ISSN 1840-4855
e-ISSN 2233-0046

Original scientific article
http://dx.doi.org/10.70102/afts.2025.1833.472

HIGH-SPEED HARDWARE EDGE DETECTION IMPLEMENTATION ON FPGA USING PIPELINED SOBEL AND CANNY ALGORITHMS

Ancy Joy[1], Jisha Jacob[2], Basil Roy[3]

[1]Assistant Professor, Department of Electronics and Communication Engineering, APJ Abdul Kalam Technological University, Kerala, India, Muthoot Institute of Technology and Science, Cochin, Kerala, India. e-mail: ancyjoy32@gmail.com, orcid: https://orcid.org/0000-0002-2671-3813
[2]Assistant Professor, Department of Electronics and Communication Engineering, APJ Abdul Kalam Technological University (Kerala), India, Muthoot Institute of Technology and Science, Cochin, Kerala, India. e-mail: jishajacob@mgits.ac.in, orcid: https://orcid.org/0000-0002-6246-2388
[3]Undergraduate, Department of Electronics and Communication Engineering, APJ Abdul Kalam Technological University (Kerala), India, Muthoot Institute of Technology and Science, Cochin, Kerala, India. e-mail: 21ec295@mgits.ac.in, orcid: https://orcid.org/0009-0001-9449-0367

Received: June 17, 2025; Revised: September 02, 2025; Accepted: September 25, 2025; Published: October 30, 2025

SUMMARY

Edge detection is a critical image processing operation and is a core aspect of feature extraction as well as object identification. In this paper, we introduce an uncomplicated pipelined hardware architecture for Sobel edge detection and Canny edge detection on an FPGA for comparing their edge detection performance, on-chip power, and resource usage. The Sobel edge method delivers computation simplicity and efficiency, while the Canny algorithm offers more robust and reliable detection of thin edges, which are important in enabling supporting technologies such as self-driving cars, computer vision, medical imaging, etc. It is thus important to have a dedicated hardware design for performing both edge detection algorithms on an embedded system, thereby reducing the dependency on the general processing part of the system. The hardware was designed using hardware description language (Verilog) and was implemented on the Zedboard FPGA. The Zedboard, containing all programmable SoC (AP SoC) Xilinx Zynq-7000, was used for the purpose of testing and analysis of input grayscale images. The edge detection accuracy, power utilization, and resource utilization of both algorithms are analysed in real-time. As a result, this work demonstrates that the Canny edge detection algorithm outperforms Sobel edge detection algorithm for its precise detection of thin edges when implemented on an FPGA. The latter algorithm uses less power and resources on the FPGA, but it can't be used for critical applications.

Key words: *FPGA, edge detection, sobel, canny, embedded system*.

INTRODUCTION

The evolution of edge detection algorithms has been closely tied to advancements in both computer vision theory and hardware technology. The Sobel operator, developed in 1968, represented a significant

breakthrough as a simple yet effective first-order edge detection method. Its computational efficiency made it particularly suitable for early digital image processing systems. In 1986, John Canny introduced his eponymous algorithm, which set new standards for edge detection accuracy through its multi-stage approach incorporating noise reduction and edge thinning. The development of FPGA technology in the 1980s and 1990s opened new possibilities for hardware-accelerated image processing, with researchers beginning to explore FPGA implementations of edge detection algorithms in the early 2000s. Recent years have seen numerous optimizations and variations of both Sobel and Canny algorithms specifically designed for FPGA implementation, focusing on aspects such as parallel processing, memory optimization, and power efficiency. Our work builds upon this rich history by combining the strengths of both algorithms in a single adaptive system, while incorporating modern FPGA design techniques such as pipelining and hardware-software co-design enabled by the Zynq SoC platform.

Image processing is a fundamental element of contemporary technology, facilitating great improvements in medical imaging diagnostics, autonomous navigation, and surveillance. Among the various image processing techniques, edge detection has been a prominent way of detecting object boundaries and extracting valuable structural information from images. Rapid and accurate edge detection is especially crucial in high-accuracy applications and real-time processing demands [1]. Yet, software implementations are typically bound by stringent computational complexity and latency, so hardware acceleration is a good alternative for real-time applications. This study suggests using the Sobel and Canny edge detection techniques to implement edge detection on hardware. The Sobel operator is famous for its ease of use and efficiency in computation and is therefore well-suited for low-complexity applications, while the Canny algorithm offers better detection performance through multi-stage processing, such as suppression of Gaussian noise, non-maximum suppression, and edge thinning [8]. By implementing the algorithms on specialized hardware, processing time is greatly improved with the resulting reduction in computational load on general-purpose processors. The architecture uses a pipelined hardware structure, realized as custom Intellectual Property (IP) cores, and is documented in Verilog Hardware Description Language (HDL) utilizing Xilinx Vivado design tools. A 512×512 grayscale input image is provided, and the edge-detected output after processing is displayed on a screen. System performance is measured in terms of power consumption, resource usage, and accuracy of detection [9]. Experimental results show that the proposed FPGA implementation provides real-time processing capability along with an efficient trade-off between computational efficiency, and accuracy of edge detection and is thus appropriate for real-time embedded applications [2].

