# VLSI Architecture Design and Implementation of CANNY Edge Detection

Vana Sundari M[1],Rammohan T[2],Santhosh B K [3],Rajamoorthy P [4], Jayaprakash V [5]
[1.] Student, M.E., Applied Electronics [2] Professor[3,4,5] Assistant Professor, Department of Electronics and Communication Engineering, Jaya Engineering College, Chennai

**ABSTRACT**

In One of the most essential methods in digital image processing is edge detection. Because of its capacity to detect edges even in images heavily contaminated by noise, the Canny edge detector is the most widely used edge detection algorithm. A modified canny edge detector is created in MATLAB and implemented in FPGA in this project. The mask is used for gradient calculation, and bilinear interpolation of four pixels is used in non-maximal suppression. In the iris detection subsystem, this edge detector is used as a pre-processing stage. The goal of creating the hardware modules for the canny edge detector was to minimise its complexity, improve its performance, and make it appropriate for VLSI implementation on a reconfigurable FPGA-based platform.

***Keywords:*** *Edge Detection, Canny Edge detector, FPGA*

**INTRODUCTION**

In image processing, machine vision, and computer vision, edge detection is a critical technique, especially in the fields of feature recognition and extraction. Computer vision is a branch of artificial intelligence that aims to give computers the ability to observe, identify, and interpret images in the same way that humans do, and then produce the appropriate output in a fraction of a second. The detection of edges in an image is a crucial step in comprehending visual features. Edges are made up of crucial information and relevant features. It greatly decreases the image size and filters out information that may be deemed unimportant, keeping the image's crucial structural features. Edge detection refers to a set of mathematical techniques for recognising points in a digital image where the image brightness abruptly changes or, more formally, where there are discontinuities. The sharp fluctuations in image brightness are usually grouped into a collection of curved line segments called edges.

The term edge refers to the section of the image local area where the brightness of the image fluctuates considerably in digital image processing. The first-order derivative-gradient approach, Second-Order

Derivative, and Optimal Edge Detection are the most common edge detection methods. Many edge detection techniques are available, including the Robert detector, Gauss-Laplace detector, Prewitt detector, and Canny detector. The Canny edge detector has been a standard for many years and has the best performance of all the available edge detection algorithms. The main downside of employing the Canny edge detector is that it takes a long time to compute and is difficult to implement in order to achieve real-time response.

FPGAs, as an alternative to custom ICs, can be utilised to build a complete system on a single chip (SOC). The capacity to reprogram is the main benefit of FPGA. After the FPGA has been produced, the user can reprogram it to implement a design. As a result, the term "Field Programmable" was coined. With the help of CAD tools, FPGA are simple to implement in a short amount of time. FPGAs are one of the newest VLSI hotspots. Using MATLAB and FPGA, the suggested system created an efficient programmable system with hardware.

## PROPOSED SYSTEM

Figure 1 shows a block diagram of the proposed system. The system is developed in MATLAB with a canny edge detector that is somewhat modified from the original Canny algorithm, and an efficient architecture for the method is designed for hardware implementation. The algorithm is developed using a Hardware description language (here Verilog HDL), and the result is verified using Xilinx Design tools for simulation and synthesising. For edge detection and recognition, human iris is used as an input parameter.

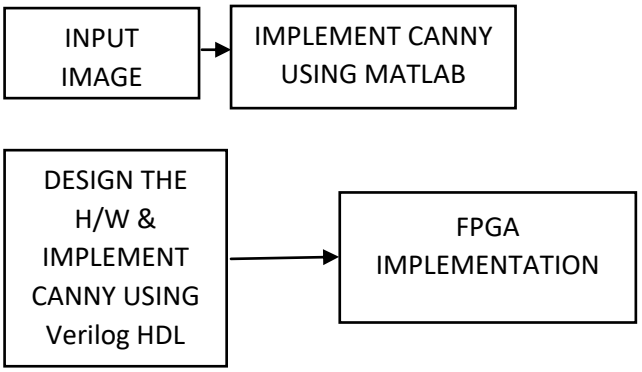



**Figure 1:** The flow diagram of proposed system

### The Canny Edge Detection Algorithm

The following are the steps in the Canny edge detection algorithm:

STEP 1

Smoothing using Gaussian filter: All images taken from a camera will contain some amount of noise, Since the Canny edge detector is susceptible to noise present in raw unprocessed image data the noise is to be removed. A filter based on a Gaussian (bell curve), where the raw image is convolved with a Gaussian filter is used to filter out the noise. The kernel of a 5X5 Gaussian filter with a standard deviation of = 1.4 is shown in Fig- 2.

