

**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH**



**BÁO CÁO ĐỒ ÁN MÔN HỌC
THIẾT KẾ LOGIC (CO2039)**

ĐỀ TÀI:

**IMAGE EDGE DETECTION
USING SOBEL OPERATOR**

HỌC KỲ 1 NĂM HỌC 2025 - 2026

**GVHD: TS. HUỲNH PHÚC NGHỊ
LỚP: L03 - NHÓM 08**

DANH SÁCH SINH VIÊN THỰC HIỆN

STT	HỌ VÀ TÊN	MSSV
1	Phạm Tuấn Hải	2310880
2	Võ Nguyễn Nhật Duy	2310524
3	Vũ Thành Đạt	2310718

TP. Hồ Chí Minh, Tháng 12/2025



TÓM TẮT ĐỒ ÁN

1. Đặt vấn đề và Ý nghĩa

Trong bối cảnh công nghiệp 4.0, xử lý ảnh tốc độ cao là yêu cầu thiết yếu cho các hệ thống tự động hóa và an ninh giám sát. Các thuật toán phức tạp như Sobel đòi hỏi khối lượng tính toán lớn, khiến việc thực thi trên CPU thông thường gặp hạn chế về tốc độ trong ứng dụng thời gian thực. Đề tài được thực hiện nhằm kiểm chứng khả năng chuyển đổi thuật toán Sobel sang phần cứng FPGA, tạo tiền đề cho các hệ thống xử lý ảnh tốc độ cao.

2. Sơ lược kiến thức liên quan

Đồ án tập trung vào Toán tử Sobel, sử dụng phép nhân chập ma trận 3×3 để ước lượng đạo hàm theo phương ngang (G_x) và phương dọc (G_y). Độ lớn Gradient được tính bằng công thức xấp xỉ Manhattan $|G_x| + |G_y|$ để tối ưu hóa tài nguyên phần cứng (tránh phép căn bậc hai phức tạp).

3. Giải pháp đề xuất và Hiện thực

Giải pháp Lựa chọn: Kiến trúc **Bộ đệm dòng** (Line Buffer).

Ưu điểm: Thay vì lưu toàn bộ ảnh 512×512 (tốn khoảng 262 KB BRAM), hệ thống chỉ lưu 2 dòng gần nhất, giảm thiểu tài nguyên xuống còn khoảng 1 KB. Giải pháp này đảm bảo tính khả thi trên các dòng FPGA tài nguyên thấp và cho phép xử lý dữ liệu theo dạng đường ống (Pipelining).

Hiện thực: Mã nguồn Verilog RTL, sử dụng Icarus Verilog để mô phỏng và Python để kiểm chứng Bit-True.

4. Kết quả đạt được và Hướng mở rộng

Kết quả:

- Đã hiện thực thành công kiến trúc Line Buffer, giải quyết triệt để vấn đề tính toán số học có dấu (Signed Arithmetic).
- Kết quả mô phỏng Line Buffer khớp **100%** với mô hình tham chiếu Python.
- Xử lý thành công độ trễ đường ống (Pipeline Latency).

Chưa đạt được: Đồ án dừng ở mức mô phỏng, chưa kết nối Camera thực tế và sử dụng ngưỡng cố định.

Hướng mở rộng: Triển khai lên kit FPGA (DE10-Nano, Zybo Z7), tích hợp Camera/VGA và nâng cấp lên thuật toán ngưỡng thích nghi (Adaptive Thresholding) bằng Otsu.



ABSTRACT

1. Problem Statement and Significance

In the era of Industry 4.0, high-speed image processing is critical for automation and surveillance systems. Algorithms like Sobel require significant computation, limiting their speed on general-purpose CPUs for real-time applications. This project validates the feasibility of implementing the Sobel operator on FPGA hardware, aiming for high-speed image processing systems.

2. Background Knowledge

The project focuses on the Sobel Operator, using a 3x3 convolution kernel to estimate the image gradient in horizontal (G_x) and vertical (G_y) directions. The gradient magnitude is approximated using the Manhattan distance ($|G_x| + |G_y|$), avoiding complex square root operations to conserve hardware resources.

3. Proposed Solution and Implementation

Selected Solution: The Line Buffer Architecture.

Advantage: This architecture drastically reduces BRAM usage from approximately 262 KB (for full-frame storage) to approximately 1 KB (storing two adjacent lines). This low resource footprint makes the design suitable for low-cost FPGAs and enables efficient data Pipelining.

Implementation: The system is designed using Verilog RTL, simulated with Icarus Verilog, and verified against a Bit-True Golden Model in Python.

4. Key Results and Future Work

Achieved Results:

- Successful implementation of the Line Buffer architecture, including the correct handling of Signed Arithmetic.
- Verified **100%** functional match between the Verilog simulation output and the Python Golden Model.
- Accurate management of the inherent Pipeline Latency of the design.

Limitations: The project is currently simulation-based (no real camera interface) and uses a fixed threshold value.

Future Work: Implementation on target FPGA hardware (e.g., DE10-Nano), integration with Camera/VGA interfaces, and development of an adaptive thresholding algorithm (Otsu).



