

Correspondence

Parallel and Local Feature Extraction: A Real-Time Approach to Road Boundary Detection

Alberto Broggi

Abstract—This work presents a system for the extraction of road boundaries from an image taken in an out-of-town environment. In this application, computational speed and performance play a fundamental role in the selection of the hardware platform and the design of algorithms. The algorithm has been designed to be implemented on a special-purpose mesh-connected SIMD architecture, PAPRICA, which will be fitted to the vehicle. This presentation focuses on the algorithms and in particular on processing speed.

I. INTRODUCTION

A great deal of work has recently been addressed to the problem of automated navigation and the integration of many different sensors on a vehicle. This work focuses on the use of a passive sensor, such as a camera, to acquire images of the road path, and to derive visual information about the scene.

For this purpose, two approaches have been considered and developed in our laboratory. They are based respectively on optical flow field computation for the detection of obstacles and the determination of their time-to-impact, and on the identification of road boundaries for the detection and tracking of lane position. Since both of these approaches are computationally intensive and must be applied in a real-time environment, their execution imposes intrinsic limitations on vehicle speed. In the first case this problem has been dealt with by using a self-tuning algorithm [1], which automatically reconfigures its parameters according to vehicle speed. However, in the case of road boundary identification, the complex algorithm described in this work is executed only once every few frames, while the intermediate frames are processed through a different and simpler tracking algorithm [5].

To cope with both the high throughput needed for low-level image processing and the constraints imposed by a real-time response, a special-purpose parallel architecture, (PAPRICA [6], [4], [12]) is used. PAPRICA is a massively parallel SIMD architecture which has been designed as a specialized coprocessor to be attached to a general purpose host workstation. In the present implementation PAPRICA is composed of 256 single-bit processing elements (PE's), each one with full 8-neighbor connectivity. The block diagram of the complete vision system is shown in Fig. 1. The use of PAPRICA special-purpose architecture is due to the specific set of features needed for the two real-time tasks mentioned above. These include easy programmability of the applications, the possibility of rearranging run-time the image parameters to simulate a pyramidal architecture, and the scalability of the system. Other architectures with the same characteristics and a morphology-based computational paradigm [16] could also fulfill the requirements for this kind of application.

Manuscript received July 27, 1992; revised November 13, 1993. This work was supported by CNR Progetto Finalizzato Trasporti within the framework of the Eureka PROMETHEUS project, under Contract 91.01031.PF93. The associate editor coordinating the review of this paper and approving it for publication was Prof. Roland T. Chin.

The author is with the Dipartimento di Ingegneria dell'Informazione, Università di Parma, Parma, Italy.

IEEE Log Number 9407598.

This correspondence is organized as follows: Section II gives an overview of the whole system; Section III addresses the problem of real-time edge detection; Section IV discusses the feature extraction approach from a theoretical point of view and presents the actual implementation. Finally, Sections V and VI present some performance analysis together with concluding remarks.

II. SYSTEM OVERVIEW

The feature extraction algorithm described in this correspondence is based on a set of classical filters, including clustering, edge detection, correlation, and morphological filters. Since the main goal of this work is to achieve real-time performance, traditional implementation of these filters requires a speed improvement.

Referring to Fig. 2, the input image comes directly from the digitizer device, and consists of a 256×256 array of 8 bit values (256 grey levels). The features to be extracted (road boundaries) are present in the original image as straight lines determined by high brightness discontinuities and pointing toward a special point in the image (focus of expansion, FOE). In order to extract such information, the following filtering sequence is used: a) Extract the brightness discontinuities; b) retain only the straight lines that point toward the FOE; and c) discard the short segments, usually generated by noisy features.

As shown in Fig. 2, step (a) can be accomplished by preliminary clustering, followed by gradient-based filtering to detect the image edges, and then thresholding and thinning to obtain a final binary image containing segments with single-pixel width. Step (b) in Fig. 2 is accomplished by the determination of the local curvature of the lines, followed by a thresholding operation in order to preserve only the straight lines. The feature extraction algorithm is obtained through correlation with a synthetic image which encodes the knowledge about the position of the FOE. Step (c) in Fig. 2 consists of morphological filters for the final filtering of noisy details.

III. EXTRACTION OF BRIGHTNESS DISCONTINUITIES

Many algorithms for edge extraction are proposed in the literature, each with different features.

For example, the well-known Canny [7] algorithm gives good results, but requires the computation of two derivatives, a convolution with a Gaussian mask, a gradient computation, and thresholding and thinning filterings. These operations are not suitable for implementation on massively parallel architectures with extremely simple single-bit PE's.

The iterative Bayesian approach [3], [10], [14] tends to maximize the *a posteriori* probability following a set of hypothesis. Even its simpler implementation (*weak membrane* [11]), based on the *weak continuity* assumption, needs a lot of floating point computations. This approach is thus not suitable for real-time applications.

On the other hand, the mere application of a convolution with a 3×3 kernel and a threshold would achieve high performance in terms of speed, but would be too sensitive to noise and to the threshold value itself.

