Computer Vision and Image Understanding 114 (2010) 1164–1179

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

A real-time versatile roadway path extraction and tracking on an FPGA platform

Roberto Marzotto [a], Paul Zoratti [b], Daniele Bagni [c], Andrea Colombari [a], Vittorio Murino [a,d,e,*]

[a] eVS embedded Vision Systems S.r.l., Ca' Vignal 2, Strada Le Grazie 15, 37134 Verona, Italy
[b] Xilinx, Inc., Detroit, USA
[c] Xilinx, Inc., Milan, Italy
[d] Dipartimento di Informatica, University of Verona, Ca' Vignal 2, Strada Le Grazie 15, Verona, Italy
[e] Istituto Italiano di Tecnologia (IIT), Via Morego 30, 16163 Genova, Italy

ARTICLE INFO

Article history:
Received 26 January 2009
Accepted 17 March 2010
Available online 4 May 2010

Keywords:
Lane detection
Lane tracking
Road modeling
Lane departure warning
FPGA processing
Embedded computer vision
Automotive

ABSTRACT

This paper presents an algorithm for roadway path extraction and tracking and its implementation in a Field Programmable Gate Array (FPGA) device. The implementation is particularly suitable for use as a core component of a Lane Departure Warning (LDW) system, which requires high-performance digital image processing as well as low-cost semiconductor devices, appropriate for the high volume production of the automotive market. The FPGA technology proved to be a proper platform to meet these two contrasting requirements. The proposed algorithm is specifically designed to be completely embedded in FPGA hardware to process wide VGA resolution video sequences at 30 frames per second. The main contributions of this work lie in (i) the proper selection, customization and integration of the main functions for road extraction and tracking to cope with the addressed application, and (ii) the subsequent FPGA hardware implementation as a modular architecture of specialized blocks. Experiments on real road scenario video sequences running on the FPGA device illustrate the good performance of the proposed system prototype and its ability to adapt to varying common roadway conditions, without the need for a per-installation calibration procedure.

© 2010 Elsevier Inc. All rights reserved.

1. Introduction

The huge population of motor vehicles and the related large number of accidents occurring on the roads have a dramatic impact on the cost to society in terms of medical expenses and lost productivity. Since a high percentage of accidents are mainly caused by human errors (due to driver distraction or other factors), Driver Assistance (DA) systems that support drivers and help preventing their mistakes can be effective in reducing such accidents and associated fatalities. Therefore, there is strong interest from many industrial organizations and research institutes to improve automotive safety through DA system development, which is considered a very challenging task from both scientific and technological points of view.

This paper presents a computer vision system prototype for roadway path extraction and tracking, which is the result of a project conducted between engineers at Xilinx and Embedded Vision Systems. The goal of the project was to implement a video processing algorithm suitable for use as a core component of a Lane Departure Warning (LDW) system in an embedded device, with a focus on achieving overall high-performance, low-cost and short development time [1].

The objective of an LDW system[1] is to alert the driver when the vehicle inadvertently strays from its road lane. Typically, it uses a forward-looking camera installed on the front of a vehicle to capture images of the roadway to identify and track the markings corresponding to the lane boundaries and to locate the host vehicle's position with respect to them. When the vehicle crosses the lane bounds, the system issues a warning.

Since LDW systems require high-performance digital image processing as well as low-cost devices appropriate for the high volumes of the automotive market, we decided to develop and implement the lane detection and tracking procedures in a single Field Programmable Gate Array (FPGA) device, which represents a suitable platform to meet these two contrasting requirements [2].

This choice is motivated by several reasons. First, FPGAs for automotive applications can also cope with the demanding requirements requested by automotive safety systems, in terms of harsh working conditions and type of redundancy needed. Actually, they can work at high temperature range, e.g., Xilinx Automotive devices are available in two temperature grades, up to 100 degrees and up to

* Corresponding author at: Dipartimento di Informatica, University of Verona, Ca' Vignal 2, Strada Le Grazie 15, Verona, Italy. Fax: +39 045 802 7068.
E-mail address: vittorio.murino@univr.it (V. Murino).

[1] In particular, authors are referring to camera-based LDW systems working in the visible range. There also exist LDW systems based on infrared technology and other types of sensors.

1077-3142/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.cviu.2010.03.015

125 degrees junction temperature, and are all AEC-Q100 qualified to meet automotive performance. Also, Safety Integrity Level (SIL) is an important aspect which is integrated in FPGAs: for instance, for Xilinx FPGAs in particular, several methods or features can be leveraged to meet a system wide SIL level, like the inherent parallel processing capability that can support double or triple mode redundancy and the continuous configuration monitoring done automatically in the background. These features can all be used in combination by system designers to ensure system level integrity.

In addition, FPGAs offer:

- low-power performance and design tools to be "power consumption aware" [3];
- the possibility to reconfigure the algorithm as various roadway conditions and scenarios are encountered;
- great flexibility in manipulating the algorithm with high abstraction level design and synthesis tools, besides the popular Hardware Description Language (HDL) method;
- large scalability to add more enhanced features such as road sign recognition or intelligent headlamp control.

The last two features are particularly important advantages that FPGAs offer over competing Application-Specific Standard Product (ASSP) or Application-Specific Integrated Circuit (ASIC) devices. While ASSP/ASIC technology might reduce the price production cost per piece, this only makes sense once the design has reached its full maturity. In fact, in most cases, early system development phase requires extensive and iterative coding and testing, and FPGAs represent a good trade-off among flexibility of development, possibility of reconfigurability and testing the algorithms, real-time performances, and costs. The processing flexibility offered by the hardware parallelism and programmability of an FPGA is unmatched by the fixed functionality of ASSP/ASIC and is a critical factor in risk reduction of system development. Driver Assistance is still an emerging market and new algorithms/methods are being invented as engineers gain more experience with the incredibly wide variety of roadway scenarios in which such systems must operate. An FPGA implementation allows design engineers to add/modify processing algorithms much later in the development cycle than traditional ASSP/ASIC implementations, thereby reducing development risk. Furthermore, as semiconductor processing nodes continue to decrease, now heading towards 32 nm, ASSP and ASIC development costs are rising exponentially, increasing required volumes to support development cost amortization. A related FPGA advantage is the ability to scale the density of devices (and therefore the cost) based on the feature set to be provided.

The development of the proposed prototype consisted in tackling three main issues:

- the extraction of lane marking candidates from the edge map while discarding spurious edges;
- the fitting of a proper roadway model to the extracted data;
- the tracking of the model parameters to get a more robust and reliable outcome.

To solve these issues, an image pre-processing pipeline was designed to extract the lane marking candidates from the input images, and model fitting and tracking are subsequently performed by a module that estimates the road model parameters as seen from the vehicle.

The image pre-processing pipeline consists of various stages of 2D filters: (a) Gaussian noise reduction, (b) histogram stretching, (c) edge-detection devoted to compute gradient magnitudes and phases, (d) edge thinning with automatic thresholding generating a binary edge map, (e) morphological filtering to clean the resulting data, and, finally, (f) lane marking pattern search aimed at selecting a subset of edge points having a particular configuration and orientation, thus effectively facilitating the subsequent lane extraction.

The model fitting and tracking subsystem has the purpose of estimating the road lane model: a cascade of two RANdom SAmple and Consensus (RANSAC) stages is applied to robustly fit the parameters of a parabolic model on the extracted lane marking data, followed by Kalman filtering to track them in real-time.

Overall, the main contributions of this work lie in (i) the design of a complex computer vision system as a combination and integration of algorithms working both in the pixel and feature domains, and (ii) their proper customization and tuning for an FPGA implementation, which required the design of original solutions aimed at balancing the computational burden, the functional performances, and the usage of the device resources.

The proposed system architecture is composed of self-contained logic modules that need no support from programmable micro-controllers, DSP processors or external memories: all the modules are completely implemented inside the FPGA. This is a very important novelty, making the proposed system quite unique because not only traditional low-level image processing algorithms, but also high-level fitting and tracking strategies have been designed and implemented in FPGA. Typically, other video processing systems make only a partial use of the FPGA devices, combining them with other external components like DSP processors [4]. In fact, to the best of the authors' knowledge, only the work proposed in [5] utilizes a solution completely embedded in an FPGA, similar to ours, but the approach is quite different. It implements a soft-CPU that runs simplified C/C++ code, hence not exploiting the customization property of the FPGAs and their capability to optimize parallel processing.

Furthermore, the adoption of the model-based design [1] offered an efficient and straightforward method for transitioning from a purely software model to a real-time FPGA-based hardware prototype, without the need of complex HDL coding. In contrast to the relatively slow processing of the pure software model, the FPGA implementation supports processing wide VGA (WVGA) images at 30 frames per second.

It is also worth noticing that the described system does not require an *ad hoc* camera calibration for each installation, but only few parameters have to be initially fixed depending on some non-stringent hardware specifications on the camera setup.[2]

During the experimental phase of the project, the system proved to be robust and suitable to cope with several road conditions as reliable results have been obtained even when roads were cluttered, shadows were present, and lane markings were unclear or partly masked.

The remainder of the paper is organized as follows: Section 2 outlines the previous work for lane detection and tracking from a point of view oriented to the target FPGA implementation, and consequently enriched with proper considerations. Section 3 describes in detail the proposed solution starting from the adopted road model: for each stage of the system, the characteristics of the implemented algorithms are also reported as compared to the current literature. Section 4 is devoted to describe the FPGA implementation and the specific issues faced during this phase. Section 5 illustrates the results using actual freeway image sequences, and a critical analysis is provided to show performances and limitations. The work is summarized and conclusions are reported in Section 6.

2. Related work

Several algorithms for roadway path extraction and tracking have been proposed in the last decade [6–26], whose differences

[2] Values for camera height from the road plane, focal length, dimensions of the CCD, and tilt angle are sufficient.

mainly consist in the adopted image pre-processing pipeline, the lane model, the selected model fitting method and the tracking strategy.

In the literature, only few systems use FPGA devices or other low-cost and low-power consumption architectures. For example, the system called GOLD [6] is based on the PArallel PRocessor for Image Checking and Analysis (PAPRICA) co-processor which is a low-cost special-purpose massively parallel architecture composed of 256 slow clocked processing elements integrated on a single VME board connected to a SPARC-based architecture. In [7], in order to reduce the computational burden of the main processor, an FPGA co-processor is employed for some low-level tasks like pre-filtering and sub-sampling. Similarly, the LDW system proposed by TRW[3] uses the FPGA only for edge detection and feature extraction. In [17], lane detection is carried out using 3D data extracted by a hardware–software co-designed system implemented in FPGA, while the rest of the processing is performed by an embedded processor. Another example can be found in [32], where an embedded portable LDW system based on an ARM processor and FPGA is proposed. In this architecture, FPGA is exploited for the image pre-processing (low-pass filtering and edge detection) and data transfer among the memory bus, the sensor device, and the LCD display. The rest of the processing (lane detection and lane departure warning mechanism based on the spatio-temporal procedure) is carried out by the processor in software. In [33], a real-time algorithm for line keeping is presented, which is implemented on a Celoxica RC203 board equipped with a Xilinx Virtex II FPGA. The steering angle estimation (actualy, no tracking is performed) is based on the detection of the lane markings via road segmentation using adaptive histogram thresholding. The performances are quite good in terms of speed (39 frames per second) and steering angle accuracy on roads freed by nearby cars. Finally, as already quoted in Section 1, [5] utilizes a solution completely embedded in an FPGA but, implementing a soft-CPU core as main component, it suffers from significant limitations in exploiting the capability to optimize the parallel processing.

