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

BÁO CÁO GIỮA KÌ MÔN
XỬ LÝ ẢNH SỐ

Người hướng dẫn: **GV.TRỊNH HÙNG CƯỜNG**

Người thực hiện: **TRẦN THỊ VẸN – 52100674**

NGUYỄN XUÂN TOÀN - 51800137

Lớp **: 21050301**

Khoá **: 25**

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

BÁO CÁO GIỮA KÌ MÔN
XỬ LÝ ẢNH SỐ

Người hướng dẫn: **GV. TRỊNH HÙNG CƯỜNG**

Người thực hiện: **TRẦN THỊ VẸN – 52100674**

NGUYỄN XUÂN TOÀN - 51800137

Lớp **: 21050301**

Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Trong suốt quá trình học tập và rèn luyện, chúng em đã nhận được rất nhiều sự giúp đỡ tận tình, sự quan tâm, chăm sóc của GV. Ngoài ra, chúng em còn được GV truyền đạt những kiến thức, phương pháp mới về toán hay ho và thú vị, thầy cô còn giúp sinh viên có được nhiều niềm vui trong việc học và cảm thấy thoải mái, … Chúng em xin chân thành cảm ơn các thầy cô rất nhiều trong suốt quá trình học tập này!

Bởi lượng kiến thức của chúng em còn hạn hẹp và gặp nhiều vấn đề trong quá trình học nên báo cáo này sẽ còn nhiều thiếu sót và cần được học hỏi thêm. Chúng em rất mong em sẽ nhận được sự góp ý của quý thầy cô về bài báo cáo này để chúng em rút kinh nghiệm trong những môn học sắp tới. Cuối cùng, chúng em xin chân thành cảm ơn quý thầy cô.

TP Hồ Chí Minh, ngày 01 tháng 11 năm 2023
Sinh viên:
Trần Thị Vẹn – 52100674
Nguyễn Xuân Toàn - 51800137

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

Tôi xin cam đoan đây là sản phẩm đồ án của chúng tôi và được sự hướng dẫn của GV. Trịnh Hùng Cường. 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 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 tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày 01 tháng 11 năm 2023

Tác giả

(ký tên và ghi rõ họ tên)

Trần Thị Vẹn

Nguyễn Xuân Toàn

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

Phương pháp kết hợp các khoảng màu đỏ cùng việc sử dụng morphology làm tăng cường màu sắc đặc biệt hiệu quả.

Xử lý từng màu sắc độc lập giúp giữ lại những đặc điểm riêng biệt của từng màu, cải thiện độ chính xác của quá trình xử lý.

Sự kết hợp của các kỹ thuật thresholding, morphology, và Canny edge detection mang lại hiệu suất tốt trong việc nhận diện và đánh dấu vùng quan trọng trên ảnh.

Như vậy, phương pháp đề xuất trong nghiên cứu đã mang lại những kết quả khả quan và có tiềm năng ứng dụng rộng rãi trong lĩnh vực xử lý ảnh và nhận dạng vùng quan trọng.