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}.$$

**Figure 2:** Gaussian filter

STEP 2:

**Find the intensity gradient of the image:** The Canny's algorithm marks boundaries at maxima in the magnitude of the image gradient in the direction of the gradient. These areas are found by determining gradients of the image. The original Canny's code use the mask shown in Fig-3 to compute X and Y components of the gradient.

An edge in an image may point in a variety of directions, so new implementation uses the mask [-1 0 1] to compute first differences in four directions: H(horizontal), V(vertical), D1 and D2 (diagonal). The X and Y components of the gradients are computed by projecting the diagonal differences onto the axes are GX = H + D1+D2 /2 and $GY = V + D1 - D2/2$. The gradient magnitudes (also known as the edge strengths) can then be determined as a Euclidean distance measure by applying the law of Pythagoras equation

$$G = \sqrt{G_x^2 + G_y^2}..$$

The direction of the edges is determined and stored to use for further processing as shown in Equation below.

$$\theta = \arctan(\frac{G_y}{G_x})$$

$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix}$$

**Figure 3:** Mask to compute Gradients

STEP 3:

**Non-maximum suppression:** The objective of this method is to turn the "blurred" edges in the gradient magnitudes image into "crisp" edges. Basically, all local maxima in the gradient image are preserved, and everything else is deleted. Each pixel in the gradient image has its own algorithm. A search is then conducted using the image gradient estimates to see if the gradient magnitude assumes a local maximum in the gradient direction. The edge strengths are shown by colours and numbers, with arrows indicating the gradient directions. Almost every pixel in the image has a gradient direction that points north. As a result, they are compared to the pixels above and below them. White borders are drawn around the pixels that turn out to be the most in this comparison. The rest of the pixels will be hidden. White borders are drawn around the edge pixels as a result.

A 9-pixel neighbourhood is used in the original canny algorithm. The gradient (the normal to the edge direction) is depicted as an arrow, and it contains components (ux, uy). There are only discrete values of the gradient at locations Pi, j to non-maximize suppress the gradient amplitude in this direction. For non-maximum suppression, three points are required: one is P x, y, and the other two are estimates of the gradient magnitude at places displaced from P x, y by the vector u.

Consider the two locations in the 8-pixel neighbourhood of Px, y that are closest to the line passing through P x, y in direction u for any u. The magnitude of the gradient at these two places, as well as the gradient at the point P x, y, establish a plane that slices the gradient magnitude surface at these spots. The plane is used to estimate the value at a point on the line and to simulate the surface locally. For example, in Fig. -4, the value of a point on the line between P x, y+1 and Px+1, y+1 is assessed. The interpolated gradient's value is

$$G_1 = \frac{u_x}{u_y} G(x+1, y+1) + \frac{u_y - u_x}{u_y} G(x, y+1)$$

Similarly, the interpolated gradient at a point on the opposite side of Px, y is

$$G_2 = \frac{u_x}{u_y} G(x-1, y-1) + \frac{u_y - u_x}{u_y} G(x, y-1)$$

If G (x, y) > G1 and G (x, y) > G2, the point Px, y is identified as a maximum. For other gradient directions, the interpolation is similar, and it always involves one diagonal and one non-diagonal point. The divisions can be avoided in practise by multiplying by uy. This strategy uses five multiplications per point, which isn't excessive, and it outperforms a simpler scheme that compares the point Px, y with two of its neighbours. It also outperforms a strategy that uses an averaged value for the gradient along the edge instead of simply the value at Px, y.

The central pixel Px, y is compared in this study to two intensities G1 and G2, where G1 and G2 are the gradients derived by bilinear interpolation of four surrounding cells. The four cells are the four pixels closest to the point where the gradient orientation meets a radius 2 circle.