The next subsection presents a clustering algorithm, derived from Gaussian filtering, which greatly improves performance and needs few iterations on integer data. Successive application of the 3×3

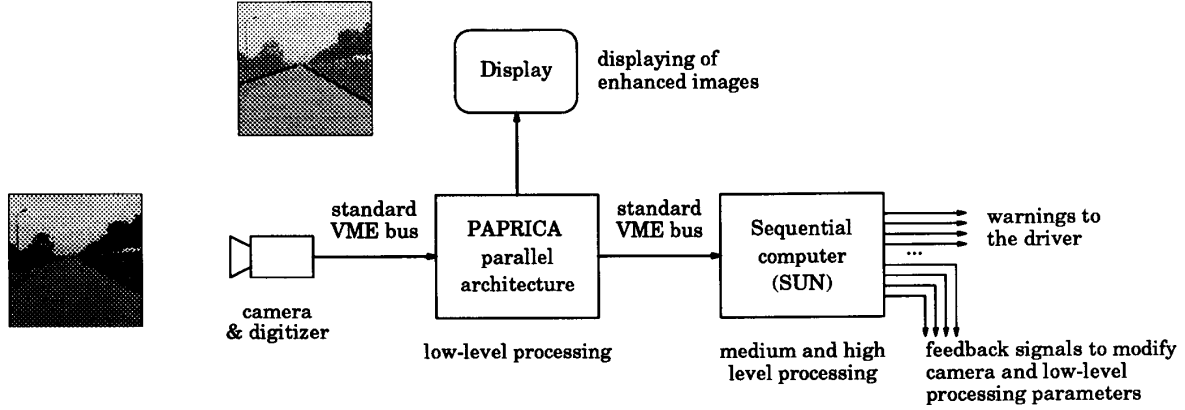


Fig. 1. Block diagram of the on-board vision system: PAPRICA preprocesses the incoming images with local and massively parallel computations; the results, which may be displayed on an on-board monitor, are then fed to a serial computer (PAPRICA host computer) for higher-level processing. The host computer can then warn the driver through special output devices; furthermore, it can generate some feedback signals to modify the parameters of the camera and the behavior of the set of low-level filters.

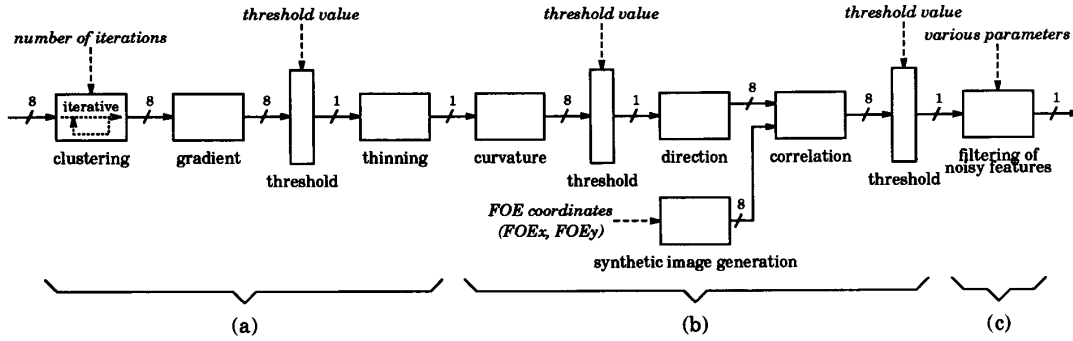


Fig. 2. Block diagram of PAPRICA low-level processing.

filter mentioned above would now produce better and more significant results, thanks to the previous strong clustering.

A. The Clustering Algorithm

This algorithm is a revision of the *adaptive smoothing* algorithm [17], [9], known also as *anisotropic diffusion* [15]: It substitutes a fixed threshold with a function of neighborhood, in order to enhance also weak and isolated intensity discontinuities. Let us define $I^n(x, y)$ as the luminance intensity of the pixel at coordinates (x, y) and iteration number n , belonging to image I ; the value $I^{n+1}(x, y)$ is computed as follows:

$$I^{n+1}(x, y) \triangleq \frac{\sum_{i=-1}^{+1} \sum_{j=-1}^{+1} I^n(x+i, y+j) \times w_{x,y}^n(x+i, y+j)}{\sum_{i=-1}^{+1} \sum_{j=-1}^{+1} w_{x,y}^n(x+i, y+j)} \quad (1)$$

$$\text{with } w_{x,y}^n(u, v) \triangleq e^{-\frac{|g^n(u, v)|^2}{2\bar{K}^2}}$$

where $w_{x,y}^n(u, v)$ represents the weight of the pixel at coordinates (u, v) at iteration number n when computing the new value of $I(x, y)$, and is a function of the gradient magnitude $g^n(u, v)$. In its original formulation, the definition of w (independent of x and

y) comprises constant parameter \bar{K} (equivalent to σ in Gaussian smoothing [7]), which fixes the discontinuities to be preserved; parameter \bar{K} essentially acts as a fixed threshold. The revision of this algorithm eliminates the lack of dependence of threshold \bar{K} on the neighborhood. Thus, $w_{x,y}^n(u, v)$ is here defined as:

$$w_{x,y}^n(u, v) \triangleq e^{-\frac{|g^n(u, v)|^2}{2|k^n(x, y)|^2}}, \quad (2)$$

$$\text{with } k^n(x, y) = M \cdot \max_{i,j=-1,0,+1} g^n(x+i, y+j).$$

Parameter M measures the deviation from the classical average filter ($M \rightarrow \infty$): the lower the value of M , the higher the ratio between weights $w_{x,y}^n(x+i, y+j)$, with $i, j = -1, 0, +1$. Fig. 3(a)–(b) shows the application of the original algorithm, while Fig. 3(c) presents the result obtained by the proposed revision.