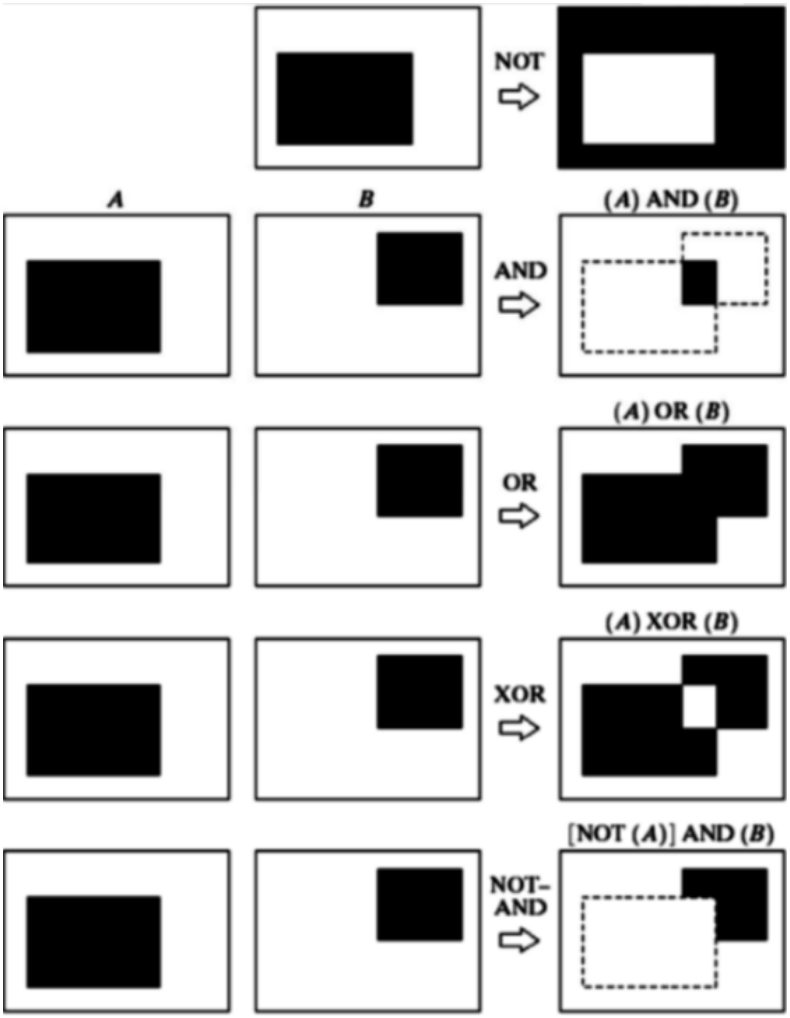
MỤC LỤC

DANH MỤC HÌNH VẼ

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

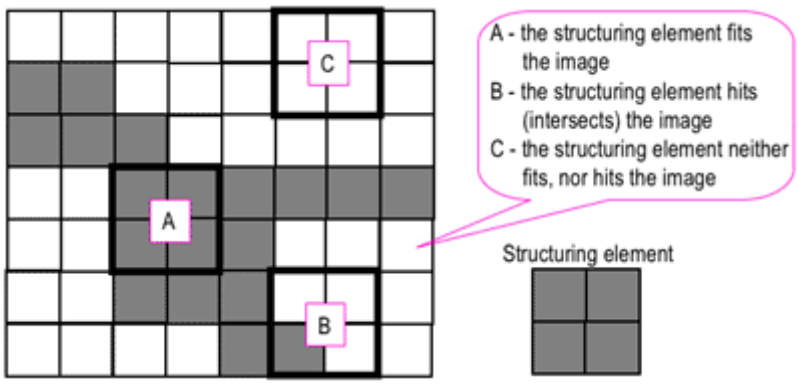
1.1 Xử lý hình thái học (Morphological Image Processing)

Morphological Image Processing là tập hợp các phép toán phi tuyến tính (non-linear) tác động đến hình dạng hoặc hình thái của các điểm nhị phân trong ảnh. Dựa trên các phép toán AND OR XOR NOT để biến đổi các điểm nhị phân.



Hình 1: Phép toán cơ bản trong Morphological Image Processing

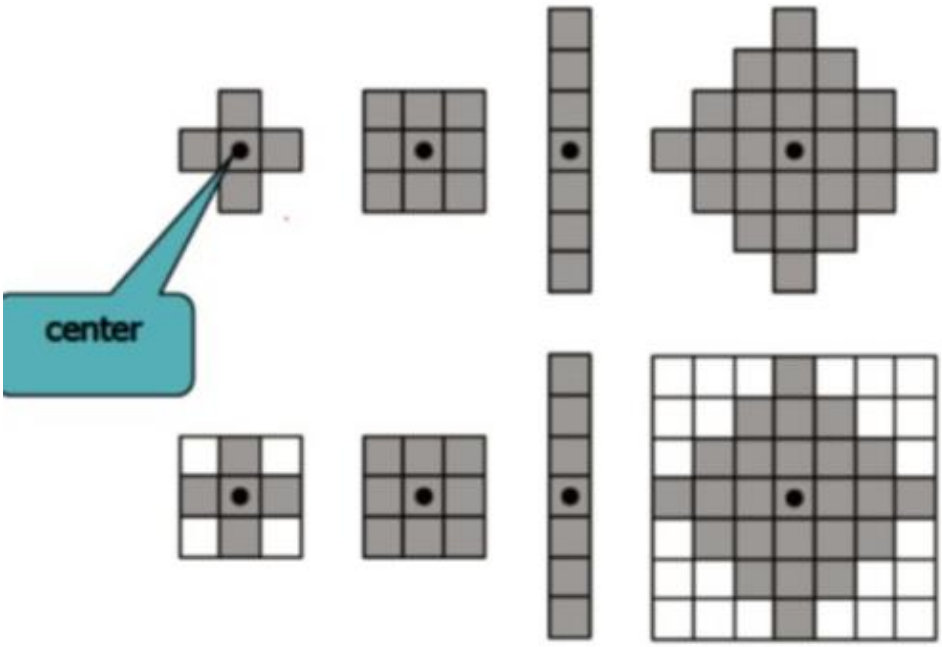
Morphological Operator sẽ sử dụng một cấu trúc nhỏ quét qua và áp dụng toàn bộ ảnh được gọi là Structuring Element (SE). SE sẽ quét qua toàn bộ vị trí trong hình ảnh và so sánh với các pixel lân cận nằm trong vùng sẽ tương ứng.



