Fast and Resource-Efficient Hardware Implementation of Modified Line Segment Detector

Fuqiang Zhou, Yu Cao, and Xinming Wang

***Abstract*— Lines are significant features enclosing high-level information in an image. The line segment Detector (LSD) Algorithm with low error rate is a widely used method to extract lines in images effectively and accurately. However, the algorithm on PC performs too costly both in time and resources for the real-time video processing. This paper provides a fast and resource-efficient hardware implementation solution for a modified LSD algorithm on Field Programmable Gate Arrays (FPGA) for real-time line detection. The task-level pipeline structures are exploited fully in a stream process mapped to the hardware architecture free of frame buffer. Our proposed hardware implementation processes in a stream-in–stream-out manner with little consumption of the on-chip block RAM to store intermediate values. We first employ hardware Gaussian filter and adjust Canny edge detection to obtain an edge map at single-pixel width. Then, a novel structure of region growing model based on dynamic rooted tree is used to detect line segment regions accurately with a latency of only a few rows of pixels. The low cost in time, on-chip resources, and power consumption makes our proposed algorithm suitable for portable real-time streaming video processing applications using line segment features, such as Lane departure warning systems. It can also be applied in real-time machine vision systems that use line segments information for further recognition or stereo correspondence and many others. The proposed algorithm is synthesized and tested on XC7Z020 FPGA with high reliability, accuracy speed, and low cost in both resources and energy.**

***Index Terms*— Embedded image processing, FPGA, line segment detector, real-time.**

I. Introduction

LINE detection plays a key role in several embedded vision application domains. Given that line segments provide a high-level description of the objects in an image [1], straight line detection has been widely used in many industrial applications such as image analysis, smart robots, intelligent vehicles, pattern recognitions [2], crack detection in materials [3], stereo matching [4] and many others [5]. In a real-time Lane Departure Warning System [6], the first step is to detect the lanes fast, accurately and robustly. An FPGA-based line detection system will save much time, hardware resources and energy. For a binocular measurement system, with several prominent lines in both left and right images extracted successfully, stereo correspondence will be made based on line parameters. This method is more simplified compared with the traditional methods applying the matching algorithm on the whole image or part of the image with an empirical parallax searching range. The line-based matching operation makes full use of the acquired line information, accelerating the whole binocular measurement system, which is of vital importance in real-time visual measurement. Therefore, as a front-end step of video processing in vision systems, line detection must be realized with high accuracy and speed on a hardware platform. Among the existing line detection algorithms, Hough Transform has remained a standard method for many years because its theory is complete and its performance is stable on low-texture pictures. In recent years, many variants of Hough Transform have emerged [8], [9], both on software and hardware platforms. However, all Hough-based methods consume enormous time and resources due to its operations of transferring points into Hough space onwards and backwards with many non-linear calculations. Most importantly, under complex images with much texture, false detection emerges quickly, usually producing a serious problem for the following video processing. In summary, Hough-based methods cannot satisfy the demand of line detection for video processing with respect to time and accuracy efficiency. Researchers also developed some other line detection methods. Kahn *et al.* [10] only used gradient orientations rather than magnitude, with a selection criterion as a final step. Some propositions of such criteria are exploited by experts [11], [12]. However, false detection control relative to these classical methods is still not satisfactory.

To solve the existing problems of the line detection methods, we urgently need a new line detection method with good line detection and few false detection. In 2010, the Line Segment Detector (LSD) algorithm proposed by von Gioi *et al.* [13] cumulated most of the advantages of the previous algorithms without most of their drawbacks. This linear-time line segment detector requires no parameter tuning and yields good line detection results as shown in [14]. This algorithm is mainly composed of five computational steps and a validation step. In LSD algorithm, level-line angle is owned by each pixel to produce a "level-line field" which is segmented into connected regions of pixels called "line support regions". The pixels that share the same level-line angle up to a certain tolerance as shown in Fig. 1 are collected in a certain region. Each line

Manuscript received November 20, 2016; revised March 1, 2017 and July 31, 2017; accepted August 26, 2017. Date of publication August 30, 2017; date of current version November 5, 2018. This work was supported by the National Natural Science Foundation of China under Grant 61372177. This paper was recommended by Associate Editor P. Schelkens. *(Fuqiang Zhou and Yu Cao contributed equally to this work.) (Corresponding author: Fuqiang Zhou.)*

The authors are with Beihang University, Beijing 100083, China (e-mail: zfq@buaa.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2017.2746753

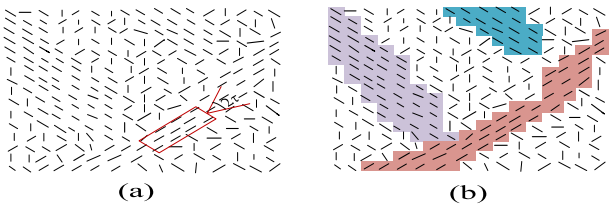

Fig. 1. LLA diagram. (a) Level-line field; (b) Line support regions.

support region is a candidate for a line segment which is first approximated by a rectangle with the region's principal inertial axis being used as the rectangle's main direction. The final line parameters including start point, end point and line width, are determined via the corresponding rectangle parameters. In general, the five computational steps are summarized as follows: 1) image filtering; 2) gradient and level line angle computation; 3) gradient pseudo-ordering; 4) region growing; and 5) rectangular approximation and NFA computation. The validation step is based on an "a contrario approach" and the Helmholtz principle, which states that there should be no perception (line detection) on an image of noise.

The hardware implementation of the LSD algorithm should perform a fast and accurate line detection for the whole vision system. Benefited from FPGA's pipeline and parallel features, it is suitable for many image-processing areas with low cost in time, on-chip resource and power. The FPGA-based video processing area includes background subtraction [18], high-speed face detection [19], stereo vision processing [20], [21], feature extraction [22] and many others [5], [7]. Video pre-processing part involving much convolutional processing is fully implemented on FPGA. Video pixels stream into the FPGA in raw greyscale data. After processing, line segments' parameters and the edge map stream out for further processing.

In this paper, based on Grompone's software-implemented LSD [14], a modified all-hardware LSD algorithm on FPGA is first introduced for real-time line extraction. Our focus is on implementing the line extraction with a fixed low latency, high accuracy, low cost and a large throughput so that it can be used in embedded real-time video processing applications. Directly applying the original LSD algorithm on the dedicated hardware is difficult because it requires much memory that exceeds the limits of the FPGA hardware resources. Therefore, in this paper, we simplify the algorithm to fit the hardware architecture of the FPGA. We use Gaussian filter to smoothen the image and Canny detector [15] to detect the pixel-width edge ahead of the line segment detection. Filtered Canny edge map is used for line segment region growing with a convolutional dynamic tree architecture. Finally, the regions that satisfy the line segment criteria will trigger the output signal. We implement the whole modified algorithm on the FPGA platform. In order to test the detection effect of our modified LSD algorithm, the parameters calculated from FPGA are sent to a personal computer (PC) and then the detected lines are drawn.

The remainder of this paper is organized as follows: Section II reviews the software implemented LSD algorithm. Section III presents the architecture of our modified LSD hardware implementation. Section IV analyzes the proposed algorithm in validation of the detected lines and computational complexity. Section V shows the experiments and the assessment of our algorithm. Finally, the paper concludes in Section VI.

II. Description of the LSD Algorithm

The LSD algorithm detects the line segment by growing region of points with aligned Level Line Angle as suggested in Fig. 1. Each small line represents one pixel with the angle it stands indicating its LLA (Level Line Angle). The software implementation of the LSD algorithm can be divided into five parts called image filtering, gradient computation, gradient pseudo-ordering, region growing and rectangular approximation and NFA computation. Details are as follows.

A. Image Filtering

The input image is first filtered with a Gaussian kernel to avoid aliasing. Images are always affected by noise, Gaussian down sampling is adopted to filter out the noise and preserve good line features.

