International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 7 Issue 5, September-October 2023 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

An Efficient Hardware Implementation of Canny Edge Detection Algorithm

Keerthana P[1], Mr. K. Raja[2]

[1]PG Student, [2]Assistant Professor,
[1,2]Department of ECE, Gnanamani College of Technology, Namakkal, Tamil Nadu, India

ABSTRACT

Edge detection is an essential technique used in many image processing applications. Among the various edge detection techniques available, the canny edge detection algorithm has been widely recognized for its superior performance. However, its implementation in real-time systems can be computationally complex and expensive in terms of hardware costs, leading to increased latency. To address these challenges, a novel approach to canny edge detection has been proposed. This algorithm utilizes approximation methods to replace complex operations, thereby reducing computational complexity. In addition, pipelining techniques are employed to further decrease latency. The proposed canny edge detection algorithm has been implemented on Xilinx Virtex-5 FPGA, a field-programmable gate array. Compared to previous hardware architectures for canny edge detection, the new architecture requires fewer hardware resources, resulting in reduced costs. Furthermore, the algorithm is able to detect the edges of a 512 x 512 image in just 1ms. In conclusion, the proposed canny edge detection algorithm offers an efficient and cost-effective solution for real-time image processing applications. By utilizing approximation methods and pipelining techniques, it achieves superior performance while minimizing hardware costs and latency.

KEYWORDS: *dge detection, canny edge detection algorithm, real-time systems, computational complexity, hardware costs, latency, approximation methods, pipelining techniques, Xilinx Virtex-5 FPGA*

How to cite this paper: Keerthana P | Mr. K. Raja "An Efficient Hardware Implementation of Canny Edge Detection Algorithm" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-7 | Issue-5, October 2023, pp.309-314, URL: www.ijtsrd.com/papers/ijtsrd59863.pdf

IJTSRD59863

I. INTRODUCTION

Edge detection is an important step in computer vision and imaging applications, as it helps define object boundaries within an image. However, implementing an edge detection system in hardware poses challenges due to noise interference, lighting conditions, processing speed, and accuracy requirements. Various edge detection techniques have been proposed, with Canny edge detection offering superior performance by addressing the limitations of other detectors. However, it is computationally complex and results in high latency for real-time applications. Several FPGA-based implementations have been explored to achieve real-time performance, but they often suffer from performance degradation and increased latency. To reduce latency, a parallel implementation has been proposed that processes pixels concurrently, but this leads to increased memory access. Previous hardware implementations also use constant low and high thresholds, which simplifies the complexity but degrades performance. Recent efforts have tried to reduce computational complexity and latency by using absolute operations instead of square and square root operations, resulting in lower accuracy. The proper computation of thresholds with low latency is crucial for the canny edge detection algorithm's performance. A robust threshold computation method with higher accuracy has been introduced, but it requires more resources and increases latency.

II. SOFTWARE IMPLEMENTATION

MATLAB is a versatile tool used extensively in computational mathematics. It finds application in a wide range of mathematical calculations, including:

- Manipulating matrices and arrays

- Plotting and creating graphics in both 2D and 3D dimensions
- Solving linear algebra problems
- Solving algebraic equations
- Handling non-linear functions
- Performing statistical analyses
- Analyzing data
- Carrying out calculus and differential equations
- Conducting numerical calculations
- Performing integrations
- Applying transforms

III. FEATURES OF MATLAB

It is a powerful high-level language that is primarily used for numerical computation, visualization, and application development. Additionally, it offers an interactive environment that facilitates iterative exploration, problem-solving, and design.

MATLAB provides an extensive library of mathematical functions covering areas such as linear algebra, statistics, Fourier analysis, filtering, optimization, numerical integration, and solving ordinary differential equations. This library allows users to effortlessly perform complex mathematical operations.

The language also includes built-in graphics capabilities, enabling users to easily visualize data and create custom plots. Furthermore, MATLAB's programming interface offers development tools that enhance code quality, maintainability, and performance optimization.

For those looking to build applications with custom graphical interfaces, MATLAB provides convenient tools that simplify the process.

Moreover, MATLAB allows for seamless integration of MATLAB-based algorithms with external applications and programming languages, including C, Java, .NET, and Microsoft Excel. This feature allows users to leverage the power of MATLAB alongside their preferred software.

