segmentation (blobs) need to be merged, assigning one, several or no blob to each cluster. The idea is to use the clusters and blobs computed in the previous steps to obtain sets of pixels with an associated velocity vector. Each set of pixels identifies an object with apparent movement. Each blob determines the shape, position, and size of the object, and the cluster represents its dynamic

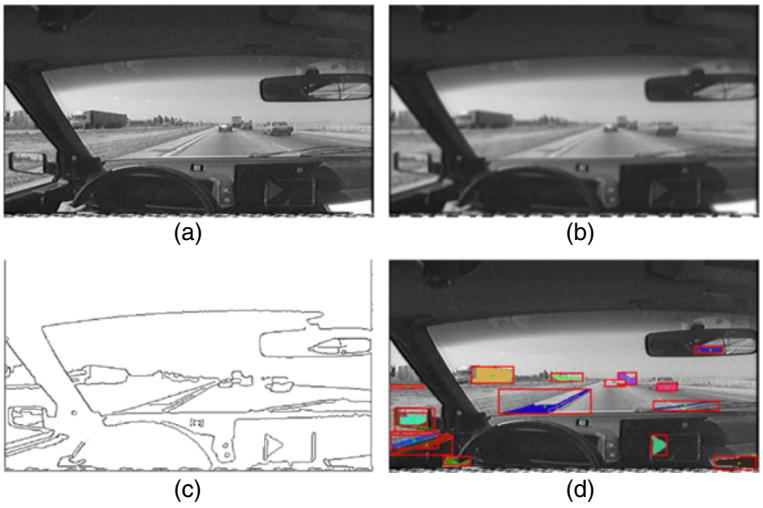

Figure 13 Simple example of an image (static) segmentation from a highway sequence. (a) Original image. **(b)** Smoothed image. **(c)** Contours closed. **(d)** Blobs.

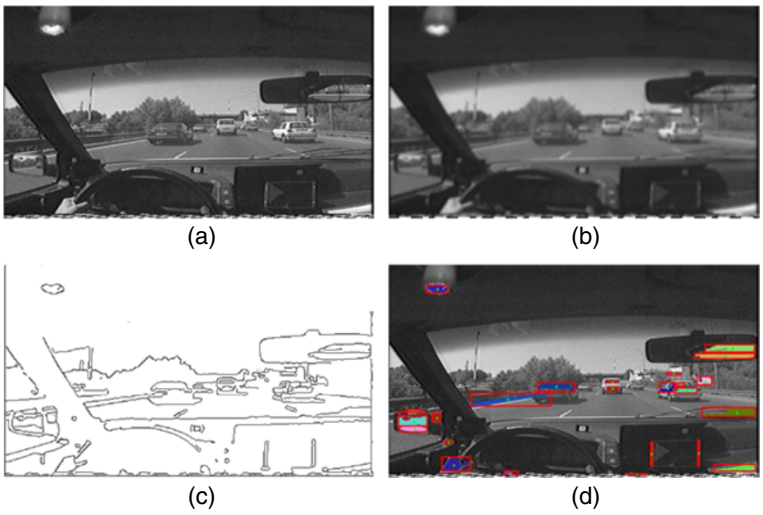

Figure 14 Complex example of an image (static) segmentation from a highway sequence. (a) Original image. **(b)** Smoothed image. **(c)** Contours closed. **(d)** Blobs.

properties, i.e., the velocity vector. However, the association between clusters and blobs is not trivial, since a single object can consist several blobs and/or several clusters.

The strategy used to calculate the optical flow clusters of an image is based on obtaining the smallest possible number of clusters, and if possible, only for moving objects, without considering their shape or size, because the optical flow does not preserve these properties. However, the strategy for the computation of blobs is completely different; the idea in this case is to obtain potential blobs in the image, in such a way that each group of pixels with relatively homogeneous values should result in a blob. The resulting large number of blobs does not pose a problem since any blobs clearly separated from any cluster are subsequently discarded as image background. As clusters necessarily need to be close to an object, any cluster without at least one associated blob is interpreted as erroneous and discarded. Therefore, according to the strategy used for the computation of blobs and clusters, clusters lead the combined process, as more importance and reliability is assigned to them than to blobs. As the search is oriented to finding objects in motion and this information is provided by the optical flow, the focus is on the clusters as they are generated by the optical flow segmentation.

For the final results, the shape, position, and size of the objects are derived from the blobs and only the velocity vector from the clusters. Cluster positions are discarded because the pixels inside uniform objects do not generate optical flow at all. Only their edges cause optical flow; therefore, object shapes are not preserved. A cluster can be associated to more than one blob and is associated mainly by the size of their intersection area or by their proximity. For reasons of efficiency, the intersection area is not computed by using the actual pixels of the blob and the cluster, but by an estimate based on the minimum rectangle containing both.

