Real-Time Vehicle Tracking for Driving Assistance

Andrea Fossati
CVLab - EPFL
1015 Lausanne - Switzerland
andrea.fossati@epfl.ch

Patrick Schönmann
Cinetis SA
1920 Martigny - Switzerland
patrick.schoenmann@cinetis.ch

Pascal Fua
CVLab - EPFL
1015 Lausanne - Switzerland
pascal.fua@epfl.ch

Abstract

Detecting car taillights at night is a task which can nowadays be accomplished very fast on cheap hardware. We rely on such detections to build a vision-based system that, coupling them in a rule-based fashion, is able to detect and track vehicles. This allows the generation of an interface that informs a driver of the relative distance and velocity of other vehicles in real time and triggers a warning when a potentially dangerous situation arises. We demonstrate the system using sequences shot using a camera mounted behind a car's windshield.

1. Introduction

One of the main particularities that distinguishes newer cars from older ones is the growing use of embedded electronic components whose role is to improve driver safety. Even purely mechanical devices, such as brakes, have now been electronically enhanced. While some cars are now equipped with proximity sensors for parking assistance, the new trend is now to equip them with long-range radar sensors or video-based assistance functions to protect the vehicle's occupants, those of other cars and pedestrians [8]. In this paper we describe a low cost video-based assistance system, that relies on taillight tracking to locate other vehicles at night-time, to analyze their trajectory and to avoid collisions. We chose night-time because it is a delicate situation, in which the driver's perception of distance is far worse than during the daytime, which makes our system potentially very helpful. The need for reliable real-time performance makes it a challenging problem as it imposes restrictions on the algorithms that may be used.

We demonstrate our approach on sequences shot with a small camera mounted behind the cars windshield and pointing forwards. Essentially the same approach could have been used with a camera pointing backwards to monitor the situation behind the car by detecting headlights instead of taillights.

Our approach includes the following steps: We first detect independent candidate taillights using standard low-level image processing. We then couple them using adequate criteria. Finally we track the resulting pairs over time to infer the presence and location of vehicles. This lets us generate a radar-like view that includes distance to other cars and their relative velocity. As shown in Fig. 1, it can be very helpful to the driver. By further analysis of the input image the system can also detect if a blinker is activated.

The contribution of this paper is therefore twofold: First, we propose a simple, effective and real-time technique to track multiple vehicles at night-time, both in urban and rural environments while realistically estimating their trajectories and velocities with respect to the camera. Secondly, we provide the driver with a valuable interface that can help him understand the situation by augmenting his perception of the traffic conditions and triggering a warning when events that require an increased level of attention are likely to happen.

2. Related Work

The standard approach for robust and accurate vehicle tracking in traffic consists in adopting sophisticated instruments like radars or lidars, as for example in [14]. However, this has the drawback of being very expensive, in contrast to standard video cameras. For this reason, in recent years, several authors have investigated the video-based vehicle detection and tracking issue. Some researchers focused on analyzing traffic and vehicles' motion [12, 4] or

Figure 1. Sample output of our Real-Time Tracking System. On the left part tracked vehicles are marked and their distance in meters to the camera is overlaid on them. On the right part a radar-like view of the scene is drawn: A green rectangle represents the car on which the camera is mounted and yellow rectangles represent other vehicles. The purple segment on them indicates their relative velocity with respect to the camera.

on detecting cars [5] using a static camera. Others studied instead what kind of information can be retrieved using vision algorithms on sequences captured using a camera that is itself mounted on a vehicle. Among these [6] used a video camera to detect, however not tracking them in time, light sources which could correspond to other vehicles, using the light source spectral distribution for discriminating purposes. The work in [1] has an approach similar to ours in the sense that uses taillights to detect vehicles, but does not determine their location in space and to run in real time needs a very low-resolution input and a hybrid hardware/software implementation. It also fails in case a blinker is activated. Similarly [7] detects oncoming cars through their headlights but also this work only focuses on determining if a car is present, and not where it is located.

