International Journal of Latest Research in Science and Technology **ISSN (Online):2278-5299**
Volume 3, Issue 1: Page No.1-8 ,January-February 2014
https://www.mnkpublication.com/journal/ijlrst/index.php

A COMPUTER VISION-BASED DRIVER ASSISTANCE SYSTEM FOR IMPLEMENTATION ON RECONFIGURABLE HARDWARE

[1]Ricardo Acevedo, [2]Miguel Gonzalez, [3]Andres Garcia
[1,2,3] Department of Postgraduate Studies, Tecnológico de Monterrey Campus Estado de México.
Atizapán de Zaragoza, Estado de México, México

Abstract - This paper presents the design of a driver assistance system based on computer vision that provides information about the environment and the vehicle's current driving state. The system has been developed to be implemented on an embedded computation platform. Physical limitations such as memory and computing power are a major concern; this renders existing image processing algorithms based on PC platforms unsuitable for direct use in this application. Instead, simplified and efficient image processing algorithms must first be developed for deployment on a reconfigurable architecture. The driver assistance system provides three main functionalities: Lane Detection, Lane-Change Detection and Obstacle Detection. The core algorithms have been realized as custom hardware co-processors to be executed on Field Programmable Gate Array (FPGA) hardware, a soft-core CPU performs general system control and final decision making based on co-processor output.

Keywords - Embedded System Design; Image Processing; Hardware Development; Embedded Computer Vision; Smart Vehicles

I. INTRODUCTION

Road traffic accidents are a serious socio-economic problem and one of the top ten causes of death in the world, as shown by a recent World Health Organization report [1]. The potential human and economic implications are large and cause continuous spending. According to the Report on Road Traffic Injury Prevention [2], road traffic accidents will be the third cause of human death at world scale in the year 2020. The research on vehicle safety and accident prevention systems is therefore presented as an essential component needed to solve this issue. One of the many problems automated vehicle safety design faces is to provide the vehicle with the right means to perceive and act upon the current environment. Sensors such as lasers, sonar devices and radars are a common solution for this purpose; however, recent advances in image processing technologies suggest a new approach that should require a single sensor to provide all the useful information about the external world; a potential solution is to implement smart vehicles equipped with artificial vision gear.

Computer vision-based driver assistance systems refers to a set of technologies designed and developed for improving traffic safety using a video camera as the sole sensor. It relies on existing road infrastructure and is typically powered by general computing platforms such as Personal Computers (PC) that run several algorithms aimed to support human driving (e.g., road recognition, lane and vehicle detection, pedestrian detection, etc.). Rather than developing a driver assistance system for use on a PC, it is possible to design a real-time embedded system for deployment on-board the vehicle. Embedded systems focus on a very specific task, requiring a limited amount of resources to be able to efficiently perform their functions. However, not all image processing algorithms are suitable for a feasible embedded implementation. The hardware complexity of a computer vision algorithm is often related to the kind of image processing tasks it must perform. Image processing tasks can be classified into three categories [3]: low-level, intermediate-level and high-level computations. Raw pixel data processing is a crucial task in the low-level category, and often involves the heaviest amount of computational load. Common low-level computations are intended for feature extraction, and may include down sampling, filtering or edge detection. Intermediate-level operations include object recognition, scene interpretation, segmentation and labelling. These tasks rely directly on low-level operations and usually require less pixel data. High-level tasks, in turn, depend directly on intermediate-level operations, and are far-less data intensive than the previous tasks, usually limited to decision-making, and may include a conceptual description of a scene in terms of activity, intention and behaviour. A full embedded driver assistance system should be able to perform tasks of all the three previous categories in real-time by off-loading each processing task to dedicated co-processors, a Central Processing Unit (CPU) can act as a master controller, performing only supervisor activities and being the general interface to the user. The embedded approach allows the full system to be contained in one single chip. The Field-Programmable Gate Array (FPGA) technology efficiently allows the building, debugging and reconfiguration of a Custom Computer Machine [4]. This work discusses the design and implementation of a Computer Vision-Based Driver Assistance System on FPGA hardware. An Altera Cyclone II EP2C70 FPGA is used on this work due to its low cost, board-integrated components and easy design environment.

Publication History
Manuscript Received : 12 February 2014
Manuscript Accepted : 19 February 2014
Revision Received : 20 February 2014
Manuscript Published : 28 February 2014

This paper is organized as follows: Section II covers related work, Section III describes the algorithms used, Section IV describes their translation to custom hardware and the final implementation. In Section V performance and results are discussed. Conclusions and future work are presented in Section VI.

