ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN-ĐIỆN TỬ

BỘ MÔN VIỄN THÔNG

BÀI TẬP LỚN: XỬ LÝ ẢNH

XÂY DỰNG PHẦN MỀM SCAN TÀI LIỆU SỬ DỤNG THƯ VIỆN OPENCV

Giáo viên hướng dẫn: ThS. Nguyễn Khánh Lợi

Nhóm: 01

Lớp: DT01

Nhóm trưởng: Bùi Thanh Bình

MSSV: 2010925

TP. Hồ Chí Minh, 2024

BÁO CÁO KẾT QUẢ LÀM VIỆC NHÓM VÀ BẢNG ĐIỂM BTL

Học phần:EE3035

Lớp: DT01 HK:233 Năm học:2023-2024

Đề tài: Xây dựng phần mềm scan tài liệu sử dụng thư viện OPENCV

STT	Mã số SV	Họ và Tên	Đóng góp
1	2010925	Bùi Thanh Bình	100%
2	2010701	Phan Vũ Bảo Tín	100%
3	2010283	Lê Đình Minh Huy	100%

Họ và tên nhóm trưởng: Bùi Thanh Bình

Email: binh.buithanh27@hcmut.edu.vn

Nhận xét của GV:

.....

.....

GIẢNG VIÊN

(Ký và ghi rõ họ, tên)

NHÓM TRƯỞNG

(Ký và ghi rõ họ, tên)

Bùi Thanh Bình

GIỚI THIỆU

1. Đặt vấn đề.

Ngày nay, với sự phát triển mạnh mẽ của công nghệ, việc số hóa tài liệu trở thành một nhu cầu cấp thiết trong nhiều lĩnh vực, đặc biệt là trong quản lý tài liệu và lưu trữ thông tin. Số hóa tài liệu không chỉ giúp bảo quản tài liệu lâu dài mà còn tạo điều kiện thuận lợi cho việc tìm kiếm, truy cập và chia sẻ thông tin. Các công nghệ xử lý ảnh và nhận dạng ký tự quang học (OCR) đóng vai trò quan trọng trong việc chuyển đổi các tài liệu giấy sang định dạng số.

Tuy nhiên, để phát triển một hệ thống scan tài liệu hiệu quả, việc ứng dụng các thuật toán xử lý ảnh hiện đại là cần thiết. Thư viện OpenCV là một công cụ mạnh mẽ và phổ biến, hỗ trợ các nhà phát triển trong việc xây dựng các ứng dụng xử lý ảnh, bao gồm cả việc phát triển phần mềm scan tài liệu.

2. Mục đích nghiên cứu

Mục đích của báo cáo này là đưa ra một bản ôn tập về lý thuyết đã học như lấy ngưỡng, phát hiện cạnh, xử lý ảnh hình thái học. Từ đó ứng dụng và xây dựng một phần mềm scan tài liệu nhằm cung cấp một công cụ đơn giản, hiệu quả để quét và xử lý hình ảnh của các tài liệu, giúp biến đổi hình ảnh từ dạng ảnh gốc sang định dạng mà đọc và lưu trữ dễ dàng hơn.

Báo cáo được phân chia thành hai phần chính:

- **File Word/PDF**: Chứa nội dung về cơ sở lý thuyết, đồng thời tóm tắt và rút ra kết luận của nhóm sau quá trình thực hiện đề tài.
- **Files Code Python kết quả ứng dụng**: Bao gồm các mã nguồn của sản phẩm và giao diện chính cho người dùng.

3. Bố cục báo cáo

Báo cáo gồm 3 chương:

- *Chương 1: Cơ sở lý thuyết*
- *Chương 2: Xây dựng phần mềm scan tài liệu sử dụng thư viện OpenCV.*
- *Chương 3: Tài liệu tham khảo*

DANH MỤC BẢNG

DANH MỤC HÌNH ẢNH

NỘI DUNG

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1 Giới thiệu về xử lý ảnh

Xử lý ảnh là một lĩnh vực quan trọng và đa dạng trong khoa học máy tính và kỹ thuật, được sử dụng rộng rãi trong nhiều ngành công nghiệp từ y tế, giám sát an ninh đến công nghệ giải trí. Về cơ bản, xử lý ảnh đề cập đến các kỹ thuật số hóa và thao tác với hình ảnh để cải thiện chất lượng, trích xuất thông tin, hoặc phục vụ cho các mục đích khác nhau như nhận dạng đối tượng, phân đoạn, hoặc tái tạo hình ảnh.

Về mặt kỹ thuật, xử lý ảnh số bao gồm một loạt các bước và kỹ thuật phức tạp. Bước đầu tiên thường là thu nhận ảnh, nơi hình ảnh từ thế giới thực được chuyển đổi thành dữ liệu kỹ thuật số. Sau khi thu nhận, hình ảnh có thể được xử lý để cải thiện chất lượng hoặc phân tích thông tin. Các kỹ thuật phổ biến bao gồm biến đổi Fourier để phân tích ảnh trong miền tần số, giúp lọc bỏ nhiễu hoặc làm rõ các đặc điểm không dễ thấy trong miền không gian.

Bên cạnh đó, hình thái học toán học là một lĩnh vực quan trọng trong xử lý ảnh, tập trung vào việc phân tích và xử lý các cấu trúc hình học trong hình ảnh. Các phép toán hình thái học như giãn nở, co hẹp, mở rộng và đóng góp phần quan trọng trong việc xử lý ảnh để trích xuất các đặc điểm hình học cụ thể, phân đoạn ảnh, và nhận diện các cấu trúc nhất định trong ảnh. Những kỹ thuật này thường được sử dụng trong phân tích ảnh y học, viễn thám, và nhiều ứng dụng khác đòi hỏi sự chính xác cao.

Một yếu tố quan trọng khác trong xử lý ảnh là không gian màu. Hình ảnh kỹ thuật số thường được biểu diễn trong các không gian màu khác nhau như RGB, HSV,…, và việc chuyển đổi giữa các không gian màu là cần thiết để áp dụng các kỹ thuật xử lý ảnh phù hợp. Ví dụ, trong nhiều trường hợp, việc xử lý trên không gian màu HSV giúp dễ dàng hơn trong việc nhận dạng màu sắc và phân đoạn ảnh theo màu.

Ngoài ra, xử lý ảnh còn liên quan đến các kỹ thuật nhận dạng đối tượng, nơi các đặc trưng của ảnh như cạnh, góc, hoặc các hình dạng cụ thể được trích xuất và phân loại để nhận diện đối tượng trong ảnh. Các kỹ thuật nhận dạng đối tượng này có thể được áp dụng trong nhiều lĩnh vực từ giám sát an ninh đến tự động hóa công nghiệp và robot. Cùng với sự phát triển của công nghệ và các công cụ lập trình như

Python kết hợp với thư viện OpenCV, xử lý ảnh đã trở thành một lĩnh vực nghiên cứu và ứng dụng mạnh mẽ. OpenCV là một trong những thư viện mã nguồn mở phổ biến nhất được sử dụng trong xử lý ảnh, cung cấp một loạt các công cụ và thuật toán mạnh mẽ để thực hiện các tác vụ như phát hiện cạnh, nhận diện khuôn mặt, và theo dõi đối tượng trong thời gian thực.

Tóm lại, xử lý ảnh không chỉ là một lĩnh vực kỹ thuật quan trọng mà còn là một công cụ mạnh mẽ trong nhiều ngành công nghiệp, từ y tế, an ninh đến công nghệ giải trí. Với các kỹ thuật ngày càng tiên tiến và sự hỗ trợ mạnh mẽ từ các công cụ phần mềm, xử lý ảnh đang đóng góp ngày càng lớn vào việc cải thiện chất lượng cuộc sống và thúc đẩy sự phát triển của khoa học và công nghệ.

1.2 Tiền xử lý ảnh

Tiền xử lý ảnh đóng vai trò vô cùng quan trọng trong quá trình xử lý và phân tích hình ảnh, vì nó giúp cải thiện chất lượng ảnh và chuẩn bị dữ liệu cho các bước xử lý tiếp theo. Các kỹ thuật tiền xử lý như làm mờ, làm sắc nét, chuyển đổi không gian màu, và lấy ngưỡng giúp bỏ nhiễu, tăng cường chi tiết, và tách biệt các đối tượng trong ảnh một cách hiệu quả, từ đó giúp tăng độ chính xác của các thuật toán phát hiện cạnh, nhận dạng đối tượng,…

1.2.1 Các hệ tọa độ màu thông dụng và chuyển đổi không gian màu.

Trong xử lý ảnh, có nhiều hệ tọa độ màu (hay không gian màu) như thang xám (Grayscale), RGB, HSV,…

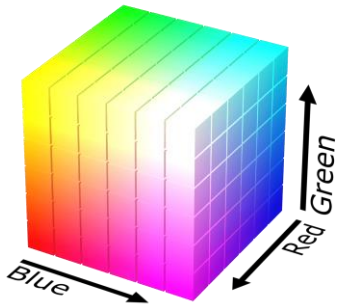
a) Thang xám (Grayscale)

Thang xám là một hệ màu đơn sắc, trong đó mỗi điểm ảnh chỉ có một giá trị cường độ, thường nằm giữa 0 (đen) và 255 (trắng). Đây là hệ màu đơn giản, thường được sử dụng trong các ứng dụng xử lý ảnh nhị phân hoặc các bước tiền xử lý trước khi áp dụng các thuật toán phức tạp hơn.

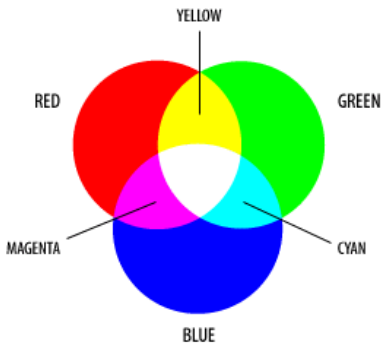
Hình 1.2.1-1: Minh họa thang xám có giá trị từ 0 đến 255.

b) Thang màu RGB

Đây là hệ tọa độ màu phổ biến nhất, nơi mỗi màu sắc được biểu diễn bằng ba thành phần: Đỏ (Red), Xanh lá (Green), và Xanh dương (Blue). Các màu khác nhau được tạo ra bằng cách thay đổi cường độ của mỗi thành phần này.



Hình 1.2.1-2: Hệ màu RGB



Hình 1.2.1-3: Cách tạo các màu khác nhau bằng cách pha trộn 3 màu cơ bản

c) Thang màu HSV

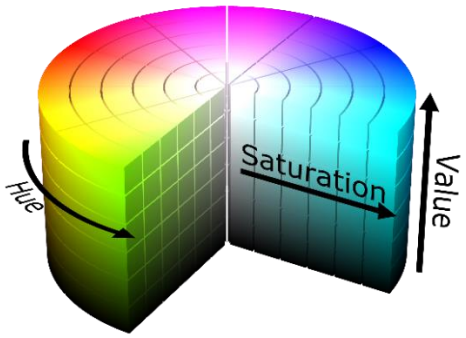
HSV là một hệ màu dựa trên nhận thức của con người về màu sắc:

+ Vùng màu (Hue): Đại diện cho màu sắc thực tế.

+ Độ bão hòa (Saturation): Đại diện cho mức độ tinh khiết của màu sắc.

+ Độ sáng (Value): Đại diện cho độ sáng của màu sắc,

thường được sử dụng trong các ứng dụng xử lý ảnh liên quan đến phân đoạn và nhận dạng màu sắc.



Hình 1.2.1-4: Hệ màu HSV

d) Chuyển đổi không gian màu

Chuyển đổi giữa các không gian màu là một trong những thao tác cơ bản trong xử lý ảnh, tụi em chủ yếu trình bày ở đây hai công thức chuyển từ hệ màu RGB và HSV sang thang mức xám (Grayscale):

+ Chuyển đổi từ RGB sang Grayscale (Y):

$$Y = \frac{R+G+B}{3} \text{ (độc lập HVS)}$$

$$Y = 0.299 \times R + 0.587 \times G + 0.114 \times B \text{ (phụ thuộc HVS)}$$

+ Chuyển đổi từ HSV sang Grayscale: Sử dụng giá trị Value (V).

