Hardware-in-the-Loop (HIL) Simulation of a Sobel Edge Detection Algorithm for Real-Time Image Processing

Majed Mohammed Alzouri[1,*], Mohammed Gronfula[2]

[1]College of Engineering, Alasala Colleges, Dammam 31435, Eastern Province, Saudi Arabia
[2]College of Engineering, Alasala Colleges, Dammam 31435, Eastern Province, Saudi Arabia

Email: 11800001@alasala.edu.sa, mohammed.gronfula@alasala.edu.sa

Manuscript received November 28, 2025; accepted December 29, 2025

*Corresponding author: Majed Mohammed Alzouri

***Abstract*—This paper presents a comprehensive Hardware-in-the-Loop (HIL) simulation framework for developing and verifying a real-time Sobel edge detection system. Modern image processing applications, such as surveillance and autonomous vehicles, require high-speed edge detection capable of processing video streams in real time. Traditional software implementations often fail to meet strict timing constraints. To address this, we propose a simulation-based design methodology using MATLAB and Simulink to model hardware behavior and verify performance without the immediate need for physical hardware. The study implements an optimized Sobel algorithm using an absolute sum approximation method, which replaces computationally expensive square root operations. We developed a Simulink model with a pipelined architecture to mimic hardware parallelism. The system was validated through HIL simulation, processing video streams with an average processing time of 39.95 milliseconds per frame, achieving a frame rate of 25.03 frames per second. The optimized algorithm achieved a speedup factor of up to 1.46× compared to the standard implementation while maintaining structural similarity above 96%. The results demonstrate that the proposed simulation framework provides an accurate and efficient method for validating real-time image processing systems.**

***Index Terms*—Real-time systems, Sobel edge detection, HIL simulation, FPGA, Simulink, Image processing, Hardware modeling.**

I. Introduction

IMAGE processing has become a critical technology in various modern applications, ranging from autonomous driving and surveillance systems to medical imaging [1]. These applications rely heavily on the ability to analyze visual data quickly and accurately. A fundamental step in this analysis is edge detection, which identifies boundaries between objects by detecting significant changes in image intensity. By extracting edges, systems can reduce the amount of data to be processed while preserving essential structural information [2].

However, real-time processing poses a significant challenge. For a video stream to be considered real-time, each frame must be processed before the next one arrives. For standard video at 25 or 30 frames per second (fps), the processing time per frame must be less than 40 milliseconds. Software implementations running on general-purpose computers often struggle to meet these speed requirements due to the sequential nature of their execution [3]. This limitation drives the need for specialized hardware solutions, such as Field Programmable Gate Arrays (FPGAs), which can perform parallel processing [4].

Designing hardware directly is complex, time-consuming, and expensive. Errors discovered late in the development process can be costly to fix. To mitigate these risks, Hardware-in-the-Loop (HIL) simulation offers a powerful alternative [5]. HIL simulation allows engineers to model hardware behavior and test system performance in a virtual environment that mimics real-world conditions. This approach enables thorough verification of timing and functionality before any physical hardware is built [6].

This paper focuses on the design, simulation, and verification of a Sobel edge detection system using MATLAB and Simulink. We aim to bridge the gap between algorithm development and hardware implementation by providing a complete simulation workflow. Our contributions are as follows:

- Implementation of an optimized Sobel edge detection algorithm using absolute sum approximation to reduce computational complexity.
- Development of a Simulink model that mimics hardware behavior, including pipelining and line buffering, for real-time video processing.
- Comprehensive HIL simulation to validate system performance, targeting a processing time of less than 40 ms per frame.
- Comparative analysis between standard and optimized implementations, demonstrating significant speedup with minimal loss in accuracy.

The remainder of this paper is organized as follows. Section II presents a literature review of edge detection and HIL simulation. Section III details the methodology and algorithm optimization. Section IV describes the system implementation in MATLAB and Simulink. Section V presents the experimental results and discussion. Finally, Section VI concludes the paper.

II. Literature Review

Edge detection is a well-researched area in image processing. Various operators have been developed, including Prewitt, Roberts, and Canny. The Sobel operator remains one of the most popular choices due to its balance between simplicity and effectiveness [7]. It uses a pair of 3×3 convolution kernels to detect gradients in horizontal and vertical directions.

Recent studies have explored hardware implementations of edge detection to achieve real-time performance. Navinkumar et al. [4] demonstrated an FPGA implementation of the Sobel algorithm, highlighting the efficiency of hardware over software solutions. Similarly, Ravichandran et al. [8] proposed a parallel processing architecture for Sobel edge detection on FPGA, achieving high frame rates for high-resolution images. These studies confirm that hardware acceleration is essential for real-time applications [9], [10].