LITERATURE REVIEW

D. Rao et al. describe a low-power FPGA design of the Sobel edge detection algorithm for embedded applications, but they do not consider the integration and analysis of the Canny edge detection algorithm on the same hardware [4] [14]. The work of Mohammed Sabah and R. Sundara Guru reviews an enhanced Canny edge detection method and points out that the lack of optimized hardware architectures is a major drawback [3], [6]. Z. Othman et al. compares both Sobel and Canny edge detection algorithms in MRI imaging, illustrating their suitability for medical use but without pre-processed data utilization while not investigating pipelined hardware structures [11]. A. Kumar et al. suggest an FPGA-based real-time Sobel and Canny algorithm-based edge detection system, highlighting the advantages of pipelined architectures for high-speed processing but not exploring the merging of the two algorithms into one common hardware platform [5]. C. Li et al. develop a hybrid FPGA architecture for Sobel and Canny edge detection in autonomous vehicles without a comprehensive analysis of power consumption and resource utilization [15]. Likewise, Mamta Joshi and Ashutosh Vyas have compared Canny, Sobel, and Prewitt edge detectors in various image formats but limited their work to software implementations, with a major omission of hardware validation [7].

These constraints are overcome in this work, where both the Sobel and Canny edge detection techniques are applied on the Zynq-7000 SoC Development Board (ZedBoard). The system proposed makes use of the FPGA fabric for parallel processing at high speeds and the processing system for software implementation. Besides, the low-power 9T SRAM design presented in [13], with significant power savings over traditional 6T SRAM, is applicable to this work. Since real-time FPGA-based edge

detection systems require fast and efficient transfer of intermediate data storage, such low-power SRAM designs can significantly enhance processing speed while reducing energy consumption.

The presented work fuses both Sobel and Canny algorithms into a unified hardware architecture and delivers an integrated edge detection system considering power consumption, resource utilization, and detection accuracy. The combined FPGA-based hardware implementation represents a trade-off between detection accuracy and computational efficiency, making it suitable for real-time applications that need fast and reliable edge detection, such as autonomous navigation, medical imaging, and surveillance systems [12].

METHODOLOGY

The design of the Sobel and Canny edge detection algorithms proposed in this work is implemented and simulated with a Hardware Description Language (HDL). Pipelining is adopted to reduce idle cycles within the processing system for better overall computing efficiency. The system processes a 512×512 black-and-white image, with each pixel described in terms of 8-bit intensity data. For achieving a pipelined architecture, four-line buffers are used for storing the consequent four rows of the input image. Each line buffer is of size 512 for storing each row. These line buffers are used in both the Sobel and Canny edge detection algorithms [10]. Sobel edge detection is implemented with one set of line buffer, while Canny edge detection is implemented in such a way that it contains an independent set of line buffers between individual complex stages. These are designed using Verilog language and are developed as individual IPs (Individual Property) in Vivado software. Later, it is integrated with other standard IPs and integrated with the Zynq processing system for testing with an input image and a display output image using the VGA interface.

While looking at both algorithms, Sobel detects the edges successfully even in areas of minute pixel intensity differences. This can bring up the problem of detection of false edges, which can add unwanted noise to the output image. Also, the single thresholding of this algorithm accounts for less noise immunity. This is improved in Canny by the process of double thresholding.

This section is organized as follows: Section A discusses line buffers and their control logic, which constitute the pipelined architecture, while Sections B and C provides the design of Sobel and Canny edge detection IPs with various processes included in them.

A. Line Buffers and Control Logic

This module implements a simple line buffer used for image processing applications, particularly for storing and retrieving pixel data efficiently. It receives an 8-bit pixel input along with a valid signal and stores the incoming pixel values in a 512-element register array, which acts as a line buffer. A write pointer is used to track the current write position, incrementing each time a valid pixel is received. This allows sequential storage of pixel data in the buffer. The reset signal initializes the write pointer to zero when activated. The module enables continuous pixel storage while ensuring that data is written to the correct location in the buffer. The figure 1 shows the architecture of the line buffers used in the design.

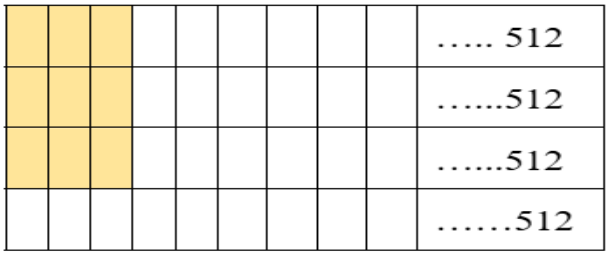

Figure 1. Line buffers

When the chunk of image data is read, the module outputs a data window that consists of 24 bits of data from the buffer, which comprises three successive 8-bit pixels read from the buffer starting from the

read pointer, on the read pointer + 1 and the read pointer + 2. Therefore, the image data provides a 3-pixel wide window to use for convolution, and other image processing techniques. When the image is moving through the read data signal assertion, the read pointer will start to increment until reaching the end of the image line. This design allows for image data to be streamed in sequentially through a buffer, which is desirable for near real-time edge detection and image-filtering applications by providing a stable stream of overlapping sets of pixels to be processed.

