WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

A Multiplierless FPGA Edge-Detection Pipeline with Run-Time Selectable Gradient Operators and Max-Plus-Half-Min Magnitude

Arihant Kamdar[a,*], Nainitya Bombewar[a], Riya Nikam[a], Chirag Naitam[a], A. P. Rathkanthiwar[a]

[a]Department of Electronics and Communication Engineering, Priyadarshini College of Engineering, Nagpur 440019, Maharashtra, India

A B S T R A C T :

Edge detectors on field-programmable gate arrays (FPGAs) usually compute the image gradient with general multipliers, which occupy digital signal processing (DSP) slices, or with a fixed circuit for a single operator. We present a multiplierless streaming edge-detection pipeline for the Xilinx Zynq-7000. One gradient engine computes the Sobel, Prewitt or Scharr operator, chosen at run time by a register. The three operators share their difference terms, and their weights are built from shifts and additions. The gradient magnitude is approximated as the larger component plus half the smaller, and an Otsu threshold is computed from a histogram collected in hardware. After place and route, the pipeline uses no DSP slices where a multiplier-based design with identical outputs needs 20, at a cost of 137 look-up tables (LUTs). The shared engine needs 80% fewer LUTs than one built from LUT-based multipliers, and 56% fewer than three separate engines. The magnitude approximation cuts the orientation dependence of the edge response from 45–51% to at most 15.5%. The pipeline processes one pixel per clock cycle, and its outputs matched a software reference exactly in 58 simulation runs. After smoothing, the choice of operator changes the edge map very little.

Keywords: Edge detection; Field-programmable gate array; Multiplierless arithmetic; Gradient magnitude approximation; Otsu thresholding

Introduction

Edge detection reduces an image to the places where intensity changes sharply, and later stages use these for segmentation, measurement and recognition [1]. Gradient operators such as Sobel [2], Prewitt [3] and Scharr [4] estimate this change over a 3×3 neighbourhood. Canny's detector [5] wraps the same gradient in smoothing, thinning and hysteresis. Every output pixel needs the same small computation, so these operators suit streaming FPGA hardware that takes one pixel per clock, and many such designs have been published [6–9].

Two choices decide what such a design costs. The first is how the kernel products are computed. The coefficients are small integers, so shifts and additions can replace the multipliers [10]. Many designs still use general multipliers, which map to DSP slices, because they accept any coefficient. The second is how the gradient magnitude is formed. The usual cheap form, $|G_x| + |G_y|$, makes the response depend on the direction of the edge. The word reconfigurable also needs care. Some papers use it for the FPGA platform itself [11]. Here it means that one fixed datapath computes any of three operators, picked by a register while the design is running.

This paper describes such a pipeline on a Zynq-7000 XC7Z020 and evaluates it through simulation and implementation. The building blocks are known. What we add is (i) one multiplierless engine in which the three operators share their difference terms, (ii) a comparison with multiplier-based and separate-engine versions under the same device and timing constraints, and (iii) a measurement of how much the operator and the magnitude approximation actually change the output. All results come from simulation and implementation tools; the design has not yet been measured on the board.

Related work

Hsiao *et al.* [6] combined approximate Gaussian smoothing with a pipelined gradient stage. Singh *et al.* [7] computed all directions of the Sobel compass operator with one processing element and reduced resources by more than 40%. Guo and Wu [8] built a simplified Canny detector with an Otsu threshold, and Ravichandran *et al.* [9] evaluated the two Sobel kernels in parallel. Santos *et al.* [12] implemented five edge operators as separate circuits and compared their cost and output quality.

In all of these, each operator has its own circuit. It is well known that the Sobel, Prewitt and Scharr kernels differ only in their smoothing weights [4], so the idea of sharing them is not new. Our contribution is to build them as one selectable datapath and to measure what that saves against the obvious alternatives. The magnitude estimate max + min/2 is likewise a standard signal-processing approximation [13]; we measure its cost and its effect inside the pipeline.

Proposed method

1.1. Smoothing

The input is first smoothed with a 5×5 binomial kernel built from [1 4 6 4 1], which approximates a Gaussian with σ = 1 pixel. Its weights are powers of two or sums of two powers of two, so the stage needs only shifts and additions.

1.2. Gradient

For a 3×3 window $p(r, c)$ of smoothed pixels, let $D_r = p(r, 0) - p(r, 2)$ and $E_c = p(0, c) - p(2, c)$. All three operators can then be written as

$$G_x = a(D_0 + D_2) + bD_1,\ G_y = a(E_0 + E_2) + bE_1 \qquad (1)$$

with $(a, b) = (1, 2)$ for Sobel, (1, 1) for Prewitt and (3, 10) for Scharr. The terms $D_0 + D_2$, D_1, $E_0 + E_2$ and E_1 are the same for all three operators; only the weights change (Fig. 1). Each weight is a shift, or a shift plus one addition. For example, $10D_1 = (D_1 \ll 3) + (D_1 \ll 1)$. A direct implementation with multipliers needs 18 products for every output pixel.

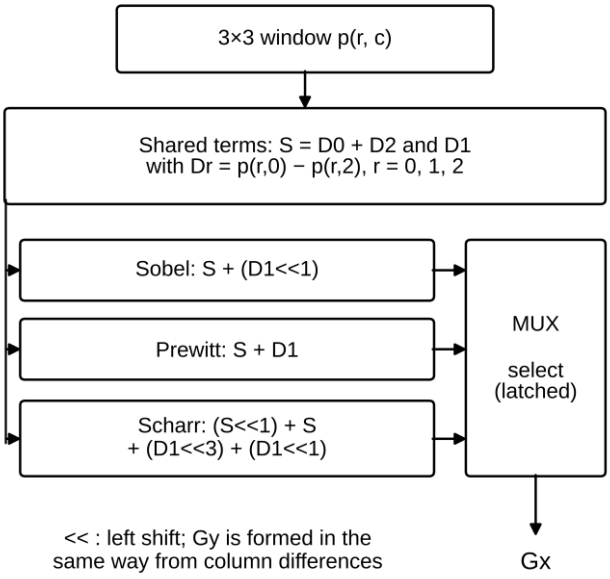


Fig. 1 – Shared gradient computation for G_x. The common terms are computed once, the operator weights are applied with shifts and additions, and a multiplexer selects the operator

1.3. Magnitude and threshold

The Euclidean magnitude needs two squares and a square root. The pipeline uses

$$M = max\left(|G_x|, |G_y|\right) + min\left(|G_x|, |G_y|\right)/2 \qquad (2)$$

which costs one comparison, one shift and one addition. For a gradient of unit length it never exceeds 1.118, while $|G_x| + |G_y|$ reaches 1.414 on diagonal edges. The magnitude is scaled to 8 bits, and a 256-bin histogram of it is collected for every frame. The processor computes Otsu's threshold [14] from this histogram, and a pixel is marked as an edge when its value is above the threshold.

Hardware architecture

Fig. 2 shows the pipeline. Every stage accepts one pixel per clock cycle. Line buffers in distributed RAM feed the smoothing stage and then a 3×3 window generator, and the gradient engine is pipelined over five stages. The operator select and the threshold are latched at the start of each frame, so a frame never mixes two operators. Each stage also restarts at a frame start, which means a lost pixel cannot disturb later frames. The histogram has two banks: the hardware fills one while the processor reads the other.

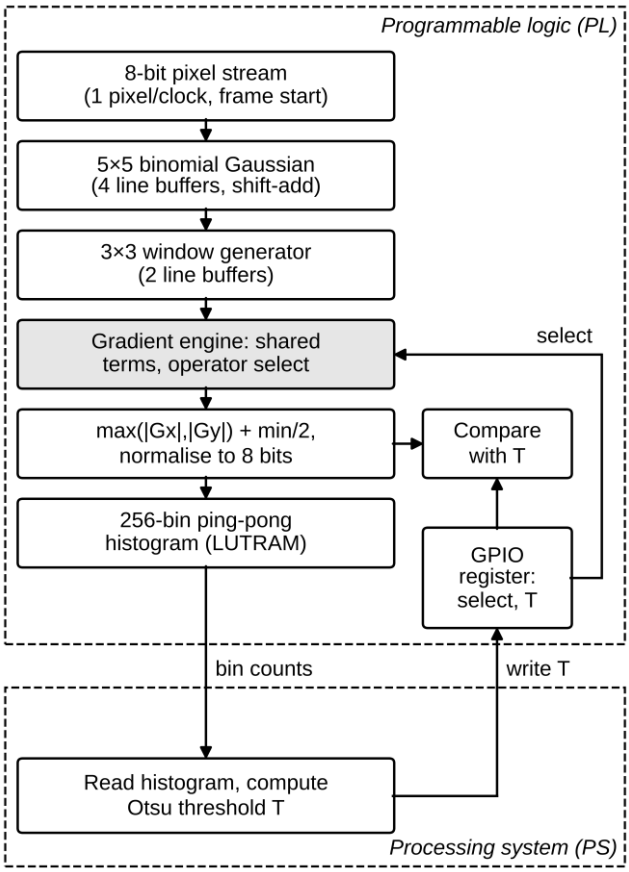


Fig. 2 – Pipeline architecture. The arithmetic runs in the programmable logic (PL); the processing system (PS) reads the histogram, computes the threshold T and sets the operator

For comparison we built a baseline engine that computes the same 18 products with general multipliers and reads its coefficients from a table. It produces identical outputs, and synthesis maps it to 20 DSP slices. For the PYNQ-Z2 board the pipeline is wrapped with an AXI4-Stream interface and an output buffer, which we also checked in simulation.

Experimental setup

All designs were implemented in Vivado 2024.1 for the XC7Z020 with a 200 MHz clock constraint and the same settings for every run. We compared the complete proposed pipeline with a baseline that differs only in its multiplier-based engine. We also implemented the engines on their own: the shared engine, the multiplier engine on DSP slices, the same engine with multipliers built from LUTs, and three single-operator engines.

The test inputs were six images from the BSDS500 data set [15] at 256×256, four road images [16] at 320×240 and five synthetic patterns. Every RTL simulation was compared, output by output, with a Python reference model, and the routed netlists were simulated as well. Power figures are Vivado estimates based on switching activity recorded in simulation.

Results and discussion

1.4. Verification and throughput

All 58 RTL runs matched the reference model exactly, across all images, frame sizes and operators, and also with random input stalls, lost pixels and an operator change in the middle of a frame. The routed netlists reproduced the RTL outputs. The multiplier-based and shift-add pipelines gave identical results, so the comparisons below are between designs that compute exactly the same thing. The pipeline produces one output per clock cycle; at 100 MHz a 256×256 frame takes 0.66 ms of processing time, not counting data transfer.

1.5. Resources and power

Table 1 – Complete pipelines after place and route

Metric	Proposed	Baseline
LUTs	1183	1046
Flip-flops	819	705
DSP slices	0	20
Estimated dynamic power (mW)	45–46	51–53

Power range covers the three operators.

Table 1 compares the complete pipelines. The proposed design uses no DSP slices instead of 20, which is 9% of the device, and pays for this with 137 extra LUTs, about a quarter of one percent of the device. Its estimated dynamic power is 45–46 mW against 51–53 mW. These are tool estimates and still need to be confirmed on the board. Both designs are within a few picoseconds of the 200 MHz constraint, so neither has a meaningful speed advantage.

Table 2 – Gradient engines alone after place and route

Engine	LUTs	Flip-flops	DSP
Shared shift-add (proposed)	193	184	0
18 multipliers on DSP slices	55	54	20
18 multipliers built from LUTs	951	438	0

Table 2 isolates the engines. When DSP slices are not to be used, the fair comparison is with multipliers built from LUTs, and there the shared engine needs 80% fewer LUTs and 58% fewer flip-flops. The multiplier engines accept any coefficients while ours is fixed to three operators, so part of this saving is the price of that flexibility. We also checked whether three separate single-operator engines would be a better choice. After synthesis they need 440 LUTs together, while the shared engine needs 195, only 18 more than the largest single engine (Scharr).

1.6. Edge maps

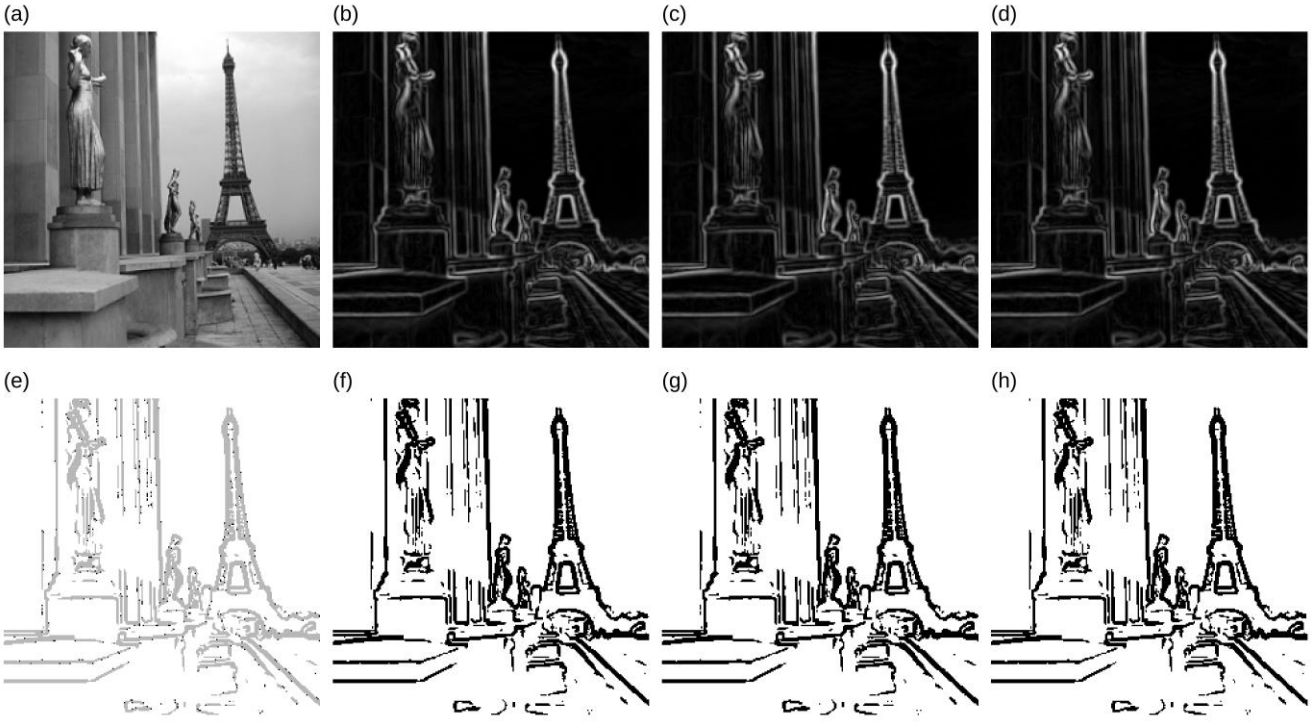


Fig. 3 – Image 223004 and pipeline outputs: (a) input; (b)–(d) normalised magnitude for Sobel, Prewitt and Scharr; (f)–(h) their edge maps; (e) union of the three edge maps (grey) and the 364 pixels where they disagree (black)

Fig. 3 shows the outputs for one image and Fig. 4 the Sobel edge maps for six natural and two road images. The three operators give almost the same edge maps. On the six natural images they disagree on 364 to 905 of 62,500 pixels, and the Jaccard index between any two of them is at least 0.932. Replacing (2) with $|G_x| + |G_y|$ changes the map more than any change of operator does (Jaccard index 0.873–0.920). In the road images the lane markings, road edges and vehicles are all kept.

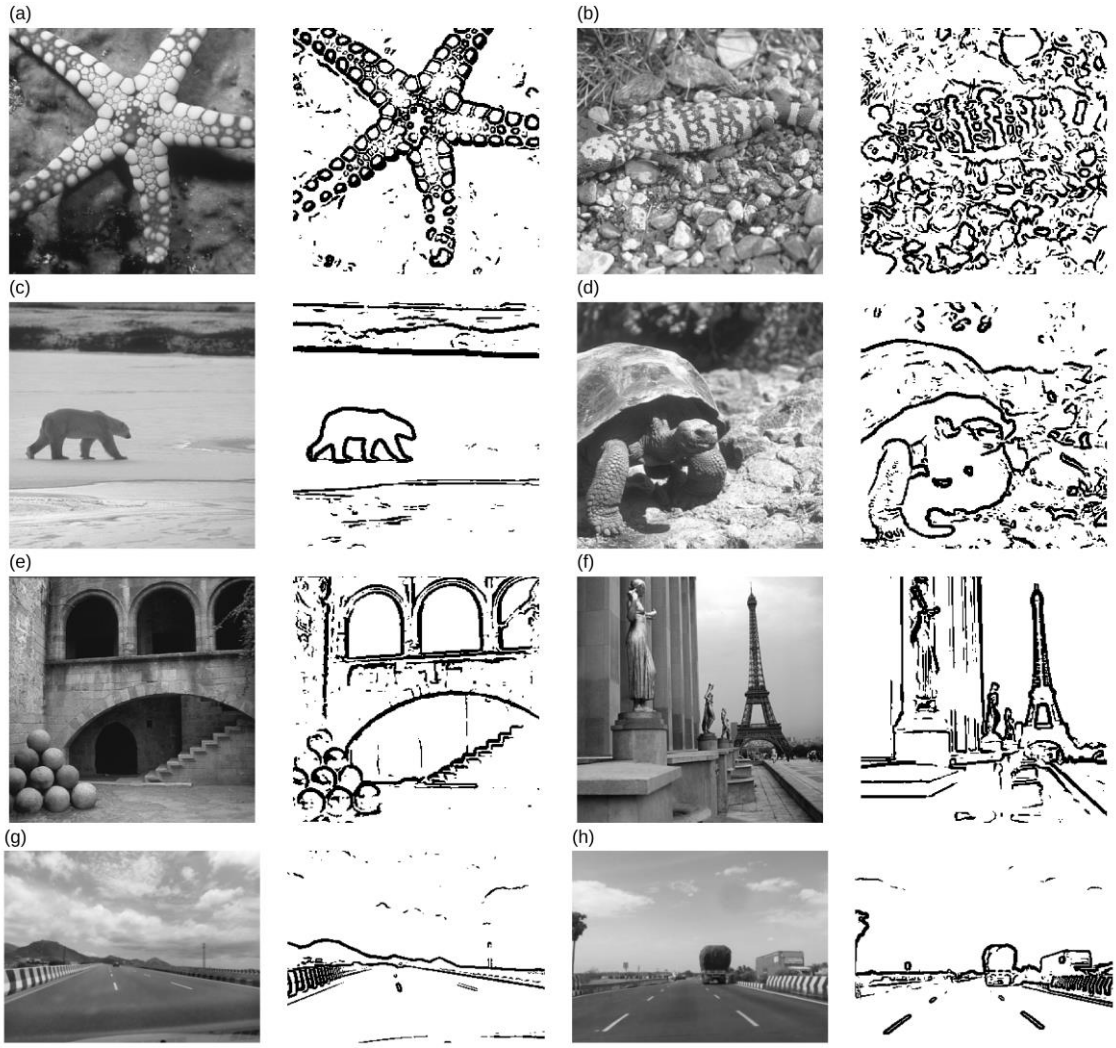


Fig. 4 – Inputs and Sobel edge maps from RTL simulation: (a)–(f) six BSDS500 images at 256×256; (g), (h) two road images [16] at 320×240

1.7. Magnitude and noise

Fig. 5(a) shows the response to a straight edge as it is rotated. With $|G_x| + |G_y|$ the response at 45° is 1.45 to 1.51 times the response at 0°. With (2) the worst deviation is 11.5–15.5%, and it costs 7 extra LUTs. Fig. 5(b) shows the F-measure on a synthetic scene with added Gaussian noise. With smoothing, all three operators stay between 0.950 and 0.968 up to $\sigma = 20$. Without smoothing they drop to about 0.3. In this pipeline, then, smoothing matters far more than the choice of operator. Both tests used the verified software model rather than hardware simulation, and the noise test uses a single synthetic scene, so the result should not be stretched too far.

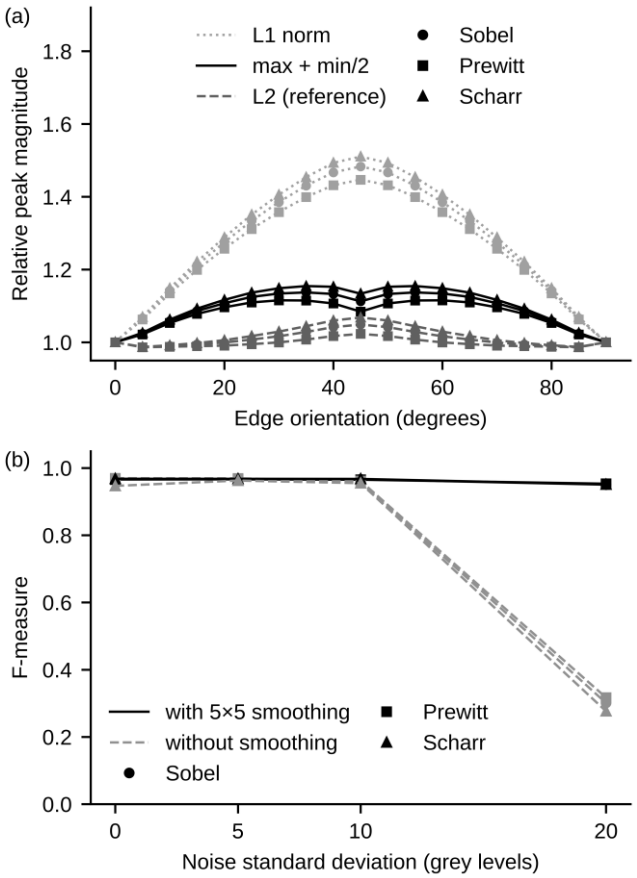


Fig. 5 – (a) Peak response to a step edge against orientation, relative to 0°, for $|G_x| + |G_y|$ (L1), the approximation (2) and the exact magnitude (L2); (b) F-measure against noise level with and without smoothing

1.8. Limitations

The shared engine supports only its three operators, and adding a fourth means changing the design. It trades DSP slices for LUTs, which helps only when the DSP slices are needed for something else. The power values are estimates. The pipeline does not thin edges or apply hysteresis, so edges are several pixels wide and strong texture is marked as edge. We did not measure accuracy against human-labelled boundaries, and nothing reported here has been measured on the board yet.

Conclusion

We built a multiplierless edge-detection pipeline for the Zynq-7000 that switches between the Sobel, Prewitt and Scharr operators at run time. Sharing the difference terms gives a 193-LUT gradient engine with no DSP slices, much smaller than multiplier-based or separate-engine alternatives, and the max + min/2 magnitude removes most of the direction bias of the usual $|G_x| + |G_y|$ form for 7 LUTs. The outputs matched a software reference exactly. The measurements also show that, once the image is smoothed, the operator matters little; the smoothing and the magnitude approximation matter more. The next step is to run the design on the PYNQ-Z2 board and measure throughput, latency and power there.

Acknowledgements

The authors thank the Department of Electronics and Communication Engineering, Priyadarshini College of Engineering, Nagpur, for its support.

Declarations

Funding: This work received no specific grant from any funding agency.
Competing interests: The authors declare no competing interests.
Data availability: The BSDS500 images [15] and the road images [16] are public. Other data are available from the corresponding author on reasonable request.
Author contributions: Arihant Kamdar: conceptualisation, methodology, hardware design, verification, formal analysis, writing – original draft. Nainitya Bombewar: data curation (dataset preparation), investigation (literature review and survey). Riya Nikam: data curation (dataset preparation), investigation (literature review and survey). Chirag Naitam: documentation. A. P. Rathkanthiwar: supervision, writing – review and editing.

Use of AI tools: The authors used Claude (Anthropic) to help draft and edit the text, write code and prepare figures. The design decisions, all simulation and implementation runs and the data are the authors' own. The authors checked every reported value against the tool reports and take full responsibility for the content.

REFERENCES

[1] Gonzalez, R. C., & Woods, R. E. (2018). *Digital image processing* (4th ed.). Pearson.

[2] Sobel, I., & Feldman, G. (1968). *A 3×3 isotropic gradient operator for image processing* [Presentation]. Stanford Artificial Intelligence Project, Stanford, CA, USA.

[3] Prewitt, J. M. S. (1970). Object enhancement and extraction. In B. S. Lipkin & A. Rosenfeld (Eds.), *Picture processing and psychopictorics* (pp. 75–149). Academic Press.

[4] Jähne, B., Scharr, H., & Körkel, S. (1999). Principles of filter design. In B. Jähne, H. Haußecker, & P. Geißler (Eds.), *Handbook of computer vision and applications* (Vol. 2, pp. 125–151). Academic Press.

[5] Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 8*(6), 679–698. https://doi.org/10.1109/TPAMI.1986.4767851

[6] Hsiao, P.-Y., Chen, C.-H., Wen, H., & Chen, S.-J. (2006). Real-time realisation of noise-immune gradient-based edge detector. *IEE Proceedings – Computers and Digital Techniques, 153*(4), 261–269. https://doi.org/10.1049/ip-cdt:20050199

[7] Singh, S., Saini, A. K., Saini, R., Mandal, A. S., Shekhar, C., & Vohra, A. (2013). Area optimized FPGA-based implementation of the Sobel compass edge detector. *ISRN Machine Vision, 2013*, Article 820216. https://doi.org/10.1155/2013/820216

[8] Guo, L., & Wu, S. (2023). FPGA implementation of a real-time edge detection system based on an improved Canny algorithm. *Applied Sciences, 13*(2), Article 870. https://doi.org/10.3390/app13020870

[9] Ravichandran, S., Su, H.-K., Kuo, W.-K., Dhanasekaran, D., Mahalingam, M., & Yang, J.-P. (2025). Parallel processing of Sobel edge detection on FPGA: Enhancing real-time image analysis. *Sensors, 25*(12), Article 3649. https://doi.org/10.3390/s25123649

[10] Dempster, A. G., & Macleod, M. D. (1995). Use of minimum-adder multiplier blocks in FIR digital filters. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, 42*(9), 569–577. https://doi.org/10.1109/82.466647

[11] Xu, L., & Zheng, D. (2022). A novel Sobel edge detection accelerator based on reconfigurable architecture. *Traitement du Signal, 39*(4), 1421–1427. https://doi.org/10.18280/ts.390436

[12] Santos, D., Zolett, D., Belli, M., Viel, F., & Zeferino, C. (2020). An analysis of the implementation of edge detection operators in FPGA. In *Proceedings of the X Brazilian Symposium on Computing Systems Engineering (SBESC), Extended Annals* (pp. 163–167). https://doi.org/10.5753/sbesc_estendido.2020.13107

[13] Lyons, R. G. (2011). *Understanding digital signal processing* (3rd ed.). Prentice Hall.

[14] Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics, 9*(1), 62–66. https://doi.org/10.1109/TSMC.1979.4310076

[15] Arbeláez, P., Maire, M., Fowlkes, C., & Malik, J. (2011). Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 33*(5), 898–916. https://doi.org/10.1109/TPAMI.2010.161

[16] avinashsivakumar20-design. (n.d.). *lane-detection-segmentation: Indian road lane images and segmentation models* [GitHub repository, MIT License]. Retrieved October 20, 2026, from https://github.com/avinashsivakumar20-design/lane-detection-segmentation