LaRA SideCam: a Fast and Robust Vision-Based Blindspot Detection System

Nicolas Blanc, Bruno Steux, *Ecole des Mines de Paris.* Thomas Hinz, *NXP, Hamburg.*

***Abstract*— While shifting lane on the road, the presence of a car in the blindspot can cause many accidents, since the driver does not always turn his head. Therefore, a blindspot car detection is likely to become an essential part of modern vehicles. We developed a program that detects cars in the particular configuration of blindspot using video data taken from the left or right mirror of a car, using on the one hand edge detection and support vector machine (SVM) learning and on the other hand template matching. This makes this program simple, fast and adaptative thanks to SVM learning. The program only uses basical functions of the Ecole des Mines' Camellia open-source image processing library [1], which is close to Intel's IPL library. Thus the program is easy to adapt to another API; it has already been adapted to an embedded system currently in development at NXP Semiconductors (formerly Philips Semiconductors). The source code was tested using the valgrind code checking tool [3] and was validated on real-world video sequences.**

Fig. 1. The Citroën C3 that was used for the video sequence recording, with the CCD camera

I. Hardware-Software Description

The video camera is located close to the left wing mirror of the car, with a 320×240, 30 fps resolution. It provides YUV444 color frames, but since the entire algorithm is based on grayscale images, only the Y channel is used. A 19 minutes sequence was recorded in Paris at daytime (figure 1), in normal weather conditions, both in main roads and in the city, to enable application development.

The program was written in C++, using the Ecole des Mines' Camellia image processing library [1] and LaRAPerception development framework [2]. Since it was intended to be subsequently implemented on an embedded system, its core uses neither floating-point nor dynamic memory allocation nor the C++ Standard Template Library. It fully complies with the ANSI C++ standard and was tested using the valgrind code checking tool [3].

The algorithm is divided into two parts: the first part is intended to detect characteristical elements on the front of the cars and uses SVM artificial learning. The second part is intended to detect the wheels of the cars, when their front is not visible any more, using template matching.

II. Car Front Detection Algorithm

A. Algorithm overview

Blindspot car detection using image processing can be achieved in many different ways, for example optical flow analysis [4] [5] or pattern recognition [6]. A very common and promising way to detect a car both from rear and frontal views is to look for horizontal edges in the picture [7]. This is the first method used in LaRASideCam; it is done here by warping the image to align the horizontal edges of the car with the picture borders, applying a morphological gradient and summing the resulting pixels row by row. Horizontal edges are located thanks to the peaks they generate on the resulting curve. Statistical values about each peak are then computed, and used to characterize a vehicle. This characterization was first done manually, but this first characterization algorithm is not portable to any other hardware configuration, since its parameters depend on many factors such as the image resolution and the optic parameters of the video camera. So a SVM artificial learning algorithm is used to solve this problem automatically. The whole algorithm is simplified by the fact that there is no need to locate the car in its environment, neither to detect a partially hidden car.

1-4244-1068-1/07/$25.00 ©2007 IEEE.

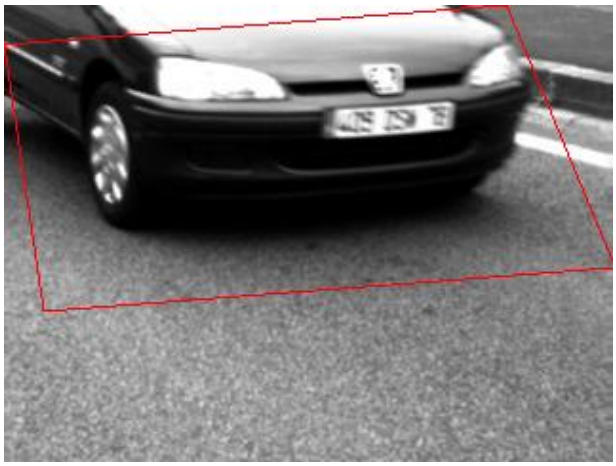

Fig. 2. Picture taken from the left wing mirror camera, with the warping region of interest in red

Fig. 3. Resulting image after warping

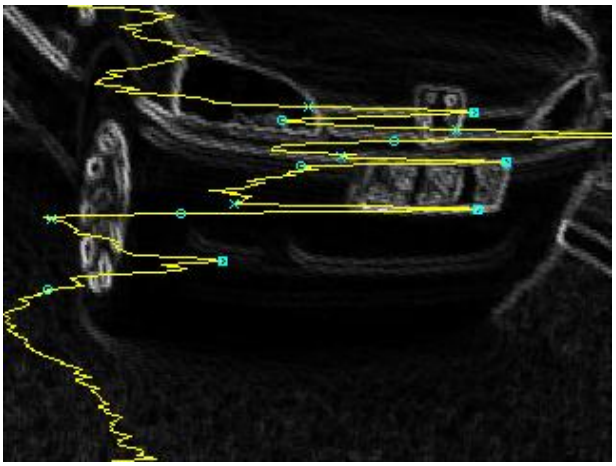

Fig. 4. Resulting image after having applied a morphological gradient, with summation result and detected peaks