On the other hand some works tackled the real-time tracking problem, which is the one we try to solve. Different techniques have been proposed in the literature, for example using a mixture of sonar and vision information [10] or using stereo vision [11]. For tracking purposes [13] suggested to mount a few artificial landmarks on the car to be followed, while [3] used templates of a car's back to perform the tracking. In [2] a system for tracking cars on highways was proposed, using edges and templates, which was able to run in real-time thanks to an *ad hoc* system architecture. Finally [15] describes a system used by an autonomous car to track and follow a lead vehicle at night, which uses taillights to compute an approximate trajectory. This method can follow only one car and requires manual initialization of the location of the lead vehicle. Our contribution has therefore been to unify and improve the latter approaches to build a fully automatic system able to track multiple cars, at night, both in highways and city environments. Moreover, our system does not need any special hardware but still runs in real time, which gives it the ability to promptly trigger useful warnings for the driver.

3. Algorithm Overview

Our framework is composed of two main parts. The first part, described in Section 4, involves analyzing the input frames separately and outputting a list of detected vehicles, together with a confidence value for each such candidate. This list is obtained by first finding image patches most likely to correspond to taillights and then coupling them by selecting among all possible pairs those that satisfy certain criteria.

The second part takes those detected pairs as input, as discussed in Section 5. It then generates a time-consistent reconstruction of the scene by linking detections across consecutive frames. This allows the system to find those that truly correspond to actual vehicles and to reconstruct their trajectories with respect to the camera. Finally such information is used to generate the radar-like view of the scene and to predict potential collisions.

4. Vehicle Detection

4.1. Light coupling for vehicle detection

The vehicle detection algorithm consists in analyzing the input images separately: In each frame we first detect all the n light patches, secondly we generate the $\frac{1}{2}n(n-1)$ possible pairs of candidate taillights and finally we filter out those which *a priori* cannot be a vehicle. In this section we will first describe how the light pairs are filtered, to give a global view of how vehicle detection is performed. We will then explain in detail how we detect light patches in the input images and characterize them.

4.1.1 Pair Filtering

First of all, we need to define a few measures and thresholds in order to choose the pairs of lights that are the most likely to be the two main taillights of a vehicle. To do this, we begin by introducing a set of conditions C_i to be necessarily fulfilled. Any pair of lights that fails at complying to at least one of them will be immediately removed from the list of potential vehicles. To avoid missing some vehicles, all the parameters used to compute these conditions are set in a way that minimizes the number of false negatives, thus including also some outliers that will be filtered out in the successive tracking phase.

The most important threshold to be applied is the need for a pair to have both its lights on the same horizontal line. We therefore begin by defining the first condition C_1 as the

fact that the angle between them must be smaller than a very low threshold ϵ as follows:

$$C_1 : \arctan\left(\frac{\Delta\mu_y}{\Delta\mu_x}\right) \leqslant \epsilon,$$

where $\Delta\mu_y$ is the vertical distance between two light patches and $\Delta\mu_x$ is their horizontal distance. We will explain in Section 4.2.3 how to compute such quantities.

Secondly, we assume that the shape of both taillights should be similar, assuming minimal variation in the lamp. We therefore choose a reasonable threshold ζ which has been computed after validation on the training sequences. Condition C_2 can then be defined as

$$C_2 : \Delta\text{shape} \leqslant \zeta,$$

where Δshape is computed as will be defined in Section 4.2.4.

As the area of lights theoretically should not vary too much either, we define a third condition for removing pairs that have a large difference between the areas of both lights:

$$C_3 : \Delta\text{area} \leqslant \overline{\text{area}}.$$

In Section 4.2.4 we will show how the area of a candidate taillight is computed. The quantity Δarea defines the difference of area between two lights, while $\overline{\text{area}}$ is their average area.

Finally condition C_4 represents the fact that two candidate taillights should have a similar appearance:

$$C_4 : \Delta\text{type} \leqslant \eta.$$

The type of a candidate taillight is defined as described in Section 4.2.5, where also a quantitative measure to compute the difference Δtype between two light patches will be introduced.