Regarding the individual processing stages, many algorithms have been proposed in the past. A class of methods for image pre-processing usually computes an edge map of lane marking candidate points [9–11,15,18–21,25,26], which is subsequently input to the model-based fitting to detect the road model. The edge map can be obtained using standard methods such as Canny [11,15,18] or Sobel [9,10,12,13], or by custom methods based on dark–light–dark transition detection [6,7,22,26]. Other algorithms prefer to fit the model directly on the whole gradient map by maximizing a likelihood function [8,24]. In order to prune out spurious edge points, some constraints can be exploited. For example, to detect plausible directions of the lane markings, steerable filters are used [19], which also helps to filter out edges with disqualifying gradient directions. The same idea is used in [10,17], where the Hough Transform accumulator space is reduced around plausible values. In [21,26], left and right lane markings candidate points are discriminated by using the gradient sign of the edge pixels. Lane marking width [23,26] together with or in alternative to road and lane marking colors [12,15,23] can be used for pixel classification purposes. Typically, in order to obtain a better edge map, edge detection can be preceded by a Gaussian filter to reduce noise [15], or followed by edge thinning stage to get more accurate contours [9]. In [6,19], the image is first warped to get a bird-eye view in order to subsequently exploit the parallelism between lane markings using various heuristics. This constraint can also be exploited without the time consuming warping stage at the Euclidean coordinates level using projection formulas [9,16]. Notice that, despite good performances, the methods using image or coordinates' projection require sensor calibration which can be problematic in real-world automotive applications.

The lane model may vary from a simple one to a complex one: straight-lines [10,15,17,18], piece-wise linear [9,23], parabolic approximation on the flat (road) plane [8,12,13,19–21,24,27], cubic B-splines [11], cubic splines [25], and clothoid [7,14,16,22,26]; the latter being a 3D model which should be combined with stereo systems (also requiring accurate calibration themselves). Obviously, more complicated models require more processing resources for the fitting stage because of the larger number of parameters to be estimated. Since the final goal is a low-cost implementation on an FPGA platform, the decision about which model to adopt is critical. In the authors' opinion, the model with the best trade-off between modeling capabilities and performance impact is the parabolic approximation.

Once the road model has been selected, the data-model fitting can be carried out in many ways. A possible class of methods is constituted by the Hough Transform (HT) [10,11,15,17], and its variants like the Randomized Hough Transform (RHT) [24] and the Adaptive Randomized Hough Transform (ARHT) [12]. Alternatives to HT-based techniques are the RANSAC algorithm and its variants [21,25], the minimization of a target function [11,26], or exploiting probabilistic approaches [8,20,24,25].

RHT implements HT as a series of random trials. As elucidated in [28], RHT is similar to RANSAC, but the main difference is that RANSAC does not use score accumulation in the parameter space, so it is not appropriate for finding multiple model instances but just the most voted. ARHT is a variant of the RHT inspired by particle filtering where all the image points are considered in the random sampling stage, but they are weighted using gradient information. Moreover, ARHT also combines a multi-resolution strategy to increase accuracy. In order to have a more robust fitting, an interesting constraint is given by vanishing points: the parallelism between lane markings on the ground corresponds to a vanishing point constraint in the image plane. For example, this constraint has been used in [10,18] to reduce the HT accumulator space, and in [11] to compute vanishing point candidates, subsequently used for line classification purposes.

In the proposed work, considering that the final algorithm is to be implemented in FPGA, effective but simple[4] fitting and tracking procedures were chosen. For model fitting, both probabilistic and minimization-based approaches, as well as ARHT were discarded because they were considered too computationally expensive. The storage requirements of the classical HT can be a limit for an FPGA implementation, so it was also discarded. Though the improvements offered by RHT (small storage requirements, improved performances, and higher parameter resolution) seem suitable for being embedded in an FPGA, RHT needs dynamic accumulator construction but dynamic memory allocation is difficult to be implemented in an FPGA.

Consequently, the final choice for the model fitting algorithm in the proposed prototype is a RANSAC-like approach due to the following advantages: (i) it is not memory consuming, (ii) it does not need a peak detection stage, (iii) it is robust to outliers, (iv) it is flexible as it can be adapted to different lane models without radically changing the implementation, and (v) its processing time can be trimmed by changing the number of iterations.

Model tracking, when present, is usually performed by a Kalman Filter (KF) [17,19,26], an Extended Kalman Filter (EKF) [7,14,22] or a Particle Filter (PF) [15,25,20]. Some approaches do not differentiate between model fitting and tracking, but they use a single global tracking stage able to directly relate image data

[3] http://www.trw.com/.

[4] Simple in terms of number and type of the operations, in particular floating point usage and memory consumption are critical from the point of view of resource occupation.

to model parameters. For example, [7,14,22] use an EKF to directly relate lane marking points to the clothoid model parameters. In some cases, e.g., [19], vehicle state information is also combined with the extracted data for tracking purposes but this obviously implies some sort of calibration. Always keeping in mind the FPGA as target platform, the selected tracking strategy was the traditional KF adapted to the chosen model: compared to an EKF or a PF, it needs less floating point operations.

In summary, as described above, a lot of algorithms for lane detection and tracking have been proposed in the past and they differ in many ways from the low-level filtering to the high-level model fitting and tracking, also employing in a variety of hardware or hybrid hardware/software architectures. Nevertheless, the literature does not offer evidence of image/video processing analysis systems entirely implemented in a single FPGA platform, as the pipeline of logic blocks proposed in this work.

Actually, our proposal does not care about particular "special" hardware configuration to cope with specific features of the algorithms: our hardware reference is the FPGA and the best functional performances (i.e., lane detection and tracking) at the maximum speed constitute the target of the work. The advantage of having all of the video processing on a single FPGA centers on: (1) the availability of the parallel processing to perform pixel-level image processing functions resulting in robust system performance, (2) system reliability (via fewer components – e.g., FPGA-based design does not use external RAM), and (3) reduced system level cost (including PC-board complexity and power supply design).

3. The proposed method

This section describes in detail the several stages of the system. Starting from the mathematical formulation of the adopted road model, the image pre-processing stage is described, explaining the various steps involved. The road model fitting based on RANSAC is then illustrated, focusing on the algorithm customization and the heuristics adopted. Finally, the model tracking stage based on Kalman filtering is reported.

3.1. Road model

In 1994, Kluge [27] proposed a road model that is used in other more recent work [8,12,21,24] and has been adopted in this project too. The model is an approximation of a clothoid with no vertical curvature and with constant horizontal curvature. It assumes that pavement edges and lane markings can be approximated by circular arcs on a flat ground plane. For small to moderate lane curves, a circular arc can be approximated by a parabola in the *object space*. The object space is the reference frame fixed with the vehicle on the road plane (bird-eye view) where the x axis points in the direction of the running vehicle. The object space differs from the *image space* which is the reference frame of the pixel coordinates in the image: (x, y) represents a point in the object space, while (r, c) or (u, v) represents a point in the image space.

Note that the geometry of this model, shown in Fig. 1, assumes that the camera optical center is located on the z axis and that the camera view direction intersects the x axis, i.e., panning angle equal to zero, and the tilt angle θ is assumed being different from zero. The reference frame (r, c) in the image space (u, v) is fixed so that $r = 0$ corresponds to the horizon row $v = v_0$, and $c = 0$ corresponds to the central column $u = u_0$ (or to the vanishing point column to compensate for little panning angles).

The parabolic lane model incorporates position, angle, and curvature, which are the parameters to be estimated. Lane width and horizon row are assumed *locally* constant, but they will be updated and tracked using Kalman filtering to provide model flexibility

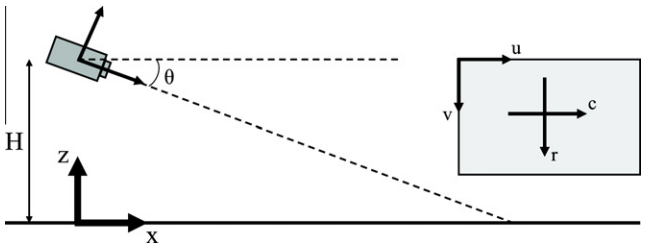

Fig. 1. Relationship between object and image spaces.

(Section 3.4). The parabolic model in the object space is illustrated in Fig. 2.

The following equation defines the parabolic model of the lane in the object space:

$$y = 0.5 \times k \times x^2 + m \times x + b \tag{1}$$

where k is the curvature, m is the angle and b is the position. Left and right lines are assumed to be parallel so that they have the same k and m parameters whilst only b differs. Therefore, the lane model is characterized by four parameters k, m, b_L, and b_R in the object space.

Note that b_L and b_R are related to the camera location with respect to the lane markings. More precisely, their absolute values are the distances between the camera and the left and right lane markings, respectively. On one hand, if the forward-looking camera is located in the center of a car placed in the middle of the lane, b_L and b_R have the same absolute value but opposite sign; on the other hand, if the camera is closer to the left lane marking $|b_L|$ decreases and $|b_R|$ increases, and if the camera crosses the left line both b_L and b_R have the same sign. This could be exploited for lane departure warning purposes. Moreover, the difference between b_L and b_R corresponds to the lane width which is assumed to be locally constant.

In the non-tilted camera case, i.e., $\theta = 0$, the horizon corresponds to the central row. Hence, object space points (x, y) and image space points (r, c) are related by the following equations:

$$x = \frac{H}{r \times r_f} \tag{2}$$

$$y = c \times c_f \times x \tag{3}$$

where H is the height of the camera with respect to the ground plane, r_f and c_f are the height and width of the pixel, i.e., the effective pixel dimensions divided by the focal length.

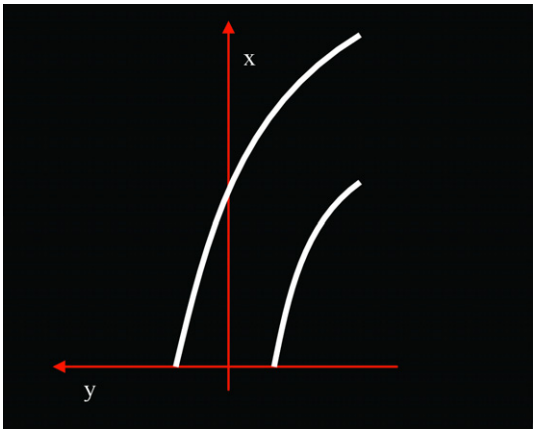

Fig. 2. Schematic representation of the object space.

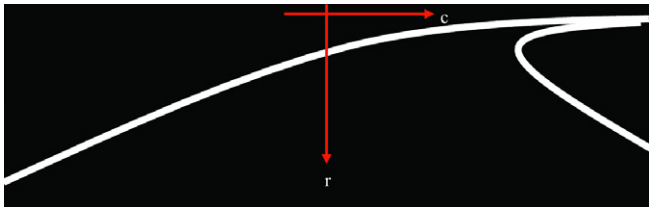

Fig. 3. Schematic representation of the image space.