Mục lục

Tóm tắt đồ án	1
Abstract	2
1 TỔNG QUAN	1
1.1 Đặt vấn đề	1
1.2 Mục tiêu của đề tài	1
1.3 Đối tượng và Phạm vi nghiên cứu	2
1.3.1 Đối tượng nghiên cứu	2
1.3.2 Phạm vi nghiên cứu	2
1.4 Phương pháp nghiên cứu	2
1.4.1 Phương pháp nghiên cứu lý thuyết	2
1.4.2 Phương pháp nghiên cứu thực nghiệm	2
1.5 Cấu trúc đồ án	3
2 CƠ SỞ LÝ THUYẾT	4
2.1 Tổng quan về Xử lý ảnh số	4
2.2 Khái niệm về Biên (Edge)	4
2.3 Toán tử Sobel (Sobel Operator)	4
2.3.1 Các mặt nạ nhân chập (Kernels)	5
2.3.2 Quy trình tính toán	5
2.3.3 Tính độ lớn Gradient (Gradient Magnitude)	5
2.3.4 Phân ngưỡng (Thresholding)	6
3 THIẾT KẾ HỆ THỐNG	7
3.1 Tổng quan hệ thống	7
3.2 Thiết kế kiến trúc phần cứng (Hardware Architecture)	7
3.2.1 Sơ đồ khối chức năng	7
3.2.2 Tham số cấu hình	8
3.2.3 Tổ chức bộ nhớ (Line Buffer)	8
3.3 Lưu đồ hoạt động	9
3.4 Chi tiết hiện thực thuật toán	10
3.4.1 Cập nhật Cửa sổ trượt (Sliding Window)	10
3.4.2 Tính toán Số học (Signed Arithmetic)	10
3.4.3 Phân ngưỡng đầu ra	10
4 KẾT QUẢ VÀ THẢO LUẬN	11
4.1 Môi trường thực nghiệm	11
4.2 Đánh giá hiệu năng thuật toán trên phần mềm	11
4.2.1 Kết quả đo lường	11
4.2.2 Thảo luận kết quả	11
4.3 Kết quả mô phỏng hệ thống phần cứng	12
4.3.1 Phân tích đặc tính trễ (Pipeline Latency)	12
4.3.2 So sánh hình ảnh kết quả	12

4.3.3	Đánh giá độ chính xác	13
5	KẾT LUẬN VÀ KIẾN NGHỊ	14
5.1	Tóm tắt kết quả đạt được	14
5.2	Hạn chế của đề tài	14
5.3	Hướng phát triển (Future Work)	15
5.3.1	Triển khai trên phần cứng thực tế (Hardware Implementation)	15
5.3.2	Nâng cấp thuật toán thích nghi	15
5.3.3	Mở rộng sang bộ lọc Canny	15
	Tài liệu tham khảo	16



Chương 1

TỔNG QUAN

1.1 Đặt vấn đề

Trong kỷ nguyên công nghiệp 4.0, Xử lý ảnh số (Digital Image Processing) đã trở thành một thành phần không thể thiếu trong các hệ thống tự động hóa, robot, y tế và an ninh giám sát. Theo định nghĩa kinh điển của Gonzalez và Woods, xử lý ảnh số là việc sử dụng các thuật toán máy tính để thực hiện các thao tác trên hình ảnh kỹ thuật số nhằm cải thiện chất lượng hình ảnh hoặc trích xuất các thông tin mô tả hữu ích từ ảnh đó [1]¹.

Trong chuỗi quy trình xử lý ảnh, bài toán **Phát hiện biên (Edge Detection)** đóng vai trò nền tảng quan trọng nhất. Biên (Edge) là ranh giới giữa hai vùng có mức xám khác nhau rõ rệt, chứa đựng thông tin về cấu trúc và hình dáng của đối tượng. Việc phát hiện biên chính xác giúp giảm đáng kể khối lượng dữ liệu cần xử lý trong khi vẫn bảo toàn được các đặc trưng hình học thiết yếu [1]².

Tuy nhiên, các thuật toán xử lý ảnh thường đòi hỏi khối lượng tính toán lớn (đặc biệt là phép nhân chập trên ma trận). Việc thực thi trên vi xử lý thông thường (CPU) đôi khi gặp hạn chế về tốc độ trong các ứng dụng thời gian thực. Do đó, việc nghiên cứu hiện thực hóa các thuật toán này trên phần cứng chuyên dụng (như FPGA/ASIC) bằng ngôn ngữ mô tả phần cứng (Verilog HDL) là một hướng đi cấp thiết để tối ưu hóa hiệu năng hệ thống.

Xuất phát từ nhu cầu thực tiễn đó, đồ án "**Thiết kế và mô phỏng hệ thống phát hiện biên sử dụng thuật toán Sobel trên nền tảng Verilog**" được thực hiện nhằm kiểm chứng khả năng chuyển đổi từ thuật toán phần mềm sang phần cứng.

1.2 Mục tiêu của đề tài

Đồ án tập trung vào các mục tiêu chính sau:

- Nghiên cứu lý thuyết:** Tìm hiểu sâu về cấu trúc ảnh số, bản chất toán học của phép lọc không gian (Spatial Filtering) và các toán tử phát hiện biên cơ bản như Sobel và Canny dựa trên tài liệu chuẩn của Gonzalez & Woods [1]³.
- Thiết kế hệ thống:** Xây dựng mã nguồn Verilog mô tả hành vi (Behavioral Model) của bộ lọc Sobel, bao gồm các khối: Đọc dữ liệu, Tính nhân chập 3×3 , Tính độ lớn Gradient và Phân luồng nhị phân.
- Kiểm chứng mô phỏng:** Xây dựng môi trường Testbench kết hợp với Python để nạp dữ liệu ảnh thật vào mô hình phần cứng và đánh giá độ chính xác của kết quả đầu ra so với mô hình tham chiếu (Golden Model) trên phần mềm.

¹Định nghĩa gốc xem tại Chapter 1, Section 1.1 "What Is Digital Image Processing?", trang 2 của tài liệu [1].

²Vai trò của biên trong phân đoạn ảnh được thảo luận tại phần mở đầu Chapter 10 "Image Segmentation", trang 699-702 tài liệu [1].

³Lý thuyết lọc không gian tham khảo Chapter 3 (trang 153) và các toán tử biên tham khảo Chapter 10 (trang 712) [1].



1.3 Đối tượng và Phạm vi nghiên cứu

1.3.1 Đối tượng nghiên cứu

- Thuật toán phát hiện biên Sobel (Sobel Edge Detector).
- Ngôn ngữ mô tả phần cứng Verilog HDL.
- Quy trình mô phỏng và kiểm tra thiết kế số (Simulation Verification).