1.2.2 Làm mờ ảnh (Image blurring)

Làm mờ ảnh (Image blurring) là một kỹ thuật tiền xử lý phổ biến nhằm giảm nhiễu và làm mịn các chi tiết không mong muốn trong ảnh. Trong báo cáo này tụi em sử dụng hai bộ lọc phổ biến nhất:

a) Làm mờ/mịn Gaussian (Gaussian blur)

Gaussian blur được sử dụng để giảm nhiễu và các chi tiết trong ảnh, giúp làm mượt các cạnh và vùng có sự thay đổi lớn về cường độ sáng. Kỹ thuật này được thực hiện bằng cách áp dụng một hàm Gaussian để tính toán giá trị của mỗi điểm ảnh trong hình.

Hàm Gaussian 2D được định nghĩa như sau:

$$G(x,y) = \frac{1}{\sqrt{2\pi}\sigma} . e^{-\frac{(z-\mu)^2}{2\sigma^2}} \quad (1)$$

Trong đó:

- **z, μ:** Giá trị mức xám và giá trị mức xám trung bình
- **σ:** Độ lệch chuẩn của phân bố Gaussian. Khi σ càng lớn, ảnh càng mờ.

Hàm Gaussian phân bổ trọng số cho các điểm ảnh xung quanh để tính giá trị trung bình mới cho mỗi điểm ảnh, với các điểm càng gần tâm sẽ có trọng số càng lớn.

Hình 1.2.2-1: Ảnh trước và sau khi áp dụng Gausian blur

Khi áp dụng Gaussian blur lên ảnh, ta sử dụng một kernel Gaussian để thực hiện nhân tích chập với toàn bộ ảnh.

b) Bộ lọc trung vị (Median Filter)

Thay thế giá trị pixel bởi giá trị pixel ở vị trí giữa trong lân cận sau khi sắp xếp theo thứ tự tăng hoặc giảm giần. Có khả năng triệt nhiễu tốt, đặc biệt là nhiễu muối tiêu.

Ví dụ: Xét lân cận với điểm ảnh có giá trị pixel 0 (tiêu), sau khi xếp theo thứ tự: 0, 26, 27, 28, 30, 31, 32, 34, 35. Ta có giá trị trung vị: 30

32	35	24	20	21
36	28	32	35	23
37	30	0	27	25
23	34	26	31	34
37	29	24	45	23

→

32	35	24	20	21
36	28	32	35	23
37	30	**30**	27	25
23	34	26	31	34
37	29	24	45	23

1.2.3 Bộ lọc Laplacian

Laplacian của một hình ảnh làm nổi bật các khu vực có cường độ thay đổi mạnh và do đó có thể được sử dụng để phát hiện cạnh. Nếu chúng ta để I(x,y) đại diện cho cường độ của ảnh thì Laplacian của ảnh được cho bởi công thức sau:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \quad (2)$$

Có thể xác định xấp xỉ rời rạc của Laplacian tại một điểm ảnh cụ thể bằng cách lấy giá trị trung bình có trọng số của cường độ pixel trong một vùng lân cận nhỏ của pixel. Việc này cũng có thể thực hiện bằng việc sử dụng các kernel xấp xỉ (tương đương).

0	1	0
1	-4	1
0	1	0

1	1	1
1	-8	1
1	1	1

Hình 1.2.3-1: Phát hiện các nét cạnh trên ảnh nhờ bộ lọc Laplacian

1.2.4 Lấy ngưỡng

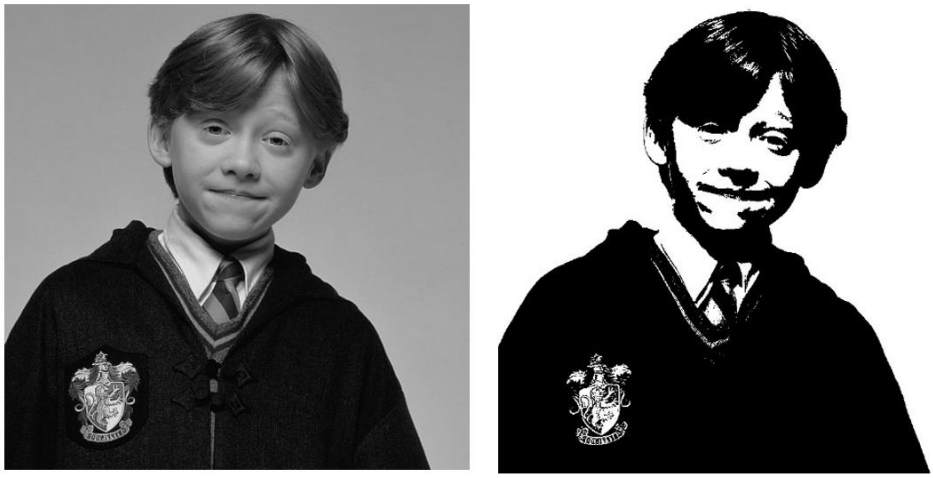
Lấy ngưỡng (Thresholding) là một kỹ thuật cơ bản trong xử lý ảnh, giúp phân đoạn ảnh đẻ cách ly một đối tượng quan tâm khỏi nền. Thông qua lấy ngưỡng, một ảnh gốc có thể được chuyển thành ảnh nhị phân, trong đó các pixel có cường độ sáng nằm trên ngưỡng sẽ được chuyển thành một giá trị (thường là trắng), và các pixel có cường độ thấp hơn ngưỡng sẽ được chuyển thành giá trị khác (thường là đen).

$$s = \begin{cases} 0, r \leq threshold \\ L, r > threshold \end{cases} \quad (3)$$

Các phương pháp lấy ngưỡng cũng có 3 loại:

a) Lấy ngưỡng cứng

Chỉ sử dụng một giá trị ngưỡng threshold cố định cho toàn bộ ảnh. Nhưng việc chỉ sử dụng một giá trị ngưỡng khiến cho ảnh dễ ảnh hưởng bởi nhiễu và bởi thay đổi độ sáng.



Hình 1.2.4-1: Ảnh âm bản sử dụng phương pháp lấy ngưỡng cứng

b) Lấy ngưỡng toàn cục

Lấy ngưỡng toàn cục (hay lấy ngưỡng thích nghi) sẽ không sử dụng một giá trị ngưỡng threshold cố định mà giá trị này được tính toán cho mỗi vùng nhỏ của ảnh, giúp xử lý tốt hơn các ảnh có độ sáng không đồng đều. Giải thuật:

- **Bước 1:** Xác định giá trị khởi tạo của T (thường là trung bình mức xám).
- **Bước 2:** Chia ảnh thành 2 vùng: W1 (điểm ảnh mức xám $\geq$ threshold) và vùng W2 (điểm ảnh mức xám < threshold).
- **Bước 3:** Tính giá trị trung bình (mean) mức xám của W1 là μ_1,W2 là μ_2.
- **Bước 4:** Cập nhật T = $(\mu_1 + \mu_2)/2$.
- **Bước 5:** Quay lại bước 2 đến khi nào $\Delta threshold \leq \varepsilon$.

Hình 1.2.4-2: Ảnh âm bản sử dụng phương pháp lấy ngưỡng toàn cục

c) Lấy ngưỡng tối ưu Otsu

Đây là phương pháp tự động tính toán giá trị ngưỡng tối ưu bằng cách phân tích histogram của ảnh. Giải thuật:

- **Bước 1**: Tính toán giá trị histogram chuẩn hóa

$$q_1 = \sum_{i=0}^{t} p(i) \tag{4}$$

- **Bước 2**: Tính tổng tích lũy $q_1(t)$ với t = 0,1,…,L
- **Bước 3**: Tính giá trị trung bình tích lũy $\mu(t)$ với t = 0,1,…,L

$$\mu(t) = \sum_{i=0}^{t} i \times p(i) \tag{5}$$

- **Bước 4:** Tính giá trị mức xám trung bình toàn cục μ_τ

$$\mu_\tau = \sum_{i=0}^{L} i \times p(i) \tag{6}$$

- **Bước 5:** Tính between-class variance $\sigma_B^2(t)$ với t = 0,1,…,L

$$\sigma_B^2(t) = \frac{[\mu_\tau q_1(t) - \mu(t)]^2}{q_1(t)[1 - q_1(t)]} \tag{7}$$

- **Bước 6:** Tìm ngưỡng tối ưu Otsu t* tại đó có giá trị $\sigma_B^2(t^*)$ lớn nhất. Nếu có nhiều giá trị t, xác định t* bằng cách lấy trung bình.
- **Bước 7:** Separability measure $\eta^* = \frac{\sigma_B^2(t^*)}{\mu_\tau}$

Hình 1.2.4-3: Ảnh âm bản sử dụng phương pháp lấy ngưỡng Otsu

1.3 Các phương pháp phát hiện cạnh

1.3.1 Edge Detection

Phát hiện cạnh (Edge Detection) là một kỹ thuật quan trọng trong xử lý ảnh, được sử dụng để xác định các điểm trong ảnh mà tại đó cường độ sáng thay đổi đột ngột. Những điểm này thường đại diện cho các biên của đối tượng trong ảnh.

Phát hiện cạnh là bước đầu tiên trong nhiều ứng dụng xử lý ảnh như nhận diện đối tượng, phân đoạn ảnh, và theo dõi chuyển động. Sau thời gian phát triển thì toán tử hoàn thiện đầu tiên được sử dụng là toán tử Sobel.

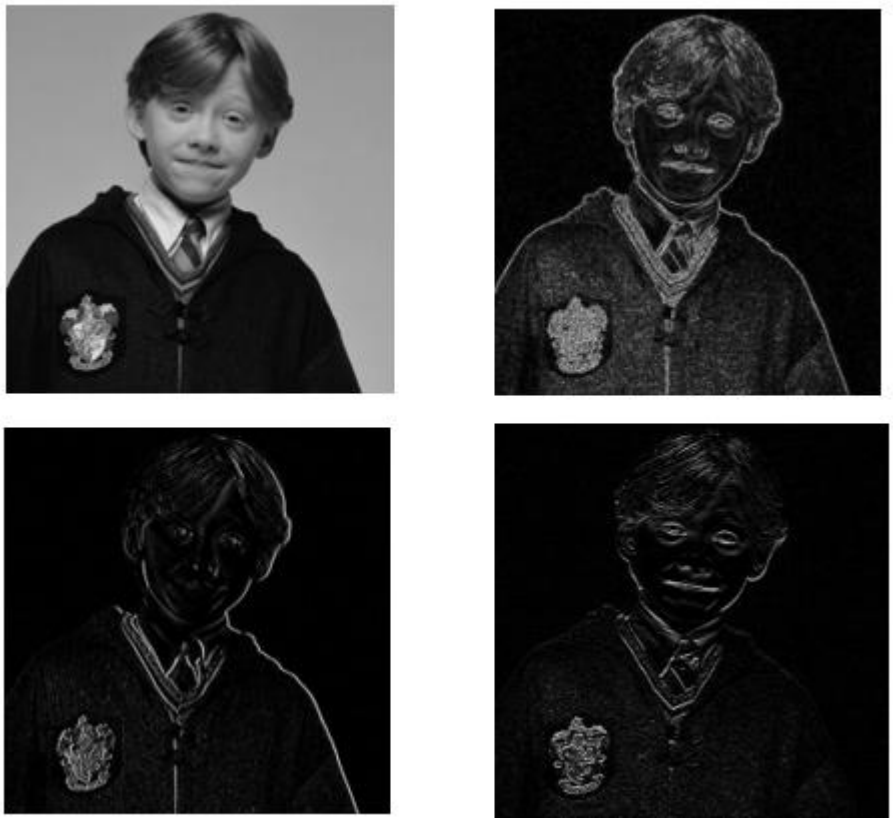
Đây là một kỹ thuật thuật phổ biến để phát hiện cạnh, đặc biệt nhạy cảm với các cạnh ngang và dọc. Nó sử dụng các bộ lọc (kernel) để tính toán gradient cường độ sáng theo các hướng x và y:

- Kernel Sobel theo trục x:

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

- Kernel Sobel theo trục y:

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



Hình 1.3.1-1:Phát hiện cạnh theo trục x, trục y và cả 2 trục.