HIL simulation has emerged as a standard verification methodology in many engineering fields. Mihalič et al. [5] provided a historical overview of HIL challenges, emphasizing its importance in reducing development time. In the context of image processing, Komorkiewicz et al. [11] presented an FPGA-based HIL environment for automotive camera systems, using video injection to test hardware. However, there is a need for more accessible simulation frameworks that allow algorithm developers to verify hardware behavior using high-level tools like MATLAB and Simulink before moving to HDL coding [12], [13].

This work addresses this gap by providing a detailed model-based design approach. We utilize Simulink's capabilities to model hardware-specific features such as line buffers and data streams, allowing for accurate performance estimation in the early design phases [14], [15].

III. Methodology

This section outlines the design approach adopted for developing the Sobel edge detection system. The methodology follows a systematic flow from algorithm verification in MATLAB to hardware behavior modeling in Simulink, and finally to HIL simulation.

A. Design Flow

The development process consists of three main phases, as illustrated in Fig. 1. 1) **MATLAB Implementation:** The algorithm is first implemented and optimized in MATLAB to establish a baseline for accuracy and performance using static images. 2) **Simulink Modeling:** The verified algorithm is translated into a Simulink model. This model is designed to mimic hardware behavior, processing video data as a stream of pixels rather than full frames. 3) **HIL Simulation:** The complete system is simulated with video inputs to verify real-time performance and system stability.

B. Sobel Algorithm and Optimization

The standard Sobel operator uses two kernels, G_x and G_y, to calculate the horizontal and vertical gradients of the image intensity. The kernels are defined as:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1}$$

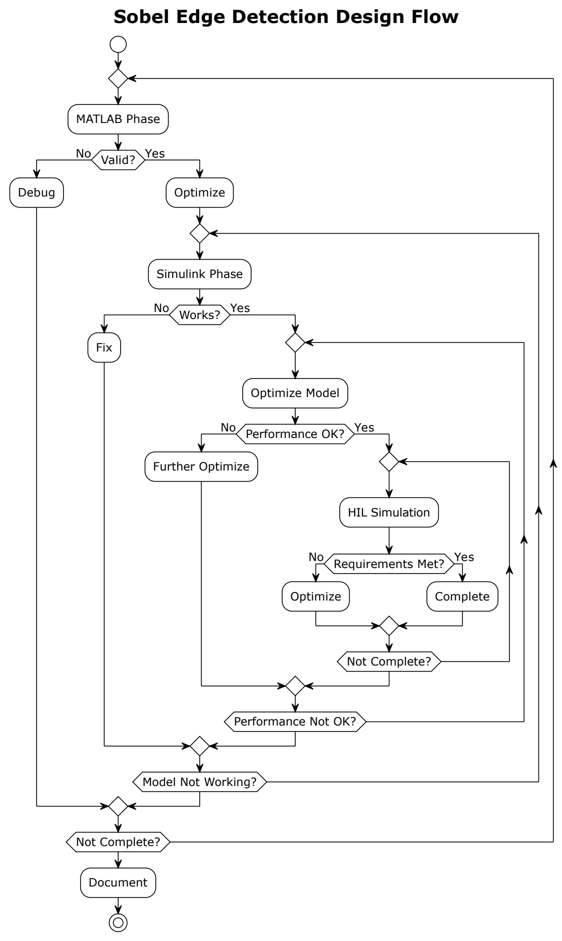

Fig. 1. Overall design flow for the Sobel edge detection simulation system.

The convolution operation calculates the gradient components at each pixel location. For a pixel at position (i, j) in the image, the horizontal gradient $g_x(i, j)$ and vertical gradient $g_y(i, j)$ are computed by convolving the image with the respective kernels defined in (1). The gradient magnitude $|G|$ is traditionally calculated using the Euclidean distance:

$$|G| = \sqrt{g_x^2 + g_y^2} \tag{2}$$

Calculating the square root in (2) is computationally expensive and resource-intensive for hardware implementation. To optimize the algorithm for real-time processing, we use the "Absolute Sum Approximation." This method approximates the magnitude by summing the absolute values of the gradients:

$$|G| \approx |g_x| + |g_y| \tag{3}$$

This optimization in (3) replaces the complex square root and square operations with simple addition and absolute value operations, significantly reducing computational complexity [16]. The process involves convolving the input image with the kernels defined in (1), calculating the approximate magnitude using (3), and then applying a threshold to generate the binary edge map [17].