4.1.2 Dissimilarity Indicators

We can now define four dissimilarity measures D_i from the left terms of the previous inequalities by normalizing them with their right term, thus obtaining variables whose value is between 0 and 1 if their corresponding condition C_i is fulfilled:

$$D_1 = \frac{\arctan(\frac{\Delta\mu_y}{\Delta\mu_x})}{\epsilon} \qquad D_2 = \frac{\Delta\text{shape}}{\zeta}$$

$$D_3 = \frac{\Delta\text{area}}{\overline{\text{area}}} \qquad D_4 = \frac{\Delta\text{type}}{\eta}.$$

The final dissimilarity measure D consists of the sum of all D_i:

$$D = \sum_{i=1}^{4} D_i. \tag{1}$$

Once the previous conditions and dissimilarity measures are defined, the vehicle detection algorithm is as simple as generating each possible pair of taillights and storing those which satisfy all the necessary conditions C_i along with their D value.

4.1.3 Detection Results

As can be seen in Fig. 2, only four out of the about 150 possible pairs of the input frame satisfy all the conditions C_i and have been kept, half of them being real vehicles. Moreover, all individual detected vehicles are coherent without further analysis of the scene.

Figure 2. Detection Results. Top Row: Input frame. Center Row: Taillight Detection. Bottom Row: Vehicle Detection.

4.2. Light Detection and Characterization

We will now explain in more detail how the candidate light patches are obtained and how their features are computed, to clarify the similarity measures introduced in the previous section. Our approach to detecting candidate taillights goes through several steps. First, a score function is computed at every pixel in the input frame. This basic function applies a threshold on the input pixels to retain only those which have a high probability of pertaining to a car's light, and outputs a value related to this probability that can further be used to weight the pixel's contribution to the computation of the light's descriptor. Then, we group the pixels together according to the light they belong to. Finally, we extract from such sets of pixels the features that allow the computation of the conditions C_i presented in the previous section.

4.2.1 Score Function

To make the computation fast we decided to select a color space in which a simple thresholding is enough to discrim-

inate the candidate lights, and HSV was our final choice after some experiments. In fact, being the Value channel a representation of the human perception of brightness, it already gives a good representation where lights differ a lot from other elements in the input scene. We therefore simply define our score function as being the value of such channel if greater than a certain threshold. In our experiments this threshold τ was always fixed to a quarter of the biggest possible value:

$$score(v) = \left\{ \begin{array}{ll} v & \text{if } v \geqslant \tau \\ \emptyset & \text{otherwise} \end{array} \right\}. \quad (2)$$

As we can see in Fig. 3, despite its simplicity, this score function does quite a good job at detecting cars' lights. Most of the false positives have indeed very similar properties to the true positives, and they do not seem to be removable on a per pixel basis without further knowledge of the surrounding region.

Figure 3. Results - Score. Top Row: Input frame. Center Row: Score Function. Bottom Row: Light Detection.

4.2.2 Pixels Grouping

By using a thresholded score measure, in addition to save processing time, we also most likely get disconnected groups of pixels. However, assuming each connected set of pixels is a light is a reasonable choice. Fig. 3 illustrates well the pros and cons of such an approach if we compare the taillights of the closest car at the center of the frame and the several headlights of the on-coming vehicles on the left. Indeed, when applying this approach to the input image depicted in the top row of Fig. 3, the taillights of the near vehicle will be correctly detected. On the other hand, we will have up to five different light sources detected as a single light on the left side of the image. Our algorithm avoids this by using first regrouping connected pixels together, and later splitting them into two or more lights only if necessary, during the tracking phase.

The algorithm we use for grouping pixels is an efficient version of the *Connected Component Labeling* algorithm [16]. We found this algorithm to be a fair tradeoff between overgrouping and creation of small noise-only lights due to artifacts in the border of the real lights. Noise appearing in objects reflecting light, such as road signs, is also likely to be grouped using such an approach, which helps preventing parts of it from being detected as vehicles in a later phase. Reducing the number of those small groups of pixels also helps saving processing time as less pairs of lights will be generated and tested in the vehicle detection phase.

4.2.3 Light Location

We characterize a light by its location and shape, which we formally define as

$$\left(\begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \begin{pmatrix} \sigma_x \\ \sigma_y \end{pmatrix} \right),$$

where (μ_x, μ_y) denotes the centroid and (σ_x, σ_y) denotes the spatial standard deviation of the group of pixels. These simple descriptors of position and shape are sufficient to compute most derived features that we will define in the following subsection.