IV. MATLAB PRODUCT DESCRIPTION

MATLAB is a powerful platform that combines a high-level programming language with an interactive environment, making it ideal for numerical computation, visualization, and programming tasks. With MATLAB, you can easily analyze data, develop algorithms, and create models and applications. Its extensive library of built-in math functions and tools allows you to explore various approaches and find solutions more quickly compared to traditional programming languages or spreadsheets. MATLAB has a wide range of applications, including signal processing, image and video processing, control systems, test and measurement, computational finance, and computational biology. It is widely used by millions of engineers and scientists in both industry and academia as the go-to language for technical computing tasks.

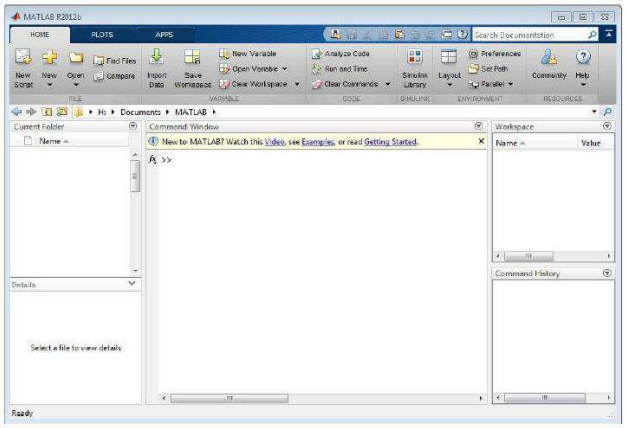

Figure I. Matlab Window

V. MATLAB INTRODUCTION

Matlab, short for Matrix Laboratory, is a programming language and numerical computing environment developed by MathWorks. It is designed for matrix manipulations, data plotting, algorithm implementation, and creation of user interfaces. Matlab can also interface with other programming languages such as C, C++, Java, and FORTRAN. While it is primarily used for numerical computation, it also has a symbolic computing toolbox called MuPAD for symbolic computations.

In addition, Matlab offers a package called Simulink, which provides graphical simulation and model-based design capabilities for dynamic and embedded systems. As of 2004, Matlab had over a million users in industry and academia, and it is widely used in fields such as engineering, science, and economics.

The first chapter of Matlab and numerical computing introduces the language by showcasing several programs that explore elementary mathematical problems. If you have prior programming experience, studying these programs will help you understand how Matlab functions. For a more comprehensive introduction, you can access the Help tab in the Matlab command window, which provides documentation and resources. The MathWorks website also offers tutorials, videos, and a PDF manual called "Getting Started with MATLAB" for beginners.

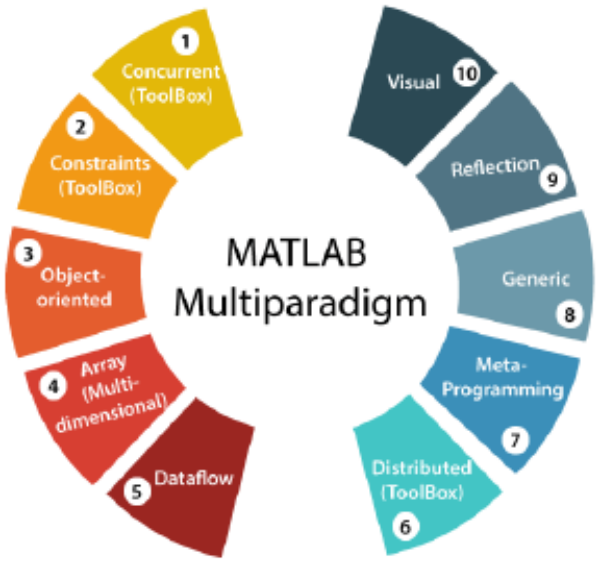

Figure II. MATLAB Multiparadigm

VI. THE MATLAB ENVIRONMENT

The MATLAB environment is set up like a typical word processor, with menus, buttons, and a writing area. The main writing area is called the command window, where you can input commands for MATLAB to execute. You can start a program by typing its name in the command window. The command window can also be used as a scientific calculator or a graphing tool. However, for longer programs, it is more convenient to write the program code in a separate window and then execute it in the command window. In the command window, you will see a prompt (>>) where you can type your commands. After typing a command, press Enter to execute it. If you need to interrupt a running command, you can do so by typing Ctrl + C.