B. A Real-Time-Oriented Approach

The computation of nine exponential values for each pixel is time-consuming and is not easy to implement on the extremely simple PE's of massively parallel architectures. An approximation is thus needed in order to make the algorithm more suitable for real-time

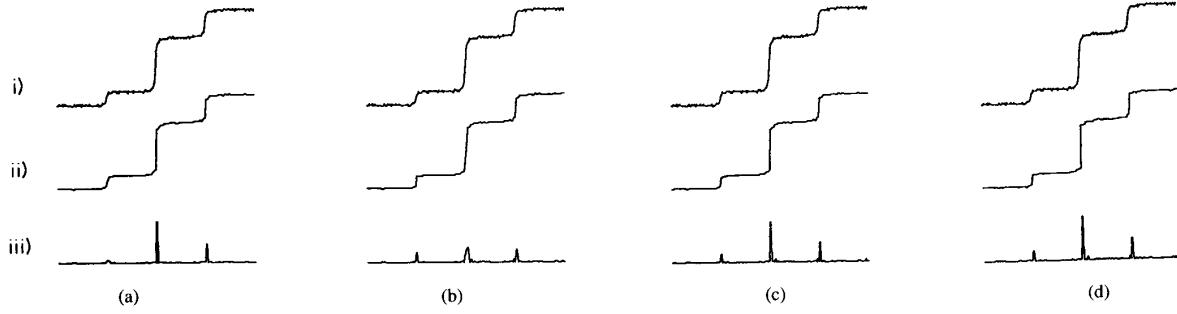


Fig. 3. *i*) Brightness distribution over a linear image profile with 3 edges (with respective strength 20, 100, and 50) and with additive white Gaussian noise ($\sigma = 2$, $\mu = 0$, and magnitude $A = 3$); *ii*) filtered output; *iii*) output gradient magnitude. (a) Application of the original algorithm (with threshold K equal to 3): the high threshold clearly detects only two of the three edges, and smooths out the third; (b) application of the original algorithm (with threshold K equal to 0.5): K has been made small enough to detect the first weak edge, but now the algorithm also identifies as edges the ascending and descending ramps of the other two edges, decreasing their strength; (c) application of the proposed clustering algorithm, with the parameter M equal to 0.35: since the filter also retains the weak edges, in the regions a long way from the real edges, the output gradient is not as uniform as before, but in this case a small threshold will enable the detection of all three edges; (d) the result obtained by the approximation of (5).

processing. The definition of w can be rewritten as:

$$w_{x,y}^n(u, v) = e^{-\frac{1}{2M^2} \left| \frac{g^n(u, v)}{\max_{i,j=-1,0,+1} g^n(x+i, y+j)} \right|^2}. \quad (3)$$

The value inside the modulus can assume values in the range $[0 \div 1]$, thus the argument of the exponential can range from 0 to $-\frac{1}{2M^2}$. Therefore the greater the value of M , the shorter the exponential argument range interval, and thus the more successful (sharper) the approximation of w with a polynomial expression. Unfortunately, the greater the value of M , the less effective this kind of filtering becomes, becoming closer to the classical fixed weight average filter. Experimental tests have shown that $M = 0.3 \div 0.4$ represents a good trade-off, and w can be efficiently approximated by a first order polynomial expression:

$$w_{x,y}^n(u, v) \simeq 1 - \frac{g^n(u, v)}{\max_{i,j=-1,0,+1} g^n(x+i, y+j)}. \quad (4)$$

Moreover, since w represents a weight, it can be multiplied by an arbitrary factor f ; using $f = 1 / \max_{i,j=-1,0,+1} g^n(x+i, y+j)$, (4) becomes

$$w_{x,y}^n(u, v) \simeq \max_{i,j=-1,0,+1} g^n(x+i, y+j) - g^n(u, v). \quad (5)$$

Fig. 3(d) shows the application of (5): the gradient function has the same profile as in Fig. 3(c).

The approximation due to the application of (5) instead of (2) is mainly restricted to the first few iterations, when the clusters begin to appear and the argument of the modulus in (3) is almost equally distributed in the range $[0 \div 1]$; however, in subsequent iterations, it will be polarized around 0 or 1, and the approximation error will be smaller. Fig. 4(a)-(b) shows the original and the smoothed image, respectively.

C. Fast Edge Detection

Since the image obtained so far consists of a set of almost disjointed clusters with well-marked borders, an approximated version of a gradient-based filter, such as an extremely simple convolution with a 3×3 fixed kernel, is sufficient to determine the image edges (Fig. 4(c)). This step is followed by a binarization (Fig. 4(d)), and by a thinning [2] or a non-maximum-suppress [7] algorithm to decrease line thickness. It is important to note that the thresholded

image needs few thinning iterations, since it was obtained from a clustered image, which, by definition, contains only thin brightness discontinuities. Fig. 4(e) shows the final result.