II. RELATED WORK

We begin by examining real system prototypes that are being currently developed in the automotive security industry. Nissan Motor Company [5] has recently developed a sensor-based security prototype called the All Around Collision Free System (AACFS) to help protect the vehicle from potential risks coming from multiple directions. One of its main features is the Side Collision Prevention System (SCPS) in which vehicle-mounted sensors activate a warning if an approaching vehicle is detected in the driver's current lane, thus avoiding a potential collision. This solution encompasses a series of motion sensors grouped together to form a robust system which can even take control of the vehicle in a situation of extreme danger. Khan et al. [6] proposes a similar motion-sensing system based on radars using the Nios II soft-core CPU developed by Altera; the Multiple Target Tracking (MTT) system offers the advantages of a long range sensor as opposed to cameras, but is dependent on surface material and random presence of multiple objects can limit the efficiency of this approach. The Honda Motor Company also developed its own on-board security system [7] called the Advanced Safety Vehicle Version 3 System (ASV-3). This set-up uses a wide range of devices including radar, lasers, global position sensors and cameras. One of the core features of the system lies on its capability to transfer data between other ASV3-equipped vehicles. If an ASV3 unit encounters a dangerous situation, it is capable of transferring this knowledge to other surrounding vehicles, minimizing the chance of an accident.

Optic-based sensors offer a clear advantage over other passive sensors: They can use the existing road infrastructure as direct input without the need of extra environmental modifications. The goal is to be able to develop a smart system capable of perceiving and understanding its surroundings in a visual manner, similar to what a human would do[8]. A good practical example of what an already existing embedded device can achieve is proposed by Feixiang et al. [9]. The system offers a computer vision-based Lane Departure Warning System (LDWS) implemented on a mobile phone. The detection method is based on the Hough Transform, an often-used algorithm for line detection. Data-intensive computing algorithms such as the Hough transform are executed in real-time seamlessly inside the embedded central processing unit at a frequency of 620 MHz coupled with an on-board memory of 128 MB for storage purposes. Another system design is discussed by Cario et al. [10]; it focuses on an efficient method for designing a LDWS. The algorithm detects lane markings on the road and models them as finite line segments. It is then possible to track their angle (and position) at any time. The two line angles can be added up and compared against a certain threshold. This information will tell the driver if the vehicle is leaving the lane as well as the vehicle's position between two road lanes.

The solutions reviewed so far are applicable only to PC platforms if real-time execution is needed. Much of these algorithms make use of floating-point operations, a simplification is needed for these algorithms to be implemented as electronic components on fixed-point architectures. One possible simplification for road detection would require the transformation of the camera image into a simpler domain, where line detection would become easy to perform with straightforward operations. According to Broggi [11] one such transformation would be the Inverse Perspective Mapping (IPM) transform. IPM performs a projective geometry transformation that attempts to correct the distortion caused by the foreshortening factor and vanishing point effects. In other words, IPM re-maps all pixels in a source image to a new position in a new image according to a defined mapping rule. The resulting image will resemble a bird's eye view of the source image, effectively removing the perspective distortion effect. This is an excellent road simplification model that allows the IPM transform to be implemented, under certain conditions, as a simple data movement operation.

III.SYSTEM DESIGN

Fixed-Point Models

Design begins with the development of a software version of the system that will define the needs and restrictions of each processing stage that composes the complete pipeline. Using this design methodology is possible to analyse and debug each component quickly and easily. Once the functionality of a specific component is tested, we can simplify and optimize its design. The Matlab Design Environment is used to model each one of these stages. The general outline of the system is proposed as follows:

Image acquisition at a resolution of 640 x 480 pixels, image RGB conversion to grey-scale, image filtering, image processing, object analysis (e.g., tracking and recognition) and finally, decision making. Data noise frequently occurs and deteriorates the classification and tracking performance, therefore, an effective data cleansing mechanism must be implemented. The grey-scale source image runs through a basic filtering stage composed of dilate and threshold operations. The road model simplification is applied using IPM. At this point, a clear (binary - black and white) approximate model of the road will be generated; with this information the system can easily detect, estimate and track the road markings and evaluate the vehicle's current position within the road.

Image Filtering

Mathematical morphology is described as a set-theoretic method of image analysis providing a quantitative description of geometrical structures, and is composed of a series of morphological operators [12]. These operators manipulate the interaction between an image f(x,y) (the object of interest) and a structuring element h(x,y) (that could be expressed as a convolution mask) to produce an output image g(x,y):

$$g(x,y) = h(x,y) * f(x,y) \quad (1)$$

Where:

$$h(x,y) = \max\{neighbourhood(x,y)\} \quad (2)$$

The dilation operation (Eq. 2) essentially expands an image object, effectively "filling up" holes produced by a faulty image acquisition, refer to Fig. 1. Note that structuring element h(x,y) operates on one pixel at a time. In this work, a basic dilate filter is applied to an incoming grey-scale image implementing a square structuring element of size 3x3. This will provide the necessary noise cleaning and will allow to reproduce the morphological operator using an array of hardware elements. It is important to consider, however, that this filter typically operates on an image object stored in memory. In a hardware pipeline, the image object is not directly stored, instead it is constituted as a continuous serial stream of bytes coming directly from the acquisition stage.