1.3.2 Thuật toán Canny Edge Detection

Toán tử Canny là một phương pháp phát hiện cạnh mạnh mẽ và được sử dụng rộng rãi nhất hiện nay nhờ khả năng phát hiện cạnh rõ ràng và giảm nhiễu tốt. Canny Edge Detection bao gồm các bước chính:

- **Bước 1:** Lọc nhiễu: Giảm nhiễu trong ảnh bằng cách làm mờ ảnh với bộ lọc Gaussian.
- **Bước 2:** Tính gradient: Tính gradient của ảnh bằng các toán tử như Sobel để tìm ra hướng và cường độ của cạnh. Tính toán gradient:

$$M(x,y) = \|\nabla \mathrm{f}(\mathrm{x},\mathrm{y})\| = \sqrt{g_x^2(x,y) + g_y^2(x,y)} \quad (8)$$

Tính hướng gradient:

$$\alpha(x,y) = \tan^{-1}\left[\frac{g_y(x,y)}{g_x(x,y)}\right] \quad (9)$$

- **Bước 3:** Non-maximum suppression: Giữ lại các pixel là cực đại theo hướng gradient và loại bỏ các điểm không phải là cạnh.
- **Bước 4:** Ngưỡng hóa kép (Double thresholding): Xác định các cạnh tiềm năng bằng cách áp dụng hai ngưỡng, một cao và một thấp.
- **Bước 5:** Theo dõi cạnh qua lân cận (Edge tracking by hysteresis): Kết nối các cạnh bằng cách xem xét các điểm xung quanh.

Hình 1.3.2-1: Phát hiện cạnh sử dụng thuật toán Canny

1.3.3 Line detection

Phát hiện đường thẳng (Line Detection) được sử dụng để nhận diện các đường thẳng trong ảnh. Trong các phương pháp phát hiện đường thẳng trong ảnh, Biến đổi Hough (Hough Transform) là phổ biến nhất.

Biến đổi Hough là một kỹ thuật để tìm kiếm các hình dạng cụ thể trong ảnh, như đường thẳng, hình tròn,... Đối với đường thẳng, kỹ thuật này hoạt động bằng

cách chuyển đổi các điểm trong không gian ảnh sang không gian Hough và xác định các đường thẳng dựa trên sự giao nhau của các đường trong không gian này.

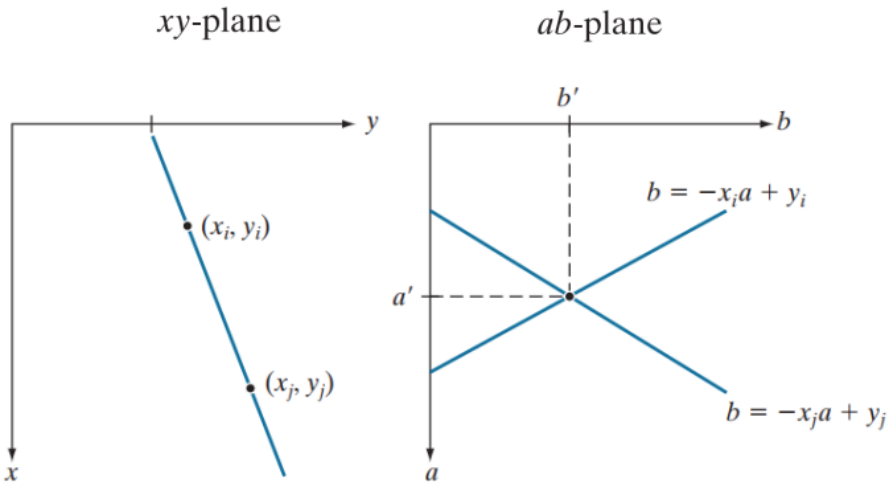
Trong không gian ảnh thông thường, đường thẳng có thể được biểu diễn bằng phương trình:

$$y = ax + b$$

Ta chuyển qua mặt phẳng ab:

$$b = -ax + y$$

Với mỗi điểm trong mặt phẳng xy, sẽ là một đường thẳng trong mặt phẳng ab và điểm giao nhau giữa các đường thẳng trong mặt phẳng ab là một đường thẳng đi qua tất cả các điểm trong mặt phẳng xy.

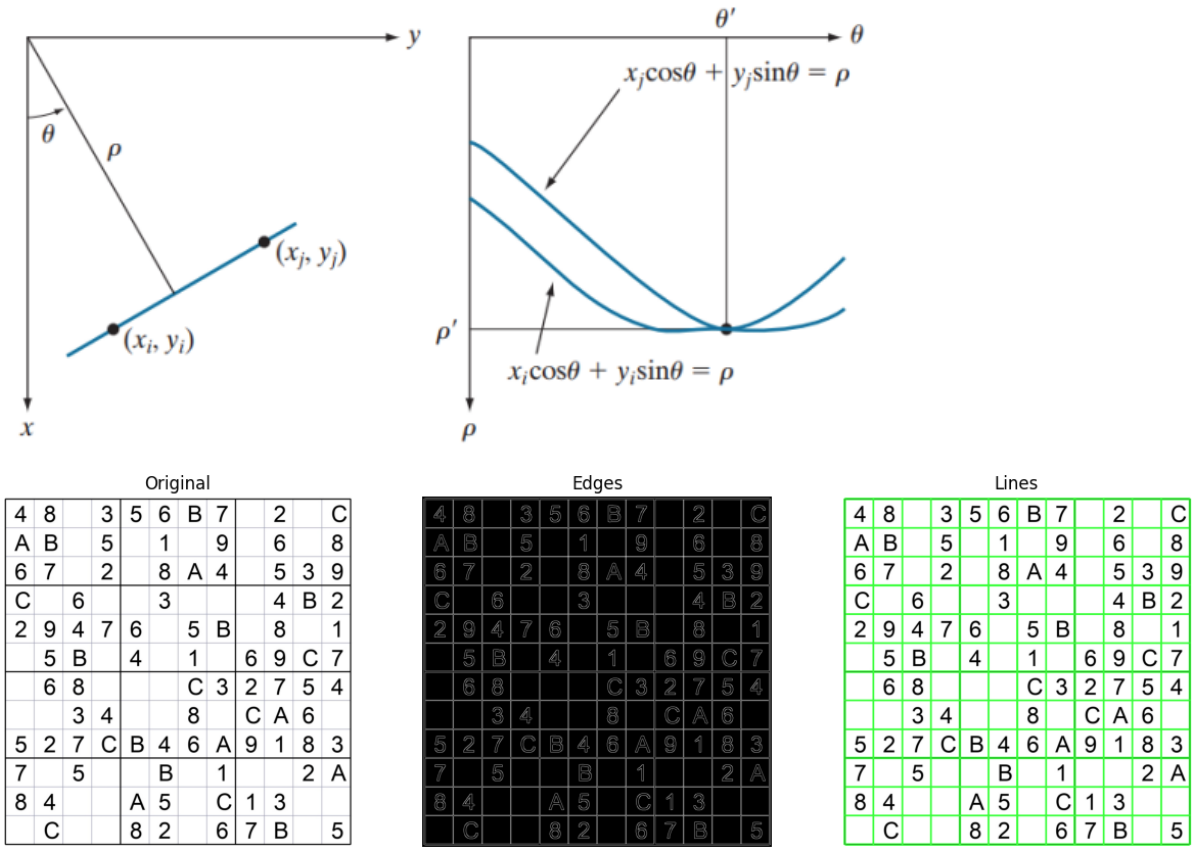


Tuy nhiên, phương trình này không ổn định khi a tiến đến vô cùng. Do đó, biến đổi Hough sử dụng một biểu diễn khác:

$$\rho = x.cos\theta + y.sin\theta$$

Trong đó:

- **ρ:** Khoảng các từ gốc tọa độ đến đường thẳng.
- **θ:** Góc giữa đường vuông góc với đường thẳng và trục x



Hình 1.3.3-1: Phát hiện đường thẳng sử dụng biến đổi Hough

1.4 Các phép toán hình thái học

1.4.1 Khái niệm cơ bản về hình thái học toán học

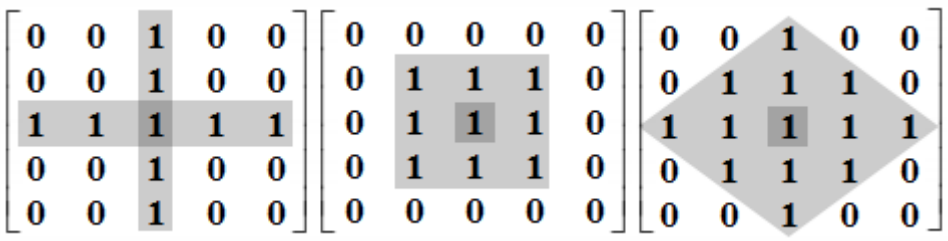
Các phép toán hình thái học (Mathematical Morphology Operations) là một nhóm các kỹ thuật trong xử lý ảnh, đặc biệt hữu ích trong việc phân tích các cấu trúc hình học trong ảnh, chủ yếu là ảnh nhị phân hoặc ảnh xám. Các phép toán này được sử dụng để trích xuất các đặc trưng hình học, loại bỏ nhiễu, lấp đầy lỗ hổng, và tách rời các đối tượng trong ảnh. Các phép toán hình thái học hoạt động dựa trên các phép toán tập hợp, sử dụng một phần tử cấu trúc (structuring element) để tương tác với ảnh.

Structuring Element là một ảnh nhị phân nhỏ. Tức là một ma trận nhỏ gồm các pixel mang giá trị 0 và 1:

+ Kích thước của ma trận xác định kích thước của SE.

+ Pixl có giá trị 0 được bỏ qua trong quá trình tính toán.

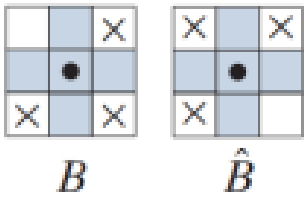
+ Luôn có một pixel làm mốc trong ma trận SE.



Có hai phép biến đổi cơ bản trong xử lý ảnh là:

Phản chiếu (Reflection): biến đổi phản chiếu của một (SE) B là:

$$\hat{B} = \{w|w = -b, for\ b \in B\}\,; (x, y) \rightarrow (-x, -y)$$



Tịnh tiến (Translation): Biến đổi tịnh tiến của một (SE) B với điểm z = (z1,z2) là:

$$(B)_z = \{c|c = b + z, for\ b \in B\}; (x, y) \rightarrow (x + z_1, y + z_2)$$

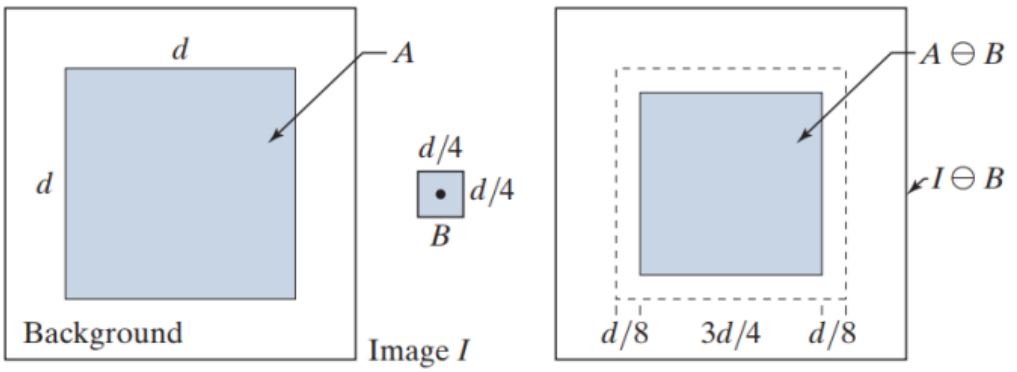
1.4.2 Các phép toán hình thái cơ bản:

a) Xói mòn (Erosion)