Hình 2: Structuring Element

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.
- Pixel có giá trị 0 được bỏ qua trong quá trình tính toán.
- Luôn có một pixel làm mốc trong ma trận SE.



Hình 3: Structuring Element hoạt động

Structuring Element hoạt động giống như ma trận tích chập (convolution kernel) trong lọc ảnh tuyến tính (linear image filtering). SE sẽ dịch chuyển toàn bộ ảnh với điểm mốc.

Thay vì tạo các phần tử cấu trúc theo cách thủ công trong các ví dụ trước với sự trợ giúp của Numpy. OpenCV có một hàm cv.getStructuringElement(). Bạn chỉ cần truyền hình dạng và kích thước của kernel là bạn sẽ có được kernel như ý.

```
# Hình chữ nhật
>>> cv.getStructuringElement(cv.MORPH_RECT,(5,5))
    array([[1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1]], dtype=uint8)
# Elliptical Kernel
>>> cv.getStructuringElement(cv.MORPH_ELLIPSE,(5,5))
    array([[0, 0, 1, 0, 0],
           [1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1],
           [0, 0, 1, 0, 0]], dtype=uint8)
# Cross-shaped Kernel
>>> cv.getStructuringElement(cv.MORPH_CROSS,(5,5))
    array([[0, 0, 1, 0, 0],
           [0, 0, 1, 0, 0],
           [1, 1, 1, 1, 1],
           [0, 0, 1, 0, 0],
           [0, 0, 1, 0, 0]], dtype=uint8)
```

1.1.1 Phép giãn (Dilation)

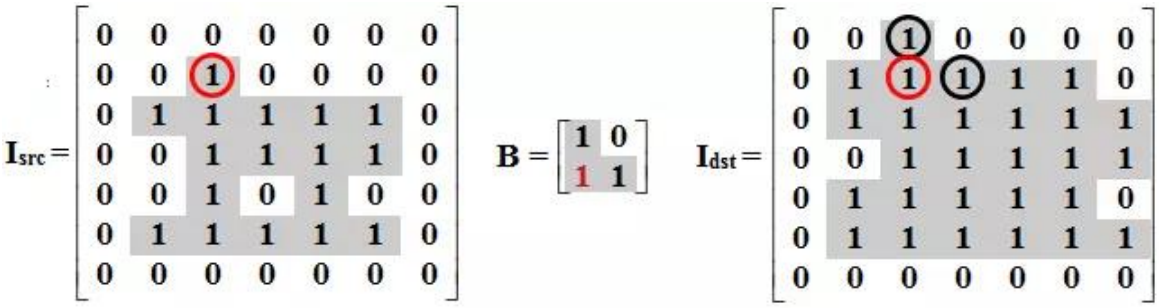
Phép toán giãn nở được được định nghĩa theo công thức dưới đây, A là đối tượng trong ảnh, B là một cấu trúc phần tử ảnh. Phép toán này có tác dụng làm cho đối tượng ban đầu trong ảnh tăng lên về kích thước (giãn nở ra).

$$A \oplus B = \{c \mid c = a + b,\ a \in A, b \in B\}$$

- A: Ma trận điểm ảnh của ảnh nhị phân.
- B: Là phần tử cấu trúc.

Phép giãn nở (Dilation) ảnh sẽ cho ra một tập điểm ảnh c thuộc D(i), bạn hoàn toàn dễ dàng thấy rằng đây là một phép tổng giữa A và B.

dilation = cv.dilate(img,kernel,iterations = 1)



Hình 4: Phép tính ma trận phép giãn

Hình 5: Ảnh trước và sau Dilation

1.1.2 Phép co (Erosion)

Phép toán co (Erosion) là một trong hai hoạt động cơ bản (khác phép giãn nở) trong hình thái học có ứng dụng trong việc giảm kích thước của đối tượng, tách rời các đối tượng gần nhau, làm mảnh và tìm xương của đối tượng.

$$A \ominus B = \{c \mid (B)_c \subseteq A\}$$

Trong đó:

- A: Ma trận điểm ảnh của ảnh nhị phân.
- B: Là phần tử cấu trúc.