4.2.4 Light Shape and Area

We approximate the candidate lights as being rectangles centered at (μ_x, μ_y) whose horizontal and vertical dimension are respectively $4\sigma_x$ and $4\sigma_y$. Therefore we have a straightforward way of defining the area of a light as

$$\text{area} = 4\sigma_x \cdot 4\sigma_y. \quad (3)$$

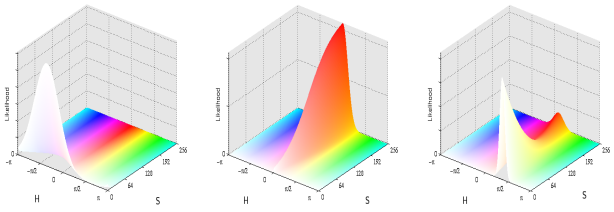
Concurrently, the shape can be defined as the ratio between both sides of the rectangle:

$$\text{shape} = \frac{\sigma_x}{\sigma_y}. \quad (4)$$

4.2.5 Appearance Information and Light Type

To distinguish among headlights, taillights and blinkers we have analyzed several training videos and built, for each of the light types, a probability distribution on the HS space, considering the fact that the V Channel has already been used for thresholding. The task of such distributions is to assign to every input pixel a likelihood to belong to each one of the light types. Fig. 4 shows the 3 different likelihood functions $\omega_T(h, s)$, where $T = \{\text{Headlight}|\text{Taillight}|\text{Blinker}\}$ represents the light type.

At this point we simply need to extend this calculation to measure the likelihood of a given patch of pixels to be of one type or another.



$\omega_{\text{Headlight}}(h,s)$ $\omega_{\text{Taillight}}(h,s)$ $\omega_{\text{Blinker}}(h,s)$

Figure 4. Likelihood distributions for the 3 different light types, based on the H and S channels.

Extending Pixel Measurement to a Light Patch The best choice for measuring the likelihood of a whole light patch is theoretically to multiply the likelihoods of all the pixels (or, to avoid rounding problems in the implementation, to sum the logarithms of their likelihoods). However, this would be based on the assumption of independence of such pixels, which has not proved to be robust enough in our experiments. We have therefore chosen to use a different method and average them instead, which is more robust. The likelihood for a light patch L, given a light type T, is then defined as

$$p(L|T) = \frac{\sum_{(h,s)\in L} \omega_T(h,s)}{\|L\|}, \tag{5}$$

where $\|L\|$ indicates the number of pixels inside the light patch L.

Bayes Factors Finally we need to find a way of measuring the confidence that a light is effectively of a detected type. We chose to compare the probabilities relatively to each other instead of looking at their absolute value, and we will use Bayes factors [9] for that purpose. Bayes factors let us select amongst several probability models the one that most likely has generated a given set of events. The Bayes factor K is defined as

$$K = \frac{p(L|T_1)}{p(L|T_2)}, \tag{6}$$

where T_1 and T_2 are the 2 hypotheses that we want to compare, which in our case represent 2 different light types.

The logarithm of K is called *weight of evidence* and can be measured in different units depending the base of the chosen logarithm. The logarithmic unit on base 10 is called ban, which we will use for the following definition:

$$\text{Weight of Evidence} = 10 \cdot \log_{10}(K) \text{ [deciban]}. \tag{7}$$

The main advantage of Bayes factors amongst other statistical hypothesis testing method resides in the interpretation of the output value. Indeed, the use of a logarithmic unit such as the deciban gives us results that are linear with respect to an intuitive notion of confidence, and a scale of interpretation of K in deciban is given in Table 1 as proposed by Jeffreys [9].

Weight of evidence	Strength of evidence
< 0	Negative (supports T_2)
0 to 5	Barely worth mentioning
5 to 10	Substantial
10 to 15	Strong
15 to 20	Very strong
> 20	Decisive

Table 1. Interpretation of Bayes factors

Light Type We can therefore compute a simple and meaningful measure of the confidence that a light pertains to its most likely type by testing it against all the other types, and then using the lowest of the weights of evidence. We can now define the quantity Δ_{type} between two light patches as the difference of this measure of confidence if they are most likely to belong to the same type, or as ∞ otherwise.