B. Gradient Computation

Given that the LSD algorithm demands the independence between pixels when conducting gradient computation, equation (1) shows the 2×2 mask adopted by von Gioi *et al.* [14]. The gradients in x and y directions are calculated as follows. The Level Line Angle of a pixel is defined as equation (2).

$$g_x = \frac{i(x+1,y)+i(x+1,y+1)-i(x,y)-i(x,y+1)}{2}$$
$$g_y = \frac{i(x,y+1)+i(x+1,y+1)-i(x,y)-i(x+1,y)}{2} \tag{1}$$

$$LLA = \arctan\left(\frac{g_x(x,y)}{-g_y(x,y)}\right) \tag{2}$$

C. Gradient Pseudo-Ordering

In software implementation of LSD, 1024 bins are created corresponding to equal gradient magnitude intervals between the smallest meaningful gradient magnitude and largest value. The least meaningful gradient value ρ is determined by the equation (3). In the original LSD paper, the value is set to 5 accordingly. Pixels with gradient less than ρ will not be further processed because they are considered meaningless for the line segment.

$$\rho = \frac{q}{\sin\tau} \tag{3}$$

D. Region Growing

Connected pixels with similar gradient level line angles are considered to be elements of one common line-support region. If the level line angles' difference value between the existing region and the neighboring candidate pixel P is in the range $[-\tau, \tau]$ defined by tolerance angle τ, the pixel will be

considered as an aligned point and added to this region. The LLA of the region will be updated according to equation (4).

$$\theta_{new} = \arctan\left(\frac{\sin(LLA_{region}) + \sin(LLA_P)}{\cos(LLA_{region}) + \cos(LLA_P)}\right) \quad (4)$$

$$\theta_{new} = \frac{LLA_{region} + LLA_P}{2} \quad (5)$$

By using trigonometric function formula, we transform (4) to (5) free of error. Each pixel in the ordered list is compared with its eight-connected neighbors to grow a line segment region.

E. Rectangular Approximation and NFA Test

Every line segment region in the processed image i will be approximated by a $M \times N$ rectangle r covering all its pixels. The rectangle will be tested to know if the number of aligned points k on the total number of points of the rectangle is surprisingly big with respect to the background model. The region's Number of False Alarms (NFA) is defined as equation (6), indicating the expectation of the number of events that a rectangle r in a random image I has the same or more aligned points. In equation (6), N_{test} is the total number of tests of considered rectangles. P_{H_0} is the probability of the event on the *a contrario* model H_0, which is a noise model for the image gradient orientation.

$$NFA = N_{test} \cdot P_{H_0}[k\,(r, I) \geq k\,(r, i)] \quad (6)$$

$$NFA = (NM)^{5/2}\gamma \cdot \sum_{j=k}^{n} \binom{n}{j} p^j\,(1-p)^{n-j},\ p = \frac{\tau}{\pi} \quad (7)$$

NFA is calculated as equation (7), where the precision value p is the probability that a pixel on H_0 is an aligned point. τ is the tolerance angle and π is the angle in the radian system. γ is the number that different precision values p potentially tried. When the rectangle satisfies $NFA \leq \varepsilon$, it passes the test. The threshold ε is set to 1, meaning an acceptance of one false detection per image on average. If a line region's rectangle passes the NFA test and has aligned point density greater than the threshold, it will be counted as a line segment. The rectangle's start and end points will be used to represent the line.

III. Proposed Hardware Architecture of LSD

In this section, our hardware implementation of modified LSD is introduced in detail.

A. Overall System Architecture

Our modified LSD algorithm mainly consists of six modules: Gaussian filtering, gradient and LLA computation, Non-Maximum Suppression, double threshold connection, mirror and region growing. Canny edge detector is well known for low error rate, good location of edge points and single edge point response, which can help FPGA eliminate massive background pixels with a single-point response in the edge. We adopt the Canny detector to simplify line segment detection. In our implementation, regions grown from single-pixel width edge map will always satisfy the NFA and aligned points' density requirement, which is proven in Section IV. So the rectangle approximation and NFA computation part is abandoned in our modified algorithm on the FPGA. Fig. 2 shows fully pipelined tasks mapped to FPGA's architecture. Data level parallelism is within each task. Tasks in different level are finished in a pipeline manner, with convolutional window bringing about only a few rows of clocks delay between them.

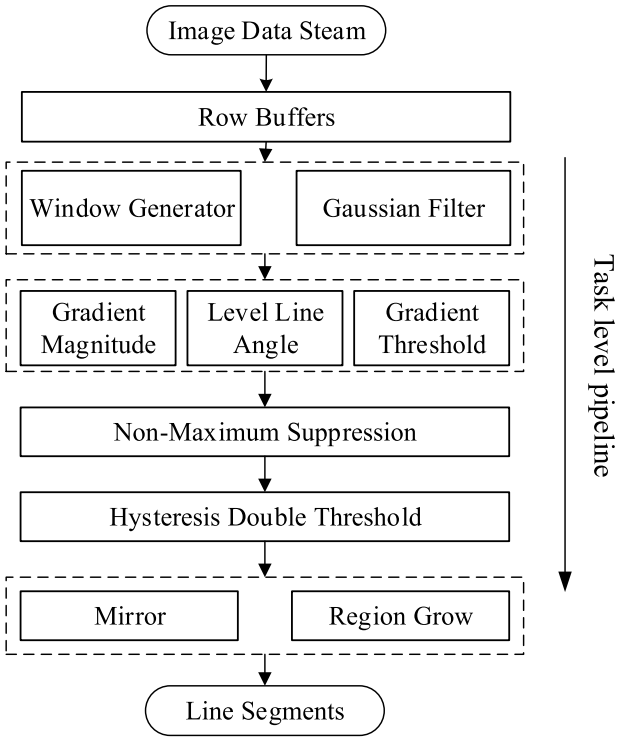

Fig. 2. Overall architecture of the proposed modified LSD algorithm.

B. Gaussian Filtering Module

First we put forward a structure of FPGA convolution module "Sliding Convolutional Window". FPGA with parallel structure performs best in convolutional applications. The convolution is done by a window sliding the image from the first column of the first row until the last column of the last row to process every pixel with its neighborhood to conduct the convolutional result. The image pixels go in as a stream from the first point in the upper left to the last one in the lower right. Fig. 3 shows the architecture and direction of a 3-by-3 "Sliding Convolutional Window".

Gaussian filtering is suitable for real-time image processing to reduce the noise influence [23]. To preserve the image brightness, the Gaussian kernel coefficients are normalized as equation (8). In our modified LSD algorithm, we use a 5×5 template described in Fig. 4(a). Coefficients are the theoretical values multiplied by a factor of 256. This avoids problems in the division operation on the FPGA. Every pixel is first represented by 17 bits, with the lower eight bits representing the fractional part and the first bit denoting the sign of the value. The coefficients of 1-D kernel can be computed by equation (9). In our paper, we choose σ as 1 [23], and the

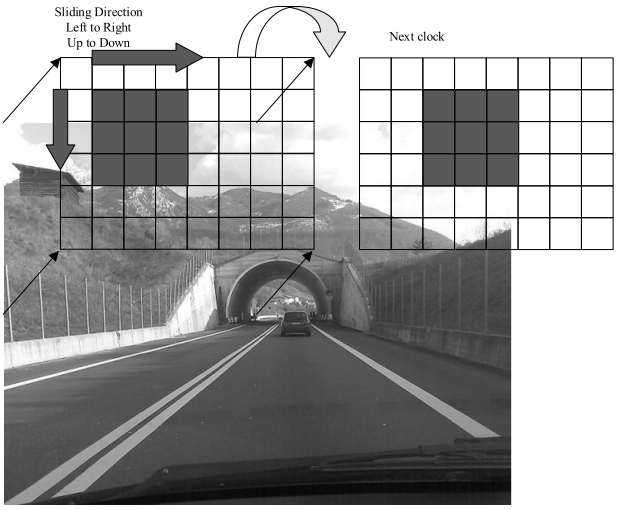

Fig. 3. Sliding convolutional window.