The algorithm works through the full blob list, one at a time, trying to associate the best possible cluster to each blob, and if no suitable cluster is found, the blob is discarded. Next, all the blobs associated with the same cluster are grouped together in a single blob, recomputing its mass center where the cluster velocity vector will be placed. Finally, clusters with no associated blob are discarded, too. To select the most appropriate cluster to be associated with a blob, the following cases must be considered:

1. The blob intersects with exactly one cluster: if the rectangle overlying blob B intersects with the rectangle of a single cluster C, the blob is associated to this cluster, as shown in Figure 15, where I is the intersection rectangle. The result, shown in Figure 15, has the shape of the blob and the velocity

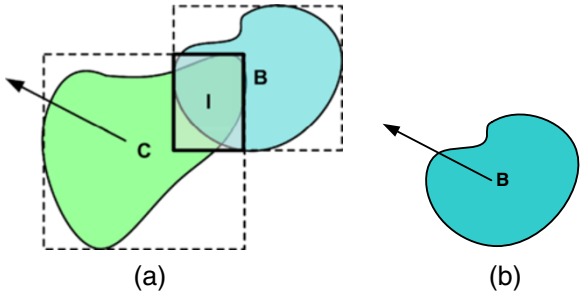

Figure 15 Combination of segmentation results: intersection of a unique blob with one cluster. (a) Intersection of blob and cluster. **(b)** Result.

vector of the cluster starting from the geometric center of the blob.

2. The blob intersects with several clusters: if the rectangle overlying blob B intersects with more than one cluster, as shown in Figure 16 in which blob B intersects with clusters C_1 and C_2, then the blob is associated to the cluster with the largest intersection area. In the example shown in Figure 16, area I_1 is greater than I_2; therefore, blob B is associated to cluster C_1.
3. The blob does not intersect with any cluster: when the rectangle covering blob B does not intersect with any cluster, the nearest cluster is considered. If the distance between the blob and cluster does not exceed a certain threshold, blob B is associated to this cluster; otherwise, the blob is discarded and not associated to any cluster. The example in Figure 17 shows blob B not intersecting with any cluster, but being close to clusters C_1 and C_2, so the distances to both clusters d_1 and d_2 are calculated. As the minimum distance is d_2 and does not exceed the maximum threshold, the blob is associated to cluster C_2, as shown in Figure 17.

This algorithm associates one or more blobs to each cluster, because the usual case is a moving vehicle generating a single cluster but with many blobs, due to the diversity of its component parts (wheels, windows, body, etc.). All of the parts generate a similar optical flow but different blobs. This algorithm is very efficient as the computation of a rectangle intersection is almost trivial, the result either being void or another rectangle.

6.1 Results of associating clusters and blobs

Figure 18 is an example of the association of clusters and blobs. This figure shows five vehicles traveling in the same direction as the vehicle carrying the camera, but driving at a slower speed. One of them is quite close, and the four others quite far from the camera. Figure 18a shows the blobs found, one obtained for the closest car, and only

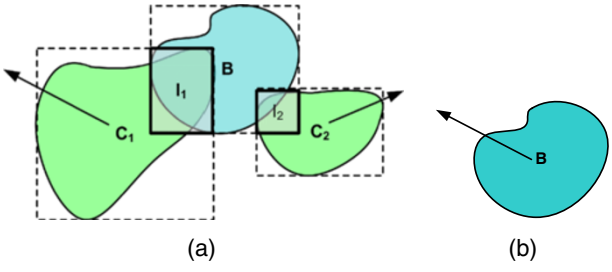

Figure 16 Combination of segmentation results: intersection of a unique blob with more than one cluster. (a) Intersection of blob and cluster. **(b)** Result.

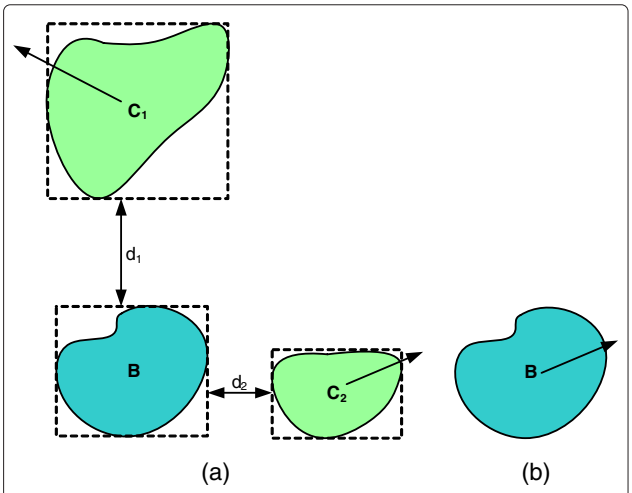

Figure 17 Combination of image and optical flow segmentation results if there is no blob-cluster intersection. (a) Intersection of blob and cluster. **(b)** Result.

three more for the four remote vehicles, because they are quite close together. Two blobs are also found in the central reservation. Figure 18b shows the clusters: one near the first car and two more related to the remote vehicles. Furthermore, a cluster is also found in the middle of the road. After associating (Figure 18c), it can be seen that all the blobs related to the distant vehicles have been grouped and associated to a single cluster, the one closest, while discarding the other cluster in spite of its proximity. Another successful association was obtained for the close car, and finally, the blobs and cluster related to the road were removed as there is no association between them due to their relative distances.