5. Vehicle Tracking

5.1. The Tracking Algorithm

The next step towards a full understanding of the scene is then to ensure consistency in time. We have therefore developed an algorithm for tracking the detected vehicles from frame to frame. For tracking to be robust, we define a criterion to evaluate the confidence that the tracked objects are indeed real vehicles.

The list of detected vehicles, obtained as described in Section 4, is first merged with a list of tracked vehicles, retained from the previous frame: vehicles of which a match is found, meaning that they were already been tracked, have their confidence value updated through a weighted average between the D value of their taillights and their previous confidence value. Then newly detected vehicles, that do not have a match in the list of tracked vehicles, are added. For this we make some basic assumptions about plausible locations where vehicles may appear and optionally assign them a start bonus or malus accordingly. This process is explained in details in Section 5.1.2.

Finally, we sort the list of tracked vehicles according to the vehicles' confidence and loop through it again for detecting physical incoherencies between them, as described in Section 5.1.3.

As the confidence of the tracked vehicles is updated at each iteration of this algorithm, we finish by removing those whose confidence drops below a certain threshold.

5.1.1 Tracked Vehicle Lookup and Special Cases Handling

Once a vehicle is detected in the current scene, its descriptor $[\mu_x, \mu_y, \sigma_x, \sigma_y, \text{area}, \text{shape}, \text{type}]$ is compared to those of the vehicles that are stored in the tracking list, and the nearest neighbor is selected for the match. If there are no neighbors which are close enough, then a new entry is generated in the list.

However, there are two special cases that need to be detected and handled: merged lights from distinct vehicles and lights merged with blinkers. The common point between these two situations is that both provoke a sudden increase of the light's area, which can easily be detected during the nearest neighbor matching phase.

In such case, we want to split a set of connected pixels into two or more lights, and for this purpose we choose to use an approach originally inspired from non-maxima suppression. Lights are split by raising the score threshold τ of Section 5.1 until disconnected groups of pixels are obtained. If the size of one of the latter is less than an adaptive threshold, defined in Eq. 8, we remove its pixels and continue raising τ until we get other disconnected groups. Finally, if we remove all pixels without finding a satisfying partition, we do not split the light. The stopping criterion is hence

$$\frac{\|L_n\|}{\sum_{i=1}^{N} \|L_i\|} < \frac{0.25}{N}, \tag{8}$$

where $\|L\|$ defines the size in pixel of a light patch and N is the number of groups.

If two or more lights are found by the splitting algorithm, these are considered as separated and their dissimilarity measures to other light patches updated accordingly. Otherwise, the increase of the light's area is most probably caused by a blinker. As the light could not be split, we need to find another way of separating the main light and its blinker.

Therefore the center both of the light and the blinker is obtained according to the evolution of the detected light's center between $t-1$ and t. We first set the main light's center to its extrapolated value computed from the position and velocity at $t-1$, and the blinker's center to the center of the detected light. We can then compute the blinker's raw features and type. If the blinker's detected type is *blinker* or any other type with at most a *substantial* weight of evidence compared to *blinker*, we assume that the detected light indeed contained a blinker.

5.1.2 New Detected Vehicles

When a detected vehicle is not matched to any vehicle in the tracking list, a new entry is generated. To increase robustness to noise, newly detected vehicles which are not on the horizon line or at the limit of the camera's viewing angle are given a penalty to overcome issues due to reflections on the license plate of overtaking vehicles, or to the presence of accessory pairs of headlights on their bottom part. Moreover, the main taillights of a vehicle being the most distant ones, we inevitably start tracking those pairs of light sources before the second main light appears in the camera's angle of view, thus giving them a non-negligible advantage.

At the same time, we also prevent most of the false positives to interfere in tracking, based on the assumption that these are likely to appear and disappear in short intervals. A vehicle has to be tracked for at least a certain amount of time before being displayed in an output video or taken into account in the collision prediction phase described in Section 5.2.3. Persistent noise is also filtered out with the step described in the next section.

5.1.3 Physical Inconsistencies

To avoid a physically inconsistent reconstruction of the scene, which can be produced by false positives, we define a box with the approximate proportions of a car around the pairs of taillights and consider that no other vehicles are allowed to have a light in that area. Then, correction is done by comparing each vehicle to others that have a higher confidence and decreasing the confidence of the former if one or more of its taillights are in the other car's box, or one or more of the other's lights are inside its box.