The gradient direction can also be calculated using the arctangent function:

$$\theta = \arctan\left(\frac{g_y}{g_x}\right) \tag{4}$$

where θ represents the edge direction in radians. However, for this work, we focus primarily on edge magnitude, as it is sufficient for most real-time applications.

C. Accuracy Assessment

While the approximation in (3) improves speed, it is crucial to ensure that it does not significantly degrade detection accuracy. We evaluate the accuracy using the Structural Similarity Index (SSIM) and Mean Squared Error (MSE) defined in (6) by comparing the output of the optimized algorithm against the standard implementation. A high similarity score confirms that the optimization is suitable for the target application.

IV. System Implementation

This section describes the detailed implementation of the system in MATLAB and Simulink, focusing on the transition from a software-based algorithm to a hardware-oriented simulation model.

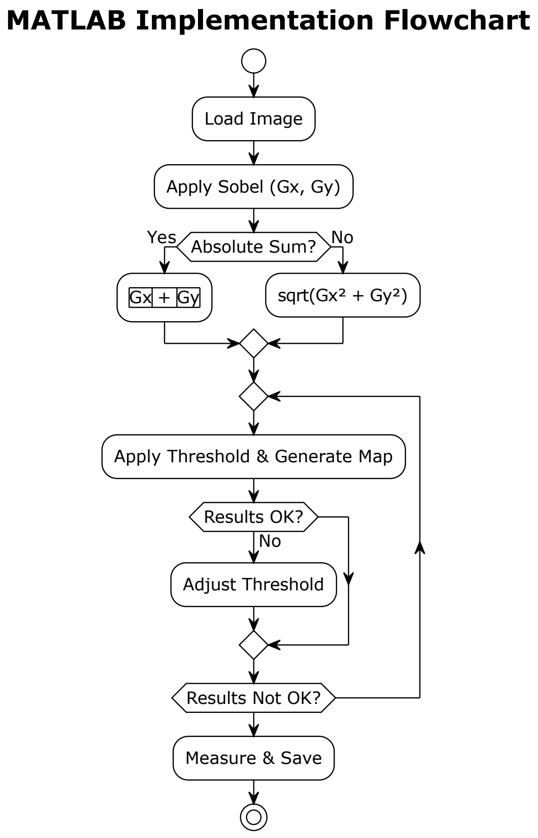

Fig. 2. MATLAB implementation flowchart for Sobel edge detection.

A. MATLAB Implementation

The MATLAB implementation serves as the ”Golden Reference” for the project. It processes images using matrix operations, which is efficient for software but does not reflect hardware behavior. The code is structured to handle image loading, grayscale conversion, kernel convolution, and thresholding, as illustrated in Fig. 2. We implemented both the basic version using (2) and the optimized version using (3). Performance was measured using high-resolution images up to 2048×2048 pixels. The results from this phase were used to verify the correctness of the subsequent Simulink models.

B. Simulink Model Architecture

The Simulink model is designed to process video streams in a way that mimics an FPGA or ASIC implementation. The key architectural feature is the use of a ”Pixel-Stream” processing approach [18]. The model architecture, shown in Fig. 3, comprises three main subsystems: 1) **Input Subsystem:** Handles video acquisition and converts frames into a pixel stream. 2) **Processing Subsystem:** Contains the core Sobel logic. It uses line buffers to store incoming pixels, allowing the 3×3 convolution kernels to operate on a continuous stream of data. This simulates the internal memory buffers of a hardware device. 3) **Output Subsystem:** Converts the processed pixel stream back into video frames for display and analysis.

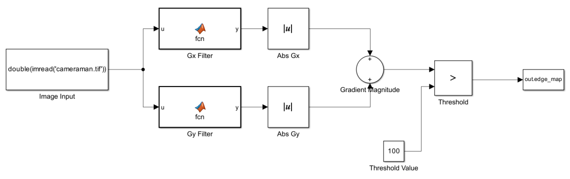

Fig. 3. Simulink Sobel edge detection model: Gx and Gy filters, absolute values, gradient magnitude, and thresholding (threshold = 100).

C. HIL Simulation Setup

For the HIL simulation, we expanded the model to include real-time monitoring capabilities. As illustrated in Fig. 4, the model includes parallel processing paths to validate performance:

- **Sobel with Timing:** Uses a MATLAB Function block to execute the algorithm and measure the exact execution time using internal timers (‘tic‘ and ‘toc‘). This provides precise data on how long the algorithm takes to process each frame.
- **Manual Sobel:** Implements the algorithm using discrete Simulink blocks (Add, Abs, Gain) to verify the logic flow visually.

The simulation is configured to process a standard video stream. Performance monitors track the frame count, processing time per frame, and compliance with the real-time deadline (40 ms).