The control logic ensures the correct flow of pixel data through the four module set of line buffers, for buffering and buffering image information in a manner preferably for further processing. The control logic accepts an 8-bit input pixel data and valid signal for the pixel and then buffers the incoming pixel into four line parallel line buffers as they are coming in. There is a write pointer which is incremented through four modules to store pixel values in a sequence order; there is also a counter which keeps track of how many pixels are processed. Furthermore, there is a second counter that guarantees that enough pixels have been buffered before the read process begins. This module operates in a two-state system: IDLE, where it waits for an adequate number of pixels. The read operation is controlled by the read counter, which increments with each clock cycle when the buffer is being read. Figure 2 shows how the pixel data is entered into processing modules. This line buffers and their control are used in both Edge Sobel and Canny Edge detection algorithms for maintaining a pipelined architecture. Once 512 pixels have been processed, the module resets to IDLE state and generates an interrupt. The output is constructed by concatenating the pixel data from three adjacent lines to form a 3x3 matrix for convolution operations. The control signals determine which buffers are being read in each cycle. This design ensures a continuous and efficient streaming of pixel data, making it suitable for applications like image filtering and edge detection, where processing requires access to neighbouring pixel values.

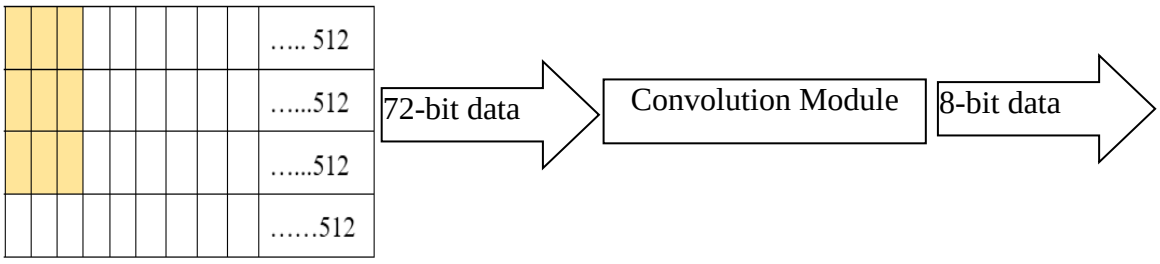

Figure 2. Line buffer data control

B. Sobel Edge Detection Ip

This module operates on image data using a convolution operation in an FPGA-based system. It accepts an 8-bit pixel stream via an AXI streaming slave interface, and is subsequently processed through multiple stages. The first is a Line Buffer module that buffers the incoming pixel data, shrinks and formats the data as a 72-bit window that represents a 3x3 neighbourhood, and indicates when it is ready to send the valid pixel data for processing. This pixel data then proceeds to the convolution module where the convolution operation outputs the relevant edge or feature information from the image, as well as its valid signal. The convolution output is then sent to the next stage along with its valid signal. At the same time, the interrupt signal can be activated as an interrupt when the image data is actively being processed or is complete. The processed convolution output sends the pixel data to a FIFO generator IP, which receives the pixel data stream, acts as an output buffer, and sends a stream of pixel data to the IP AXI Master interface. FIFO output also manages the data stream and flow, to ensure that pixel data is not overflowing, using a programmable full signal that indicates when the component is ready to accept further data, when the processed data is complete. When the processed pixel data is available as an output, it will also be accompanied by a valid signal.

$$g = \sqrt{g_x^2 + g_y^2} \ldots\ldots \quad (1)$$

$$\theta = \tan^{-1}\left(\frac{g_y}{gx}\right)\ldots\ldots\ldots \quad (2)$$

Where "g" represents the edge's strength and "Θ" represents the edge's direction.

This convolution module applies a 3x3 Sobel convolution on an input pixel window to detect edges within an image. The module takes a 72-bit input for a total of a 3x3 pixel block. The convolutional module applies the two Sobel kernels (Kernel 1 detects a horizontal gradient; Kernel 2 detects a vertical gradient). The module first multiplies every input pixel by a corresponding value in the kernel and saves both outputs in two different places. Once the two multiplied values are obtained, they are summed to get the sums of gradients in the x direction and y direction, as well as the g_x^2 and g_y^2. We need to square the sums of the x and y since we want to add them together to find the final gradient magnitude. Additionally, all of these operations are done in a pipelined fashion by utilizing multiple clock cycles to allow for more efficient implementations. Finally, it performs the thresholding operation as normal to determine if the pixel is an edge pixel. If its gradient magnitude was greater than a threshold of 8000, the final output is 255; otherwise, it is 0. The valid signal is present to determine that valid data is output in a correct manner. This module is significant to edge detection applications like Sobel Filtering, where it operates to show the change in intensity in an image.