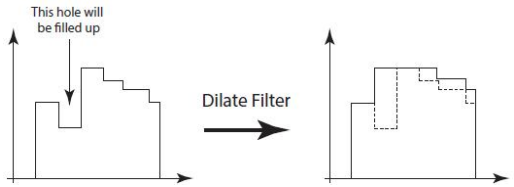

Fig. 1 Dilate filter operation. After applying the filter, holes (black pixels) in the original image will be "filled up".

Once the source image is filtered by the dilate operation, a thresholding operation is applied. The basic idea is to clearly separate the white lane markings from anything else on the scene, producing a binary image. This is accomplished by comparing each incoming pixel against a fixed threshold value T, as show in Eq. 3. Note that the actual implementation of the operation is a simple comparison.

$$g(x,y) = \begin{cases} 0 \text{ if } f(x,y) \le T \\ 1 \text{ otherwise} \end{cases} \quad (3)$$

Road Model and Inverse Perspective Mapping

The Hough Transform is a commonly used software-based method for detecting lines and circles and is constantly applied in lane detection scenarios due to its robustness to light variations and image noise [13, 14]. However, its complexity demands significant computer power [15]; way beyond the hardware capability of this application. Instead of using the Hough Transform method for line detection, we will simplify the road model using IMP. Fig. 2 shows the image distorted by the camera, along the true image that can be obtained if a perspective correction method is applied (also known as projective transformation). It is important to note that this operation is only a pixel mapping between two image planes: image plane and world plane.

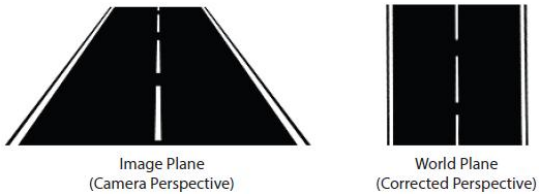

Fig. 2 Distortion of the world plane and true world plane

Mathematically, IPM can be described as a projection from a 3D (image plane) Euclidean space onto a 2D plane (world plane). A perspective mapping is described by Mallot et al. [16] in the following form:

$$\mathbf{R}^3 \Rightarrow \mathbf{R}^2 \quad (4)$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \Rightarrow \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \quad (5)$$

The mapping between the two planes can be represented by a set of matrices. The perspective distortion can be modelled by the following projective transformation, as written in Eq. 6, where $\bar{\mathbf{X}}$ and $\bar{\mathbf{X}}'$ are three-element vectors representing a point (pixel) and $\bar{\mathbf{H}}$ is a homogeneous non-singular 3x3 matrix.

$$\bar{\mathbf{X}}' = \bar{\mathbf{H}} \cdot \bar{\mathbf{X}} \quad (6)$$

Here, $\bar{\mathbf{X}}$ represents the original (world) image points, matrix $\bar{\mathbf{H}}$ models the camera's distortion as a "homography matrix", and $\bar{\mathbf{X}}'$ contains the distorted (camera) image points. The world and pixel coordinate systems are related by a set of physical parameters known as *extrinsic* and *intrinsic* camera parameters. If we combine both parameters we can generate a projection matrix. For a projective transformation, the system can be modelled as the following linear transformation [20]:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} * \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (7)$$

The method used for the perspective distortion correction requires four non-collinear points to create the mapping between the two planes. It is possible to further simply the process if the area of the image that will be mapped to the world plane is assumed constant at all frames. (i.e., the camera is placed in a fixed position, always facing the same portion of the road). The constant area is surrounded with a red polygon (trapezoid) in Fig. 3 The four points needed for the transformation correspond to each one of the four corners of the trapezoid shown on the image plane; their equivalent positions on the world plane are also shown. These points can be selected manually. With all input information it is possible to generate a set of eight equations [17, 18]. The equations can be solved off-line using any mathematical software and a pair of matrices containing the matching between the world and image plane pixels can be obtained. With the four point correspondences discussed above, it is possible to extract eight equations to generate matrix $\mathbf{H}$, as shown in Eq. 8.

$$\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 & -x_0x'_0 & -y_0x'_0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x'_2 & -y_2x'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x'_3 & -y_3x'_3 \\ 0 & 0 & 0 & x_0 & y_0 & 1 & -x_0y'_0 & -y_0y'_0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y'_2 & -y_2y'_2 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y'_3 & -y_3y'_3 \end{bmatrix} * \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x'_0 \\ x'_1 \\ x'_2 \\ x'_3 \\ y'_0 \\ y'_1 \\ y'_2 \\ y'_3 \end{bmatrix} \quad (8)$$

It is important to note that the resulting pixel mapping operation is constant, regardless of the pixel itself, as long as the same camera is used, pixels will be always remapped to the same location in the world plane. Consider the fixed trapezoid points we have established before, this will let us save the final position matrix and store it in a Look Up Table (LUT). While reading incoming pixels, it is possible to know their final position in the new plane just by looking at the LUT.