MATLAB, short for "matrix laboratory," was created in the late 1970s by Cleve Moler, who was the chairman of the computer science department at the University of New Mexico at the time. Its development was motivated by Moler's desire to provide his students with access to LINPACK and EISPACK, without requiring them to learn Fortran. As a result, MATLAB gained popularity in universities and found a strong following among applied mathematicians.

In 1983, engineer Jack Little encountered MATLAB during a visit by Moler to Stanford University. Recognizing its potential for commercial use, Little joined forces with Moler and Steve Bangert. They rewrote MATLAB in the C programming language and founded The MathWorks in 1984 to further develop the software. This new version, known as JACKPAC, utilized different libraries for matrix manipulation, specifically LAPACK.

A. Variables

In MATLAB, the assignment operator, =, is used to define variables. This programming language is considered weakly typed due to its implicit type conversion. Additionally, it is dynamically typed, which means that variables can be assigned without having to declare their type, except when they are treated as symbolic objects. Furthermore, the type of a variable can change over time. Values can be derived from constants, computations involving other variables, or from the output of a function.

B. Vectors/matrices

MATLAB is capable of creating and manipulating arrays of various dimensions, including vectors and matrices. A vector in MATLAB is a one-dimensional matrix with dimensions of 1×N or N×1. Similarly, a matrix in MATLAB is a two-dimensional array with dimensions of m×n, where both m and n are greater than or equal to one.

VII. EDGE DETECTION

In MATLAB, arrays can be created and manipulated with various dimensions, including vectors and matrices. A vector is a one-dimensional matrix with dimensions of 1×N or N×1, while a matrix is a two-dimensional array with dimensions of m×n, where both m and n are greater than or equal to one.

Edges in image processing are characterized by their length, slope angle, and coordinate of the slope midpoint. They can be caused by various factors such as surface normal discontinuity, depth discontinuity, surface color discontinuity, and illumination discontinuity. There are two types of edges: ramp edges, where intensity values change slowly, and step edges or ideal edges, where intensity values change abruptly.

Edge detection can be achieved through several approaches, but they can be grouped into two categories: gradient-based (approximation of derivative) and Laplacian-based (zero-crossing detectors) approaches. Gradient-based edge detection operators determine the level of variance between neighboring pixels by forming a mask over the center pixel whose properties need to be altered. A pixel is classified as an edge if an estimated area of the matrix overcomes the specified threshold value. Examples of gradient-based edge detectors include Prewitt, Roberts, and Sobel operators. Laplacian operators are second-order derivative operators.

The Canny edge detection method is considered the optimal edge detection method.

$$|G| = |Gx| + |Gy| \quad (1)$$

Mathematically gradient can be computed with equation.

$$|G| = \sqrt{Gx^2 + Gy^2} \quad (2)$$

VIII. EDGE DETECTOR METHODS a. Sobel Operator

The Sobel operator, as described by Maini et al. [4], is a gradient-based method used for edge detection on a grey-leveled and optionally smoothed image. It utilizes a pair of 3x3 convolution kernels that are perpendicular to each other, with one filtering the image in the vertical direction (Gx) and the other in the horizontal direction (Gy). The kernels measure the gradient in their respective directions, which are then combined to produce the absolute magnitude and orientation of the gradient.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 3: A pair of convolution kernels used by Sobel edge detector. One kernel is the other rotated by 90°. (https://homepages. inf.ed.ac.uk/rbf/HIPR2/sobel.htm)

Sobel x direction Sobel y direction

Figure 4: The two separate gradient output from the pair of convolution kernels, which will be further combined to produce the absolute magnitude and the orientation. (https://youtu.be/uihBwtPIBxM)

IX. CANNY EDGE DETECTION ALGORITHM

The Canny edge detector is a widely used algorithm for detecting edges in images. It was developed based on a set of criteria proposed by Canny, which include low error rate, good localization, and single edge point response. The first step of the algorithm involves smoothing the image using a Gaussian filter, followed by computing the image gradient using multiple first derivative operators in different directions.