C. Canny Edge Detection Ip

The figure 3 indicates a custom AXI Stream (AXI) IP core with slave and master interfaces for the Canny edge detection algorithm. It is envisioned to carry out the Canny edge detection algorithm in an organised, pipelined fashion. The dataflow begins with the pixel stream of the input image, which is received from an AXI-stream interface. The line buffer module mentioned in section *A*. It first captures the input pixel data and validates it before passing it to the Gaussian Blur module, which performs noise reduction using a Gaussian filter. The filtered output is then processed through another line buffer module, which formats the data for the Gradient Calculation module. This module computes gradient magnitudes and directions, which are then sent through another instance of the line buffers to ensure proper synchronization before being passed to the next stage.

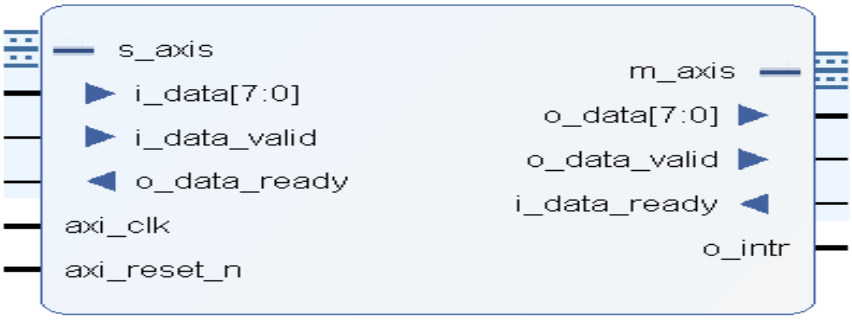

Figure 3. Canny edge detection IP

After the gradient magnitudes and directions calculation, this data becomes the input to the Non-Maximum Suppression (NMS) module, which performs non-maximum suppression to thin out edges. The refined edge data is then passed to the Double Thresholding module to classify strong and weak edges. The processed data is further handled by an instance of the line buffer module before being sent to the Edge Tracking module, which performs edge tracking by hysteresis. Finally, the fully processed edge-detected data is sent to an instance of FIFO generator IP that acts as an output buffer, which manages the AXI-stream output interface. Throughout the pipeline, line buffer modules are used to ensure proper data synchronization between processing stages, maintaining a smooth and efficient hardware-accelerated image processing flow. figure 4 demonstrates the data flow in a Canny edge detection design.