Fig. 5 details the internal structure of the Sobel Edge Detection subsystem, showing the two parallel processing paths. The top path uses a MATLAB Function block that implements the complete Sobel algorithm and measures processing time. The bottom path implements the algorithm using individual filter blocks: Gx and Gy filters for gradient calculation, absolute value blocks for magnitude approximation, a sum block for combining gradients, and a relational operator for thresholding.

V. Experimental Results and Analysis

In this section, we present the results obtained from the MATLAB verification and the HIL simulation. We analyze the system’s performance in terms of processing speed, accuracy, and resource utilization.

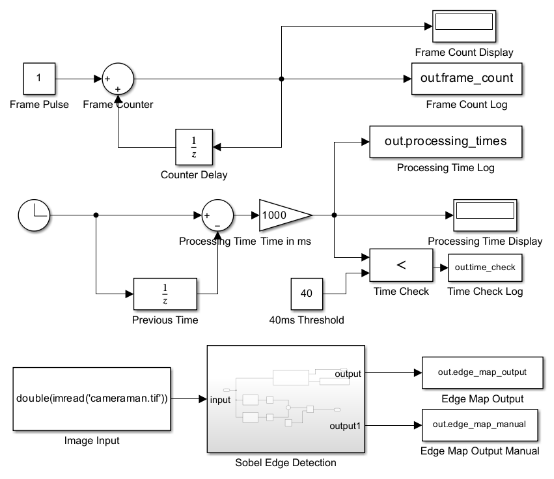

Fig. 4. HIL simulation model (`hil_simulation.slx`) showing parallel processing paths and performance monitoring blocks.

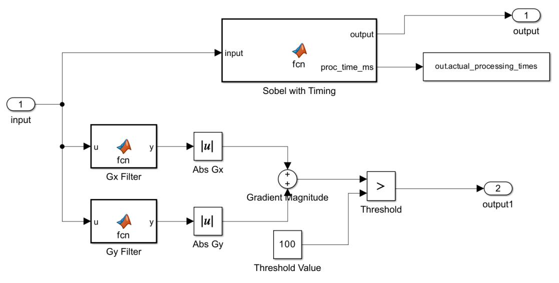

Fig. 5. Sobel Edge Detection subsystem: Parallel processing paths (Sobel with Timing and Manual Sobel).

A. MATLAB Performance Analysis

We first evaluated the performance of the optimized algorithm in MATLAB. Table I compares the processing time of the basic (Basic) and optimized (Opt) implementations across various image resolutions.

Table I
MATLAB PERFORMANCE: BASIC VS. OPTIMIZED SOBEL

Image Size	Basic (ms)	Opt (ms)	Speedup	Similarity (%)
256×256	0.94	0.64	1.46×	96.30
512×512	3.02	2.23	1.35×	96.77
1024×1024	9.37	7.08	1.32×	98.22
2048×2048	35.79	28.58	1.25×	98.99

Table II
DETAILED MATLAB PERFORMANCE RESULTS (AVERAGE OF 10 RUNS)

Image Size	Basic (ms)	Opt (ms)	Speedup	Reduct (%)
256×256	0.94 ± 1.36	0.64 ± 0.65	1.46×	31.6
512×512	3.02 ± 0.23	2.23 ± 0.25	1.35×	26.0
1024×1024	9.37 ± 0.63	7.08 ± 0.43	1.32×	24.4
2048×2048	35.79 ± 2.74	28.58 ± 0.88	1.25×	20.1

The results in Table II confirm that processing time scales quadratically with image size, consistent with the $O(n^2)$ complexity. The optimized implementation consistently outperforms the basic version, with speedup factors decreasing slightly as image size increases due to memory overhead.

Table III
OPTIMIZATION IMPACT: PERFORMANCE VS. ACCURACY

Image Size	Speedup	Time Reduct	Similarity	Loss
256×256	1.46×	31.6%	96.30%	3.70%
512×512	1.35×	26.0%	96.77%	3.23%
1024×1024	1.32×	24.4%	98.22%	1.78%
2048×2048	1.25×	20.1%	98.99%	1.01%

Table III highlights the trade-off between speed and accuracy. The accuracy loss is minimal ($< 4\%$) and decreases for larger images, making the optimization highly effective for high-resolution processing.

Optimization impact analysis evaluates the performance improvements achieved through the absolute sum approximation method. The analysis compares processing time, speedup factors, and accuracy trade-offs between the basic and optimized implementations. Fig. 6 visualizes the relationship between processing time and image size for both basic and optimized implementations, clearly demonstrating the scalability benefits of the optimization.