Figure 19 shows a more complex example. In addition to the vehicles present on the road, there is vegetation on the sides and a bridge in the background. There is a truck driving very close to the vehicle with the camera in the same direction, but at a slower speed, and two vehicles in the background with a relative speed close to zero. In the opposite direction, some small approaching vehicles can be observed, with similar textures as the background objects: central reservation, vegetation, or bridge.

The results after image segmentation, shown in Figure 19a, consist of seven blobs related to the truck due to its large size and the diversity of its component parts. There is also another blob for the bridge, one more for a truck close to one side of the bridge piers, and finally, one more for the central reservation.

Regarding the results of the optical flow segmentation, shown in Figure 19b, eleven clusters have been found. There are four clusters related to the truck: one for the pier on the right of the bridge, one for the car traveling in the same direction, two for the pier on the left of the bridge

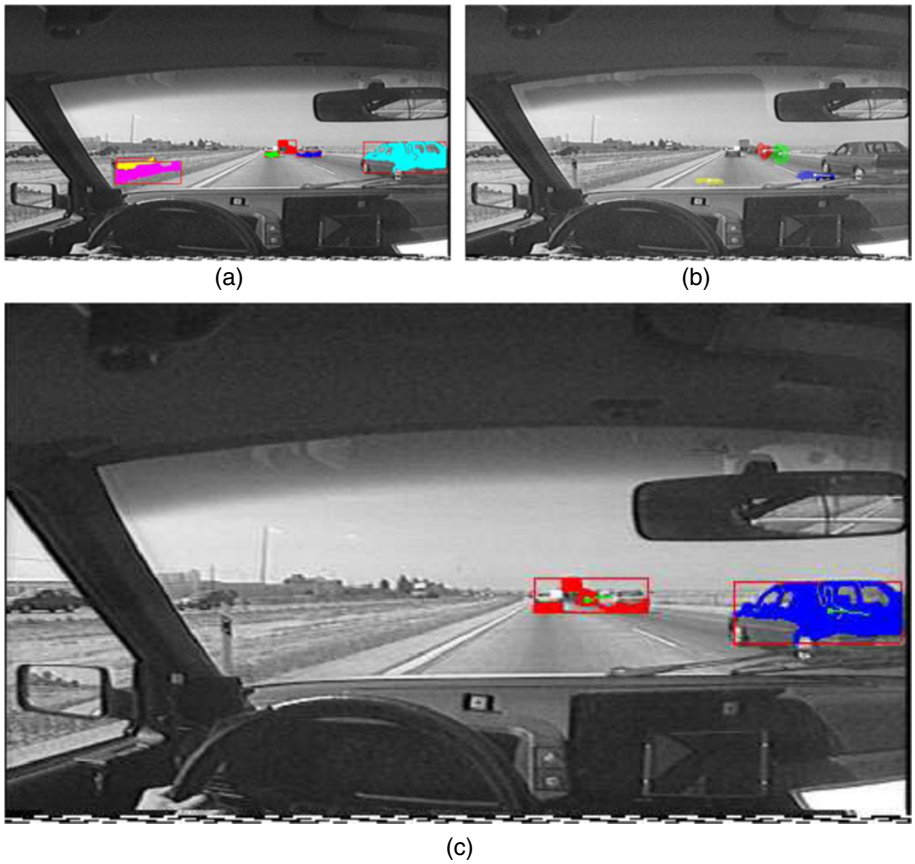

Figure 18 Simple example of blob-cluster associations. (a) Blobs. **(b)** Clusters. **(c)** Association results.

(mixed with one of the trucks circulating in the opposite direction), two for a tree on the left, and finally, a cluster due to noise next to the windshield of the camera vehicle.

The association of clusters and blobs, Figure 19c, shows the truck perfectly identified by two objects, obtained by grouping several blobs together. Also, the distant bridge is identified but with a small velocity vector due to its distance. Regarding the opposite lanes, one object has been located near the left pier of the bridge, but it is mixed with an object from the truck due to their similar distance and texture. Finally, one blob in the central reservation has been associated with a cluster generated by a vehicle traveling in the opposite direction, leading to a false result. This is mainly due to the absence of blobs for these vehicles as they have a very small size and are very similar to the background texture.

7 Results

This section looks at the segmentation of multiple image sequences, in which all process steps are considered: image segmentation, blob computation, optical flow computation followed by segmentation, cluster recognition, and finally, cluster-blob association. The process followed is described below:

1. Starting from the original image, the segmentation process described in Section 5 is applied, obtaining a list of blobs in an XML file.
2. Optical flow is generated for the image sequence using the Lucas-Kanade algorithm.
3. Optical flow segmentation is applied to obtain a list of clusters in an XML file.
4. Blobs and clusters created in the previous steps are associated.