Xói mòn là phép toán làm giảm kích thước của các đối tượng sáng trong ảnh. Nó loại bỏ các pixel ở biên của các đối tượng. Phép toán này thường được sử dụng để loại bỏ nhiễu nhỏ, tách các đối tượng gần nhau, hoặc giảm kích thước của các đối tượng.. A và B là hai tập hợp con trong Z2 , thực hiện phép toán Erosion trong A theo B, kí hiệu là

$$A \ominus B = \left\{ z \middle| (B)_z \subseteq A \right\}$$

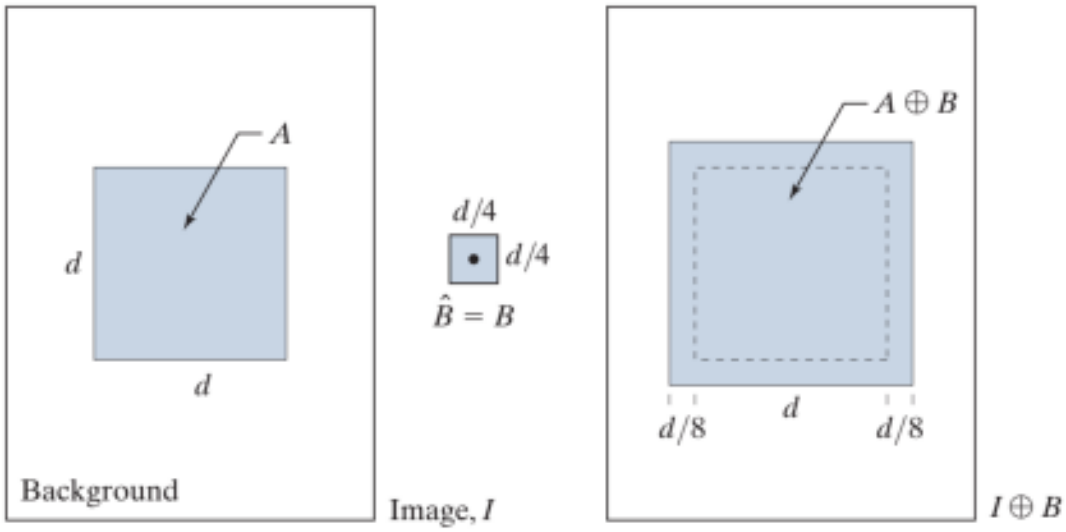
Phép co ảnh của tập hợp A bởi phần tử cấu trúc B là tập hợp các điểm z nằm ở tâm điểm của phần tử cấu trúc B sao cho Bz là tập con của A.



b) Giãn nở (Dilation)

Giãn nở là phép toán làm tăng kích thước của các đối tượng sáng trong ảnh. Nó thêm các pixel vào biên của các đối tượng. Phép toán này thường được sử dụng để lấp đầy các lỗ nhỏ, kết nối các đối tượng gần nhau, hoặc làm tăng kích thước của các đối tượng. A và B tương tự Erosion là hai tập con trong Z2 , thục hiện phép toán Dilation trong A theo B kí hiệu là

$$A \oplus B = \left\{ z \,\middle|\, [(\hat{B})_z \cap A] \subseteq A \right\}$$



Đối với mỗi pixel trong ảnh, phần tử cấu trúc được đặt sao cho tâm của nó trùng với pixel đó. Nếu ít nhất một pixel trong phần tử cấu trúc khớp với các pixel tương ứng trong ảnh, pixel tại vị trí trung tâm của phần tử cấu trúc sẽ được đặt giá trị 1 (trắng).

CHƯƠNG 2: XÂY DỰNG PHẦN MỀM SCAN TÀI LIỆU SỬ DỤNG THƯ VIỆN OPENCV

2.1 Giới thiệu

2.1.1 Mục tiêu và phạm vi của phần mềm

Phần mềm scan tài liệu được xây dựng nhằm cung cấp một công cụ đơn giản và hiệu quả để quét và xử lý hình ảnh của các tài liệu, giúp biến đổi hình ảnh từ dạng ảnh gốc sang định dạng dễ dàng đọc và lưu trữ. Mục tiêu chính là xây dựng một hệ thống có khả năng phát hiện, căn chỉnh và xử lý tài liệu từ ảnh chụp, đưa ra kết quả là một phiên bản số hóa chính xác của tài liệu. Phần mềm sử dụng thư viện OpenCV để thực hiện các tác vụ xử lý ảnh, với mục đích tạo ra một công cụ có giao diện đơn giản, dễ sử dụng nhưng vẫn đảm bảo chất lượng cao cho kết quả cuối cùng.

2.1.2 Các công cụ và công nghệ sử dụng

a) Pycharm IDE

PyCharm IDE là một môi trường phát triển tích hợp (IDE) hàng đầu dành cho lập trình Python, được phát triển bởi JetBrains. PyCharm nổi tiếng với các tính năng mạnh mẽ như hỗ trợ tự động hoàn thành mã, gỡ lỗi tích hợp, kiểm tra lỗi cú pháp theo thời gian thực, và quản lý môi trường ảo. Đặc biệt, PyCharm hỗ trợ rất tốt cho các dự án xử lý ảnh, khi lập trình viên cần tích hợp nhiều thư viện và công cụ khác nhau như NumPy, OpenCV, và Tkinter. Giao diện người dùng thân thiện, cùng với các tính năng quản lý dự án mạnh mẽ của PyCharm, giúp tối ưu hóa quá trình phát triển phần mềm, từ khâu viết mã đến thử nghiệm và triển khai. Nhờ vào các tiện ích như quản lý phiên bản (Git), tích hợp Jupyter notebook, và hệ thống plugin phong phú, PyCharm trở thành lựa chọn lý tưởng cho các dự án xử lý ảnh yêu cầu môi trường phát triển hiệu quả và đáng tin cậy.

b) Ngôn ngữ lập trình Python trong xử lý ảnh

Python là một trong những ngôn ngữ lập trình phổ biến nhất hiện nay, đặc biệt trong lĩnh vực xử lý ảnh. Sự phổ biến của Python trong cộng đồng lập trình viên chủ yếu đến từ cú pháp đơn giản, dễ học, cùng với một hệ sinh thái phong phú các thư viện hỗ trợ. Trong xử lý ảnh, Python không chỉ cung cấp các công cụ mạnh mẽ như NumPy và OpenCV mà còn có khả năng tích hợp với các ngôn ngữ

và thư viện khác, mở rộng khả năng của các ứng dụng. Khả năng quản lý bộ nhớ hiệu quả, cùng với sự hỗ trợ tốt từ các thư viện ngoại vi và cộng đồng mã nguồn mở lớn, làm cho Python trở thành ngôn ngữ lý tưởng cho các tác vụ xử lý ảnh từ cơ bản đến nâng cao. Ngoài ra, Python còn cho phép dễ dàng triển khai các ứng dụng xử lý ảnh trên nhiều nền tảng khác nhau, từ máy tính cá nhân đến các hệ thống nhúng, nhờ vào tính linh hoạt và khả năng mở rộng cao của nó.

c) Thư viện NumPy: Xử lý số học

NumPy (Numerical Python) là một thư viện nền tảng trong Python, cung cấp các công cụ cần thiết để thực hiện các phép toán số học trên mảng đa chiều và ma trận. Trong xử lý ảnh, NumPy đóng vai trò cực kỳ quan trọng, khi mỗi hình ảnh có thể được coi là một mảng hai chiều (hoặc ba chiều với ảnh màu) chứa các giá trị pixel. Với khả năng thao tác nhanh chóng và hiệu quả trên các mảng dữ liệu lớn, NumPy giúp tăng tốc độ xử lý và tối ưu hóa tài nguyên hệ thống. Ngoài ra, NumPy cung cấp các hàm toán học cao cấp cho các phép biến đổi tuyến tính, thống kê, và xử lý dữ liệu phức tạp, làm cho nó trở thành công cụ không thể thiếu trong mọi ứng dụng xử lý ảnh. Khả năng tương thích cao của NumPy với các thư viện khác như OpenCV và khả năng mở rộng với C/C++ thông qua các extension, giúp nó trở thành một thư viện cốt lõi trong hệ sinh thái xử lý ảnh bằng Python.

d) Thư viện OpenCV: Thực hiện các tác vụ xử lý ảnh

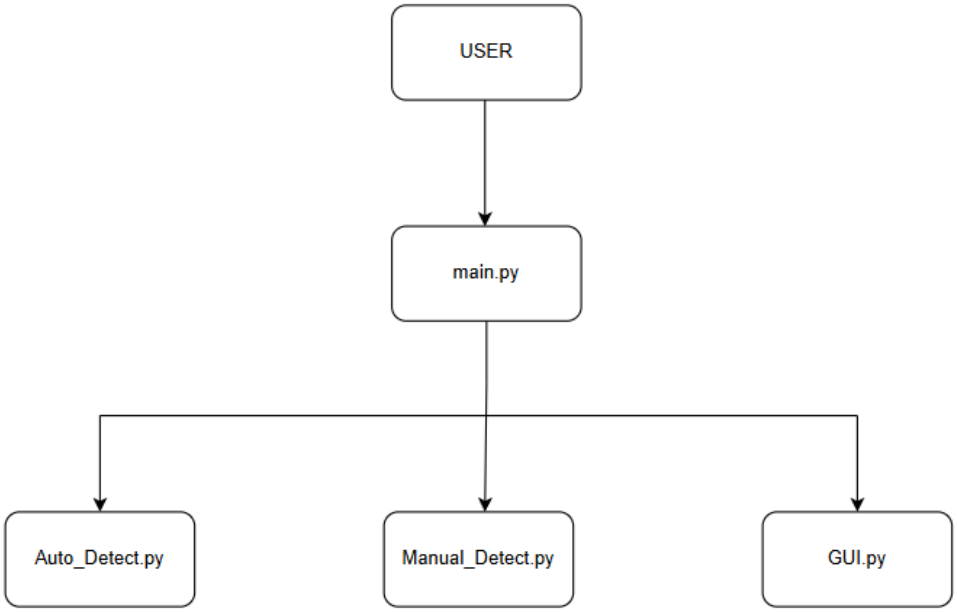
OpenCV (Open-Source Computer Vision Library) là một thư viện mã nguồn mở phổ biến nhất cho các tác vụ thị giác máy tính và xử lý ảnh. Được phát triển với mục tiêu cung cấp các công cụ mạnh mẽ nhưng dễ sử dụng, OpenCV hỗ trợ hầu hết các nền tảng phổ biến như Windows, Linux, macOS, và Android. Trong xử lý ảnh, OpenCV cung cấp một bộ công cụ toàn diện để thực hiện các tác vụ từ cơ bản như chuyển đổi không gian màu, lọc ảnh, đến các kỹ thuật phức tạp hơn như phát hiện cạnh, nhận diện khuôn mặt, và theo dõi đối tượng. Khả năng xử lý ảnh thời gian thực của OpenCV là một trong những ưu điểm nổi bật, giúp nó trở thành lựa chọn hàng đầu cho các ứng dụng yêu cầu hiệu suất cao. Ngoài ra, OpenCV còn hỗ trợ tích hợp với các thư viện khác như NumPy để mở rộng khả năng xử lý và cung cấp các công cụ tiện ích cho việc phát triển các ứng dụng thị giác máy tính hiện đại.

e) Thư việnTkinter: Xây dựng giao diện người dùng

Tkinter là thư viện tiêu chuẩn của Python để phát triển giao diện người dùng đồ họa (GUI). Tkinter cho phép các lập trình viên tạo ra các ứng dụng có giao diện người dùng trực quan và dễ sử dụng, từ các cửa sổ đơn giản đến các ứng dụng phức tạp với nhiều thành phần tương tác. Trong lĩnh vực xử lý ảnh, Tkinter thường được sử dụng để xây dựng các công cụ và phần mềm có giao diện người dùng, cho phép người dùng tương tác với các chức năng xử lý ảnh một cách trực quan mà không cần kiến thức chuyên sâu về lập trình. Tkinter cung cấp các widget đa dạng như nút bấm, khung nhập liệu, thanh trượt, và cửa sổ đối thoại, giúp dễ dàng triển khai các chức năng như lựa chọn và tải ảnh, điều chỉnh thông số xử lý, và hiển thị kết quả trực tiếp trên màn hình. Nhờ vào sự tích hợp chặt chẽ với Python và các thư viện khác, Tkinter giúp tạo ra các ứng dụng xử lý ảnh hoàn chỉnh với giao diện người dùng thân thiện và dễ tiếp cận.

2.2 Thiết kế hệ thống và lưu đồ thuật toán

Trong phần này, nhóm sẽ phân tích và trình bày chi tiết thiết kế của phần mềm scan tài liệu, bao gồm việc quản lý giao diện đồ họa người dùng (GUI), phát hiện tài liệu thủ công và tự động, cũng như tích hợp các chức năng để tạo thành một phần mềm quét tài liệu hoàn chỉnh. Mỗi thành phần của hệ thống sẽ được mô tả chi tiết, bao gồm cả cấu trúc mã nguồn và vai trò của từng tệp tin trong hệ thống.



Hình 2.1.2-1: Sơ đồ khối phần mềm scan tài liệu

2.2.1 GUI.py: Quản lý giao diện đồ họa

```
import cv2
import numpy as np

def stackImages(imgArray, scale, labels=[]):
    rows = len(imgArray)
    cols = len(imgArray[0])
    rowsAvailable = isinstance(imgArray[0], list)
    width = imgArray[0][0].shape[1]
    height = imgArray[0][0].shape[0]

    if rowsAvailable:
        for x in range(0, rows):
            for y in range(0, cols):
                imgArray[x][y] = cv2.resize(imgArray[x][y],
(0, 0), None, scale, scale)
                if len(imgArray[x][y].shape) == 2:
                    imgArray[x][y] =
cv2.cvtColor(imgArray[x][y], cv2.COLOR_GRAY2BGR)

        imageBlank = np.zeros((height, width, 3), np.uint8)
        hor = [imageBlank] * rows
        for x in range(0, rows):
            hor[x] = np.hstack(imgArray[x])
        ver = np.vstack(hor)
    else:
        for x in range(0, rows):
            imgArray[x] = cv2.resize(imgArray[x], (0, 0),
None, scale, scale)
            if len(imgArray[x].shape) == 2:
                imgArray[x] = cv2.cvtColor(imgArray[x],
cv2.COLOR_GRAY2BGR)
        ver = np.hstack(imgArray)

    if len(labels) != 0:
        eachImgWidth = int(ver.shape[1] / cols)
        eachImgHeight = int(ver.shape[0] / rows)
        for d in range(0, rows):
            for c in range(0, cols):
                label = labels[d][c]
                cv2.rectangle(ver, (c * eachImgWidth,
eachImgHeight * d),
                              (c * eachImgWidth + len(label) *
13 + 27, 30 + eachImgHeight * d),
                              (255, 255, 255), cv2.FILLED)
                cv2.putText(ver, label, (eachImgWidth * c +
10, eachImgHeight * d + 20),
                            cv2.FONT_HERSHEY_COMPLEX, 0.7,
(255, 0, 255), 2)

    return ver

def reorder(myPoints):
    myPoints = myPoints.reshape((4, 2))
    myPointsNew = np.zeros((4, 1, 2), dtype=np.int32)
    add = myPoints.sum(1)

    myPointsNew[0] = myPoints[np.argmin(add)]
    myPointsNew[3] = myPoints[np.argmax(add)]
    diff = np.diff(myPoints, axis=1)
    myPointsNew[1] = myPoints[np.argmin(diff)]
    myPointsNew[2] = myPoints[np.argmax(diff)]

    return myPointsNew
```

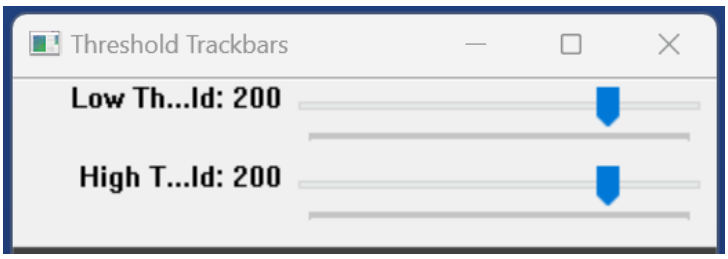
```
def biggestContour(contours):
    biggest = np.array([])
    max_area = 0
    for i in contours:
        area = cv2.contourArea(i)
        if area > 5000:
            peri = cv2.arcLength(i, True)
            approx = cv2.approxPolyDP(i, 0.02 * peri, True)
            if area > max_area and len(approx) == 4:
                biggest = approx
                max_area = area
    return biggest, max_area

def drawRectangle(img, biggest, thickness):
    cv2.line(img, (biggest[0][0][0], biggest[0][0][1]),
(biggest[1][0][0], biggest[1][0][1]), (0, 255, 0), thickness)
    cv2.line(img, (biggest[0][0][0], biggest[0][0][1]),
(biggest[2][0][0], biggest[2][0][1]), (0, 255, 0), thickness)
    cv2.line(img, (biggest[3][0][0], biggest[3][0][1]),
(biggest[2][0][0], biggest[2][0][1]), (0, 255, 0), thickness)
    cv2.line(img, (biggest[3][0][0], biggest[3][0][1]),
(biggest[1][0][0], biggest[1][0][1]), (0, 255, 0), thickness)

    return img

def nothing(x):
    pass

def initializeTrackbars(intialTracbarVals=0):
    cv2.namedWindow("Threshold Trackbars")
    cv2.resizeWindow("Threshold Trackbars", 340, 240)
    cv2.createTrackbar("Low Threshold", "Threshold Trackbars",
200, 255, nothing)
    cv2.createTrackbar("High Threshold", "Threshold
Trackbars", 200, 255, nothing)


def valTrackbars():
    Threshold1 = cv2.getTrackbarPos("Low Threshold",
"Threshold Trackbars")
    Threshold2 = cv2.getTrackbarPos("High Threshold",
"Threshold Trackbars")
    src = Threshold1, Threshold2
    return src
```

Bảng 2.2.1-1: Mã nguồn của file GUI.py

File GUI.py chịu trách nhiệm chính trong việc quản lý giao diện người dùng đồ họa (GUI) của phần mềm. Thư viện Tkinter được sử dụng để xây dựng các thành phần giao diện như cửa sổ, nút bấm, và các thanh trượt (trackbars) giúp người dùng có thể tương tác trực tiếp với phần mềm.

a) initializeTrackbars():

Hàm có chức năng tạo ra hai thanh trượt tương ứng với ngưỡng dưới và ngưỡng trên. Người dùng có thể điều chỉnh ngưỡng này để tìm ra giá trị phù hợp cho việc phát hiện cạnh.



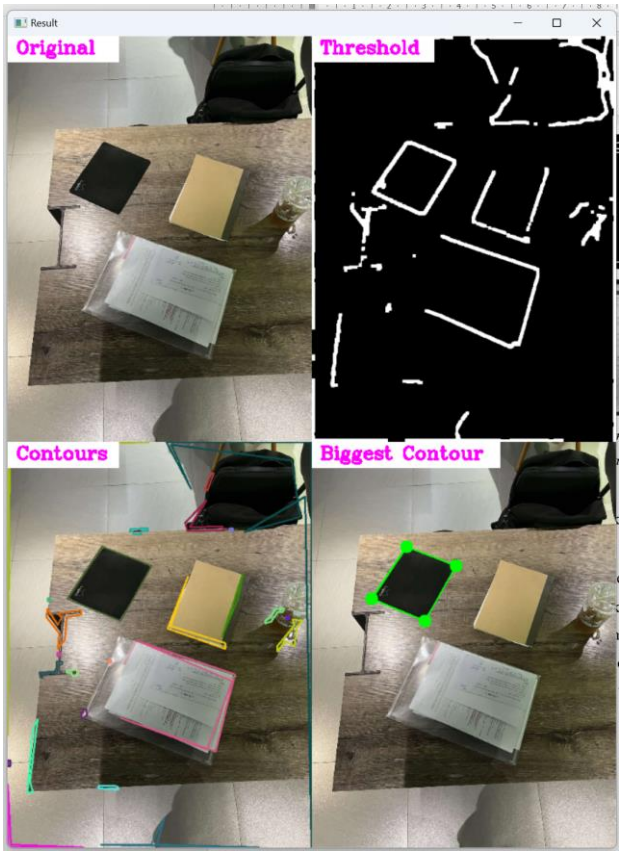
Hình 2.2.1-1: Cửa sổ giao diện thanh trượt do hàm initializeTrackbars() tạo ra.

b) valTrackbars():

Hàm này sẽ trả về giá trị hiện tại của hai thanh trượt này dưới dạng tuple.. Những giá trị này sẽ được sử dụng trong các thuật toán phát hiện cạnh, cụ thể là trong hàm cv2.Canny (trong file Auto_Detect.py).

c) stackImages(imgArray, scale, labels=[]):

Hàm này sắp xếp và hiển thị nhiều hình ảnh khác nhau trong một cửa sổ duy nhất, giúp người dùng dễ dàng so sánh các bước khác nhau trong quá trình xử lý ảnh. Hàm sử dụng các phương thức "cv2.resize" để điều chỉnh kích thước ảnh theo tỷ lệ scale và chuyển đổi ảnh xám sang ảnh màu nếu cần thiết ("cv2.cvtColor"). Các ảnh sẽ được xếp chồng lên nhau theo hàng và cột, sau đó được hiển thị cùng các nhãn chú thích để người dùng dễ nhận diện.



Hình 2.2.1-2: Cửa sổ thể hiện cách hình ảnh do hàm stackImages(imgArray, scale, labels=[]) tạo ra.

d) nothing(x):

Đây là một hàm mặc định được sử dụng khi cần một hàm callback cho các trackbar nhưng không yêu cầu thực hiện bất kỳ thao tác gì khi giá trị thay đổi.

e) reorder():

Hàm giúp sắp xếp lại các điểm góc của tứ giác dùng để đánh dấu trang tài liệu được xác định để đảm bảo rằng chúng được đặt theo đúng thứ tự: trên trái, trên phải, dưới trái, dưới phải.

f) biggestContour():

Hàm có chức năng tìm kiếm và trả về đường viền lớn nhất có hình dạng giống hình chữ nhật (4 điểm góc). Đây là đường viền tương ứng với tài liệu được phát hiện trong ảnh.

g) drawRectangle():

Hàm có vẽ một hình chữ nhật xung quanh tài liệu được phát hiện bằng cách sử dụng các điểm góc đã được sắp xếp.

2.2.2 Manual_detect.py: Phát hiện tài liệu thủ công

```
import cv2
import numpy as np

class ManualDetect:
    def __init__(self, image_path):
        self.image = cv2.imread(image_path)
        self.points = []
        self.selected_point = None
        self.dragging = False
        self.window_name = 'Adjust Rectangle'

    def apply_clahe(self, image, clip_limit=2.0,
tile_grid_size=(16, 16)):
        clahe = cv2.createCLAHE(clipLimit=clip_limit,
tileGridSize=tile_grid_size)
        return clahe.apply(image)

    def draw_rectangle(self, image, pts):
        temp_image = image.copy()
        cv2.polylines(temp_image, [pts], isClosed=True,
color=(0, 255, 0), thickness=2)
        return temp_image

    def window_popup(self, title, image):
        cv2.namedWindow(title, cv2.WINDOW_GUI_NORMAL)
        screen_res = 1920, 1080
        scale_width = screen_res[0] / image.shape[1]
        scale_height = screen_res[1] / image.shape[0]
        scale = min(scale_width, scale_height)
        window_width = int(image.shape[1] * scale)
        window_height = int(image.shape[0] * scale)
        cv2.resizeWindow(title, window_width, window_height)
        cv2.imshow(title, image)

    def mouse_handler(self, event, x, y, flags, param):
```

```
        if event == cv2.EVENT_LBUTTONDOWN:
            for i, pt in enumerate(self.points):
                if np.linalg.norm(pt - np.array([x, y])) < 10:
                    self.selected_point = i
                    self.dragging = True
                    break

        elif event == cv2.EVENT_MOUSEMOVE and self.dragging:
            if self.selected_point is not None:
                x = max(0, min(x, self.image.shape[1] - 1))
                y = max(0, min(y, self.image.shape[0] - 1))
                self.points[self.selected_point] =
np.array([x, y])
                temp_image = self.draw_rectangle(self.image,
np.array(self.points))
                cv2.imshow(self.window_name, temp_image)

        elif event == cv2.EVENT_LBUTTONUP:
            self.dragging = False
            self.selected_point = None

    def scan(self, image):
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        sharpen = cv2.GaussianBlur(gray, (0, 0), 3)
        sharpen = cv2.addWeighted(gray, 1.5, sharpen, -0.5, 0)
        thresh = cv2.adaptiveThreshold(sharpen, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 21, 15)
        return thresh

    def process_image(self):
        h, w = self.image.shape[:2]
        self.points = [np.array([int(w * 0.2), int(h * 0.2)]),
np.array([int(w * 0.8), int(h * 0.2)]),
                       np.array([int(w * 0.8), int(h * 0.8)]),
np.array([int(w * 0.2), int(h * 0.8)])]

        img_with_rect = self.draw_rectangle(self.image,
np.array(self.points))
        self.window_popup(self.window_name, img_with_rect)
        cv2.setMouseCallback(self.window_name,
self.mouse_handler)

        while True:
            key = cv2.waitKey(1) & 0xFF
            if key == ord('s'):
                break

        if len(self.points) == 4:
            dst_points = np.array([[0, 0], [w, 0], [w, h], [0,
h]], dtype="float32")
            M =
cv2.getPerspectiveTransform(np.array(self.points,
dtype="float32"), dst_points)
            warped = cv2.warpPerspective(self.image, M, (w,
h))
            self.window_popup('Warped', self.scan(warped))
            cv2.waitKey(0)

        cv2.destroyAllWindows()
```