IV. PARALLEL AND LOCAL FEATURE EXTRACTION

Owing to the specific hardware architecture used, the feature extraction algorithm must be based only on local computations. Ideally, each pixel determines if it belongs to the feature or not, i.e., it measures the significance of the information contained in its neighborhood with respect to the local properties of the feature. A function f is first used to assign a *descriptive state* $\mathcal{D}(x, y)$ to each pixel $p(x, y)$, using the luminance values $I(u, v)$ of pixels $p(u, v)$ belonging to its neighborhood $N_{x,y}$:

$$\mathcal{D}(x, y) = f(I(u, v) : p(u, v) \in N_{x,y}) \quad (6)$$

A *characteristics state* $\mathcal{C}(x, y)$ is also assigned to each pixel $p(x, y)$, applying the same function f to a neighborhood configuration where the image brightness $I'(u, v)$ corresponds to the presence of the feature of interest:

$$\mathcal{C}(x, y) = f(I'(u, v) : p(u, v) \in N_{x,y}). \quad (7)$$

From a computational point of view, (7) is of no practical use, since the specification of all the possible configurations of $I'(u, v)$ is not feasible. Thus, another form is used:

$$\mathcal{C}(x, y) = g(x, y, \mathcal{H}) \quad (8)$$

where \mathcal{H} is a set of hypotheses which completely define the feature of interest. Function g is such that it returns the same values that would be obtained by f in (7).

States \mathcal{D} and \mathcal{C} are then compared and the pixel significance (with respect to the feature of interest) is encoded in a *significance state* $\mathcal{S}(x, y)$, which measures the distance between \mathcal{D} and \mathcal{C} :

$$\mathcal{S}(x, y) = \left\| \mathcal{D}(x, y) - \mathcal{C}(x, y) \right\|. \quad (9)$$

The determination of f and g , and the choice of set \mathcal{H} , are the main points in the definition of the feature extraction algorithm.

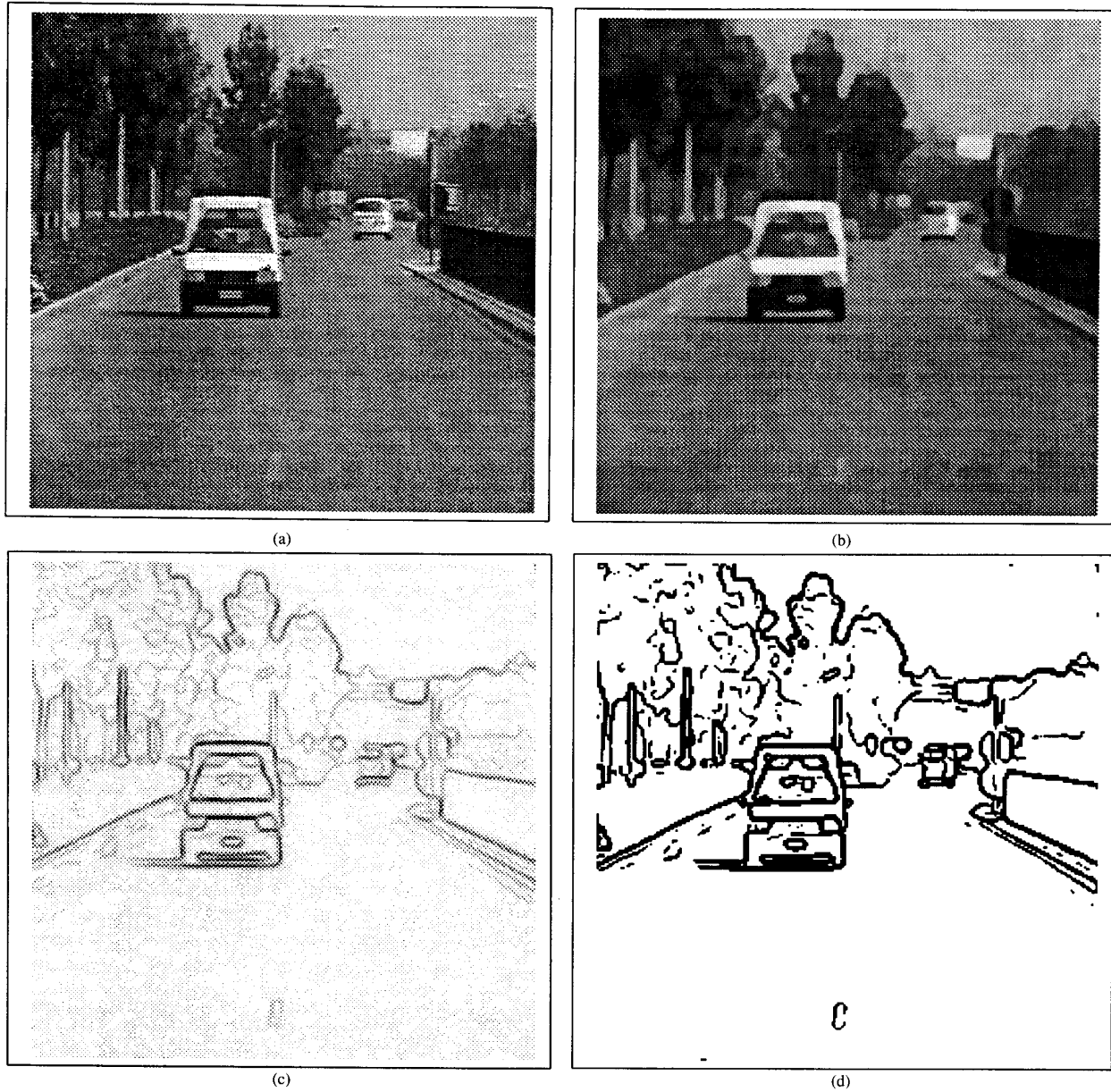


Fig. 4. An example of a complete processing, showing the whole sequence of intermediate results.