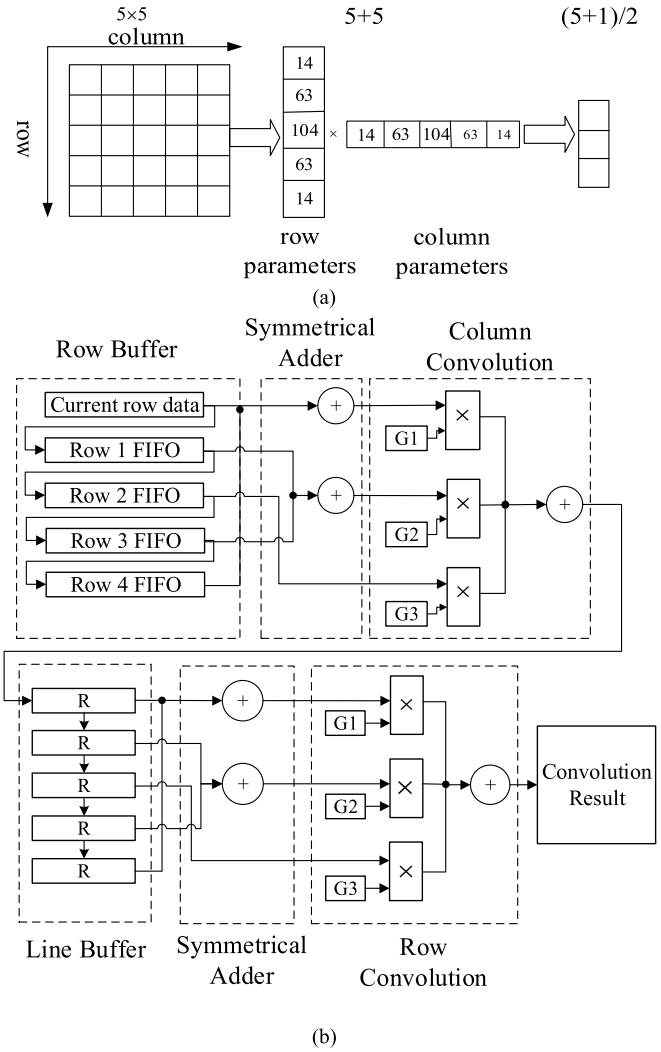

Fig. 4. Gaussian module. (a) Gaussian blur template. (b) Hardware structure of Gaussian convolution.

coefficients are shown in Fig. 4. After the filtering process, only the higher nine bits of each pixel are used to the following modules for further processing while the lower bits

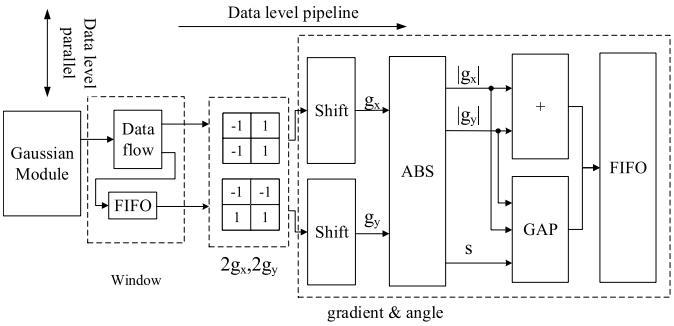

Fig. 5. Gradient calculation module.

are abandoned.

$$h(i,j) = \frac{g(i,j)}{\sum_i \sum_j g(i,j)} \tag{8}$$

The coefficients of 1-D template:

$$\begin{cases} g(i) = e^{-\frac{i^2}{2\sigma^2}} \\ h(i) = \frac{g(i)\times 256}{\sum_i g(i)} + 0.5 \end{cases} \tag{9}$$

To avoid wasting limited resources on the FPGA board, we divide the one-step convolution process into two using a pattern to convolute the columns first and then the rows. Since the symmetrical columns and rows must be multiplied by the same coefficient, they are added together first and then the sum is multiplied. The number of the multipliers is thus reduced to 6 from the original 25 as shown in Fig. 4(b).

C. Gradient and LLA Computation Module

Gradient computation is conducted after the Gaussian filtering process. The LSD algorithm adopts a 2-by-2 gradient computation masks shown in Fig. 5.

After the gradients of x and y axes of a pixel are calculated, an ABS module is used to calculate the absolute value and sign of gradients in both axes. A register stores a flag s indicating if the two gradients have different signs. In our proposed hardware implementation, the computation of square root is replaced by the absolute operation in equation (10). According to the original LSD paper, pixels with gradient smaller than $\rho = 5$ will be assigned 0 gradient.

$$G(x,y) = |g_x(x,y)| + |g_y(x,y)| \tag{10}$$

For hardware platform, deploying the arctangent operation to calculate the level line angle using CORDIC method costs too much time and resources. Therefore, in our optimized algorithm, we apply a "Gradient Angle Pseudo-ordering" (GAP) module to quantize the gradient directions within $[0, \pi]$ range. As Fig. 6(a) shows, we use multiplication and comparison to substitute dividing operation due to their better speed and accuracy performance on FPGA. We first quantize all the LLAs into eight bins to have a 3-bit gradient angle for each point. As shown in Fig. 6(b), the eight bins are then merged into four bins because the 2-bit LLAs are used for further processing. It means that we choose the tolerance angle τ

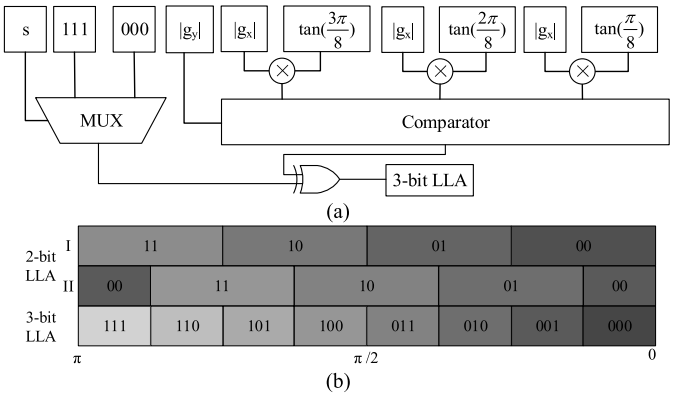

Fig. 6. (a) Gradient Angle Pseudo-ordering module. (b) Dual-division mechanism.

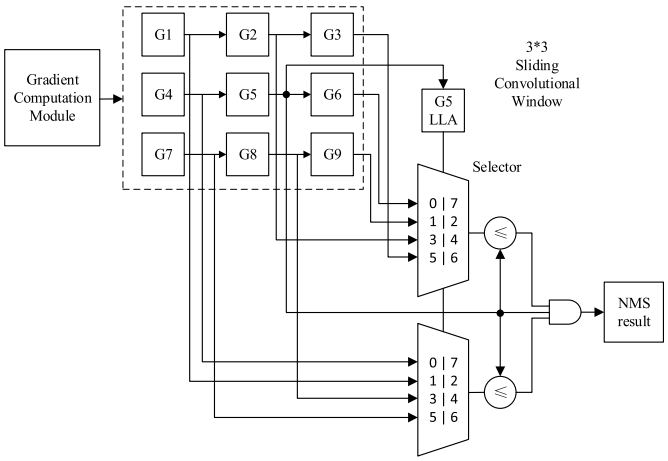

Fig. 7. Non-Maximum Suppression module.

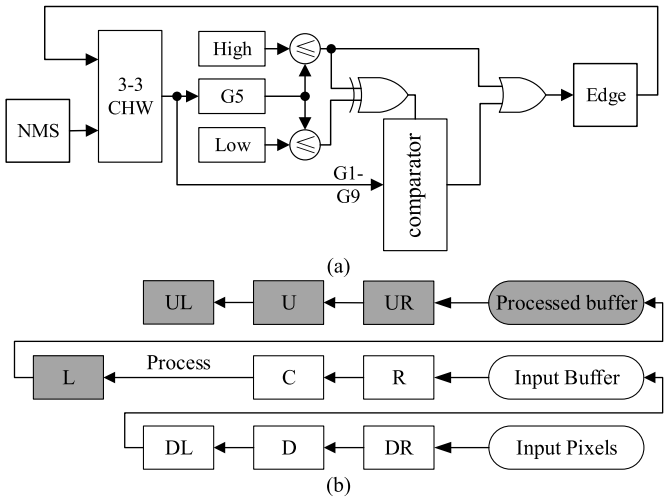

Fig. 8. (a) Double threshold connection module diagram. (b) Convolutional Heterogeneous Windows module.

as $\frac{\pi}{4}$, which is a balance between a good detection rate and a low error rate. In addition, to avoid the binning problem that some lines with angles at the borders of those bins are hard to be detected, a dual-division method is adopted. We shift the borders of one division method by half the size of the interval, ensuring that the angles on the border of one division method are on the center of another division as shown in Fig. 6(b). Then, each pixel is with two quantized LLAs based on two different division methods. In the last module of region growing, two identical modules are processing different quantized gradient angle data. Detailed experiment results on the division methods are shown in Section V.

