Road Detection for Driving Applications based on Histograms

I. Cousseau
UNS
Bahia Blanca, Argentina

D. Gigena Ivanovich
IIIE
CONICET
Bahia Blanca, Argentina

P. Julian
DIEC-IIIE
UNS-CONICET
Bahia Blanca, Argentina

***Abstract*—A simple method to detect the boundaries of a road from images is proposed. General considerations for the calculation of a bird's-eye view transformation and the use of histograms are established. A high-level algorithm is developed and implemented at RTL level, which is further synthesized for an FPGA.**

***Index Terms*—Road detection, lane detection, histograms.**

I. Introduction

Road detection is a central issue in the development of autonomous vehicles. A method designed for this purpose has to fulfill certain requirements: low processing time, low power consumption, robustness and portability. In particular, a low processing time is critical for use in real-time applications.

FPGA (Field-Programmable Gate Array) devices are a useful platform for hardware computational acceleration due to their low cost and reconfigurability. Applications for road detection are developed in [1]–[3], focused on the efficient implementation of the Hough Transform, which is a commonly used technique for line detection in images. However, the detection performance of the Hough Transform declines if the road is not well marked, i.e., it presents incomplete or unexpected lane markers [4]. Moreover, road shape differs from a line in a number of practical situations, for example, if the road lines painting is degraded or the road curb is irregular.

This motivates the development of a different approach in our paper, oriented to take into account more general conditions. A simple detection method is implemented using histograms. Besides, a reduced number of operations tends to result in low processing time and resource utilization, and serves as a good basis to add greater complexity in future applications.

Previous work based on the calculation of histograms has been done in [5], where histograms are used to detect potential lane-marker objects which are then analyzed based on their geometries, shapes and positions. Our approach incorporates the use of a bird's-eye view transformation (also called Inverse Perspective Transformation). This transformation is used to greatly simplify the boundary detection problem by converting its shape to an almost straight vertical line.

The architecture is developed with the specific purpose of handling video sequences produced by an event based CMOS imager [6].

II. Preliminary Concepts

A. Bird's-eye view transformation

A bird's-eye view (BEV) of a scene is obtained if the observer or camera is situated at a certain height perpendicularly to the ground plane. This view removes perspective effects from the captured images and allows distances in pixels to be related linearly to real world distances in meters [7]. If the camera intrinsic (focal length and principal point) and extrinsic (height and inclination angles) parameters are known, an homography can be defined [8].

Assuming a pinhole camera model, the camera's mapping between the 3D world and the 2D image is defined by the camera matrix

$$P = K\,[R \mid t] \tag{1}$$

where *K* is the intrinsic or calibration matrix, *R* is the rotation matrix and *t* is the translation matrix (these last two are related to the camera location). In homogeneous coordinates

$$\vec{x_p} = P\vec{X_m} = P \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

where $\vec{x_p}$ is an image point (in pixels) and $\vec{X_m}$ is a 3D-world point (in meters). It is assumed that the ground plane satisfies *Z = 0* and the camera matrix can be reduced to

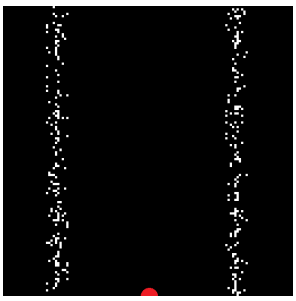
$$\vec{x_p} = H\vec{x_m} \tag{3}$$

where *H* is the homography matrix and $\vec{x_m}$ is a 2D-world point (in meters). The inverse of the homography matrix is then used to retrieve the original scene information,

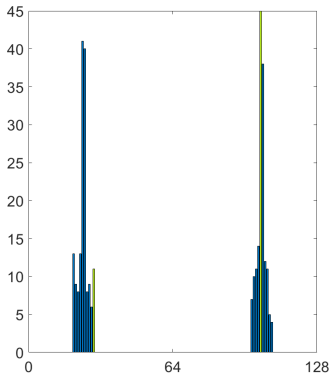
$$\vec{x_m} = H^{-1}\vec{x_p}. \tag{4}$$