A. Identifying Road Boundaries

In this specific case, the hypothesis used to define states \mathcal{D} and \mathcal{C} is that the incoming images are acquired by a camera mounted on the front of a car. In this case, the road boundaries can be defined as the two straight lines starting from the bottom of the image and pointing to the FOE.

Each edge pixel is thus labeled with two 8-bit values representing the local direction d and the local curvature c of the line to which the pixel belongs. In the continuous case it is:

$$d(P) \triangleq \lim_{P' \rightarrow P} \frac{y' - y}{x' - x} \quad \text{and} \quad c(P) \triangleq \lim_{P' \rightarrow P} \frac{1}{R} \quad (10)$$

where symbols $P(x, y)$, $P'(x', y')$, and R refer to Fig. 5(a). Given the high discretization errors obtained in the analysis of a small neighborhood (which corresponds to the $\lim_{P' \rightarrow P}$), neighborhood dimensions (generally 3×3) should be increased in order to obtain higher precision. Unfortunately, the more extended the neighborhood (that is, the higher the distance between P and P'), the greater the difference between the actual result and the theoretical one, in which P and P' should be infinitely close. Experimental studies [8], [18] have derived the value of 11×11 as a good trade-off for the dimension of the neighborhood to be analyzed. Therefore, calling $P^{(1)} = p(x^{(1)}, y^{(1)})$, $P^{(2)} = p(x^{(2)}, y^{(2)})$, ..., $P^{(n)} = p(x^{(n)}, y^{(n)})$ the sequence of n edge pixels, following the hypothesis of an 11×11 neighborhood, and assuming $i > 5$ and $i \leq n - 5$, $d(P^{(i)})$ and