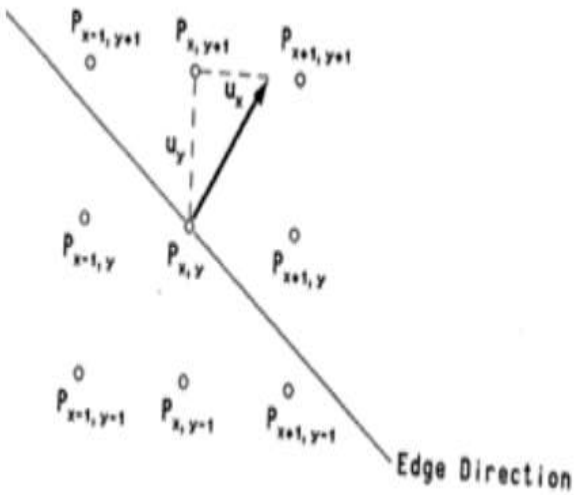



**Figure 4:** Support of the non-maximum suppression operator

STEP 4

**Bilinear interpolation:** Bilinear interpolation is a simple resampling

technique used in computer vision and image processing. In order to determine the right colour intensity values for a particular pixel, bilinear interpolation employs just the 4 nearest pixel values that are positioned in diagonal directions from that pixel.

Bilinear interpolation takes into account the closest 2x2 neighbourhood of known pixel values surrounding the estimated location of the unknown pixel. The final, interpolated value is calculated by taking a weighted average of these four pixels. The computed pixel's distance (in 2D space) from each of the known points determines the weight assigned to each of the four pixel values.

The following equations can be used to determine G1 the bilinear interpolation of four pixels tl, tr, bl, and br from Fig-5. The fractional parts of horizontal and vertical offset corresponding to the given orientation, i.e. fractional parts of xoff and yoff, are represented by hfrac and vfrac in Fig-5. Bilinear interpolation is used to calculate the upper and lower averages first.

Then, using bilinear interpolation and the upper and lower avgs , G1 may be calculated as

$$G_1 = (1 - v_{frac})\,upper\ avg + v_{frac}\,lower\ avg$$
$$= upper\ avg + (lower\ avg - upper\ avg)h_{frac}$$

The effect of non-maximal suppression on the test image is seen in Fig-4. A set of edge points in the form of a binary picture is obtained from this stage, referred known as non-maximum suppression. "Thin edges" is a term used to describe these.

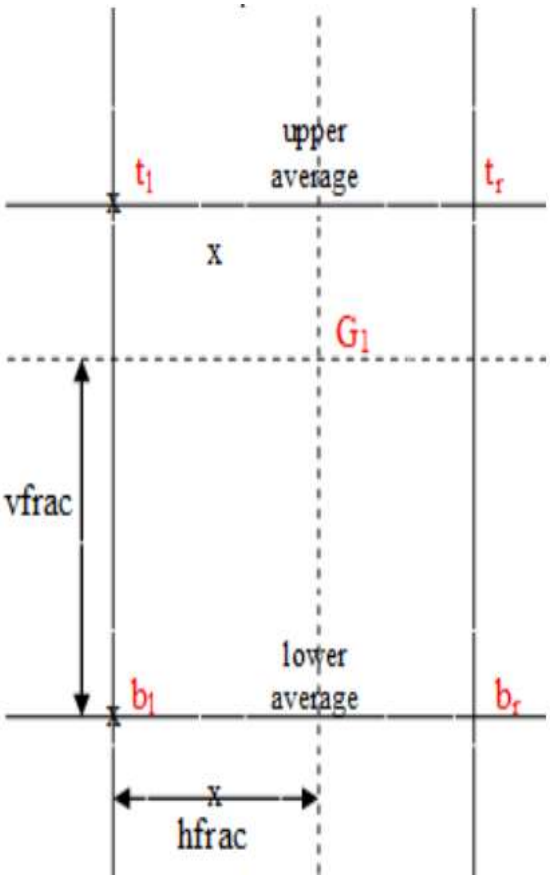



**Figure 5:** Bilinear interpolation

$$upper\ avg = (1 - h_{frac})t_l + h_{frac}t_r = t_l + (t_r - t_l)h_{frac}$$
$$lower\ avg = (1 - h_{frac})b_l + h_{frac}b_r$$
$$= b_l + (b_r - b_l)h_{frac}$$

STEP 5

**Edge tracking by hysteresis thresholding:** Edges are more likely to correlate to large intensity gradients than minor intensity gradients. In most circumstances, specifying a threshold at which a given intensity

gradient transitions from corresponding to an edge to not doing so is impossible. As a result, Canny employs hysteresis thresholding. The edge-pixels that remain after the non-maximum suppression stage are (still) pixel-by-pixel annotated with their strength. Many of them will almost certainly be actual image edges, but some may be created by noise or colour fluctuations, such as those caused by rough surfaces. The simplest way to distinguish between these would be to apply a threshold, with only the strongest edges preserving a specific value.