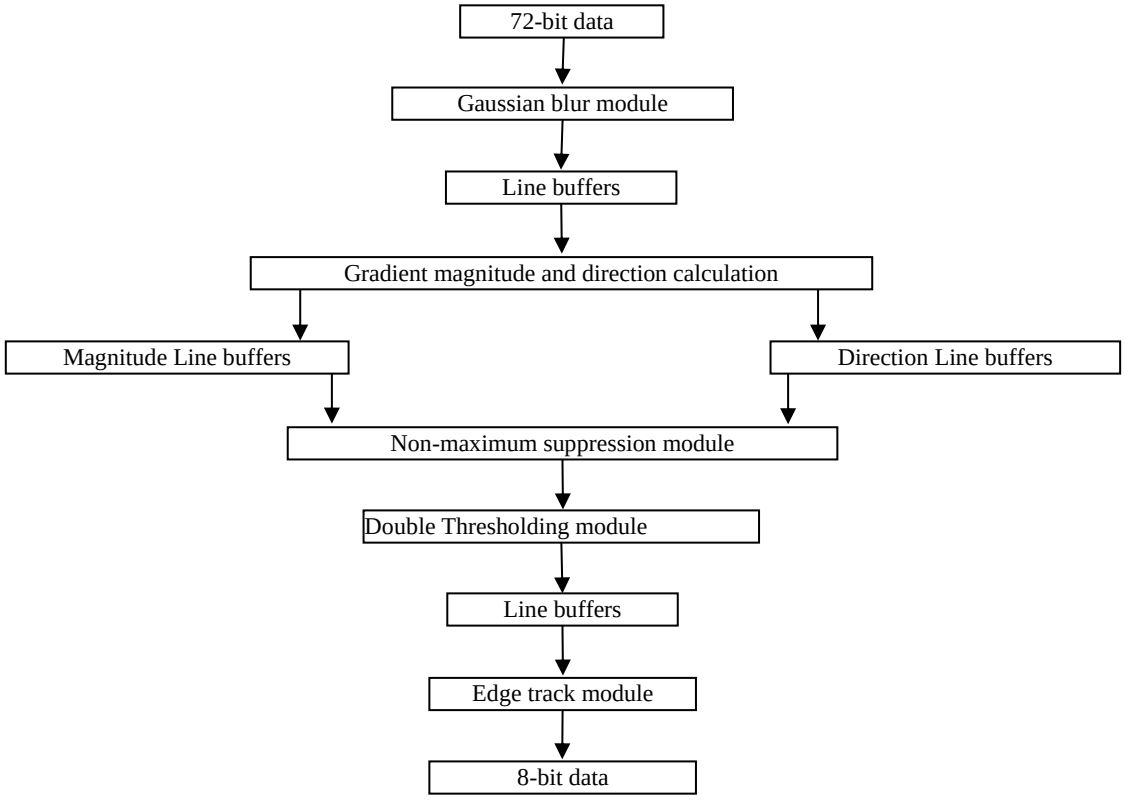

Figure 4. Dataflow in canny edge detection IP

The Gaussian Blur module applies a fixed 3×3 Gaussian filter to a 3×3 pixel window, multiplying each 8-bit pixel by corresponding kernel weights, summing the results, and normalizing by dividing by 16 to produce the blurred output. It operates in a pipelined structure with sequential and combinational always blocks, ensuring synchronized processing and low latency on FPGA hardware. The blurred 8-bit output, with valid data signals, is sent to the line buffer module for further edge detection processing, such as the Sobel and Canny algorithms. The Gradient Calculation module uses the Sobel edge detection algorithm to compute gradient magnitude and direction from a 3×3 pixel window by multiplying pixels with the Sobel X and Y kernels to obtain Gx and Gy. A division IP core and hardware square root function are used to classify edge directions (0°, 45°, 90°, 135°) and calculate gradient magnitude, ensuring accurate edge detection. The pipelined design with valid signal synchronization enables efficient FPGA implementation, outputting 8-bit magnitude and direction data to line buffer modules for further processing. The Vivado Divider Generator IP performs signed 32-bit integer division of Gy (dividend) by Gx (divisor) to compute the gradient direction for the Sobel edge detection. It operates synchronously with valid signal control, outputs the quotient for edge direction classification (0°, 45°, 90°, 135°), and prevents division-by-zero errors. Optimized for FPGA with pipelined architecture and AXI streaming support, it ensures high-speed, low-latency operation suitable for real-time image processing.

The Non-Maximum Suppression (NMS) module refines edges in the Canny pipeline by processing synchronized 72-bit gradient magnitude and direction inputs. It classifies edge directions (0°, 45°, 90°, 135°), compares the magnitude of the centre pixel with its two neighbours along the edge direction, and suppresses non-maximal pixels to retain only strong edge candidates. Operating synchronously with valid signal control, it outputs an 8-bit processed pixel stream for the next stage, typically double thresholding.

The Double Thresholding module classifies edge pixels based on intensity using two thresholds: pixels above 220 are marked as strong edges (255), those below 85 are suppressed to 0, and intermediate values are treated as weak edges. It operates synchronously, updating outputs only on the positive clock edge when the valid signal is asserted, ensuring proper pipeline timing. This stage refines edge detection by distinguishing strong and weak edges, preparing data for edge tracking by hysteresis in the next stage.

The Edge Tracking by Hysteresis module refines edges by evaluating a 72-bit pixel window, classifying the centre pixel as a strong edge (255) if its value is above 100, or suppressing it (0) if below 50. Pixels between the two thresholds are retained as edges only if at least one neighbouring pixel is a strong edge, ensuring edge continuity. Operating synchronously with valid signal control, this stage removes weak, isolated edges, producing a cleaner and more accurate final edge map.

HARDWARE IMPLEMENTATION

The design for the hardware was created with Xilinx Vivado design tools and heavily simulated to ensure functionality and remove potential bugs before actual implementation in hardware. A 512×512 grayscale image was employed to evaluate and verify the performance of the proposed design. Real-time implementation was performed on a ZedBoard development board, which includes a Zynq-7000 All Programmable System-on-Chip (APSoC). The edge-detected processed output, derived utilizing Sobel as well as Canny algorithms, was presented on a VGA-connected monitor. The entire hardware setup for testing design functionality on the ZedBoard is shown in figure 5. The architecture is designed to be completely compatible with the ZedBoard's FPGA fabric. Both edge detection algorithms are designed as custom Intellectual Property (IP) cores, which are interfaced with the Zynq Processing System (PS) to test hardware performance. The test image employed for testing is presented in figure 6.

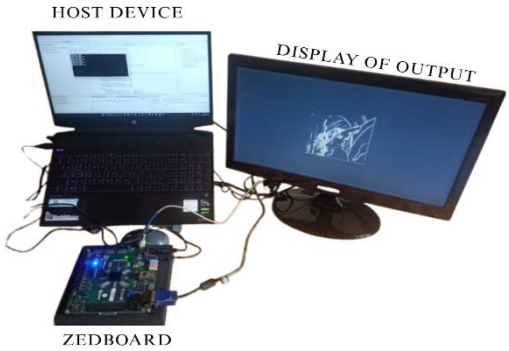

Figure 5. Test setup

Figure 6. Input image

EXPERIMENTAL RESULTS

The performance and effectiveness of the two hardware implementations are measured and contrasted in terms of important parameters, such as edge detection accuracy, power efficiency, and usage of hardware resources. The comparative evaluation gives a clear picture of the relative merits of each algorithm and how one performs better than the other when run on FPGA hardware.