5.2. Motion Analysis

5.2.1 Position Estimation

A standard formula for computing the distance Z to an object of known size simply consists of the focal length f multiplied by the ratio between the size L of the real object and the size l of the object's projection on the image plane. This assumes that the road can be considered planar, which is true in most cases. Then

$$Z = \frac{f \cdot L}{l} = \frac{f \cdot W_{\text{real}}}{W_{\text{sensor}}}, \tag{9}$$

where W_{real} is the real width of the vehicle, which is assumed to be constant for all vehicles, and W_{sensor} is the vehicle's width on the imaging sensor, which is derived from the camera calibration and the vehicle's width in pixels W_{pixels}, that can be computed using the light patches location as

$$W_{\text{pixels}} = (\mu_x^{\text{right}} + 2 \cdot \sigma_x^{\text{right}}) - (\mu_x^{\text{left}} - 2 \cdot \sigma_x^{\text{left}}). \tag{10}$$

Knowing the depth Z of the plane in which the vehicle is situated, Eq. 11 can be derived from it. This defines the lateral shift X between the camera's center and the vehicle's

position, assuming that other vehicles travel in a direction which is approximately parallel to the camera's optical axis:

$$X = \frac{Z \cdot S_{\text{sensor}}}{f}, \tag{11}$$

where S_{sensor} is the horizontal distance on the imaging sensor between the detected vehicle's center and the image center and can be computed with the help of camera calibration.

5.2.2 Data Smoothing and Interpolation

When using only Eq. 9 and Eq. 11 for computing the vehicles' position, the obtained results are correct but their evolution in time is sometimes jittering. We therefore smooth them through spline interpolation as follows: At each frame, vehicles are pushed into a buffer storing one second of data (25 frames in our case), which is an appropriate time interval with enough data to compute stable splines. When the vehicle has been tracked for less than three frames (i.e. there is not enough data to be interpolated), we use the raw values (μ_x, μ_y) and (σ_x, σ_y) of its lights for the computations. After three frames, these values are corrected using standard 2D spline fitting and interpolation, since we are not interested in the relative distance along the vertical Y-axis.

Finally, we apply exactly the same method for correcting the distance, itself computed from the interpolated values of μ and σ. It is important to notice that the smoothed values are neither stored nor taken into account for the following interpolations. Only the raw values are stored in order to make the trajectory reconstruction stick to the real data.

5.2.3 Collision Prediction and Warnings

As previously mentioned, we already have access to the position, velocity and acceleration of vehicles from the spline interpolations. We therefore have all the information we need for making trajectory predictions and thus detect potential imminent collisions.

This can help preventing accidents by warning the driver. If, within a given safety delay (e.g. 4 seconds), a vehicle's extrapolated trajectory will intersect, also considering a spatial safety margin, the trajectory of the car on which the camera is mounted, a warning is triggered.

Apart from predicting collisions, we also provide warnings indicating active blinkers. This is achieved by setting a blinker flag to vehicles at each frame in which either the blinker correction algorithm had to be used, or the headlight's detected type is *blinker*. In order to prevent a flickering effect, we make the warning last as long as the vehicle has a blinker flag set in any of the ten last frames.

6. Results

We present here the results of our vehicle detection and tracking system. The screenshot in Fig. 1 shows the final output of our system in a standard situation. Note that all vehicles present in the input image are detected and their positions and relative velocities estimated. Fig. 5 depicts a video grabbed when arriving in a city center. It confirms that the system works well even in situations of medium artificial lighting if the exposure of the camera is set appropriately. In fact, the system works better when the camera's exposure is set to be very low, so that only strong lights are visible on the sensor.

Figure 5. Results - In town, stopped at traffic lights.

We also illustrate additional visualization features of the radar-like view. As shown in Fig. 6, vehicles further than fifty meters are displayed in the top band reserved for that purpose. Their frontal distance, rounded to ten meters, is displayed above them. Their lateral distance is represented, as for other vehicles, by the lateral offset of the vehicle in the radar-like frame. Fig. 7 shows the representation of a car whose right blinker is active, the small triangle representing the direction the driver indicates by activating such blinker. Finally, vehicles that trigger a warning due to a danger of collision in the next few seconds are colored in red, as in Fig. 8.