D. Non-Maximum Suppression Module

A highly recognized property of canny edge detector is its low error rate. Non-Maximum Suppression module is the key to detecting edges more accurately by eliminating meaningless points on the rim of an edge while preserving the most significant candidate points. To obtain better line segment detection on the hardware platform, edges with a single-pixel width should be detected correctly.

A local maximum in the edge direction is a better candidate for edge point than the neighboring pixels. In this module, we compare the central pixel with the nearby two pixels chosen according to its gradient direction. If it is smaller than any of them, it will not be eligible for the edge; thus, its gradient will be set to 0. Only the central point being the local maximum will retain its gradient value. The strict NMS module is implemented as shown in Fig. 7. The output gradient magnitude and direction of the last module are denoted with G and a number indicating its location in the convolutional pattern. The mechanism of choosing neighboring points according to the 3-bit gradient direction is depicted in detail in Fig. 7.

E. Double Threshold Connection Module

The output of NMS still has many fake edge points or broken edges. To obtain the final Canny edge map, a double threshold connection module is used to connect the edge points as shown in Fig. 8(a). The low threshold can filter out the noisy or weak edge pixels, while the high threshold can determine the strong edge points. Therefore, there are three situations for a pixel to handle as follows.

1) Gradient Higher Than the High Threshold: These points are regarded as strong edge points and are accepted as one of the edge points immediately. Its gradient magnitude will be set to 255 and its gradient angle holds.

2) Gradient Lower Than the Low Threshold: These points are regarded as noise output or fake edges, and are not counted as part of an edge. We set its gradient magnitude to 0 and its gradient angle value to 4, meaning invalid.

3) Gradient Value in Between: These points are vague, which require one more validation step to determine their status. If they are connected to edge points, they belong to a certain edge and then treated like strong edge points. Else, if none of the eight-connected neighboring points is an edge point, it will be regarded as a weak edge point and treated as noise output.

To lower the error rate, we use a "convolutional heterogeneous windows" (CHW) to deal with the vague pixels. Heterogeneous character means that pixels in the windows come from different source. Given that their upper and left

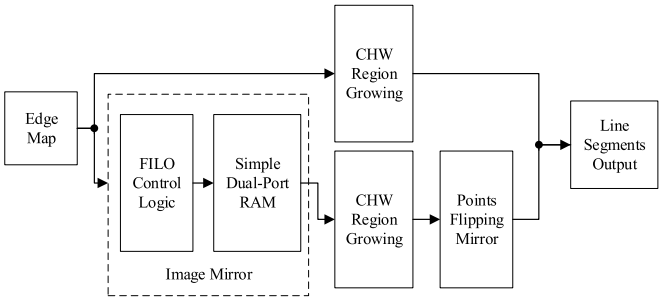

Fig. 9. Diagram of the mirror module.

neighboring pixels are post-processed, we use the results of this module to compose the left and upper part of the windows. Thus, these four points are from the final edge map instead of the prior NMS module as shown in Fig. 8(b). Post-processed points are darkened and the arrows' direction shows the flow of the data stream.

F. Image Mirror Module

Our region growing method follows the direction the FPGA scans through the whole image, which is from up to down and left to right as shown in Fig. 3. However, we should scan the line segment region in the direction of how they grow. From up to down, the right lanes on Fig. 3 grow from left to right while those left and central lanes grow from right to left. Detecting these lines requires processing in a contrary direction of scanning. To detect these lines, a mirroring module is applied to implement a scanning direction contrary to growing direction of the column number as shown in Fig. 9. After the image mirror module, the mirrored and original edge maps are processed simultaneously to obtain the line detection results.

A simple dual port Block RAM is suitable for applications with customized writing and reading memory because writing and reading are accessed via separate ports. Therefore, we use a specified controlled dual port BRAM to implement the mirror module. Before an eligible line segment triggers the output, its pixel coordinates are flipped in the horizon to fit the original pixel map. However, this mirror operation certainly brings one row of extra latency due to its First-In-Last-Out (FILO) manner.

G. Region Growing Module

After the Canny edge detector module, we have an edge map to do line segment region growing to detect lines as shown in Fig. 10. A line segment is a group of aligned points with gradient angles near the LLA of the region. A line segment is represented by a tree with each point being a node. We use a novel dynamic rooted tree to grow and record the region information. This information includes an 8-bit counter to count the number of region points, 3-bit region LLA and two 19-bit addresses denoting the starting point and ending point, respectively. Basically $R[i]$ records the region information on the column label $[i]$ until the last row of the currently processing pixel. An FIFO is used to store region

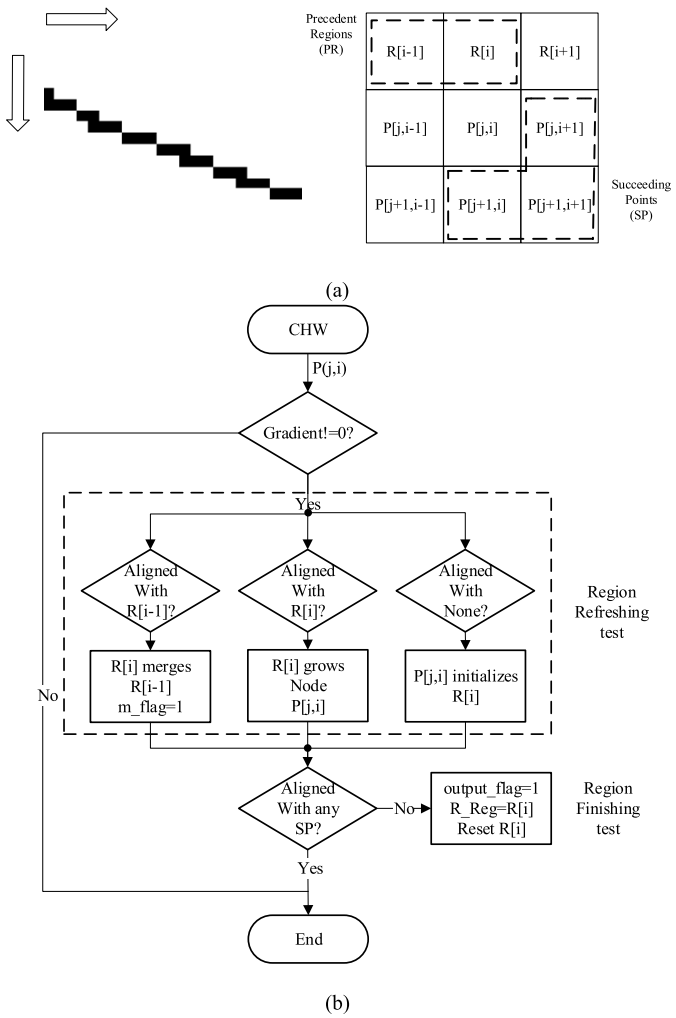

Fig. 10. Diagram of region growing convolution operation. (a) Convolutional region growing direction and pattern. (b) Region growing chart.

information. The CHW is again applied as shown in the right side of Fig. 10(a).

The gradient angle of the central point will be compared with its neighboring regions and points using the "and" operation. Given that our gradient angle is quantized with a 2-bit number, a positive output denotes that their LLAs' absolute difference is less than the angle tolerance threshold, resulting in aligned angles. Otherwise they are not aligned. The precedent regions (PR) are then compared using the region refreshing test to determine how they are refreshed. The region finish test is done by comparing with the succeeding points (SP) to judge if a region has ended growing. Both tests are done in the first rising edge, ensuring a correct refreshing of region $R[i]$. The growing process shown in Fig. 10 mainly deals with three situations.