Substituting Eqs. (2) and (3) into Eq. (1), after some algebraic manipulations the following relationship can be obtained

$$c = K\frac{1}{r} + B_{L(R)}\, r + M \tag{4}$$

where

$$K = \frac{0.5 \times H}{r_f \times c_f}\, k, \quad B_{L(R)} = \frac{r_f}{H \times c_f}\, b_{L(R)}, \quad \text{and} \quad M = \frac{1}{c_f}\, m$$

Eq. (4) is valid even in the tilted camera case [27], though the derivation of the proper equations for K, M, B_L and B_R is more complicated. Therefore, the parabolic model in the object space becomes a hyperbolic model in the image space (see Eq. (4) and Fig. 3) which is described by a set of four parameters $\mathscr{M} = \{K, M, B_L, B_R\}$.

$B_{L(R)}$ and $b_{L(R)}$ differ just for a scale factor, therefore in both fitting and tracking stages the following constraint can be used, similarly to [18]:

$$w = \frac{B_L - B_R}{\beta}, \quad \text{where} \quad \beta = \frac{r_f}{H \times c_f} \tag{5}$$

Moreover, as mentioned before, b_L and b_R are related to the location of the vehicle within the lane. So, a lane departure warning could be generated when $|b_L|$ or $|b_L|$ is less than a threshold T_b, e.g., when the following condition occurs:

$$\left|\frac{B_L}{\beta}\right| < T_b \quad \text{or} \quad \left|\frac{B_R}{\beta}\right| < T_b \tag{6}$$

Parameters like H, r_f, c_f, v_0 and u_0 depend on the camera location, but the proposed method does not need a per-installation calibration procedure: all what is needed is the knowledge of the above mentioned specifications.

3.2. Pre-processing pipeline

The various stages of the image pre-processing subsystem are shown in Fig. 4. The purpose of this subsystem is to extract roadway image points that most likely represent lane markings. The pipeline consists of several steps that can be summarized as follows:

1. cropping of a Region Of Interest (ROI) corresponding to the road area which is the image portion under the horizon row;
2. 2D 5×5 FIR filter to reduce the Gaussian noise and to improve the performance of the edge detection with respect to noise;
3. histogram stretching to enhance the contrast of the image by exploiting as much as possible the entire grey-level range;
4. horizontal/vertical gradient computation via convolution with a 2D 5×5 Sobel kernel;
5. gradient magnitude and phase calculation;
6. edge-detection and thinning to determine which points are edges by thresholding the gradient magnitude with an automatic threshold computation based on the cumulative histogram and applying non-maxima suppression to generate thin contours (one-pixel thick);
7. lane marking pattern search (LMPS), a sort of "smart" image filtering that selects a subset of edge points having a proper configuration removing spurious edge points due to shadows, other passing vehicles, trees, signs and other details in the scene;
8. 2D 3×3 morphological filtering for the final cleaning of the lane marking candidates map.

Stages 2–6 and 8 are standard image processing operations, whereas 7 has some uniqueness that needs to be highlighted.

The LMPS algorithm scans the edge points row-by-row from the top-left to the bottom-right and identifies an edge point P_i as a lane marking candidate point if all of the following four conditions are simultaneously true:

1. For each (candidate) edge point P_i on the left side of a lane marking there is a corresponding (conjugate) edge point P_{i+1} on the right side (along the same scan line), as illustrated in Fig. 5.
2. The distance between P_i and P_{i+1} has to be less than the lane marking width W_r; due to the perspective, W_r varies with the image row index r.
3. The gradient phase of P_i has to be in the interval $[\alpha, \pi - \alpha]$, where α is a threshold on the maximum inclination of the line ($\alpha = 15°$).
4. The phase of P_{i+1} should be approximately the opposite of the phase of P_i, as shown in Fig. 6.

Note that for each candidate-conjugate pair (P_i, P_{i+1}) satisfying the four mentioned conditions, only P_i is selected as a lane marking candidate and so the LMPS output corresponds to the left boundary of a painted marking.

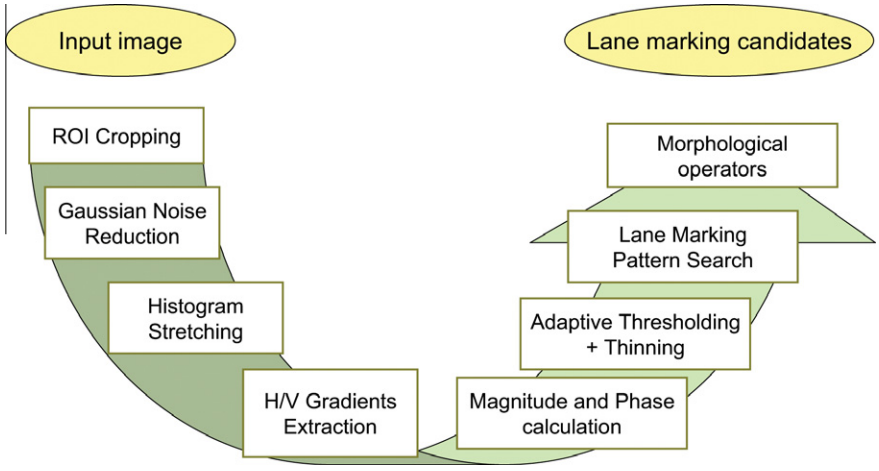

Fig. 4. The pre-processing pipeline.

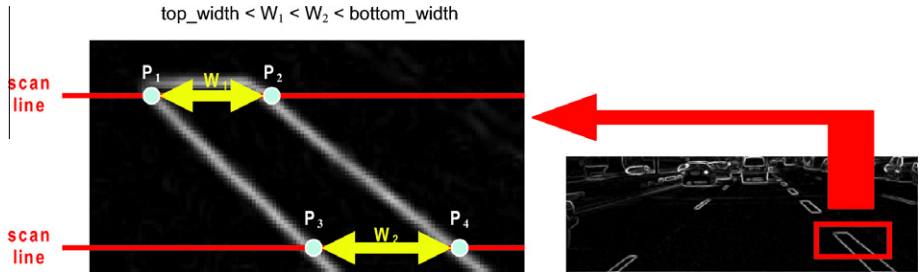

Fig. 5. Distance between candidate and conjugate points.

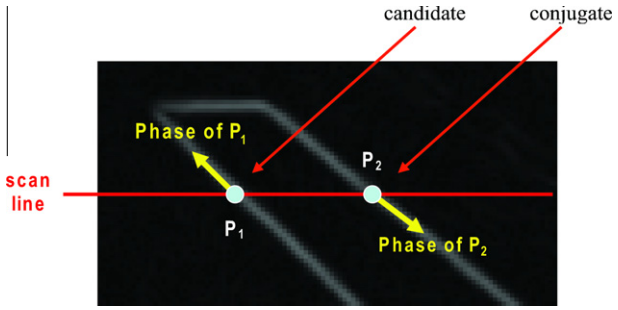

Fig. 6. The candidate point phase angle is about the opposite of the conjugate point phase angle.

In summary, only four parameters need to be defined for the LMPS algorithm: two marking width limits (corresponding to the top and the bottom rows of the ROI) that allow the line width W_r to be computed by linear interpolation for each row r in the ROI (condition 2); the threshold α on the edge point phase (condition 3); a tolerance to evaluate if two points have approximately opposite phase (condition 4).

The authors' former work [1] gives some more details on the first stage of the image pre-processing pipeline, namely the 2D GNR FIR filter, with particular emphasis on the methodology with a high abstraction level design tool requiring no HDL knowledge to the user.

3.3. Model fitting

Using the model $\mathscr{M}$ described in Section 3.1, the lane detection problem can be reduced in estimating the four parameters K, M, B_L, and B_R. To enhance the flexibility of the model with respect to the original one [27], two further parameters are allowed to smoothly change between consecutive frames: the horizon row v_0 and the lane width w. Adding these parameters allows the model to manage roads with small and gradual changes of slope and roads where the distance between the lane markings smoothly changes. Hence, the final model is $\mathscr{M} = \{K, M, B_L, B_R, v_0, w\}$.

The input of the model fitting algorithm is given by a binary map where the foreground pixels are the lane marking candidate points (LMPS output), and the map of the gradient phase.

Each lane marking candidate point P_i corresponds to a triplet (r_i, c_i, gd_i) where r_i and c_i are the coordinates of the point and gd_i is the gradient direction, i.e., the local feature tangent derivable from the gradient phase.

The model fitting algorithm is a based on the RANSAC approach [29] which can be summarized as follows. Given a model that requires a minimum of m data points to instantiate its free parameters, and a set of data points S containing outliers:

1. *Hypothesis formulation*: randomly select a subset of m points from S and instantiate the model from this subset.
2. *Voting*: determine the *consensus set* S_k of data points that are within an error tolerance T of the model.
3. *Solution refinement*: if the size of S_k is greater than a threshold N_I (minimum number of inliers), re-estimate the model (possibly via least-squares) using S_k (the set of inliers).
4. If the size of S_k is less than N_I, repeat from step 1.
5. Repeat 1–4 for N_T trials and choose the largest consensus set found.

The following main differences from the standard formulation of RANSAC [29] make the chosen model fitting approach particularly suited for an FPGA implementation:

- *Lack of the refinement step*. Since fitting results are then smoothed by a tracking stage, tracking is assumed to be enough to refine the solution because of temporal consistency. Therefore, the fitting result will be given by the model with the largest consensus set.
- *Alternative voting scheme*. Similarly to [20,21], in the proposed voting scheme the original RANSAC 0-1 voting logic is replaced by one that takes into account the closeness of a point to the model. Even if a tolerance T for inlier/outlier discrimination is still used, the vote of each inlier is continuous in the interval [0, 1].

Hence, let $\varepsilon(P_i, \mathscr{M})$ be a distance (function) between a point P_i and a model $\mathscr{M}$, the point is classified as an inlier if $\varepsilon(P_i, \mathscr{M}) < T$, and its vote $\rho(P_i, \mathscr{M})$ is computed as follows:

$$\rho(P_i, \mathscr{M}) = e^{-\frac{\varepsilon(P_i, \mathscr{M})}{\lambda}} \tag{7}$$

where λ is a normalizing factor obtained experimentally.

- *Control of the solution space*. The solution space of the model parameters is changed run-time "around" the solution predicted by the tracking algorithm so that the fitting procedure cannot vote for solutions with low probability. In the beginning, when no information about a plausible solution is available, the solution space is initialized as big as enough for RANSAC to simply find the most voted solution with no other constraints. Once a plausible solution is established, the solution space corresponds to a proper neighborhood based on the tracker prediction (see Section 3.4 for details).

Due to perspective, the tolerance T used for inlier/outlier discrimination is not a constant value but is a function of the row $T(r)$. In order to give a proper definition of $T(r)$, the following relationship was derived from Eqs. (2) and (3):

$$\Delta c(\Delta y, r) = \frac{\Delta y \times r \times r_f}{c_f \times H}$$

and $T(r)$ was defined as

$$T(r) = \Delta c(\Delta y, r), \quad \text{with a fixed } \Delta y \tag{8}$$

Given two points (r_1, c_1) and (r_2, c_2) belonging to the lane markings, K and M parameters are related to the image gradient directions gd_1 and gd_2 by the following equations that do not contain $B_{L(R)}$ [27]:

$$K = \frac{(c_1 - c_2) + gd_1 \times r_1 - gd_2 \times r_2}{2 \times \left(\frac{1}{r_1} - \frac{1}{r_2}\right)} \tag{9}$$

$$M = c_1 - \frac{2K}{r_1} + gd_1 \times r_1 \tag{10}$$

Eqs. (9) and (10) allow to split the fitting procedure in two subsequent stages as suggested by [12,24]: two RANSAC-like fitting procedures applied in cascade to estimate first K and M, and then B_L and B_R.