Bảng 2.2.2-1:Mã nguồn của file Manuel_Detect.py

File Manual_Detect.py cung cấp các chức năng cần thiết để người dùng có thể xác định thủ công các góc của tài liệu trong ảnh, cho phép họ điều chỉnh và

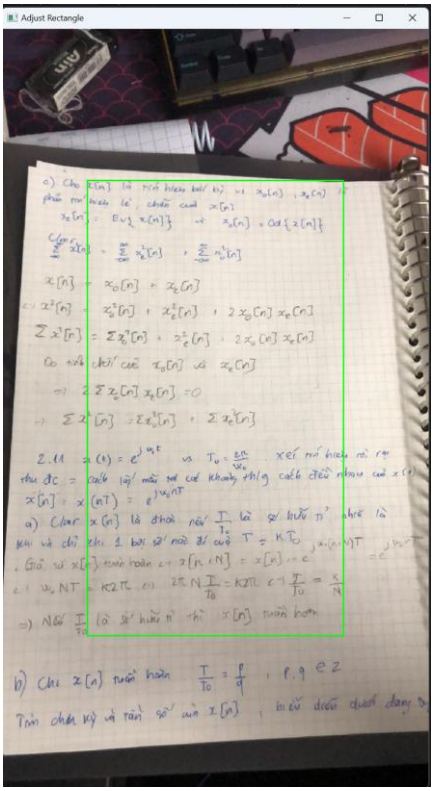
kiểm soát chính xác hơn quá trình phát hiện và xử lý tài liệu. Điều này rất hữu ích khi tài liệu có góc chụp không chuẩn hoặc có hình dạng phức tạp.

a) apply_clahe(image, clip_limit, tile_grid_size):

Hàm này áp dụng phương pháp tăng cường tương phản bằng kỹ thuật CLAHE (Contrast Limited Adaptive Histogram Equalization). "cv2.createCLAHE" tạo đối tượng CLAHE với các tham số clip_limit (giới hạn độ tương phản) và tile_grid_size (kích thước ô lưới). Hàm này rất hữu ích trong việc cải thiện độ tương phản của hình ảnh, đặc biệt là với các ảnh có độ sáng không đồng đều.

b) draw_rectangle(image, pts):

Hàm này vẽ một hình chữ nhật dựa trên các điểm pts (được xác định bởi người dùng) bằng cách sử dụng cv2.polylines. Hình chữ nhật này đánh dấu vùng mà người dùng muốn chọn làm tài liệu cần quét.



Hình 2.2.2-1:Hình chữ nhật màu xanh do hàm draw_rectangle(image, pts) tạo ra.

c) window_popup(title, image):

Hàm này mở một cửa sổ pop-up để hiển thị ảnh với tiêu đề title. Hàm này cũng thực hiện tính toán để đảm bảo ảnh được hiển thị với kích thước tối ưu trên màn hình, sử dụng cv2.imshow để hiển thị và cv2.resizeWindow để thay đổi kích thước cửa sổ nếu cần.

d) mouse_handler(event, x, y, flags, param):

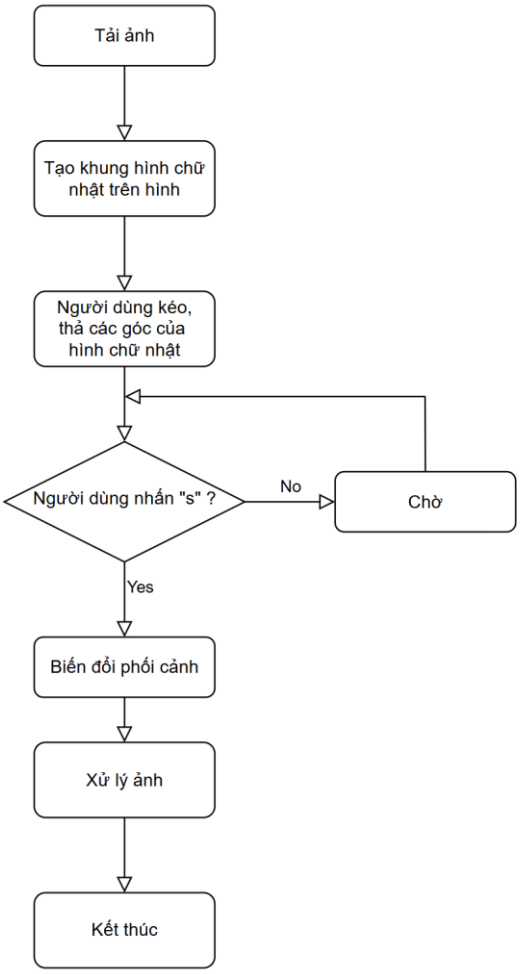
Đây là hàm xử lý sự kiện chuột, cho phép người dùng kéo và thả các điểm góc của tài liệu. Hàm này sẽ cập nhật vị trí các điểm góc khi người dùng di chuyển chuột, và sau đó vẽ lại hình chữ nhật trên ảnh để phản ánh những thay đổi này.

e) scan(image):

Hàm này thực hiện việc chuyển đổi ảnh sang thang độ xám (cv2.cvtColor), làm sắc nét ảnh bằng kỹ thuật GaussianBlur và áp dụng ngưỡng thích nghi (cv2.adaptiveThreshold). Những bước này giúp tăng cường độ tương phản và làm nổi bật tài liệu trong ảnh, chuẩn bị cho quá trình biến đổi phối cảnh.

f) process_image():

Hàm có chức năng chính là tích hợp các chức năng khác để thực hiện toàn bộ quá trình phát hiện và quét tài liệu thủ công. Nó quản lý việc hiển thị các điểm góc, xử lý sự kiện chuột, và cuối cùng là thực hiện biến đổi phối cảnh để tạo ra ảnh tài liệu đã được quét.



Hình 2.2.2-2: Lưu đồ giải thuật chính của file Manual_Detect.py

2.2.3 Auto_detect.py: Phát hiện tài liệu tự động

```
import cv2
import numpy as np
import GUI
import random
class AutoDetect:
    def __init__(self, image_path):
        self.pathImage = image_path
        self.heightImg = 640
        self.widthImg = 480
    def process_image(self):
        GUI.initializeTrackbars()
        img = cv2.imread(self.pathImage)

        while True:
            img = cv2.resize(img, (self.widthImg,
self.heightImg))
            imgBlank = np.zeros((self.heightImg,
self.widthImg, 3), np.uint8)
            imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            imgBlur = cv2.GaussianBlur(imgGray, (5, 5), 3)

            height, width = img.shape[:2]
            imgRectangle = cv2.rectangle(imgBlur.copy(), (0,
0), (width - 1, height - 1), (255, 255, 255), 2)

            thres = GUI.valTrackbars()
            imgThreshold = cv2.Canny(imgRectangle, thres[0],
thres[1])
            kernel = np.ones((5, 5))
            imgDial = cv2.dilate(imgThreshold, kernel,
iterations=2)
            imgThreshold = cv2.erode(imgDial, kernel,
iterations=1)

            imgContours = img.copy()
            imgBigContour = img.copy()
            contours, _ = cv2.findContours(imgThreshold,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            contours = sorted(contours, key=cv2.contourArea,
reverse=True)

            for contour in contours:
                epsilon = 0.02 * cv2.arcLength(contour, True)
                contour = cv2.approxPolyDP(contour, epsilon,
True)
                color = [random.randint(0, 255) for _ in
range(3)]
                cv2.drawContours(imgContours, [contour], -1,
color, 2)

            biggest, maxArea = GUI.biggestContour(contours)
            if biggest.size != 0:
                biggest = GUI.reorder(biggest)
                cv2.drawContours(imgBigContour, biggest, -1,
(0, 255, 0), 20)
                imgBigContour =
GUI.drawRectangle(imgBigContour, biggest, 2)
                pts1 = np.float32(biggest)
                pts2 = np.float32([[0, 0], [self.widthImg, 0],
[0, self.heightImg], [self.widthImg, self.heightImg]])
                matrix = cv2.getPerspectiveTransform(pts1,
pts2)
```

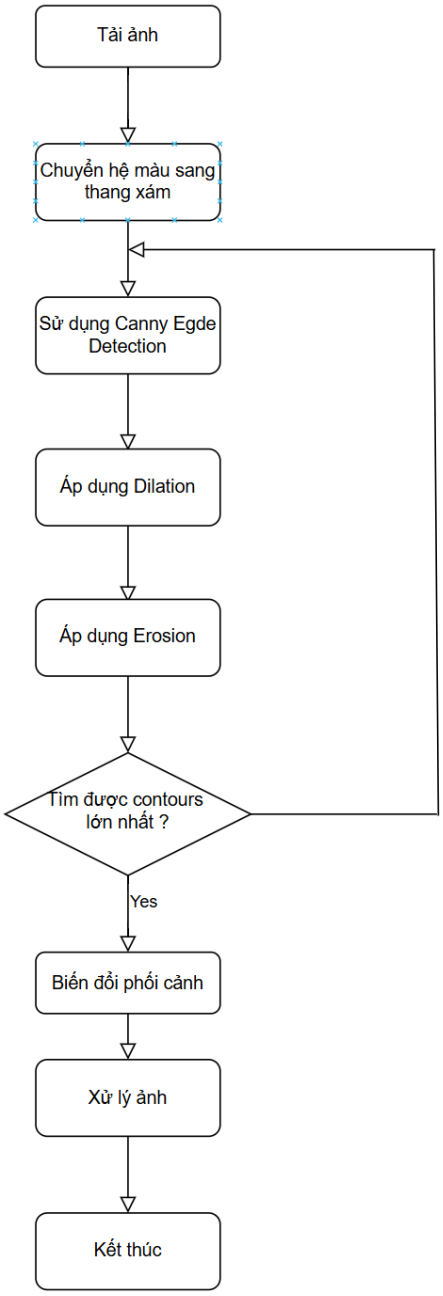
```
                imgWarpColored = cv2.warpPerspective(img,
matrix, (self.widthImg, self.heightImg))

                imgWarpColored =
imgWarpColored[20:imgWarpColored.shape[0] - 20,
20:imgWarpColored.shape[1] - 20]
                imgWarpColored = cv2.resize(imgWarpColored,
(self.widthImg, self.heightImg))

                imgWarpGray = cv2.cvtColor(imgWarpColored,
cv2.COLOR_BGR2GRAY)
                imgAdaptiveThre =
cv2.adaptiveThreshold(imgWarpGray, 255, 1, 1, 7, 2)
                imgAdaptiveThre =
cv2.bitwise_not(imgAdaptiveThre)
                imgAdaptiveThre =
cv2.medianBlur(imgAdaptiveThre, 3)

                imageArray = ([img, imgThreshold],
                              [imgContours, imgBigContour])

            else:
                imageArray = ([img, imgThreshold],
                              [imgContours, imgBigContour])

            labels = [["Original", "Threshold", "Contours"],
                      ["Contours", "Biggest Contour"]]

            stackedImage = GUI.stackImages(imageArray, 0.75,
labels)
            cv2.imshow("Result", stackedImage)

            if cv2.waitKey(0) & 0xFF == ord('s'):
                cv2.imshow('Auto_Detect', imgAdaptiveThre)
                cv2.waitKey(300)
                break
```

Bảng 2.2.3-1: Mã nguồn của file Auto_Detect.py

File Auto_detect.py chứa các chức năng tự động phát hiện tài liệu trong ảnh mà không cần sự can thiệp thủ công của người dùng. Đây là chế độ dễ sử dụng nhất trong phần mềm.

a) process_image()

Đây là hàm chính trong tệp Auto_detect.py, chịu trách nhiệm toàn bộ quy trình quét tài liệu tự động. Hàm bắt đầu bằng cách đọc ảnh từ đường dẫn image_path (cv2.imread) và thay đổi kích thước ảnh về widthImg và heightImg (cv2.resize). Tiếp theo, ảnh được chuyển sang thang độ xám (cv2.cvtColor), làm mờ để giảm nhiễu (cv2.GaussianBlur), và tạo ra một bản sao với khung chữ nhật xung quanh (cv2.rectangle). Sau đó, thuật toán phát hiện cạnh (cv2.Canny) được áp dụng với các giá trị ngưỡng từ thanh trượt (trackbars) để tìm kiếm các cạnh trong ảnh.