B. Road image analysis with histograms

An expected binary bird's-eye view image of a road is shown in Fig. 1, which represents an ideal detection scenario. A histogram is calculated summing the pixel values corresponding to each column. It is assumed that the camera is aligned with the center of the image and, under normal driving conditions, each side of the road is located in each half of the image. Starting from the center of the histogram, the first

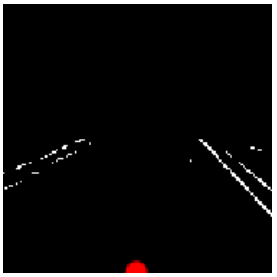


(a) Bird's eye view image. Camera position is indicated with a red dot.

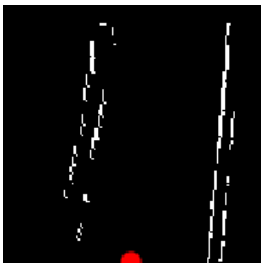


(b) Histogram calculated from the bird's eye view image. The first local maximum for each side of the histogram (starting from the center) is indicated in green.

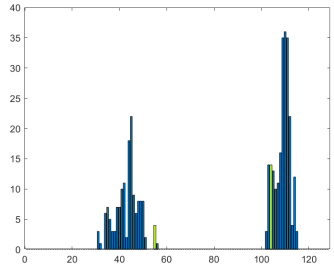
Figure 1: Expected bird's eye view image to use for road detection and its corresponding histogram.



(a) Original view frame.



(b) Bird's-eye view transformation.



(c) Histogram calculated from bird's eye view. The first local maximum for each side of the histogram (starting from the center) is indicated in green.

Figure 2: Bird's eye view transformation and histogram calculation.

local maximum in both directions (from right to left for the right half of the histogram, from left to right for the left half of the histogram) provides information about the approximate location of the road border.

However, this basic calculation is vulnerable to detection errors that may arise in real world situations. For example, due to road irregularities there could be small spots in the road that may result in erroneous maxima detected. Fig. 2 shows an image taken with an event camera and its associated bird's-eye view. The detection can be made more robust by imposing that the local maximum must be higher than a fixed threshold value.

For the purpose of establishing this value, the road is approximated by a solid straight line with a given width and inclination angle (to a vertical line). It is assumed that histogram values have to be higher than the hypothetical histogram values of a minimum reference line. This threshold value can be approximated by

$$thr = \frac{w}{\sin(a)} \tag{5}$$

where w is the expected minimum width of the road (in pixels) and a is the maximum expected inclination angle (usually between $10°$ and $20°$). The minimum expected width of the road in pixels can be related to a distance in meters (usually between 5 and 10 cm).

Moreover, the BEV image could be cut in an arbitrary number of horizontal slices and then a histogram would be calculated for each slice. The threshold value is valid for each histogram and can be applied to them. To choose a reasonable number of slices it must be considered that the slice pixel height must be kept relatively distant from the threshold value.

Dividing the image in slices and applying the analysis described before results in a number of points that belong to the road for each side of the image, instead of just one. These points could be compared among themselves to discard outliers and improve the robustness of this method. This approach is not developed in this paper and it is left for future work.

Finally, for simplicity, the BEV image could also be cut in two halves vertically to calculate a histogram for each of them. As mentioned before, it is assumed that each half of the image corresponds to each side of the road. The analysis for each half of the histogram only differs in the local maximum search direction, i.e., for the histogram that belongs to the left half of the image, the first maximum is searched from right to left and, for the histogram that belongs to the right half of the image, the first maximum is searched from left to right.