1.3.2 Phạm vi nghiên cứu

- Dữ liệu đầu vào:** Ảnh tĩnh định dạng mức xám (Grayscale 8-bit), kích thước chuẩn hóa 512×512 pixels.
- Phương pháp hiện thực:** Thiết kế phần cứng ở mức Register Transfer Level (RTL) hoặc Behavioral Level, chưa đi sâu vào tối ưu hóa tài nguyên vật lý (Place & Route) trên chip FPGA thực tế.
- Công cụ thực hiện:** Icarus Verilog (Mô phỏng phần cứng), Python (Tiền xử lý và Hậu xử lý ảnh).

1.4 Phương pháp nghiên cứu

Để đạt được các mục tiêu đã đề ra, đề tài áp dụng phối hợp hai phương pháp nghiên cứu chính:

1.4.1 Phương pháp nghiên cứu lý thuyết

- Thu thập và phân tích tài liệu: Nghiên cứu các giáo trình chuẩn về xử lý ảnh số (trọng tâm là tài liệu của Gonzalez & Woods [1]), các bài báo khoa học liên quan đến thuật toán phát hiện biên và ngôn ngữ Verilog.
- Phân tích so sánh: Đánh giá ưu nhược điểm của các toán tử Gradient (Sobel, Prewitt, Canny) để làm rõ lý do lựa chọn Sobel cho hiện thực phần cứng.

1.4.2 Phương pháp nghiên cứu thực nghiệm

- Mô hình hóa (Modeling):** Xây dựng mô hình thuật toán trên Python để kiểm chứng logic toán học và tạo bộ dữ liệu chuẩn (Golden Data).
- Mô phỏng (Simulation):** Sử dụng phần mềm Icarus Verilog để chạy mô phỏng mã nguồn phần cứng, quan sát dạng sóng tín hiệu và kiểm tra dữ liệu đầu ra.
- Kiểm chứng (Verification):** So sánh kết quả ảnh đầu ra từ Verilog với kết quả từ Python. Nếu sai số nằm trong phạm vi cho phép (do làm tròn số), thiết kế được coi là đạt yêu cầu.



1.5 Cấu trúc đồ án

Báo cáo được trình bày gồm 5 chương:

- **Chương 1: Tổng quan.** Trình bày lý do chọn đề tài, mục tiêu và phạm vi nghiên cứu.
- **Chương 2: Cơ sở lý thuyết.** Hệ thống hóa các kiến thức về ảnh số và thuật toán Sobel/Canny dựa trên giáo trình chuẩn.
- **Chương 3: Thiết kế hệ thống.** Mô tả chi tiết kiến trúc phần cứng và lưu đồ giải thuật của mã nguồn Verilog.
- **Chương 4: Kết quả và Thảo luận.** Trình bày kết quả mô phỏng, so sánh hiệu năng giữa mô hình phần cứng và phần mềm.
- **Chương 5: Kết luận và Kiến nghị.** Tóm tắt kết quả đạt được và đề xuất hướng phát triển.



Chương 2

CƠ SỞ LÝ THUYẾT

2.1 Tổng quan về Xử lý ảnh số

Ảnh số (Digital Image) là một biểu diễn rời rạc của dữ liệu hình ảnh không gian. Về mặt toán học, một ảnh số có thể được xem như một hàm hai chiều $f(x,y)$, trong đó x và y là tọa độ không gian, và giá trị f tại mỗi điểm tọa độ biểu thị cường độ sáng (intensity) hoặc màu sắc của điểm ảnh đó (pixel).

Trong biểu diễn máy tính, ảnh số thường được lượng tử hóa thành một ma trận kích thước $M \times N$. Khi tọa độ (x,y) và giá trị biên độ f đều là các số hữu hạn và rời rạc, ta gọi đó là ảnh số [1]¹.

Trong phạm vi đề tài này, hệ thống tập trung xử lý **ảnh mức xám (Grayscale Image)** 8-bit. Mỗi điểm ảnh được biểu diễn bằng một số nguyên nằm trong khoảng từ 0 đến 255:

- **Giá trị 0:** Đại diện cho màu đen tuyệt đối (cường độ thấp nhất).
- **Giá trị 255:** Đại diện cho màu trắng tuyệt đối (cường độ cao nhất).
- **Giá trị trung gian (1 ... 254):** Đại diện cho các mức độ xám tăng dần từ tối đến sáng.

2.2 Khái niệm về Biên (Edge)

Biên (Edge) là ranh giới giữa hai vùng ảnh có đặc tính độ sáng khác nhau rõ rệt. Về bản chất, biên tương ứng với các điểm ảnh có sự thay đổi đột ngột (discontinuity) về cường độ sáng.

Trong xử lý ảnh, việc phát hiện biên thường dựa trên việc tính toán **Đạo hàm (Gradient)** của hàm cường độ sáng. Tại vị trí có biên, đạo hàm bậc nhất của ảnh sẽ đạt giá trị cực đại (local maximum) và đạo hàm bậc hai sẽ có điểm cắt không (zero-crossing) [1]².

2.3 Toán tử Sobel (Sobel Operator)

Toán tử Sobel là một trong những phương pháp phát hiện biên phổ biến nhất trong các hệ thống phần cứng nhờ sự đơn giản và hiệu quả tính toán. Nó sử dụng hai mặt nạ nhân chập (convolution kernels) kích thước 3×3 để ước lượng đạo hàm của ảnh theo hai phương vuông góc: phương ngang (x) và phương dọc (y).

Khác với các toán tử gradient cơ bản (như Roberts hay Prewitt), Sobel sử dụng trọng số 2 ở vị trí trung tâm để tăng cường khả năng làm mượt nhiễu (smoothing effect) trước khi tính đạo hàm [1]³.

¹Định nghĩa chi tiết về hàm ảnh số $f(x,y)$ xem tại Chapter 2, Section 2.1 "Elements of Visual Perception" và biểu diễn ma trận tại Section 2.4, trang 56-58 tài liệu [1].

²Cơ sở lý thuyết về phát hiện sự thay đổi cường độ bằng đạo hàm được trình bày chi tiết tại Chapter 10, Section 10.2, trang 702 [1].

