TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN CUỐI KỲ

MÔN NHẬP MÔN XỬ LÝ ẢNH SỐ

ĐỒ ÁN CUỐI KỲ

Người hướng dẫn: **TS. Phạm Văn Huy**

Người thực hiện: **Nguyễn Minh Chí – 52000189**

Nguyễn Minh Hoàng Chương – 52000744

Nhóm : 1

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN CUỐI KỲ
MÔN NHẬP MÔN XỬ LÝ ẢNH SỐ

ĐỒ ÁN CUỐI KỲ

Người hướng dẫn: **TS. Phạm Văn Huy**
Người thực hiện: **Nguyễn Minh Chí – 52000189**
Nguyễn Minh Hoàng Chương – 52000744
Nhóm : 1

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Lời đầu tiên, chúng em xin chân thành cảm ơn Khoa Công Nghệ Thông Tin, Trường Đại Học Tôn Đức Thắng đã tạo điều kiện về cơ sở vật chất, và mở môn học giúp chúng em học tập, tìm hiểu và thực hiện bài báo cáo này.

Chúng em cũng xin chân thành cảm ơn thầy Phạm Văn Huy đã tận tình truyền đạt cho chúng em và các bạn nhóm 1 trong học kì qua. Trong học kì I vừa rồi thầy giúp đỡ em và các bạn học một cách tốt nhất. Trong quá trình học tập thầy rất vui vẻ, nhiệt tình, và tận tụy với sinh viên. Thầy có nhiều cách hay giúp chúng em hiểu bài và làm nhanh. Thầy đã truyền đạt nhiều kiến thức của bản thân nhưng em vẫn còn những hạn chế nhất định, không tránh được những thiếu sót. Em mong nhận được những đóng góp ý kiến từ thầy về bài báo cáo. Em xin chúc thầy nhiều sức khỏe, hạnh phúc và luôn được mọi người yêu mến.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng chúng tôi và được sự hướng dẫn của TS. Phạm Văn Huy. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào chúng tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do chúng tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày tháng năm

Tác giả

(ký tên và ghi rõ họ tên)

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Xử lý ảnh số(Digital Image Processing) là một môn học trong ngành công nghệ thông tin về cách xử lý và biểu diễn hình ảnh số sử dụng thư viện OpenCV. OpenCV là một thư viện mã nguồn mở cho xử lý hình ảnh và video trong lập trình, và có thể được sử dụng để giải quyết các bài toán như phân tích hình ảnh, nhận dạng đối tượng, và nhận dạng ký tự.

Trong báo cáo chúng ta sử dụng OpenCV để giải quyết bài toán Sudoku bằng cách nhận dạng và phân tích các số trên bảng SudoKu, sau đó sử dụng các thuật toán tìm kiếm tối ưu để giải quyết bài toán. Các bước cụ thể có thể bao gồm:

- Tiền xử lý hình ảnh. Sử dụng các hàm của OpenCV để biến đổi hình ảnh đầu vào thành hình ảnh xám và giảm nhiễu.
- Nhận dạng các số. Sử dụng các thuật toán nhận dạng ký tự để nhận dạng và phân tích các số trên bảng SudoKu.
- Giải quyết bài toán. Sử dụng các thuật toán tìm kiếm tối ưu để giải quyết bài toán SudoKu.

MỤC LỤC

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

1. Doanh mục hình

2. Doanh mục bảng

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

1.1 Khái niệm về ảnh xử lý ảnh số

- Ảnh số là : hình ảnh được biểu diễn dưới dạng các giá trị số, được lưu trữ trong máy tính và có thể được xử lý và phân tích bằng máy tính. Thay vì biểu diễn hình ảnh dưới dạng các điểm màu, ảnh số biểu diễn hình ảnh dưới dạng một tập hợp các giá trị màu sắc, độ sáng và tối ưu cho mỗi pixel trên hình ảnh. Các giá trị này được lưu trữ trong một ma trận và có thể xử lý bằng các thuật toán toán học và kỹ thuật xử lý hình ảnh. Xử lý ảnh số là một lĩnh vực rộng lớn trong khoa học máy tính và đang được sử dụng trong rất nhiều lĩnh vực khác nhau, chẳng hạn như nghiên cứu khoa học, y tế, giải trí, và công nghệ thông tin.

- Phép biến đổi hình ảnh: Là các thao tác xử lý hình ảnh như cắt, co dãn, xoay, v.v, để thay đổi các thuộc tính của hình ảnh. Một kỹ thuật trong xử lý ảnh số, cho phép thay đổi tỉ lệ, kích thước, vị trí, hình dạng hoặc màu sắc của một hình ảnh. Các phép biến đổi hình ảnh thông dụng bao gồm: phóng to/thu nhỏ, xoay, lật, đảo trái phải, đổi tỉ lệ màu sắc, và cắt/gộp hình ảnh. Chúng ta có thể sử dụng các phép biến đổi hình ảnh để giải quyết các vấn đề trong xử lý hình ảnh, như làm cho hình ảnh phù hợp với yêu cầu của bài toán, hoặc tối ưu hóa chất lượng hình ảnh.

- Phân tích hình ảnh: Là quá trình phân tích các thông tin của hình ảnh, bao gồm phân tích kỹ thuật số và phân tích nội dung. Nhận dạng đối tượng: quá trình xác định và phân loại các đối tượng trong hình ảnh số.

- OpenCV: OpenCV là viết tắt của "Open Source Computer Vision Library", là một thư viện phần mềm mã nguồn mở dành cho xử lý ảnh và video. Nó được viết bằng ngôn ngữ lập trình C++ và cung cấp rất nhiều công cụ và thuật toán để xử lý và nhận dạng đối tượng trong hình ảnh và video. OpenCV cung cấp một giao diện lập trình dễ sử dụng cho phép các nhà phát triển phần mềm tích hợp các tính năng xử lý hình ảnh

vào các ứng dụng của họ một cách dễ dàng. Nó được sử dụng rộng rãi trong các lĩnh vực như nghiên cứu khoa học, điều khiển robot, an ninh, và phát triển game.

1.2 Phương pháp giải quyết hình ảnh Sodoku

Trong đề tài đồ án cuối kỳ về xử lý hình ảnh sodoku, chúng ta sẽ sử dụng thuật toán xử lý ảnh để nhận diện và giải quyết ảnh SUDOKU từ hình ảnh đầu vào.

Chúng ta sử dụng OpenCV để xử lý hình ảnh và tìm các đường thẳng trong hình. Sau đó, bạn sẽ sử dụng các thuật toán xử lý ảnh như nhận dạng ký tự, biến đổi hình dạng, và tìm kiếm đối tượng để xác định vị trí của số trong hình và tiến hành đọc những chỗ có chữ số. Một vài thuật toán có thể sử dụng cho bài toán nhận diện đường nét trong ảnh nhị phân:

- Sử dụng thuật toán tô chàm và phóng to: Tô chàm sẽ giúp loại bỏ các nền trắng và phóng to sẽ giúp tăng độ sắc nét của đường nét.
- Sử dụng thuật toán Hough transform: Hough transform có thể được sử dụng để nhận diện ra các đường thẳng trong hình ảnh nhị phân.
- Sử dụng thuật toán contour detection: Contour detection có thể được sử dụng để tìm các đối tượng có biên liên tục và nổi bật trong hình ảnh nhị phân.
- Sử dụng thuật toán edge detection: Edge detection có thể được sử dụng để tìm các đường nét của các đối tượng trong hình ảnh nhị phân.
- Các bước để giải quyết bài toán:
    - Step 1: Nhận diện ảnh và đưa ra các đường thẳng trong hình ảnh SudoKu.
        - Gaussian Blur: Sử dụng Gaussian Blur để làm mờ những nhiễu trong ảnh. Điều này sẽ giúp cho việc nhận diện đường thẳng trở nên chính xác hơn.
        - Adaptive Thresholding: Sử dụng adaptive thresholding để chuyển đổi ảnh sang dạng đen trắng. Điều này sẽ giúp cho việc nhận diện đường thẳng trở nên dễ dàng hơn.