Phép co ảnh sẽ cho ra một tập điểm ảnh c thuộc A, nếu bạn đi chuyển phần tử cấu trúc B theo c, thì B nằm trong đối tượng A. E(i) là một tập con của tập ảnh bị co A. Chú ý: Nhận xét này không toàn toàn đúng với trường hợp phần tử cấu trúc B không có gốc (Origin) hay nói cách khác là gốc (Origin) mang giá trị 0.

erosion = cv.erode(img,kernel,iterations = 1)

Hình 6: Ảnh trước và sau Erosion

1.1.3 Phép mở (Opening)

Phép toán mở (Opening) là sự kết hợp của phép co (Erosion) và phép giãn nở (Dilation), thực hiện phép co (Erosion) trước sau đó mới thực hiện phép giãn nở (Dilation). Giúp làm mượt các đường viền, phá vỡ các khe nhỏ, loại bỏ các đối tượng nhỏ, làm mượt các đỉnh lồi.

opening = cv.morphologyEx(img, cv.MORPH_OPEN, kernel)

$$A \circ B = (A \ominus B) \oplus B.$$

Hình 7: Ảnh trước và sau khi Opening

1.1.4 Phép đóng (Closing)

Phép toán đóng (Closing) là sự kết hợp của phép co (Erosion) và phép giãn nở (Dilation), thực hiện phép giãn nở (Dilation) trước sau đó mới thực hiện phép co

(Erosion). Giúp làm mượt các đường viền, loại bỏ các lỗ nhỏ, làm mượt các đỉnh khe hẹp.

closing = cv.morphologyEx(img, cv.MORPH_CLOSE, kernel)

$$A \bullet B = (A \oplus B) \ominus B.$$

Hình 8: Ảnh trước và sau khi Closing

1.1.5 Morphological Gradient

Đó là sự khác biệt giữa sự giãn nở và xói mòn của hình ảnh. Kết quả sẽ trông giống như đường viền của đối tượng.

gradient = cv.morphologyEx(img, cv.MORPH_GRADIENT, kernel)

Hình 9: Ảnh sau khi Morphological Gradient

1.2 Phép toán Bitwise

Bitwise operations bao gồm các phép AND, OR, XOR, NOT. Bảng chân lý của các phép toán này được thể hiện ở hình dưới đây:

Khi thực hiện trên ảnh, X, Y sẽ đại diện cho giá trị pixel của ảnh khi đó:

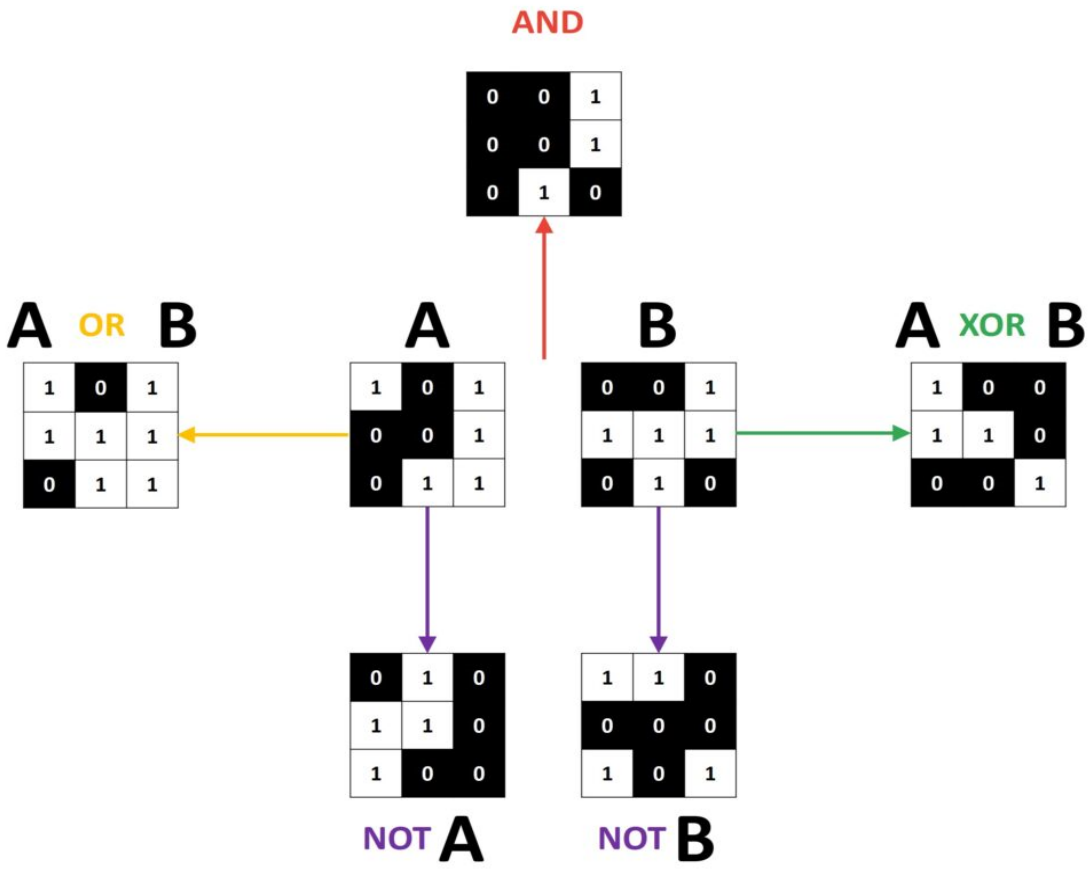
- AND có giá trị 1 khi 2 pixel có giá trị lớn hơn 0.
- OR có giá trị 1 nếu một trong 2 pixel có giá trị lớn hơn 0.