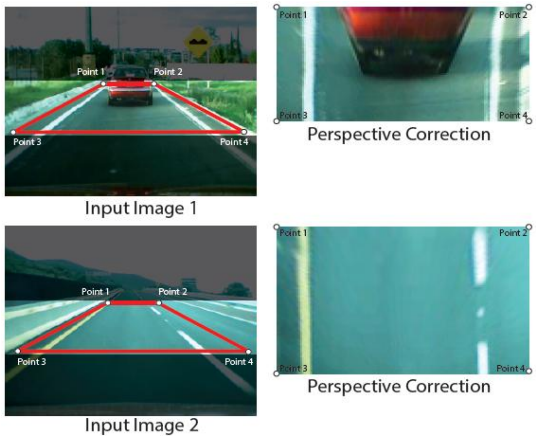

Fig. 3 Camera input image (left) and transformed image (right).

Road Model and Inverse Perspective Mapping

Once the image has been transformed, it is necessary to estimate the position (as coordinates) of the two lanes along the horizontal axis. Fig. 4 shows the ideal image obtained after the perspective transformation is applied.

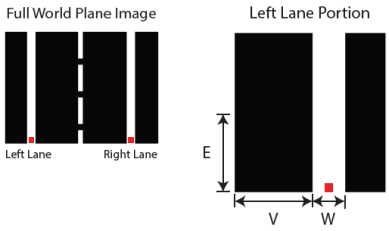

Fig. 4 Ideal transformed image. Each lane centroid is marked with a red square.

Each white lane can be represented as two perfect rectangles, with maximum width **W** and minimum height **E**. If we consider the distance **V** from the bottom left to the start of the white rectangle, then, the centre horizontal coordinate of this lane can be directly calculated as:

$$C_t = V + \frac{W}{2} \tag{9}$$

In a real image obtained after the perspective correction, however, the two lanes hardly resemble perfect rectangles, as shown in Fig. 5. It is possible to process each row of the image independently and then compute the average of each value to obtain the final centre coordinate of the white lane.

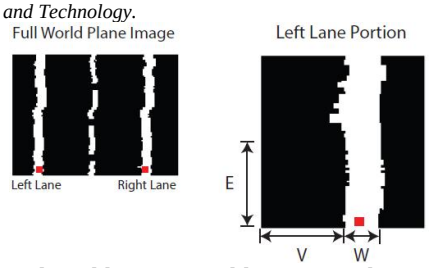

Fig. 5 Real world image. Real lane centroids are slightly shifted to the left and right.

To estimate the real positions of each horizontal coordinate we can proceed as follows: consider each row of the image is processed from left to right. First, count each black pixel and store the final value into an accumulator variable $\mathbf{V_n}$. The white pixels are also counted and the final sum stored into another accumulator variable $\mathbf{W_n}$. Consider also that, after the last white pixel has been counted, we process the next row of the image. If a row has been processed, the row counter variable **E** must be increased in one unit, if the number of processed rows has reached an **E** predefined bound, the processing is over and we need to compute the average of each accumulator variable.

$$For\ black\ pixels: V_t = \frac{V_n}{E} \tag{10}$$

$$For\ white\ pixels: W_t = \frac{W_n}{E} \tag{11}$$

The lane centroid in the horizontal axis can be finally computed as:

$$C_x = V_t + \frac{W_t}{2} \tag{12}$$

Algorithm 1. Shows the basic steps of the lane detection system approach. Assume a constant value of **E** = 16 (i.e., the number of processed image rows). This simplification will let us analyse the last 16 rows of the image; the advantage of setting this value will let us represent the divisions by **E** as a right shift in hardware, as shifting right by **n** bits on an unsigned binary number has the effect of dividing it by 2^n In this case $\mathbf{E} = 2^4 = 16$.

Input*: Transformed image as a continuous stream of pixels*

Output*: A point in the image that represents the lane position on the horizontal axis.*

1. Set $\mathbf{V_n} = 0$, $\mathbf{W_n} = 0$ and **processedRows** = 0.
2. Process new image row. Set **flagVariable** = 0.
3. Receive new incoming pixel.
   If incoming pixel is black, count incoming pixel and store result in variable ($\mathbf{V_n} = \mathbf{V_n} + 1$).
   If last processed pixel was white, set **flagVariable** = 1 and compute $\mathbf{V_n} = \mathbf{V_n} - 1$
4. If **flagVariable** = 1 go to line 5, else, return to line 3.
5. Compute **processedRows** = **processedRows** + 1
6. If **processedRows** = 16, compute the final results as:
   $\mathbf{V_t} = \mathbf{V_n}$ / **processedRows**
   $\mathbf{W_t} = \mathbf{W_n}$ / **processedRows**
   $\mathbf{C_x} = \mathbf{V_t} + \mathbf{W_t}/2$
   Else, go to line 2.