- Xử lý hình thái (Erode và Dilate): Sử dụng xử lý hình thái để làm cho đường thẳng trở nên rõ ràng hơn.

- Step 2: Tô chàm: Xử lý hình ảnh để loại bỏ nền và tăng độ chính xác khi nhận diện số.
  - Sử dụng thuật toán tô chàm để tìm ra các điểm trên ảnh mà chúng ta muốn nhận diện đường thẳng.
- Step 3: Hough Transform. Sử dụng thuật toán giải quyết SudoKu để giải quyết bài toán.
  - Sử dụng thuật toán Hough Transform để tìm ra các đường thẳng trong ảnh. Chúng ta có thể sử dụng các tham số khác nhau để tìm ra các đường thẳng mong muốn.

CHƯƠNG 2: TRÌNH BÀY VÀ PHÂN TÍCH THUẬT TOÁN

2.1 Thuật toán Gaussian Blur

2.1.1 Thuật toán Gaussian Blur là gì?

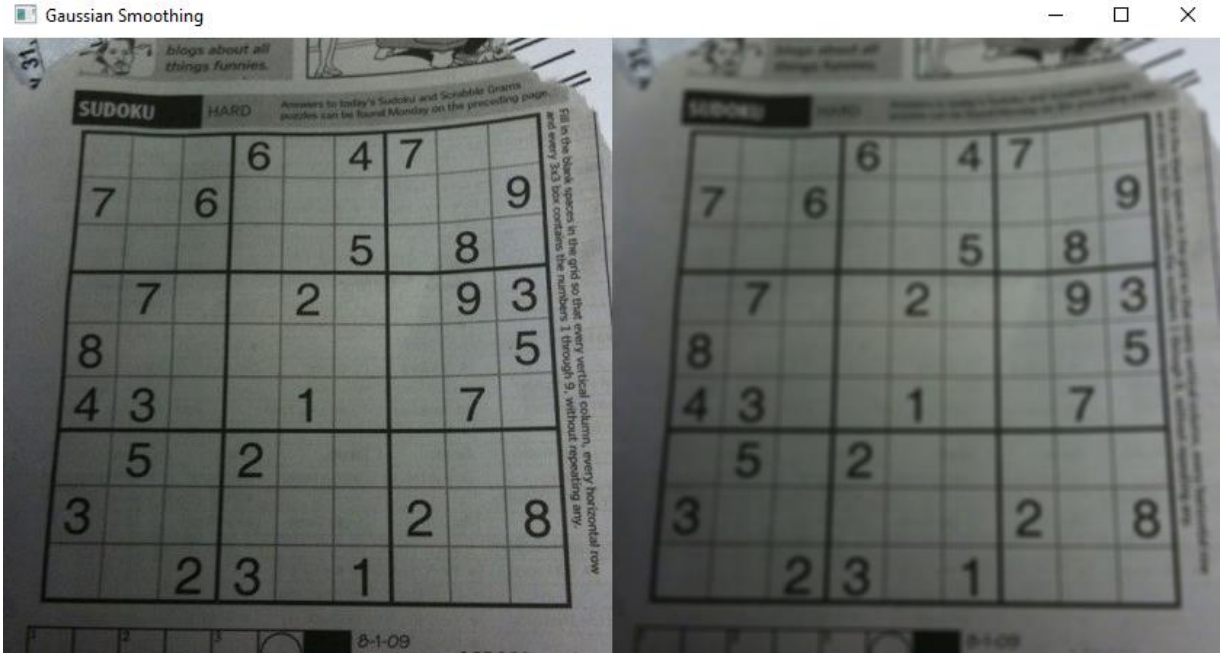
Gaussian blur là một thuật toán để làm mờ hình ảnh. Nó là một trong những phương pháp để xử lý hình ảnh mà sử dụng một ma trận trọng số tạo ra bởi hàm Gaussian.

Thực tế, Gaussian blur sử dụng một mẫu lọc có dạng hình tròn để tính toán một giá trị trung bình cho mỗi pixel trong hình ảnh. Kết quả của thuật toán này là một hình ảnh mờ hơn với các đường viền và sự chênh lệch giữa các pixel trở nên ít hơn. Gaussian blur có thể được sử dụng để loại bỏ nhiễu, tăng độ mịn hoặc để tạo ra hiệu ứng mờ.