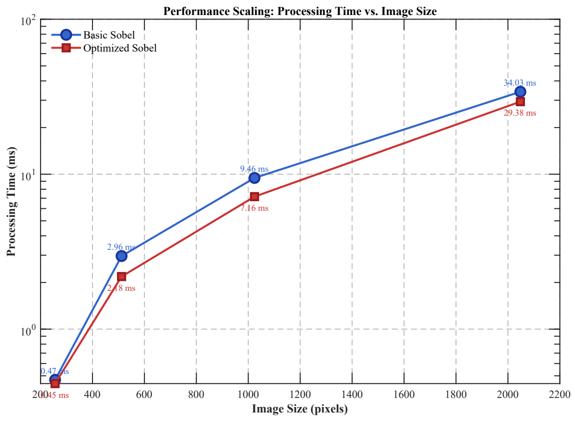

Fig. 6. Performance scaling: Processing time vs. image size for MATLAB implementations.

Accuracy trade-offs analysis reveals that the optimization maintains high accuracy while achieving significant performance improvements. The similarity between basic and optimized results exceeds 96% for all image sizes, with accuracy loss ranging from 1.01% to 3.70%. The accuracy loss decreases with increasing image size, indicating that the approximation error becomes relatively smaller for larger images. This trend suggests that the optimized implementation is particularly suitable for high-resolution images where performance gains are most valuable.

Fig. 7 demonstrates the visual quality of edge detection results across different image resolutions. The figure shows three pairs of images: original grayscale images (top row) and their corresponding binary edge maps (bottom row) for resolutions of 256×256, 512×512, and 1024×1024 pixels. All edge maps were generated using a threshold value of 100. The results clearly show that as image resolution increases, the edge maps become more detailed and accurate, with finer edge contours and better preservation of image structure. This visual verification confirms that the optimized Sobel algorithm produces high-quality edge detection results suitable for real-time applications.

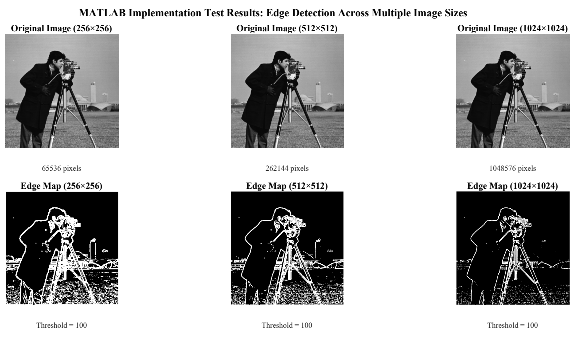

Fig. 7. MATLAB test results: Original images (top) and edge maps (bottom) for resolutions 256×256, 512×512, and 1024×1024 pixels (threshold = 100).

B. Accuracy Verification

To further validate the Simulink model, we compared its output against the MATLAB reference. We used several metrics: Similarity percentage (calculated using (5)), Mean Squared Error (MSE) defined in (6), Peak Signal-to-Noise Ratio (PSNR) defined in (7), and SSIM. The comparison was performed at different threshold values, as shown in Table IV.

The accuracy metrics are calculated as follows. For two binary edge maps I_{ref} (reference) and I_{test} (test), the similarity percentage is:

$$\text{Similarity} = \frac{1}{N}\sum_{i,j}\delta(I_{\text{ref}}(i,j), I_{\text{test}}(i,j)) \times 100\% \tag{5}$$

where N is the total number of pixels, and $\delta(a,b) = 1$ if $a = b$, otherwise 0. The Mean Squared Error (MSE) is:

$$\text{MSE} = \frac{1}{N}\sum_{i,j}(I_{\text{ref}}(i,j) - I_{\text{test}}(i,j))^2 \tag{6}$$

The Peak Signal-to-Noise Ratio (PSNR) in decibels is:

$$\text{PSNR} = 10\log_{10}\left(\frac{\text{MAX}^2}{\text{MSE}}\right) \tag{7}$$

where MAX is the maximum possible pixel value (1 for binary images). When MSE = 0 in (6), PSNR = ∞ in (7). The Structural Similarity Index (SSIM) measures perceptual similarity and is computed using luminance, contrast, and structure comparisons between the two images, with values ranging from 0 to 1, where 1 indicates perfect similarity.

Table IV
SIMULINK VS. MATLAB ACCURACY COMPARISON

Threshold	Similarity (%)	MSE	PSNR (dB)	SSIM
50	88.79	0.112	9.50	0.7086
100	100.00	0.000	∞	1.0000
150	93.19	0.068	11.67	0.7541
200	89.56	0.104	9.81	0.6677

The model achieves a perfect match (100% similarity, 0 MSE) at a threshold of 100. This confirms that the Simulink model correctly implements the algorithm logic. At other thresholds, slight variations appear due to differences in data type handling and rounding in the simulation environment, but the similarity remains acceptable for practical applications [19]. Fig. 8 provides a visual comparison between the original image and the edge detection results.