7. The lane centroid (along the x axis) is stored in $\mathbf{C_x}$, Terminate algorithm.

Algorithm 1. Algorithm that implements the lane extraction operation.

IV. HARDWARE IMPLEMENTATION

Image pre-processing stages are included before the actual perspective correction component to increase the system robustness to light variations and other environmental noise. The basic operations that are studied in this work are thresholding and dilation on grey-scale data. The threshold filter, as seen earlier, is a fairly straightforward operation: compare the grey-scale value of a pixel against a certain, fixed value. If the pixel is equal or greater than the threshold value, output a white pixel, if not, output a black pixel. The dilate filter, on the other hand, requires particular hardware considerations, as discussed below.

Morphological Operation

The dilate filter will help define certain areas of the image by filling in holes that could possibly produce a faulty detection, as seen in Fig. 6. The main challenge that this filter presents is that it requires information of at least eight neighbouring pixels for a 3x3 matrix application (the simplest implementation of an image filter) before outputting an actual result. We must design a hardware component that receives a stream of pixels (encoded in eight bits for grey-scale images) each clock cycle, for each incoming pixel, the component must compute its result based on the values of the incoming pixel and its neighbourhood. The basic conceptual schematic diagram of this component is shown in Fig 7.

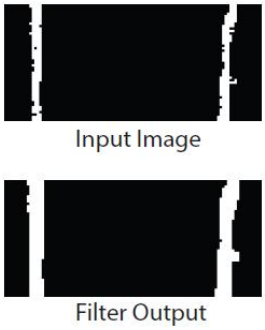

Fig. 6 Output of dilate filter with a structuring element of size 3x3, note the "expansion" of each white pixel.

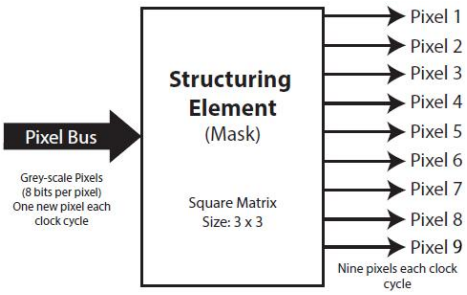

Fig. 7 Conceptual diagram of the dilate filter.

A continuous serial stream of pixels is presented at the input of the component, with this in mind it is necessary to design a method for storing at least the nine pixels required for the image filter to output valid data. Once all the needed pixels are stored, it is possible to immediately operate on them on the same clock cycle. In the following clock cycle a new pixel will be received, its neighbourhood data must be updated as well. The new pixel-neighbourhood will consist of the same data shifted one pixel to the left, so neighbourhood information previously gathered must not be discarded. In fact, we will only discard this information once the filter has processed three rows of the input image; to operate on a serial stream of pixels an array of First In First Out (FIFO) memory blocks is proposed. The FIFO memory will hold the data needed for the dilate operation.

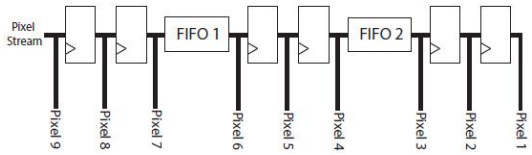

Fig. 8 Memory array for arranging a 3 x 3 pixel matrix.

Refer to Fig. 8 Each FIFO memory holds an entire image row minus three positions; those three positions will fill up the 3x3 matrix, and this pixel data will be held (or delayed) by three registers. Pixel 1 will be the first pixel entering the memory array, Pixel 2 will be second, and Pixel 3 will be third. The FIFO 2 block holds the remaining pixels for that particular image row. The FIFO array is repeated for the second image row (i.e., Pixel 4, Pixel 5, Pixel 6 and FIFO 2). The last three pixels are stored directly in registers. It is necessary to consider, however, that the first pixel will arrive to the position Pixel 1 exactly after the last pixel has entered position Pixel 9, this will introduce a minimum latency for this component. After the 3x3 array has been filled up with valid data, it is possible to search for the maximum (in the case of the dilate operation) grey-value that is currently stored inside the matrix. This value will be the actual filter's output. Each processed pixel will then be compared against a threshold value to create a final, filtered binary image.

Inverse Perspective Mapping

As seen on Section III, once matrix $\mathbf{H}$ (homography matrix) is obtained, the world plane pixel positions for any image plane pixel can be computed. The world plane pixel positions can be stored in Read-Only Memory. This means that in the proposed FPGA architecture, each pixel composing the distorted image can be re-assigned to a new position according to the data stored in ROM; as shown in Fig. 9.