3.3.1. K and M estimation

Starting from a given solution space and a set of lane marking candidate points P_i with coordinates (r_i, c_i) centered in (u_0, v_0), the algorithm performs the following operations for each iteration:

1. random sampling of a pair of points (r_1, c_1) and (r_2, c_2);
2. random picking a delta horizon value δ_h in a small set of possible values, e.g., $[-8, -4, -2, 0, 2, 4, 8]$ at wide VGA resolution 752×480; this is used to adapt the horizon row v_0 to small deviations from the current lane model as previously mentioned;
3. hypothesis formulation by calculating candidate K and M using the selected pair of points and Eqs. (9) and (10), also considering δ_h;
4. if K and M are inside the solution space then voting follows, otherwise repeat from step 1;
5. during voting each lane marking candidate point $P_i = (r_i, c_i)$ is classified as inlier if the following condition is satisfied

$$\varepsilon(P_i, \{K, M, \delta_h\}) = \left| \left(\frac{2K}{r_i - \delta_h} - gd_i (r_i - \delta_h) + M \right) - c_i \right| < T(r_i)$$

   where $T(r_i)$ is defined in Eq. (8). Therefore, if the point is an inlier, its vote is computed by using Eq. (7);
6. the model $\mathscr{M} = \{K, M, \delta_h\}$ with the largest sum of votes $\Sigma_{\mathscr{M}}$ will be the solution if its number of inliers is larger than a threshold $N_I^{K,M}$.

Since the most critical inliers for K and M are those at the top of the image where the curvature mostly biases the shape of the lane markings, the votes computation was modified in order to reduce the value of the votes at the bottom as compared to those at the top on equal residual values. This was obtained just multiplying Eq. (7) by a linear function of the row index obtaining:

$$\rho'(P_i, \mathscr{M}) = \rho(P_i, \mathscr{M}) \times \left(1 - \frac{r_i}{2 \times r_{max}}\right) \tag{11}$$

where $r_{\max}$ is the last row of the cropped ROI.

3.3.2. B_L and B_R estimation

This function outputs B_L and B_R starting from a given solution space, the parameters K, M, and δ_h computed by the previous phase, the set of lane marking candidate points $P'_i = (r'_i, c'_i)$ with respect to the new horizon $v'_0 = v_0 + \delta_h$, and the currently estimated lane width w.

For each iteration, the following operations are carried out:

1. random sampling of a pair of points (r'_1, c'_1) and (r'_2, c'_2). Notice that, similarly to [21], the two points are now selected from two different partitions of the candidate points set on the basis of the sign of the gradient direction, so that the pair will be composed of a left and a right point;
2. hypothesis formulation by calculating candidate B_L and B_R using the selected pair of points and Eq. (4);
3. compute the lane width w using Eq. (5);
4. if B_L, B_R, and w are inside the solution space, voting is performed, otherwise repeat from step 1;
5. voting is done independently for the two parameters B_L and B_R; during voting, each lane marking candidate point $P'_i = (r'_i, c'_i)$ is classified as inlier if the following condition is satisfied

$$\varepsilon(P'_i, \{K, M, B_{L(R)}, v'_0, w\}) = \left| \frac{K}{r'_i} + r'_i B_{L(R)} + M - c'_i \right| < T(r'_i)$$

   where $T(r'_i)$ is defined in Eq. (8). Therefore, if the point is an inlier, its vote is computed by using Eq. (7);
6. the model $\mathscr{M} = \{K, M, B_L, B_R, v'_0, w\}$ obtaining the largest sum of votes $\Sigma_{\mathscr{M}}$ will be the solution if the number of inliers is larger than a threshold N_I^B.

There are three particular cases in which only one B (B_L or B_R) can be obtained:

- if one of the two partitions defined in the random sampling step is not enough populated;
- if only B_L or B_R is in the solution space;
- if the constraint on the lane width w fails.

In the first case, points are picked from the bigger partition only; in the second case, only the parameter in the solution space is considered for further processing; in the third case, only the most voted parameter is considered valid.

3.4. Model tracking

The tracking algorithm is composed of three independent KFs working respectively on K, M, and on the quadruple $\{B_L, B_R, h, w\}$. A constant velocity motion model is used for K, M, B_L and B_R, whereas h and w are modeled as constant values. Moreover, Eq. (5) allows to suppose that B_L and B_R vary together at the same velocity. The notation used in this section is coherent with the one used in [30].

The used KFs assume no control input, so that the true state $\mathbf{x}$ of the i-th KF at time k is evolved from the state at instant $(k-1)$ according to

$$\mathbf{x}_{i,k} = \mathbf{F}_i \mathbf{x}_{i,k-1} + \boldsymbol{\omega}_\mathbf{i} \tag{12}$$

where $\mathbf{F}_i$ is the state transition matrix of the i-th KF and $\boldsymbol{\omega}_\mathbf{i}$ is the correspondent noise process which is assumed to be uniformly distributed as $\boldsymbol{\omega}_\mathbf{i} \sim N(0, \mathbf{Q}_i)$. At time k, an observation $\mathbf{z}_{i,k}$ of the i-th KF is modeled according to

$$\mathbf{z}_{i,k} = \mathbf{H}_i \mathbf{x}_{i,k} + \boldsymbol{v}_{i,k}$$

where $\mathbf{H}_i$ is the observation matrix of the i-th KF and $\mathbf{v}_k$ is the correspondent observation noise which is assumed to be distributed as $\boldsymbol{v}_k \sim N(0, \mathbf{R}_k)$.

As the used notation suggests, the observation covariance matrix $\mathbf{R}_{i,k}$ changes run-time whereas matrices $\mathbf{F}_i$, $\mathbf{Q}_i$, and $\mathbf{H}_i$ are assumed to be a priori fixed. In fact, $\mathbf{R}_{i,k}$ is updated after the fitting stage by using the following equation:

$$\mathbf{R}_{i,k} = \mathbf{R}_i^{max} \left[e^{-\left(\frac{\Sigma_{\mathscr{M}}}{\gamma}\right)^2} \right]^2 \tag{13}$$

where $\Sigma_{\mathscr{M}}$ is the sum of votes of the RANSAC solution, $\mathbf{R}_i^{max}$ is a diagonal covariance matrix containing very large variance values and γ is a normalization factor. Notice that the exponential function can be seen as a probability: the lower the sum of votes, the larger the probability of obtaining a corrupted fitting. In this way, the smoothing level is proportional to the unreliability of the output of the fitting.

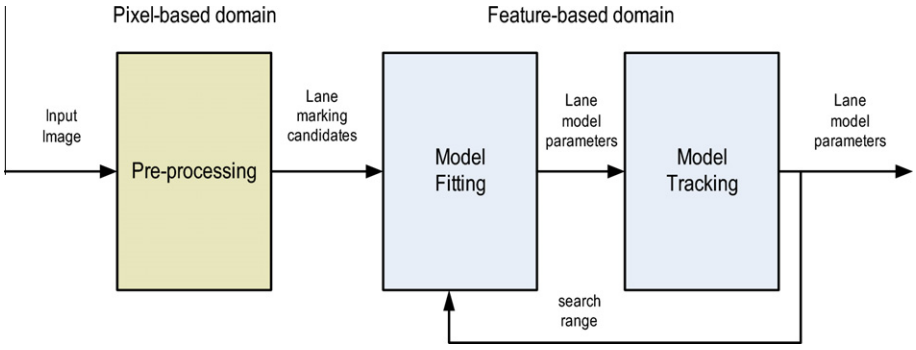

Fig. 7. Conceptual scheme of the hardware implementation.

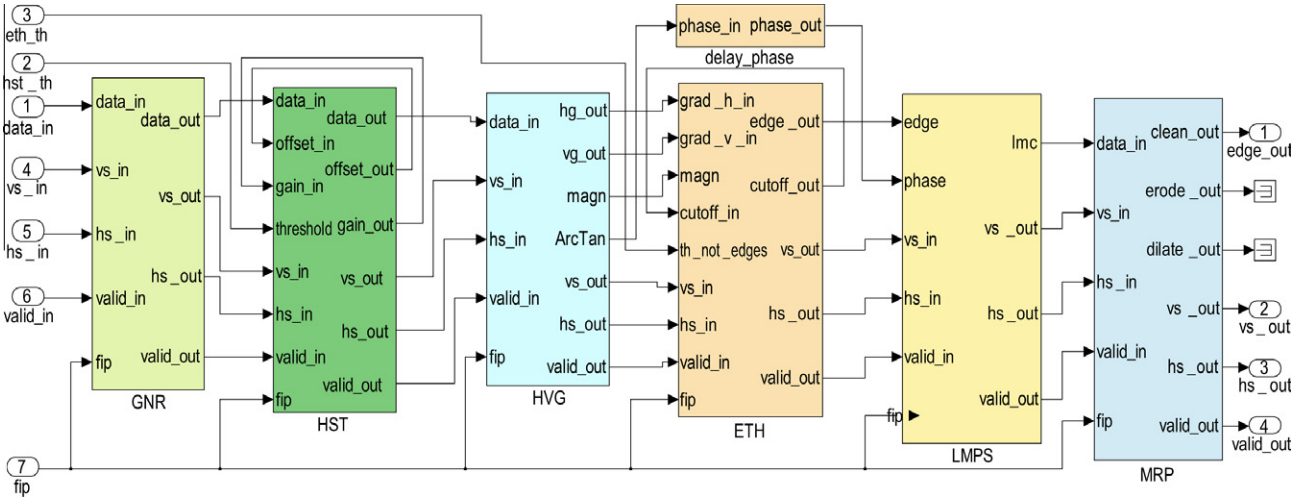

Fig. 8. System Generator for DSP top-level scheme of the **Pre-processing** block.

In particular, the state and the observation vectors are so defined:

$$\mathbf{x}_1 = \begin{bmatrix} \widehat{K} & v_K \end{bmatrix} \qquad \mathbf{z}_1 = K$$

$$\mathbf{x}_2 = \begin{bmatrix} \widehat{M} & v_M \end{bmatrix} \qquad \mathbf{z}_2 = M$$

$$\mathbf{x}_3 = \begin{bmatrix} \widehat{B}_L & \widehat{B}_R & \hat{h} & \hat{w} & v_B \end{bmatrix} \qquad \mathbf{z}_3 = [B_L \quad B_R \quad h]$$

The noise process matrices are diagonal and initialized with values experimentally determined. The state transition and observation matrices are defined as follows:

$$\mathbf{F}_{1,2} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{F}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1/\beta & -1/\beta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{H}_{1,2} = [1 \quad 0] \qquad \mathbf{H}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Besides smoothing the measurements obtained from the model fitting procedure and replacing missing measurements, tracking is also used to limit the solution space of the next model fitting stage around the KFs predictions: once the tracked parameters $\hat{\mathbf{x}}_{i,k}$ are computed, they are used to obtain an estimate of the next state $\mathbf{x}_{i,k+1}$ as follows:

$$\hat{\mathbf{x}}^-_{i,k+1} = \mathbf{F}_i \hat{\mathbf{x}}_{i,k}$$

The solution space for the next fitting is defined around $\hat{\mathbf{x}}^-_{i,k+1}$ using the measurements covariance matrices $\mathbf{R}_{i,k}$, the a-posteriori estimate error covariance matrices $\mathbf{P}_{i,k}$, and the process noise covariance matrix $\mathbf{Q}_i$: so that the larger the variance values, the bigger the solution space.