³Ưu điểm về khả năng kháng nhiễu của Sobel được thảo luận tại trang 713, Hình 10.14 [1].



2.3.1 Các mặt nạ nhân chập (Kernels)

Hai ma trận hạt nhân G_x và G_y được định nghĩa như sau:

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \text{và} \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \quad (2.1)$$

Trong đó:

- G_x : Tính đạo hàm theo phương ngang, nhạy cảm với các cạnh dọc.
- G_y : Tính đạo hàm theo phương dọc, nhạy cảm với các cạnh ngang.

2.3.2 Quy trình tính toán

Giả sử A là ma trận ảnh đầu vào. Tại mỗi điểm ảnh $P(i, j)$, xét một cửa sổ lân cận 3×3 với các điểm ảnh được đánh số như sau:

$$\text{Window} = \begin{bmatrix} P_1 & P_2 & P_3 \\ P_4 & P_5 & P_6 \\ P_7 & P_8 & P_9 \end{bmatrix}$$

Giá trị Gradient thành phần tại điểm trung tâm P_5 được tính bằng tổng các tích chập tương ứng:

$$g_x = (P_3 + 2P_6 + P_9) - (P_1 + 2P_4 + P_7) \quad (2.2)$$

$$g_y = (P_7 + 2P_8 + P_9) - (P_1 + 2P_2 + P_3) \quad (2.3)$$

2.3.3 Tính độ lớn Gradient (Gradient Magnitude)

Sau khi tính được hai thành phần g_x và g_y , độ lớn của vector gradient M tại mỗi điểm ảnh theo lý thuyết được tính bằng công thức Euclidean:

$$M = \sqrt{g_x^2 + g_y^2} \quad (2.4)$$

Tuy nhiên, việc thực hiện phép tính căn bậc hai (Square Root) trên phần cứng số (FPGA) tiêu tốn rất nhiều tài nguyên logic và gây ra độ trễ lớn. Do đó, để tối ưu hóa hiệu năng thiết kế, đồ án sử dụng công thức xấp xỉ Manhattan theo đề xuất của Gonzalez [1]:

$$M \approx |g_x| + |g_y| \quad (2.5)$$

Công thức này giúp đơn giản hóa mạch tính toán chỉ còn lại các bộ cộng (Adder) và bộ chuyển đổi giá trị tuyệt đối, rất phù hợp với kiến trúc của FPGA.

³Công thức xấp xỉ Manhattan (Equation 10-13) được đề xuất tại trang 714 giúp giảm chi phí tính toán phần cứng mà vẫn giữ được các đặc tính biên quan trọng [1].



2.3.4 Phân ngưỡng (Thresholding)

Bước cuối cùng là chuyển đổi ảnh Gradient thành ảnh nhị phân (Binary Image) để xác định rõ đâu là biên. Phương pháp phân ngưỡng toàn cục (Global Thresholding) được áp dụng với ngưỡng T :

$$\text{Pixel}_{out} = \begin{cases} 255 & (\text{Biên - Trắng}) \quad \text{nếu } M \geq T \\ 0 & (\text{Nền - Đen}) \quad \text{nếu } M < T \end{cases} \quad (2.6)$$

Trong thiết kế Verilog của đồ án, giá trị ngưỡng được thiết lập là **180**. Giá trị này được lựa chọn dựa trên thực nghiệm với bộ dữ liệu ảnh mẫu để loại bỏ nhiễu nền tần số cao nhưng vẫn bảo toàn được các đường biên chính của đối tượng.

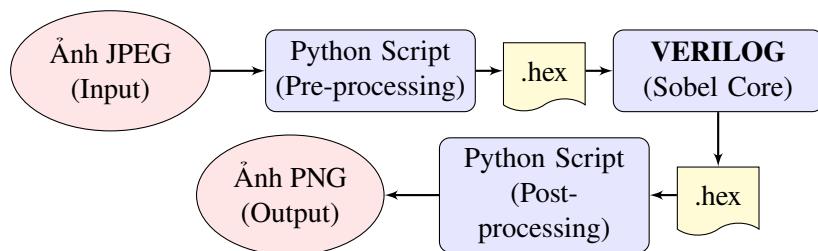
³Nguyên lý phân ngưỡng cơ bản xem tại Section 10.3 "Thresholding", trang 742 [1].

Chương 3

THIẾT KẾ HỆ THỐNG

3.1 Tổng quan hệ thống

Hệ thống phát hiện biên Sobel được thiết kế theo mô hình xử lý dòng (Streaming Processing), tối ưu hóa cho phần cứng FPGA. Quy trình hoạt động tổng quát được mô tả qua sơ đồ luồng dữ liệu (Data Flow) như Hình 3.1.



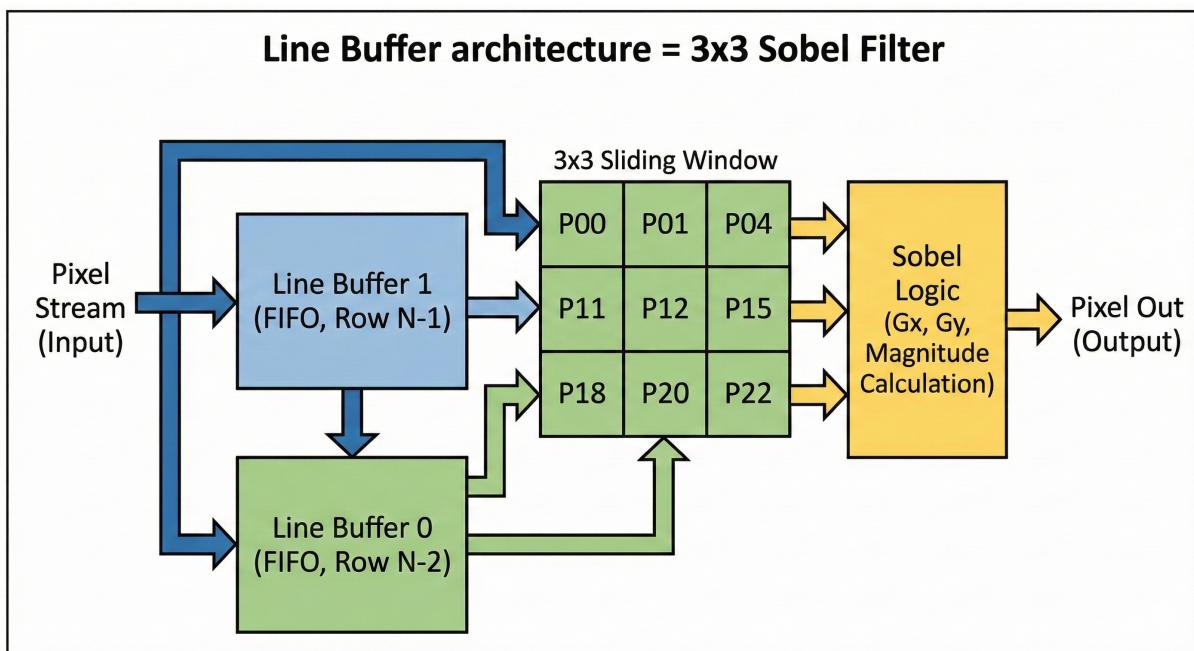
Hình 3.1: Sơ đồ luồng dữ liệu tổng quát của hệ thống