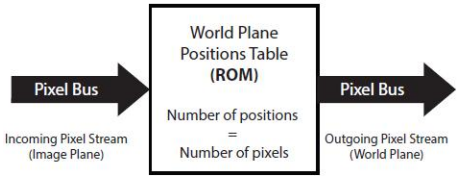

Fig. 9 Conceptual diagram of the IPM component. Each incoming pixel is located in a new position according to the data stored in a ROM block.

To fully implement this idea in hardware three major components are needed: Read-Only Memory to store the transformed position data, Random-Access Memory to hold each incoming video frame (previously filtered) and a RAM/ROM controller to sync-up the data transmission between the two memory blocks. The RAM/ROM controller is designed as a finite-state machine model with four main states (See Fig. 10).

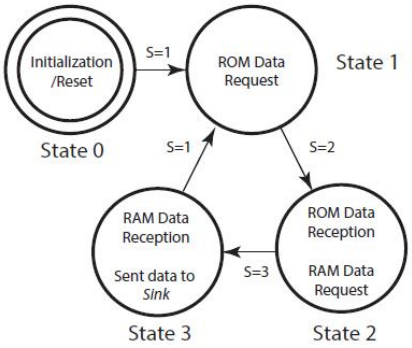

Fig. 10 The finite-state machine that syncs up and controls the data transmission between the two memory blocks.

This component will effectively apply a perspective correction to the input image, it is worth noting, however, that it requires at least four cycles to produce its output, this will, again, introduce a minimum latency to the overall system.

Lane Coordinates Estimation

A pure hardware architecture is the final implementation for this algorithm. The divisions appearing in Eqs. 10, 11 and 12 need to be implemented as digital circuits. As discussed in Section III, we solve this issue by proposing a **E** value of 2^4 = 16 and implementing the division as an arithmetic right shift. In the case of Eq. 12, it is possible to right-shift the value just one position. For the actual hardware component, Algorithm 1 was implemented in Very High-Speed Integrated Circuit Hardware Description Language (VHDL). The algorithm can be adjusted to compute the two lane coordinates in parallel. Fig. 11 shows the output of the actual hardware component.

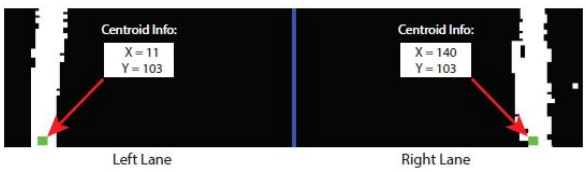

Fig. 11 Output of the lane coordinates extraction component. Lane centroids are represented as two green squares.

Data Processing and Lane Change Detection

After computing both centroids, we can finally determine if a lane is present on the road, a lane-change is taking place or if an obstacle is lying directly between these two points. To detect an obstacle the system can threshold the area between the two lanes, if an obstacle exists, it will show up as a blob of white pixels. If the blob of pixels exceeds a certain security distance, an alert can be issued to the driver. In order to detect a lane change, the processing is a little bit complex, as a time-tracking of both lane centroids are needed; this is accomplished by defining a processing window. The processing window will evaluate the changes presented in 10 video frames, an embedded software algorithm will then conclude if the vehicle has, in fact, changed lane.

Obstacle Detection Output 1

Obstacle Detection Output 2

Fig. 12 Obstacle Detection. The corresponding object identified as an obstacle is easily tracked through time with a blue frame in the RGB space.

The basic idea is simple: divide the bottom of the image in two major areas: Left and Right. Further divide those areas in two halves, so we end up with four sub-areas as shown in Fig. 13, each lane centroid will cross each region at different times, depending on the vehicle's direction.

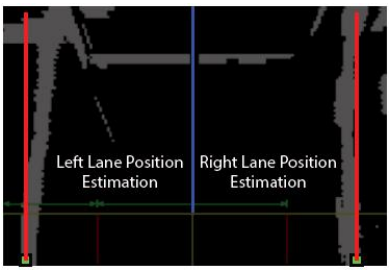

Fig. 13 Lane-change detection. Note the four sub-areas at the bottom. The red vertical lines represent each of the idealized lanes.

If the left centroid moves to the right and the right centroid moves to the left, disappearing from the scene, then, a change to the left (probably) has occurred. Conversely, if the right centroid moves to the left and the left centroid eventually disappears, the system could infer that a change to the right has just occurred. A certain threshold value is also needed to be perfectly sure when a vehicle is really changing direction. We define the minimum distance thresholds that both centroids have to cross as the two halves of the left and right areas in Fig. 13 (red lines).