4. FPGA implementation

The entire FPGA system implementation was based on the Xilinx System Generator™ for DSP (Rel. 10.1.3), a design and synthesis tool[5] which works within the Simulink® model-based environment from The MathWorks™. System Generator benefits from the Xilinx DSP blockset for Simulink and will produce highly optimized netlists for the DSP building blocks. In System Generator for DSP the total number of bits and the binary position of every signal can be defined and manipulated as a fractional number in fixed-point arithmetic. The simulation results are cycle-accurate and bit-true and can easily be compared against floating-point reference results generated with either MATLAB® scripts or Simulink blocks to check for quantization errors.

The conceptual scheme of the overall design can be modeled as in Fig. 7. The **Pre-processing** pipeline described in Section 3.2 (and shown in Fig. 4) is illustrated in Fig. 8: all the modules are designed with a compact configuration and a common interface to provide a high level of modularity. In particular, the horizontal sync (hs), the

[5] See "Xilinx System Generator for DSP Getting Started Guide", release 10.1.3, available on Xilinx Website (http://www.xilinx.com/support/sw_manuals/sysgen_gs.pdf).

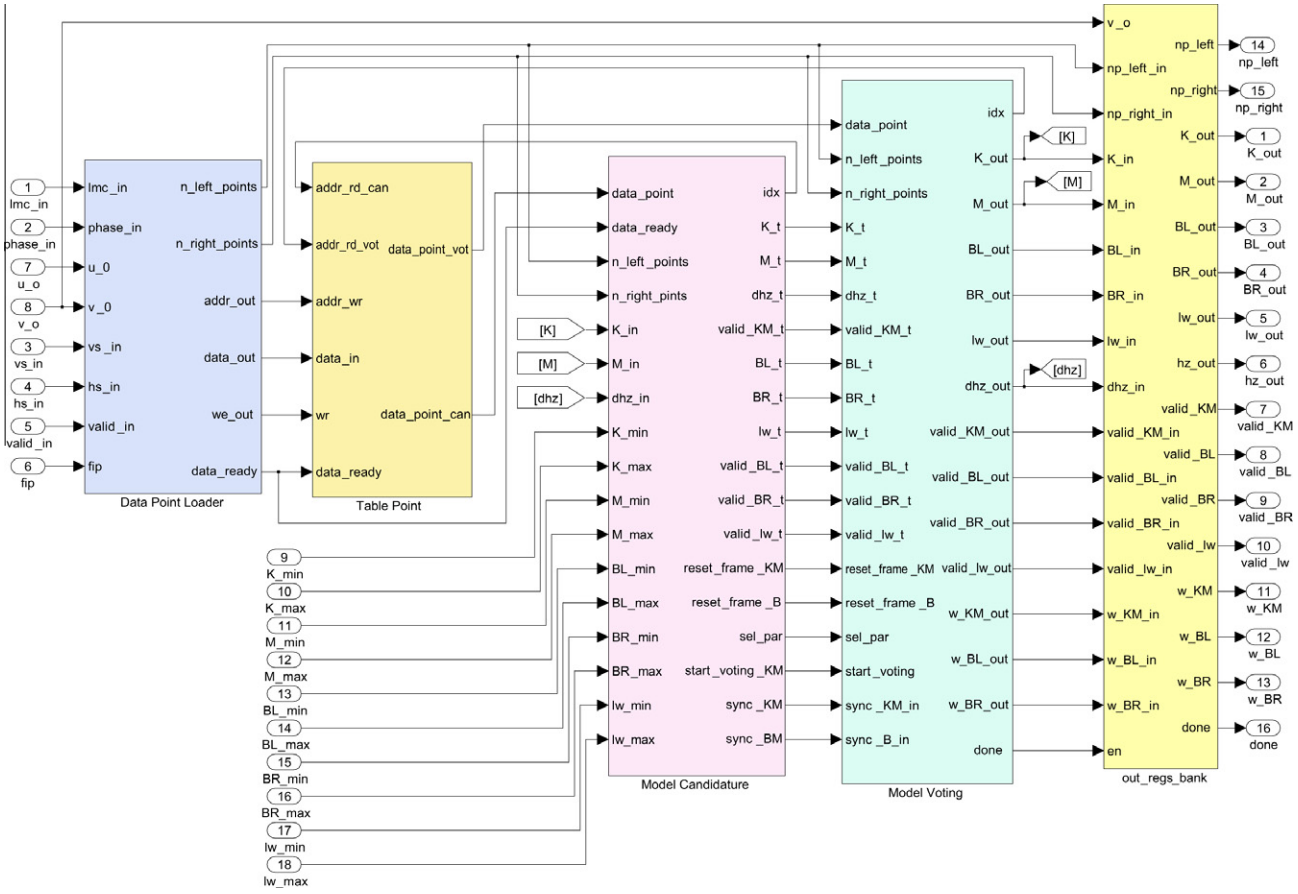

Fig. 9. System Generator for DSP top-level scheme of the **Model Fitting** block.

vertical sync (vs), and the frame init pulse (fip) are standard control signals used to specify the active area of the frame that facilitate an easy insertion in practical video systems; the valid input (valid_in) signal allows to specify the ROI within the active image area. The output of the **Pre-processing** block is a bitonal map containing only the lane marking candidate points.

The **Model Fitting** and the **Model Tracking** blocks represent the high-level processing. They do not work in the pixel domain like the **Pre-processing** block, but in a feature-based domain. In particular, the **Model Fitting** block works only on lane marking candidates points while the **Model Tracking** works on the lane model parameters.

An exhaustive description of the hardware implementation with System Generator for DSP is beyond the purpose of this paper: the level of detail is more suitable for an application note. Hence, in the following, we will mainly focus on the **Model Fitting** subsystem, whose top-level view is illustrated in Fig. 9. There are four main blocks: *Data Point Loader*, *Table Point*, *Model Candidature*, and *Model Voting*.

The *Data Point Loader* analyzes the stream of the binary map coming from the **Pre-processing** pipeline by looking at edge points and, when one is found, its coordinates and gradient direction are combined to form a tern (r_i, c_i, gd_i) which is inserted in a dedicated memory inside the *Table Point* module; this table is implemented using the internal memory of the FPGA (elementary blocks called Block RAM, or BRAM). The table depth is a-priori fixed to 1024 words.

The *Model Candidature* subsystem formulates the hypothesis (random sampling and computation of the candidate model) for both RANSAC functions and the *Model Voting* block executes the voting and finds the best solution. Once the data point table relative to the current frame is filled, the RANSAC for K and M estimation starts, followed by the RANSAC for B_L and B_R estimation. Since RANSAC samples data points from the table randomly, the pseudo-random numbers generation circuit is obtained by concatenating a battery of four bit LFSRs (Linear Feedback Shift Registers). These shift registers are basic blocks from System Generator for DSP.

The *Model Candidature* and *Model Voting* stages work in *parallel*, so that during the voting of the n-th candidate model, the algorithm can calculate the $(n+1)$-th candidate model. This strategy reduces the processing delay of the **Model Fitting** subsystem and increases the overall throughput. Note that this parallel implementation is one of the advantages offered by the FPGA.

The cumulative latency of the **Model Fitting** subsystem is equal to the time needed for filling the table plus the latency of the two RANSAC stages (which depends on the size of the table and on the number of iterations). To increase the throughput of the module, two tables are really allocated instead of one and they are involved in a "dual buffer" strategy: while the first buffer is being filled with the data points of the frame i, the second one is already filled with the data points of the frame $i-1$ and ready to be accessed by RANSAC. In this way, the new filling operation can be carried out without waiting for the end of the previous data processing.

The *Model Candidature* subsystem is responsible for computing the lane model parameters K and M from a pair of randomly selected points. Fig. 10 shows how the various System Generator for DSP blocks are connected together in the scheme of the K and M estimation module. Eqs. (9) and (10) were manipulated in order to minimize the resource occupation, the latency of the circuit, and the loss of accuracy. Note that only one division is needed and it is implemented using one multiplier and a look-up-table of pre-computed $1/x$ values. The two multiplications are computed using the same multiplier by time division multiplexing the inputs. Similar

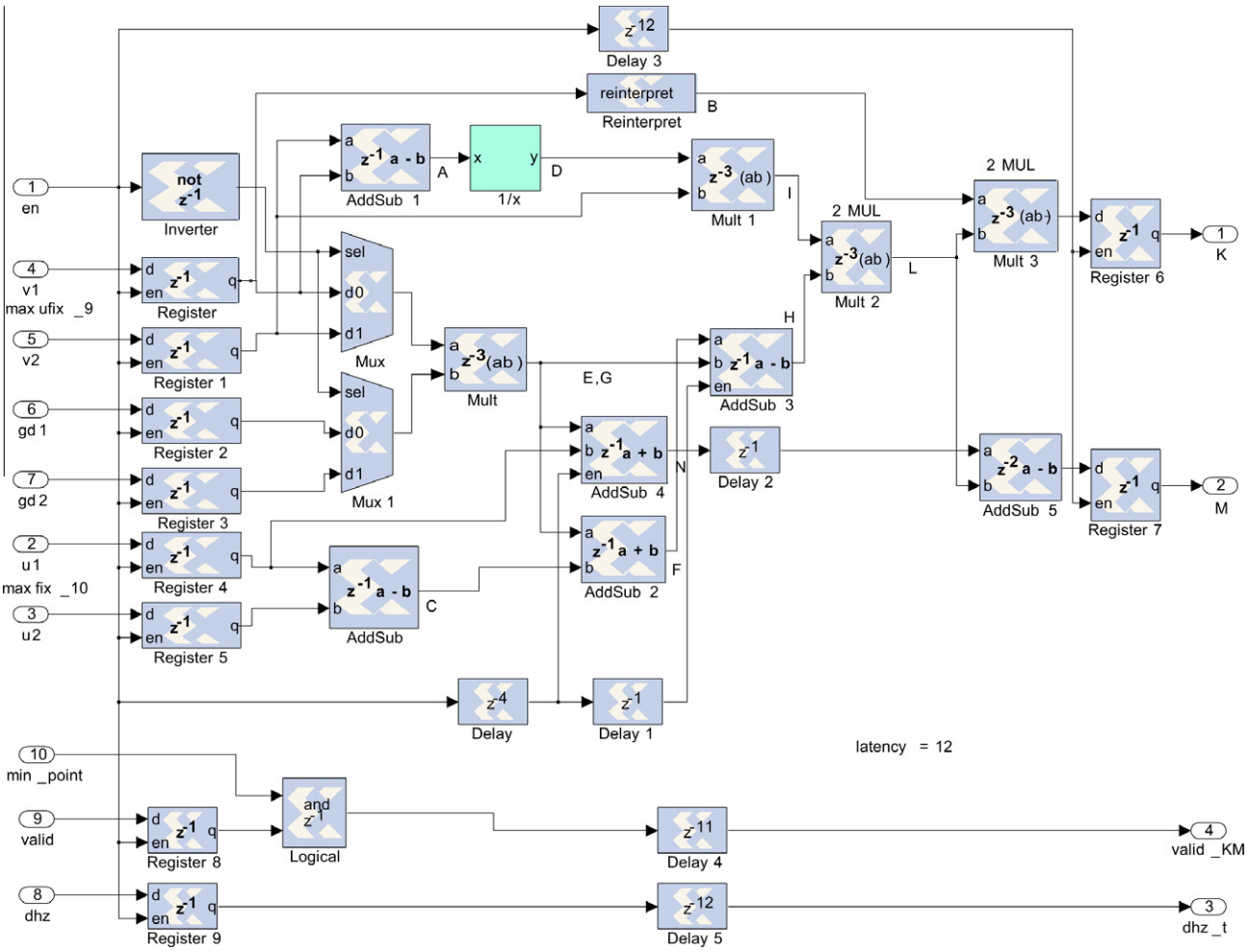

Fig. 10. Scheme of K and M computations within the *Model Candidature* subsystem.