The only input data is the image sequences to be analyzed. Several test sequences, around 50 images each, have been recorded with a 25 interleaved frames per second camera at 720×576 pixels. Therefore, every 40 ms, a complete image frame is generated, after combining two image fields provided by the interleaved camera at 20-ms intervals. The main problem of using an interleaving camera derives from the delay in capturing both image fields, 20 ms, causing a mismatch in the final image. For example, a car traveling at 90 km/h (25 m/s) will advance

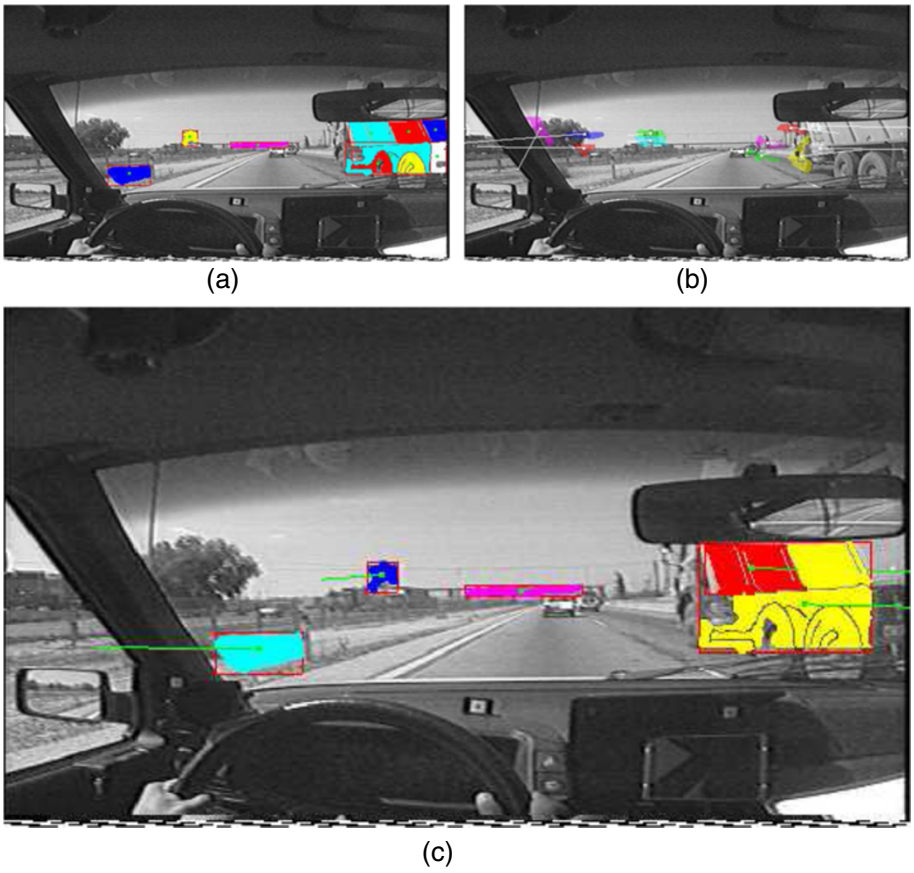

Figure 19 Complex example of blob-cluster associations. (a) Blobs. **(b)** Clusters. **(c)** Association results.

50 cm in 20 ms. One solution to fix this problem is to use only the even or the odd image fields and to ignore the other half of the original image, or to use a more expensive full-frame camera. The results in this section analyze both 720×576 interleaved and 720×288 non-interleaved images.

The camera is placed inside the experiment vehicle, and the images acquired include the road but also part of the experimental vehicle such as the dashboard, steering wheel, and roof areas. To simplify the localization process, the search will only consider the area of the image showing the road.

7.1 Two-way road

This sequence shows a two-way road with heavy traffic including different sized vehicles, speeds, and distances from the vehicle camera. In the image shown in Figure 20a, a group of distant vehicles and a truck traveling in the opposite direction can be seen. The result of the image segmentation and blob searching algorithm is shown in Figure 20b.

Vehicles traveling in the same direction as the camera are grouped in two blobs, one which comprises a single vehicle and another grouping of two vehicles. The truck moving in the opposite direction is converted into one blob, but including some parts of the horizon because of its similar color. In this case, two more blobs appear, but they are considered noise: the first one is close to the truck, and the other represents the right road shoulder.

In the optical flow segmentation shown in Figure 21, several clusters are obtained for the vehicles traveling in the same direction as the camera and for the oncoming truck. Other clusters such as those near the edge of the windshield as well as the one corresponding to the central reservation are considered as noise.