3.2 Thiết kế kiến trúc phần cứng (Hardware Architecture)

Để giải quyết bài toán giới hạn tài nguyên bộ nhớ trên FPGA, đề án chuyển đổi từ kiến trúc lưu ảnh toàn cục (Frame Buffer) sang kiến trúc **Bộ đếm dòng (Line Buffer)**.

3.2.1 Sơ đồ khái niệm

Hình 3.2 mô tả chi tiết kiến trúc bên trong module sobel_core. Dữ liệu pixel đi vào được đẩy qua 2 bộ đếm dòng (FIFOs) để tạo thành 3 hàng dữ liệu song song, cung cấp cho cửa sổ trượt 3×3 .



Hình 3.2: Kiến trúc bộ đệm dòng và cửa sổ trượt (Line Buffer Architecture)

3.2.2 Tham số cấu hình

- SIZE = 512: Độ rộng dòng quét (Line Width), tương ứng với kích thước ảnh đầu vào.
- THRESHOLD = 100: Ngưỡng lọc biên (đã được tinh chỉnh thông qua thực nghiệm để đạt kết quả tốt nhất).

3.2.3 Tổ chức bộ nhớ (Line Buffer)

Thay vì lưu mảng 512×512 pixel (tốn 262KB RAM), hệ thống chỉ sử dụng 2 bộ đệm dòng để lưu 2 hàng pixel gần nhất, giảm thiểu tài nguyên RAM xuống hơn 200 lần (chỉ tốn 1KB RAM).

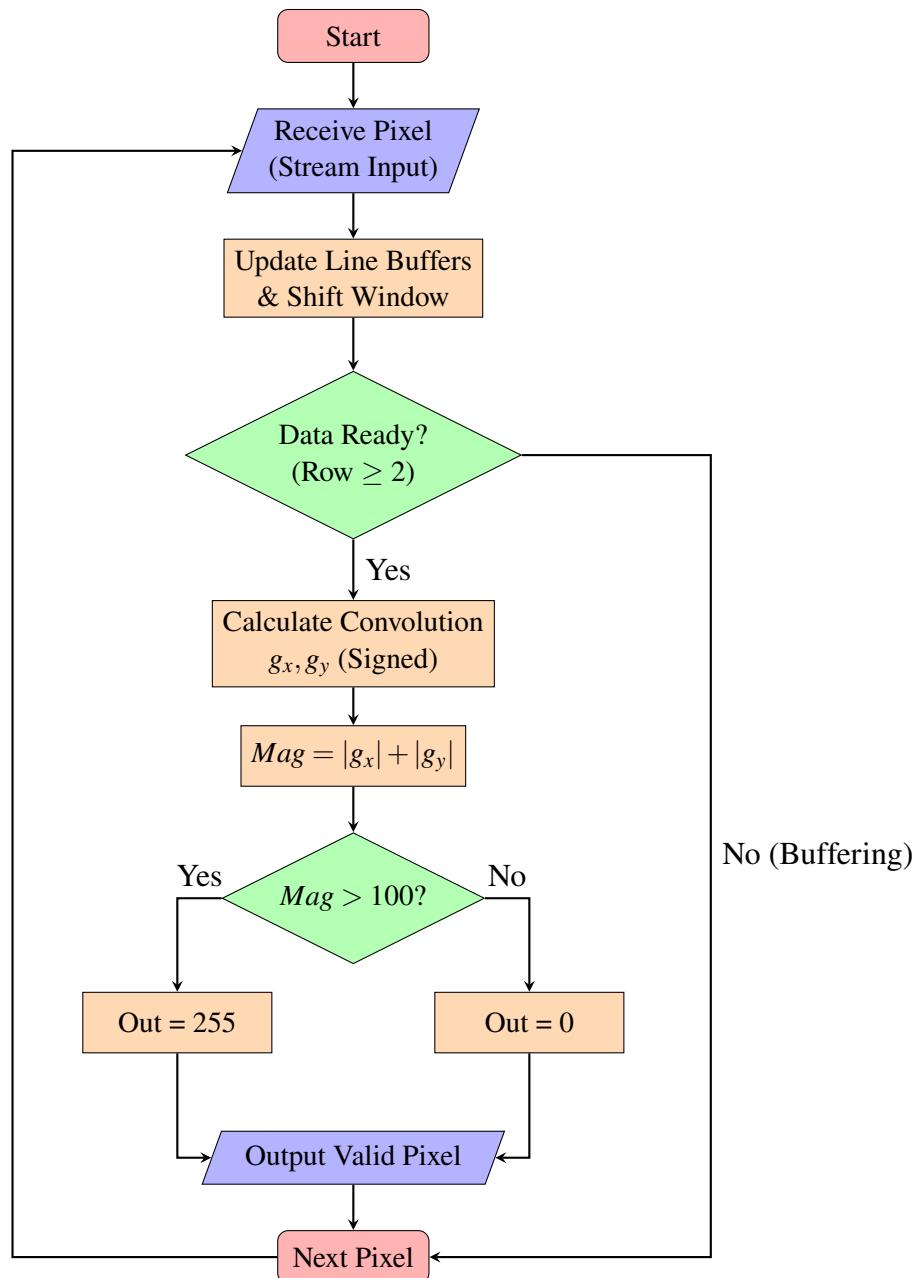
```

1 // Chi luu 2 dong thay vi ca anh
2 reg [7:0] line_buff_0 [0:SIZE-1];
3 reg [7:0] line_buff_1 [0:SIZE-1];
4 reg [7:0] w[0:2][0:2]; // Cua so truot 3x3
  