Double thresholding is used in the Canny edge detection technique. Edge pixels that are stronger than the high threshold are marked as strong; edge pixels that are weaker than the low threshold are suppressed; and edge pixels that are in the middle are labelled as weak.

Strong edges are considered as "certain edges" and can be incorporated into the final edge image right away. Weak edges are only included if and when they are linked to strong edges. The argument is, of course, that noise and other minor fluctuations are unlikely to result in a strong edge (if the threshold levels are properly adjusted). As a result, strong edges in the original image will nearly entirely be due to genuine edges. True edges or noise/color changes can cause the edges to be faint. The latter type will most likely be dispersed across the entire image independently of edges, with only a small proportion next to strong edges. Weak edges caused by genuine edges are far more likely to be coupled to strong edges directly.

The result of this operation is a binary image in which each pixel is labelled as either an edge pixel or a no edge pixel. The binary edge map acquired in this approach can also be considered as a set of edge curves, which can be represented as polygons in the picture domain following further processing using complimentary output from the edge tracing stage.

**MATLAB Implementation**

MATLAB is a computer programming language used in engineering, science, and applied mathematics. It comes with a robust programming language, stunning graphics, and a wealth of expert knowledge. The Math Works, Inc. publishes and trademarks MATLAB. With a little change in the gradient computation, the canny edge detection is implemented in MATLAB. Parallel computing is now available as an option in MATLAB. However, when it comes to high-performance computation, MATLAB is rarely the tool of choice.

The steps for implementation in MATLAB are

- Read the image

•Define different parameters: scaling, sigma, hi_thres, lo_thres, vert.horz,gamma,radius

- Function Gradient: Perform Gaussian filtering for noise removal and calculate magnitude and orientation of

intensity gradient at each pixel

- Function Adjgamma: Adjusts image gamma. Gamma values in the range 0-1 enhance contrast of bright regions, values > 1 enhance contrast in dark regions.

- Function Non max sup: Gradient magnitude is non maxima suppressed in the gradient direction

- Function Hysthresh: Function performs hysteresis thresholding of an image.

**Introduction to FPGA**

A two-dimensional array of logic blocks and linkages between logic blocks makes up an FPGA. The logic blocks as well as the links can be programmed. The interconnects are programmed using the switch boxes to connect the logic blocks, and the logic blocks are programmed to implement a desired function. All of the sub functions built in logic blocks must be connected in order to achieve our desired design, which is done through programming. One Lookup Table (LUT) and one Flip-flop make up a Xilinx logic block. A LUT can be used to implement a variety of functions.

**The Canny Edge Detection System Architecture**

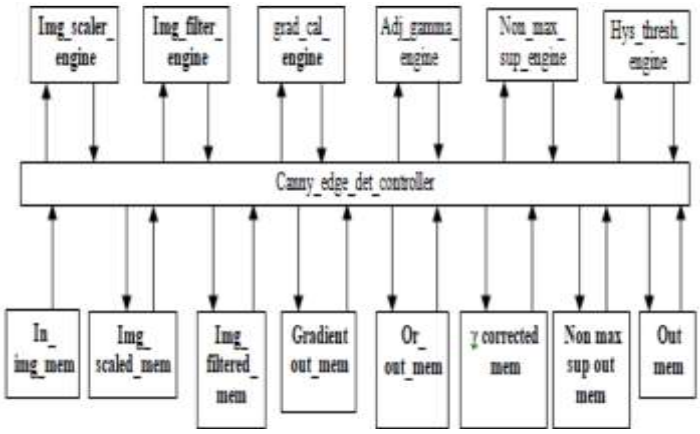
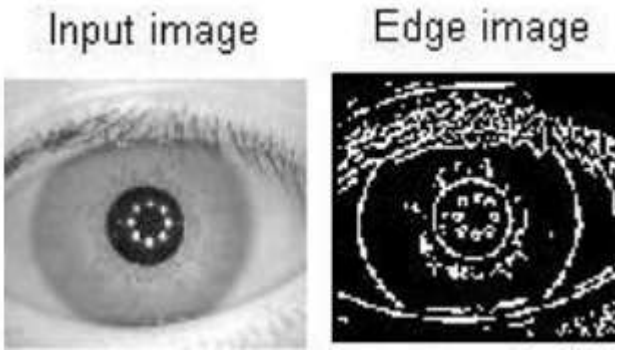