The association of the partial results obtained in the previous steps shown in Figure 22 removes all noise clusters and blobs. The objects are correctly identified for vehicles traveling in the same direction as the experimental vehicle. However, some are grouped, because they are far away and the images acquired do not allow distinguishing them clearly. Furthermore, an object is obtained for the oncoming truck with an additional blob due to noise because both objects are very close. Finally, the results are considered satisfactory, because every moving vehicle is

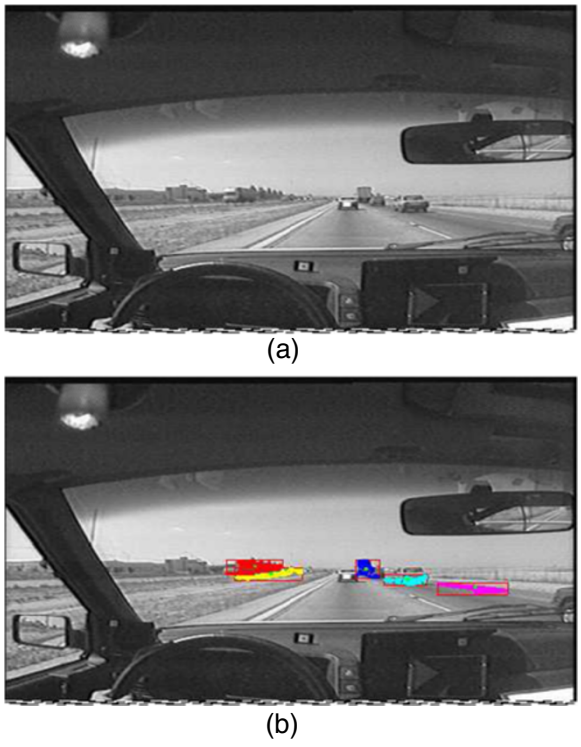

Figure 20 Example of image segmentation results from a two-way road sequence. (a) Original image. **(b)** Blobs found.

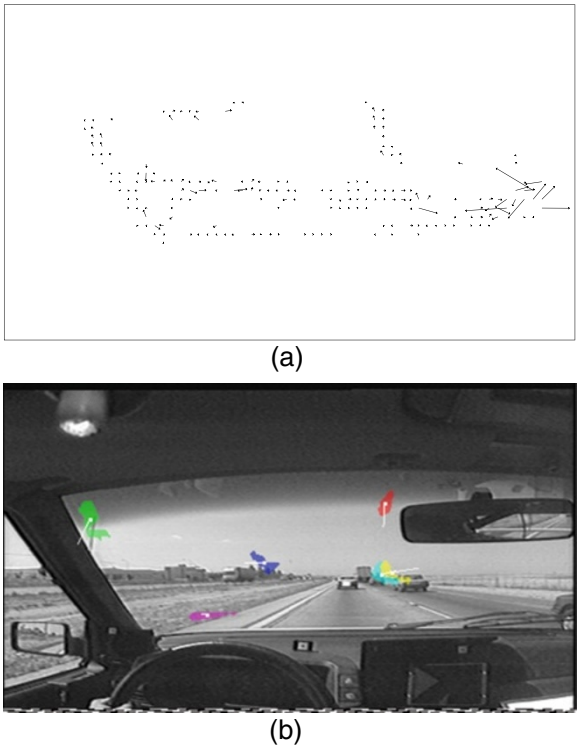

Figure 21 Optical flow segmentation results for the frame shown in Figure 20. (a) Optical flow. **(b)** Clusters.

identified and the noise objects obtained in intermediate steps are removed.

7.2 One-way highway

This example shows a one-way highway image in which the vehicle with the camera is traveling faster than the other vehicles. As shown in Figure 23a, there is a car close to the right of the experimental vehicle preceded by a small van, and there is another vehicle in front of the experimental car driving on the same lane. All three vehicles are approaching the experimental vehicle due to their lower speeds. An added difficulty compared to the previous example is that the road sides are lined by a wall on the right and a kind of overhanging roof on the left.

In the image segmentation result shown in Figure 23b, several blobs are obtained for the car on the right side, because of its large size. This effect does not cause a problem, because a subsequent step will group all of the blobs belonging to the same object and associate them to a cluster. The other vehicles are correctly located: a blob has been assigned for each one. Finally, some other blobs appear on the roof structure and around the road lines.

Figure 24b shows the optical flow segmentation in which the right-hand car is identified by two clusters, because of its large size and the multiple parts composing it. Two distant vehicles are detected with a cluster assigned to each one. Finally, several clusters are obtained for the roof on the left side of the road, and even though they do not represent actual movement, they do show apparent motion, their detection thus being correct.

Figure 25 shows the final result after associating the clusters and blobs obtained in the previous steps. Two objects have been obtained for the car close on the right by grouping a large number of blobs into two clusters. Several blobs and clusters have been grouped for two distant vehicles and for the roof structure on the left. In the latter case, the roof appears to be moving with respect to the

Figure 22 Final result for the combination of segmentation results applied to the image shown in Figure 20.