Figure 6. Results - Far away vehicle.

Figure 7. Results - Blinker activated.

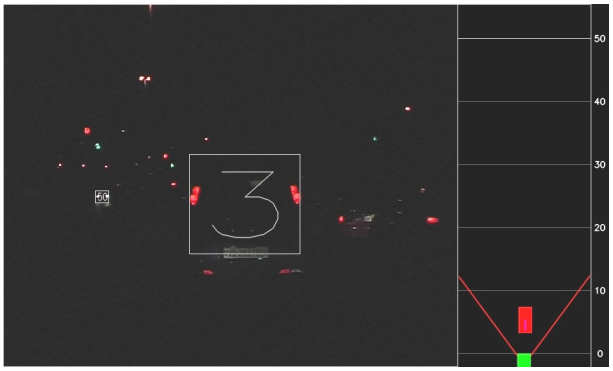


Figure 8. Results - Security warning.

Since we believe that the performance of our algorithm can be better judged in video sequences, we provide them as supplemental material. As all the figures presented in this section, they are composed of two parts. The left one represents the original video with the cars' bounding boxes and their distance in meters overlaid. The right part is a radar-like view of the scene. The green box and the two red lines represent the front part of the car in which the camera is installed, and the angle of view of the camera. The yellow rectangles represent the vehicles on the road, and the purple line starting at their center is their velocity vector.

Finally we present some failure modes of our framework: It intrinsically cannot track motorbikes or vehicles that have a broken taillight. It also does not explicitly handle occlusions, but we think that this is acceptable since the most critical vehicles for the driver's security are the closer, and therefore occluding, ones. When the occluded vehicles will re-appear they will automatically be re-detected and tracked. Obviously it also cannot cope with bumps on the road and large slopes. Finally, given our assumptions about the constant width of vehicles, the distance estimates for big trucks or very small cars will suffer from some inaccuracy. Apart from such cases, we believe that our technique for vehicle detection and tracking is generally able to provide an accurate and realistic trajectory estimation, and is therefore a good starting point for making night driving safer.

6.1. Quantitative evaluation

To better evaluate the accuracy of the distance estimation algorithm, we designed a simple experiment to obtain some quantitative results. Since doing it in the dynamical case would require special and expensive equipment (i.e. a radar), we focused our attention on the static case, which is anyway a good approximation since all the input frames are analyzed separately. We therefore took several pictures of 3 different cars at 3 different distances each (10, 20 and 50 meters). We tested both the case in which the cars are straight in front of the camera and shifted on one side as if they were on a parallel lane. For each one of such cases we took 10 measurements, whose averages are shown in Table 2. As can be noticed there is a slight degradation of accuracy in the vehicles when they are further away and on a parallel lane, but we still believe such figures to represent acceptable distance estimations.

Car Location	10 m	20 m	50 m
Straight in the front	6.16%	6.92%	7.81%
One lane shift	8.04%	8.39%	9.23%

Table 2. Average errors of distance estimations in the static case.

Finally to demonstrate the robustness of the algorithm to missed detections and false positives we manually labeled several test video sequences, making a distinction between the easier highway scenarios and the more challenging city center ones. Then we evaluated our algorithm frame by frame, counting the number of correct detections, missed detections and false positives. The results of these experiments are summarized in Table 3. It can easily be seen in the table that there is a very low ratio of false positives, which in the highway scenario are totally absent. Moreover these experiments show that even the ratio of false negatives is low and, as expected, is better in the highway scenes than in the city center ones, where the clutter and the traffic have more influence.

Scenario	Correct Detections	Missed Detections	False Positives
Highway	10365	187 (1.77%)	0
City Center	3370	200 (5.60%)	5

Table 3. Evaluation of the number of correct vehicle detections, missed detection and false positives.