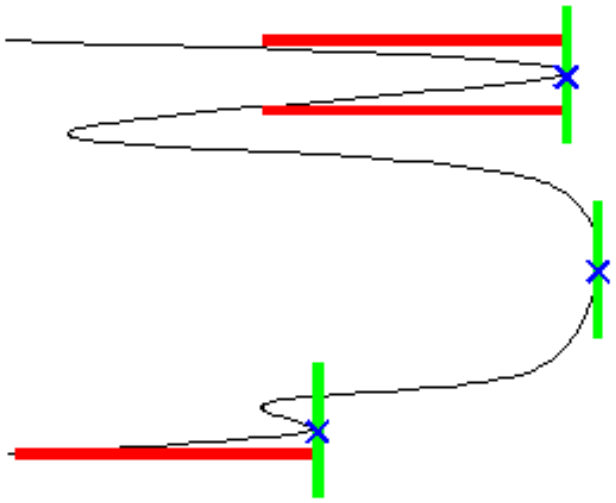

Fig. 5. Peak detection algorithm. The minimal height a peak should have is in red, the maximal width in green. The first peak fills all the conditions, the second "peak" is too wide, and the third peak is not high enough. The statistical values extracted from the peaks are computed inside of the red "minimal height" bounds.

B. Algorithm step by step

Figure 2 represents a typical example of what can be seen using the camera located close to the left wing mirror. The first thing to do is warping the image to straighten the horizontal lines of the car (figure 3).

To detect the edges on the resulting picture, the most obvious solution would have been to use an operator that selects the horizontal edges only, as for example an horizontal Sobel operator. However, by testing different operators it was noticed that such an operator is too restrictive, since in practice all the cars do not have exactly the same orientation on the image. Therefore a morphological gradient operator with a 3×3 square structural element is applied to the picture. The pixels in each row of the resulting image are summed afterwards, using an horizontal summing function (figure 4).

After having shifted all the values of the resulting curve such that the minimal value is 0, the most simple way to detect peaks in this curve would have been to do a thresholding. However, this cannot be done here because there can be a variable background in the curve, mainly caused by noise or by the edges that are not horizontal. So a "peak template" has been defined by its minimal height and maximal width, which are a percentage of the standard deviation of the curve and the height of the image respectively (figure 5), and the program looks for peaks that fit this template.

Now, the most important step is to extract from the curve all the possible information that could help to characterize a car. The exhaustive list of the extracted values is: mean value of the curve, standard deviation of the curve, and for each peak: mean value, standard deviation, third and fourth moment. For the peaks, these values are computed inside of the left and right bounds determined by the peak detection algorithm, which are

the first values around the peak that fit the minimal height condition (figure 5). All these values are then used for the computation of the SVM prediction.

C. SVM artificial learning

Chih-Chung Chang and Chih-Jen Lin's LIBSVM library [8] is used, with a radial basis kernel ($K(x_i, x_j) = \exp(-\gamma \parallel x_i - x_j \parallel^2)$, $\gamma > 0$). Besides the SVM classification and prediction algorithms, it provides an useful grid search-based parameter selection tool that looks automatically for the best-suited C and γ parameters (respectively penalty parameter of the error term in the optimization problem, and kernel parameter).

Before using SVM artificial learning, a hand-made car detection algorithm was first built using all the statistical values cited above. This allowed to understand how a car could be distinguished from the environment. For example it was possible to find out which of the statistical values seemed relevant to distinguish the car and no-car situations, and in which situation the algorithm could reasonably be able to detect the car, which was an essential information for the training data building. The car and no-car situations are distinguished as follows: there should be a positive answer only if an important part of the car front is in the warping region of interest, namely vertically from the bodywork bottom to the headlight top, and horizontally at least one half of the car front. This ensures that there will be enough relevant information in the curve.