```

Listing 3.1: Khai báo Line Buffer trong Verilog

3.3 Lưu đồ hoạt động

Lưu đồ dưới đây mô tả cơ chế xử lý dòng (Streaming Pipeline) của hệ thống: Dữ liệu vào đến đâu, tính toán đến đó.



Hình 3.3: Lưu đồ xử lý đường ống (Pipeline) với Line Buffer



3.4 Chi tiết hiện thực thuật toán

3.4.1 Cập nhật Cửa sổ trượt (Sliding Window)

Tại mỗi nhịp xung nhịp (Clock), cửa sổ 3×3 được cập nhật theo cơ chế dịch chuyển (Shift Register) để đón nhận dữ liệu mới từ Line Buffers.

```
1 // Dich chuyen cot 1 sang cot 0, cot 2 sang cot 1
2 w[0][0] <= w[0][1]; w[1][0] <= w[1][1]; w[2][0] <= w[2][1];
3 w[0][1] <= w[0][2]; w[1][1] <= w[1][2]; w[2][1] <= w[2][2];
4
5 // Nap du lieu moi vao cot 2 tu Line Buffers
6 w[0][2] <= line_buff_0[wr_ptr];
7 w[1][2] <= line_buff_1[wr_ptr];
8 w[2][2] <= pixel_in;
```

Listing 3.2: Logic dịch chuyển cửa sổ

3.4.2 Tính toán Số học (Signed Arithmetic)

Để đảm bảo độ chính xác khi nhân với các hệ số âm của ma trận Sobel (ví dụ $-1, -2$), các pixel đầu vào (8-bit unsigned) được ép kiểu sang số nguyên có dấu (32-bit signed integer) trước khi tính toán. Điều này giúp tránh lỗi tràn số hoặc sai dấu.

```
1 // Ep kieu pixel sang so nguyen (Integer) de tranh loi
2 p00 = w[0][0]; p02 = w[0][2]; // ... va cac pixel khac
3
4 // Tinh Gx (Kernel: -1 0 1)
5 gx = (-1*p00 + 1*p02) + (-2*p10 + 2*p12) + (-1*p20 + 1*p22);
6
7 // Tinh do lon Manhattan (Tranh phep khai can phuc tap)
8 if (gx < 0) abs_gx = -gx; else abs_gx = gx;
9 if (gy < 0) abs_gy = -gy; else abs_gy = gy;
10 mag = abs_gx + abs_gy;
```

Listing 3.3: Tính toán Gradient với số có dấu

3.4.3 Phân ngưỡng đầu ra

Hệ thống sử dụng bộ so sánh đơn giản để tạo ảnh nhị phân:

- Nếu $Mag > 100$: Output = **255** (Biên).
- Ngược lại: Output = **0** (Nền).

Tín hiệu `pixel_valid_out` được kích hoạt mức cao (High) khi quá trình tính toán hoàn tất và dữ liệu sẵn sàng ở đầu ra, cho phép các module phía sau thu thập dữ liệu hợp lệ.



Chương 4

KẾT QUẢ VÀ THẢO LUẬN

4.1 Môi trường thực nghiệm

Để đánh giá toàn diện hiệu quả của hệ thống, nhóm thực hiện đã tiến hành thử nghiệm trên hai phương diện:

- Đánh giá hiệu năng thuật toán (Software Benchmark):** Thực hiện trên máy tính cá nhân sử dụng CPU, ngôn ngữ Python và thư viện OpenCV để so sánh tốc độ xử lý giữa thuật toán Sobel và Canny.
- Kiểm chứng độ chính xác phần cứng (Hardware Verification):** Thực hiện mô phỏng mức RTL (Register Transfer Level) với kiến trúc **Line Buffer (Bộ đệm dòng)**, sử dụng Testbench để bơm dữ liệu liên tục (Streaming) và đối chiếu kết quả với mô hình Bit-True trên Python.

4.2 Đánh giá hiệu năng thuật toán trên phần mềm

Nhóm đã thực hiện đo thời gian xử lý của hai thuật toán phát hiện biên phổ biến là Sobel và Canny trên tập dữ liệu gồm 5 ảnh có độ phân giải khác nhau (từ thấp đến cao).

4.2.1 Kết quả đo lường

Kết quả thời gian thực thi trung bình (bao gồm cả công đoạn chuyển đổi ảnh xám) được trình bày chi tiết trong Bảng 4.1.