C. FRIS2D event based imager

The FRIS2D is the fourth design version in the series of read-out ICs for infrared imagers using in-pixel sigma-delta converters and automatic pixel selection [6]. This chip contains an array of 127×127 pixels on a $40\,\mu\mathrm{m}$ pitch and has been fabricated in an IBM 90 nm process. The schematics of the FRIS2D cell is shown in Fig. 3. It consists of a photodiode that discharges an integrating capacitance, followed by a synchronous sigma-delta converter that drives a feedback transistor that injects charge to balance the photodiode current. The output of the converter feeds a decimator that has a non-uniformity correction per pixel.

The FRIS2D is capable of outputting pixel values under certain conditions, for example when the change in the absolute or relative value has changed by a certain programmable amount. As a consequence, a sparse output is obtained with

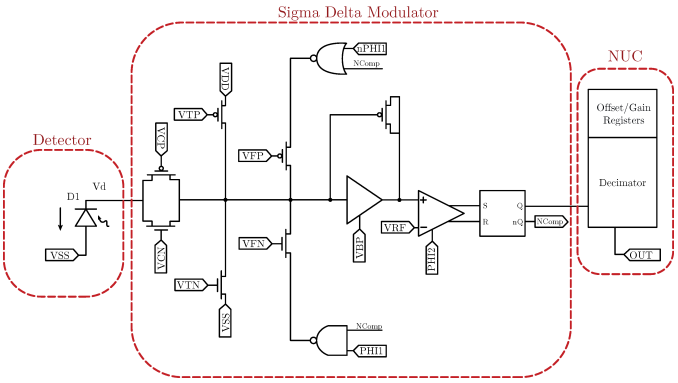


Figure 3: FRIS2D schematics.

only significant information. For the purposes of the current paper, a standard camera was used to collect images and an emulator was developed to produce an event based output as is explained next.

D. Data collection

A camera was characterized and used to collect data. Camera intrinsic parameters were obtained using the Camera Calibrator App in MATLAB™ [9]. The camera was located at a fixed position in order to take a set of 30 images of a calibration board in different positions.

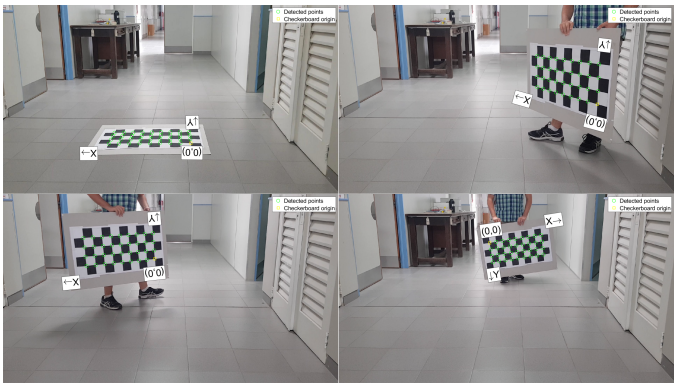


Figure 4: Checkerboard images used for camera calibration.

As shown in Fig. 4, the calibration pattern is a checkerboard. The size of the squares is known and the application uses this information to estimate the parameters. The procedure to obtain the camera extrinsic parameters is similar but the checkerboard is located on the ground and the application infers the reference zero height plane from it.

In order to collect real data, the camera was mounted on the front of a vehicle facing forward in the road direction and a Full-HD video was recorded. A Matlab script is used to mimic the output of an event detection imager. First, successive frames are read from the video and converted from RGB to grayscale. Value variation between frames for each pixel is analyzed. If this variation is greater than a minimum threshold and lower than a maximum threshold, it is stored in a new image. Then, Matlab's image resizing function (*imresize*) is used to downsize video resolution to 128×128. This is an emulation of ideal event detection and does not model FRIS2D pixel noise, but inherits its image resolution. Finally, the obtained images are binarized.

III. Proposed Algorithm

We are assuming that there is a table already preloaded with a list of BEV pixel coordinates and the corresponding coordinates of the pixels in the original image. This table is built to avoid the computation of coordinates during execution time. Fig. 5 illustrates the table that contains row and column of the BEV image on the left and the associated row and column of the original image on the right. Whenever a BEV pixel needs to be computed, the table is used to obtain the original image coordinates, access the original image and get the value of the pixel.