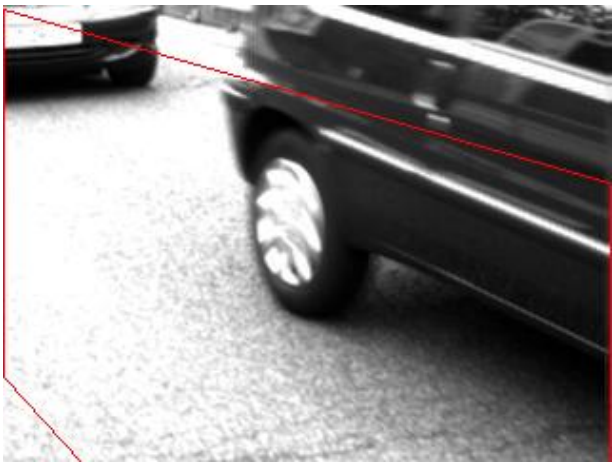

Fig. 6. Input image with the warping region of interest in red. It has been chosen to have a lateral view of the vehicules using as much area as possible.

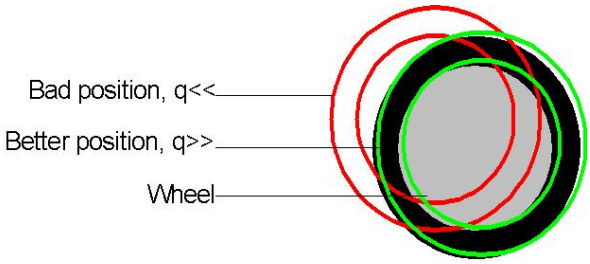

Fig. 7. Template matching algorithm

III. Car Side Detection Algorithm

A. Algorithm overview

This second algorithm is complementary to the first one: cars need to be detected even if their front is not visible on the image, and in that case their wheels are generally visible. Moreover they are very characteristic for a car. To detect them, it was first tried to apply the circular Hough algorithm to a straightened image to detect the wheel rims, distinguishing the true wheels from false detections by using the Hough accumulator. This criterion however appeared not to be very useful, since many wheel rims (and tires) did not have any clear circular outline. It was chosen to apply a template matching algorithm instead: a wheel can always be characterized by a very dark ring surrounding a brighter area. The car bodywork bottom was also detected to improve the response of the algorithm, using an edge detection algorithm similar to that that was used for the car front detection. The algorithm detects potential wheels and tracks the wheels once they have been recognized.

B. Algorithm step by step

First, Camellia's inverse perspective mapping function is used to have a plain view of the side of the car. This function computes a perspective inversion assuming that the input region of interest corresponds to a vertical plane in the input image (figures 6, 8).

To do the template matching, different wheel templates are used, each one of them being defined by its rim and tire radii. It appeared that only three templates already cover a wide range of different wheels sufficiently, since these templates work well even if the wheel does not have exactly the same dimensions. The principle of the template matching algorithm used afterwards is, for a given possible wheel position on the screen, to compute the sum of all the pixels in the rim and in the tire areas (n_r and n_t respectively), and to compute their quotient $q = \frac{n_r}{n_t}$. The probability of being a wheel at this position is proportional to the value of the quotient (figure 7). In the main algorithm, each possible wheel position is tested in a certain image part with a certain pace, and the position with the best quotient is selected.

The algorithm can be divided into two main steps: a hypothesis generation step where the full image is scanned for a wheel, but with a poor accuracy and discrimination, and a hypothesis verification step where only a limited area is scanned with a more accurate precision. Furthermore, since there can never be two wheels close to each other, the image is vertically

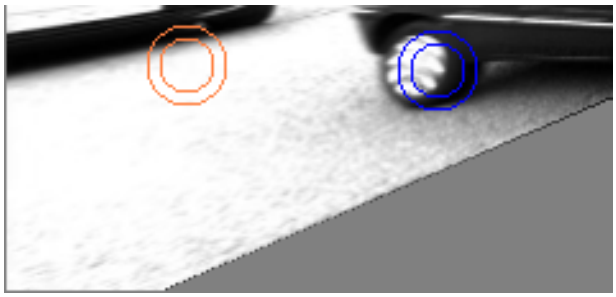

Fig. 8. Situation after having run the inverse perspective mapping and the rough template matching in the left part of the image: a potential wheel was found in this part of the image (orange). The blue wheel in the right part corresponds to the position of a previously tracked wheel in the last frame.