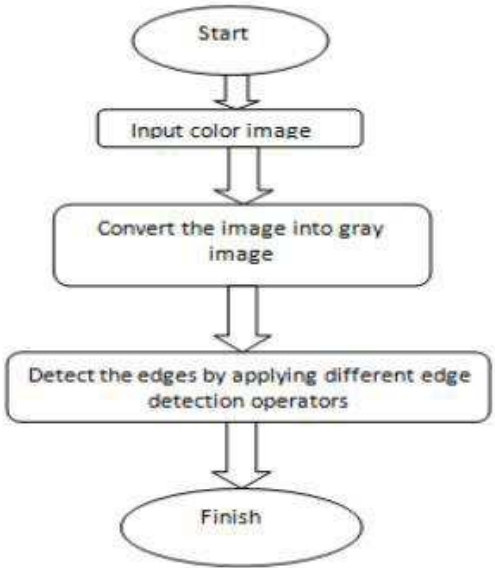

Figure III. Flow Chart of Edge Detection X. OUTPUT

unction edges = detectEdges(image, sigma, lowThreshold, highThreshold)

% This method implements the Canny edge detection algorithm.

% Inputs:
% - image: the input image

% - sigma: the standard deviation for the Gaussian filter

% - lowThreshold: the lower threshold value for hysteresis

% - highThreshold: the higher threshold value for hysteresis

% Output:
% - edges: the binary image of detected edges

% Convert the image to grayscale

if size(image, 3) == 3

image = rgb2gray(image);

end

% Apply Gaussian filter

filteredImage = imgaussfilt(image, sigma);

% Compute image gradient using first

derivative operators

gx = imfilter(filteredImage, [-1 0 1]);

gy = imfilter(filteredImage, [-1; 0; 1]);

% Compute gradient magnitude and orientation

gradientMagnitude = sqrt(gx.^2 + gy.^2);

gradientOrientation = atan2d(gy, gx);

% Perform non-maximum suppression to get rid of spurious responses

suppressedMagnitude = nonMaxSuppression(gradientMagnitude, gradientOrientation);

% Perform hysteresis thresholding to get final edges edges =

hysteresisThresholding(suppressedMagnitude, lowThreshold, highThreshold);