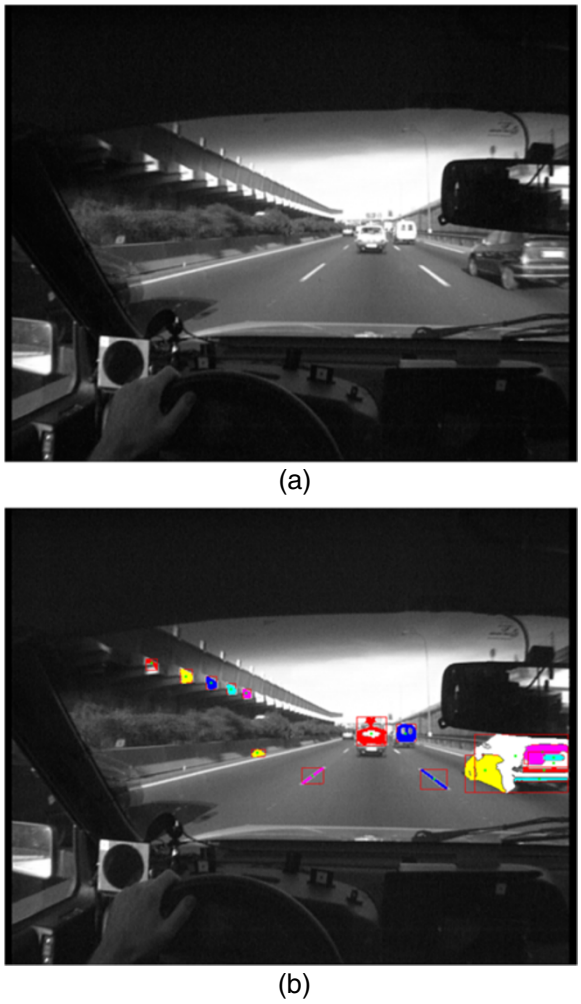

Figure 23 Image segmentation result for a one-way highway example. (a) Original image. **(b)** Blobs found in the previous image.

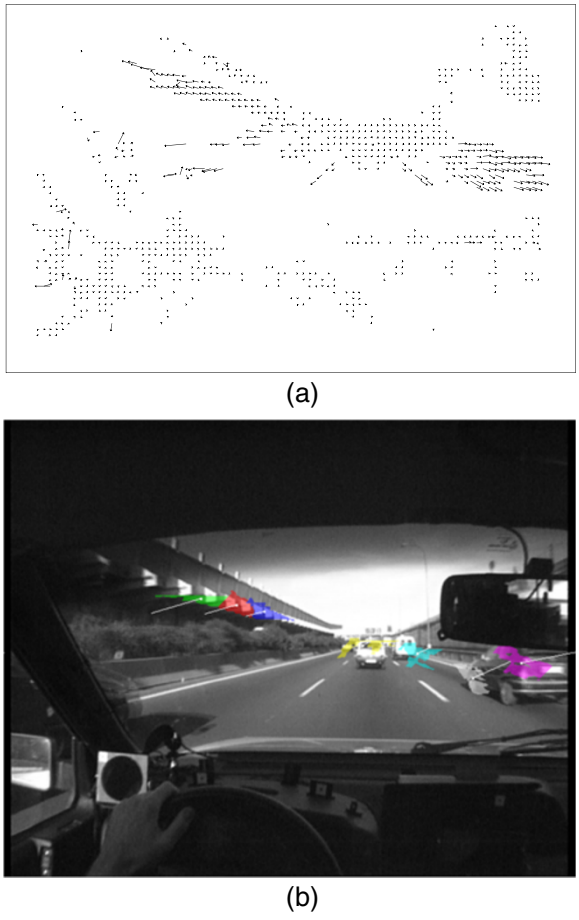

Figure 24 Optical flow segmentation for the one-way highway example of Figure 23. (a) Optical flow. **(b)** Clusters.

camera, resulting in several blobs being associated to their corresponding clusters.

The results are satisfactory in spite of the great complexity of the scene, because all vehicles except one have been located. The exception is related to the small vehicle on the left which is far from the experimental car and is almost hidden by the background.

8 Conclusion

This paper has addressed the search and detection of moving objects in an image sequence using a moving camera. Starting from the optical flow and image segmentation processes, the objects in the scene and their motion (both direction and magnitude) are obtained.

To obtain an estimate of the apparent movement, the optical flow of each image in the sequence is calculated. This optical flow is then segmented, and similar vectors

Figure 25 Final cluster and blob association for the one-way highway example of Figure 23.

are grouped since they are expected to belong to the same object. In a third step, the segmentation of images is addressed to distinguish the objects present in the scene. Finally, the results of the two kinds of segmentation are grouped to associate each object to its apparent motion.

The method described has been applied to detect moving objects as perceived by a driver which could interfere with the driver's behavior, either by causing a distraction or by posing real dangers that require immediate attention. The input data consists of video sequences recorded by a camera mounted on a conventional vehicle driving on public roads, implying several challenges when calculating the optical flow: moving camera (all objects appear to move) and variable natural lighting (constantly changing shadows and reflections).

The results show that most of the relevant objects from the scene are properly detected and associated to their corresponding movement, while unrelated objects are only grouped when they are quite far and very difficult to distinguish, even to the human eye.

Endnote

[a]The speedup is defined as the sequential execution time divided by the parallel execution time to know how much a parallel algorithm is faster than a corresponding sequential algorithm.

Competing interests
The authors declare that they have no competing interests.