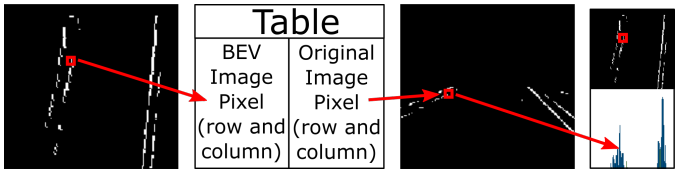


Figure 5: The path of a BEV image pixel to its corresponding histogram value.

The proposed algorithm is summarized in Fig. 6. It begins with the storage of the binary image of the road. As the objective of the algorithm is to find only the first local maximum of a set of column histograms, it is not necessary to calculate all the histograms. The BEV image pixel values can be summed and compared in an orderly fashion, i.e., summing only those pixels corresponding to a column to produce one element of the histogram and then proceed to compare it with the previous one to determine if it corresponds to a local maximum.

A. Architecture

The resolution of the BEV image to be analyzed is 128×128 pixels. This image is subdivided into two halves horizontally and then in eight slices vertically, resulting in 16 blocks. For each of these blocks, the previous algorithm is applied. Fig. 7 shows the resulting histograms for the case when they are calculated completely. Fig. 8 shows the detected histogram maxima in the original, event detection emulation and bird's eye view images.

Each of the blocks obtained after dividing the image is a 16-by-64 set of pixel addresses (14-bit registers). As mentioned before, the inverse BEV transformation is implemented as a list of related pairs of pixel addresses (i.e. a BEV frame pixel address and its corresponding original frame pixel address) stored in a table. The BEV pixel address is used as the table address to be read and the original frame address as the value read. Then, the original frame pixel address is used to retrieve a binary pixel value from the image stored in memory.

The table is sorted in a particular way. First, the 16 BEV image pixels that correspond to each block column are

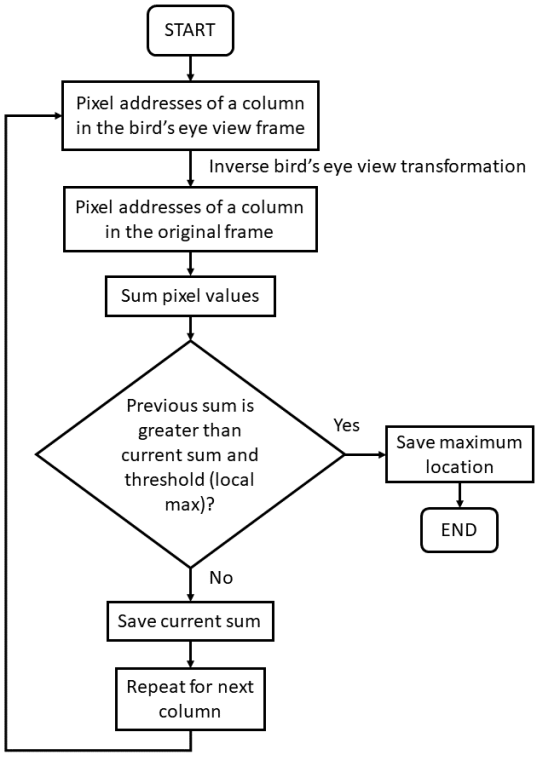


Figure 6: Algorithm flowchart.

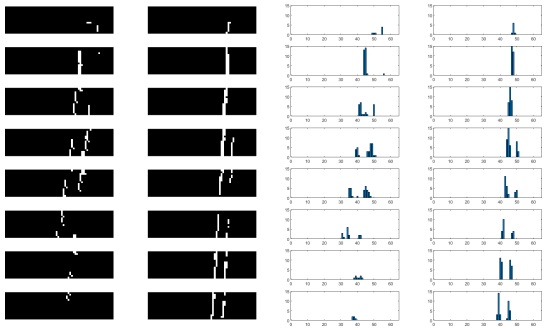
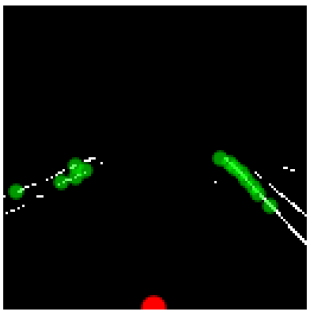


Figure 7: Split bird's-eye view frame (for Fig. 2-a) and the corresponding histogram for each block.

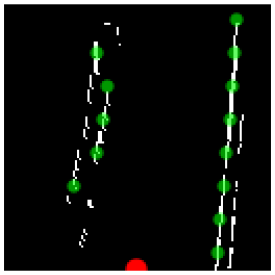
grouped. Then, these groups are sorted in the same order as the columns would be picked and compared for histogram calculation. Thus, the table can be read sequentially by reading a set of 16 successive table values (block column) at a time. Table calculation, sorting and loading into memory is done offline. Table dimensions are 14 bits $\times$ 2^{14}.

Once read, pixel values are summed and compared with the previous sum (first, the previous sum has to be higher than the detection threshold). At the same time, the BEV column address is checked to verify if the end of a block was reached. In general, the result of these two comparisons determines the next BEV column address to be read (next address in normal operation or jump to next block if a local maximum was found) and the value stored as the previous sum (save current sum in normal operation or reinitialize to zero if a local maximum was

(a) Original frame.



(b) Emulated FRIS2D frame (event detection and 128 $\times$ 128 resolution).



(c) Bird's eye view frame.

Figure 8: Application of FRIS2D emulation and histogram detection to a frame. Detected maxima in histograms are indicated with green dots (in the center of their corresponding bird's eye view horizontal slices). Camera position is indicated with a red dot.

found). Finally, the BEV column address is used to access its corresponding BEV pixel addresses in the table and the cycle starts over. At any time, if a local maximum is found, the previous BEV column address is stored (this is the maximum column or location).

A simplified description of the design is shown in Fig. 9. It can be divided into two functional blocks. The inverse BEV block is used to manage the inverse BEV transformation, retrieve original image information stored in memory and provide this data to the processing block. The processing block executes the add/compare operations needed to process data and determine histogram maxima location. Parallelization is added through the utilization of more add/compare units (a total of 4 units as indicated in Fig. 9) and segmentation of the inverse BEV transformation table.

These two functional blocks work simultaneously in a short pipeline, which in one clock cycle performs the following operations:

- a set of pixel values is processed by the processing block.
- the next set to be processed is loaded by the inverse BEV block and made available at the input of the processing block for the next cycle.

A few extra registers are added for this purpose. Moreover, a special case arises in the immediate cycle after maximum detection: the next data to be processed is already loaded