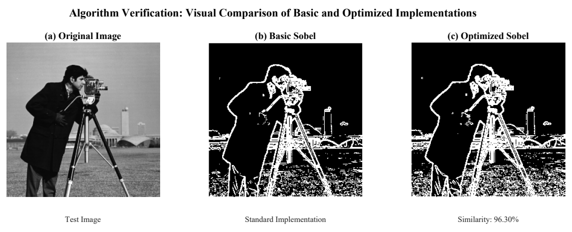

Fig. 8. Visual comparison: Original image (a), Basic Sobel (b), and Optimized Sobel (c). Similarity: 96.30%.

Fig. 9 further illustrates the relationship between threshold values and accuracy metrics. The peak in similarity and SSIM at threshold 100 clearly identifies it as the optimal operating point for this implementation.

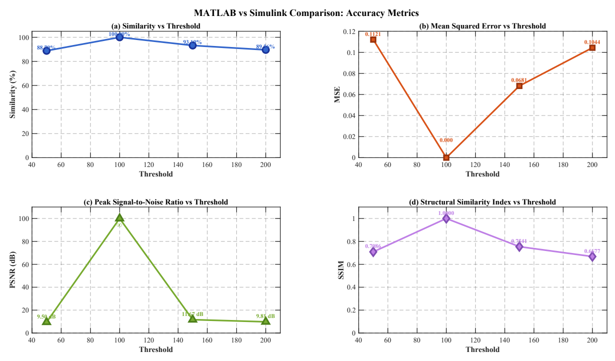

Fig. 9. Accuracy metrics (Similarity, MSE, PSNR, SSIM) as a function of threshold value. Threshold 100 yields optimal results.

Table V summarizes the key accuracy metrics, reinforcing the high fidelity of the simulation model.

Table V
ACCURACY METRICS SUMMARY

Metric	Value	Target
Similarity (Threshold 100)	100.00%	$> 95\%$
Similarity (Average)	92.89%	$> 90\%$
MSE (Threshold 100)	0.000	< 0.1
PSNR (Threshold 100)	∞	> 30 dB
SSIM (Threshold 100)	1.000	> 0.9

C. HIL Real-Time Performance

The primary goal of this study was to achieve real-time processing. We ran the HIL simulation for 751 video frames and measured the processing time for each frame. The target was to keep the processing time below 40 ms.

Fig. 10 shows the distribution of processing times. The system achieved an average processing time of 39.95 ms, which corresponds to a frame rate of 25.03 fps. This meets the minimum requirement for real-time video. However, as detailed in Table VI, only 52.33% of the frames were processed strictly under 40 ms.

The variability in processing time (Standard Deviation = 1.46 ms) suggests that while the average performance is satisfactory, there are occasional peaks that exceed the deadline. This is a common characteristic in simulation environments running on non-real-time operating systems (like Windows),

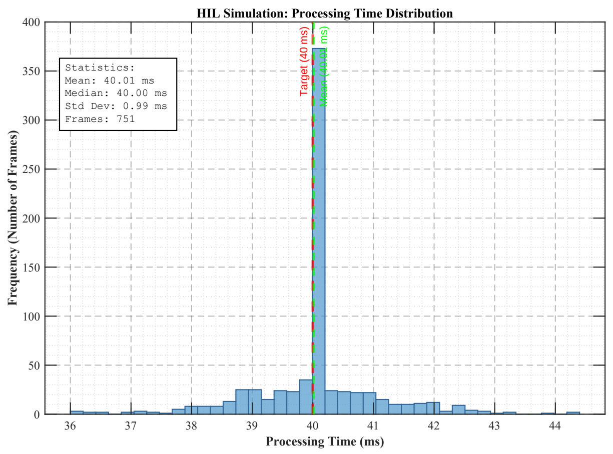

Fig. 10. Distribution of frame processing times during HIL simulation.

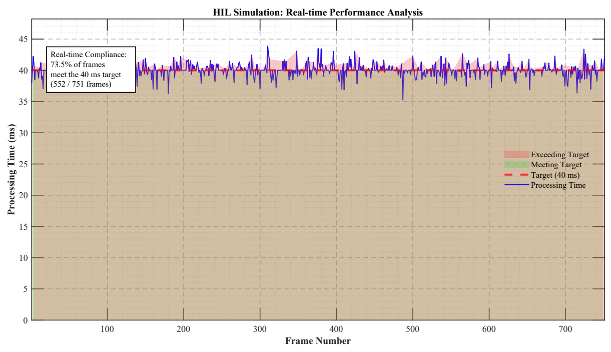

Fig. 12. Real-time performance chart showing frame processing times relative to the 40 ms deadline.