Received: 12 November 2013 Accepted: 11 April 2014
Published: 1 May 2014

References

1. A Garcia-Dopico, JL Pedraza, M Nieto, A Pérez, S Rodríguez, J Navas, Parallelization of the optical flow computation in sequences from moving cameras. EURASIP J. Image Video Process. **2014**, 18 (2014)
2. J Barron, D Fleet, SS Beauchemin, Performance of optical flow techniques. Int. J. Comput. Vis. **12**(1), 43–47 (1994)
3. T Brox, A Bruhn, N Papenberg, J Weickert, ed. by T Pajdla, J Matas, High accuracy optical flow estimation based on a theory for warping, in *Lecture Notes in Computer Science (LNCS): European Conference on Computer Vision (ECCV)*, vol. 3024 (Springer Berlin, 2004), pp. 25–36
4. SN Tamgade, VR Bora, Motion vector estimation of video image by pyramidal implementation of Lucas-Kanade optical flow. Paper presented at the 2nd international conference on emerging trends in engineering and technology (ICETET), Nagpur, 16–18 Dec 2009, pp. 914–917
5. K Pauwels, MM Van Hulle, Optic flow from unstable sequences through local velocity constancy maximization. Image Vis. Comput. **27**(5), 579–587 (2009)
6. A Doshi, AG Bors, Smoothing of optical flow using robustified diffusion kernels. Image Vis. Comput. **28**(12), 1575–1589 (2010)
7. A Bruhn, J Weickert, C Schnörr, Lucas/Kanade meets Horn/Schunck: combining local and global optic flow methods. Int. J. Comput. Vis. **61**(3), 211–231 (2005)
8. M Drulea, IR Peter, S Nedevschi, Optical flow a combined local-global approach using L1 norm. Paper presented at the 2010 IEEE 6th international conference on intelligent computer communication and processing, Cluj-Napoca, 26–28 Aug 2010, pp. 217–222
9. T Brox, J Malik, Large displacement optical flow: descriptor matching in variational motion estimation. IEEE Trans. Pattern Anal. Mach. Intell. **33**(3), 500–513 (2011)
10. JS Zelek, Towards Bayesian real-time optical flow. Image Vis. Comput. **22**(12), 1051–1069 (2004)
11. S Tan, J Dale, A Anderson, A Johnston, Inverse perspective mapping and optic flow: a calibration method and a quantitative analysis. Image Vis. Comput. **24**(2), 153–165 (2006)
12. H Liu, TH Hong, M Herman, R Chellappa, ed. by B Buxton, R Cipolla, Accuracy vs. efficiency trade-offs in optical flow algorithms, in *LNCS: 4th European Conference on Computer Vision*, vol. 1065 (Springer Berlin, 1996), pp. 271–286
13. B Buxton, B Stephenson, H Buxton, Parallel computations of optic flow in early image processing. IEEE Proc. F: Commun. Radar Signal Process. **131**(6), 593–602 (1984)
14. K Sakurai, S Kyo, S Okazaki, Overtaking vehicle detection method and its implementation using IMAPCAR highly parallel image processor. IEICE Trans. **91-D**(7), 1899–1905 (2008)
15. H Bulthoff, J Little, T Poggio, A parallel algorithm for real-time computation of optical flow. Nature. **337**(6207), 549–553 (1989)
16. A Del Bimbo, P Nesi, Optical flow estimation on Connection-Machine 2. Paper presented at the computer architectures for machine perception, New Orleans, 15–17 Dec 1993, pp. 267–274
17. H Wang, J Brady, I Page, A fast algorithm for computing optic flow and its implementation on a transputer array. Paper presented at the British machine vision conference (BMVC90), Oxford, 24–27 Sept 1990, pp. 175–180
18. C Colombo, A Del Bimbo, S Santini, A multilayer massively parallel architecture for optical flow computation. Paper presented at the 11th IAPR international conference on pattern recognition, 1992. Vol. IV. Conference D: architectures for vision and pattern recognition, The Hague, 30 Aug–3 Sept 1992, pp. 209–213
19. MG Milanova, AC Campilho, MV Correia, Cellular neural networks for motion estimation. Paper presented at the 15th international conference on pattern recognition (ICPR'00), Barcelona, 3–7 Sept 2000, pp. 819–822
20. J Sosa, J Boluda, F Pardo, R Gómez-Fabela, Change-driven data flow image processing architecture for optical flow computation. J. Real-Time Image Process. **2**(4), 259–270 (2007)
21. J Díaz, E Ros, R Agís, JL Bernier, Superpipelined high-performance optical-flow computation architecture. Comput. Vis. Image Underst. **112**(3), 262–273 (2008)
22. N Devi, V Nagarajan, FPGA based high performance optical flow computation using parallel architecture. Int. J. Soft. Comput. Eng. **2**(1), 433–437 (2012)
23. Y Mizukami, K Tadamura, Optical flow computation on compute unified device architecture. Paper presented at the the 14th international conference image analysis and processing (ICIAP), Modena, 10–14 Sept 2007, pp. 179–184
24. M Gong, Real-time joint disparity and disparity flow estimation on programmable graphics hardware. Comput. Vis. Image Underst. **113**(1), 90–100 (2009)
25. MV Correia, AC Campilho, ed. by AC Campilho, MS Kamel, A pipelined real-time optical flow algorithm, in *LNCS: Image Analysis and Recognition*, vol. 3212 (Springer Berlin, 2004), pp. 372–380
26. J Chase, B Nelson, J Bodily, Z Wei, DJ Lee, Real-time optical flow calculations on FPGA and GPU architectures: a comparison study. Paper presented at the 16th international symposium on field programmable custom computing machines, Palo Alto, 14–15 April 2008, pp. 173–182
27. K Pauwels, M Tomasi, JD Alonso, E Ros, NM Van-Hulle, A Comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features. IEEE Trans. Comput. **61**(7), 999–1012 (2012)
28. B Lucas, T Kanade, An iterative image registration technique with an application to stereo vision. Paper presented at 7th international joint conference on artificial intelligence (IJCAI), Vancouver, 24–28 Aug 1981, pp. 674–679
29. B Lucas, Generalized image matching by method of differences. PhD Thesis, Department of Computer Science, Carnegie-Mellon University, 1984
30. Top 500 Supercomputer Sites (2014). http://www.top500.org. Accessed 25 April 2014
31. M Fleury, AF Clark, AC Downton, Evaluating optical-flow algorithms on a parallel machine. Image Vis. Comput. **19**(3), 131–143 (2001)
32. A García Dopico, M Correia, J Santos, L Nunes, ed. by M Bubak, G van Albada, P Sloot, and J Dongarra, Distributed computation of optical flow,

