ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC HÀNH 01

Môn: Thị Giác Máy Tính Nâng Cao

CÁC TOÁN TỬ HÌNH THÁI HỌC

Giảng viên hướng dẫn:
PGS.TS. Lý Quốc Ngọc
ThS. Nguyễn Mạnh Hùng

Sinh viên thực hiện:
Trần Nguyên Huân - 21127050

Tháng 7, 2024 – Thành phố Hồ Chí Minh

`

Mục lục

1. Bảng đánh giá kết quả công việc 3
2. Công cụ, phiên bản, và môi trường sử dụng 4
3. Các toán tử hình thái học trên ảnh nhị phân 4
3.1. Erosion 4
3.2. Dilation 5
3.3. Opening 6
3.4. Closing 6
3.5. Hit-or-miss 7
3.6. Boundary Extraction 8
3.7. Hole Filling 9
3.8. Extraction of Connected Components 9
3.9. Convex Hull 10
3.10. Thinning 12
3.11. Thickening 13
3.12. Skeleton 14
3.13. Pruning 15
3.14. Morphological Reconstruction 16
4. Các toán tử hình thái học trên ảnh xám 18
4.1. Erosion 18
4.2. Dilation 19
4.3. Opening 20
4.4. Closing 20
4.5. Smoothing 21
4.6. Gradient 21
4.7. Top-hat transformation 22
4.8. Black-hat Transformation 22
4.9. Granulometry 23
4.10. Mophological Reconstruction 24
5. Nguồn tham khảo 26

`

1. Bảng đánh giá kết quả công việc

Loại ảnh	Toán tử	Kết quả thực hiện
Ảnh nhị phân	Erosion	100%
	Dilation	100%
	Opening	100%
	Closing	100%
	Hit-or-miss	100%
	Boundary Extraction	100%
	Hole Filling	100%
	Extraction of Connected Components	100%
	Convex Hull	100%
	Thinning	95%
	Thickening	95%
	Skeleton	100%
	Pruning	100%
	Mophological Reconstruction	95%

`

Ảnh xám	Erosion	100%
	Dilation	100%
	Opening	100%
	Smoothing	100%
	Gradient	100%
	Top-hat transformation	100%
	Bottom-hat Transformation	100%
	Granulometry	95%
	Mophological Reconstruction	95%

2. Công cụ, phiên bản, và môi trường sử dụng

- **Phiên bản Python sử dụng:** Python 3.11.7
- **IDE / Text Editor:** Visual Studio Code (VSCode)
- **Thư viện sử dụng:** NumPy, OpenCV

3. Các toán tử hình thái học trên ảnh nhị phân

3.1.Erosion

Ý tưởng thuật toán:

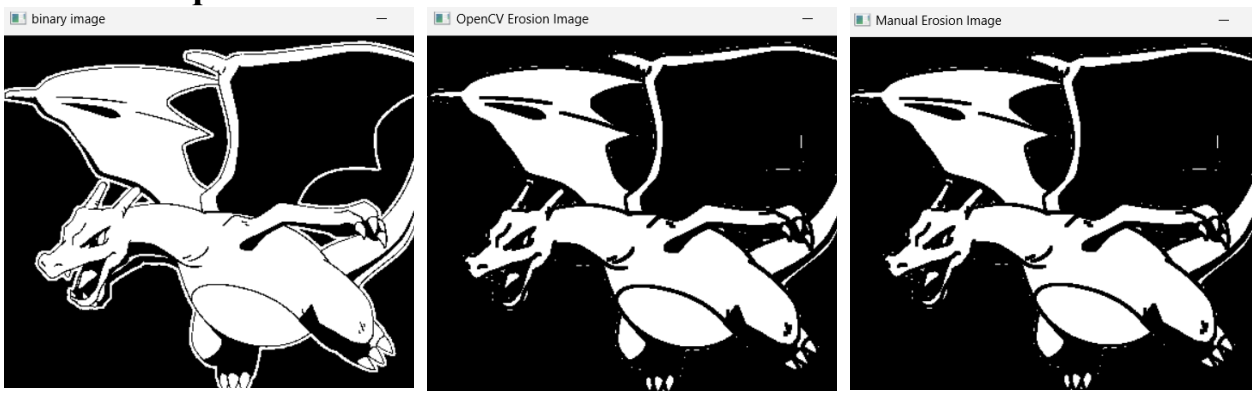
1. Kernel và tâm của Kernel: Xác định tâm của kernel (một ma trận con) và đếm số lượng pixel có giá trị 1 trong kernel.
2. Tạo ảnh mới: Tạo một ảnh mới với kích thước lớn hơn ảnh gốc để xử lý biên.
3. Mở rộng ảnh gốc: Thêm các cột và hàng chứa giá trị 0 vào cạnh phải và cạnh dưới của ảnh gốc để đảm bảo kernel có thể di chuyển qua toàn bộ ảnh.
4. Quét qua ảnh gốc: Duyệt qua từng pixel của ảnh gốc và áp dụng kernel:
5. Nếu tất cả các pixel dưới kernel đều là 1, pixel tại tâm của kernel trong ảnh

mới được thiết lập thành 255.

6. Trả về ảnh kết quả: Cắt bỏ phần mở rộng của ảnh mới để trả về ảnh có cùng kích thước với ảnh gốc.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **erode** -t <wait key time>

So sánh kết quả:



Ảnh nhị phân gốc | OpenCV | Thủ công

Nhận xét: Kết quả chính xác tương tự OpenCV.

3.2.Dilation

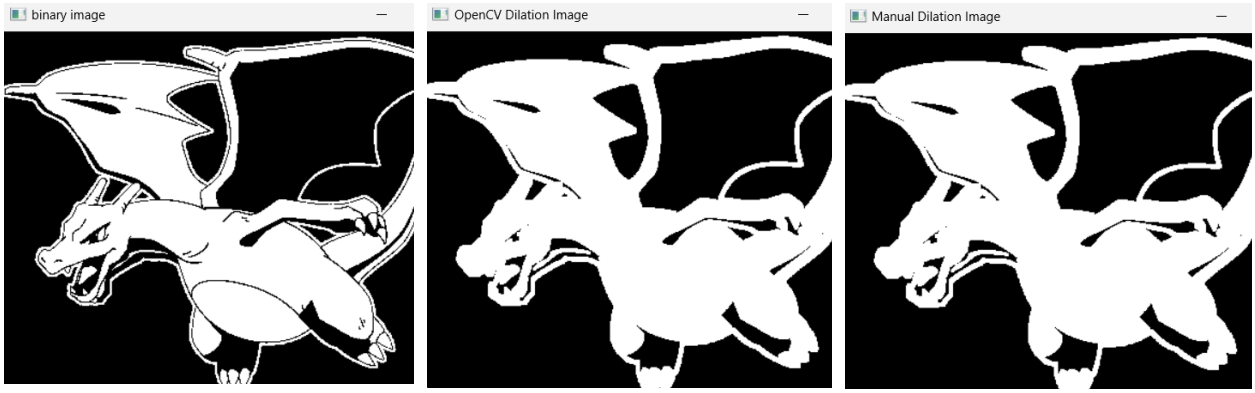
Ý tưởng thuật toán:

1. Kernel và tâm của Kernel: Xác định tâm của kernel (một ma trận con) và đếm số lượng pixel có giá trị 1 trong kernel.
2. Tạo ảnh mới: Tạo một ảnh mới với kích thước lớn hơn ảnh gốc để xử lý biên.
3. Mở rộng ảnh gốc: Thêm các cột và hàng chứa giá trị 0 vào cạnh phải và cạnh dưới của ảnh gốc để đảm bảo kernel có thể di chuyển qua toàn bộ ảnh.
4. Quét qua ảnh gốc: Duyệt qua từng pixel của ảnh gốc và áp dụng kernel:
5. Nếu pixel tại tâm của kernel trong ảnh gốc có giá trị 255, tất cả các pixel dưới kernel trong ảnh mới được thiết lập thành 255.
6. Trả về ảnh kết quả: Cắt bỏ phần mở rộng của ảnh mới để trả về ảnh có cùng kích thước với ảnh gốc.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **dilate** -t <wait key time>

So sánh kết quả:

`



Ảnh nhị phân gốc OpenCV Thủ công

Nhận xét: Kết quả chính xác tương tự OpenCV.