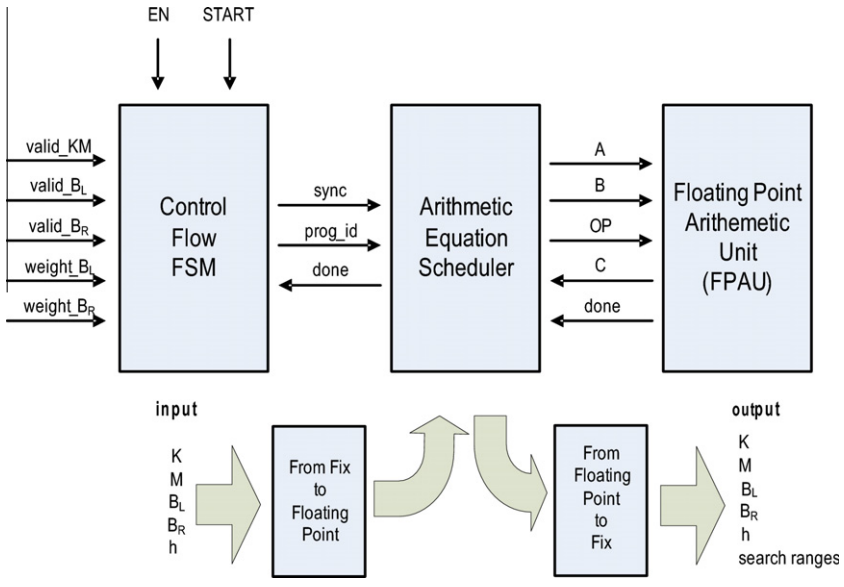

Fig. 11. High level scheme of the lane **Model Tracking** block.

optimization techniques are applied for the B_L and B_R parameters computation (which are obtained from a pair of selected points by applying Eq. (4), where K and M are the values previously estimated).

The *Model Voting* subsystem executes the voting process for both of the RANSAC procedures. The voting process is more critical than the hypothesis formulation phase because it requires computing a residual value for each point of the table. In other words, the

Table 1
FPGA resources occupation of the implemented LDW system.

	Pre-processing		Fitting		Tracking		Tot.	
DSP48s	12	9%	17	13.5%	5	4%	34	27%
BRAMs	16	12%	14	11.1%	2	1.6%	32	25.4%
Slices	2594	10%	3120	13.1%	2684	11.2%	8398	35.2%

voting stage is the bottleneck of the overall **Model Fitting** module design; therefore, the *Model Voting* block is optimized to compute a residual at each clock cycle and it is over-sampled to work three times faster than the rest of the design in order to improve the global timing performance.

The **Model Tracking** system architecture algorithm is illustrated in Fig. 11. The algorithm needs a considerable amount of mathematical calculations generally on data with different orders of magnitude. To avoid any loss of precision due to overflow or underflow situations, floating point operators are applied.

The first stage in Fig. 11, named *Control Flow FSM*, is a Finite State Machine (FSM) that leads the execution flow through the different branches of the algorithm by evaluating the different conditions. As such, it is responsible to decide which set of arithmetical operations has to be executed by observing both the reliability of data given by the fitting module and the current number of fitting failures. For example, it can (i) re-initialize the KFs, (ii) recover B_L or B_R by exploiting the lane width constraint if one of them is missing, (iii) update the internal status of the Kalman filters using the current fitting measurements or using the prediction only.

The *Arithmetic Equation Scheduler*, the second stage in Fig. 11, executes a certain set of arithmetical operations under the control the FSM block. It contains two types of memories: one read/write memory to store the data (the input and output parameters, the current status of the filters, and partial results), and one read-only memory storing the code (the different sets of equations). It keeps the FSM status frozen until the end of the calculation. The purpose of such a scheduler is to solve the KF equations by delivering operands and operators to the Floating Point Arithmetic Unit (FPAU) in the proper order and timing. The FPAU is able to compute (IEEE Compliant) single precision floating point additions, subtractions, multiplications and divisions. The fixed-point input data coming from the **Model Fitting** must be converted to floating point, and the resulting floating-point data must be re-converted in fixed point before returning back to the block.

5. Experimental results and discussion

5.1. FPGA performance

The FPGA implementation performance is measured in terms of

- resource utilization from the Xilinx Spartan-3A DSP 3400 target device, related to its Configurable Logic Block (shortly, “slices”) and RAM elements (named “BRAMs”) and programmable multiplier-accumulator units (named “DSP48”);
- timing: maximum clock frequency and I/O data rates.

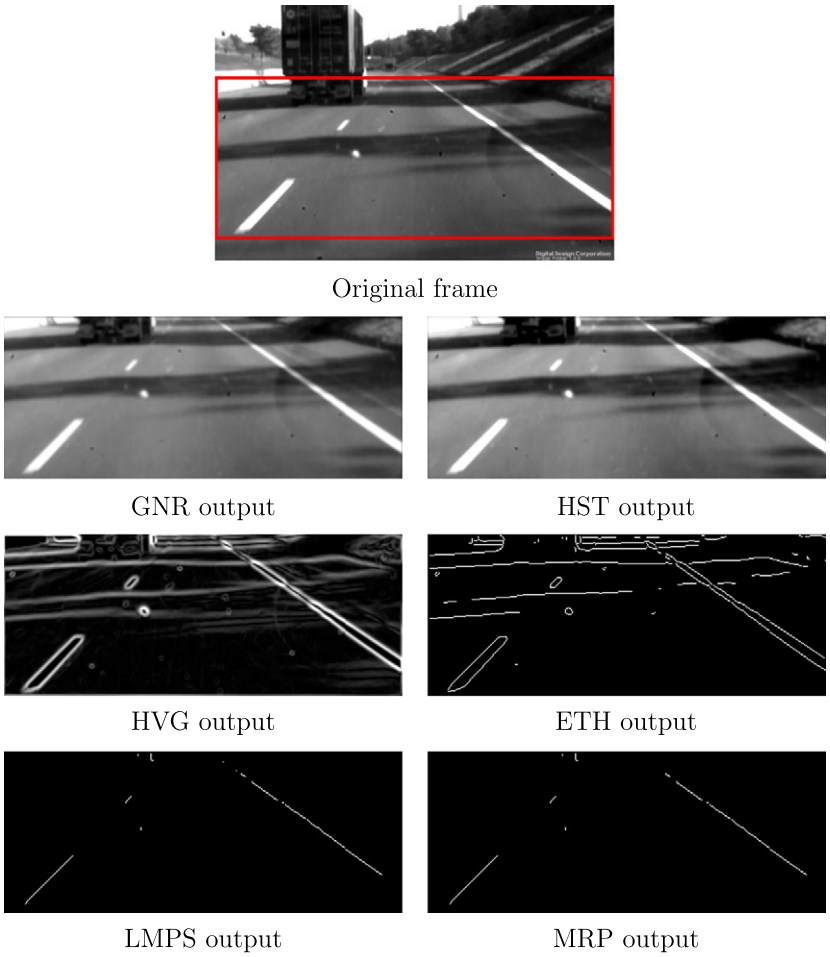

Fig. 12. Intermediate results of the pre-processing pipeline.

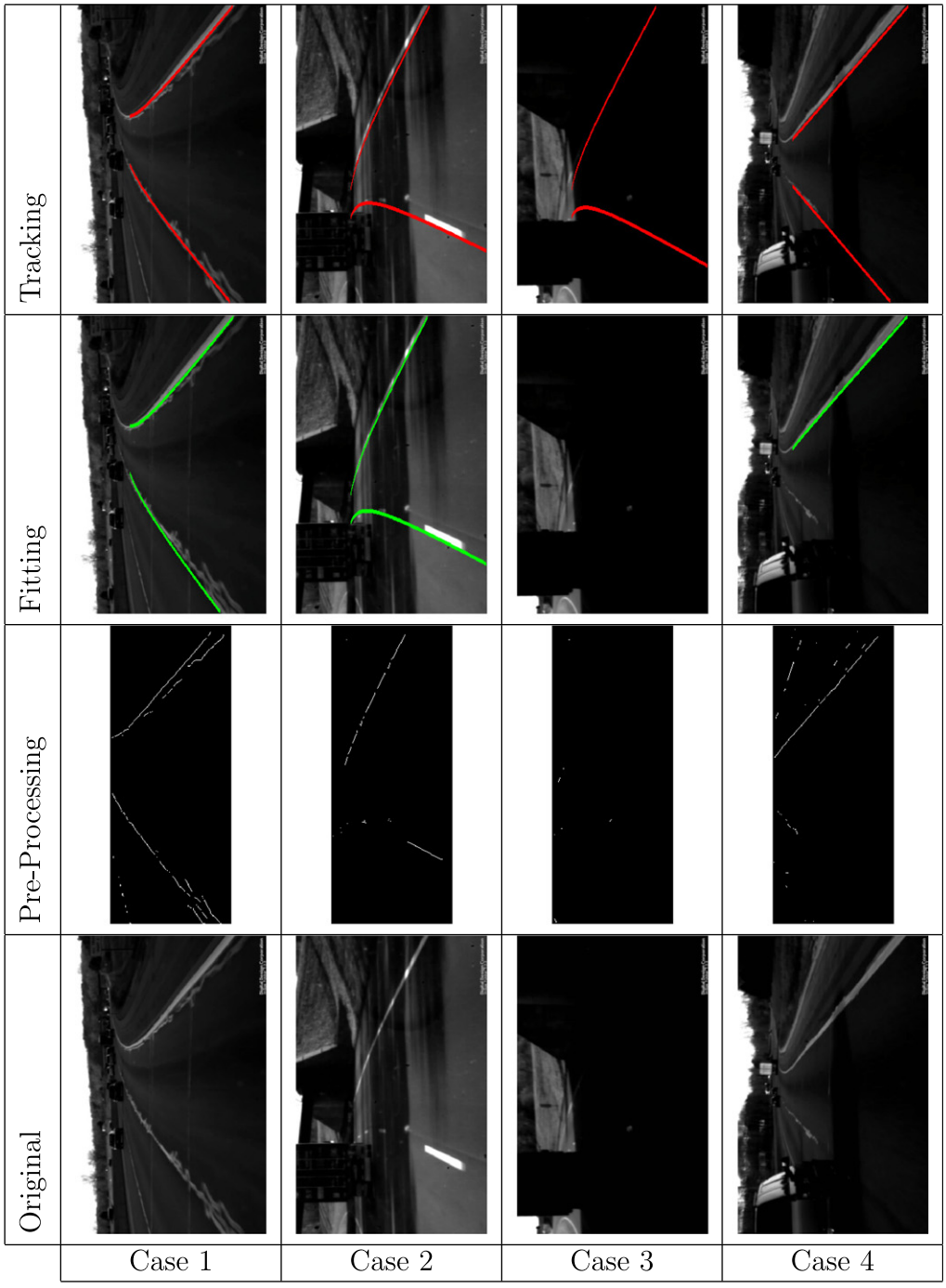

Fig. 13. Results on various scenarios: cases 1–4.