Previously, a processing window of 10 frames was defined. If the system notices that both centroids are beyond the minimum distance threshold in at least 6 frames out of 10, it can conclude that a real lane-change is taking place in that instant in time. This algorithm has been implemented as a program written in C running on a NIOS II soft-core CPU, the NIOS II is a 32-bit embedded-processor architecture designed specifically for the Altera FPGAs. Its configuration for this particular application can be listed as follows:

1. Standard core.
2. 20 K Bytes of on-chip memory.
3. UART JTAG for host communication/debugging.

V. RESULTS

The first iteration of the system is focused on detecting a lane-change event. Tests on real-world scenarios were carried out with the results evaluated by human designers. Each test involved the evaluation of 10 frames of video in a sequence of video of varying length. It is important to note that the system yields a result after 10 frames of video are evaluated, that way, if detection shows up in at least 6 of 10 frames we can conclude that, in fact, that detection is positive. The system was tested at a processing speed of 20 frames per second. The test vehicle changed lanes randomly through the whole video. Fig. 14 shows the detection of a right lane-change with the design discussed in this paper.

The system is actively detecting and identifying lanes and lane-change events with enough precision, provided that the lane markers are correctly painted on the road and the camera is correctly positioned and fixed during system evaluation. Tests performed on the lane detection and lane-change sequence resulted in acceptable accuracy for each video sequence, in the best detection conditions, the system correctly identified 9 lane-change events out of 10, in typical conditions the system correct detection rate was 8 out of 10. When testing the obstacle and vehicle detection system, however, results showed that the current version of this system can improve in future iterations.

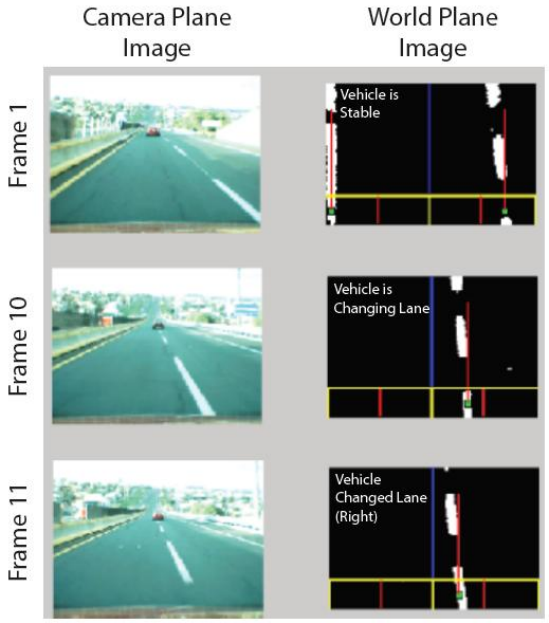

Fig. 14 Lane-change detection test from a random video sequence.

The simple obstacle detection discussed in this paper can only detect one obstacle at a time, with our approach; the obstacle closer to the vehicle will be identified and tracked in time. Out of 10 test video sequences, the system correctly identified 6 obstacles lying in front of the vehicle. One point to note, however, is that no occlusion [19] occurred during these particular test sequences and hence performance was expected to improve. The full design can be implemented on the EP2C70 FPGA circuit, as show on Table 1. The core, simplified, architecture of the proposed system is showed in Fig 15.

Table 1. Final Component Specification

Parameter	Value
Max Frequency	44.69 MHz
Input Image Width	640 px
Input Image Height	480 px
Input Pixel Size	8 bit (Grey-scale)
Memory Consumption	568,854 bits
Total Logic Elements	4,166

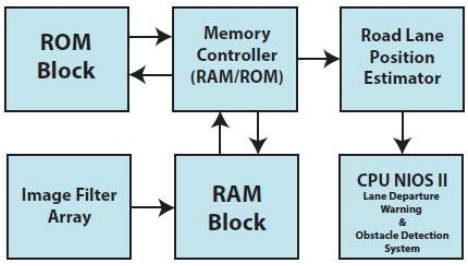

Fig. 16 Simplified system architecture.

VI. CONCLUSIONS

In this paper we proposed an implementation of a Driver Assistance System running on embedded hardware, which is a real and promising solution for improved traffic and road security. A simplified model of road and lane detection using a perspective transformation technique was developed to take advantage of a hardware-configurable environment; additional hardware-based algorithms were designed in order to extract relevant data from the scene, these algorithms proved to be efficient and appropriate for an embedded application. One of the crucial components of this system is the perspective corrector based on IPM, it is important to note that, as shown in Eq. 8 this solution can assume constant extrinsic and intrinsic camera parameters as long as the same camera is used and correctly positioned on the vehicle, we must be aware, however, that this approach will transform a fixed part of the scene (the area covered by the red trapezoid in Fig. 3) We must note, also, that we're just processing 16 image rows instead of the complete image, this simplification let us implement arithmetic integer divisions as shift operations. Practical tests we carried out to determine the minimum rows required for acceptable lane identification. At an image resolution of 640 x 480, 10 to 15 rows showed no significant deviation in the lane's position estimation for this application.