end

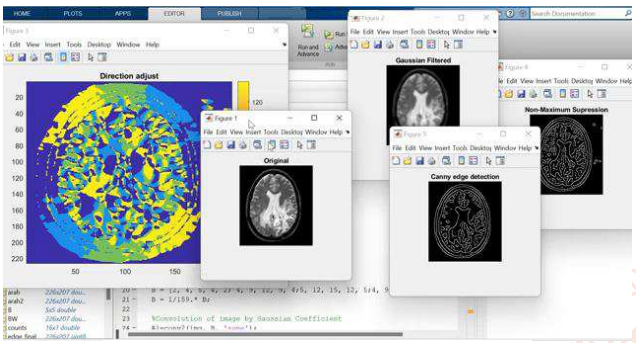

Figure IV. Candy Edge Detection

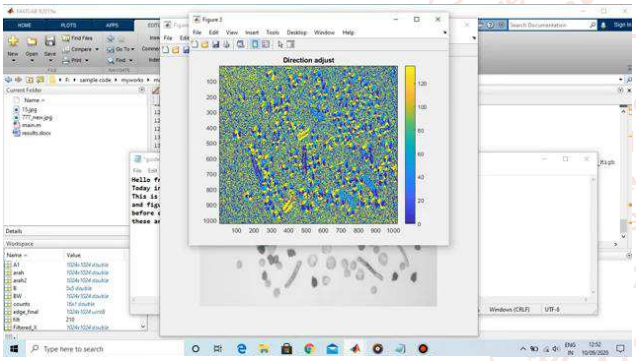

Figure V. Direction Adjust

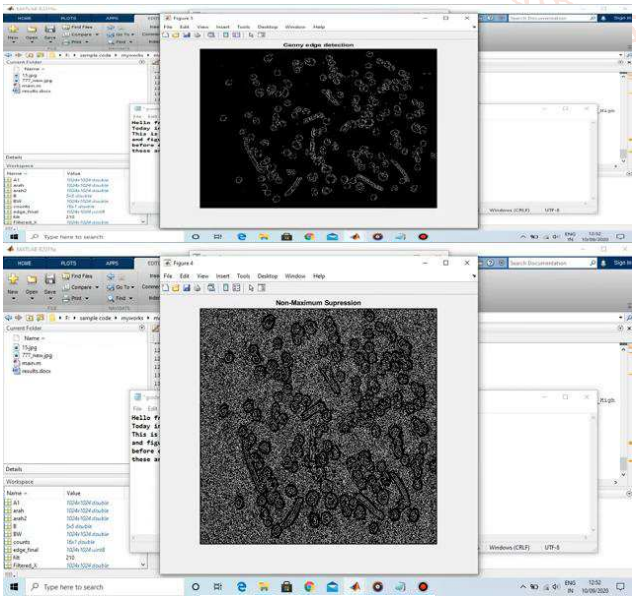

Figure VI. Output

X. CONCLUSION

In conclusion, our work focuses on implementing an image processing algorithm using Xilinx FPGA and Xilinx System Generator. The proposed block diagram emphasizes achieving high performance, low cost, and short development time. We prefer a reliable edge detection algorithm that produces efficient output even for noisy images. FPGA offers parallelism and in-built high-speed multipliers, which significantly reduce processing time. Although most hardware implementation techniques use a high-level language for coding, Xilinx System Generator simplifies the development process by allowing users to model the system. This tool eliminates the need for emulating floating-point algorithms in HDL code and allows for testing and verification through functional simulations, post-simulation processes, and generating bitstream files.

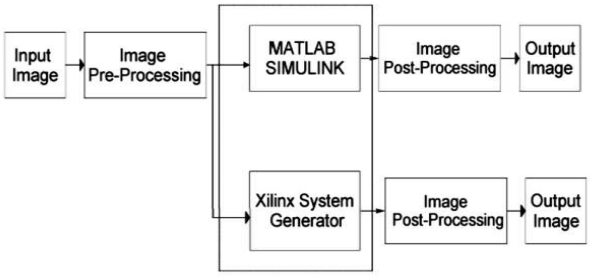

Figure VII. Proposed system

XI. REFERENCE

[1] L. G. Roberts. ―Machine perception of 3D solids,‖ Optical and ElectroOptical Information Processing. MIT Press, 1965.

[2] R. C. Gonzalez, R. E. Wood. ―Digital image processing,‖ Second Edition. Prentice Hall, 2002.

[3] N. Kanopouls, N. Vasanthavada, R.L.Baker, ―Design of an image edge detection filter using the sobel operator‖, IEEE journal of Solid-State Circuits, Apr.1988, pp.358-367.

[4] E. R. Davies. ―Constraints on the design of template masks for edge detection,‖ Pattern Recognition Lett., vol. 4, pp. 111-120, Apr,1986.

[5] J. F. Canny, ―A computation approach to edge detection,‖ IEEE Trans. Pattern Analysis and Machine Intelligence, vol. 8, no. 6, pp. 769-798, November 1986.

[6] Y. Luo and R. Duraiswami, ―Canny edge detection on nvidia cuda,‖ Computer Vision and Pattern Recognition Workshop, vol. 0, pp. 1–8, 2008.

[7] V. Rao and M. Venkatesan, ―An efficient reconfigurable architecture and implementation of edge detection algorithm using Handle-C, ‖ IEEE Conference on Information Technology: Coding and Computing (ITCC), vol. 2, pp. 843 – 847, Apr. 2004.

[8] H. Neoh, A. Hazanchuck, ―Adaptive edge detection for real-time video processing using FPGAs, ‖ Application notes, Altera Corporation, 2005.

[9] Gentsos, C. Sotiropoulou, S. Nikolaidis, N. Vassiliadis, ―Real-Time canny edge detection parallel implementation for FPGAs, ‖ IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 499-502, Dec. 2010.

[10] W. He and K. Yuan, ―An improved canny edge detector and its realization on FPGA, ‖ IEEE Proceedings of the 7th World Congress on Intelligent Control and Automation (WCICA), pp. 6561 –6564, June 2008.