Table 1 reports the FPGA resource occupation of the three main modules of the system. The percentage of used resources with respect to those available on the chip is also specified and the total is shown in the last column on the right. Actually, only about a third of the total FPGA area is utilized. Therefore, it can be concluded that a much smaller FPGA device could be used in production to provide the functionality described in this paper. However, it is reasonable to leave a certain amount of FPGA resources available for future addition of other functions or cores (such as a micro-controller or peripherals and bus interfaces).

The most resource-consuming module is the **Model Fitting** block requiring 17 DSP48 units to implement the fixed-point multipliers mainly used in the *Model Candidature* and in the *Model Voting* blocks. The 14 BRAMs are applied to store the table of points, the vectors of tolerances and weights used in the voting stages, and the look-up tables to implement some arithmetic operators such as the $1/x$ divider.

The **Pre-processing** module utilizes 12 DSP48s for the 2D convolutions (GNR and HVG) and the HST block. The 16 BRAMs are used to implement the line buffers and to store the histograms in both the HST and the ETH blocks.

The **Model Tracking** module makes use of 5 DSP48s to implement the floating point operators and 2 BRAMs, one for the data and one for the code.

To analyze the timing performance of the overall circuit we used the timing analyzer tool within System Generator for DSP:

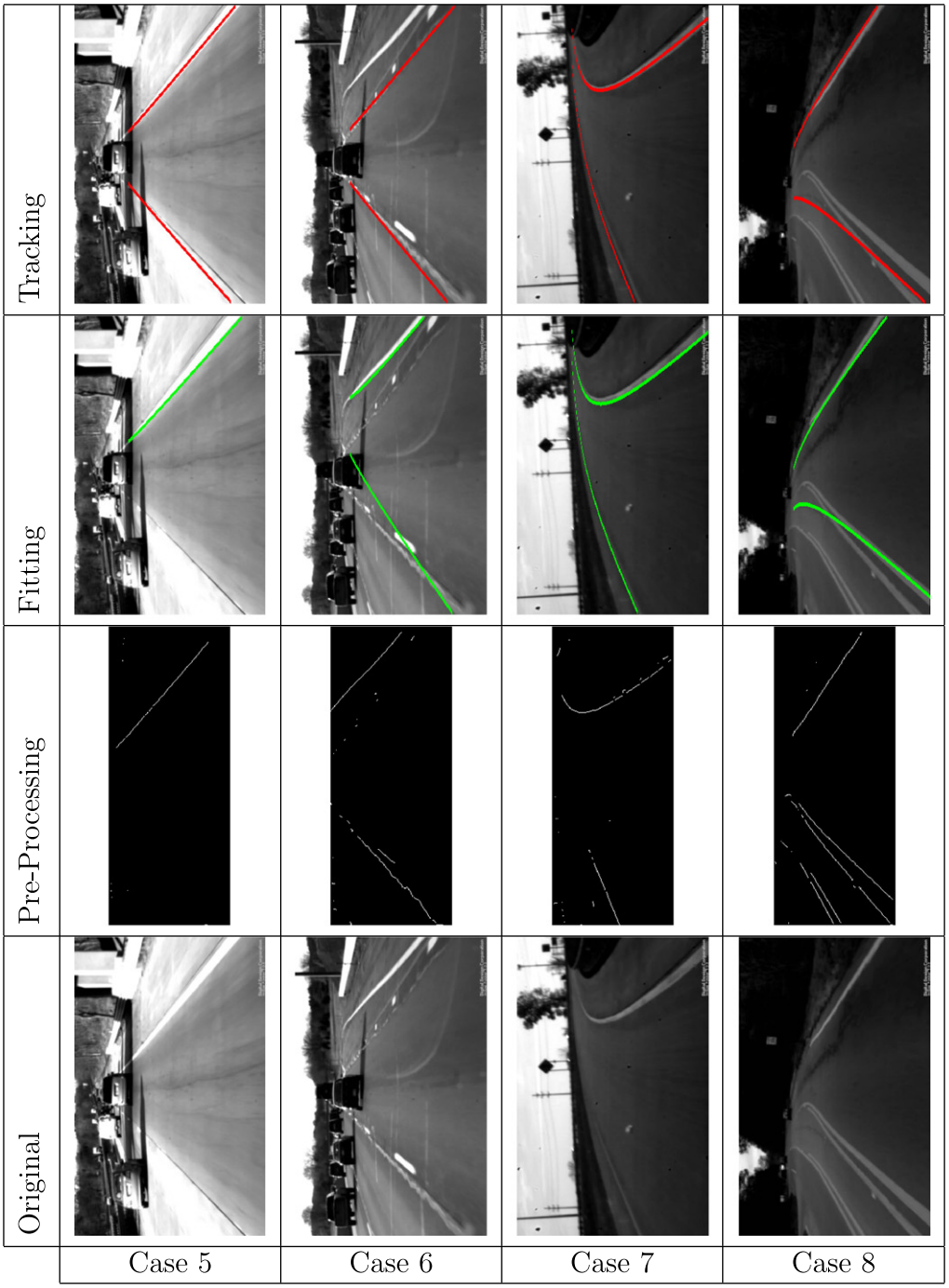

Fig. 14. Results on various scenarios: cases 5–8.

after Place and Route steps (in the synthesis flow) the maximum achievable frequency is 128.2 MHz. Since the circuit is designed to work with a time division multiplexing of 9 (i.e., 9 clock cycles for each input pixel), the data rate of the circuit is nine times slower than the maximum frequency, that is 14.2 Million of Samples Per Second (MSPS). Such timing performance allows full WVGA resolution (752×480) processing at 30 Hz frame rate, about 30 times faster than the corresponding software version of the algorithm.

5.2. Algorithm performance

To test the algorithm and its FPGA implementation, we collected a set of real video sequences depicting several typical scenarios of highways, expressway, and suburban roads. Those sequences present several levels of complexity due to curves, shadows, clutter, highlights, tar strips, camera jitter, bridge overpass, variable lighting, host vehicle weaving, subtle hills, exit ramps, oncoming vehicles passing, etc. Note that the video sequences were acquired by a camera intended for automotive use, purposely limiting the automatic exposure adjustments to generate difficult lighting scenarios.

In Fig. 12, the intermediate results of the different pre-processing blocks are shown. At the top, the original frame is shown and the cropped ROI (752×310) is highlighted. Note how the HST enhances the image contrast after the GNR and how the LMPS algorithm is able to remove most of the spurious edge points not belonging to the lane markings due to shadows, reflections, etc. In-

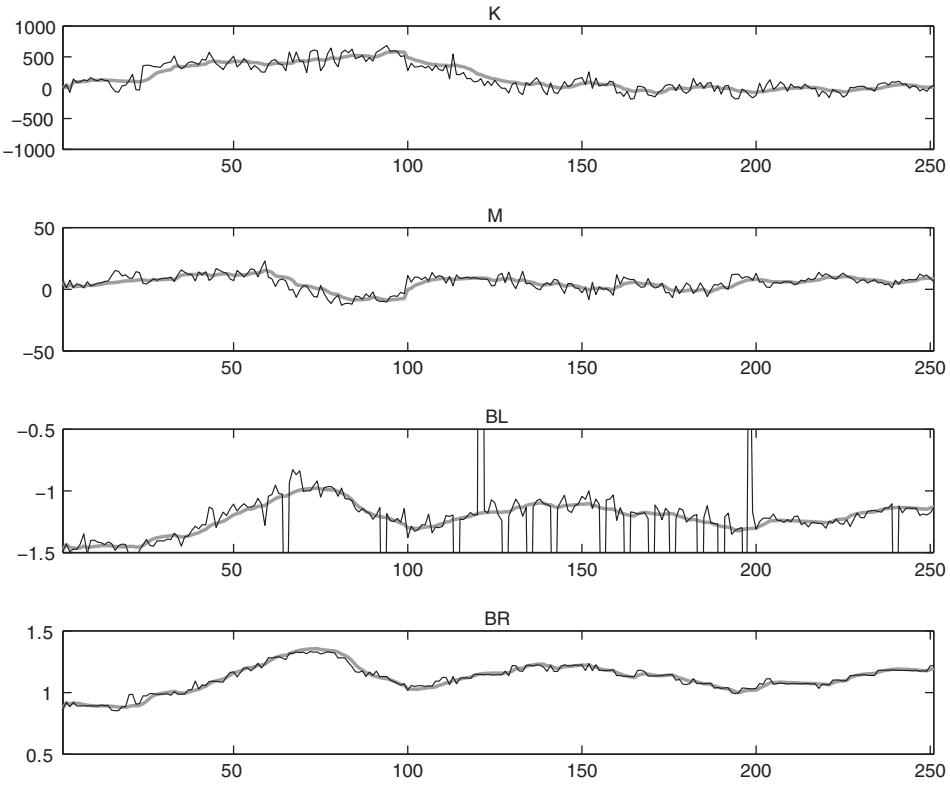

Fig. 15. Time evolution of the fitted (thin line) and tracked (thick line) model parameters.

deed, the MRP output is often very similar to the LMPS output as in the shown example.

Figs. 13 and 14 show the results obtained with frames depicting some of the most interesting cases. They have been selected to show the effectiveness of the three subsystems: **Pre-processing**, **Model Fitting**, and **Model Tracking**. In particular, note how the tracking stage can recover and/or fix the fitting outcome. Each column corresponds to a different test case. For each column, from bottom to top, the selected frame, the corresponding pre-processing output, the fitted lane markings (in green), and the tracked lane markings (in red) are shown, respectively.

In case 1, the car is in proximity of a curve in dark lighting condition and with tar strips. The algorithm is able to fit the right model of the lane.

In case 2, even though the long shadows on the road makes it difficult to extract the lane marking candidate points, the result of the fitting procedure is still correct.

Case 3 depicts a frame from the same sequence of the previous case, but some seconds later, when the car is beneath the overpass. Because of the darkness, the lane markings are not visible and the fitting fails. However, the **Model Tracking** stage keeps tracking the lane markings until the end of the overpass using only the prediction of the KF module.

Cases 4 and 5 are similar in that the fitting of one line fails in both situations. Such failure is respectively due to the occlusion of an oncoming vehicle in the former case and the image sensor saturated because of a sudden lighting change in the latter. The algorithm is able to recover the missing lane during the tracking stage by exploiting the lane width constraint.

Case 6 is an example of erroneous fitting due to a junction of the road. Properly speaking, it is not a failure because the lane actually deviates in correspondence of the exit ramp lane. Nevertheless, the tracking algorithm preserves the lost lane direction and discards the local variation.