- XOR có giá trị 1 nếu một trong 2 pixel có giá trị lớn hơn 0, nhưng đồng thời pixel còn lại phải có giá trị 0.
- NOT Đảo ngược giá trị pixel.

Với Bitwise Operations, phép toán này chỉ thực hiện trên ảnh nhị phân. Do đó để thực hiện ta cần đưa ảnh về nhị phân với quy luật đơn giản là pixel nào có giá trị lớn hơn 0 thì sẽ mang giá trị 1 và còn lại những pixel nào có giá trị 0 thì vẫn giữ nguyên giá trị là 0.

mask_red = cv2.bitwise_or(mask_red1, mask_red2)

color_result = cv2.bitwise_and(image, image, mask=mask_color)



Hình 10: Các thuật toán Bitwise

1.3 Chuyển đổi màu hệ màu

1.3.1 Chuyển đổi không gian màu RGB sang HSV

RGB là không gian màu rất phổ biến được dùng trong đồ họa máy tính và nhiều thiết bị kĩ thuật số khác. Ý tưởng chính của không gian màu này là sự kết hợp của 3 màu sắc cơ bản : màu đỏ (R, Red), xanh lục (G, Green) và xanh lơ (B, Blue) để mô tả tất cả các màu sắc khác.

HSV và cũng gần tương tự như HSL là không gian màu được dùng nhiều trong việc ch ỉnh sữa ảnh, phân tích ảnh và một phần của lĩnh vực thị giác máy tính. Hệ không gian này dựa vào 3 thông số sau để mô tả màu sắc H = Hue: màu sắc, S = Saturation: độ đậm đặc, sự bảo hòa, V = value: giá trị cường độ sáng.

hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

- Lấy R,G,B từ ảnh và tính R’ ; G’, B’ nằm trong khoảng [0;1]

R '= R / 255

G '= G / 255

B '= B / 255

- Tìm giá trị min và max trong 3 kênh màu:

Cmax = max (R ', G ', B ')

Cmin = min (R ', G ', B ')

Δ = Cmax – Cmin

- Tính toán Hue:

+ Nếu Cmax = R, Hue = 60 * ((G - B) / (Δ))

+ Nếu Cmax = G, Hue = 60 * ((B - R) / (Δ)) + 120

+ Nếu Cmax = B, Hue = 60 * ((R - G) / (Δ)) + 240

+ Nếu Cmax = 0, Hue = 0

- Tính toán Saturation:

+ Cmax = 0, Saturation = 0

+ Cmax khác 0, Saturation = (Δ) / Cmax

- Tính toán Value:

Value = Cmax

1.3.2 Chuyển đổi ảnh màu RGB thành ảnh xám (grayscale)

Để chuyển từ ảnh màu sang ảnh xám, ta thực hiện theo công thức sau

Độ sáng điểm ảnh = 0.2989*Red + 0.5870*Green + 0.1140*Blue

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

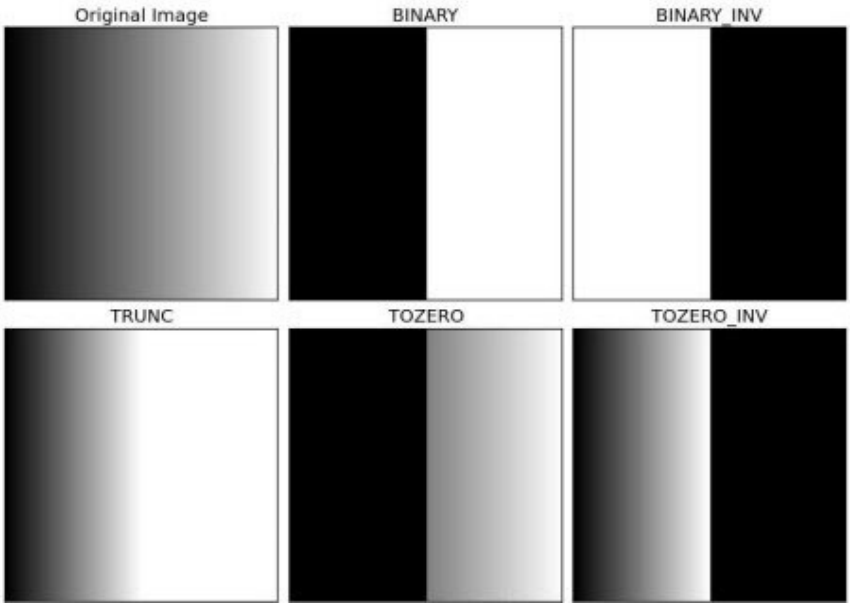
1.4 Simple Thresholding

Hàm sử dụng là threshold , tham số đầu tiên là 1 ảnh xám, tham số thứ 2 là giá trị ngưỡng, tham số thứ 3 maxval là giá trị được gán nếu giá pixel lớn hơn giá trị ngưỡng, tham số thứ 4 là loại phân ngưỡng. Tùy theo các loại phân ngưỡng mà pixel được gán giá trị khác nhau:

- THRESH_TOZERO
  Nếu giá trị pixel lớn hơn ngưỡng thì giữ nguyên giá trị
  Ngược lại gán bằng 0
- THRESH_TOZERO_INV
  Nếu giá trị pixel lớn hơn ngưỡng thì gán giá trị bằng 0
  Ngược lại giữ nguyên
- THRESH_BINARY
  Nếu giá trị pixel lớn hơn ngưỡng thì gán bằng maxval
  Ngược lại bằng gán bằng 0
- THRESH_BINARY_INV
  Nếu giá trị pixel lớn hơn ngưỡng thì gán bằng 0
  Ngược lại bằng gán bằng maxval
- THRESH_TRUNC
  Nếu giá trị pixel lớn hơn ngưỡng thì gán giá trị bằng ngưỡng
  Ngược lại giữ nguyên giá trị

_, thresholded = cv2.threshold(image, lower_threshold, 255, cv2.THRESH_TOZERO)

_, thresholded_inv = cv2.threshold(thresholded, upper_threshold, 255, cv2.THRESH_TOZERO_INV)



Hình 11: Minh họa trực quan với ngưỡng bằng 127

1.5 Gaussian Blur

Gaussian Blur là một kỹ thuật xử lý ảnh phổ biến trong lĩnh vực thị giác máy tính và xử lý ảnh số. Nó được sử dụng để làm mờ hoặc làm trơn ảnh bằng cách áp dụng một bộ lọc Gaussian lên mỗi điểm ảnh trong ảnh đầu vào.

Bộ lọc Gaussian là một bộ lọc phi tuyến tính có tính chất làm mờ ảnh dựa trên hàm Gaussian. Hàm Gaussian là một hàm có đặc trưng hình dạng hình chuông và phân phối xác suất Gauss. Bằng cách áp dụng bộ lọc Gaussian, các điểm ảnh gần nhau sẽ có tác động lên nhau và tạo ra hiệu ứng làm mờ ảnh.

Gaussian Blur được sử dụng trong nhiều ứng dụng xử lý ảnh như giảm nhiễu, làm trơn, tạo hiệu ứng mờ, tạo ảnh nền, trích xuất đặc trưng và nhiều ứng dụng khác. Nó giúp làm giảm nhiễu và loại bỏ các chi tiết không mong muốn trong ảnh, tạo ra hiệu ứng mờ mịn và tạo cảm giác mềm mại, tăng khả năng trích xuất các đặc trưng quan trọng trong ảnh.

GaussianBlur(src, ksize, sigmaX[, dst[, sigmaY[, borderType]]])

blurred = cv2.GaussianBlur(binary_image, (5, 5), 0)

- src -Nó được sử dụng để nhập một Hình ảnh.
- dst -Nó là một biến lưu trữ Hình ảnh đầu ra.