Hình 2.2.3-1: Giải thuật chính của file Auto_Detect.py

b) findContours()

Hàm này sử dụng cv2.findContours để tìm tất cả các đường viền trong ảnh nhị phân (ảnh sau khi phát hiện cạnh). Các đường viền sau đó được sắp xếp theo diện tích (cv2.contourArea) để chọn ra đường viền lớn nhất (tài liệu).

c) biggestContour(contours)

Hàm này tìm và trả về đường viền lớn nhất trong danh sách contours được tìm thấy, đây thường là đường viền của tài liệu. Nếu tìm thấy, đường viền này sẽ

được sắp xếp lại thứ tự các điểm góc để đảm bảo rằng thứ tự của chúng luôn là trên trái, trên phải, dưới phải, và dưới trái.

d) drawRectangle()

Hàm này vẽ lại hình chữ nhật bao quanh tài liệu dựa trên các điểm góc đã được sắp xếp, sau đó thực hiện biến đổi phối cảnh (cv2.getPerspectiveTransform) để cắt và căn chỉnh tài liệu về một hình chữ nhật chuẩn.

e) cv2.warpPerspective()

Hàm này áp dụng biến đổi phối cảnh lên ảnh, giúp căn chỉnh tài liệu sao cho nó thẳng đứng và dễ dàng đọc hơn. Phần thừa xung quanh tài liệu được cắt bỏ để kết quả cuối cùng gọn gàng và tập trung vào tài liệu chính.

f) cv2.adaptiveThreshold()

Ảnh sau khi biến đổi phối cảnh sẽ được chuyển về thang độ xám và áp dụng ngưỡng thích nghi để tăng cường độ tương phản, giúp tài liệu rõ nét hơn. Kết quả là một ảnh nhị phân, chỉ gồm hai màu đen trắng, lý tưởng cho việc lưu trữ hoặc in ấn.

2.2.4 Main.py: Tích hợp và thực hiện quá trình quét tài liệu

```
import tkinter as tk
from tkinter import filedialog
from Manual_Detect import ManualDetect
from Auto_Detect import AutoDetect


def browse_image():
    filename = filedialog.askopenfilename(initialdir="/",
title="Select an Image",
                                          filetypes=(("Image
files", "*.jpg;*.jpeg;*.png"), ("all files", "*.*")))
    image_path.set(filename)


def run_manual_detect():
    image_path_value = image_path.get()
    if image_path_value:
        global manual_detect
        manual_detect = ManualDetect(image_path_value)
        manual_detect.process_image()


def run_auto_detect():
    image_path_value = image_path.get()
    if image_path_value:
        global auto_detect
        auto_detect = AutoDetect(image_path_value)
        auto_detect.process_image()

def quit_app():
    root.quit()

if __name__ == '__main__':
```

```
    # main()
    global image_path, root
    root = tk.Tk()
    root.title("Image Detection")

    image_path = tk.StringVar()

    tk.Label(root, text="Select Image").pack(pady=10)
    tk.Entry(root, textvariable=image_path,
width=50).pack(pady=10)
    tk.Button(root, text="Browse",
command=browse_image).pack(pady=10)

    button_frame = tk.Frame(root)
    button_frame.pack(pady=20)

    tk.Button(button_frame, text="Manual",
command=run_manual_detect).grid(row=0, column=0, padx=(0, 20))
    tk.Button(button_frame, text="Auto",
command=run_auto_detect).grid(row=0, column=1, padx=20)
    tk.Button(button_frame, text="Quit",
command=quit_app).grid(row=0, column=2, padx=(20, 0))
    root.mainloop()
```

Bảng 2.2.4-1: Mã nguồn của file main.py

Tệp Main.py đóng vai trò là tệp điều khiển chính, nơi tất cả các chức năng từ các tệp khác được tích hợp để tạo ra một quy trình quét tài liệu hoàn chỉnh. Giao diện người dùng được xây dựng bằng Tkinter, cho phép người dùng lựa chọn giữa chế độ quét thủ công và tự động.

a) quit_app()

Hàm có chức năng để thoát khỏi ứng dụng, đóng tất cả các cửa sổ và kết thúc quá trình.

b) run_manual_detect()

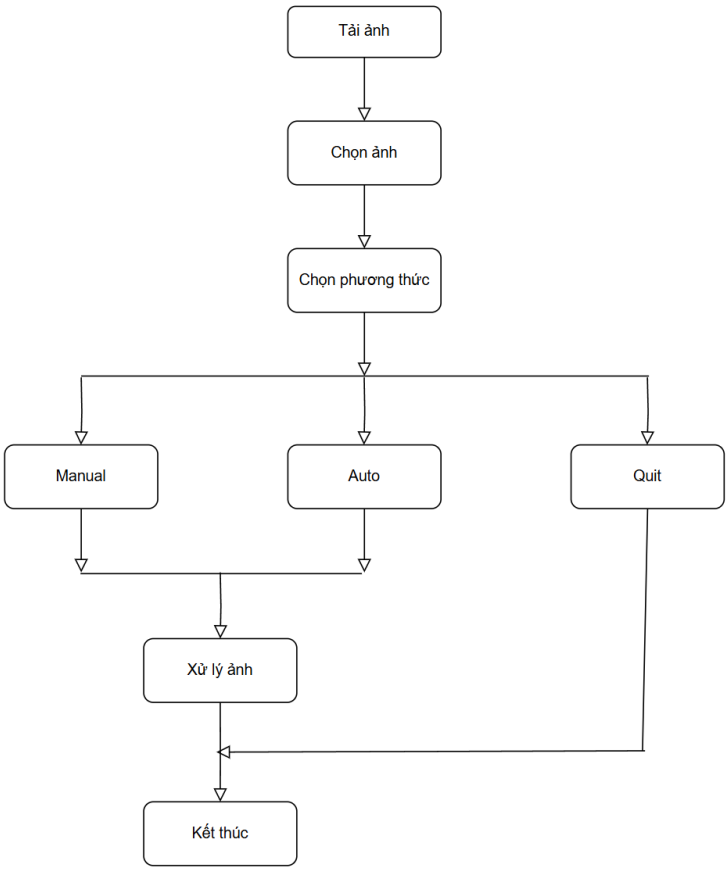
Hàm khởi chạy quy trình phát hiện tài liệu thủ công. Khi người dùng chọn phương pháp thủ công, chức năng này sẽ gọi ManualDetect và thực thi process_image() để bắt đầu quá trình phát hiện và xử lý tài liệu..

c) run_auto_detect()

Tương tự, run_auto_detect() khởi chạy quy trình phát hiện tài liệu tự động bằng cách gọi AutoDetect và thực thi process_image(). Quá trình này hoàn toàn tự động và không cần sự can thiệp của người dùng.

d) browse_image()

Hàm này cho phép người mở hộp thoại để người dùng chọn một file ảnh từ hệ thống tệp. Đường dẫn của ảnh được chọn sẽ được lưu trữ để sử dụng trong các bước xử lý tiếp theo.



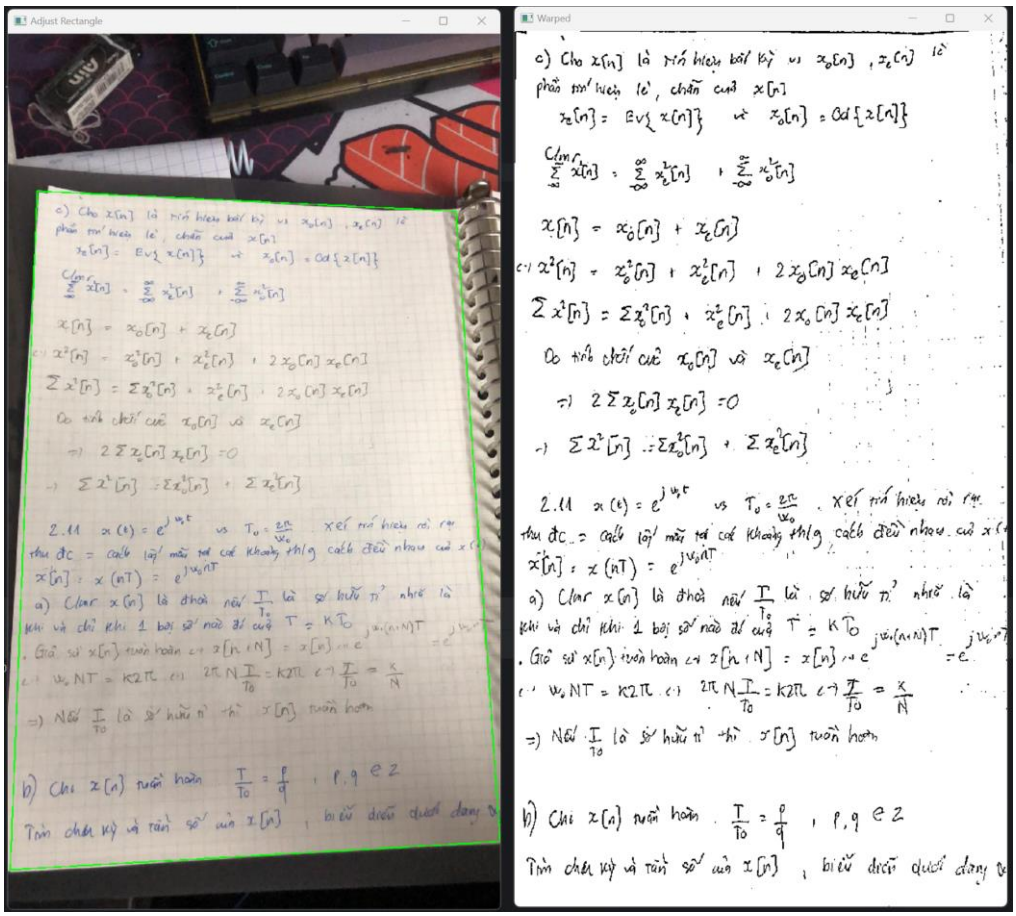
Hình 2.2.4-1:Giải thuật chính của file main.py

2.3 Kết quả

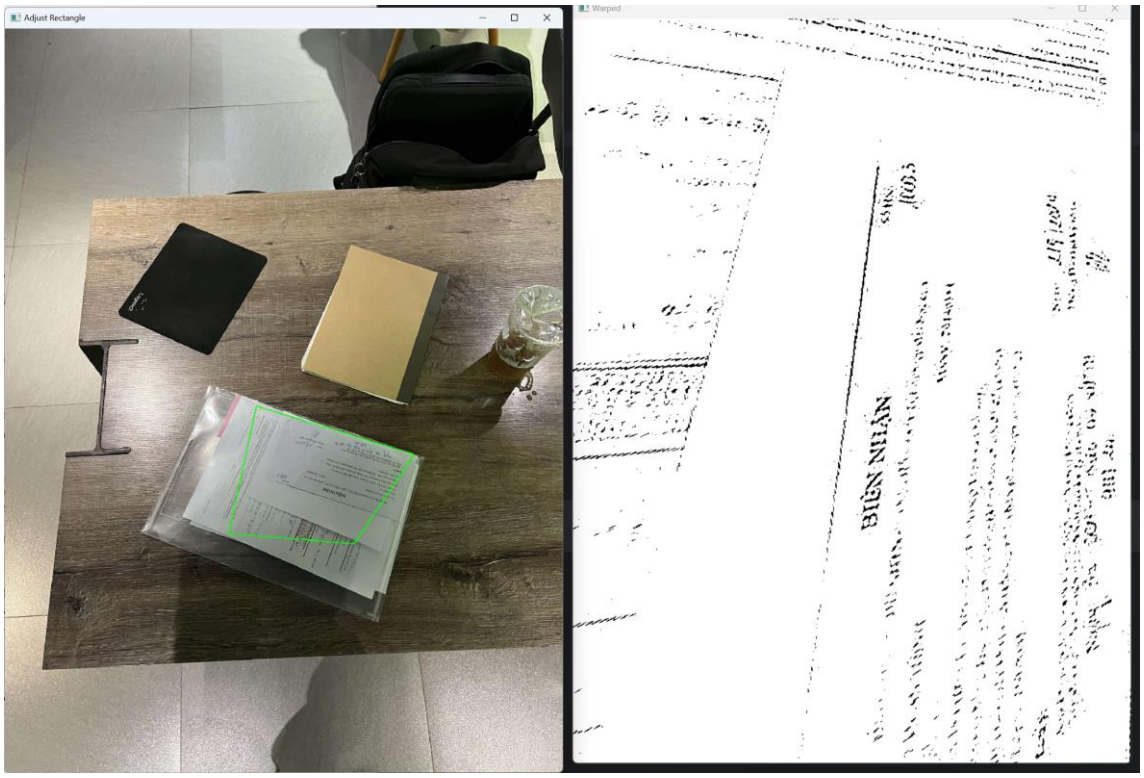
2.3.1 Trình bày kết quả từ phần mềm quét tài liệu.

a) Phương pháp thủ công

Kết quả cho thấy người dùng có thể dễ dàng xác định và điều chỉnh các điểm góc của tài liệu. Điều này giúp tối ưu hóa kết quả biến đổi phối cảnh (perspective transform) và đảm bảo tài liệu được quét với chất lượng cao. Đặc biệt, phương pháp này cho phép xử lý những hình ảnh có tài liệu không nằm trong một hình dạng cố định hoặc có các góc chụp không chuẩn, chẳng hạn như bị méo hoặc nghiêng.