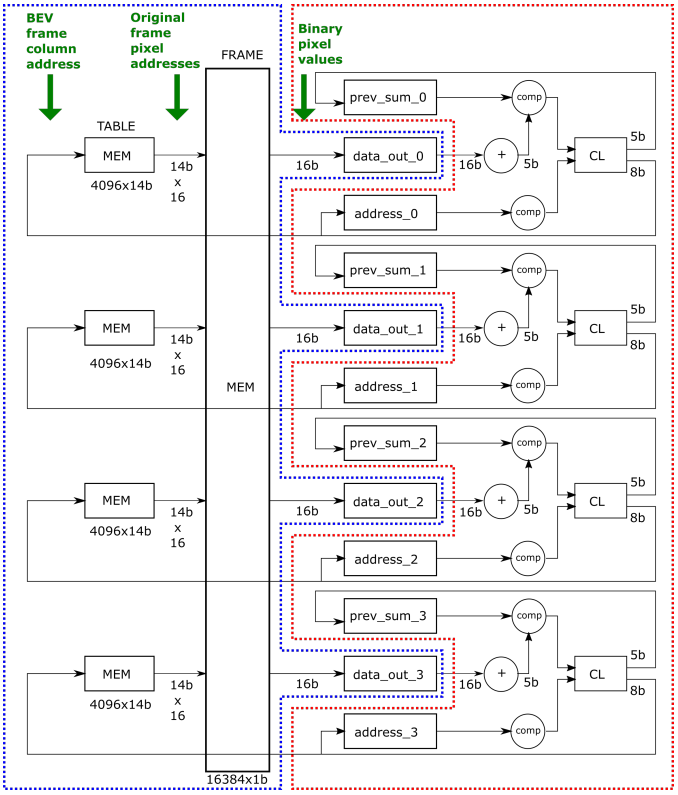


Figure 9: RTL design with functional blocks delimitation. Inverse BEV block and processing block are indicated with blue and red dashed lines respectively.

but the maximum was already found, so this data must be ignored. The processing block is implemented as an FSM to take into account this behavior, with a 1-bit state register: state 0 corresponds to normal operation, state 1 is only reached when a maximum is detected and produces a one-clock cycle wait.

B. Simulation and synthesis results

The design is written in Verilog HDL, simulated and synthesized for implementation in the Xilinx™ Arty Z7-20 Development Board with the Xilinx™ Vivado Design Suite. This design can be synthesized with a maximum clock of 100MHz. Utilization and power reports are indicated in Table I and Fig. 10 respectively. Design simulation captures are shown in Figs. 11-a and 11-b.

Table I: Vivado Synthesis utilization report.

Resource	Utilization	Available	Utilization (%)
LUT	4137	53200	7.78
FF	236	106400	0.22
IO	10	125	8.00

For the particular frame used in Fig. 11 simulation, maxima detection is accomplished in around 2000 ns. Nevertheless,

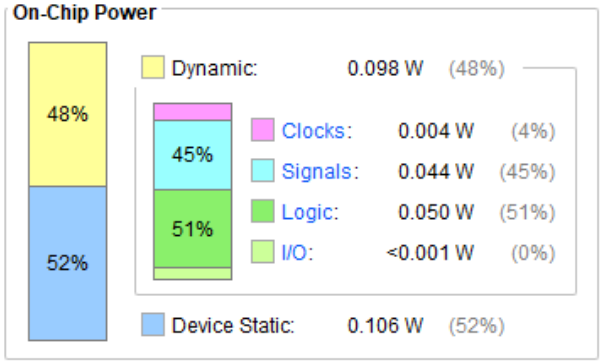


Figure 10: Vivado Synthesis power report.

a worst case scenario can be approximated by taking into account the BEV image subdivision done previously,

$$Detection\ time = \frac{Clock\ period \times Block\ width \times Blocks}{Add/compare\ units}. \quad (6)$$

If the clock frequency is 100MHz,

$$Detection\ time = \frac{10\text{ns} \times 64 \times 16}{4} = 2560\text{ns}. \quad (7)$$