A. Edge Detection Accuracy

The edge detection accuracy [figure 7 and figure 8] of both images was compared using Python with a ground truth image. The edge detection accuracy of both images was compared using metrics like Precision, Recall, F1-Score, and IOU. Precision defines how many of the detected edges are actually correct. Canny produces more accurate edge detections with fewer false positives. Recall measures how many actual edges were detected. A higher recall means fewer missed edges. Canny detects more true edges and misses fewer compared to Sobel. The harmonic mean of precision and recall, balancing both aspects, is called the F1-score. Canny achieves a better balance between precision and recall. Intersection over Union (IoU) measures the overlap between detected edges and the ground truth image. Canny has a better overlap with the ground truth edges. Canny edge detection performs better than Sobel edge detection in all four measures. It identifies more accurate edges with higher precision and closer agreement with real edges. Sobel, as a less complex gradient-based algorithm, is less precise but faster to compute. The Canny algorithm was found to be more productive by filtering the noise content and providing a clean, well-contrast image by obtaining well-defined and continuous edges.

Figure 7. Sobel edge detected image

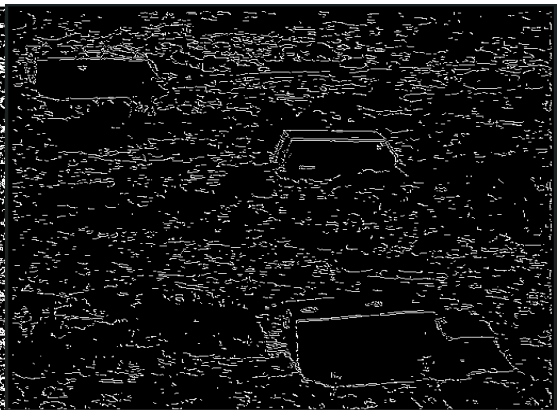

Figure 8. Canny edge detected image

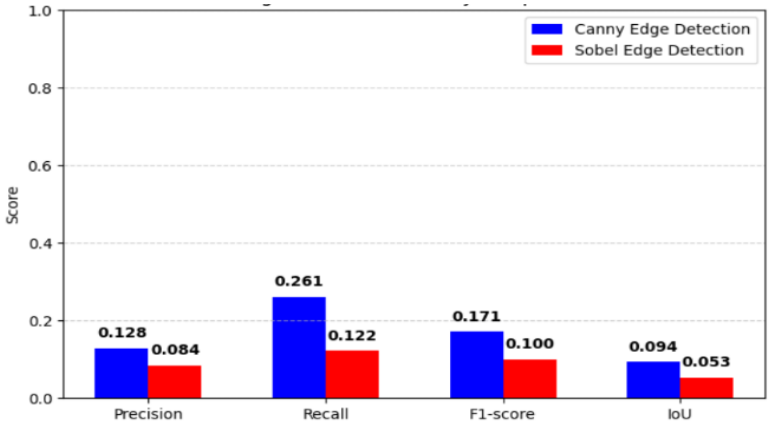

Figure 9. Edge detection accuracy comparison

B. Power Utilization

Figure 10 and figure 11 show a comprehensive breakdown of the power usage of the Sobel and Canny edge detection algorithms executed on the FPGA. The power reports were extracted directly from Xilinx Vivado during the hardware design synthesis. The on-chip total power usage is broken down into dynamic and static parts, offering insight into the energy efficiency of each implementation. The outcome shows that the Canny edge detection algorithm has a higher power consumption compared to the Sobel algorithm when executed on FPGA hardware. As a result, the Sobel edge detection algorithm is suitable for non-critical applications where low power consumption is a top priority. figure 12 shows the graph plotted between various power components and their consumed power for both detection algorithms when implemented on an FPGA.