6.2. Performance Analysis

To provide a first evaluation of the system's performance, we computed the average number of frames per second it

can process on a Pentium D 1.8 GHz dual core. To evaluate computation time only, we use a version of the program that does not provide any output video. From all the 50 videos we have processed, we removed the best (47 fps) and worst (33 fps) results. The remaining processing times varied from 39 to 46 fps, with an average of 43 fps, which means that the average processing time is 23 milliseconds, on input frames of resolution 720×576.

7. Conclusion

We presented a system to detect and track vehicles at night, estimate their position and relative velocity and predict their trajectory. Special care has been taken to make the system robust, by minimizing the false negatives and filtering out the false positives using temporal consistency. The framework also provides information that can help enhancing security by warning the driver in case of potentially dangerous situations. Finally the system runs in real-time on ordinary hardware using as input a standard video camera.

Future work will involve enhancing the system's performances by increasing the quality of its input data. By choosing an optimal camera model and finding its more appropriate exposure, we could already increase the system's accuracy while reducing processing time. We are also considering a hardware implementation of the algorithm to further speed it up.

References

[1] N. Alt, C. Claus, and W. Stechele. Hardware/software architecture of an algorithm for vision-based real-time vehicle detection in dark environments. In *DATE '08: Proceedings of the conference on Design, automation and test in Europe*, pages 176–181. ACM, 2008.

[2] M. Betke, E. Haritaoglu, and L. Davis. Real-time multiple vehicle detection and tracking from a moving vehicle. *Machine Vision and Applications*, pages 69–83, Aug. 2000.

[3] T. Chateau and J. Lapreste. Robust real time tracking of a vehicle by image processing. In *IEEE Intelligent Vehicles Symposium*, pages 315–318, 2004.

[4] B. Coifman, D. Beymer, P. Mclauchlan, and J. Malik. A real-time computer vision system for vehicle tracking and traffic surveillance. *Transportation Research Part C: Emerging Technologies*, 6(4):271–288, 1998.

[5] R. Cucchiara and M. Piccardi. Vehicle detection under day and night illumination. *Proc. of ISCS-IIA99, Special Session on Vehicle Traffic and Surveillance, Genoa, Italy, 1999.*, 1999.

[6] R. DeFauw, S. Lakshmanan, and K. Prasad. A system for small target detection, tracking, and classification. *Intelligent Transportation Systems, 1999. Proceedings. 1999 IEEE/IEEJ/JSAI International Conference on*, pages 639–644, 1999.

[7] M. Eichner and T. Breckon. Real-time video analysis for vehicle lights detection using temporal information. *Visual Media Production, 2007. IETCVMP. 4th European Conference on*, pages 1–1, Nov. 2007.

[8] D. Gavrila and S.Munder. Multi-cue pedestrian detection and tracking from a moving vehicle, 2007.

[9] H. Jeffreys. *Theory of Probability*. Clarendon Press, 1961.

[10] S. Kim, S.-Y. Oh, J. Kang, Y. Ryu, K. Kim, S.-C. Park, and K. Park. Front and rear vehicle detection and tracking in the day and night times using vision and sonar sensor fusion. *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2173–2178, Aug. 2005.

[11] X. Li, X. Yao, Y. Murphey, R. Karlsen, and G. Gerhart. A real-time vehicle detection and tracking system in outdoor traffic scenes. *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, 2:761–764 Vol.2, Aug. 2004.

[12] S. Lin, Y. Chen, and B. Wu. A real-time multiple-vehicle detection and tracking system with prior occlusion detection and resolution. In *Proceedings of the 18th International Conference on Pattern Recognition*, pages 828–831, 2006.

[13] F. Marmoiton, F. Collange, J. Alizon, and J. Derutin. 3d localization of a car observed through a monocular video camera. In *Proceedings of the IEEE International Conference on Intelligent Vehicles*, 1998.

[14] M. Sergi, C. Shankwitz, and M. Donath. Lidar-based vehicle tracking for a virtual mirror. In *Intelligent Vehicles Symposium. Proceedings. IEEE*, pages 333–338, 2003.

[15] R. Sukthankar. Raccoon: A real-time autonomous car chaser operating optimally at night. In *Proceedings of IEEE Intelligent Vehicles*, 1993.

[16] K. Suzuki, I. Horiba, and N. Sugie. Linear-time connected-component labeling based on sequential local operations. *Computer Vision and Image Understanding*, 2003.