in *LNCS: Computational Science*, vol. 3037 (Springer Berlin, 2004), pp. 380–387
33. T Kohlberger, C Schnorr, A Bruhn, J Weickert, Domain decomposition for variational optical-flow computation. IEEE Trans. Image Process. **14**(8), 1125–1137 (2005)
34. EM Kalmoun, H Köstler, U Rüde, 3D optical flow computation using a parallel variational multigrid scheme with application to cardiac C-arm CT motion. Image Vis. Comput. **25**(9), 1482–1494 (2007)
35. A Pérez, MI García, M Nieto, JL Pedraza, S Rodríguez, J Zamorano, Argos: an advanced in-vehicle data recorder on a massively sensorized vehicle for car driver behavior experimentation. IEEE Trans. Intell. Transport. Syst. **11**(2), 463–473 (2010)
36. G Adiv, Determining three-dimensional motion and structure from optical flow generated by several moving objects. IEEE Trans. Pattern Anal. Mach. Intell. **7**(4), 384–401 (1985)
37. W Thompson, T Pong, Determining moving objects. Int. J. Comput. Vis. **4**(1), 39–57 (1990)
38. JG Choia, SD Kim. Multi-stage segmentation of optical flow field.Signal Process. **54**(2), 109–118 (1996)
39. D Chung, WJ MacLean, S Dickinson, Integrating region and boundary information for spatially coherent object tracking. Image Vis. Comput. **24**(7), 680–692 (2006)
40. J Klappstein, T Vaudrey, C Rabe, A Wedel, R Klette, ed. by T Wada, F Huang, and S Lin, Moving object segmentation using optical flow and depth information, in *LNCS: Advances in image and Video Technology*, vol. 5414 (Springer Berlin, 2009), pp. 611–623
41. K Pauwels, N Krüger, F Wörgötter, NM Van-Hulle. J. Vis. **10**(10), 18 (2010)
42. H Samija, I Markovic, I Petrovic, Optical flow field segmentation in an omnidirectional camera image based on known camera motion. Paper presented at the 34th international convention MIPRO, Opatija, 23–27 May 2011, pp. 805–809
43. R Namdev, A Kundu, K Krishna, C Jawahar, Motion segmentation of multiple objects from a freely moving monocular camera. Paper presented at the IEEE international conference on robotics and automation (ICRA), Saint Paul, MN, USA, 14–18 May 2012, pp. 4092–4099
44. M Correia, A Campilho, J Santos, L Nunes, Optical flow techniques applied to the calibration of visual perception experiments. Paper presented at the 13th international conference on pattern recognition (ICPR), Vienna, 25–29 Aug 1996, pp. 498–502
45. M Correia, A Campilho, Implementation of a real-time optical flow algorithm on a pipeline processor. Paper presented at the international conference of computer based experiments, learning and teaching, Szklarska Poreba. 28 Sept–1 Oct 1999
46. B Fornberg, Generation of finite difference formulas on arbitrarily spaced grids. Math. Comput. **51**, 699–706 (1988)
47. E Simoncelli, E Adelson, D Heeger, Probability distributions of optical flow. Paper presented at the IEEE conference on computer vision and pattern recognition, Maui, 3–6 June 1991, pp. 310–315
48. J Canny, A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8**(6), 679–698 (1986)
49. R Deriche, J Cocquerez, G Almouzny, An efficient method to build early image description. Paper presented at the 9th international conference on pattern recognition, Rome, 14–17 Nov 1988, pp. 588–590

doi:10.1186/1687-5281-2014-24
Cite this article as: Garcia-Dopico *et al.*: **Locating moving objects in car-driving sequences.** *EURASIP Journal on Image and Video Processing* 2014 **2014**:24.