1) Region Growing: When point $P[j,i]$ has an aligned direction with the line segment tree $R[i]$ exactly on its position, the region $R[i]$ grows downwards. The root keeps its location while enclosing one more node. Thus $R[i]$ will be updated with the number counter plus one and the end point is set to the instant address.

When an edge point $P[j,i]$ does not have an aligned region previously, it will initialize a region $R[i]$ with its address as

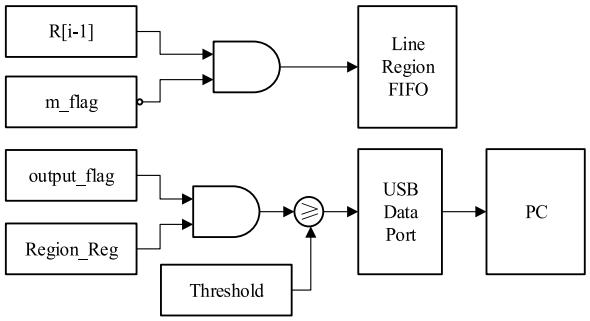

Fig. 11. Line region FIFO refreshing and output module.

the starting point, its direction as the region LLA and the counter will be set to one. A new tree starts from the point.

2) Region Merge: When a point and its left region $R[i-1]$ share the same LLA direction, the left region will be merged. Thus the root of the left tree is transferred to the right. The right tree $R[i]$ will merge with the left tree by inheriting its information, growing the point as a node and then resetting the left region. This situation turns a merge-flag register m_flag to positive and an all-zero number representing the tree located in $[i-1]$ is going to be stacked into the FIFO. Therefore, the FIFO will be renewed in a clock's latency because regions might be merged by its right point and thus be reset. However, the next time to grow $R[i-1]$ is a row of clocks later, meaning the one-clock latency will not result in any delay of the successive process.

3) Region Reset: If none of the succeeding points (SP) aligns with the current region, the region cannot grow anymore and is sent to the output module. Refreshed $R[i]$ value will be given to output module and $R[i]$ is set to all-zero. If the $R[i]$ has enough number of points, its information including starting point and ending point addresses and counter will be output to PC through a USB port. Fig. 11 depicts the region FIFO refreshing and output module.

After processing the whole edge map by growing the regions' dynamic rooted trees, the hardware finishes the line segment detection.

IV. Algorithm Analysis

This section proves that the lines detected by our modified hardware implemented LSD algorithm are meaningful in the original software implemented LSD's definition. Furthermore, we make a computational complexity analysis of our proposed algorithm and compare it with that of the original LSD.

A. Validation of Line Segment Region

In our proposed implementation, we eliminate the rectangle approximation and NFA computation modules because they require serial float-point processing that can hardly be mapped to hardware's parallel architecture. We will prove that the detected lines are still valid in the original definition of LSD.

First a region grown by our algorithm is noted as a set of n points, whose LLA angles are within the same angle range.

$$\{P_1, P_2, \ldots P_n\}, \alpha_i \in [\frac{m}{4}\pi, \frac{m+1}{4}\pi], i = 1, 2, \ldots n$$

The line segment region is grown from a one-pixel width edge map, as shown in Fig. 10(a). Then, we use a $1 \times n$ rectangle to approximate the region. According to original LSD, its regional LLA angle will be updated as equation (4), which equals to equation (5). From mathematical induction we know that the region's LLA angle α_0 must be within the same range as all the region points. Thus every region point's absolute angle difference with α_0 is smaller than τ, making it an aligned point. Therefore we have $N = k = n$ for every detected region, whose aligned point density will be 1. It is greater than the original density threshold 0.7.

The NFA of the totally aligned region is computed as follows.

$$NFA(n) = (NM)^{5/2}\gamma \cdot \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j} = n^{5/2} \cdot \left(\frac{1}{4}\right)^n$$

By analyzing its derivative function, we know the maximum value is $NFA(4) = 0.125$. It is smaller than $\varepsilon = 1$. Therefore, we conclude that the detected line segment regions always meet the NFA and aligned point density requirement in the original LSD.

In general, we process the edge map in single-pixel width by a strict region growing method requiring the aligned points' density to be one, which ensures a valid line detection performance with low error rate.

B. Computational Complexity Analysis

In our proposed implementation, we exploit the massively parallel architecture of the hardware by operating all the pixels in the same convolutional manner within each module. For the Gaussian blur module, each pixel goes through a same number of multiplication and adding operations to obtain the smooth grayscale value. Then, gradient computation module calculates the magnitude and pseudo-orders the angle of every pixel in linear time. NMS and double threshold connection modules operate every pixel's gradient identically to determine an edge map. Finally, the image mirror and region growing modules also process every point in the same mechanism.

In summary, processing every pixel in an identical manner makes our algorithm at a computational complexity of $O(N)$ where N is the total number of pixels in the image.

The original LSD algorithm implementation [14] also has an execution time proportional to N. However, it requires serial float-point processing modules of rectangle approximation and NFA computation, which also might involves recursive operation as some failing line segment region needs re-processing with automatically changed parameters. Such modules are eliminated in our system, making our proposed method much less computational complex. Computation time is thus reduced greatly on hardware with much slower clock rate. Detailed analysis of comparison is presented on Section V.

V. Test and Assessment

In this section, the experimental results and assessment are presented in two aspects. First, both noise-free drawn images and noisy life image are experimented to verify the reliability

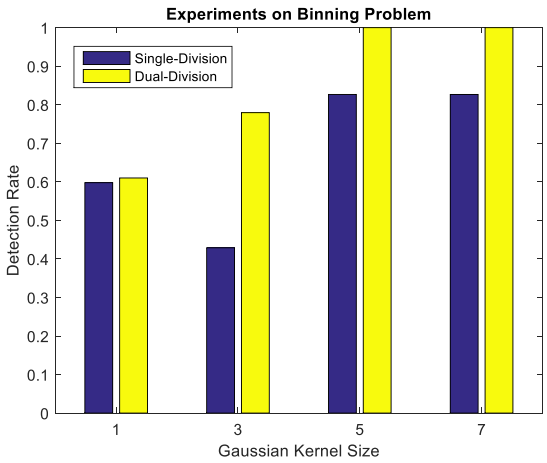

Fig. 12. Bar chart of detection rate on experiments of binning problem.

and accuracy of our modified LSD algorithm Second, the consumption of hardware resources is evaluated, especially on the massive use of BRAMs as row buffers. The overall time latency analysis showing real-time processing ability is given in details.

A. Reliability and Accuracy

1) Noise-Free Drawn Lines Detection Test: First, to test the performance of our method on the binning problem, we have drawn 1800 lines respectively with angles in $[0, 180°]$ with an interval of 0.1° to test our algorithm's line detection rate. Four Gaussian blur kernels are tried while each of them is tested with both single-division and dual-division gradient LLA pseudo-ordering method mentioned in Section III. The detection rate is calculated by the number of detected lines divided by 1800.

The line detection rate results are shown in Fig. 12. The dual-division method enhances the detection rate greatly as it makes up for those lines with angles near the border of the LLA bins in the single-division situation. The results of the dual-division method also show that Gaussian blurred images with a smooth edge obtain better detection. With the dual-division method and a Gaussian kernel size of five, the detection rate reaches 100%. So in the implementation, we choose the Gaussian kernel size as five and use a dual-division method to obtain a good detection rate.

To test the reliability and accuracy of our modified LSD algorithm implemented on hardware, we compare our result with that of the software implementation of the original LSD. A total of 16 lines are drawn in an image with line angles spread in an average of 360° with an interval of 22.5°. To observe the line detection result, the detected parameters are transferred to PC via USB port and then the detected lines are drawn on an empty image with the same size as the input image. The comparison results are shown in Fig. 13.

Our adjusted hardware Canny detector provides a good result of the edge map with all the line segments well connected and accurately located. Given that our canny implementation performs a strict non-maximum suppression, Fig. 13(e) shows that only the most significant edge points along the edge direction are preserved. Although our hardware Canny detection results contain minor noisy points, those isolated points are prevented from having an aligned LLA with its neighbouring points by the NMS module. Therefore the noisy outputs will not affect our line detection.

Software LSD is set without any parameter tuning so that it is applicable for general purpose to have a good line detection. However, a fixed parameter can result in many redundant line detections on non-linear edges in complex images. For example, minimum region size is the threshold of number of points in a line segment region. It is calculated according to the image's size in the original LSD. This results in a small threshold taking great number of insignificant lines into the result. As shown in Fig. 13(c), arcs in the images will be counted as lines and are not removable. In our hardware implementation, we could set the minimum region size to control the results with reasonable line detections as shown in Fig. 13(f).

To test further the accuracy of our modified LSD, the detected parameters are compared with the software results. The error rates are defined as equation (11) (12). The error rates along the x and y axes and in general are calculated.

$$error = \left| \frac{value_{hw} - value_{sw}}{value_{sw}} \right| \tag{11}$$

$$error_{general} = \sqrt{\frac{(x_{hw} - x_{sw})^2 + (y_{hw} - y_{sw})^2}{x_{sw}^2 + y_{sw}^2}} \tag{12}$$

The test errors of the drawn lines' parameters are shown in Fig. 14. From the statistics we can see that the relative error is always less than 2.5%. The typical value of a general error rate is around 1%, which shows a high accuracy of hardware implementation. With regard to all angles of lines, the error is typically small and acceptable.

2) Noisy Road Lines Detection Test: On the system of Lane Departure Warning System, it is crucial to detect the lanes fast and accurately at first. Our fast and resource-efficient hardware implementation of LSD algorithm can be applied in these systems because lanes are long and straight in most cases. In addition, our implementation uses very little energy to detect line segments. Hence it is ideal to embed our design on the vehicle. We take the image of the road scene from the car to test our algorithm shown in Fig. 15.

The same parameters are adopted in our hardware Gaussian filter and canny detector as those of the software. The software result has a smooth edge map while it fails to get rid of many meaningless edge points due to its exhausted iterative method of double threshold connection. Our adjusted hardware implementation only scans the image once and preserves those strongest edges. It is suitable for later region growing algorithm because long and straight lanes remained significant in the edge map while their angle map is also well aligned. At the same time, the pixels on the background are obviously not aligned.

The software LSD's minimum region size threshold takes many insignificant lines into result shown in Fig. 15(e). This redundant result can be hazardous for successive processing in practical applications. Our algorithm has a flexible minimum

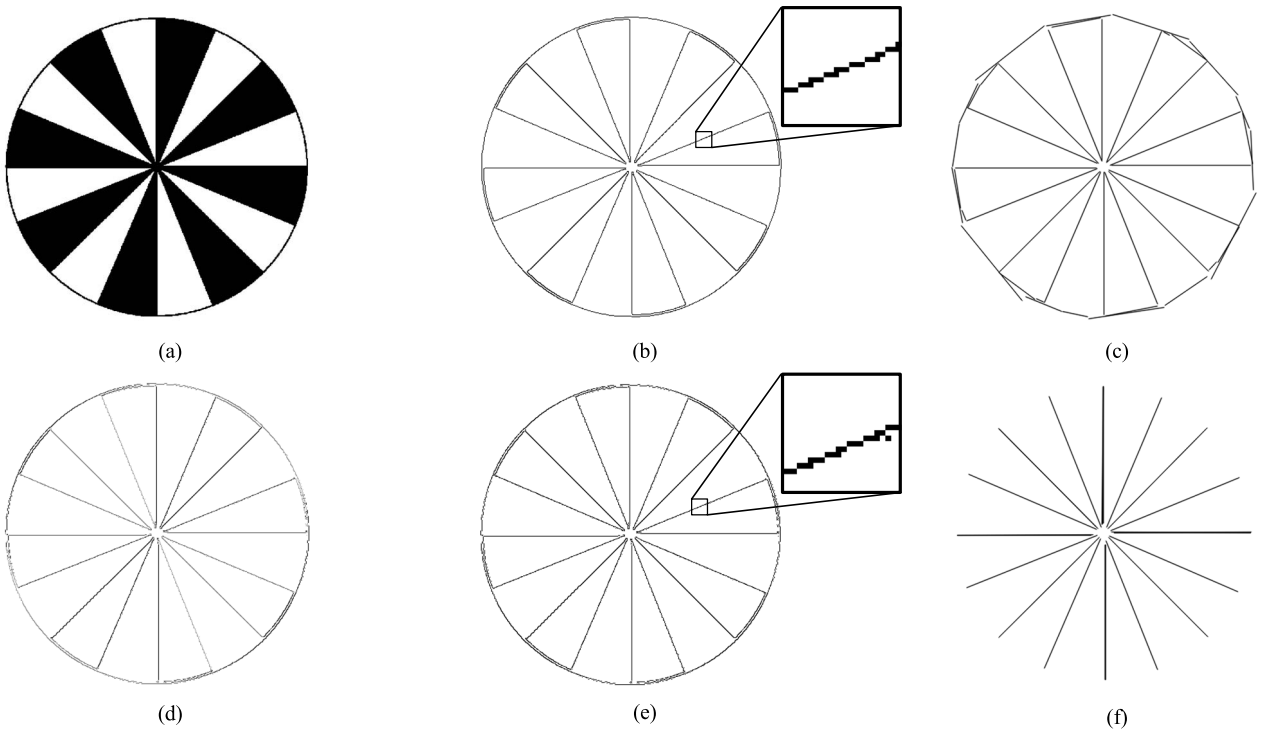

Fig. 13. Comparison results on image with drawn lines from 16 different angles. (a) Original image with 16 lines. (b) Software Canny detection. (c) Software LSD detection result. (d) Hardware LLA gradient angles map. (e) Hardware Canny detection edge map. (f) Hardware LSD line detection result.

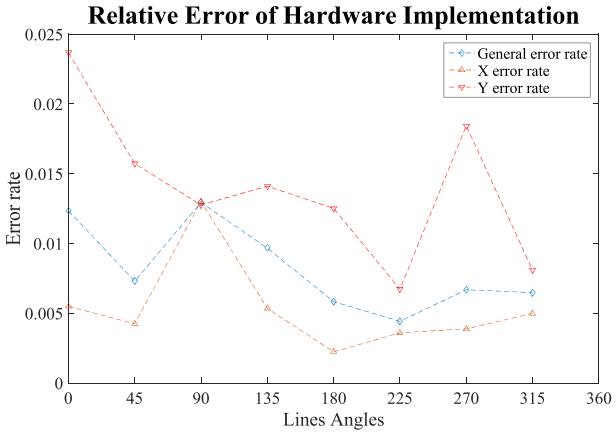

Fig. 14. Relative error rate diagram.

region size threshold set by users to help dedicated hardware provide better line detection results in specific applications. For example, lane detection results with higher threshold are shown in Fig. 15(d).

The results show that all straight lanes are detected correctly. For a binocular vision system, we can use camera intrinsics and extrinsic matrix to further reconstruct the road from the lane detection results. Thus, the position of the camera can be located to identify correctly if the vehicle departs from the lanes.

We also conducted experiments on the images from the original LSD paper using the same minimum region size threshold as software LSD. The results are shown in Fig. 16. Both noise-free and noisy images are performed with similar line detection

TABLE I
FPGA IMPLEMENTATION RESOURCE CONSUMPTION

Resource Categories	Used	Available	Utilization
Flip-flops	4810	106400	4.52%
Slice LUTs	4437	53200	8.34%
Memory LUTs	378	17400	2.17%
DSP blocks	8[1]	220	3.64 %

[1] DSP blocks are used to do multiplications more accurately and efficiently.

results while our proposed algorithm provides with less line detection as we analysed in Section IV. Furthermore, in the image of chairs, the original LSD detects some words on the labels on the back of the chair as lines, which are errors of line detection. In the result of our implementation, those words are not detected as lines, which indicates a lower error rate compared with the original LSD algorithm due to our stricter region growing mechanism.

In general, under the same setting of minimum region size, our method could further reduce the error detection rate while preserve most of the significant lines.

B. Resource and Time Consumption Analysis

The architecture was successfully synthesized, placed and routed with the designing tool Vivado 15.2 software, Xilinx. Table I shows the important hardware resources usages on platform XC7Z020 FPGA.

1) Resources Consumption Analysis: The memory usage is described in Table II. The BRAMs serve as FIFOs to buffer the intermediate values of rows in the convolutional processing.

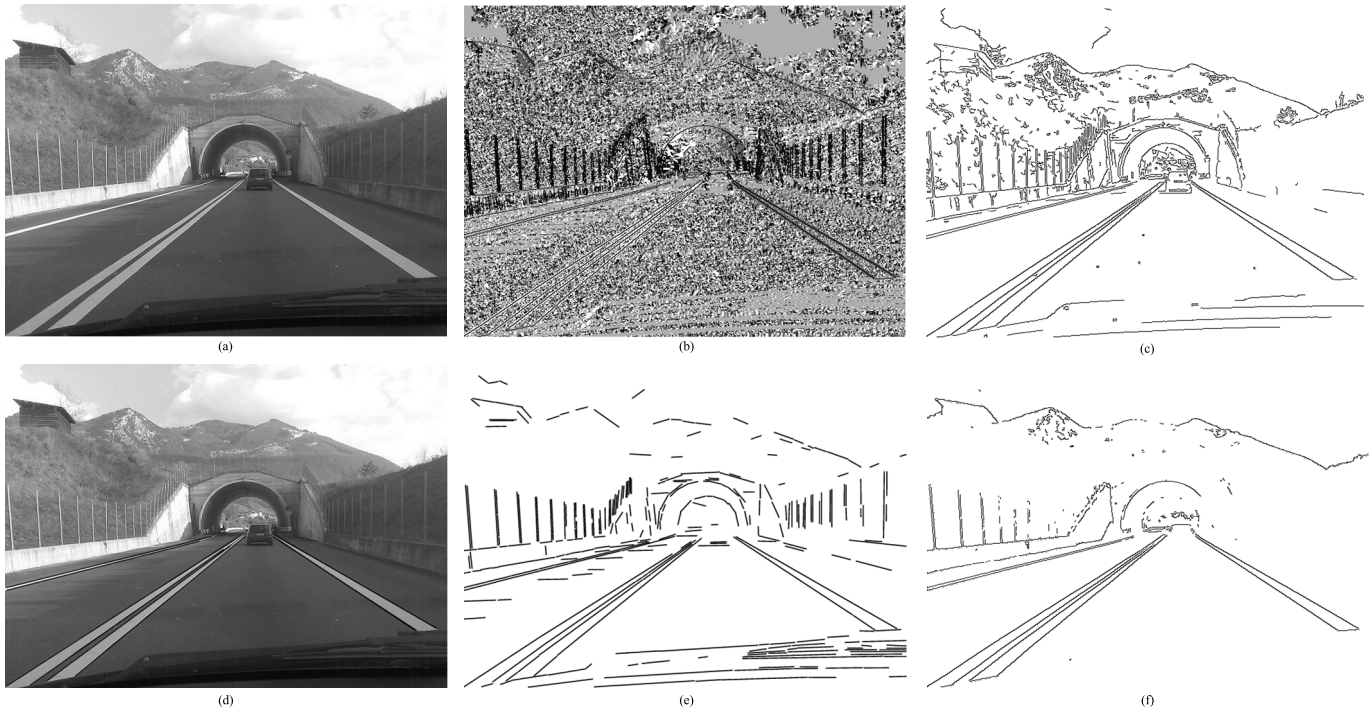

Fig. 15. Comparison results on life image with lanes from different angles. (a) Original image with lanes. (b) Hardware Level Line Angles map of the whole image. (c) Software canny edge detection. (d) Hardware detected lines on original image. (e) Software implemented LSD result. (f) Hardware canny edge map result.

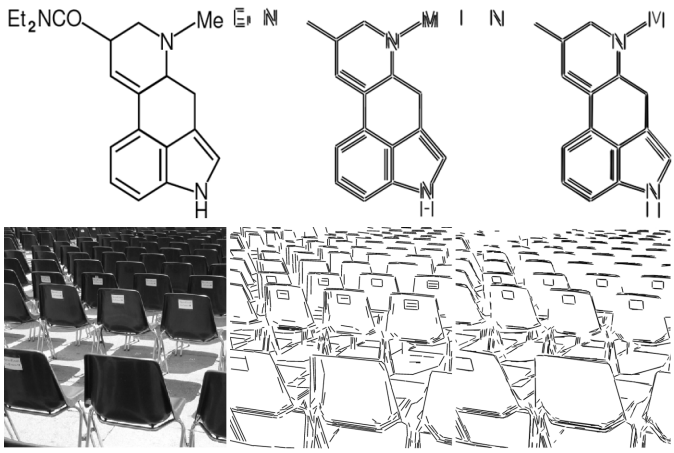

Fig. 16. Comparison results on test images from the original LSD paper. Columns from left to right are original images, LSD software results, and the proposed modified LSD results.

The size of the convolutional kernel and the data width in each module determine the number of utilized BRAMs. In total, less than 15% of the FPGA's on-chip BRAM is used, with the remaining memory having enough space for an image buffer for further processing.

In general, the streaming process of convolutions requires only row buffers, making our method resource-efficient.

2) Time and Power Consumption Analysis: In this paper, FPGA runs at a clock of 100MHz generated from digital clock managers inside the chip provided by Xilinx for the whole system. With the task-level fully pipelined architecture and data-level parallelism inside each module, time consumption is determined by the longest sub-module. For the whole system, time consumption is the sum of the main pipeline modules. In our modified LSD based line detection, time consumption is directly proportional to the size of the input image regardless of the exact number of lines. According to the time consumption analysis in Fig. 17 we can calculate the Initial Latency (IL) as well as total Processing Latency (PL) of hardware LSD by equation (13) (14). The IL is the time delay between a pixel entering the system and coming out after all the processing finished. The PL is the latency from the first pixel that enters the hardware until the last pixel finishes processing. It indicates the longest period of the whole algorithm. We use W as the image's width and H to denote height.

TABLE II
MEMORY USAGE

Buffer depth	1024	
Memory Usage	Gaussian filtering	90Kb
	Gradient computation & pseudo-ordering	54Kb
	NMS	54Kb
	Double Threshold	54Kb
	Mirror	18Kb
	Region growing	360Kb
Total memory usage	630Kb/5040Kb	

$$IL = 21 + 7^{*}W \tag{13}$$

$$PL = W^{*}H + (21 + 7^{*}W) \tag{14}$$

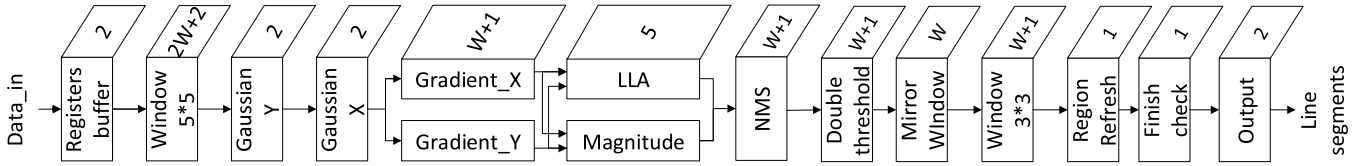

Fig. 17. Overall time latency analysis.

TABLE III
FPGA IMPLEMENTATION POWER CONSUMPTION*

Resource (640×480)	CPU Time(ms)	CPU Power(mJ)	FPGA Time(ms)	FPGA Power(mJ)
Drawn lines	103.5	4657	3.12	1.43
Road image	109	4905	3.12	1.43

*CPU runs at a clock rate of 3.5 GHz and FPGA's clock rate is 100MHz.

In the implementation of our modified LSD, we take images in the size of 640×480 for example. The PL is 3.12ms satisfying the real-time video processing speed requirement. The IL is only $45\mu s$, meaning that the line segment detection finishes almost the same time after the image scanned only once. Considering the power consumption, we have FPGA running at dynamic 0.263W and static 0.194W power consumption. In total its power consumption is 0.457W, approximately 1% of that of the CPU. Software implementation is on a 45W CPU to compare the time and power consumption in Table III.

From Table III, we can summarize as follows. On account of the parallel architecture of FPGA and the simplification of the original LSD algorithm, the processing time is shortened significantly, from the approximately 100ms on PC to the constant 3.12ms on FPGA regardless of the image's texture. This speed is capable of processing a video stream of a VGA resolution at a frame rate faster than 100 per second.

VI. CONCLUSION

The LSD algorithm gains a great reputation in image line extraction due to its good detection rate and low error rate. It has been implemented in the OpenCV library and is used widely dealing with lines. In this paper, a modified fast and resource-efficient hardware LSD algorithm on FPGA with task-level fully pipeline architecture is first implemented for real-time video processing systems. First, we deploy the Gaussian blur to smoothen the image to reduce noise. Second, an adjusted strict Canny edge detector is used to obtain the edge map with single pixel width. Then we apply the mirror module to detect lines of all directions. Finally, a dynamic rooted tree structure is first employed to do line segment region grow. The combination with canny edge detector enables FPGA to detect line segments in sequence with only one scan of the image free of any frame buffer. Furthermore, it uses a small number of row FIFOs to store intermediate values, saving much on-chip resources and time. Our modified LSD performs effectively with good reliability, high accuracy, low cost on time, power and on-chip resources. These attributes make our proposed algorithm suitable for real-time video processing systems dealing with line segments like the LDWS, binocular vision systems and many others.

The authors would also like to thank Key laboratory of Precision Opto-mechatronics Technology for providing experiment equipment for our work. Fuqiang Zhou and Yu Cao contributed equally to this paper.

REFERENCES

[1] C. Akinlar and C. Topal, "EDLines: A real-time line segment detector with a false detection control," *Pattern Recognit. Lett.*, vol. 32, no. 13, pp. 1633–1642, 2011.
[2] D. Kim, S. H. Jin, N. T. Thuy, K. H. Kim, and J. W. Jeon, "A real-time finite line detection system based on FPGA," in *Proc. 6th IEEE Int. Conf. Ind. Inform.*, Jul. 2008, pp. 655–660.
[3] S. Mahadevan and D. P. Casasent, "Detection of triple junction parameters in microscope images," *Proc. SPIE*, vol. 4387, pp. 204–214, Mar. 2001.
[4] Q. Yang, "Hardware-efficient bilateral filtering for stereo matching," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 5, pp. 1026–1032, May 2014.
[5] Y. Zheng, H. Li, and D. Doermann, "A parallel-line detection algorithm based on HMM decoding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 5, pp. 777–792, May 2005.
[6] V. Gaikwad and S. Lokhande, "Lane departure identification for advanced driver assistance," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 2, pp. 910–918, Apr. 2015.
[7] T. Santti, O. Lahdenoja, A. Paasio, M. Laiho, and J. Poikonen, "Line detection on FPGA with parallel sensor-level segmentation," in *Proc. 14th Int. Workshop Cellular Nanosc. Netw. Appl. (CNNA)*, 2014, pp. 1–2.
[8] 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*, vol. 13, no. 7, pp. 9223–9247, 2013.
[9] X. Zhou, Y. Ito, and K. Nakano, "An FPGA implementation of Hough transform using DSP blocks and block RAMs," *Bull. Netw., Comput., Syst., Softw.*, vol. 2, no. 1, pp. 18–24, 2013.
[10] P. Kahn, L. Kitchen, and E. M. Riseman, "A fast line finder for vision-guided robot navigation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 12, no. 11, pp. 1098–1102, Nov. 1990.
[11] C. Galambos, J. Kittler, and J. Matas, "Gradient based progressive probabilistic Hough transform," *IEE Proc.-Vis., Image Signal Process.*, vol. 148, no. 3, pp. 158–165, Jun. 2001.
[12] J. Matas, C. Galambos, and J. Kittler, "Robust detection of lines using the progressive probabilistic Hough transform," *Comput. Vis. Image Understand.*, vol. 78, no. 1, pp. 119–137, 2000.
[13] R. G. von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall, "LSD: A fast line segment detector with a false detection control," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 4, pp. 722–732, Apr. 2008.
[14] R. G. von Gioi, J. Jakubowicz, J. M. Morel, and G. Randall, "LSD: A line segment detector," *Image Process. Line*, vol. 2, pp. 35–55, Mar. 2012.
[15] Q. Xu, S. Varadarajan, C. Chakrabarti, and L. J. Karam, "A distributed canny edge detector: Algorithm and FPGA implementation," *IEEE Trans. Image Process.*, vol. 23, no. 7, pp. 2944–2960, Jul. 2014.
[16] P. R. Possa, S. A. Mahmoudi, N. Harb, C. Valderrama, and P. Manneback, "A multi-resolution FPGA-based architecture for real-time edge and corner detection," *IEEE Trans. Comput.*, vol. 63, no. 10, pp. 2376–2388, Oct. 2014.
[17] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. Hoboken, NJ, USA: Wiley, 2011, pp. 239–251.
[18] R. Rodriguez-Gomez, E. J. Fernandez-Sanchez, J. Diaz, and E. Ros, "FPGA implementation for real-time background subtraction based on horprasert model," *Sensors*, vol. 12, pp. 585–611, 2012.

[19] S. Jin, D. Kim, T. T. Nguyen, D. Kim, M. Kim, and J. W. Jeon, "Design and implementation of a pipelined datapath for high-speed face detection using FPGA," *IEEE Trans. Ind. Informat.*, vol. 8, no. 1, pp. 158–167, Feb. 2012.
[20] W. Wang, J. Yan, N. Xu, Y. Wang, and F.-H. Hsu, "Real-time high-quality stereo vision system in FPGA," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 10, pp. 1696–1708, Oct. 2015.
[21] N. Isakova, S. Basak, and A. C. Sonmez, "FPGA design and implementation of a real-time stereo vision system," in *Proc. Int. Symp. Innov. Intell. Syst. Appl. (INISTA)*, 2012, pp. 1–5.
[22] J. Jiang, X. Li, and G. Zhang, "SIFT hardware implementation for real-time image feature extraction," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 7, pp. 1209–1220, Jul. 2014.
[23] H. Luo, X. Gai, Z. Chang, and B. Hui, "A real-time multi-scale 2D Gaussian filter based on FPGA," in *Proc. Int. Symp. Optoelectron. Technol. Appl.*, 2014, pp. 930104-1–930104-7.
[24] L. Daoud, D. Zydek, and H. Selvaraj, "A survey on design and implementation of floating point adder in FPGA," in *Progress in Systems Engineering*. Cham, Switzerland: Springer, 2015, pp. 885–892. [Online]. Available: https://doi.org/10.1007/978-3-319-08422-0_129, doi: 10.1007/978-3-319-08422-0_129.
[25] P.-Y. Hsiao, C.-H. Chen, S.-S. Chou, L.-T. Li, and S.-J. Chen, "A parameterizable digital-approximated 2D Gaussian smoothing filter for edge detection in noisy image," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2006, p. 4.
[26] L.-C. Chiu, T.-S. Chang, J.-Y. Chen, and N. Y.-C. Chang, "Fast SIFT design for real-time visual feature extraction," *IEEE Trans. Image Process.*, vol. 22, no. 8, pp. 3158–3167, Aug. 2013.
[27] J. Zhu and T. Maruyama, "Real-time detection of line segments on FPGA," in *Proc. Int. Conf. Field-Programm. Technol. (FPT)*, Beijing, China, 2010, pp. 192–199.

Fuqiang Zhou received the B.S., M.S., and Ph.D. degrees in instrument, measurement and test technology from Tianjin University, China, in 1994, 1997, and 2000, respectively. He joined the School of Automation Science and Electrical Engineering at Beihang University as a Post-Doctoral Research Fellow in 2000. He is currently a Professor with the School of Instrumentation Science and Opto-electronics Engineering, Beihang University, China. His research interests include computer vision, image processing, and optical metrology.

Yu Cao received the B.S. degree from Beihang University, Beijing, China, in 2015, where he is currently pursuing the M.S. degree with the School of Instrumentation Science and Opto-electronics Engineering.

His research interests include computer vision and pattern recognition.

Xinming Wang received the B.S. degree from the China University of Petroleum, Shandong, China, in 2013, and the M.S. degree from the School of Instrumentation Science and Opto-electronics Engineering, Beihang University. Her research interests include machine vision and measurement.