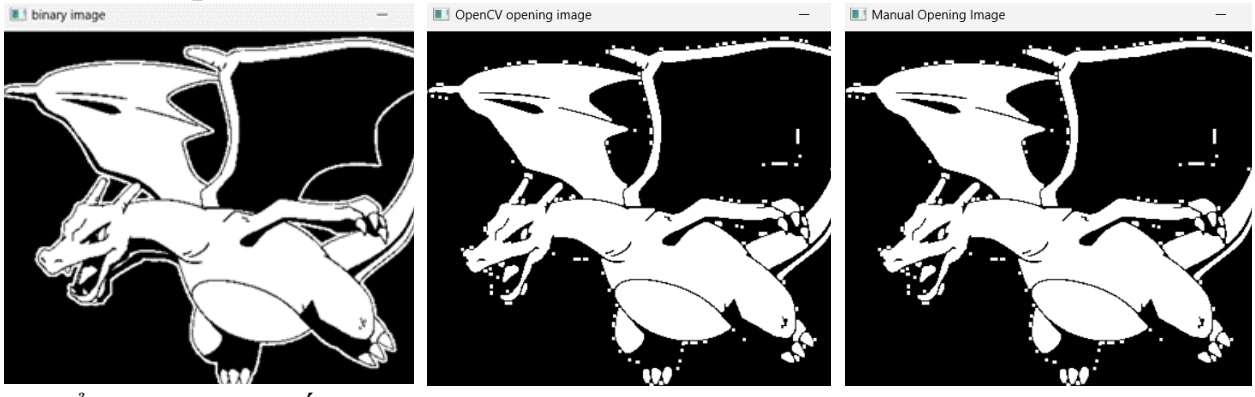
3.3.Opening

Ý tưởng thuật toán:

1. Erode: Hàm erode (xói mòn) làm giảm kích thước của các vùng sáng (foreground) trong ảnh, giúp loại bỏ các điểm nhiễu nhỏ và tách rời các phần nhỏ khỏi đối tượng chính.
2. Dilate: Hàm dilate (giãn nở) được áp dụng sau khi xói mòn để làm tăng kích thước của các vùng sáng, khôi phục lại kích thước ban đầu của các đối tượng nhưng vẫn giữ lại hiệu quả làm sạch nhiễu từ bước xói mòn.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **opening** -t <wait key time>

So sánh kết quả:



Ảnh nhị phân gốc OpenCV Thủ công

Nhận xét: Kết quả chính xác tương tự OpenCV.

3.4.Closing

Ý tưởng thuật toán:

1. Dilate: Hàm dilate (giãn nở) làm tăng kích thước của các vùng sáng

(foreground) trong ảnh, giúp nối các vùng sáng bị đứt đoạn và lấp đầy các lỗ hổng nhỏ.

2. Erode: Hàm erode (xói mòn) được áp dụng sau khi giãn nở để làm giảm kích thước của các vùng sáng, khôi phục lại hình dạng ban đầu của các đối tượng nhưng vẫn giữ lại hiệu quả nối và lấp đầy từ bước giãn nở.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **closing** -t <wait key time>

So sánh kết quả:

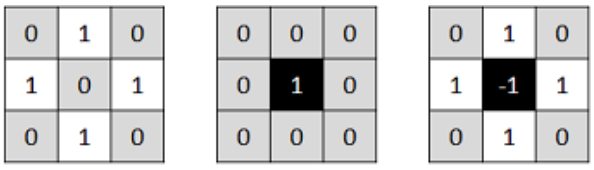
Ảnh nhị phân gốc | OpenCV | Thủ công

Nhận xét: Kết quả chính xác tương tự OpenCV.

3.5. Hit-or-miss

Quy tắc: Kernel truyền vào được tạo sinh theo quy tắc như hình vẽ (final combined kernel). Sau đó hàm hitmiss sẽ phân tách kernel này thành kernel hit và kernel miss.

Nguồn: opencv



Structuring elements (kernels). Left: kernel to 'hit'. Middle: kernel to 'miss'. Right: final combined kernel

Ý tưởng thuật toán:

1. Tách kernel thành hit và miss:
   - kernel_hit: Bao gồm các vị trí của kernel có giá trị bằng 1.
   - kernel_miss: Bao gồm các vị trí của kernel có giá trị bằng -1.
2. Erode với hit kernel:
   - Hàm erode được áp dụng trên ảnh với kernel_hit để tìm các vị trí trúng (hit).
3. Erode với miss kernel:
   - Hàm erode được áp dụng trên ảnh nghịch đảo (255 - img) với kernel_miss để

tìm các vị trí trượt (miss).

4. Kết hợp kết quả:

- Dùng phép toán bitwise_and để kết hợp kết quả từ hai bước xói mòn (erode). Chỉ những điểm nào trúng cả hai điều kiện hit và miss mới được giữ lại.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **hit_or_miss** -t <wait key time>

So sánh kết quả:

Ảnh nhị phân gốc | OpenCV | Thủ công

Nhận xét: Kết quả chính xác tương tự OpenCV.

3.6.Boundary Extraction

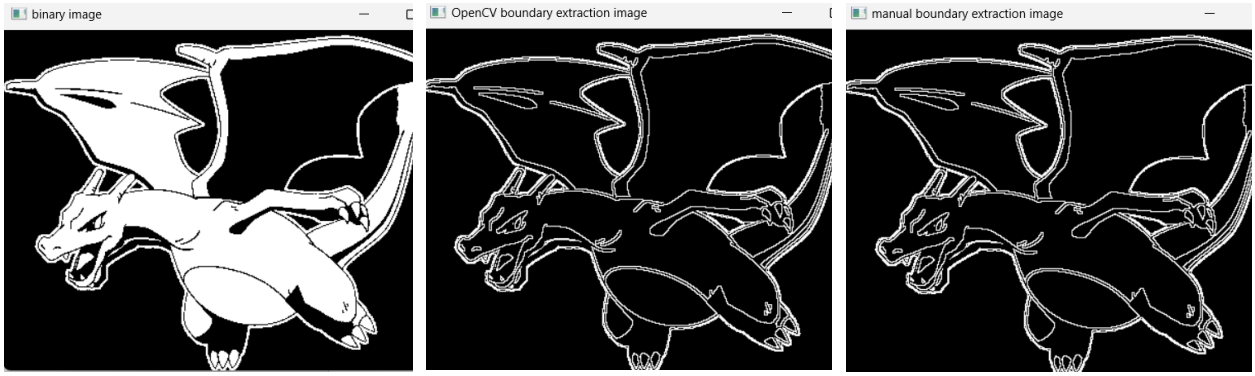
Ý tưởng thuật toán:

1. Xói mòn ảnh gốc: Hàm erode được áp dụng trên ảnh gốc với một kernel để làm giảm kích thước của các vùng sáng trong ảnh.
2. Tìm hiệu giữa ảnh gốc và ảnh đã xói mòn: Lấy hiệu giữa ảnh gốc và ảnh đã xói mòn để xác định các điểm thuộc đường biên. Những điểm này là những pixel bị loại bỏ trong quá trình xói mòn.
3. Kết quả là các đường biên: Kết quả cuối cùng là các đường biên được nhân với 255.0 để đảm bảo các giá trị pixel là 255 (trắng) cho các đường biên và 0 (đen) cho các điểm khác.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **extract_boundary** -t <wait key time>

So sánh kết quả:

`



Ảnh nhị phân gốc | OpenCV | Thủ công

Nhận xét: Kết quả chính xác tương tự OpenCV.

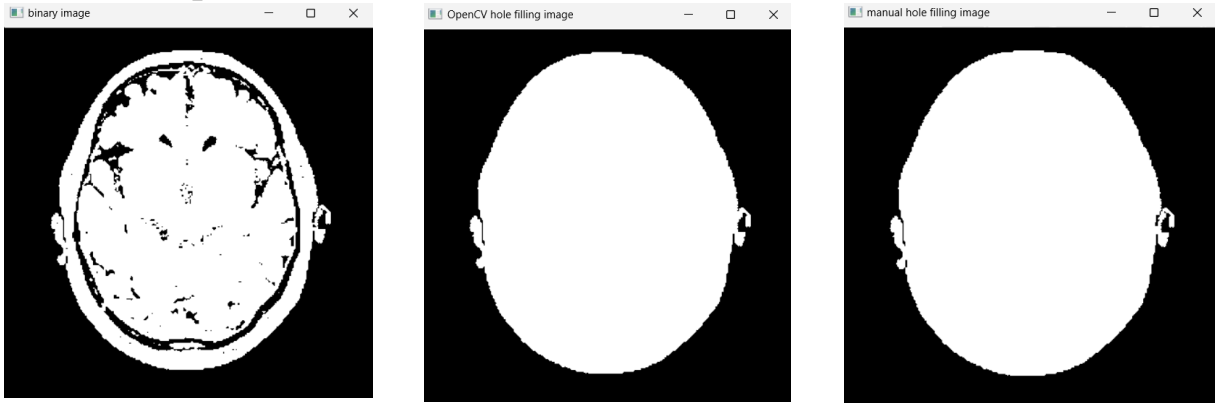
3.7.Hole Filling

Ý tưởng thuật toán:

1. Xói mòn ảnh gốc: Hàm erode được áp dụng trên ảnh gốc với một kernel để làm giảm kích thước của các vùng sáng trong ảnh.
2. Tìm hiệu giữa ảnh gốc và ảnh đã xói mòn: Lấy hiệu giữa ảnh gốc và ảnh đã xói mòn để xác định các điểm thuộc đường biên. Những điểm này là những pixel bị loại bỏ trong quá trình xói mòn.
3. Kết quả là các đường biên: Kết quả cuối cùng là các đường biên được nhân với 255.0 để đảm bảo các giá trị pixel là 255 (trắng) cho các đường biên và 0 (đen) cho các điểm khác.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **fillhole** -t <wait key time>

So sánh kết quả:



Ảnh nhị phân gốc | OpenCV | Thủ công

Nhận xét: Kết quả chính xác tương tự OpenCV.

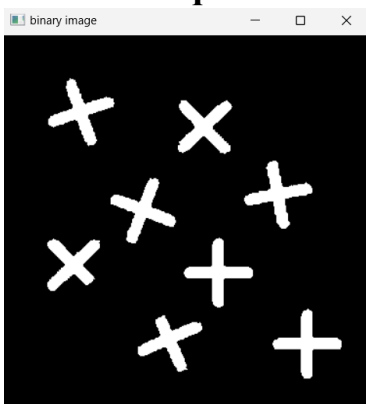
3.8.Extraction of Connected Components

Ý tưởng thuật toán:

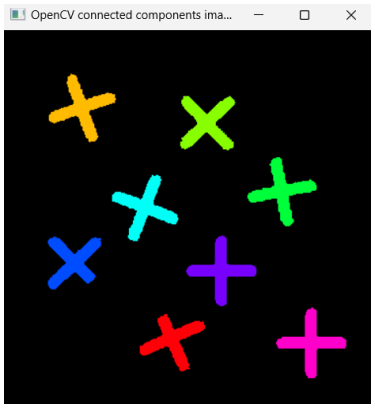
1. Tạo ma trận nhãn: Khởi tạo ma trận nhãn với cùng kích thước với ảnh gốc để lưu trữ nhãn của các thành phần liên thông.
2. Lặp qua ảnh gốc: Lặp cho đến khi tất cả các pixel trong ảnh tạm thời trở thành 0.
3. Tìm điểm sáng đầu tiên: Tìm điểm sáng đầu tiên trong ảnh tạm thời và khởi tạo một ma trận A chứa điểm này.
4. Giãn nở và phép toán AND: Giãn nở ma trận A và lấy phép toán AND với ảnh gốc để tìm thành phần liên thông.
5. Lưu thông tin: Lưu số lượng thành phần liên thông và số lượng pixel trong mỗi thành phần, cập nhật ảnh tạm thời và ma trận nhãn.
6. Trả về kết quả: Trả về thông tin về các thành phần liên thông và ma trận nhãn.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **fillhole** -t <wait key time>

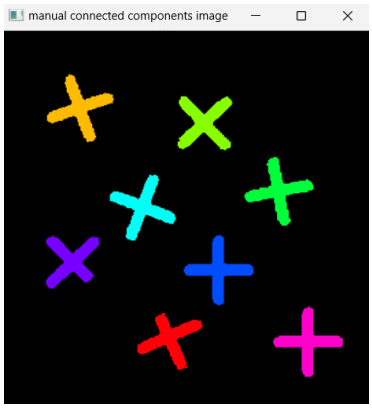
So sánh kết quả:



Ảnh nhị phân gốc



OpenCV



Thủ công

Nhận xét: Kết quả trả về là ảnh đã gán nhãn cho các thành phần liên thông, số lượng thành phần liên thông và số pixel thuộc mỗi thành phần liên thông. Ảnh hiển thị sẽ tô các màu khác nhau cho các thành phần liên thông khác nhau. Kết quả hiển thị của hàm tự cài đặt tương tự kết quả hiển thị khi dùng hàm của opencv.

3.9. Convex Hull

Ý tưởng thuật toán:

1. Tìm các đường viền (Contours): Để tìm các đường viền của các đối tượng trong ảnh nhị phân, chúng ta sử dụng thuật toán BFS (Breadth-First Search). Ý tưởng là duyệt qua từng pixel trong ảnh và kiểm tra xem pixel đó có phải là một phần của đối tượng hay không. Nếu có, ta bắt đầu từ pixel đó và sử dụng BFS để lần theo đường viền của đối tượng đó.

   Chi tiết tường bước:

`

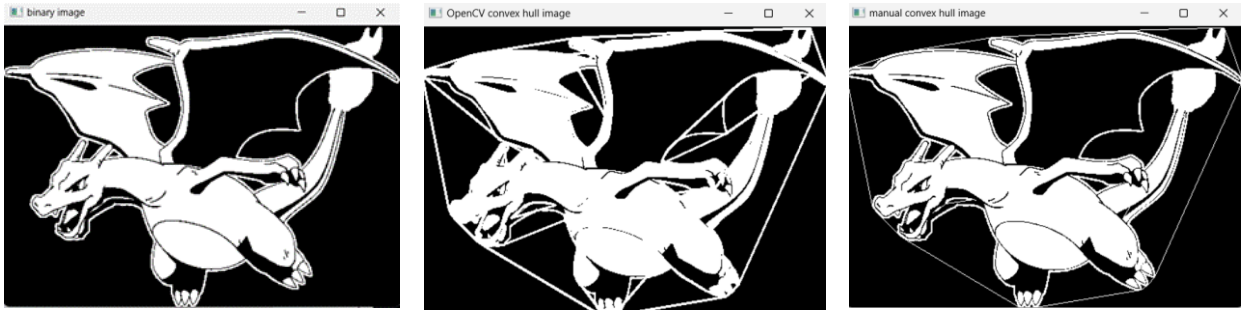
- Khởi tạo một ma trận visited có kích thước bằng ảnh và giá trị ban đầu là False.
- Định nghĩa các hướng đi lân cận (trên, dưới, trái, phải).
- Định nghĩa hàm bfs(start) để thực hiện BFS từ một điểm bắt đầu. Trong quá trình này, ta duyệt qua các điểm lân cận của điểm hiện tại và đánh dấu các điểm đã duyệt.
- Duyệt qua từng pixel trong ảnh. Nếu pixel đó chưa được duyệt và là một phần của đối tượng (img[i, j] == 1), ta sẽ bắt đầu BFS từ pixel đó để tìm đường viền của đối tượng.
- Mỗi khi tìm thấy một đường viền, lưu lại các điểm của đường viền đó vào danh sách contours.

2. Tìm bao lồi (Convex Hull): Sau khi đã tìm được các đường viền (contours), tiếp theo là tìm bao lồi cho mỗi đường viền đó. Bao lồi của một tập hợp các điểm là một đa giác lồi nhỏ nhất mà bao phủ tất cả các điểm trong tập hợp đó. Để tìm bao lồi, chúng ta sử dụng thuật toán Graham scan.
   Chi tiết từng bước của Graham scan:
   - Sắp xếp các điểm theo thứ tự lexicographically.
   - Xây dựng bao lồi bằng cách sử dụng một ngăn xếp (stack) để lưu trữ các điểm thuộc bao lồi.
   - Bắt đầu từ điểm thấp nhất (góc phía dưới bên trái của hình chữ nhật bao quanh các điểm).
   - Duyệt qua từng điểm và sử dụng cross product để kiểm tra xem điểm hiện tại có thuộc bao lồi hay không.
   - Nếu không thuộc, loại bỏ điểm từ ngăn xếp cho đến khi điểm hiện tại thuộc bao lồi.
   - Thêm điểm hiện tại vào ngăn xếp.
3. Tóm tắt:
   - Thuật toán Convex Hull bao gồm hai bước chính: tìm các đường viền (contours) và tìm bao lồi (convex hull) cho mỗi đường viền.
   - Các đường viền được tìm bằng BFS từ các pixel trong ảnh nhị phân.
   - Bao lồi được tìm bằng thuật toán Graham scan, sắp xếp các điểm và sử dụng stack để xây dựng bao lồi.
   - Kết quả cuối cùng là các đường viền và bao lồi được vẽ lên ảnh gốc để phân tích hoặc hiển thị các đối tượng trong hình ảnh.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **convexhull** -t <wait key time>

So sánh kết quả:

`



Ảnh nhị phân gốc OpenCV Thủ công

Nhận xét: Kết quả chính xác tương tự OpenCV nhưng đường viền vẽ bằng numpy có đôi chút mỏng hơn.

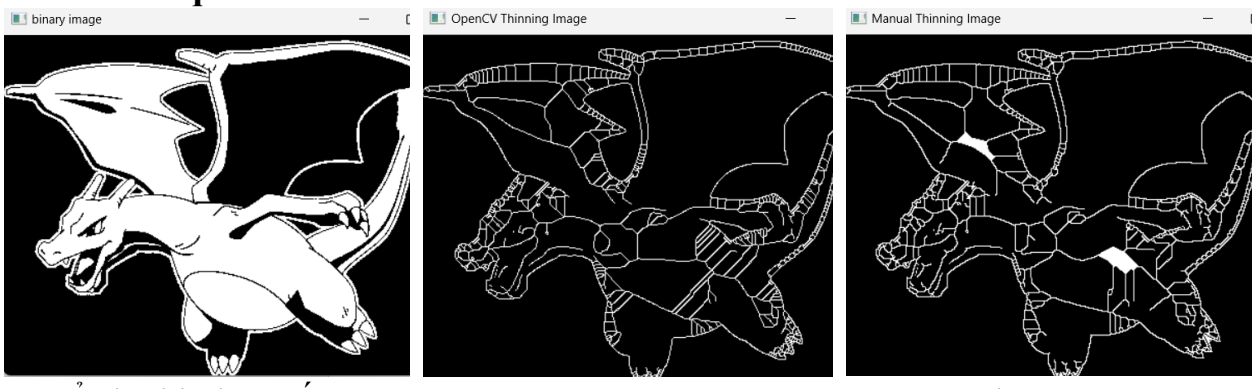
3.10. Thinning

Ý tưởng thuật toán:

1. Tạo set 8 kernel: Khởi tạo 8 kernel đặc biệt.
2. Loại bỏ pixel theo từng kernel:
   - Thực hiện phép toán hit-or-miss với từng kernel trong set 8 và loại bỏ các pixel phù hợp.
   - Quá trình này được thực hiện tuần tự từ kernel đầu tiên đến kernel thứ tám.
3. Lặp lại quá trình loại bỏ:
   - Sử dụng vòng lặp vô hạn để tiếp tục loại bỏ pixel cho đến khi ảnh không còn thay đổi sau khi xử lý với tất cả các kernel.
   - Nếu không có pixel nào bị loại bỏ trong một vòng lặp đầy đủ với tất cả các kernel, trả về ảnh hiện tại.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **thinning** -t <wait key time>

So sánh kết quả:



Ảnh nhị phân gốc OpenCV Thủ công

`

Nhận xét: Kết quả gần giống OpenCV. Sự khác biệt là do hàm tự cài đặt không có bước chuyển đổi m-connectivity để loại bỏ multiple paths.

3.11. Thickening

Mục đích của thuật toán:

- Làm nổi bật các đối tượng quan trọng trong ảnh.
- Tạo các biên dạng dày hơn cho các đối tượng để dễ dàng nhận diện.
- Chuẩn bị ảnh cho các bước xử lý tiếp theo như trích xuất đặc trưng hoặc phân đoạn ảnh.

Ý tưởng chính thuật toán:

1. Khởi tạo các kernel:
   - Có 8 kernel khác nhau được sử dụng trong thuật toán. Mỗi kernel là một ma trận 3x3 với các giá trị -1, 0, và 1.
   - Các kernel này được thiết kế để xác định các mẫu cụ thể trong ảnh nhị phân, giúp nhận diện các vị trí cần làm dày.
2. Khởi tạo ảnh kết quả:
   - Ảnh kết quả ban đầu được khởi tạo là ảnh gốc (img).
3. Duyệt qua các kernel:
   - Với mỗi kernel trong danh sách các kernel, thuật toán sẽ thực hiện phép toán hit-or-miss để tìm các vị trí trong ảnh cần làm dày.
   - Phép toán hit-or-miss kiểm tra các vị trí trong ảnh có khớp với mẫu của kernel hay không.
4. Cập nhật ảnh kết quả:
   - Nếu một vị trí trong ảnh khớp với mẫu của kernel, pixel tại vị trí đó sẽ được làm dày (tức là biến thành màu trắng).
   - Ảnh kết quả được cập nhật bằng cách thêm các pixel trắng vào các vị trí cần làm dày.
5. Trả về ảnh đã làm dày:
   - Sau khi duyệt qua tất cả các kernel, thuật toán trả về ảnh đã được làm dày.

Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p thicken -t <wait key time>

`

Ảnh nhị phân gốc

Thủ công

Nhận xét*:* Kết quả cho ra khá tốt.

3.12. Skeleton

Ý tưởng thuật toán:

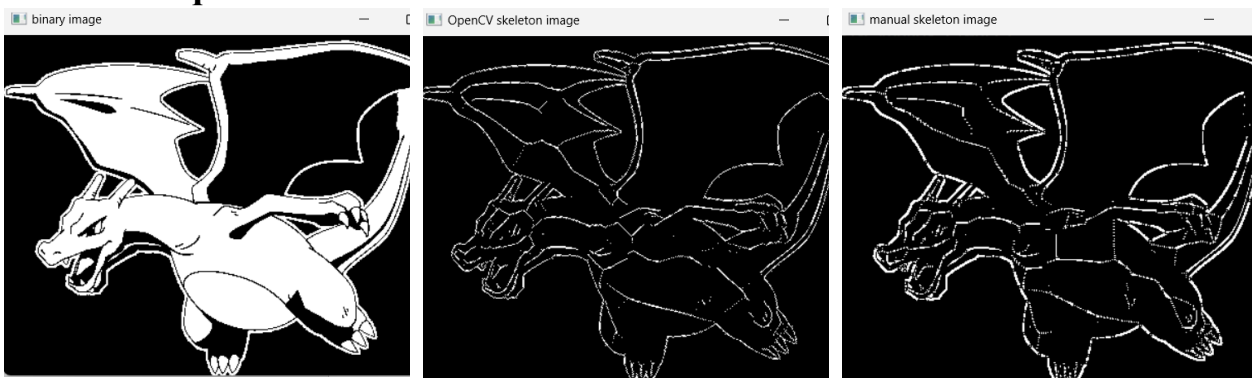
1. Khởi tạo ảnh khung xương: Tạo một ảnh mới với cùng kích thước như ảnh gốc, nhưng toàn bộ giá trị pixel là 0.
2. Xói mòn và giãn nở: Thực hiện phép xói mòn trên ảnh và sau đó giãn nở ảnh đã xói mòn.
3. Trừ và cập nhật: Trừ ảnh đã giãn nở từ ảnh gốc và lưu kết quả vào ảnh khung xương. Cập nhật ảnh gốc thành ảnh đã xói mòn.
4. Lặp lại quá trình: Lặp lại quá trình cho đến khi không còn pixel nào sáng trong ảnh gốc.

Quá trình chi tiết:

1. Khởi tạo ảnh khung xương:
   - Tạo một ảnh mới skel với cùng kích thước như ảnh gốc nhưng toàn bộ giá trị pixel là 0.
2. Vòng lặp chính:
   - Thực hiện phép xói mòn trên ảnh gốc với kernel element.
   - Thực hiện phép giãn nở trên ảnh đã xói mòn.
   - Trừ ảnh đã giãn nở từ ảnh gốc bằng hàm subtract và lưu kết quả vào temp.
   - Sử dụng phép toán OR bitwise để kết hợp skel và temp, cập nhật skel.
   - Cập nhật ảnh gốc img thành ảnh đã xói mòn.
   - Kiểm tra nếu ảnh gốc không còn pixel sáng, dừng vòng lặp.
3. Trả về ảnh khung xương:
   - Trả về ảnh skel, chứa khung xương của các đối tượng trong ảnh gốc.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **skeleton** -t <wait key time>

So sánh kết quả:



Ảnh nhị phân gốc | OpenCV | Thủ công

Nhận xét: Kết quả gần giống với OpenCV

3.13. Pruning

Ý tưởng thuật toán:

1. Khởi tạo set 8 kernel:
   - Tạo ra 8 kernel đặc biệt được thiết kế để tìm và loại bỏ các chi tiết thừa trong ảnh.
2. Loại bỏ các chi tiết thừa:
   - Thực hiện phép toán hit-or-miss với từng kernel trong set 8 để loại bỏ các chi tiết thừa.
   - Cập nhật ảnh tạm thời qua từng bước xử lý với các kernel.
3. Cập nhật ảnh đã loại bỏ chi tiết thừa:
   - Sau khi xử lý với tất cả các kernel, tạo một bản sao của ảnh đã được loại bỏ chi tiết thừa.
   - Tiếp tục thực hiện phép toán hit-or-miss với ảnh này và các kernel, cập nhật kết quả vào một ảnh mới.
   - Thực hiện phép giãn nở trên ảnh kết quả và lấy phép toán bitwise AND với ảnh gốc để tạo ra một ảnh chứa các chi tiết cần giữ lại.
1. Trả về ảnh đã được cắt tỉa:
   - Kết hợp ảnh đã loại bỏ chi tiết thừa và ảnh chứa các chi tiết cần giữ lại để tạo ra ảnh cuối cùng.

Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **prun** -t <wait key time>

So sánh kết quả:

`

Ảnh nhị phân gốc OpenCV Thủ công

Nhận xét: Kết quả tương tự với OpenCV

3.14. Morphological Reconstruction

Mã nguồn cài đặt bao gồm Reconstruction by Dilation, Reconstruction by Erosion, Opening by Reconstruction, Closing by Reconstruction.

Reconstruction by Dilation

Thuật toán reconstruct_by_dilation sử dụng quá trình giãn nở lặp đi lặp lại để khôi phục (reconstruct) ảnh marker_img trong phạm vi của ảnh mask_img. Quá trình này dừng lại khi không có sự thay đổi nào giữa các lần lặp giãn nở liên tiếp.

Ý tưởng thuật toán:

1. Mục tiêu: Khôi phục ảnh marker_img trong phạm vi của mask_img bằng phép giãn nở.
2. Phương pháp: Lặp lại quá trình giãn nở và kết hợp với mask_img cho đến khi không còn sự thay đổi nào giữa các lần lặp.
3. Kết quả: Ảnh cuối cùng được khôi phục D_pre là ảnh đã được giãn nở tối đa mà vẫn nằm trong phạm vi của mask_img.

Quá trình thực hiện:

1. Khởi tạo:
   - Biến D_pre được khởi tạo bằng marker_img. Đây là ảnh khởi điểm cho quá trình giãn nở.
2. Vòng lặp giãn nở:
   - Giãn nở: Ảnh hiện tại D_pre được giãn nở bằng kernel đã cho, kết quả được lưu vào dilated_img.
   - Kết hợp với mask: Ảnh giãn nở dilated_img được kết hợp với mask_img bằng phép toán bitwise AND, kết quả được lưu vào D_after. Việc này đảm bảo rằng các pixel trong ảnh giãn nở không vượt ra ngoài phạm vi của mask_img.
3. Kiểm tra hội tụ:
   - Nếu D_pre và D_after giống nhau (không có sự thay đổi giữa các lần lặp), quá trình giãn nở dừng lại và trả về D_pre.
   - Nếu có sự thay đổi, D_pre được cập nhật bằng D_after và vòng lặp tiếp

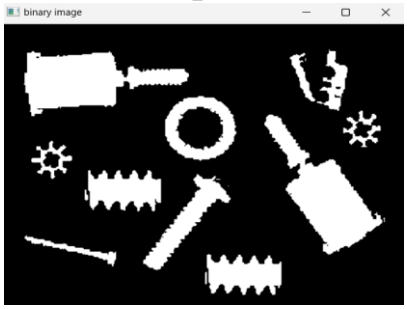
`

tục.

4. Kết thúc:
    - Trả về kết quả cuối cùng là D_pre, ảnh đã được khôi phục hoàn chỉnh.

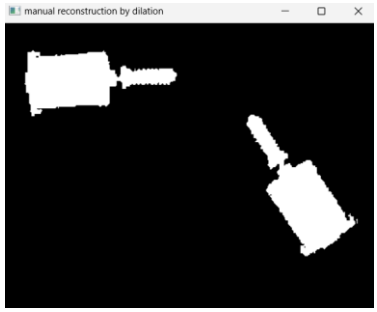
Cách sử dụng (chạy câu lệnh): python main.py -i <input file> -o <output file> -p **reconstruct_by_dilation** -t <wait key time>

So sánh kết quả:



Ảnh mask

Ảnh marker



Thủ công

Nhận xét: Kết quả đã phục hồi được các đối tượng dựa vào ảnh marker.

Opening by Reconstruction

Thuật toán open_by_reconstruction thực hiện quá trình mở bằng tái tạo (opening by reconstruction) cho một ảnh marker_img với số lần lặp nhất định, sử dụng phép xói mòn (erosion) và phép giãn nở (dilation) kèm theo mask. Dưới đây là phần giải thích chi tiết về ý tưởng chính của thuật toán qua đoạn mã đã cho

Ý tưởng thuật toán:

1. Mục tiêu: Loại bỏ các chi tiết nhỏ và tiếng ồn trong ảnh marker_img trong khi bảo toàn các cấu trúc chính bằng quá trình mở bằng tái tạo.
2. Phương pháp: Kết hợp phép xói mòn lặp lại với quá trình tái tạo bằng giãn nở trong phạm vi của mask_img.
3. Kết quả: Ảnh cuối cùng được mở bằng tái tạo (result) là ảnh đã được loại bỏ các chi tiết nhỏ và tiếng ồn.

Quá trình thực hiện:

1. Khởi tạo:
    - Biến tmp_before được khởi tạo bằng marker_img. Đây là ảnh khởi điểm cho quá trình xói mòn.
2. Xói mòn lặp đi lặp lại:
    - Trong vòng lặp với số lần lặp iteration đã cho, ảnh hiện tại tmp_before được xói mòn bằng kernel đã cho.
    - Kết quả xói mòn được lưu vào tmp_after, sau đó tmp_before được cập nhật bằng tmp_after.
3. Tái tạo bằng giãn nở:

`

- Sau khi hoàn tất quá trình xói mòn lặp lại, thuật toán sử dụng hàm reconstruct_by_dilation để tái tạo ảnh từ kết quả xói mòn cuối cùng (tmp_before) trong phạm vi của mask_img.
- Kết quả tái tạo được lưu vào result.

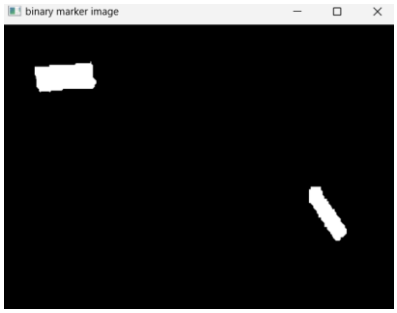
4. Trả về kết quả:

- Trả về kết quả cuối cùng là result, ảnh đã được mở bằng tái tạo.

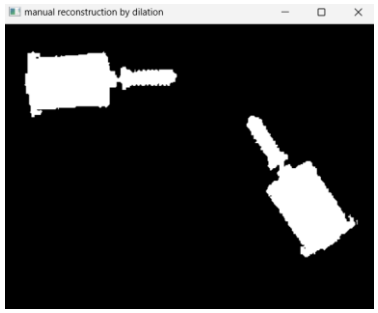
Cách sử dụng (chạy câu lệnh): python main.py -i <mask file> -m <marker file> -k <số lần erosion> -o <output file> -p **opening_by_reconstruction**-t <wait key time>

So sánh kết quả:

Ảnh mask



Ảnh marker



Thủ công với 3 lần erosion

Ảnh mask

Ảnh marker

Thủ công với 4 lần erosion

Nhận xét: Phục hồi được các đối tượng dựa vào ảnh marker, có thể "lựa chọn" được đối tượng cần phục hồi dựa vào số lần erosion truyền vào.

4. Các toán tử hình thái học trên ảnh xám

4.1. Erosion

Nhìn chung, thuật toán Erosion trên ảnh độ xám cũng tương tự như Erosion trên ảnh nhưng khác biệt ở một số chỗ như sau:

1. **Điều kiện khớp:**

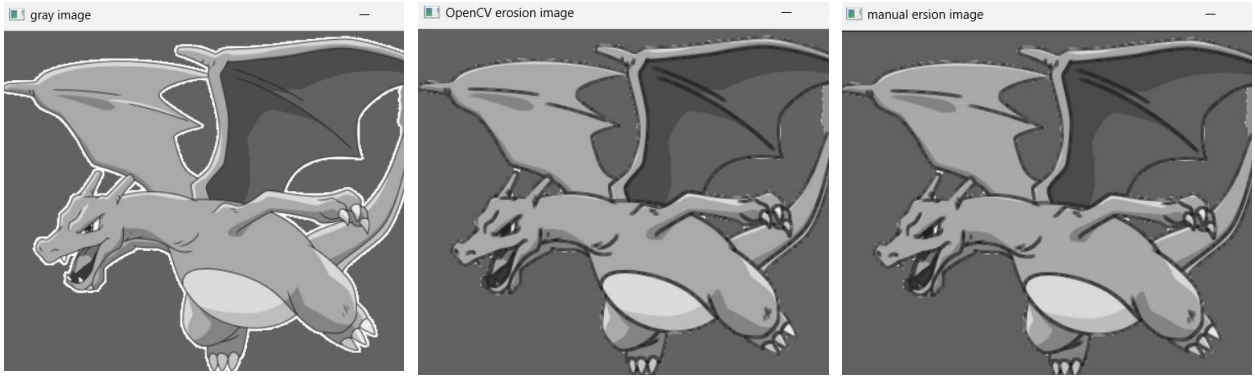
- **Erosion binary:** So sánh tổng của các pixel trong cửa sổ con với tổng của các giá trị trong kernel (đã nhân với giá trị pixel gốc 255). Chỉ khi nào tất cả các pixel trong cửa sổ con khớp hoàn toàn với kernel thì giá trị pixel tại tâm

sẽ được gán là 255.

- **Erosion gray:** Thực hiện trừ kernel từ cửa sổ con của ảnh và gán giá trị nhỏ nhất vào ảnh xói mòn. Điều này đảm bảo rằng các chi tiết nhỏ hơn kernel sẽ bị loại bỏ.

2. **Kết quả trả về**
   - **Erosion binary:** Trả về ảnh nhị phân đã xói mòn.
   - **Erosion gray:** Trả về ảnh mức xám đã xói mòn và chuẩn hóa lại giá trị pixel về phạm vi [0, 1].

Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p **erode_gray** -t <wait key time>



Ảnh độ xám gốc | OpenCV | Thủ công

Nhận xét: Kết quả cho ra tương đối giống so với kết quả của Open CV

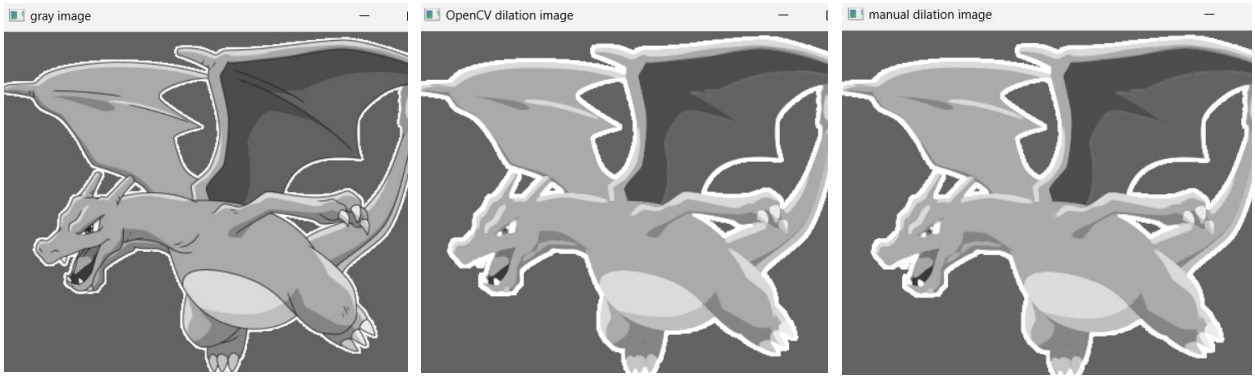
4.2.Dilation

Nhìn chung, thuật toán Dilation trên ảnh độ xám cũng tương tự như Dilation trên ảnh nhưng khác biệt ở một số chỗ như sau:

1. **Điều kiện khớp:**
   - **Dilation binary:** Chỉ cần kiểm tra xem pixel tại vị trí tâm của kernel có giá trị là 255 hay không. Nếu có, thì gán giá trị 255 cho tất cả các pixel trong cửa sổ con của ảnh.
   - **Dilation gray:** Tính toán giá trị mới cho mỗi pixel bằng cách cộng kernel với cửa sổ con của ảnh. Sau đó, chọn giá trị lớn nhất và gán lại vào vị trí tương ứng trong ảnh đã giãn nở.bị loại bỏ.
2. **Kết quả trả về**
   - **Dilation binary:** Trả về ảnh nhị phân đã giãn nở.
   - **Dilation gray:** Trả về ảnh mức xám đã giãn nở và chuẩn hóa lại giá trị pixel về phạm vi [0, 1].

Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p **dilate_gray** -t <wait key time>

`



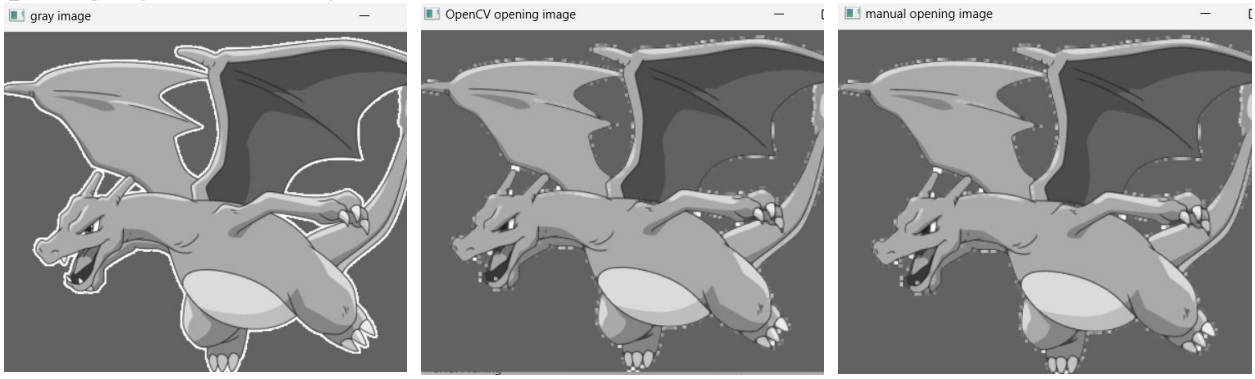
Ảnh độ xám gốc OpenCV Thủ công

Nhận xét*:* Kết quả cho ra tương đối giống so với kết quả của Open CV

4.3. Opening

Nhìn chung, thuật toán Opening trên ảnh độ xám cũng tương tự như Opening trên ảnh nhưng khác biệt ở chỗ: Sau bước đầu tiên xói mòn thì ảnh cần được chuẩn hóa về khoảng [0, 255] để làm đầu vào cho bước giãn nở tiếp theo.

Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p **open_gray** -t <wait key time>



Ảnh độ xám gốc OpenCV Thủ công

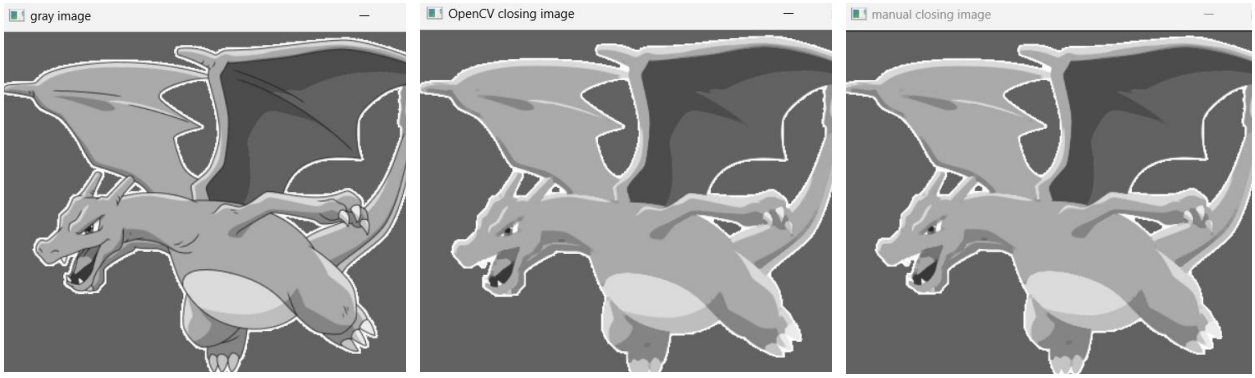
Nhận xét*:* Kết quả cho ra tương đối giống so với kết quả của Open CV

4.4. Closing

Nhìn chung, thuật toán Opening trên ảnh độ xám cũng tương tự như Opening trên ảnh nhưng khác biệt ở chỗ: Sau bước đầu tiên xói mòn thì ảnh cần được chuẩn hóa về khoảng [0, 255] để làm đầu vào cho bước giãn nở tiếp theo.

Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p **close_gray** -t <wait key time>

`



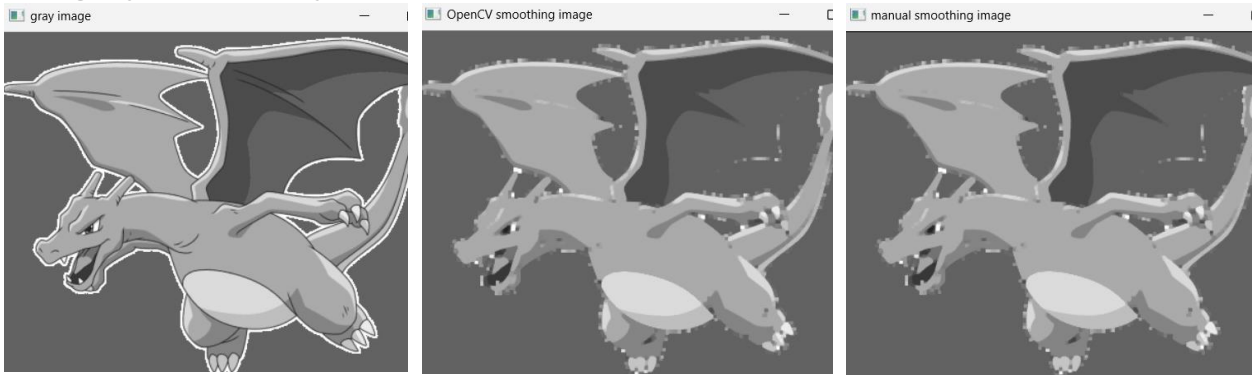
Ảnh độ xám gốc OpenCV Thủ công

Nhận xét*:* Kết quả cho ra tương đối giống so với kết quả của Open CV

4.5.Smoothing

Thuật toán Smoothing sử dụng phép Opening để loại bỏ các chi tiết nhỏ và sau đó áp dụng phép Closing để làm mịn và đặc biệt hơn trên ảnh, dẫn đến một hiệu ứng làm mịn toàn cảnh của ảnh đầu vào.

Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p **close_gray** -t <wait key time>



Ảnh độ xám gốc OpenCV Thủ công

Nhận xét*:* Kết quả cho ra tương đối giống so với kết quả của Open CV

4.6.Gradient

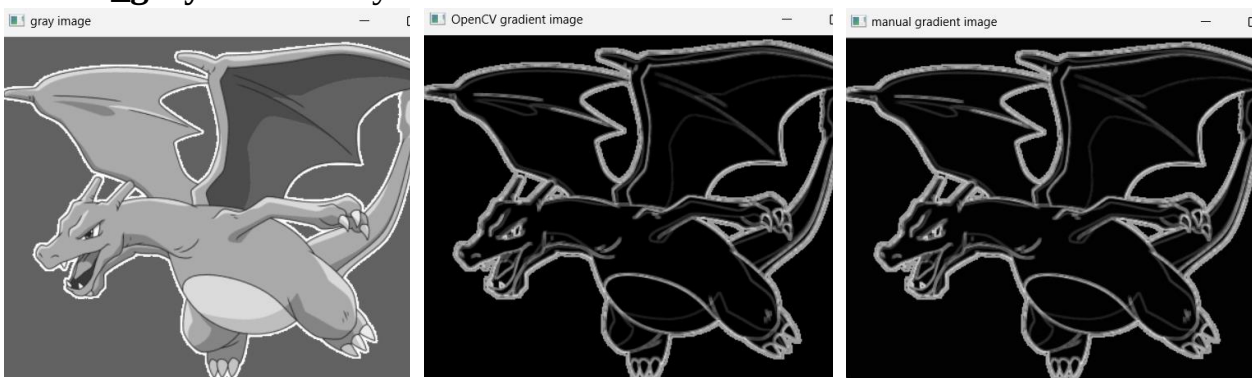
Hàm Gradient được sử dụng để phát hiện các biên cạnh trong ảnh bằng cách tính sự khác biệt giữa ảnh đã được Dilation và ảnh đã được Erosion với cùng một kernel.

Ý tưởng chính thuật toán:

1. Áp dụng phép Dilation với kernel cho ảnh img, để tăng giá trị pixel lên xung quanh các cạnh và cấu trúc bên trong.
2. Áp dụng phép Erosion với kernel cho ảnh img, để giảm giá trị pixel xung quanh các cạnh và cấu trúc bên trong.
3. Trả về sự khác biệt giữa kết quả của dilate và erode, làm nổi bật các cạnh và biên trong ảnh.

Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p

gradient_gray -t <wait key time>



Ảnh độ xám gốc OpenCV Thủ công

Nhận xét: Kết quả cho ra tương đối giống so với kết quả của Open CV

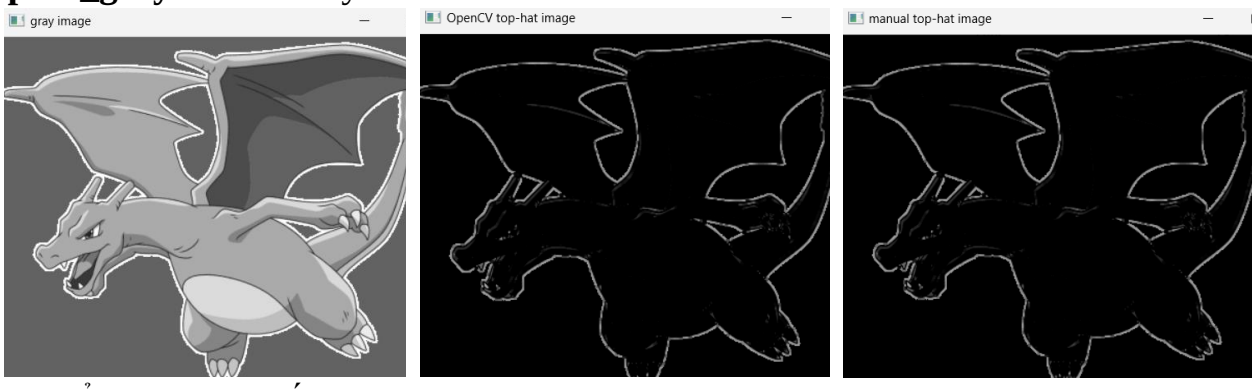
4.7. Top-hat transformation

Hàm top_hat giúp phát hiện các đặc trưng nhỏ và cấu trúc nổi bật trong ảnh bằng cách tính sự khác biệt giữa ảnh gốc và kết quả của phép open. Kết quả trả về là một ảnh mức xám có giá trị nằm trong khoảng [0, 1], thể hiện sự khác biệt này.

Ý tưởng chính thuật toán:

1. Sử dụng phép Opening để làm mịn ảnh img sử dụng kernel đã cho. Phép Opening bao gồm hai bước liên tiếp là erode và dilate.
2. Kết quả của phép open được nhân với 255.0 để đảm bảo rằng kết quả là ảnh mức xám với giá trị pixel nằm trong khoảng [0, 255].
3. Trừ kết quả của Opening từ ảnh ban đầu img.
4. Chia kết quả cho 255.0 để chuẩn hóa giá trị về khoảng [0, 1].

Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p **tophat_gray** -t <wait key time>



Ảnh độ xám gốc OpenCV Thủ công

Nhận xét: Kết quả cho ra tương đối giống so với kết quả của Open CV

4.8. Black-hat Transformation

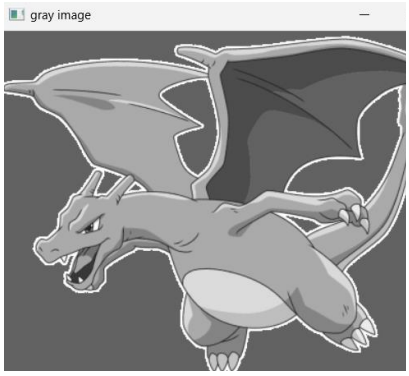
Hàm black_hat giúp phát hiện các đặc trưng nhỏ và cấu trúc nổi bật trong ảnh bằng cách tính sự khác biệt giữa ảnh đã được làm mịn bằng close và ảnh ban đầu. Kết quả

trả về là một ảnh mức xám có giá trị nằm trong khoảng [0, 1], thể hiện sự khác biệt này.

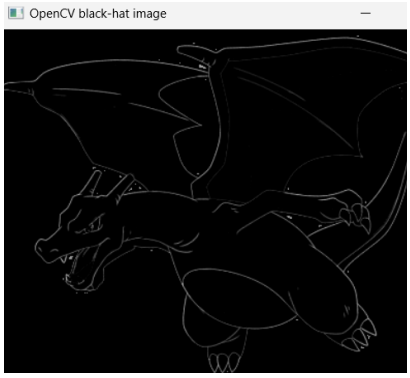
Ý tưởng chính thuật toán:

1. Sử dụng phép close để làm mịn ảnh img sử dụng kernel đã cho. Phép close bao gồm hai bước liên tiếp là dilate và erode.
2. Kết quả của phép close được nhân với 255.0 để đảm bảo rằng kết quả là ảnh mức xám với giá trị pixel nằm trong khoảng [0, 255].
3. Trừ ảnh ban đầu img từ kết quả của close.
4. Chia kết quả cho 255.0 để chuẩn hóa giá trị về khoảng [0, 1].

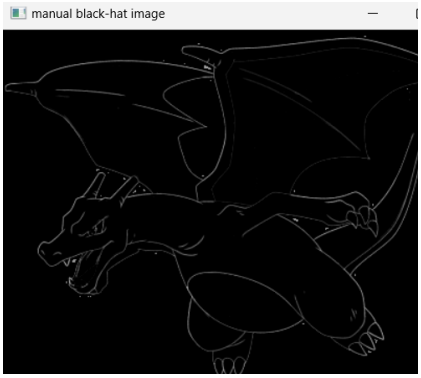
Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p **black_gray** -t <wait key time>



Ảnh độ xám gốc



OpenCV



Thủ công

Nhận xét: Kết quả cho ra tương đối giống so với kết quả của Open CV

4.9. Granulometry

Hàm Granulometry giúp phân tích và làm mịn ảnh bằng cách sử dụng các phép toán hình thái học như smooth và open. Điều này giúp loại bỏ nhiễu, làm nổi bật các cấu trúc chính trong ảnh và loại bỏ các đối tượng nhỏ hơn một kích thước nhất định.

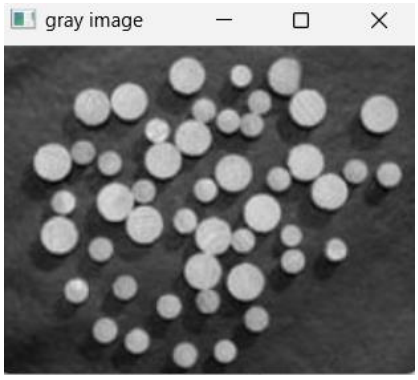
Ý tưởng chính thuật toán:

1. Làm mịn ảnh (Smoothing):
   - Giúp loại bỏ nhiễu và làm nổi bật các cấu trúc chính trong ảnh.
2. Phép toán mở rộng (Opening):
   - Co lại (Erosion): Giảm kích thước các đối tượng để loại bỏ những đối tượng nhỏ.
   - Giãn ra (Dilation): Mở rộng lại các đối tượng đã bị giảm kích thước nhưng không khôi phục các đối tượng nhỏ đã bị loại bỏ.

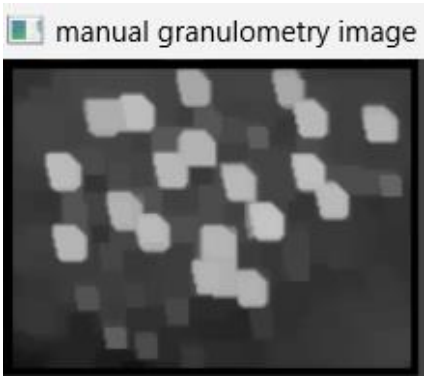
Kết quả là một ảnh mịn hơn, chỉ giữ lại các đối tượng lớn hơn một kích thước nhất định, giúp phân tích và đo lường kích thước của các cấu trúc trong ảnh.

Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p **granulometry_gray** -t <wait key time>

`



Ảnh độ xám gốc



Thủ công

Nhận xét*:* Kết quả như mong đợi.

4.10. Mophological Reconstruction

Các cài đặt bao gồm Reconstruction by Dilation, Reconstruction by Erosion, Opening by Reconstruction, Closing by Reconstruction tương tự với ảnh nhị phân.

Reconstruction by Dilation

Hàm black_hat giúp phát hiện các đặc trưng nhỏ và cấu trúc nổi bật trong ảnh bằng cách tính sự khác biệt giữa ảnh đã được làm mịn bằng close và ảnh ban đầu. Kết quả trả về là một ảnh mức xám có giá trị nằm trong khoảng [0, 1], thể hiện sự khác biệt này.

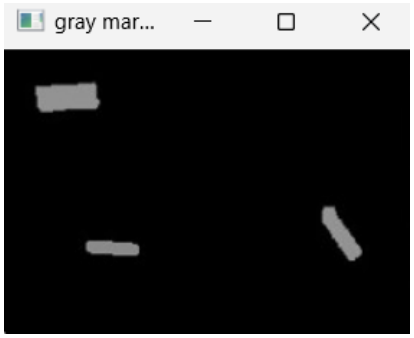
Ý tưởng chính thuật toán:

1. Sử dụng phép close để làm mịn ảnh img sử dụng kernel đã cho. Phép close bao gồm hai bước liên tiếp là dilate và erode.
2. Kết quả của phép close được nhân với 255.0 để đảm bảo rằng kết quả là ảnh mức xám với giá trị pixel nằm trong khoảng [0, 255].
3. Trừ ảnh ban đầu img từ kết quả của close.
4. Chia kết quả cho 255.0 để chuẩn hóa giá trị về khoảng [0, 1].

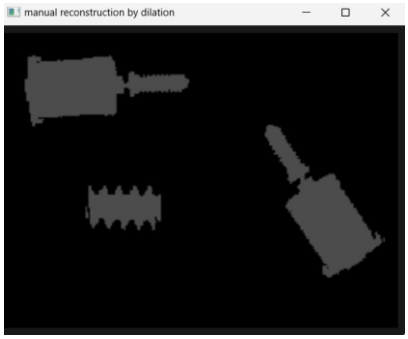
Cách sử dụng (chạy câu lệnh): main.py -i <input file> -o <output file> -p **reconstruct_by_dilation_gray** -t <wait key time>

So sánh kết quả:

Ảnh mask



Ảnh marker



Thủ công

`

Nhận xét: Kết quả đã phục hồi được các đối tượng dựa vào marker.

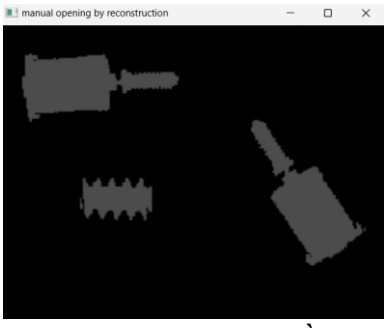
Opening by Reconstruction

Cách sử dụng (chạy câu lệnh): python main.py -i <mask file> -m <marker file> -k <số lần erosion> -o <output file> -p **opening_by_reconstruction_gray** -t <wait key time>

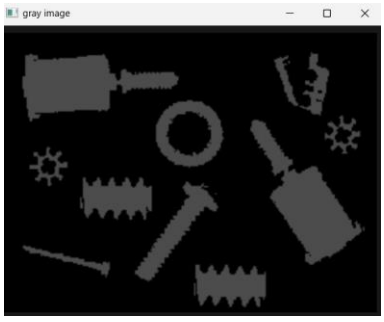
So sánh kết quả:

Ảnh mask

Ảnh marker

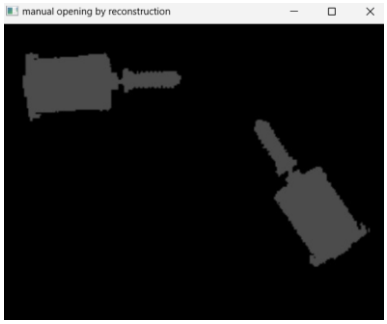


Thủ công với 2 lần erosion

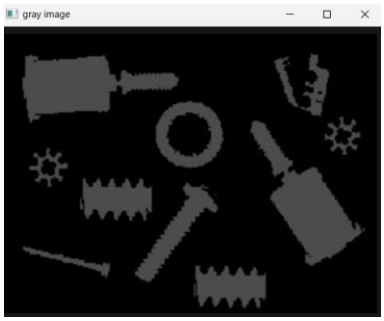


Ảnh mask

Ảnh marker

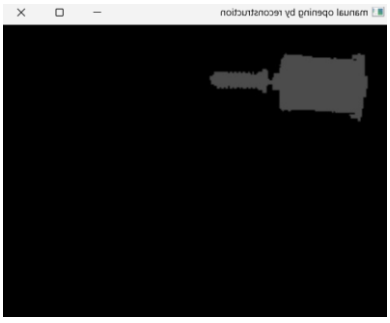


Thủ công với 3 lần erosion



Ảnh mask

Ảnh marker



Thủ công với 4 lần erosion

Nhận xét: Phục hồi được các đối tượng dựa vào ảnh marker, có thể "lựa chọn" được đối tượng cần phục hồi dựa vào số lần erosion truyền vào.

`

5. Nguồn tham khảo

1. Sách Digital Image Processing – Fourth Edition - Rafael C. Gonzalez, Richard E. Woods
2. Slide bài giảng Các toán tử hình thái học ảnh độ xám – LQN.
3. Slide bài giảng Các toán tử hình thái học ảnh nhị phân – LQN.