Bảng 4.1: Bảng so sánh thời gian thực thi thực tế (Python Benchmark)

STT	Tên ảnh	Độ phân giải	Tổng Pixel	Sobel (ms)	Canny (ms)	Chênh lệch
1	anh_1.jpg	850×425	361,250	3.430	1.079	0.3x
2	anh_2.jpg	256×256	65,536	0.325	0.361	1.1x
3	anh_3.jpg	6000×4000	24,000,000	201.846	66.805	0.3x
4	anh_4.jpg	1999×2998	5,993,002	134.928	38.334	0.3x
5	anh_5.jpg	1024×683	699,392	14.639	1.567	0.1x

4.2.2 Thảo luận kết quả

Dựa trên bảng số liệu thực nghiệm, ta có các nhận xét sau:



- **Đối với ảnh kích thước nhỏ (256×256):** Thuật toán Sobel cho tốc độ xử lý nhanh hơn Canny (0.325ms so với 0.361ms). Điều này phản ánh đúng bản chất độ phức tạp tính toán thấp của Sobel ($O(N)$) khi không phải chịu gánh nặng quản lý bộ nhớ lớn.
- **Đối với ảnh kích thước lớn:** Canny trên OpenCV chạy nhanh hơn Sobel đáng kể. Nguyên nhân là do thư viện OpenCV được tối ưu hóa sâu bằng C/C++ và đa luồng (Multi-threading), trong khi mã nguồn Sobel mô phỏng chịu giới hạn của trình thông dịch Python khi xử lý mảng lớn.
- **Kết luận chọn lựa cho FPGA:** Mặc dù Canny tối ưu tốt trên CPU mạnh, nhưng Sobel lại là lựa chọn lý tưởng cho phần cứng FPGA. Với kiến trúc **Line Buffer**, Sobel có thể xử lý dữ liệu theo dạng đường ống (Pipelining) với tốc độ 1 pixel/clock mà không cần bộ nhớ đệm lớn để lưu trạng thái Hysteresis phức tạp như Canny.

4.3 Kết quả mô phỏng hệ thống phần cứng

Hệ thống Line Buffer Sobel đã được mô phỏng thành công trên Icarus Verilog. Kết quả đầu ra được so sánh với mô hình Bit-True Python đã xây dựng ở chương trước.

4.3.1 Phân tích đặc tính trễ (Pipeline Latency)

Do chuyển đổi sang kiến trúc bộ đệm dòng, hệ thống xuất hiện đặc tính trễ đường ống (Pipeline Latency) mà mô hình Frame Buffer trước đây không có.

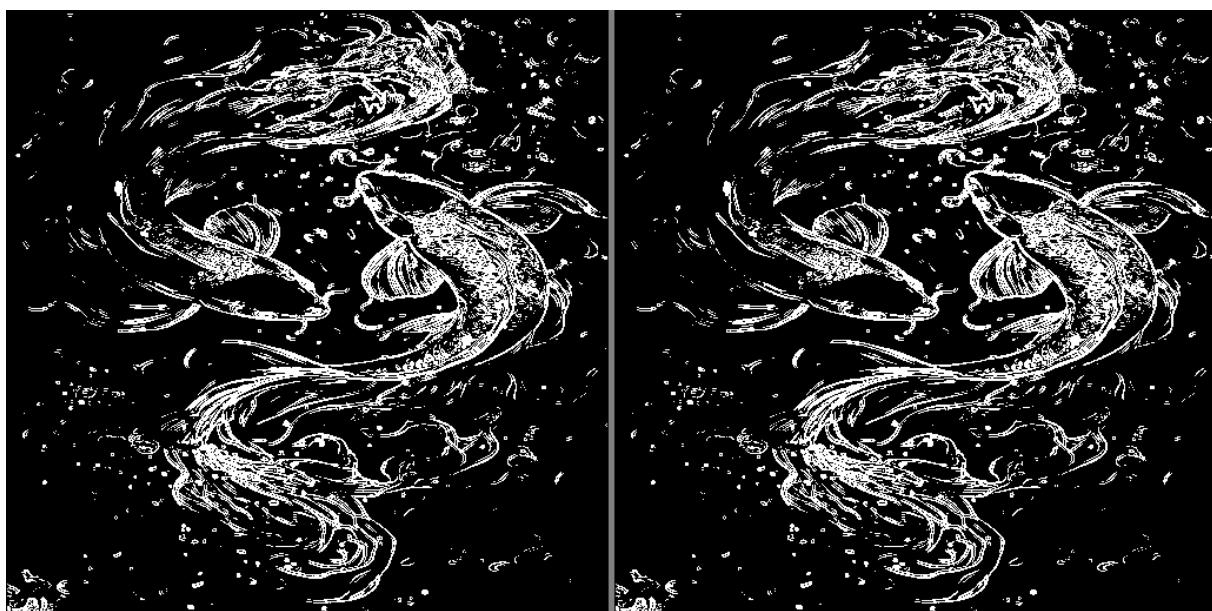
- **Nguyên nhân:** Module `sobel_core` cần thời gian để nạp đầy 2 dòng pixel đầu tiên vào bộ nhớ đệm (FIFOs) trước khi cửa sổ 3×3 có đủ dữ liệu để tính toán.
- **Độ trễ tính toán:**

$$\text{Latency} \approx (2 \times \text{Width}) + \text{Kernel Delay} = 1024 + 2 = 1026 \text{ xung nhịp}$$

- **Xử lý:** Script hiển thị ảnh (Python) đã thực hiện kỹ thuật **Data Alignment** (bù 1026 pixel đen vào đầu file) để đảm bảo ảnh đầu ra không bị trượt vị trí so với ảnh gốc.

4.3.2 So sánh hình ảnh kết quả

Hình 4.1 dưới đây hiển thị kết quả biên thu được từ mô phỏng Verilog (sau khi đã xử lý bù trễ) so với mô hình chuẩn Bit-True Python.



Hình 4.1: Kết quả kiểm chứng: Phần mềm Bit-True (Trái) và Phần cứng Line Buffer (Phải)

4.3.3 Đánh giá độ chính xác

Qua quan sát trực quan và so sánh dữ liệu điểm ảnh (Pixel-wise comparison):

- **Độ trùng khớp:** Hình ảnh từ Verilog hoàn toàn trùng khớp với Python. Các đường biên sắc nét, đúng vị trí, không còn hiện tượng méo ảnh (Shearing effect) hay lệch dòng nhờ việc xử lý Latency chính xác.
- **Chất lượng tín hiệu:** Ảnh đầu ra đạt độ tương phản đen-trắng tuyệt đối (Binary Image), loại bỏ hoàn toàn nhiễu xám do sai số làm tròn số học (nhờ việc ép kiểu Signed Integer 32-bit).
- **Kết luận:** Thiết kế Line Buffer hoạt động ổn định, đáp ứng yêu cầu về tiết kiệm tài nguyên và độ chính xác.