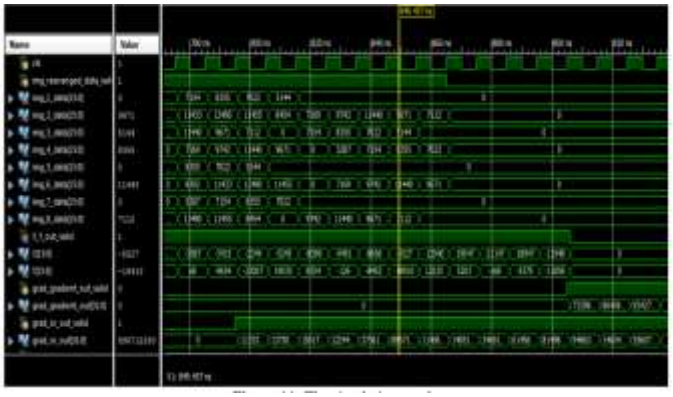

**Figure 6:** The system architecture of canny edge detector

The hardware architecture for the canny edge detection system is shown in Figure 6. It is made up of various processing blocks, input, output, and intermediate memories, as well as a controller that controls data movement between the various processing blocks and memories.

**Results and Discussions**

a) To locate the iris boundary, the edge map of original input eye image is created and output is obtained



**Implementation result and discussion**

In this case, the ISim Simulator application was utilised to replicate the design and test its functionality. The design was synthesised using the XILINX ISE Design Suite 14.7 tool once the functional verification was completed.

In order to test this modelled architecture, the proper test cases have been identified. The simulation results that explain the operation of the process have been achieved based on the identified values. This demonstrates that the modelled design is functioning appropriately.

**Simulation result**

The test bench was created to put the modelled design to the test. This built test bench will automatically force the inputs and perform the algorithm's operations.

**Conclusions**

The canny edge detection subsystem, which serves as a pre-processing portion for the iris detection system, is built in MATLAB for output verification with minor changes to both gradient mask and non-maximal suppression. This edge map is utilised to detect the inner and outer iris boundaries, as well as any iris eyelid boundaries. The implementation of VLSI is the next phase. The functionality is verified using XILINX ISE tools, and the programming is done in Verilog. This project uses a VLSI architecture that is efficient to develop and implement on a Xilinx FPGA. In terms of speed and hardware consumption, the FPGA system performs exceptionally well.

**References**

[1] E. Wolff. Anatomy of the Eye and Orbit. 7th edition. H. K. Lewis & Co. LTD, 1976

[2] J. F. Canny, "Finding edges and lines in images," M.S. thesis, Mass.Inst. Technol.; AI Lab. TR-720, 1983.

[3] John Canny. “A computational approach to edge detection”. Pattern Analysis and

Machine Intelligence, IEEE Transactions on, PAMI-8(6):679–698, Nov. 1986.

[4] L. Masek, "Recognition of Human Iris Patterns for Biometric identification", Thesis Report, The University of Western Australia, 2003.

[5] P. C. Kronfeld, “The gross anatomy and embryology of the eye,” in The Eye, vol. 1, H. Davson, Ed. London: Academic,1968, pp. 1–66.

[6] F. H. Adler, “Physiology of the Eye”. St. Louis, MO: Mosby, 1965.

[7] D. Ziou, S. Tabbone, “Edge Detection Techniques – An Overview”

[8] D. Marr and E. C. Hildreth. Theory of edge detection. Proceedings of the Royal Society, London B, 207:187- 217, 1980

[9] Jean Ponce, “Lecture 26: Edge Detection II”, 12/2/2004.

[10] S. Price, "Edges: The Canny Edge Detector", July 4, 1996.

[11] Margaret M Fleck “Some defects in finite difference edge finders” IEEE PAMI No. 3, Vol. 14. March 1992. pp 337-345

[12] Chinese Academy of Science - Institute of Automation (CASIA). Database of the Eye Grayscale Images.http:llwww.sinobiomtrics.com