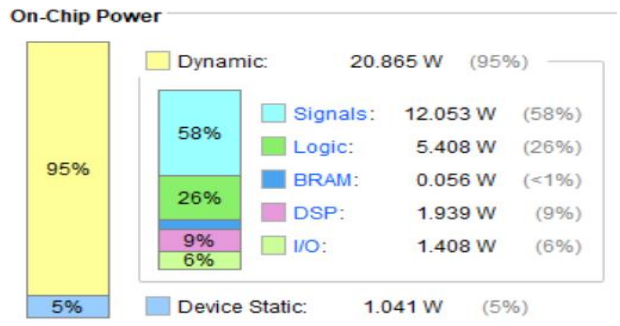

Figure 10. Power utilisation of sobel algorithm

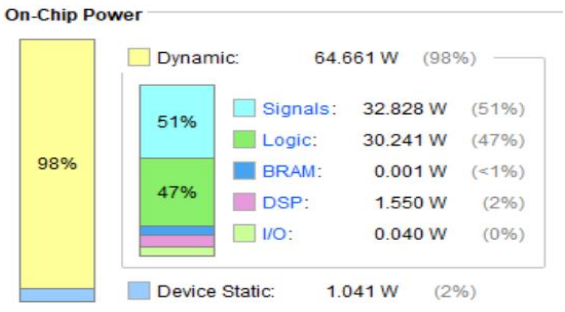

Figure 11. Power utilization of canny algorithm

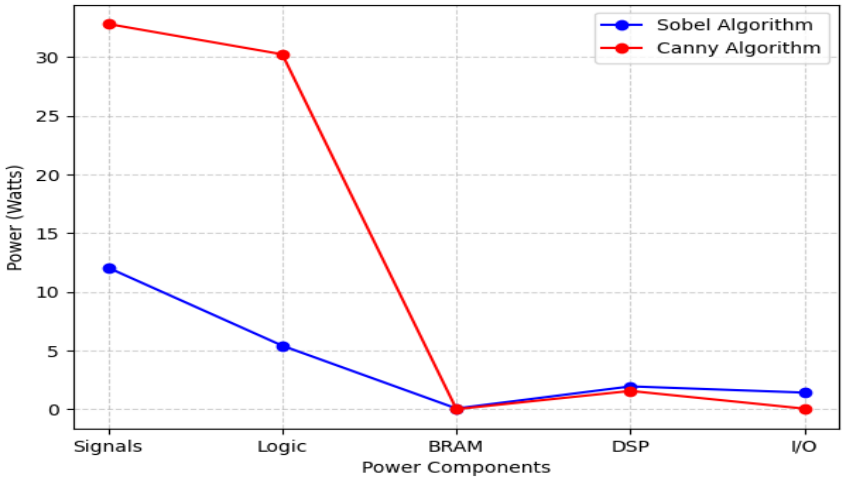

Figure 12. Comparison of power utilization of sobel and canny algorithms

C. Resource Utilization

Table 1 presents the hardware resource utilization of the Sobel edge detection algorithm on the FPGA. As shown in the table.2, Canny edge detection uses more lookup tables and flip-flops than the Sobel edge detection algorithm. This pattern reflects the increased computational memory demands of the Canny algorithm due to the more complex edge detection process. Based on the above findings, on resource utilization, it is inferred that Canny edge detection cannot be implemented as a hardware module where area and cost are critical. figure 13 shows the plot of resource utilisation of the Sobel and Canny algorithms on an FPGA.

TABLE 1. Resource utilization for the sobel algorithm

Resource	**Utilization**	**Available**	**Utilization (%)**
LUT	1905	53200	3.58
LUTRAM	1154	17400	6.63
FF	253	106400	0.24
BRAM	0.50	140	0.36
DSP	2	220	0.91
IO	23	200	11.50
BUFG	1	32	3.13

TABLE 2. Resource utilization for the canny algorithm

Resource	**Utilization**	**Available**	**Utilization (%)**
LUT	8849	53200	16.63
LUTRAM	4936	17400	28.37
FF	4170	106400	3.92
BRAM	0.50	140	0.36
DSP	2	220	0.91
IO	23	200	11.50
BUFG	1	32	3.13

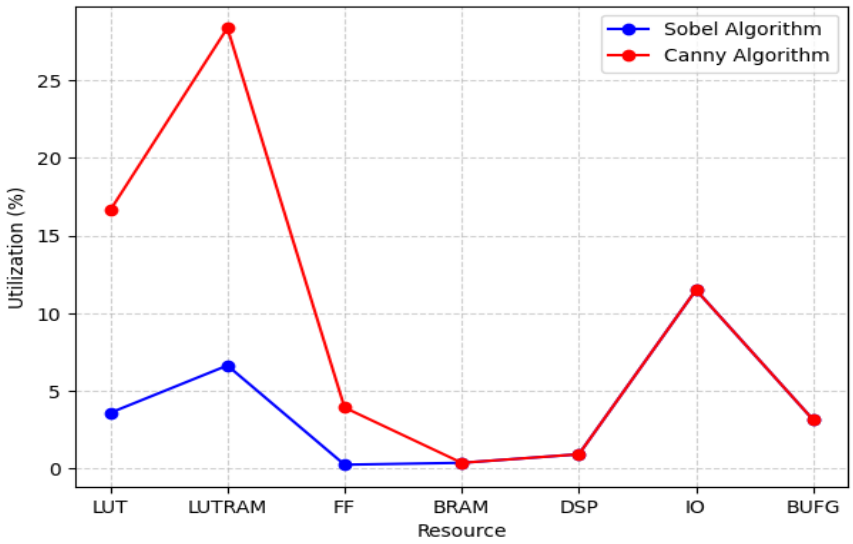

Figure 13. Comparison of the resource utilization of the sobel and canny algorithms

CONCLUSION

The application of the two edge detection algorithms in hardware on an FPGA demonstrates that the Sobel edge detection algorithm is less accurate than the Canny algorithm. Although Sobel consumes less power and requires fewer resources, its performance is limited by noise sensitivity. This is because, since it operates using a single threshold, hence it is not as stable under noisy conditions. On the contrary, the Canny edge detection is more precise in detection and demonstrates better robustness in the presence of noise, primarily because, since it uses double thresholding and advanced edge refinement processes. The improvements, however, come at the cost of higher power consumption and heavier utilization of resources in FPGA hardware.