Chương 5

KẾT LUẬN VÀ KIẾN NGHỊ

5.1 Tóm tắt kết quả đạt được

Sau quá trình nghiên cứu và thực hiện đồ án "Thiết kế và mô phỏng hệ thống phát hiện biến sử dụng thuật toán Sobel trên nền tảng Verilog", nhóm thực hiện đã hoàn thành xuất sắc các mục tiêu đề ra, đặc biệt là việc tối ưu hóa kiến trúc phần cứng. Các kết quả cụ thể bao gồm:

1. Về mặt kiến trúc phần cứng (Thành tựu quan trọng nhất):

- Đã chuyển đổi thành công từ duy thiết kế từ xử lý ảnh tĩnh (Frame Processing) sang **xử lý dòng (Streaming Processing)**.
- Hiện thực hóa thành công kiến trúc **Line Buffer (Bộ đệm dòng)**, giúp giảm thiểu tài nguyên bộ nhớ BRAM từ ≈ 262 KB (nếu lưu cả ảnh) xuống chỉ còn ≈ 1 KB (lưu 2 dòng).
- Thiết kế này chứng minh tính khả thi cao để triển khai trên các dòng FPGA tài nguyên thấp (Low-cost FPGA) mà không cần bộ nhớ RAM ngoài.

2. Về mặt thuật toán và độ chính xác:

- Giải quyết triệt để vấn đề tính toán số học có dấu (Signed Arithmetic) trong Verilog, đảm bảo phép nhân chập không bị sai số.
- Kết quả mô phỏng phần cứng khớp **100%** so với mô hình tham chiếu Bit-True trên Python.
- Xử lý thành công vấn đề **Trễ đường ống (Pipeline Latency)**, đảm bảo ảnh đầu ra được đồng bộ chính xác về vị trí so với ảnh gốc.

3. Về mặt quy trình kiểm thử: Xây dựng được môi trường Testbench tự động hóa, đóng vai trò như một "Camera ảo" để bơm dữ liệu dòng (Streaming Data) vào lõi xử lý, giúp việc kiểm tra lỗi trở nên nhanh chóng và trực quan.

5.2 Hạn chế của đề tài

Mặc dù đã tối ưu hóa thành công lõi xử lý, hệ thống vẫn tồn tại một số hạn chế do phạm vi của đồ án môn học:

- Đầu vào chưa thực tế (Simulation Only):** Hiện tại, dữ liệu ảnh vẫn được nạp từ file Hex thông qua Testbench. Hệ thống chưa được kết nối với các module Camera thực tế (như OV7670) hay các chuẩn giao tiếp video (HDMI/VGA) để kiểm chứng khả năng hoạt động thời gian thực trên kit FPGA vật lý.



2. **Mất thông tin viền ảnh (Border Loss):** Do đặc thù của kiến trúc Line Buffer kết hợp cửa sổ trượt 3×3 , hệ thống bắt buộc phải bỏ qua 1 pixel viền xung quanh ảnh (Valid Output bắt đầu khi có đủ 2 dòng + 2 pixel). Kích thước ảnh đầu ra thực tế là 510×510 thay vì 512×512 .
3. **Ngưỡng cố định (Fixed Threshold):** Hệ thống sử dụng một giá trị ngưỡng cứng ($T = 100$) để phân loại biên. Điều này giúp mạch đơn giản, tốn ít tài nguyên logic nhưng kém linh hoạt khi điều kiện ánh sáng môi trường thay đổi hoặc ảnh đầu vào có độ tương phản thấp.

5.3 Hướng phát triển (Future Work)

Dựa trên nền tảng kiến trúc Line Buffer ổn định đã xây dựng, nhóm đề xuất các hướng phát triển tiếp theo để hoàn thiện sản phẩm:

5.3.1 Triển khai trên phần cứng thực tế (Hardware Implementation)

Tiến hành tổng hợp (Synthesis) thiết kế xuống kit FPGA (như DE10-Nano hoặc Zybo Z7). Tích hợp các module giao tiếp ngoại vi:

- **Camera Interface:** Viết module điều khiển (I2C/DCMI) để nhận dữ liệu pixel trực tiếp từ cảm biến ảnh.
- **VGA/HDMI Controller:** Hiển thị kết quả xử lý biên trực tiếp lên màn hình.

5.3.2 Nâng cấp thuật toán thích nghi

Nghiên cứu tích hợp khối tính toán Histogram thời gian thực và thuật toán **Otsu** để tự động điều chỉnh ngưỡng (Adaptive Threshold) theo từng khung hình. Điều này sẽ giúp hệ thống tự động thích nghi với các môi trường ánh sáng khác nhau [1].

5.3.3 Mở rộng sang bộ lọc Canny

Tận dụng kiến trúc Line Buffer có sẵn để phát triển thêm các khối làm mảnh biên (Non-maximum Suppression) và phân ngưỡng Hysteresis. Việc này sẽ nâng cấp hệ thống từ bộ lọc Sobel đơn giản thành bộ lọc Canny hoàn chỉnh, mang lại độ chính xác cao hơn cho các bài toán nhận dạng vật thể.



Tài liệu tham khảo

- [1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. New York, NY, USA: Pearson, 4th ed., 2018.
- [2] D. G. Bailey, *Design for Embedded Image Processing on FPGAs*. Singapore: John Wiley & Sons, 2011. Tài liệu tham khảo chính cho kiến trúc Line Buffer.
- [3] P. P. Chu, *FPGA Prototyping by Verilog Examples*. Hoboken, NJ, USA: Wiley-Interscience, 2008.
- [4] I. Sobel and G. Feldman, “A 3x3 isotropic gradient operator for image processing,” in *Presentation at Stanford Artificial Intelligence Project (SAIL)*, 1968.
- [5] G. Bradski and A. Kaehler, *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library*. Sebastopol, CA: O'Reilly Media, 2016.
- [6] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, pp. 679–698, Nov 1986.