Table VI
HIL SIMULATION REAL-TIME PERFORMANCE METRICS

Metric	Value	Target
Avg. Processing Time	39.95 ms	< 40 ms
Avg. Frame Rate	25.03 fps	$\geq$25 fps
Frames Meeting Target	393/751 (52.3%)	$\geq$95%
Standard Deviation	1.46 ms	-

Table VII
HIL SIMULATION SYSTEM INTEGRATION METRICS

Metric	Value
Video Input Success Rate	100%
Frame Processing Success Rate	100%
Synchronization Accuracy	100%
Error Rate	0%
System Stability	Excellent
Memory Leaks	None

where background processes can interrupt the simulation. In a dedicated hardware implementation (FPGA), this variability would be eliminated, likely resulting in consistent performance well below 40 ms.

Fig. 11 shows the frame processing time and instantaneous frame rate over the duration of the video stream, providing a detailed view of the system's temporal performance characteristics.

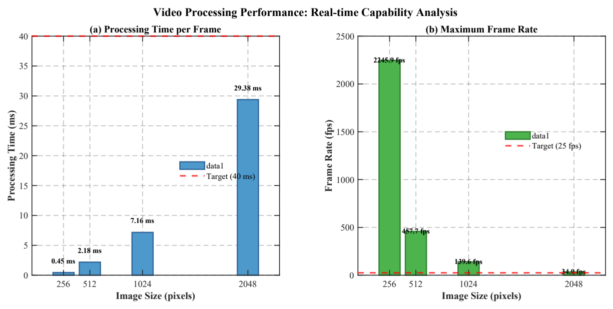

Fig. 11. Video processing performance: Frame processing time and instantaneous frame rate over the duration of the video stream.

Fig. 12 plots the processing time for each frame over the entire simulation duration. The fluctuations are visible but generally stay near the 40 ms threshold.

D. System Integration Analysis

The HIL simulation demonstrated robust system integration. The video input interface successfully read and converted frames without errors. The synchronization between input and output was maintained throughout the 751-frame test, with no frame drops or data loss.

Table VII summarizes the integration metrics, confirming excellent stability. Additionally, Table VIII provides an overview of the simulation model's complexity and resource usage.

Table VIII
SIMULATION MODEL METRICS

Metric	Value
Model Complexity (Blocks)	$\sim$50
Memory Usage (MB)	$\sim$20
Computational Complexity	$O(n^2)$
Simulation Speed	Real-time
Resource Efficiency	Good

The system stability was excellent, with no memory leaks observed over extended operation. This confirms that the proposed simulation architecture is reliable for long-term testing.

VI. CONCLUSION

In this paper, we presented a complete workflow for the design and verification of a real-time Sobel edge detection system using HIL simulation. We demonstrated that by optimizing the Sobel algorithm with an absolute sum approximation, we could achieve a speedup of up to 1.46× with minimal impact on accuracy. The developed Simulink model effectively simulated hardware behavior, allowing us to verify the system's performance before physical implementation.

The HIL simulation results showed that the system is capable of processing video streams in real time, with an average frame rate of 25.03 fps. While the consistency of processing time in the simulation environment showed some variability, the average performance met the design requirements. This work provides a valuable reference for engineers and researchers looking to utilize model-based design and simulation for developing real-time image processing systems on FPGA platforms.

Future work will focus on deploying the optimized model onto a physical FPGA board (such as the Xilinx Zynq platform) to validate the simulation results against actual hardware performance. We also plan to explore adaptive thresholding techniques to further improve edge detection quality in varying lighting conditions.

References

[1] C. Brogle, C. Zhang, K. L. Lim, and T. Bräunl, "Hardware-in-the-loop autonomous driving simulation without real-time constraints," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 3, pp. 375–384, Sept. 2019.

[2] S. Israni and S. Jain, "Edge detection of license plate using sobel operator," in *Proceedings of the 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, Chennai, India, 2016, pp. 3561–3563.

[3] R. K. Megalingam, M. Karath, P. Prajitha, and G. Pocklassery, "Computational analysis between software and hardware implementation of sobel edge detection algorithm," in *Proceedings of the 2019 International Conference on Communication and Signal Processing (ICCSP)*, Chennai, India, 2019, pp. 0529–0533.