Our proposed system design has proved to be feasible and reliable according to the experiments conducted, and it is suggested as a viable solution for the automotive security problem. There is still room for code and resource optimizations, in a near future it will be possible to use the provided modules fully integrated on an actual on-board vehicle system. During development of the system, a processing time of 0.008147 seconds per frame was achieved using gray scale source images at an initial resolution of 640 x 480. The processing time is expected to decrease in future iterations.

ACKNOWLEDGMENT

We like to express sincere appreciation and deep gratitude to all participants in this work.

REFERENCES

[1] The World Health Organization "The top 10 causes of Death" (online) http://www.who.int/mediacentre/factsheets/fs310/en/index.html (Accessed May 2013).

[2] World Health Organization World Report on Road Traffic Injury Prevention Edited by Margie Peden. Genova, 2004.

[3] Ratha, N.K., et al, (1997) "FPGA-Based Computing in Computer Vision" Computer Architecture for Machine Perception. CAMP '97. pp. 128--135.

[4] Duncan A., Buell, Jeffrey M Arnold, and Walter J. (1996) "Splash 2: FPGAs for Custom Computing Machines" Kleinfelder, Eds., IEEE Computer Society Press, Los Alamitos.

[5] Nissan's. "All Around Collision Free Prototype" [online] http://www.nissan-global.com/EN/NEWS/2008/_STORY/081111-02-e.html (Accessed May 2013).

[6] Khan, J., Niar S., Saghir M., El-Hillali, Y. and Rivenq A. (2009) "Driver Assistance System Design and its Optimization for FPGA Based MPSoC" Application Specific Processors, 2009. SASP '09. IEEE 7th Symposium, San Francisco, CA. pp. 62—64

[7] Honda Motor Company. "Honda Completes Development of ASV-3" (2005) [online] http://www.rpec.co.uk/archive/Honda\%20ASV.pdf (Accessed May 2013)

[8] Bertozzi, M., Broggi, A. and Fascioli, A. (2000) "Vision-based intelligent vehicles: State of the art and perspectives", Robot. Auton. Syst., Vol. 32, No.1 pp. 1—16.

[9] Feixiang, R., Jinsheng, H., Ruyi, J. and Reinhard, K. (2009) "Lane Detection on the iPhone" Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Vol. 30, 2010, pp. 198--205.

[10] Cario G., Casavola, A., Franze, G., Lupia, M. and Brasili, G. (2009) "Predictive Time-To-Lane-Crossing Estimation For Lane Departure Warning Systems" Paper Number 09-0312 Universit`a degli Studi della Calabri, Italia.

[11] Broggi, A. (1995), "An Image Reorganization Procedure for Automative Road Following Systems". International Conference on Image Processing. IEEE Computer Society. Vol. 3. 1995. pp. 532--535.

[12] Serra, J. (1983), "Image analysis and mathematic morphology", Academic Press, Inc. Orlando, FL, USA.

[13] Coifman, B., Beymer, D., McLauchlan, P. and Malik, J. (1998) "A Real-Time Computer Vision system for Vehicle Tracking and Traffic Surveillance" Transportation Research Part C: Emerging Technologies} Vol. 6, No. 4, August 1998, pp. 271—288.

[14] Mc. Donald, J. (2001) "Application of the Hough Transform to Lane Detection and Following on High Speed Roads" Signals Systems Group, Department of Computer Science, National University of Ireland. Maynooth, Ireland.

[15] Yang, G. (2005), "Computer Vision Hough Transform" Department of Computing, Imperial College. 2005, London.

[16] Mallot, H., Bülthoff, H., Little, J., and Bohrer, S. (1991) "Inverse Perspective Mapping simplifies Optical Flow computation" Biological Cybernetics. January 1991, Vol. 64, No. 3, pp. 177--185.

[17] Reis, B., Teixeira, J., Teichrieb, V. and KElner, J.(2009) "Perspective Correction Implementation for Embedded (Marker-Based) Augmented Reality". Universidad de Federal de Pernambuco, Centro de Informática, Grupo de Pesquisa em Realidad Virtual e Multimídia., 2009, Brasil.

[18] Hartley, R., and Zisserman, A. (2003) "Multiple View Geometry in Computer Vision" Second Edition, Cambridge University Press., Cambridge, UK.

[19] Lipski, C., Scholz,B., Berger, K., Linz, C., Stich, T. and Magnor, M. (2008) "A Fast and Robust Approach to Lane Marking Detection and Lane Tracking" SSIAI '08 Proceedings of the 2008 IEEE Southwest Symposium on Image Analysis and Interpretation. pp. 57--60.