Thus, for FPGA implementations, Canny is the preferred algorithm for high-precision edge detection applications, such as AI-based technology, robotics, and computer vision-based inspection systems. Nevertheless, the more computationally efficient one is still the Sobel algorithm and is suitable for real-time applications where low power consumption and small processing capability are the main issues. In conclusion, the Edge Sobel detection algorithm module and Canny Edge detection algorithm module can be concluded based on the following parameters and, their remarks are shown in table 3.

TABLE 3. Comparison of sobel and canny edge detection algorithms on an FPGA

Parameter	**Edge Sobel Detection Algorithm**	**Canny Edge Detection Algorithm**
Edge detection accuracy	Inaccurate	Accurate
Power utilization	Low	High
Resource utilization	Fewer resources required	More resources required
Noise tolerance	Highly prone to noise because of using a single threshold	Less prone to noise due to double thresholding

REFERENCES

[1] Joy A, Kuruvilla J. Design of 1T2R ReRAM array for in memory element-wise multiplication with distributed and majority logics. Integration. 2025 Jul 1;103: 102418. https://doi.org/10.1016/j.vlsi.2025.102418

[2] Maleki EN. Concurrent BIST for Embedded SRAMs in SoCs. International Academic Journal of Science and Engineering. 2017;4(2):59–74.

[3] Sabah M, Sundaraguru R. A Survey on Improved Canny-Edge Detection Algorithm. Perspectives in Communication, Embedded-systems and Signal-processing-PiCES. 2018 Aug 5;2(4):73-8.

[4] Wilamowski GJ. Embedded system architectures optimization for high-performance edge computing. SCCTS Journal of Embedded Systems Design and Applications. 2025;2(2):47-55.
[5] Guo L, Wu S. FPGA implementation of a real-time edge detection system based on an improved Canny algorithm. Applied sciences. 2023 Jan 8;13(2):870.
[6] Holovati JL, Zaki FM. Energy-aware task scheduling in heterogeneous GPU/TPU–FPGA embedded platforms. InECCSUBMIT Conferences 2025 Jun 17; 3(2):16-27.
[7] Joshi M, Vyas A. Comparison of Canny edge detector with Sobel and Prewitt edge detector using different image formats. Int. J. Eng. Res. Technol. 2020;1:133-7.
[8] Ramakrishnan V, Alsalami ZA, Khujanova O, Kushmanov J, Djabbarova S, Nandy M. Edge-Centric Mobile Internet Architecture for Ultra-Low Latency Services. Journal of Internet Services and Information Security. 2025;15(2):807–816. https://doi.org/10.58346/JISIS.2025.I2.053.
[9] Joy A, Kuruvilla J. optimized resistive ram using 2t2r cell and it's array performance comparison with other cells. Archives for Technical Sciences/Arhiv za Tehnicke Nauke. 2025 Jan 1(32). https://doi.org/10.70102/afts.2025.1732.044
[10] Yemunarane K, Chandramowleeswaran DG, Subramani K, ALkhayyat A, Srinivas G. Development and Management of E-Commerce Information Systems Using Edge Computing and Neural Networks. Indian Journal of Information Sources and Services. 2024;14(2):153-9. https://doi.org/10.51983/ijiss-2024.14.2.22.
[11] Othman Z, Haron H, Kadir MR, Rafiq M. Comparison of Canny and Sobel edge detection in MRI images. Computer Science, Biomechanics & Tissue Engineering Group, and Information System. 2009 Jun:133-6.
[12] Sathiyamurthy A, Gajendran Subba Naidu S. Designing edge computing solutions for real-time vessel tracking and collision avoidance. Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications. 2025;16(2):863–874. https://doi.org/10.58346/JOWUA.2025.I2.053
[13] Joy A, Kuruvilla J. A stable low power dissipating 9 T SRAM for implementation of 4× 4 memory array with high frequency analysis. Wireless Personal Communications. 2022 Oct;126(4):3305-16. https://doi.org/10.1007/s11277-022-09865-x
[14] Ravichandran S, Su HK, Kuo WK, Dhanasekaran D, Mahalingam M, Yang JP. Parallel processing of Sobel edge detection on FPGA: enhancing real-time image analysis. Sensors. 2025;25(12):3649.
[15] Baloch A, D Memon T, Memon F, Lal B, Viyas V, Jan T. Hardware synthesize and performance analysis of intelligent transportation using canny edge detection algorithm. International Journal of Engineering and Manufacturing. 2021;11(4):22-32.