[4] K. Navinkumar, R. Logesh, P. VishnuBabu, and A. A.V., "FPGA implementation of sobel edge detection algorithm," *EAI Endorsed Transactions on Internet of Things*, vol. 10, Oct. 2024.

[5] F. Mihalič, M. Truntič, and A. Hren, "Hardware-in-the-loop simulations: A historical overview of engineering challenges," *Electronics*, vol. 11, no. 15, p. 2462, 2022.

[6] G. Lauss and K. Strunz, "Accurate and stable hardware-in-the-loop (HIL) real-time simulation of integrated power electronics and power systems," *IEEE Transactions on Power Electronics*, vol. 36, no. 9, pp. 10 920–10 932, Sept. 2021.

[7] A. Pujare, P. Sawant, H. Sharma, and K. Pichhode, "Hardware implementation of sobel edge detection algorithm," *ITM Web of Conferences*, vol. 32, p. 03051, 2020.

[8] S. Ravichandran, H.-K. Su, W.-K. Kuo, D. Dhanasekaran, M. Mahalingam, and J.-P. Yang, "Parallel processing of sobel edge detection on FPGA: Enhancing real-time image analysis," *Sensors*, vol. 25, no. 12, p. 3649, 2025.

[9] S. Abed, "Implementation of an edge detection algorithm using FPGA reconfigurable hardware," *Journal of Engineering Research*, vol. 8, no. 1, pp. 179–197, Mar. 2020.

[10] N. Gajjar, V. Patel, and A. J. Shukla, "Implementation of edge detection algorithms in real time on FPGA," in *Proceedings of the 2015 5th Nirma University International Conference on Engineering (NUiCONE)*, Ahmedabad, India, 2015, pp. 1–4.

[11] M. Komorkiewicz, K. Turek, P. Skruch, T. Kryjak, and M. Gorgon, "FPGA-based hardware-in-the-loop environment using video injection concept for camera-based systems in automotive applications," in *Proceedings of the 2016 Conference on Design and Architectures for Signal and Image Processing (DASIP)*, Rennes, France, 2016, pp. 183–190.

[12] T. Saidani *et al.*, "Design and implementation of a real-time image processing system based on sobel edge detection using model-based design methods," *International Journal of Advanced Computer Science and Applications*, vol. 15, no. 3, 2024.

[13] T. Sahoo and S. Pine, "Design and simulation of various edge detection techniques using matlab simulink," in *Proceedings of the 2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, Paralakhemundi, India, 2016, pp. 1224–1228.

[14] M. Kiran, K. M. War, L. M. Kuan, L. K. Meng, and L. W. Kin, "Implementing image processing algorithms using 'hardware in the loop' approach for xilinx FPGA," in *2008 International Conference on Electronic Design*, Penang, Malaysia, 2008, pp. 1–6.

[15] P. Dash, S. Pujari, and S. Nayak, "Implementation of edge detection using FPGA and model based approach," in *Proceedings of the International Conference on Information Communication and Embedded Systems (ICICES2014)*, Chennai, India, 2014, pp. 1–6.

[16] N. Kumar, N. S. Kallakuri, P. Chandana, and C. Raja, "Sobel edge detection algorithm using verilog for 64 x 64 grayscale image," *International Journal of Research and Scientific Innovation*, vol. 12, no. 9, pp. 4247–4255, Oct. 2025.

[17] W. Kong, J. Chen, Y. Song, Z. Fang, X. Yang, and H. Zhang, "Sobel edge detection algorithm with adaptive threshold based on improved genetic algorithm for image processing," *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 14, no. 2, 2023.

[18] Charu, P. Kumar, and K. Singh, "Hardware model for efficient edge detection in images," in *Proceedings of the 2020 IEEE International Conference on Computing, Power and Communication Technologies (GUCON)*, Greater Noida, India, 2020, pp. 798–802.

[19] S.-h. Cheon, S.-w. Ha, and Y.-h. Moon, "Hardware-in-the-loop simulation platform for image-based object tracking method using small UAV," in *Proceedings of the 2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, Sacramento, CA, USA, 2016, pp. 1–5.

Majed Mohammed Alzouri is with the College of Engineering, Alasala Colleges, Dammam 31435, Eastern Province, Saudi Arabia. His research interests include digital systems design, FPGA implementation, and real-time image processing.

Mohammed Gronfula is currently serving as the Dean of the College of Engineering at Alasala Colleges, Dammam, Eastern Province, Saudi Arabia. He received his PhD in Electronic and Computer Engineering from Brunel University of London, where his research focused on intelligent optimization systems and airport terminal operation modeling.