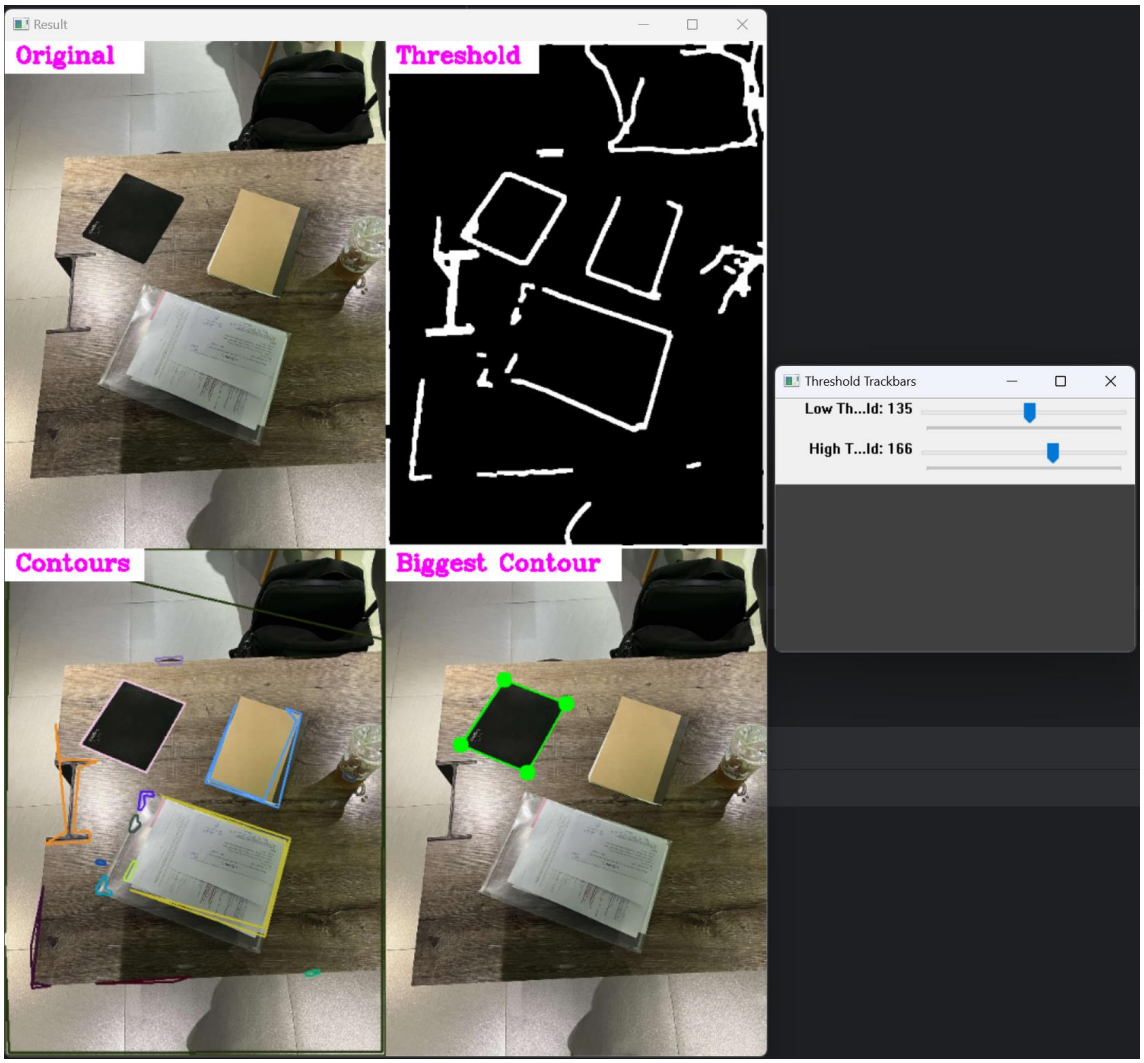
Hình 2.3.1-1: Minh họa kết quả quét hoàn chỉnh đối với ảnh có góc chụp, tỉ lệ hình "chuẩn"



Hình 2.3.1-2: Minh họa kết quả quét hoàn chỉnh đối với ảnh có góc chụp, tỉ lệ hình không "chuẩn"

b) Phương pháp tự động

Phương pháp tự động phát hiện tài liệu cho kết quả tốt trong các tình huống khi tài liệu có biên rõ ràng và không bị che khuất hoặc lẫn với các chi tiết nền phức tạp hay do giá trị ngưỡng phát hiện cạnh không hợp lý. Trong các trường hợp lý tưởng, quá trình tự động phát hiện diễn ra nhanh chóng và cho ra kết quả chính xác mà không cần sự can thiệp của người dùng.



Hình 2.3.1-3:Minh họa kết quả quét hoàn chỉnh đối với ảnh xác định không đúng do chi tiết nền phúc tạp, giá trị ngưỡng phát hiện chưa tối ưu

2.3.2 So sánh phương pháp phát hiện thủ công và tự động

a) Ưu điểm và hạn chế của phương pháp thủ công

Ưu điểm:

- **Chính xác:** Người dùng có thể can thiệp trực tiếp, đảm bảo các điểm góc được chọn đúng vị trí.

- **Linh hoạt:** Phù hợp với các hình ảnh có nhiều chi tiết phức tạp hoặc có góc chụp không chuẩn.

Hạn chế:

- **Tốn thời gian:** Yêu cầu sự can thiệp của người dùng, có thể mất thời gian khi xử lý nhiều tài liệu.
- **Đòi hỏi kỹ năng:** Người dùng cần phải có kiến thức cơ bản về thao tác hình ảnh để chọn đúng các điểm góc.

b) Ưu điểm và hạn chế của phương pháp tự động

Ưu điểm:

- **Nhanh chóng:** Tự động hoàn toàn, không yêu cầu sự can thiệp của người dùng, tiết kiệm thời gian đáng kể.
- **Dễ sử dụng:** Phù hợp với người dùng không có nhiều kinh nghiệm xử lý ảnh.

Hạn chế:

- **Độ chính xác không cao trong một số trường hợp hình ảnh có quá nhiều chi tiết cạnh**: Khi tài liệu chứa quá nhiều chi tiết hoặc có nền phức tạp, hàm cv2.findContours() có thể phát hiện nhầm nhiều cạnh thừa. Điều này dẫn đến việc phát hiện sai hình dạng tài liệu, làm giảm chất lượng của quá trình quét.
- **Đường viền không chuẩn:** Một số trường hợp, hàm cv2.findContours() không thể phát hiện chính xác các cạnh của tài liệu nếu đường viền bị mờ hoặc bị nhiễu bởi các yếu tố xung quanh. Điều này dẫn đến kết quả phát hiện không chính xác hoặc không tìm thấy tài liệu.
- **Hình ảnh gốc không chuẩn:** Nếu ảnh đầu vào có góc chụp nghiêng, bị méo hoặc có sự tương phản không tốt giữa tài liệu và nền, hệ thống tự động khó xác định chính xác các cạnh của tài liệu, dẫn đến kết quả sai lệch hoặc không đầy đủ.

2.4 Một số khó khăn và giải pháp

Quá trình tự động phát hiện tài liệu gặp phải nhiều khó khăn và thách thức, bao gồm:

- **Nhiễu và các chi tiết cạnh thừa:** Trong nhiều trường hợp, hình ảnh chứa quá nhiều chi tiết không liên quan, gây nhiễu cho quá trình phát hiện tài

liệu. Các chi tiết này có thể bao gồm văn bản, hình ảnh khác, hoặc các đường viền nhỏ không phải của tài liệu cần quét. Điều này khiến hàm cv2.findContours() nhận diện sai, dẫn đến việc phát hiện nhầm các đường viền không thuộc về tài liệu.

- **Hạn chế của hàm cv2.findContours():** Mặc dù cv2.findContours() là công cụ mạnh mẽ trong việc phát hiện các đường viền, nhưng nó vẫn có những hạn chế nhất định. Hàm này không phải lúc nào cũng chính xác trong việc xác định các đường viền phức tạp, đặc biệt là khi các cạnh của tài liệu không rõ ràng hoặc bị nhiễu. Điều này dẫn đến việc chọn sai đường viền hoặc bỏ sót các cạnh quan trọng.
- **Chất lượng hình ảnh đầu vào:** Nếu hình ảnh gốc có chất lượng kém (ví dụ như bị mờ, nhiễu, hoặc không đủ sáng), các thuật toán xử lý ảnh sẽ khó khăn hơn trong việc phát hiện chính xác các cạnh của tài liệu. Các tình huống như góc chụp không chuẩn hoặc tài liệu bị che khuất một phần cũng làm giảm độ chính xác của phương pháp tự động.

Để cải thiện khả năng tự động phát hiện tài liệu, một số giải pháp có thể được xem xét:

- **Tiền xử lý ảnh:** Cải thiện bước tiền xử lý bằng cách áp dụng các kỹ thuật như tăng cường độ tương phản, lọc nhiễu, hoặc sử dụng các thuật toán phát hiện cạnh mạnh hơn (chẳng hạn như Canny Edge Detector) để làm rõ các đường viền tài liệu.
- **Sử dụng các phương pháp phát hiện cạnh tiên tiến hơn:** Kết hợp với các kỹ thuật học máy hoặc deep learning để tự động học và phát hiện tài liệu một cách thông minh hơn, đặc biệt là trong các trường hợp hình ảnh có nhiều chi tiết nhiễu hoặc không chuẩn.
- **Tối ưu hóa hàm cv2.findContours():** Nghiên cứu và áp dụng các kỹ thuật xử lý hậu kỳ để lọc ra các đường viền không mong muốn và cải thiện việc phát hiện chính xác các cạnh tài liệu.

CHƯƠNG 3: TÀI LIỆU THAM KHẢO

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed., Prentice-Hall, 2008.

[2] Andrew Campbell, *OpenCV Document Scanner*, Truy cập từ: https://github.com/andrewdcampbell/OpenCV-Document-Scanner

[3] Stack Overflow, *Algorithm to detect corners of paper sheet in photo*, Truy cập từ: https://stackoverflow.com/questions/6555629/algorithm-to-detect-corners-of-paper-sheet-in-photo

[4] Geeks for Geeks, *Python | Detect Polygons in an Image using OpenCV*, Truy cập từ: https://www.geeksforgeeks.org/python-detect-polygons-in-an-image-using-opencv/

[5] Tutorials Point, *How to detect polygons in image using OpenCV Python?* Truy cập từ: https://www.tutorialspoint.com/how-to-detect-polygons-in-image-using-opencv-python

[6] Stack Overflow, *Finding all polygons in an image OpenCV and Python*, Truy cập từ: https://stackoverflow.com/questions/68725183/finding-all-polygons-in-an-image-opencv-and-python