- ksize -Nó định nghĩa Kích thước Kernel Gaussian [chiều rộng chiều cao]. Chiều cao và chiều rộng phải là số lẻ (1,3,5,…) và có thể có các giá trị khác nhau. Nếu ksize được đặt thành [0,0], thì ksize được tính từ giá trị sigma.
- sigmaX – Dẫn xuất tiêu chuẩn hạt nhân dọc theo trục X. (hướng nằm ngang).
- sigmaY – Dẫn xuất tiêu chuẩn hạt nhân dọc theo trục Y (hướng dọc). Nếu sigmaY = 0 thì giá trị sigmaX được lấy cho sigmaY.
- borderType – Đây là các ranh giới hình ảnh được chỉ định trong khi hạt nhân được áp dụng trên các đường viền hình ảnh. Loại đường viền có thể có là:

+ cv.BORDER_CONSTANT

+ cv.BORDER_REPLICATE

+ cv.BORDER_DEFAULT…

1.6 Canny Detection

Phát hiện cạnh Canny (Canny edge detector) là một thuật toán bao gồm nhiều giai đoạn để phát hiện một loạt các cạnh trong hình ảnh. Nó được phát triển bởi John F. Canny vào năm 1986. Canny cũng đưa ra một lý thuyết tính toán về phát hiện cạnh giải thích tại sao kỹ thuật này hoạt động.

edged = cv2.Canny(blurred, 30, 150)

Thuật toán phát hiện cạnh Canny bao gồm 5 bước:

- Giảm nhiễu

Một cách để loại bỏ nhiễu là áp dụng Gaussian Filter giúp làm mịn ảnh. Để làm như vậy, kỹ thuật tích chập hình ảnh được áp dụng với Gaussian Kernel (kích thước 3×3, 5×5, 7×7, v.v.). Kích thước kernel phụ thuộc vào hiệu ứng làm mờ mong muốn. Về cơ bản, kernel càng nhỏ, hiệu ứng mờ càng ít. Ví dụ dưới đây sử dụng 5 x 5 Gaussian kernel.

Phương trình cho một kernel bộ lọc Gaussian có kích thước (2k + 1) × (2k + 1):

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1)$$

- Tính toán Gradient độ xám của ảnh

Bước tính toán Gradient độ xám phát hiện các cạnh thông qua cường độ và hướng của gradient độ xám. Các cạnh tương ứng với sự thay đổi cường độ sáng của pixel. Để phát hiện nó, cách dễ nhất là áp dụng các Filter làm nổi bật sự thay đổi cường độ này theo cả hai hướng: ngang (x) và dọc (y)

Sau Khi hình ảnh được làm mịn, các đạo hàm Ix và Iy w.r.t. x và y được tính. Việc này có thể được thực hiện bằng cách nhân chập I với các Sobel kernel Kx và Ky, tương ứng:

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

Sobel Filter cho cả hai chiều ngang và dọc. Sau đó, độ lớn G và góc θ của gradient được tính như sau:

$$|G| = \sqrt{I_x^2 + I_y^2},$$
$$\theta(x,y) = arctan\left(\frac{I_y}{I_x}\right)$$

Cường độ và hướng của gradient

- Áp dụng Non-maximum suppression

Tốt nhất, hình ảnh cuối cùng nên có các cạnh mỏng. Vì vậy, ta phải thực hiện thuật toán Non-maximum suppression để làm mỏng chúng. Nguyên tắc rất đơn giản: thuật toán đi qua tất cả các điểm trên ma trận cường độ gradient và tìm các pixel có giá trị lớn nhất theo các hướng cạnh.

Mỗi pixel được đánh giá bằng 2 tiêu chí chính (hướng cạnh tính bằng radian và cường độ pixel (từ 0–255). Dựa trên các đầu vào này, các bước non-maximum suppression là:

+ Tạo một ma trận được khởi tạo bằng 0 có cùng kích thước của ma trận cường độ gradient ban đầu

+ Xác định hướng của gradient dựa trên giá trị góc từ ma trận góc

+ Kiểm tra xem pixel ở cùng một hướng có cường độ cao hơn pixel hiện đang được xử lý hay không

+ Trả lại hình ảnh được xử lý bằng thuật toán non-maximum suppression.

- Ngưỡng kép (Double threshold)

Bước này nhằm mục đích xác định 3 loại pixel: mạnh, yếu và ngoại lai:

+ Pixel mạnh là pixel có cường độ cao và trực tiếp góp phần vào sự hình thành cạnh.

+ Pixel yếu là pixel có giá trị cường độ không đủ để được coi là mạnh nhưng chưa đủ nhỏ để được coi là ngoại lai.

+ Các pixel khác được coi là ngoại lai.