IV. Conclusions

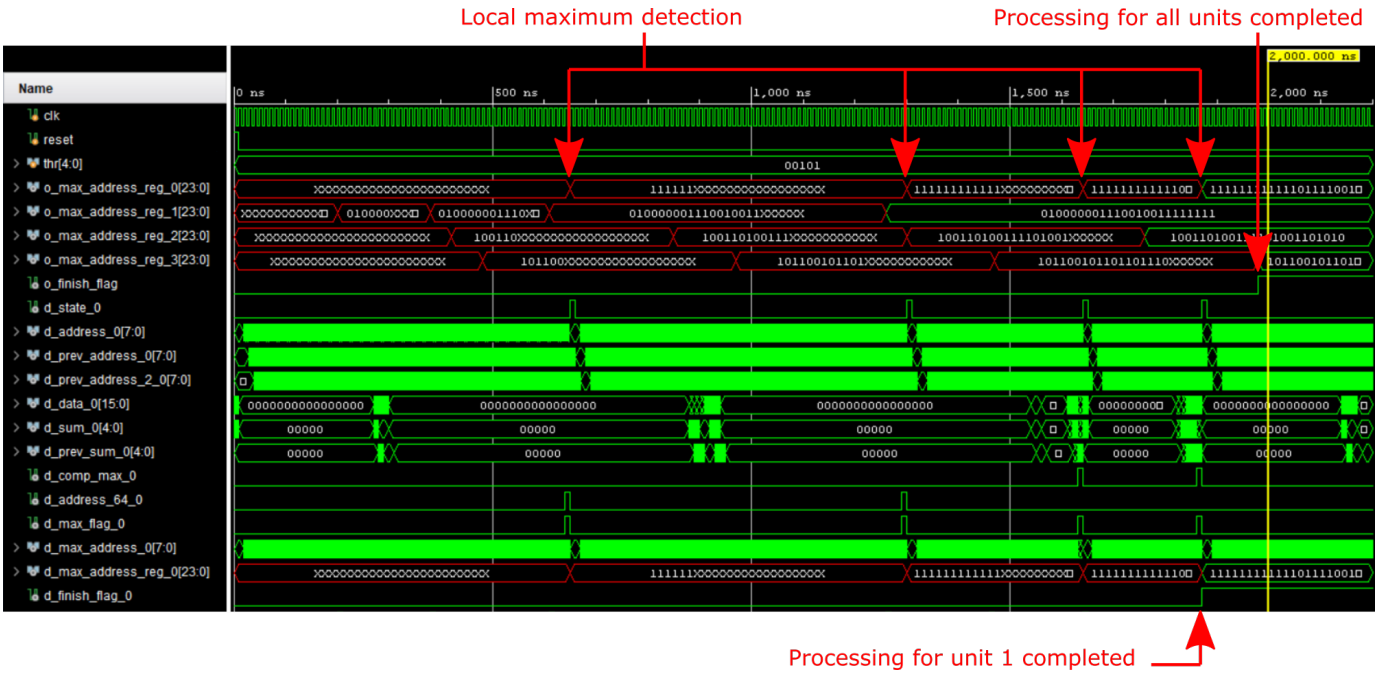
In this paper we have proposed an architecture for road detection using the BEV algorithm oriented to event based cameras. As a first step, we characterized a camera to collect real images. Then, we developed an algorithm based on histograms to detect road boarders. We implemented an inverse BEV transformation using a table-based approach. Histogram use for road detection was established with a few improvements, such as the detection threshold value and image slicing to calculate more histograms in parallel. An algorithm was proposed and translated into a simple RTL design, which was synthesized with a 100MHz clock.

The proposed method results in a number of points (as the number of maxima locations) that belong to the road border. Further processing could be done using this points. Linear interpolation to discard possible outliers is a simple correction and could greatly improve the quality of the obtained information. Comparison of points between frames in time could be implemented with a variable degree of complexity. For example, the obtained information could be fitted into the road model proposed in [10] for a Kalman filter-based tracking algorithm.

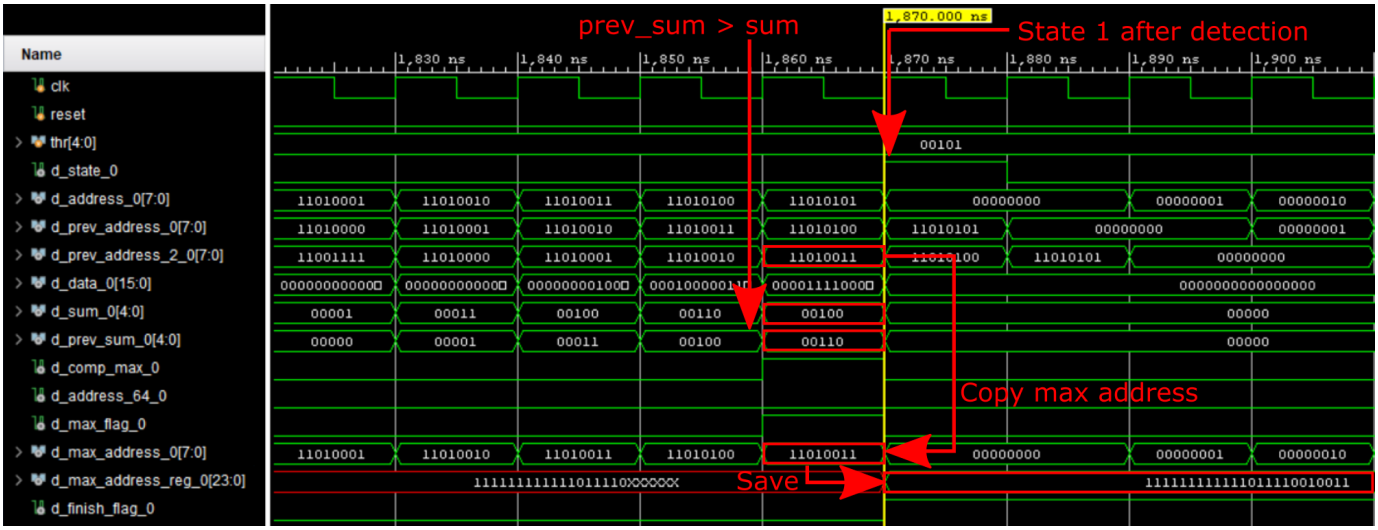
Taking into account its low resources and time cost, this method could be used paired with additional tools. For example, morphological image processing operations or clustering [11] require more complex implementations and could use the additional quick information to improve or double-check results.

References

[1] I. El Hajjouji, S. Mars, Z. Asrih, and A. El Mourabit, "A novel FPGA implementation of Hough Transform for straight lane detection", *Engineering Science and Technology, an International Journal*, vol. 23, pp. 274–280, 2019.



(a) Output signals of the complete design and all signals (including internal signals) of one add/compare unit.



(b) Zoom in to maximum detection.

Figure 11: Design simulation with a 100MHz clock frequency.

[2] E. Shang, J. Li, X. An, and H. He, "A real-time lane departure warning system based on FPGA", *14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 1243–1248, 2011.

[3] X. Lu, L. Song, S. Shen, K. He, S. Yu, and N. Ling, "Parallel Hough Transform-based straight line detection and its FPGA implementation in embedded vision", *Sensors (Basel, Switzerland)*, vol. 13, pp. 9223–47, 2013.

[4] J. B. McDonald, R. Shorten, and J. Franz, "Application of the Hough Transform to lane detection in motorway driving scenarios", *Proceeding of Irish Signals and Systems Conference*, 2001.

[5] J. P. Gonzalez and U. Ozguner, "Lane detection using histogram-based segmentation and decision trees", *2000 IEEE Intelligent Transportation Systems. Proceedings (Cat. No.00TH8493)*, pp. 346–351, 2000.

[6] J. H. Lin, P. Pouliquen, A. G. Andreou, A. Goldberg, and C. Rizk "Flexible readout and integration sensor (FRIS): a bio-inspired, system-on-chip, event-based readout architecture", *Proc. SPIE 8353, Infrared Technology and Applications XXXVIII, 83531N*, May 2012.

[7] S. Abbas and A. Zisserman, "A geometric approach to obtain a bird's eye view from an image", *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 4095–4104, 2019.

[8] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed., Cambridge University Press, ISBN: 0521540518, pp. 153–158, 2004.

[9] "Mathworks: Calibrate a monocular camera." [Online]. Available: https://www.mathworks.com/help/driving/ug/calibrate-a-monocular-camera.html

[10] H. Sahli, P. Muynck, and J. Cornelis, "A Kalman filter-based update scheme for road following", *1996 IAPR Workshop on Machine Vision Applications*, pp. 5–9, 1996.

[11] R. Ajaykumar, A. Gupta, and S. N. Merchant, "Automated lane detection by K-means clustering: A machine learning approach", *Electronic Imaging*, vol. 2016, pp. 1–6, 2016.