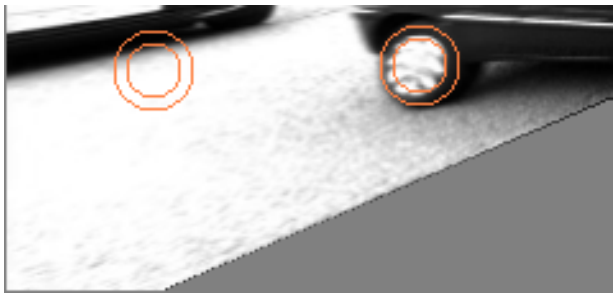

Fig. 9. Situation after having run the accurate template matching: the positions of the potential wheels have been refined, especially for the tracked wheel. The (rim pixels/tire pixels) quotients have become much more significant, too.

split into two parts where the algorithm is run almost independently.

The hypothesis generation step is only run if there was no wheel in the considered image part in the precedent frame. In that case, the considered image part is roughly scanned for a wheel using the template-matching algorithm, typically with a 6 pixels pace. The hypothesis generation algorithm uses a different template at each time it runs (figure 8).

In the next step, a more accurate scanning is carried out in a limited area close either to the position determined by the hypothesis generation step or to the position of the wheel in the precedent frame (depending on the context), and using the same template as previously. Here the typical pace is two pixels. Even if the scanning carried out in the hypothesis generation step is sufficient to locate a wheel in the image if there is one, this more accurate scanning is needed to determine whether there is actually a wheel at this position using the (rim pixels/tire pixels) quotients of the supposed wheels, that are much more significant using an accurate scanning (figure 9).

The bodywork bottom of the cars is also detected by applying the same morphological gradient, horizontal summing and peak detection functions as in the car front detection algorithm. If the Y-coordinate of the axle of

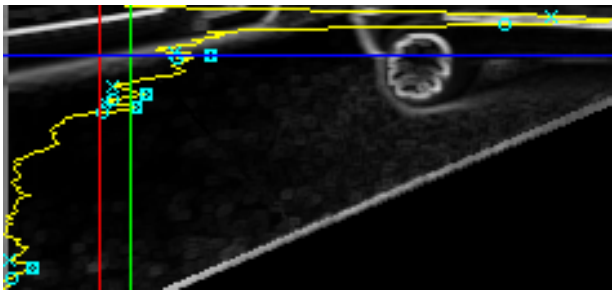

Fig. 10. Resulting image after having applied a morphological gradient, with summation result and detected peaks. The line shown in blue has been selected as the bodywork bottom of the car.

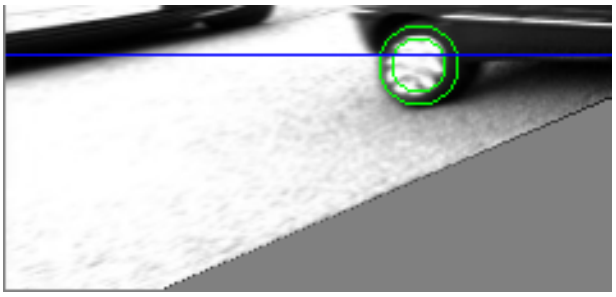

Fig. 11. The final result, after having modified the quotient of the right wheel because of the detected bodywork bottom and compared the (rim pixels/tire pixels) quotients to their respective thresholds. Only the wheel in the right screen part was considered as a real wheel.

a wheel is close enough to a detected peak, the (rim pixels/tire pixels) quotient of this wheel is multiplied by a determined factor, depending on the wheel template that was used to detect the wheel. There can be only one detected bodywork bottom in a given frame, so if each potential wheel has a different bodywork bottom candidate, only the potential wheel with the best quotient (which is supposed to be the most likely to be a real wheel) becomes associated to its bodywork bottom (figure 10).

The final wheel presence determination is done by comparing the quotient of each potential wheel to a detection threshold; this threshold being different for each wheel template (figure 11).

If the wheel detected in the latter step does not belong to the same part of the image as its predecessor, there can be the possibility of two detected wheels in the same image part. In that case, only the one with the biggest (quotient/detection threshold) ratio is kept.

C. *Parameters setting*

The parameters for the wheel/bodywork detection algorithm are: wheel templates (rim and tire radii), peak width and depth for the edge detection algorithm, factors for the quotients of wheels to be multiplied by if there is a bodywork bottom close to their axles, thresholds for wheel presence determination. There was no possibility

Recall rate	False detection rate
51,1%	1,5%

TABLE I

RESULTS FOR THE VALIDATION OF THE WHEEL DETECTION ALGORITHM

to implement a SVM prediction easily to determine these parameters because the wheels are tracked: the answer for a given frame depends on the previous frames. Moreover the training phase would have last a very long time since it is necessary to run the whole algorithm once again for each parameter change. Therefore all the parameters were set empirically.