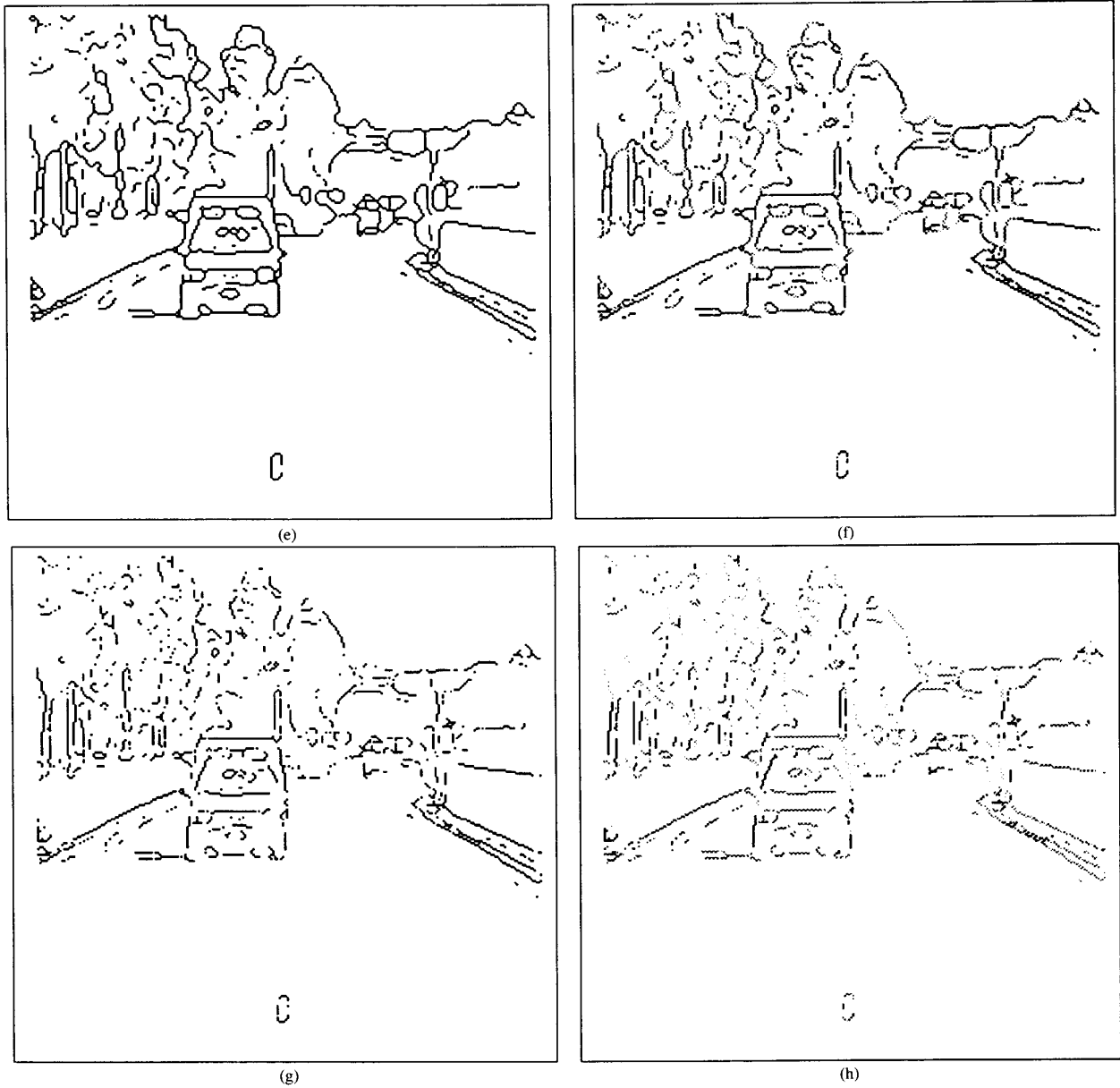


Fig. 4. (continued)

$c(P^{(i)})$ can be obtained as follows:

$$\begin{cases} d(P^{(i)}) = \frac{y^{(i-5)} - y^{(i+5)}}{x^{(i-5)} - x^{(i+5)}} \\ c(P^{(i)}) = \frac{2 |y^{(i-5)}x^{(i+5)} - x^{(i-5)}y^{(i+5)}|}{abc} \end{cases}$$

$$\text{where } \begin{cases} a = \sqrt{(x^{(i-5)})^2 + (y^{(i-5)})^2} \\ b = \sqrt{(x^{(i+5)})^2 + (y^{(i+5)})^2} \\ c = \sqrt{|x^{(i-5)} - x^{(i+5)}|^2 + |y^{(i-5)} - y^{(i+5)}|^2} \end{cases} \quad (11)$$

as shown in Fig. 5(b). Since each pixel can access its 8 neighbors,

5 iterations are needed to collect information about its 11×11 neighborhood, or, in this case, about the line morphology along an 11-step-long monodimensional neighborhood. Fig. 4(f) maps the local curvature, stored in each edge pixel, with a proportional brightness value. The information about the curvature is now used to discard the pixels not belonging to a straight line through a thresholding operation. The result of the threshold is shown in Fig. 4(g).

The direction field $d(x, y)$, assigned to each preserved pixel, is thus identified by its descriptive state $\mathcal{D}(x, y)$. Fig. 4(h) shows a representation of \mathcal{D} , where the brightness intensity is proportional to the direction. The characteristic state $\mathcal{C}(x, y)$ is defined here as the slope of the geometrical line connecting pixel $p(x, y)$ to the FOE, indicated as $p(x_{FOE}, y_{FOE})$:

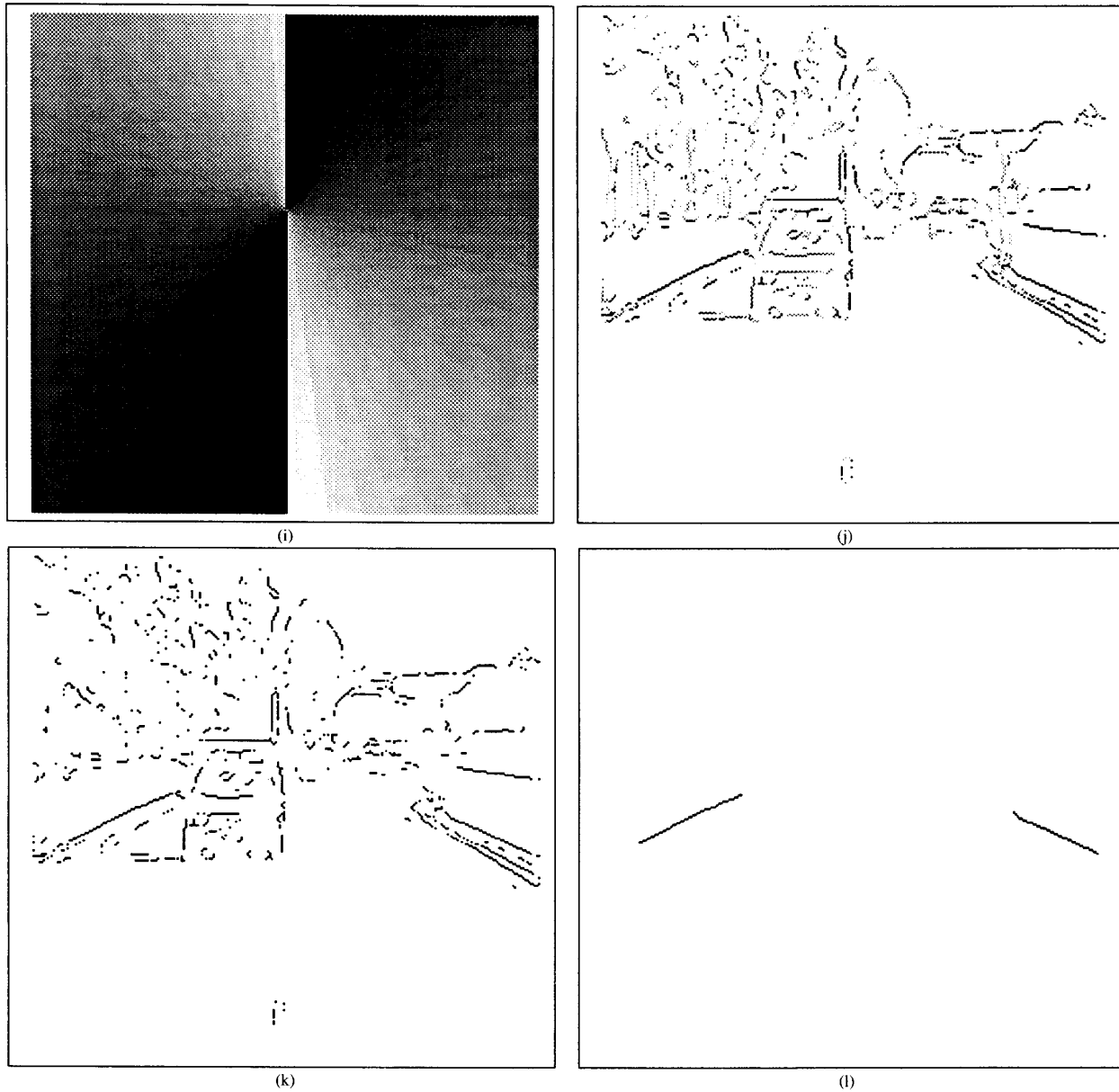


Fig. 4. (continued)

$$C(x, y) = \arctg \frac{y_{FOE} - y}{x_{FOE} - x} \quad (12)$$

Fig. 4(i) shows a representation of C , where the brightness intensity is proportional to the slope. Since the synthetic image (the set of $C(x, y)$) depends on only two parameters (x_{FOE} and y_{FOE}), it can be generated and stored in the host memory. The computation of C is thus reduced to a simple loading from memory.

Significative state S is thus computed as the distance between C and D , and the result, shown in Fig. 4(j), is then thresholded to a medium value, as depicted in Fig. 4(k), to preserve the lines pointing close to the FOE. This includes the case in which car speed has a component perpendicular to the road boundaries. A hysteresis process would reduce processing speed and produce results similar to the ones

obtained with a mere threshold. The final step of the elaboration is to eliminate segments shorter than a given threshold l . The final image is presented in Fig. 4(l).

V. PERFORMANCE ANALYSIS

In order to evaluate the performance of the implementation of the presented algorithm on PAPRICA architecture, computational complexity and communication usage must be measured for each step. The dimension of the neighborhood involved in the computations is the main performance index for computational speed, because the quantity of I/O required by computation increases with the cardinality of the neighborhoods [4].

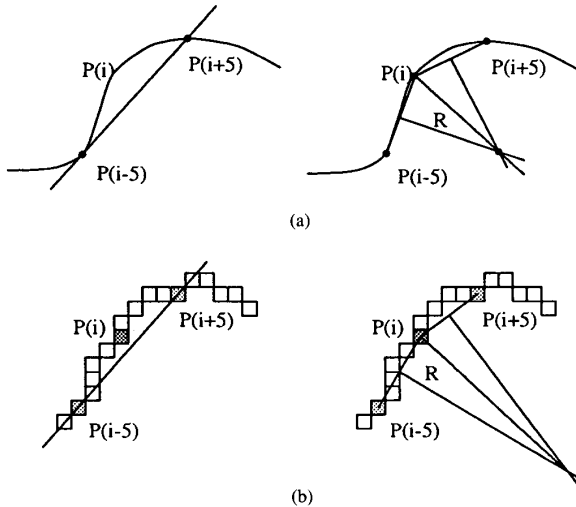


Fig. 5. Direction and curvature: (a) continuous case; (b) discrete case.

TABLE I

SIMULATION TIMES IN SECONDS: FAR RIGHT COLUMN SHOWS PERFORMANCES OBTAINED WITH PAPRICA IMPLEMENTATION, WHICH CONSISTS OF A SEQUENCE OF APPROXIMATED VERSIONS OF ALL THE FILTERINGS PRESENTED IN THIS WORK. THE APPROXIMATIONS RANGE FROM THE USE OF IMAGES WITH 6 BITS/PIXEL, AS IN THE CASE OF THE CLUSTERING STEP, TO THE USE OF MORPHOLOGICAL FILTERS FOR THE DETERMINATION OF APPROXIMATED CURVATURE AND DIRECTION VALUES. A FINAL STEP HAS BEEN ADDED THAT ELIMINATES ALL SEGMENTS LYING IN THE UPPER HALFPANE WITH REGARD TO THE FOE, SINCE THE ROAD BOUNDARIES ARE SUPPOSED TO BE BELOW. THIS STEP IS EQUIVALENT TO SKIPPING THE ELABORATION IN THE RECTANGULAR IMAGE AREA LYING ABOVE THE FOE, PROVIDING A SPEED-UP FACTOR OF ABOUT 1.5 ÷ 1.7, DEPENDING ON THE VERTICAL POSITION OF THE FOE.

HW architecture	Connection Machine CM-2				PAPRICA
	8k PE no FPU 256 × 256	16k PE with FPU 512 × 512	32k PE with FPU 512 × 512	32k PE with FPU 256 × 256	
Filterings (num. of iterations)					256 × 256
Clustering (5 iter.)	1.79	0.51	0.26	0.07	0.71
Gradient	0.01	0.01	-	-	0.09
Thinning (2÷5 iter.)	0.11	0.11	0.06	0.02	0.03
Computation of \mathcal{D}	1.25	0.69	0.36	0.11	0.28
Computation of \mathcal{C}	0.25	0.03	0.01	-	memory load
Long lines (20 iter.)	0.64	0.63	0.32	0.11	0.25
Total	3.5 ÷ 4	2 ÷ 2.5	1 ÷ 1.5	0.3 ÷ 0.4	1.4 ÷ 1.5

- The image clustering algorithm is composed of 10 iterations (5 gradients and 5 weighted averages) of a 3 × 3 neighborhood-based filter, which achieve high performance levels on PAPRICA.
- The labeling of the pixels with a curvature and a direction value is the most complex step in terms of computational complexity, and involves 5 iterations of a 3 × 3 filter.
- Finally, the step used to discard the short segments needs 2l iterations of a 3 × 3 filter.

The first version of this algorithm was implemented in C* on a Connection Machine CM-2 [13], for tuning purposes. Table I presents some numerical results, obtained using different CM configurations and images with different dimensions, compared with implementation on PAPRICA hardware board.

VI. CONCLUSIONS

This correspondence has discussed a fast and highly parallelizable approach to feature extraction. The contribution of this work is not restricted to the feature extraction algorithm itself, but can be seen

as an effort to improve the performance of some well-known basic filters such as image clustering, edge detection, and curvature and direction extraction, toward a real-time implementation.

As already mentioned, the algorithm presented can achieve good performance in terms of output quality, but is somewhat time-consuming, and thus cannot be applied to every frame of the sequence. To cope with this real-time constraint, a simpler tracking algorithm [5], based on the high correlation between two subsequent frames, is currently being tested.

REFERENCES

- [1] G. Adomi, A. Broggi, G. Conte, and V. D'Andrea. "A self-tuning system for real-time optical flow detection," in *Proc. IEEE Syst., Man, Cybernet. Conf.* (Le Toquet, France), Oct. 17–20, 1993, pp. 7–12.
- [2] C. Arcelli, L. Cordella, and S. Levialdi, "Parallel thinning of binary pictures," *Electron. Letts.*, vol. 11, pp. 148–149, 1975.
- [3] A. Blake and A. Zisserman. *Visual Reconstruction*. Cambridge, MA: MIT Press, 1987.
- [4] A. Broggi *et al.*, "PAPRICA," in *CAD and Architectures: Reports on Architectures and Algorithms for VLSI Design*. Rome: CNR, 1990, pp. 21–141.
- [5] A. Broggi and V. D'Andrea, "A multiresolution algorithm for feature tracking in image sequences," in *Proc. 7th IAPR Int. Conf. Image Anal., Processing* (Bari, Italy), Sept. 20–22, 1993, pp. 218–221.
- [6] A. Broggi, V. D'Andrea, and F. Gregoretti, "A low-cost parallel VLSI architecture for low-level vision," in *MVA'92 - IAPR Workshop Machine Vision, Applicat.* (Tokyo), 1992, pp. 11–15.
- [7] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-8, pp. 679–698, Nov. 1986.
- [8] H. Freeman and L. Davis, "A corner-finding algorithm for chain-coded curves," *IEEE Trans. Comput.*, vol. COM-26, pp. 297–307, 1977.
- [9] S. Gazit and G. Medioni, "Multi-scale contour matching in a motion sequence," in *Proc. DARPA Image Understanding Workshop* (San Mateo, CA), May 1989. Los Altos, CA: Morgan Kaufmann.
- [10] D. Geiger and F. Girosi, "Parallel and deterministic algorithms for Markov random fields: Surface reconstruction and integration," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, pp. 401–412, 1991.
- [11] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions and Bayesian restoration of images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 721–741, 1984.
- [12] F. Gregoretti *et al.*, "The PAPRICA SIMD array: Critical reviews and perspectives," in *Proc. ASAP'93 - IEEE Comput. Soc. Int. Conf. Applicat. Specific Array Processors* (Venezia, Italy), Oct. 25–27, 1993, pp. 309–320.
- [13] W. D. Hillis, *The Connection Machine*. Cambridge, MA: MIT Press, 1985.
- [14] D. Mumford and J. Shah, "Boundary detection by minimizing functionals," in *Proc. IEEE Conf. Comput. Vision, Pattern Recognit.* (San Francisco), 1985.
- [15] P. Perona and J. Malik, "Scale space and edge detection using anisotropic diffusion," in *Proc. 1987 IEEE Comput. Soc. Workshop on Computer Vision*, Nov. 30–Dec. 2, 1987, pp. 16–22; also *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 12, no. 7, pp. 629–639, July 1990.
- [16] R. M. Loughheed, D. L. McCubbrey, and S. R. Sternberg, "Cytocomputers: Architectures for parallel image processing," in *Workshop Picture Data Description*, 1980.
- [17] P. Saint-Marc and G. Medioni, "Adaptive smoothing for feature extraction," in *Proc. Image Understanding Workshop* (Boston), Apr. 1988, pp. 1100–1113.
- [18] P. Zamperoni, *Metodi dell'elaborazione digitale di immagini*. Milan: Masson, 1990.