2.1.2 Ví dụ thuật toán Gausian Blur

Code ví dụ:

```
import cv2
import numpy as np
# Load hình ảnh và chuyển sang dạng grayscale
image = cv2.imread("image.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Tính toán Gaussian blur với kích thước lọc là 5x5
blurred = cv2.GaussianBlur(gray, (5, 5), 0)

# Hiển thị hình ảnh gốc và hình ảnh mờ
cv2.imshow("Original Image", gray)
cv2.imshow("Blurred Image", blurred)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Hình 1 Ví dụ thuật toán Gausian Blur

2.1.3 Một số thuật toán tương tự

Có một số thuật toán tương tự như Gaussian blur, bao gồm:

Mean Blur: Sử dụng một mẫu lọc cố định để tính trung bình cho mỗi pixel trong hình ảnh. Kết quả là một hình ảnh mờ hơn nhưng có thể gây ra đổ màu và biến đổi mầu sắc.

Median Blur: Sử dụng trung vị của các pixel trong mẫu lọc để tính giá trị cho mỗi pixel trong hình ảnh. Kết quả là một hình ảnh mờ hơn và có thể loại bỏ nhiễu tốt hơn so với Mean Blur.

Bilateral Blur: Sử dụng một hàm trọng số để tính giá trị cho mỗi pixel trong hình ảnh, với trọng số được tính dựa trên khoảng cách giữa pixel và trọng số màu sắc. Kết quả là một hình ảnh mờ hơn và giữ nguyên độ chi tiết hơn so với các thuật toán Mean Blur và Median Blur.

2.2 Thuật toán Adaptive Thresholding

2.2.1 Thuật toán Adaptive Thresholding là gì?

Adaptive Thresholding là một kỹ thuật xử lý ảnh được sử dụng để chuyển đổi ảnh thang độ xám thành ảnh nhị phân. Ý tưởng cơ bản đằng sau ngưỡng thích ứng là tính toán ngưỡng cho từng pixel trong ảnh dựa trên giá trị cường độ của các pixel xung quanh, thay vì sử dụng ngưỡng chung cho toàn bộ ảnh.

Cách tiếp cận này đặc biệt hữu ích trong các tình huống khi độ sáng trong ảnh không nhất quán, điều này có thể gây ra sự cố với các phương pháp tạo ngưỡng toàn cầu truyền thống. Ngưỡng thích ứng tính đến các biến thể cục bộ về cường độ hình ảnh và điều chỉnh ngưỡng cho phù hợp để tạo ra sự phân đoạn đối tượng trong ảnh tốt hơn.

Có một số thuật toán cho ngưỡng thích ứng, bao gồm ngưỡng trung bình cục bộ, ngưỡng trung bình trọng số cục bộ và ngưỡng Sauvola. Các thuật toán này khác nhau ở cách chúng tính toán ngưỡng cho từng pixel, nhưng tất cả chúng đều nhằm mục đích tạo ra sự phân đoạn tốt hơn cho các đối tượng trong ảnh.

Trong ngưỡng trung bình cục bộ, ngưỡng cho mỗi pixel được tính là giá trị cường độ trung bình của các pixel xung quanh. Trong ngưỡng trung bình có trọng số cục bộ, ngưỡng cho mỗi pixel được tính là giá trị trung bình có trọng số của các pixel xung quanh, trong đó các trọng số được xác định dựa trên khoảng cách của các pixel xung quanh với pixel mục tiêu.

Sauvola thresholding là một thuật toán mới hơn dựa trên các thuộc tính thống kê của cường độ hình ảnh. Nó tính toán ngưỡng cho mỗi pixel dưới dạng hàm của giá trị trung bình và độ lệch chuẩn của các giá trị cường độ trong một cửa sổ cục bộ xung quanh pixel.

Adaptive thresholding là một công cụ hữu ích trong xử lý hình ảnh và thị giác máy tính, đặc biệt là trong các tình huống khi ánh sáng trong ảnh không nhất quán. Bằng cách tính toán ngưỡng cho từng pixel dựa trên giá trị cường độ của các pixel xung quanh, nó có thể tạo ra sự phân đoạn đối tượng trong ảnh tốt hơn so với các phương pháp tạo ngưỡng toàn cầu truyền thống.

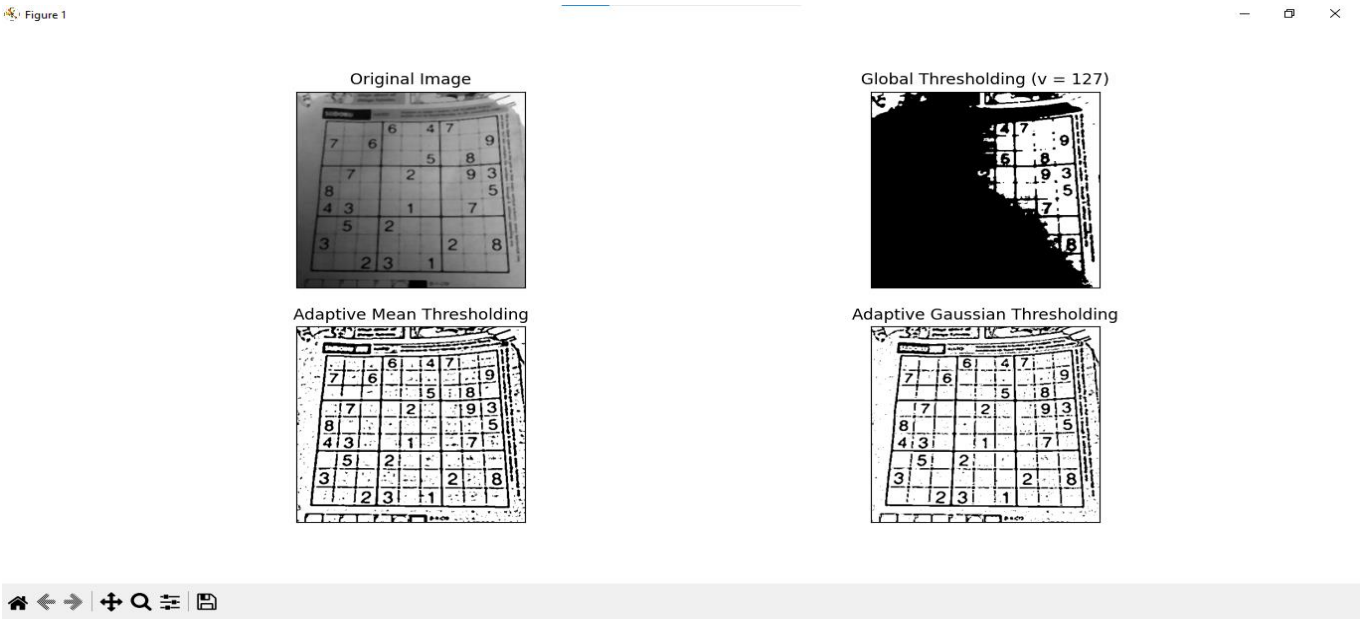
2.2.2 Ví dụ thuật toán Adaptive Thresholding

Thuật toán simple thresholding hoạt động khá tốt. Tuy nhiên, nó có 1 nhược điểm là giá trị ngưỡng bị/được gán toàn cục. Thực tế khi chụp, hình ảnh chúng ta nhận được thường bị ảnh hưởng của nhiễu, ví dụ như là bị phơi sáng, bị đèn flask, …

Một trong những cách được sử dụng để giải quyết vấn đề trên là chia nhỏ bức ảnh thành những vùng nhỏ (region), và đặt giá trị ngưỡng trên những vùng nhỏ đó -> adaptive thresholding ra đời. Opencv cung cấp cho chúng ta hai cách xác định những vùng nhỏ

Code demo:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('sudoku.png', 0)
img = cv.medianBlur(img, 5)
ret, th1 = cv.threshold(img, 127, 255, cv.THRESH_BINARY)
th2 = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_MEAN_C, \
                           cv.THRESH_BINARY, 11, 2)
th3 = cv.adaptiveThreshold(img, 255, cv.ADAPTIVE_THRESH_GAUSSIAN_C, \
                           cv.THRESH_BINARY, 11, 2)
titles = ['Original Image', 'Global Thresholding (v = 127)',
          'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]
for i in range(4):
    plt.subplot(2, 2, i + 1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])
plt.show()
```



Hình 2 Ví dụ thuật toán Adaptive Thresholding

2.2.3 Thuật toán tương tự

- Otsu's method: Đây là phương pháp phân ngưỡng toàn cục nhằm mục đích tìm giá trị ngưỡng giúp giảm thiểu sự khác biệt giữa các pixel nền trước và nền trong một hình ảnh. Nó đặc biệt hữu ích khi hình ảnh có biểu đồ hai chiều.
- Niblack thresholding: Đây là phương pháp tạo ngưỡng cục bộ tính toán ngưỡng cho từng pixel dựa trên giá trị trung bình và độ lệch chuẩn của các giá trị cường độ trong cửa sổ cục bộ xung quanh pixel. Ngưỡng sau đó được điều chỉnh bằng cách sử dụng một giá trị không đổi, giá trị này kiểm soát tầm quan trọng tương đối của giá trị trung bình và độ lệch chuẩn.
- Bernsen thresholding: Đây là phương pháp tạo ngưỡng cục bộ tính toán ngưỡng cho từng pixel dưới dạng giá trị cường độ tối thiểu và tối đa trong cửa sổ cục bộ xung quanh pixel và lấy giá trị trung bình của các giá trị này làm ngưỡng.
- Huang thresholding: Đây là phương pháp tạo ngưỡng toàn cầu tính toán ngưỡng dưới dạng hàm của các giá trị cường độ tối thiểu và tối đa trong ảnh và biểu đồ cường độ ảnh.

2.3 Thuật toán xử lý hình thái

2.3.1 Xử lý hình thái là gì?

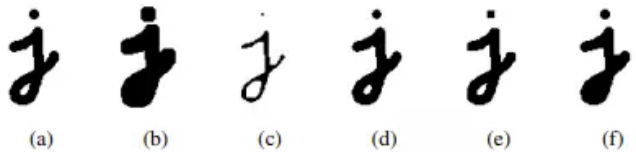
là các thao tác xử lý hình thái trong đồ họa máy tính và xử lý hình ảnh.