IV. PERFORMANCES AND VALIDATION

To build the training data for the car front detection algorithm, 16252 frames (that is, 9 minutes video data at 30 fps) were selected from our video record, corresponding to 3 continuous slots. Many various situations were taken into account: various car models and various environments (main road, city, car park). The remaining 10 minutes were used as testing data.

The same test conditions were used to test each algorithm separately, and then both algorithms working at the same time. A recall situation was defined by the negative response of the algorithm whereas there was either a very visible car front (same criterion as for the SVM training set building in section II-C) or an entirely visible wheel on the screen. A false detection situation was defined by the positive response of the algorithm whereas there was strictly no car nor car part visible on the screen. The frames that are not included in any of the two categories below are regarded as indifferent.

To test only the car front detection algorithm without the other one, many different values for the detection threshold on the SVM prediction output were tried, in order to build a recall rate vs. false detection rate diagramm (results in figure 12).

The wheels detection algorithm was then tested using the same criterion. There is no sensibility parameter here (results in table I). The high recall rate is due to the fact that a frame could be regarded as positive even if there was no entire wheel visible on the screen.

The global efficiency of both algorithms was then tested using the only available sensibility parameter, namely the detection threshold for the SVM output (results in figure 13).

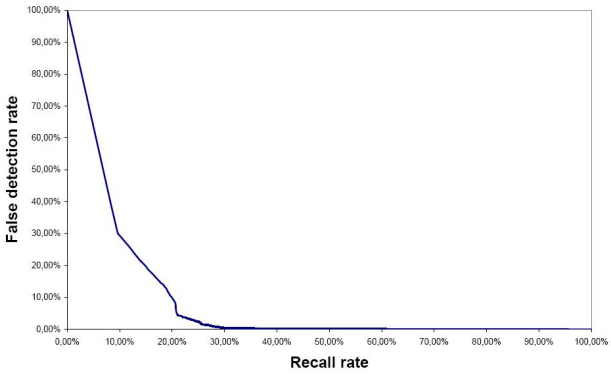

Fig. 12. Results for the validation of the car front detection algorithm

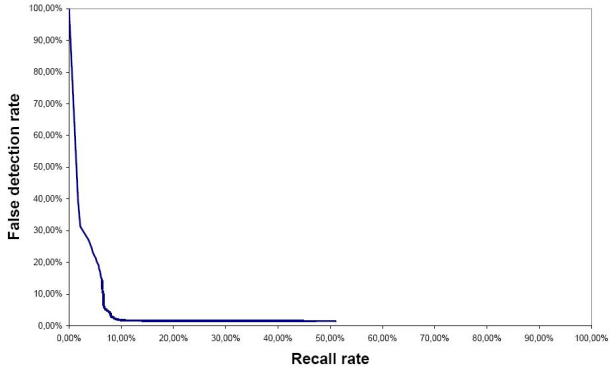

Fig. 13. Results for the global validation

CONCLUSION

With the LaRASideCam project, we had the opportunity to build a fast vision-based blindspot car detection system with interesting results, as can be seen in section IV. This application uses an innovative technique, namely SVM-based artificial learning. The combination of an appearance-based algorithm (car front detection using SVM learning) and a template-based algorithm (wheels and bodywork bottom detection) brings an increased reliability and allows further developments, as for example the determination of the position and the speed of a detected car (using position of the wheels on the screen, displacement of the peaks...). This project was achieved with the help of two instutions: Ecole des Mines de Paris and NXP Semiconductors (formerly Philips Semiconductors) and its Innovation Center in Hamburg, where this development was conducted.

REFERENCES

[1] The Camellia Open Source Image Processing Library: http://camellia.sourceforge.net
[2] The LaRA Project: http://www-rocq.inria.fr/imara/lara2/
[3] Nicholas Nethercote and Jeremy Fitzhardinge, “Bounds-Checking Entire Programs Without Recompiling.”. Informal Proceedings of the Second Workshop on Semantics, Program Analysis, and Computing Environments for Memory Management (SPACE 2004), Venice, Italy, January 2004.
[4] P.H. Batavia, D.E. Pomerleau, C.E. Thorpe, “Overtaking Vehicle Detection Using Implicit Optical Flow”, IEEE Conference on Intelligent Transportation Systems, 1997
[5] Blind Spot / Lane Change by Mobileye: http://www.mobileye-vision.com
[6] Blind Spot Information System (BLIS) by Volvo: http://www.volvo.com

[7] Z. Sun and G. Bebis, “On-Road Vehicle Detection: a Review”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, pp. 694-711, 2006

[8] Chih-Chung Chang and Chih-Jen Lin, “LIBSVM: a Library for Support Vector Machines”, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm