An Analysis of the Implementation of Edge Detection Operators in FPGA

Douglas A. Santos, Daniel Zolett, Mateus Belli, Felipe Viel, and Cesar A. Zeferino
Laboratory of Embedded and Distributed Systems – LEDS
University of Vale do Itajaí – UNIVALI
Itajaí – SC, Brazil
{douglasas, daniel.zolett, mateusbelli}@edu.univali.br, {viel, zeferino}@univali.br

***Abstract*— Computer vision systems have several stages, and one of the operators used in these systems is the edge detection filter. High-performance computing is required in many applications and stages of computer vision systems, and many designs use FPGA technology to improve performance and decrease power consumption. In this context, this work presents an analysis of five edge detection filters synthesized to FPGA, including Laplacian, Roberts, Prewitt, Sobel, and Canny. In the experiments, we compared the hardware implementations with software versions to identify the impact of fixed-point representation on the quality of the output images. We have also assessed metrics regarding performance, silicon costs, and energy consumption. The results obtained show that the Laplacian filter has the lowest costs, while the Canny operator provides the best output image at the price of much higher silicon costs and energy consumption.**

***Index Terms*—Image Processing, Computer Vision, Edge Detection, Hardware Accelerator, FPGA.**

I. Introduction

The evolution of computing systems made possible the development of new applications. Among the emerging applications, we can highlight those using computer vision systems such as unmanned underwater vehicles [1], 3D environment mapping [2], and stereo image processing for robots [3]. A growing area of research that has a high interest in fast image processing at low cost is that of autonomous electric vehicles, as evidenced in [4]. These applications often demand solutions that fulfill performance and energy consumption requirements, primarily when they are used in systems that rely on battery power. In this sense, FPGA (Field-Programmable Gate Array), combined with fixed-point arithmetic, have been employed as a promising approach to fulfill all these requirements, while providing design flexibility and meeting time-to-market constraints.

Computer vision systems apply edge detection algorithms to identify the boundaries of objects in an image in the early stages of processing [5]. Edge detection is an elementary operation in low-level image processing [6] and, therefore, the search for efficient implementations becomes recurrent. In view of this, several filters have been proposed for edge detection, and the implementation of them in hardware must find a trade-off to fulfill the application requirements (i.e., image quality, performance, power budget, and silicon costs).

Some examples of hardware-based implementation of edge detection filters are reported in the literature. In [7]–[10], the authors implemented Sobel, Roberts, Prewitt, LoG (Laplacian of Gaussian), and Canny filters on FPGA, and have only analyzed their silicon costs. Other studies [7] [10] also compared the quality of the images resulted from similar implementations in FPGA. Few works also evaluated the energy costs of edge detection filters, which is a crucial metric for embedded computer vision applications. For instance, the authors of [9] and [11] evaluated the energy consumption of hardware implementations of Canny filters, while the authors of [12] analyzed similar implementations of Sobel filter in FPGA. However, none of these works performed an analysis of different filters to compare their energy consumption.

In light of the context above, this work describes a study that analyzes the energy and silicon costs of hardware implementations of Laplacian, Roberts, Prewitt, Sobel, and Canny filters, also evaluating their performance and the quality of their output images. The results obtained allow identifying the relationship between the different design metrics and provide information for further implementations of computer vision systems. Therefore, the main contribution of this paper relies on the evaluation of FPGA-based implementations of edge detection operators in FPGA, applying a complete set of metrics usually employed in hardware designs.

The remainder of this paper is organized as follows. Section II describes the experimental setup used in this work, presenting the materials and methods employed, as well as the architecture of the edge detection filters implemented in FPGA. Following, Section III presents and discusses the experimental results and Section IV concludes with the final remarks.

II. Experimental Setup

A. Materials and methods

This study evaluates software and hardware implementations of Laplacian, Roberts, Prewitt, Sobel, and Canny edge detection operators. We first implemented each operator using Python with floating-point representation to serve as a baseline

This work was supported by CAPES – the Brazilian Federal Agency for Support and Evaluation of Graduate Education – Finance Code 001 and CNPq – the Brazilian National Council for Scientific and Technological Development – Processes 315287/2018-7 and 436982/2018-8.

to specify the hardware implementations. We also used the software implementations processing a 220×220 grayscale Lena image to help us choose the number of bits required to represent each pixel in the fixed-point arithmetic of the hardware implementations. Our goal was to find a trade-off between the costs and the quality of the output image. After that, we described each operator in VHDL using fixed-point arithmetic for synthesis in FPGA. It is worth noting that we applied the convolution kernels most used in literature [13]–[15] – i.e., a 2×2 kernel for Roberts and 3×3 kernels for the other filters.

For evaluation, we employed images from the Berkeley Segmentation Dataset and Benchmark (BSDS500) [16], which contains 500 481×321 RGB images (300 for training and 200 for testing). This data set is used in other works that implement edge detection algorithms [17]–[21]. To evaluate the quality of the image resulting from the processing performed by the hardware implementations, we employed ModelSim Altera Starter Edition to simulate the execution of each operator. We then used the complex wavelet structural similarity (CW-SSIM) index [22] to quantify the similarity between the processing of the images on software and hardware. According to [22], it is possible to categorize a pair of images as different if the CW-SSIM index is below 0.6, and highly similar if it is greater than 0.9.

To assess the silicon and energy costs, we utilized the set of tools available within the Intel® Quartus® Prime software suite. All the synthesis and power analyses were carried out for the Cyclone V SoC 5CSEMA6F31A7 device.

It is worth noting that a comparison between FPGA and GPU implementations is beyond this work scope. Evidence from [11] and [23] proves that edge detection algorithms have a lower energy cost in FPGA, although GPU results are better when considering quality metrics.

B. Hardware design

The main block used for all the implementations is the sliding window, which is responsible for temporarily storing the input pixels and outputting valid windows. The input pixels go to the delay line block, which moves the pixels among registers to generate the valid windows; a controller is necessary to define when the window is valid. Fig. 1 depicts the structure of the sliding window block. It comprises the delay line, registers to count the current pixel, and comparators to identify when the window is valid. The registers are incremented under the command of the controller. In Fig. 1, the lowercase labels are the input and output signals of the datapath, and the uppercase tags are constants defined at synthesis time and depend on the image size.

Fig. 2 presents the diagram block for the Gradient filters. The input image is convolved with the Gx and Gy masks separately. We have employed this architecture to all the Gradient filters by changing only the convolution kernel. As the Roberts filter does not consider the central pixel, we chose the upper left pixel to play this role in this case.

The Canny filter uses the gradient method. However, it differs from the Gradient filters because it is a multi-stage algorithm. As a result, the Canny filter has high computational complexity, taking significantly longer to process an image. The multiple stages attenuate image noise and provide a more accurate edge detection [13]. These additional features make the Canny filter perform better than the Gradient filters, especially when processing noisy images.

Fig. 3 depicts the diagram block of the Canny filter. The image initially enters the Gaussian filter for noise reduction. The Sobel filter calculates the magnitude and direction of the gradient, and Theta block finds the direction of the gradient. Next, the Suppress block sets the gradient direction by taking into account the outputs of Normalize and Theta blocks. The Threshold block identifies whether the magnitude of the pixel is more significant than its neighbors in the gradient direction. Based on previously defined thresholds, the output pixel is then marked as an edge or background. Finally, the Hysteresis block removes any weak edge. It is worth noting that Canny differentiates itself from the Gradient filters because it applies the Suppress, Threshold, and Hysteresis steps [15]. The output image produced by Canny is binarized, simplifying processing in further steps of computer vision systems and reducing the image size. As Canny has a noise-removal stage (the Gaussian filter), it is less sensitive to noise than the other filters analyzed in this work.

III. Experimental Results

Initially, we simulated the hardware processing of a 220×220 grayscale Lena image to identify the data word configuration (the sizes of the integer and fractional parts) to obtain a similarity (CW-SIMM index) greater than 0.8. The similarity index was obtained by comparing the images processed by software (using double-precision floating-point representation) with the images obtained after the simulation of the models described in VHDL (using a fixed-point representation of different sizes for each filter).

As Table I shows, all operators were configured with eight fractional digits. The number of integer digits varied according to the complexity of the operations performed and need for preventing overflow. The Gradient filters achieved a similarity between 0.990 and 0.999, while Canny obtained 0.876 with 22 integer bits (the highest similarity obtained) and 0.846 with 10 integer bits. These results are due to theta calculation, which is performed by the two-argument arctangent function between the results of the X and Y convolutions – i.e., $atan2(x, y)$. The implementation of this function was carried out using the COordinate Rotation DIgital Computer (CORDIC) technique [24]. CORDIC performs several iterations to calculate the angle, and our implementation executes twenty pipeline iterations. As a result, errors caused by convolutions along with theta calculation error cause a loss of accuracy in some situations. Although the Q22.8 configuration produced a higher similarity index, we chose to use the Q10.8 configuration to reduce silicon overhead and energy consumption.

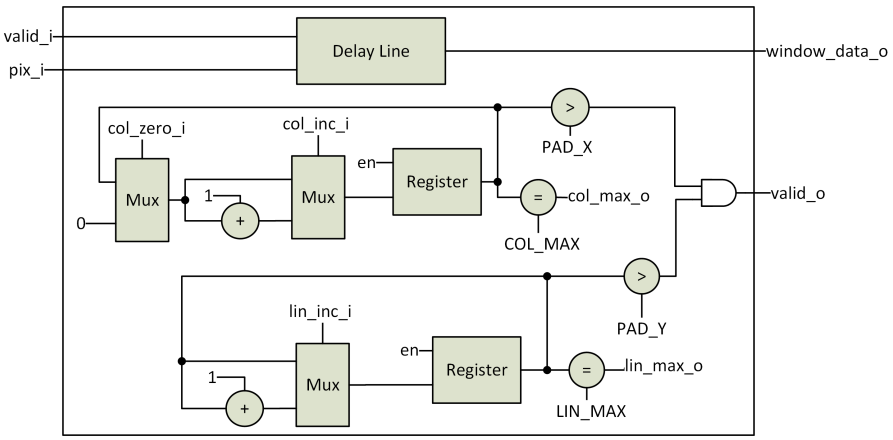


Fig. 1. Sliding Window block.

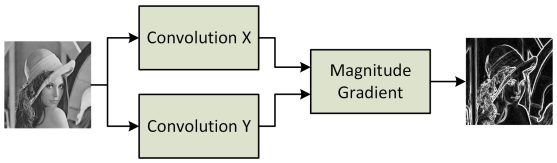


Fig. 2. Gradient filters architecture.

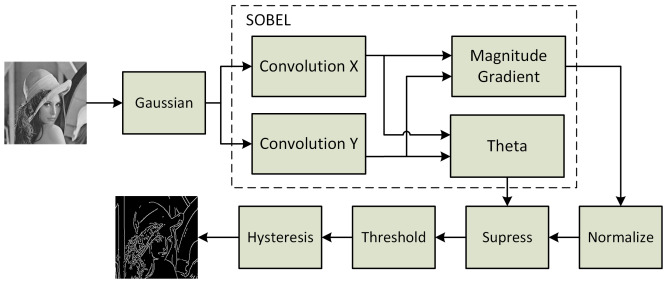


Fig. 3. Canny filter architecture.

TABLE I
SIMILARITY AMONG THE SOFTWARE-AND HARDWARE-BASED IMPLEMENTATIONS FOR THE PROCESSING OF THE LENA IMAGE

Q4.8 Laplacian	Q2.8 Roberts	Q2.8 Prewitt	Q4.8 Sobel	Q10.8 Canny	Q22.8 Canny
0.998	0.998	0.990	0.999	0.876	0.846

To evaluate the silicon costs, performance, and energy, we used the BSDS500 test images converted to grayscale. Tables II–IV present the results obtained. Silicon costs are expressed by the occupancy of resources of the FPGA. The performance metrics include the maximum operating frequency, the latency for processing a single 481×321 grayscale image, and the throughput. The power and energy consumed were obtained using the Intel® Quartus® Prime PowerPlay tool and the switching activity files obtained from the simulation of all operators operating at 100 MHz. These simulations comprised the execution of each operator when processing the images of the data set for 100ms. As expected, the Canny filter had the highest silicon and energy costs. On the other hand, all the Gradient filters had similar power consumption levels when operating at 100 MHz, with Roberts having the lowest silicon cost.

TABLE II
SILICON COSTS

Filter	ALUTs	FFs	Memory (bits)	DSPs
Q4.8 Laplacian	158	189	11,472	0
Q2.8 Roberts	219	426	478	1
Q2.8 Prewitt	326	547	954	1
Q4.8 Sobel	451	681	11,448	1
Q10.8 Canny	3,882	62,402	4,385,229	6

TABLE III
PERFORMANCE METRICS

Filter	Fmax (MHz)	Latency (cycles)	Latency (ms)	Throughput (frames/s)
Q4.8 Laplacian	130	154,401	1.188	845
Q2.8 Roberts	181	154,425	0.853	1,175
Q2.8 Prewitt	146	154,425	1.058	950
Q4.8 Sobel	104	154,425	1.485	679
Q10.8 Canny	107	305,663	2.857	352

TABLE IV
POWER AND ENERGY METRICS*

Filter	Static Power (mW)	Dynamic Power (mW)	Total Power (mW)	Energy per pixel (nJ)
Q4.8 Laplacian	411.33	5.92	434.82	4.35
Q2.8 Roberts	411.32	5.72	433.77	4.34
Q2.8 Prewitt	411.34	8.62	437.92	4.38
Q4.8 Sobel	411.38	11.21	441.18	4.41
Q10.8 Canny	420.25	600.85	1034.98	20.49

*Both operators operating at 100 MHz.

Finally, we calculated the average CW-SIMM for the BSDS500 images processed by each operator. Table V presents the results obtained. We can observe that the Gradient filters produced output images highly similar to the images produced by their software implementations. Canny, on the other hand, has obtained a lower similarity with its software implementation. We consider that obtaining a higher similarity index would require employing internal floating-point units, which would imply an even higher silicon and energy overheads.

TABLE V
AVERAGE SIMILARITY AMONG THE SOFTWARE-AND HARDWARE-BASED IMPLEMENTATIONS FOR THE PROCESSING OF BSDS500 DATASET

Q4.8 Laplacian	Q2.8 Roberts	Q2.8 Prewitt	Q4.8 Sobel	Q10.8 Canny
0.986	0.989	0.943	0.988	0.759

Fig. 4 and 5 show the results of the processing of some images on software and hardware. As we can see, Canny is the filter that better detects the edges of the images. Also, we can note that the hardware implementations can visually produce results that are visually similar to those obtained from their software implementations.

IV. CONCLUSIONS

This work evaluated the performance, costs, and quality of hardware-based edge detection filters. Results showed that the Gradient operators have lower costs than the Canny filter, but the latter generates the best output image, with sharper and thinner edges. As the silicon and energy costs of the Canny filter are much higher than those of the Gradient filters, we consider that it is worthwhile in cases where a better output is needed, and its higher energy consumption is acceptable.

The results obtained point out that the different operators could be applied in dynamically reconfigurable systems to enable the selection of the architecture that best fits the constraints. For instance, it is useful to tradeoff quality and energy to save battery in mobile computer vision systems.

As future work, we intend to use Otsu method [25] to define the threshold and improve the quality of the Canny filter. We also intend to extend this study by integrating the filters in a computer vision system to evaluate their impact on the system metrics.

REFERENCES

[1] A. Manzanilla, S. Reyes, M. Garcia, D. Mercado, and R. Lozano, "Autonomous navigation for unmanned underwater vehicles: Real-time experiments using computer vision," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1351–1356, Apr. 2019.

[2] S. Asaly, B. Ben-Moshe, and N. Shvalb, "Accurate 3D mapping algorithm for flexible antennas," *Int. J. of Antennas and Propagation*, vol. 2018, 2018.

[3] C. Ttofis, C. Kyrkou, and T. Theocharides, "A low-cost real-time embedded stereo vision system for accurate disparity estimation based on guided image filtering," *IEEE Trans. on Computers*, vol. 65, no. 9, pp. 2678–2693, Sep. 2016.

[4] M. Yih, J. M. Ota, J. D. Owens, and P. Muyan-Özçelik, "FPGA versus GPU for speed-limit-sign recognition," in *2018 21st Int. Conf. on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 843–850.

[5] P. Ganesan and G. Sajiv, "A comprehensive study of edge detection for image processing applications," in *2017 Int. Conf. on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Mar. 2017, pp. 1–6.

[6] M. Mittal, A. Verma, I. Kaur, B. Kaur, M. Sharma, L. M. Goyal, S. Roy, and T. Kim, "An efficient edge detection approach to provide better edge connectivity for image analysis," *IEEE Access*, vol. 7, pp. 33 240–33 255, 2019.

[7] G. B. Reddy and K. Anusudha, "Implementation of image edge detection on FPGA using XSG," in *2016 Int. Conf. on Circuit, Power and Computing Technologies (ICCPCT)*, Mar. 2016, pp. 1–5.

[8] G. N. Chaple, R. D. Daruwala, and M. S. Gofane, "Comparisions of Robert, Prewitt, Sobel operator based edge detection methods for real time uses on FPGA," in *2015 Int. Conf. on Technologies for Sustainable Development (ICTSD)*, Feb. 2015, pp. 1–4.

[9] J. Lee, H. Tang, and J. Park, "Energy efficient Canny edge detector for advanced mobile vision applications," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 28, no. 4, pp. 1037–1046, Apr. 2018.

[10] P. Selvakumar and S. Hariganesh, "The performance analysis of edge detection algorithms for image processing," in *2016 Int. Conf. on Computing Technologies and Intelligent Data Engineering (ICCTIDE)*, Jan. 2016, pp. 1–5.

[11] M. Qasaimeh, K. Denolf, J. Lo, K. A. Vissers, J. Zambreno, and P. H. Jones, "Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels," in *2019 Int. Conf. on Embedded Software and Systems (ICESS)*, 2019, pp. 1–8.

[12] C. Schlaak, M. Fakih, and R. Stemmer, "Power and execution time measurement methodology for SDF applications on FPGA-based MPSoCs," in *2019 Int. Wksp. on High Performance Energy Efficient Embedded Systems (HIP3ES)*, 2017, pp. 1–7.

[13] O. R. Vincent, O. Folorunso *et al.*, "A descriptive algorithm for Sobel image edge detection," in *Proc. of Information Science & IT Education Conference (InSITE)*, vol. 40. Information Science Institute California, 2009, pp. 97–107.

[14] S. S. Al-Amri, N. Kalyankar, and S. Khamitkar, "Image segmentation by using edge detection," *Int. J. on Computer Science and Engineering*, vol. 2, no. 3, pp. 804–807, 2010.

[15] L. Ding and A. Goshtasby, "On the Canny edge detector," *Pattern Recognition*, vol. 34, no. 3, pp. 721–725, 2001.

[16] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.2010.161

[17] J. He, S. Zhang, M. Yang, Y. Shan, and T. Huang, "Bi-directional cascade network for perceptual edge detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[18] A. Akbarinia, C. A. Parraga *et al.*, "Biologically-inspired edge detection through surround modulation," in *Proceedings of the British Machine Vision Conference*, 2016, pp. 1–13.

[19] S. Xie and Z. Tu, "Holistically-nested edge detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[20] P. Dollar and C. L. Zitnick, "Structured forests for fast edge detection," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[21] X. Hu, Y. Liu, K. Wang, and B. Ren, "Learning hybrid convolutional features for edge detection," *Neurocomputing*, vol. 313, pp. 377 – 385, 2018.

[22] M. P. Sampat, Z. Wang, S. Gupta, A. C. Bovik, and M. K. Markey, "Complex wavelet structural similarity: A new image similarity index," *IEEE Trans. on Image Processing*, vol. 18, no. 11, pp. 2385–2401, 2009.

[23] I. Sugiarto, G. Liu, S. Davidson, L. A. Plana, and S. B. Furber, "High performance computing on spinnaker neuromorphic platform: A case study for energy efficient image processing," in *2016 IEEE 35th Int. Performance Computing and Communications Conf. (IPCCC)*. IEEE, 2016, pp. 1–8.

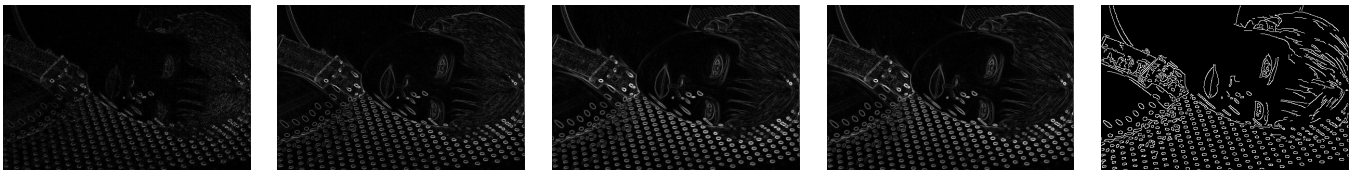
[24] J. E. Volder, "The cordic trigonometric computing technique," *IRE Trans. on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[25] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

(a) Software Laplacian (b) Software Roberts (c) Software Prewitt (d) Software Sobel (e) Software Canny

(f) Hardware Laplacian (g) Hardware Roberts (h) Hardware Prewitt (i) Hardware Sobel (j) Hardware Canny

Fig. 4. Output images from edge detection filters implemented on software and hardware processing of Lena image.



(a) Software Laplacian (b) Software Roberts (c) Software Prewitt (d) Software Sobel (e) Software Canny

(f) Hardware Laplacian (g) Hardware Roberts (h) Hardware Prewitt (i) Hardware Sobel (j) Hardware Canny

Fig. 5. Output images from edge detection filters implemented in software and in hardware processing an image of BSDS500.