Erosion: Erosion là một thao tác tẩy rửa được sử dụng để loại bỏ các điểm nhiễu nhỏ trong hình ảnh. Nó sử dụng một mẫu (còn gọi là structuring element) để di chuyển qua hình ảnh và giảm kích thước các đối tượng trong hình ảnh.

Dilation: Dilation là một thao tác tăng kích thước được sử dụng để mở rộng các đối tượng trong hình ảnh. Nó sử dụng một mẫu để di chuyển qua hình ảnh và tăng kích thước các đối tượng trong hình ảnh.

Opening: Opening là một thao tác tẩy rửa đầu tiên là erode, sau đó là dilate. Nó được sử dụng để loại bỏ các điểm nhiễu nhỏ trong hình ảnh và đồng thời mở rộng các đối tượng còn lại.

Closing: Closing là một thao tác tẩy rửa đầu tiên là dilate, sau đó là erode. Nó được sử dụng để đóng lại các vết rỗng trong hình ảnh và đồng thời giảm noise.



a-original image
b-dilation
c-erosion
e-opening
f-closing

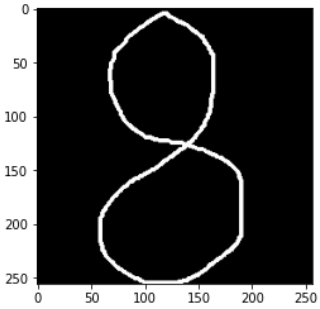
Hình 3 Các trạng thái xử lý hình thái

- Erosion và Dilation là hai phép toán xử lý hình thái thường được sử dụng trong computer vision và image processing.
- Erosion là một phép toán dùng để làm giảm kích thước của các đối tượng trong hình. Nó sẽ giảm diện tích của các đối tượng trong hình bằng cách "erode" (tiêu diệt) các pixel xung quanh các đối tượng. Điều này có thể giúp loại bỏ các noise hoặc sửa chữa các vết rạn nứt trong hình.
- Mục đích của phương pháp này sẽ giúp:
  - Loại bỏ những pixel nhiễu cô lập
  - Loại bỏ những pixel nhiễu xung quanh đối tượng giúp cho phần viền (cạnh) của đối tượng trở nên mịn hơn
  - Loại bỏ lớp viền (cạnh) của đối tượng giúp đối tượng trở nên nhỏ hơn và đặt những pixel viền đó trở thành lớp nền của đối tượng
- Dilation là phản đối của erosion. Nó sẽ tăng diện tích của các đối tượng trong hình bằng cách "dilate" (mở rộng) các pixel xung quanh các đối tượng. Điều này có thể giúp tăng kích thước của các đối tượng trong hình hoặc giảm tác động của noise trong hình. Mục đích của phương pháp này sẽ giúp:
  - Với những hình ảnh bị đứt nét có thể giúp nối liền ảnh lại
  - Với những pixel nhiễu xung quanh đối tượng sẽ trở thành viền của đối tượng
  - Giúp nổi bật đối tượng trong ảnh hơn
- Cả hai phép toán này đều cần sử dụng một mẫu (kernel) để xác định cách tác động vào hình. Kích thước của mẫu cũng có thể ảnh hưởng đến kết quả cuối cùng của phép toán.

2.3.2 Ví dụ xử lý hình thái

```
import cv2
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

img = cv2.imread('8.jpg')
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x7f8dc6d89f10>
```

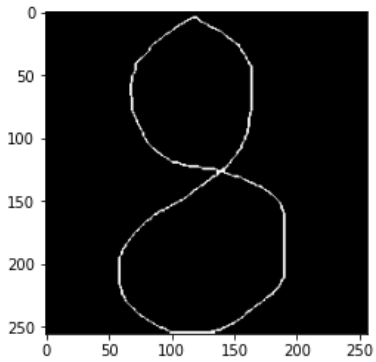


Hình 4 Input ảnh số 8

Erosion:

```
kernel = np.ones((3,3),np.uint8)
erosion = cv2.erode(img,kernel,iterations = 1)
plt.imshow(erosion)
```

```
<matplotlib.image.AxesImage at 0x7f8dc6fe4070>
```

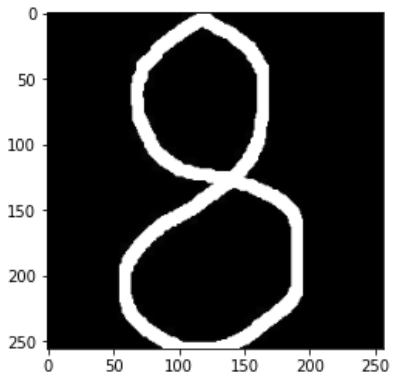


Hình 5 Phương pháp xử ảnh nhị phân Erosion

Dilation:

```
[15]  kernel = np.ones((6,6), np.uint8)
      dilation = cv2.dilate(img, kernel, iterations = 1)
      plt.imshow(dilation)
```

```
<matplotlib.image.AxesImage at 0x7f8dc6f750d0>
```

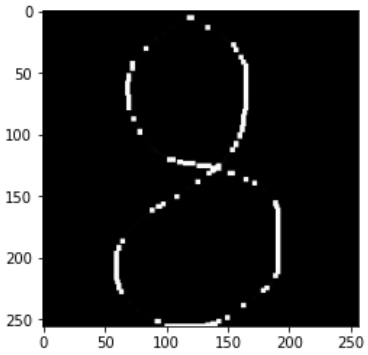


Hình 6 Phương pháp xử ảnh nhị phân dilation

Opening:

```
      kernel = np.ones((4,4), np.uint8)
      opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
      plt.imshow(opening)
```

```
<matplotlib.image.AxesImage at 0x7f8dc6c67670>
```

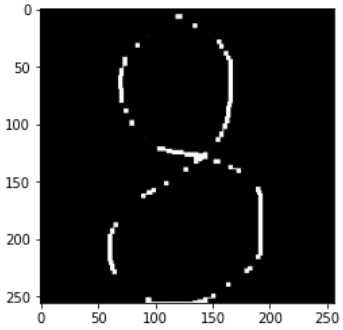


Hình 7 Phương pháp xử ảnh nhị phân Opening

Closing:

```
[17]  kernel = np.ones((4,4), np.uint8)
      closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)
      plt.imshow(closing)
```

```
<matplotlib.image.AxesImage at 0x7f8dc6c3d3a0>
```

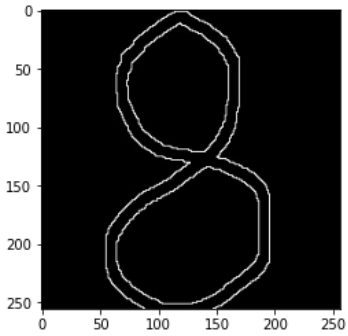


Hình 8 Phương pháp xử ảnh nhị phân Closing

Morphological Gradient:

```
[18]  kernel = np.ones((2,2), np.uint8)
      gradient = cv2.morphologyEx(dilation, cv2.MORPH_GRADIENT, kernel)
      plt.imshow(gradient)
```

```
<matplotlib.image.AxesImage at 0x7f8dc6c132b0>
```

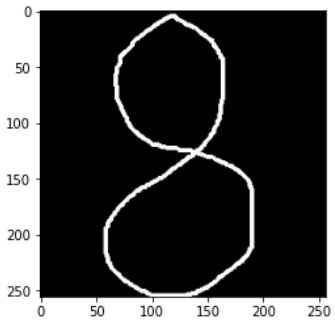


Hình 9 Phương pháp xử ảnh nhị phân Morphological Gradient

Top Hat:

```
[19]  kernel = np.ones((8,8), np.uint8)
      tophat = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel)
      plt.imshow(tophat)
```

```
<matplotlib.image.AxesImage at 0x7f8dc6b6a190>
```

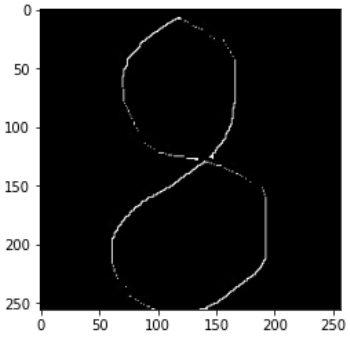


Hình 10 Phương pháp xử ảnh nhị phân Top Hat

Black Hat:

```
kernel = np.ones((4,4), np.uint8)
blackhat = cv2.morphologyEx(img, cv2.MORPH_BLACKHAT, kernel)
plt.imshow(blackhat)
```

```
<matplotlib.image.AxesImage at 0x7f8dc6b32070>
```



Hình 11 Phương pháp xử ảnh nhị phân Black Hat

2.3 Thuật toán Flood Filling

2.3.1 Thuật toán Flood Filling là gì?

- Flood filling là một thuật toán trong computer vision và image processing dùng để tô màu cho các đối tượng trong hình. Nó sử dụng một điểm bắt đầu để tìm ra các đối tượng có màu tương tự và tô màu cho chúng.
- Thuật toán sẽ tìm kiếm các đối tượng có màu giống nhau trên hình bằng cách duyệt theo các hướng từ điểm bắt đầu đã chỉ định. Nếu màu của đối tượng đang xét giống với màu của điểm bắt đầu, thì nó sẽ được tô màu và tiếp tục duyệt các đối tượng xung quanh nó. Quá trình này sẽ tiếp tục cho đến khi không còn đối tượng nào cần tô màu.
- Thuật toán Flood filling có thể dùng để tìm kiếm và tô màu cho các đối tượng trong hình, ví dụ như tìm kiếm và tô màu cho các đối tượng trong hình chữ nhật, tròn hoặc bất kỳ đối tượng nào khác.

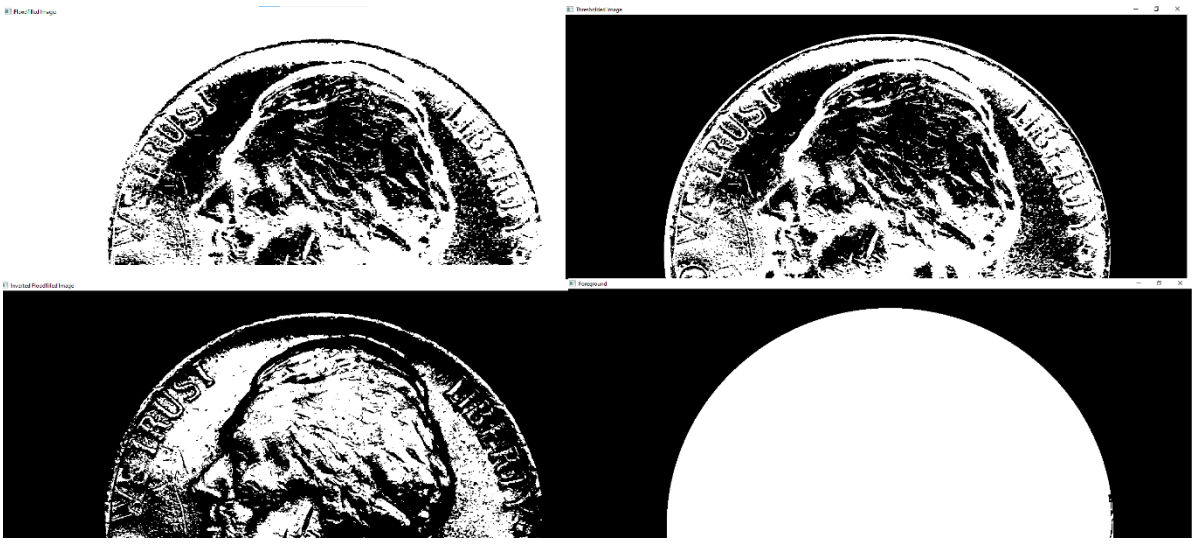
2.4.2 Ví dụ thuật toán Flood Filling

- Bước 1: Đọc trong hình ảnh.
- Bước 2: Ngưỡng hình ảnh đầu vào để thu được hình ảnh nhị phân.
- Bước 3: Lấp đầy từ pixel (0, 0). Lưu ý sự khác biệt giữa đầu ra của bước 2 và bước 3 là nền ở bước 3 hiện có màu trắng.
- Bước 4: Đảo ngược hình ảnh tràn ngập (tức là màu đen trở thành màu trắng và màu trắng trở thành màu đen).
- Bước 5: Kết hợp hình ảnh ngưỡng với hình ảnh tràn ngập ngược bằng cách sử dụng phép toán OR theo bit để có được mặt nạ tiền cảnh cuối cùng với các lỗ được lấp đầy. Hình ảnh trong Bước 4 có một số vùng màu đen bên trong đường viền. Theo thiết kế, hình ảnh ở Bước 2 đã được lấp đầy các lỗ đó. Vì vậy, chúng tôi kết hợp cả hai để có mặt nạ.

```
import cv2;
import numpy as np;

# Read image
im_in = cv2.imread("nickel.jpg", cv2.IMREAD_GRAYSCALE);

# Threshold.
```

```
# Set values equal to or above 220 to 0.
# Set values below 220 to 255.

th, im_th = cv2.threshold(im_in, 220, 255, cv2.THRESH_BINARY_INV);

# Copy the thresholded image.
im_floodfill = im_th.copy()

# Mask used to flood filling.
# Notice the size needs to be 2 pixels than the image.
h, w = im_th.shape[:2]
mask = np.zeros((h + 2, w + 2), np.uint8)

# Floodfill from point (0, 0)
cv2.floodFill(im_floodfill, mask, (0, 0), 255);

# Invert floodfilled image
im_floodfill_inv = cv2.bitwise_not(im_floodfill)

# Combine the two images to get the foreground.
im_out = im_th | im_floodfill_inv

# Display images.
cv2.imshow("Thresholded Image", im_th)
cv2.imshow("Floodfilled Image", im_floodfill)
cv2.imshow("Inverted Floodfilled Image", im_floodfill_inv)
cv2.imshow("Foreground", im_out)
cv2.waitKey(0)
```

Hình 12 Input ảnh nickel



Hình 13 Các lỗ đã được lấp đầy.

2.3.2 Thuật toán tương tự Flood Filling

a. Region growing: Đây là phương pháp phân đoạn ảnh dựa trên vùng bắt đầu từ điểm gốc và phát triển vùng bằng cách thêm các pixel tương tự với điểm gốc. Sự giống nhau có thể dựa trên cường độ, màu sắc, kết cấu hoặc các tính năng khác.

b. Watershed: Đây là phương pháp phân đoạn ảnh dựa trên phép biến đổi, xử lý ảnh như một bề mặt địa hình và xác định ranh giới của các đối tượng trong ảnh bằng cách tìm các đầu nguồn của ảnh.

c. GrabCut: Đây là phương pháp phân đoạn hình ảnh tương tác sử dụng kết hợp tăng vùng và cắt đồ thị để phân đoạn các đối tượng trong một hình ảnh. Người dùng cung cấp các điểm gốc ban đầu cho nền trước và nền sau, đồng thời thuật toán tinh chỉnh phân đoạn dựa trên đầu vào của người dùng.

d. Mean Shift: Đây là phương pháp phân đoạn ảnh dựa trên phân cụm sử dụng cửa sổ trượt để tìm các chế độ phân bố cường độ ảnh. Các chế độ tương ứng

với các đối tượng trong hình ảnh và thuật toán sử dụng các chế độ để phân chia hình ảnh thành các vùng.

Lợi ích của từng thuật toán:

- Ví dụ, Flood Fill là một thuật toán đơn giản và nhanh chóng, phù hợp với những hình ảnh có các đối tượng được xác định rõ ràng, không bị ngắt kết nối. Phát triển vùng là một lựa chọn tốt cho hình ảnh với các đối tượng có sự chuyển đổi cường độ dần dần và độ đo tương tự được xác định rõ.
- Watershed là một thuật toán mạnh mẽ dành cho hình ảnh có nhiều đối tượng và ranh giới chồng chéo, nhưng thuật toán này có thể chậm và dễ bị phân đoạn quá mức. GrabCut là một lựa chọn tốt cho hình ảnh có các đối tượng có thể nhìn thấy một phần và rất phù hợp để phân đoạn hình ảnh tương tác.
- Mean Shift là một lựa chọn tốt cho các hình ảnh có phân bố cường độ phức tạp, phi tuyến tính và nhiều đối tượng có phân bố cường độ khác nhau, nhưng nó có thể nhạy cảm với quá trình khởi tạo và có thể yêu cầu điều chỉnh các tham số cẩn thận.

2.4 Thuật toán Hough Transform

2.4.2 Thuật toán Hough Transform là gì?

- Hough Transform là một kỹ thuật thị giác máy tính được sử dụng để phát hiện đối tượng trong ảnh. Ban đầu nó được phát triển để phát hiện các đường trong ảnh, nhưng đã được mở rộng để phát hiện các hình dạng khác như hình tròn, hình elip và hình trụ tổng quát.
- Ý tưởng cơ bản của Hough Transform là biểu diễn một hình dạng trong ảnh dưới dạng một phương trình toán học và sau đó tìm kiếm các trường hợp của phương trình đó trong ảnh. Ví dụ: một đường thẳng trong ảnh có thể được biểu diễn dưới dạng một phương trình ở dạng $y = mx + b$, trong đó m là hệ số góc của đường thẳng và b là tung độ gốc của y. Thuật toán Hough Transform tìm kiếm tất cả các trường hợp của phương trình trong hình ảnh và tích lũy các

phiếu bầu trong biểu đồ 2D. Các đỉnh trong biểu đồ tương ứng với các đường trong ảnh.

- Biến đổi Hough có thể được thực hiện theo hai cách: Biến đổi Hough tiêu chuẩn và Biến đổi Hough theo xác suất. Hough Transform tiêu chuẩn là một phương pháp tính toán tốn kém, đòi hỏi dung lượng bộ nhớ lớn, nhưng nó cung cấp kết quả chính xác cho hình ảnh có ít dòng. Biến đổi Hough xác suất là một phương pháp hiệu quả hơn sử dụng kỹ thuật lấy mẫu ngẫu nhiên để giảm chi phí tính toán và yêu cầu bộ nhớ, nhưng nó có thể không chính xác như Biến đổi Hough tiêu chuẩn.
- Hough Transform được sử dụng rộng rãi trong các ứng dụng thị giác máy tính khác nhau, chẳng hạn như phát hiện làn đường trong xe tự lái, nhận dạng hình dạng và phát hiện đối tượng. Khả năng phát hiện các đường và hình dạng trong hình ảnh của nó, ngay cả khi chúng bị che khuất một phần hoặc có độ tương phản thấp, làm cho nó trở thành một công cụ hữu ích trong xử lý hình ảnh.
- Hough Transform là một kỹ thuật thị giác máy tính linh hoạt và mạnh mẽ được sử dụng rộng rãi để phát hiện đối tượng trong ảnh. Khả năng phát hiện các đường và hình dạng trong hình ảnh cũng như khả năng chống tắc nghẽn và độ tương phản thấp khiến nó trở thành một công cụ có giá trị cho nhiều tác vụ xử lý hình ảnh.

2.4.3 Thuật toán tương tự Hough Transform

- RANSAC (Random Sample Consensus): RANSAC là một thuật toán lặp được sử dụng để ước tính các tham số của một mô hình toán học từ một tập hợp dữ liệu được quan sát có chứa các giá trị ngoại lai. RANSAC tương tự như Hough Transform ở chỗ nó có thể phát hiện các đối tượng trong một hình ảnh, nhưng nó sử dụng một cách tiếp cận khác. RANSAC tạo ra các mẫu dữ liệu ngẫu nhiên và phù hợp với một mô hình cho từng mẫu. Mô hình

có nhiều giá trị nội tại nhất được coi là phù hợp nhất và được sử dụng để ước tính các tham số của đối tượng.

- Canny Edge Detector: Canny Edge Detector là một kỹ thuật thị giác máy tính được sử dụng để phát hiện cạnh trong hình ảnh. Nó tương tự như Hough Transform ở chỗ nó được sử dụng để phát hiện đối tượng, nhưng nó sử dụng một cách tiếp cận khác. Canny Edge Detector sử dụng thông tin độ dốc để phát hiện các cạnh trong ảnh, trong khi Hough Transform sử dụng thông tin hình dạng để phát hiện các đối tượng.
- Scale-Invariant Feature Transform (SIFT): SIFT là một kỹ thuật thị giác máy tính được sử dụng để nhận dạng đối tượng trong hình ảnh. Nó tương tự như Hough Transform ở chỗ nó được sử dụng để phát hiện đối tượng, nhưng nó sử dụng một cách tiếp cận khác. SIFT sử dụng các tính năng cục bộ như các điểm khóa không thay đổi tỷ lệ để phát hiện các đối tượng trong một hình ảnh.
- Speeded Up Robust Features (SURF): SURF là một kỹ thuật thị giác máy tính được sử dụng để nhận dạng đối tượng trong hình ảnh. Nó tương tự như SIFT và Hough Transform ở chỗ nó được sử dụng để phát hiện đối tượng, nhưng nó sử dụng một cách tiếp cận khác. SURF sử dụng các điểm khóa không thay đổi tỷ lệ và thuật toán tính toán nhanh để phát hiện các đối tượng trong hình ảnh.

2.4.4 Ví dụ thuật toán Hough Transform

Các bước để ví dụ:

Bước 1: Load an image

Bước 2: Phát hiện các cạnh của hình ảnh bằng cách sử dụng Canny detector:

Bây giờ chúng ta sẽ áp dụng Hough Line Transform. Chúng ta sẽ sử dụng cả hai chức năng OpenCV có sẵn cho mục đích này.

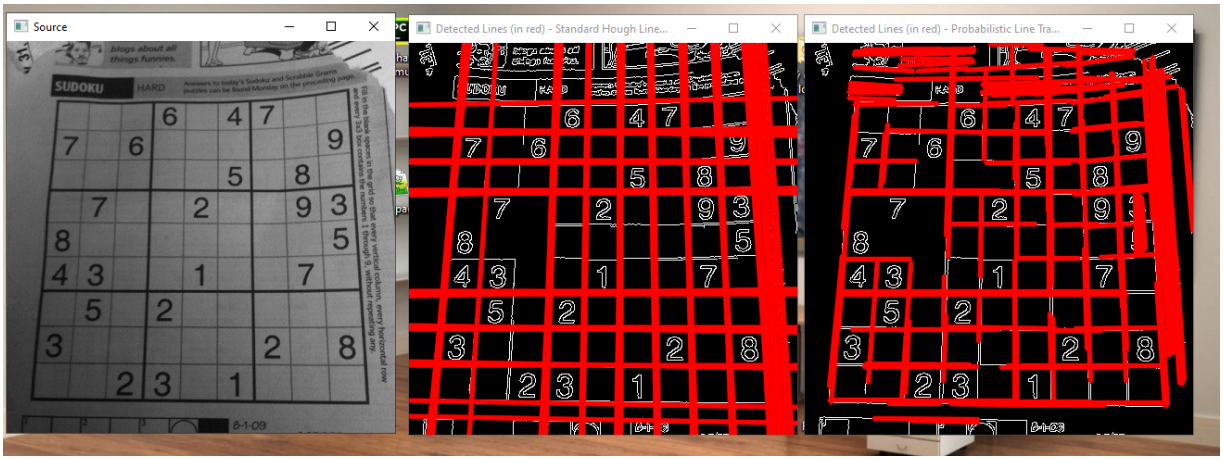
Bước 3: Standard Hough Line Transform. Bạn áp dụng Transform và sau đó bạn hiển thị kết quả bằng cách vẽ các đường.

Bước 4: Probabilistic Hough Line Transform First. Bạn áp dụng phép biến đổi, sau đó, hiển thị kết quả bằng cách vẽ các đường.

Bước 5: Hiển thị hình ảnh gốc và các dòng được phát hiện: Chờ cho đến khi người dùng thoát khỏi chương trình.

```
import sys
import math
import cv2 as cv
import numpy as np


def main(argv):
    default_file = 'sudoku.png'
    filename = argv[0] if len(argv) > 0 else default_file
    # Loads an image
    src = cv.imread(cv.samples.findFile(filename), cv.IMREAD_GRAYSCALE)
    # Check if image is loaded fine
    if src is None:
        print('Error opening image!')
        print('Usage: hough_lines.py [image_name -- default ' +
default_file + '] \n')
        return -1

    dst = cv.Canny(src, 50, 200, None, 3)

    # Copy edges to the images that will display the results in BGR
    cdst = cv.cvtColor(dst, cv.COLOR_GRAY2BGR)
    cdstP = np.copy(cdst)

    lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)

    if lines is not None:
        for i in range(0, len(lines)):
            rho = lines[i][0][0]
            theta = lines[i][0][1]
            a = math.cos(theta)
            b = math.sin(theta)
            x0 = a * rho
            y0 = b * rho
            pt1 = (int(x0 + 1000 * (-b)), int(y0 + 1000 * (a)))
            pt2 = (int(x0 - 1000 * (-b)), int(y0 - 1000 * (a)))
            cv.line(cdst, pt1, pt2, (0, 0, 255), 3, cv.LINE_AA)

    linesP = cv.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)

    if linesP is not None:
```

```
        for i in range(0, len(linesP)):
            l = linesP[i][0]
            cv.line(cdstP, (l[0], l[1]), (l[2], l[3]), (0, 0, 255), 3,
cv.LINE_AA)

    cv.imshow("Source", src)
    cv.imshow("Detected Lines (in red) - Standard Hough Line
Transform", cdst)
    cv.imshow("Detected Lines (in red) - Probabilistic Line Transform",
cdstP)

    cv.waitKey()
    return 0


if __name__ == "__main__":
    main(sys.argv[1:])
```



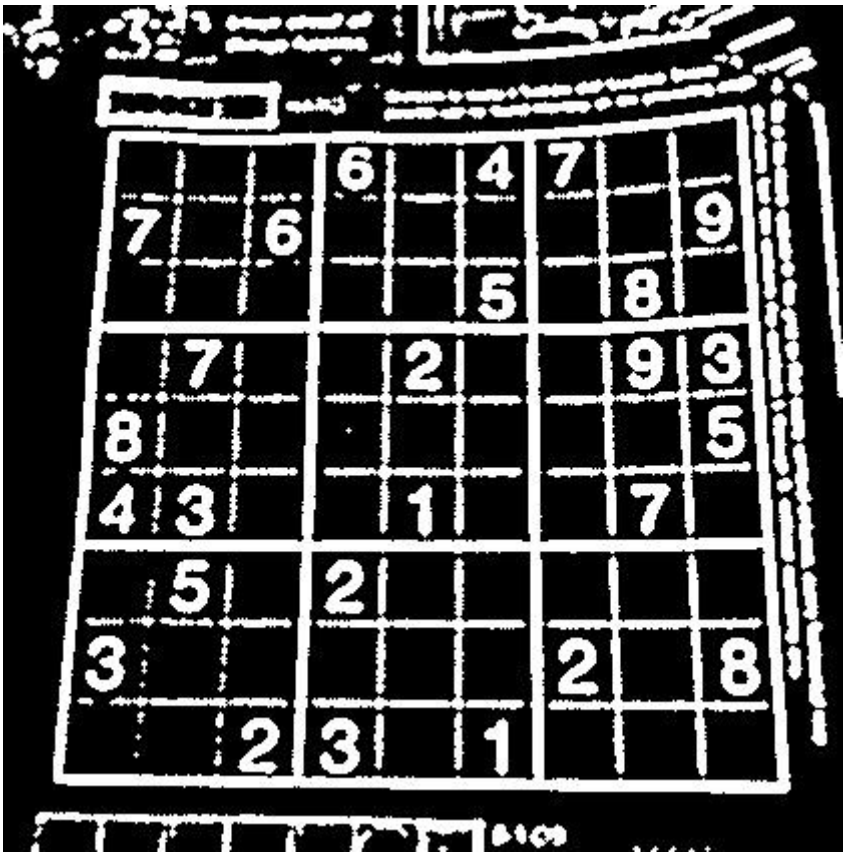
Hình 14 Ví dụ thuật toán Hough Transform input ảnh sodoku

CHƯƠNG 3: DEMO VÀ KẾT QUẢ

Step 1: Segmenting the Sudoku puzzle

Ở bước này, dùng các kĩ thuật xử lý ảnh như: thresholding, Gausian blur, xử lý hình thái,… để tạo thành một bức ảnh nhị phân

Kết quả ta được:



Hình 15 Step 1 phân biệt các dường kẻ dọc và ngang input ảnh Sodoku

Step 2: Detecting puzzle

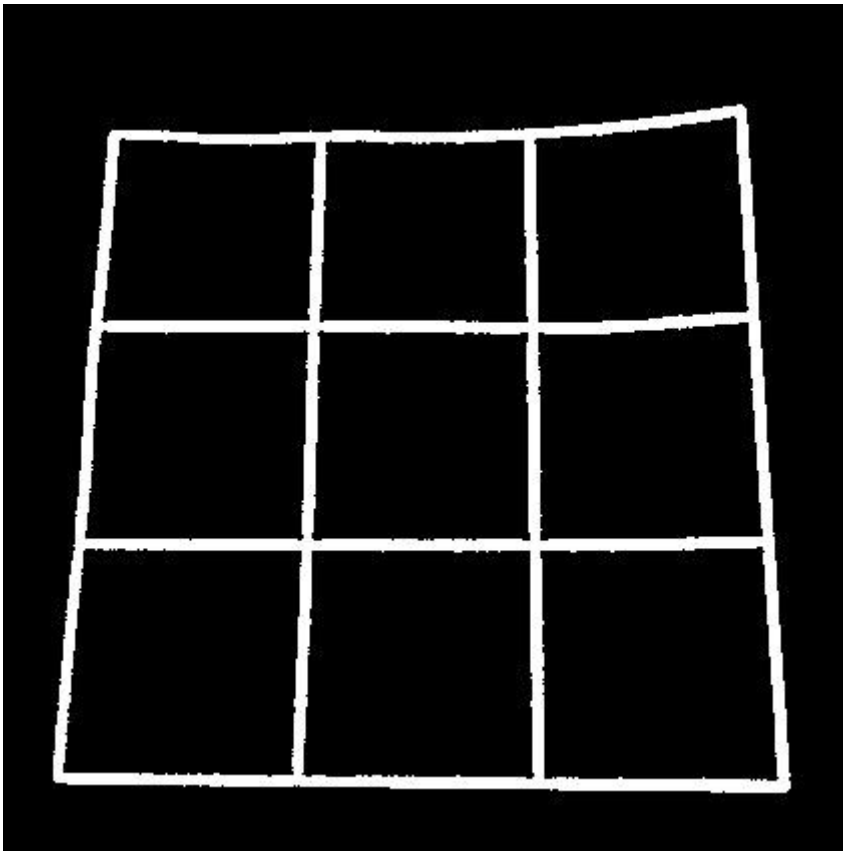
Xử dụng các kĩ thuật xử lý ảnh, nhận diện được puzzle. Bao gồm các bước cụ thể như sau:

- Tận dụng lại bức ảnh nhị phân ở bước 1
- Lặp qua từng pixel ảnh:
    - Dùng kĩ thuật tô chàm (floodFill) tô màu với các pixel có giá trị lớn hơn hoặc bằng 128, ta sẽ tô bằng 1 giá trị màu do ta đặt

- floodFile sẽ trả về giá trị là diện tích mà nó tô được
- nếu phần diện tích đó lớn hơn max (giá trị diện tích tô chàm lớn nhất hiện tại), thì set lại giá trị của max; đồng thời lưu lại giá trị của điểm pixel đó

- Sau khi lặp qua hết các pixel, ta tô chàm cho phần diện tích lớn nhất, bắt đầu từ điểm mà ta đã lưu
- Sau đó lặp qua các pixel một lần nữa, nếu pixel nào có giá trị bằng với giá trị tô chàm ở lần lặp trước và không trùng với điểm của max, ta sẽ tô phàn đó với giá trị là 0
- Sau khi tô xong, ta dùng kĩ thuật Erosion, để giảm đi các phần nhiễu
- Kết quả ta được 1 bức ảnh với viền ngoài của puzzle

Kết quả ta được:



Hình 16 Step 2 phát hiện bảng input Sodoku

Step 3: Locating the puzzle

Ở bước này, ta thực hiện các bước tuần tự như sau:

- Sử dụng Hough lines để tìm ra các đường thẳng trong bức ảnh
- Sử dụng các công thức toán học, giữ lại các đường thẳng lớn và nằm gần 4 viền của puzzle
- Sau đó, ta tìm 4 giao điểm. Từ 4 giao điểm này ta sẽ được 4 đỉnh của hình vuông

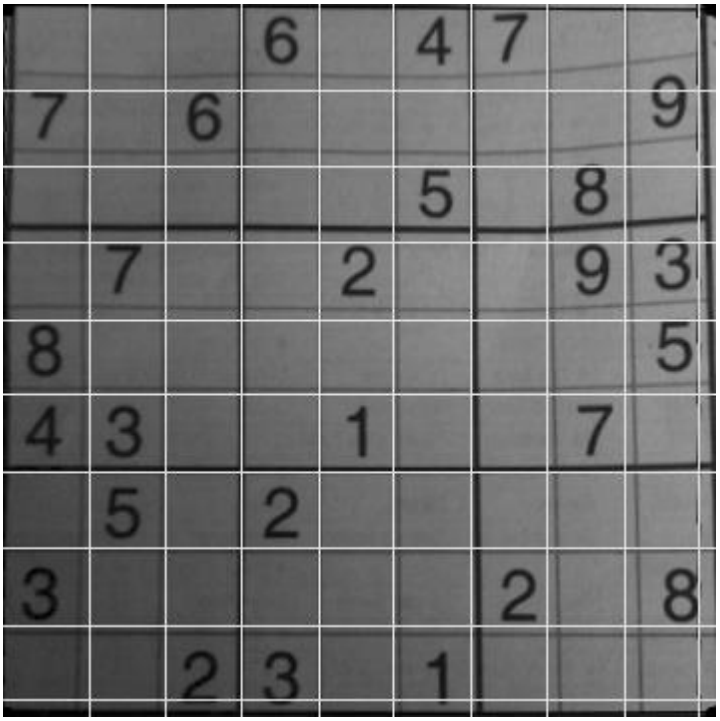
Hình 17 Step 4 định vị câu đố input ảnh Sodoku

Step 4: Fixing the image and accessing each cell

Ở bước này, ta có các bước:

- Từ vị trí 4 đỉnh, ta tìm được độ dài 4 cạnh của puzzle
- Tìm cạnh có độ dài lớn nhất

- Từ độ dài đó, ta sẽ cắt ra được hình vuông với 4 cạnh là độ dài đó. Việc này nhằm tránh khỏi việc puzzle bị cắt thiếu hoặc cắt nhầm

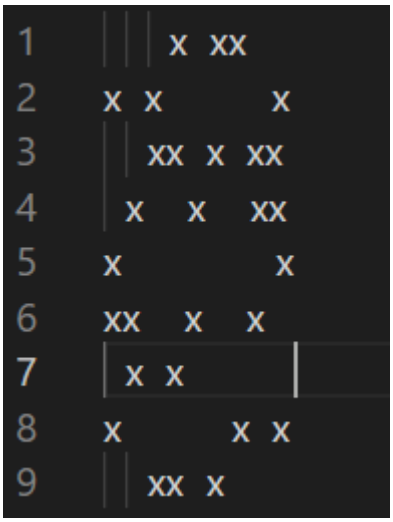


Hình 18 Step 4 sửa và access câu đố input ảnh Sodoku

Step 5: Identify the number

Sau bước 4, ta tiếp tục thực hiện:

- Chia nhỏ các bức ảnh ra thành 9 bức ảnh nhỏ
- Nhận diện chữ số cho từng bức ảnh nhỏ đó
- Nếu ảnh là số, thì ta ghi ‘x’ vào file output.txt, ngược lại thì để trống



Hình 19 Bàn cờ dưới dạng text input ảnh Sodoku

CHƯƠNG 4: KẾT LUẬN

5.1 Kết quả đạt được

- Hiểu, vận dụng được các kĩ thuật xử lý ảnh số được học trong chương trình
- Lập trình được các thuật toán xử lý ảnh bằng python
- Áp dụng thành thạo và kết hợp uyển chuyển các kĩ thuật xử lý ảnh với nhau
- Hiểu rõ hơn về ảnh số, giúp ích cho việc nghiên cứu Computer vision.

5.2 Những vấn đề chưa đạt được

- Code chưa được gọn gang
- Còn nhiều lỗi nhỏ
- Một vài chữ số chưa nhận diện được

5.3 Hướng phát triển

- Trong tương lai gần, sẽ hoàn thiện thuật toán
- Kết hợp thêm mô hình CNN giúp nhận diện số và giải Sudoku

TÀI LIỆU THAM KHẢO

1. https://aishack.in/tutorials/sudoku-grabber-opencv-plot/. Các bước xử lý ảnh Sodoku.
2. https://www.phamduytung.com/blog/2020-12-24-thresholding/ . Thuật toán adaptive thresholding.
3. https://viblo.asia/p/xu-ly-anh-erosion-dilation-opening-closing-4dbZNpWq5YM. Thuật toán xử lý hình thái
4. https://learnopencv.com/filling-holes-in-an-image-using-opencv-python-c/. Thuật toán Filling Holes