Case 7 depicts an exit ramp lane. It is one of the most challenging scenarios as the following conditions simultaneously occur: a quite narrow curve of the road, a change of width of the lane and a considerable slope of the ground plane; moreover, the left line is not clearly visible. In this case, after a brief initial adjustment of the model in correspondence to the instant when the car changes direction and turns into the exit ramp lane, the detection is successful.

Finally, case 8 depicts a failure due to the mismatch between the conditions of the road located in a hilly area and the hypotheses of the model employed. In fact, this scenario is too far from the hypotheses at the basis of the chosen model (explained in Section 3.1): the road surface is not flat at all and even though the chosen model can adapt to gradual slope changes (see Section 3.3), this sequence presents an evident and sudden change of the inclination of the road ground plane that cannot be managed by the method. Further, a sudden deformation of the lane markings that change the curvature is also visible. As a consequence, in such cases the fitting procedure does not output the correct result.

Fig. 15 shows the profile of the lane model parameters in time obtained by a simulation using one of the sequences previously shown. The model parameters estimated by the **Model Fitting** stage are drawn with a thin line, while the parameters filtered by the **Model Tracking** stage are drawn with a thick line. K is the most difficult parameter to be estimated since the curvature is mostly determined by the lane marking candidate points near the horizon, i.e., in the top lines of the ROI. Unfortunately, these points are also the more noisy and difficult to be detected, so causing the estimation of K to be very noisy as well. However, in spite of the noise, the tracking stage is able to minimize these effects.

From the profiles of B_L and B_R, it is possible to state that B_R estimate is more stable than that of B_L, whose graph presents some

spikes in the measurement trend. This behavior is likely due to the presence of a dashed line in the left lane. Actually, because of the disconnected segments, there are frames where no lane markers are present, and even when the line segment is visible, lane marking points are very few as compared to those of a solid line. Even in this case, the tracking algorithm successfully fills these gaps and recovers the missing line by exploiting the presence of the other line and the lane width constraint.

Although Figs. 12–14 report necessarily only 1 frame, the test was exhaustively done on the whole sequences using the FPGA development board of the Spartan-3A 3400 DSP device [31], in particular in the so called Ethernet point-to-point hardware-software co-simulation, a procedure allowed only by System Generator for DSP. The tool, transparently to the user, creates the Ethernet infrastructure communication between the Simulink environment running on the host PC and the FPGA target device placed on the board. The host PC transmits the frames from the video sequence to the FPGA device via shared memories, thus allowing pseudo real-time communication during the testing procedure.

In this way, the Ethernet protocol is used only to transfer the input video data from the PC side to the hardware platform and to transfer the output results from the hardware platform to the PC. This simulation allows to test the design on the real hardware and to measure actual performances as the whole processing was *on the FPGA*. Nevertheless, we recently integrated our design in a stand-alone hardware platform directly connected with an automotive CMOS camera, and preliminar tests on a real vehicle provides performances up to 60 fps at VGA resolution.

6. Conclusions

In this paper, a roadway path extraction and tracking algorithm is presented. The algorithm is composed of three main modules: (1) a pre-processing pipeline devoted to removing noise, increasing the image quality and extracting the lane marking candidates, (2) a model fitting module that estimates the parameters of the parabolic road model using a RANSAC-like algorithm, and (3) a stage to track the model parameters for a more stable and reliable operation. No camera calibration is required, and this makes the method suitable to be implemented more easily in real applications.

A prototype has been implemented on a Spartan-3A DSP 3400 FPGA device. The design consumes about 30% of the FPGA hardware resources and runs in real-time at a 30 Hz frame rate with WVGA image resolution. To the best of the authors' knowledge, this is the first time that a complete image/video processing analysis system and its control procedures, are completely implemented in an FPGA as a pipeline of specialized modules.

Results have been provided using real and challenging road sequences of different complexities collected by an onboard automotive camera, and have shown that the method has good performance in terms of both image quality and numerical accuracy. The restrictions of the system reside mainly in the limits of the road model employed and on the complexity of certain type of roads, which do not allow the extraction of road boundaries or to reliably recover model parameters.

The FPGA prototype can be considered as a good starting point for a deployable off-the-shelf core for LDW systems due to the robust and stable performances under various roadway conditions.

Future work will be devoted to improving the tracking phase which can be made more effective to manage critical situations, changing the Kalman framework into a particle filter-based method, and extending the validation phase by extensively testing the proposed method on a car equipped by a camera directly linked to the proposed system prototype. The system will be completed by adding a warning module based on the model parameters check. Finally, a repartitioning of all the included functions between the FPGA fabric and an on-chip serial processor will also be considered to provide the best optimal FPGA-based implementation.

References

[1] D. Bagni, R. Marzotto, P. Zoratti, Building automotive driver assistance system algorithms with Xilinx FPGA, Xilinx Xcell Journal (Fourth quarter) (2008) 20–26.

[2] D. Bagni, P. Zoratti, Block matching for automotive applications on Spartan-3A DSP devices, Xilinx Xcell Journal (First quarter) (2008) 16–19.

[3] P. Abusaidi, M. Klein, B. Philofsky, Virtex-5 FPGA system power design considerations, Xilinx White Paper WP285 (v1.0), February 14, 2008.

[4] M. Santarini, Driver assistance revs up on Xilinx FPGA platforms, Xilinx Xcell Journal (Fourth quarter) (2008) 8–15.

[5] Image-Based Driver Assistance Development Environment, White Paper, Altera Corporation, December 2008 (FPGAs Used in Image-Based Driver Assistance System. Nikkei Electronics Asia, April 2009; <http://techon.nikkeibp.co.jp/article/HONSHI/20090327/167919/>).

[6] M. Bertozzi, A. Broggi, GOLD: a parallel real-time stereo vision system for generic obstacle and lane detection, in: IEEE Transaction on Image Processing, 1998, pp. 62–81.

[7] J. Goldbeck, B. Huertgen, Lane detection and tracking by video sensors, in: Proceedings of the IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems, 1999, pp. 74–79.

[8] M. Beauvais, S. Lakshmanan, CLARK: a heterogeneous sensor fusion method for finding lanes and obstacles, Image and Vision Computing 18 (5) (2000) 397–413.

[9] A.H.S. Lai, N.H.C. Yung, Lane detection by orientation and length discrimination, IEEE Transactions on Systems, Man, and Cybernetics, Part B 30 (4) (2000) 539–548.

[10] J.B. McDonald, Application of the hough transform to lane detection and following on high speed roads, in: Proceedings of the Irish Machine Vision and Image Processing Conference, 2001.

[11] Y. Wang, E.K. Teoh, D. Shen, Lane detection and tracking using B-snake, Image and Vision Computing Journal 22 (4) (2004) 269–280.

[12] Q. Li, N. Zheng, H. Cheng, Springrobot: a prototype autonomous vehicle and its algorithms for lane detection, IEEE Transactions on Intelligent Transportation Systems 5 (4) (2004) 300–308.

[13] Y.U. Yim, S.Y. Oh, Three-feature based automatic lane detection algorithm (TFALDA) for autonomous driving, IEEE Transactions on Intelligent Transportation Systems 4 (4) (2003) 219–225.

[14] S. Nedevschi, R. Schmidt, T. Graf, R. Danescu, 3D lane detection system based on stereovision, in: Proceedings of the IEEE Conference on Intelligent Transportation Systems, 2004, pp. 161–166.

[15] K. Macek, B. Williams, S. Kolski, R. Siegwart, A lane detection vision module for driver assistance, in: Proceedings of the IEEE/APS Conference on Mechatronics and Robotics, 2004.

[16] K. Huh, J. Park, D. Hong, D.D. Cho, J.H. Park, Development of a vision-based lane detection system considering configuration aspects, Optics and Lasers in Engineering 43 (11) (2005) 1193–1213.

[17] J. Kaszubiak, M. Tornow, R.W. Kuhn, B. Michaelis, C. Knoeppel, Real-time vehicle and lane detection with embedded hardware, in: Proceedings of the IEEE Intelligent Vehicles Symposium, 2005, pp. 619–624.

[18] D. Schreiber, B. Alefs, M. Clabian, Single camera lane detection and tracking, in: Proceedings of the IEEE Conference on Intelligent Transportation Systems, 2005, pp. 1114–1119.

[19] J.C. McCall, M.M. Trivedi, Video-based lane estimation and tracking for driver assistance: survey, system, and evaluation, IEEE Transactions on Intelligent Transportation Systems 7 (1) (2006) 20–37.

[20] Y. Zhou, R. Xu, X. Hu, Q. Ye, A robust lane detection and tracking method based on computer vision, Measurement Science and Technology 17 (4) (2006) 736–745.

[21] H. Wang, Q. Chen, Real-time lane detection in various conditions and night cases, in: Intelligent Transportation Systems Conference, 2006, pp. 1226–1231.

[22] M. Tian, F. Liu, Z. Hu, Single camera 3D lane detection and tracking based on EKF for urban intelligent vehicle, in: Proceedings of the IEEE International Conference on Vehicular Electronics and Safety (ICVES 2006), 2006, pp. 413–418.

[23] H.Y. Cheng, B.S. Jeng, P.T. Tseng, K.C. Fan, Lane detection with moving vehicles in the traffic scenes, IEEE Transactions on Intelligent Transportation Systems 7 (4) (2006) 571–582.

[24] F. Samadzadegan, A. Sarafraz, M. Tabibi, Automatic lane detection in image sequences for vision-based navigation purpose, in: Proceedings of the ISPRS Commission V Symposium "Image Engineering and Vision Metrology", 2006.

[25] Z. Kim, Realtime lane tracking of curved local road, in: Proceedings of the IEEE Intelligent Transportation Systems Conference (ITSC 2006), 2006, pp. 1149–1155.

[26] S. Nedevschi, R. Danescu, T. Marita, F. Oniga, C. Pocol, S. Sobol, C. Tomiuc, C. Vancea, M.M. Meinecke, T. Graf, T.B. To, M.A. Obojski, A sensor for urban

driving assistance systems based on dense stereovision, in: Proceedings of the IEEE Intelligent Vehicles Symposium, 2007, pp. 276–283.
[27] K. Kluge, Extracting road curvature and orientation from image edge points without perceptual grouping into features, in: Proceedings of IEEE Intelligent Vehicles '94 Symposium, 1994, pp. 109–114.
[28] L. Xu, E. Oja, Randomized hough transform (RHT): basic mechanisms, algorithms, and computational complexities, CVGIP: Image Understanding 57 (2) (1993) 131–154.
[29] M.A. Fischler, R.C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Communications of the ACM 24 (1981) 381–395.
[30] Y. Bar-Shalom, T.E. Fortmann, Tracking and data association, Mathematics in Science and Engineering, vol. 179, Academic Press Professional, Inc., San Diego, CA, USA, 1987.
[31] T. Hill, Accelerating video development on FPGAs using the Xilinx XtremeDSP video starter kit, Xilinx Xcell Journal (Second quarter) (2008) 52–54.
[32] P-Y. Hsiao, C-W. Yeh, S-S. Huang, L-C. Fu, A portable vision-based real-time lane departure warning system: day and night, IEEE Transactions on Vehicular Technology 58 (4) (2009) 2089–2094.
[33] S. Vitabile, S. Bono, F. Sorbello, An embedded real-time lane-keeper for automatic vehicle driving, in: International Conference on Complex, Intelligent and Software Intensive Systems, 2008, pp. 279–285.