Bây giờ bạn có thể thấy vai trò của ngưỡng kép:

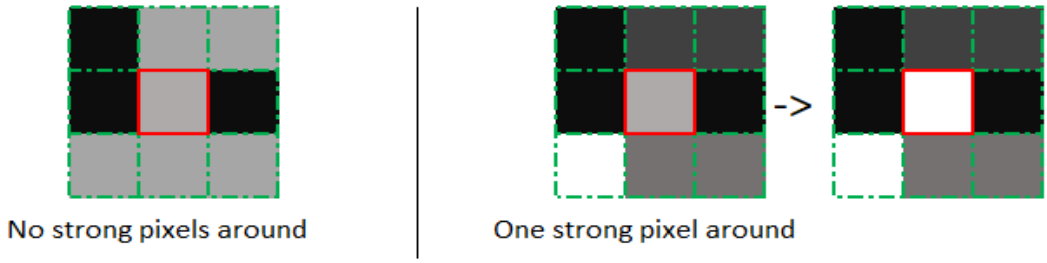
+ Ngưỡng cao được sử dụng để xác định các pixel mạnh (cường độ cao hơn ngưỡng cao)

+ Ngưỡng thấp được sử dụng để xác định các pixel ngoại lai (cường độ thấp hơn ngưỡng thấp)

+ Tất cả các điểm ảnh có cường độ giữa cả hai ngưỡng đều được gắn là yếu. Cơ chế Độ trễ (bước tiếp theo) sẽ giúp ta xác định xem các Pixel yếu sẽ được giữ lại như các pixel mạnh hay bị loại bỏ như pixel ngoại lai.

- Theo dõi cạnh bằng độ trễ (Edge Tracking by Hysteresis)

Dựa trên kết quả sau khi áp dụng ngưỡng, Hysteresis Tracking chuyển đổi các pixel yếu thành mạnh, khi và chỉ khi ít nhất một trong các pixel xung quanh pixel đang xét là pixel mạnh, các pixel yếu còn lại bị loại bỏ, như được mô tả bên dưới:



Hình 12: Kết quả của quá trình Edge Tracking by Hysteresis

1.7 findContours()

Contour được hiểu đơn giản là một đường cong liên kết toàn bộ các điểm liên tục (dọc theo đường biên) mà có cùng màu sắc hoặc giá trị cường độ. Contour rất hữu ích trong phân tích hình dạng, phát hiện vật thể và nhận diện vật thể. Một số lưu ý khi sử dụng contour.

Để độ chính xác cao hơn thì nên sử dụng hình ảnh nhị phân (chỉ gồm 2 màu đen và trắng). Do đó trước khi phát hiện contour thì nên áp dụng threshold hoặc thuật toán canny để chuyển sang ảnh nhị phân.

Hàm **findContour** và **drawContour** sẽ thay đổi hình ảnh gốc. Do đó nếu bạn muốn hình ảnh gốc sau khi tìm được contour, hãy lưu nó vào một biến khác.

Trong openCV, tìm các contours như là tìm các vật thể màu trắng từ nền màu đen. Do đó hãy nhớ rằng, object cần tìm nên là màu trắng và background nên là màu đen.

contours, _ = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

Có 3 tham số trong hàm cv2.findContours():

+ Ảnh gốc
+ Phương pháp trích xuất contours
+ Phương pháp xấp xỉ contour.

Kết quả trả ra là hình ảnh và contours. Trong đó contours là một list của toàn bộ các contours xác định trong hình ảnh. Mỗi một contour là một numpy array của các tọa độ của các điểm biên trong object.

1.8 boundingRect()

Từ contour, ta sẽ xác định tọa độ góc trên bên trái và độ dài cạnh width, height của contour thông qua hàm cv2.boundingRect(). Từ đó vẽ bounding box hình chữ nhật đứng lên hình ảnh gốc bằng cách sử dụng hàm cv.rectangle() như bên dưới:

```
cnt = contours[area_sort[1]]
x,y,w,h = cv2.boundingRect(cnt)
print('centroid: ({}, {}), (width, height): ({}, {})'.format(x, y, w, h))
img = cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
```

CHƯƠNG 2: PHƯƠNG PHÁP THỰC HIỆN VÀ KẾT QUẢ

2.1Các bước xử lý và quá trình thực hiện

Trước khi xử lý chúng ta cần import 2 thư viện:

```
import cv2
import numpy as np
```

cv2: Thư viện OpenCV cho xử lý hình ảnh

numpy: Thư viện NumPy cho các phép toán số học

2.1.1 Các bước xử lý bài tập 1

a) Câu a:

Bài toán này dường như liên quan đến việc xử lý hình ảnh và tách riêng các màu khác nhau trong ảnh. Dưới đây là các bước chính trong mã nguồn:

- **Định nghĩa hàm loại bỏ nhiễu (remove_noise)**

Sử dụng phép biến đổi hình thái để mở và đóng (morphological open và close) để loại bỏ nhiễu từ ảnh.

```
def remove_noise(image_result):
    kernel = np.ones((5, 5), np.uint8)
    opening = cv2.morphologyEx(image_result, cv2.MORPH_OPEN, kernel)
    closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)
    return closing
```

- **Định nghĩa hàm kết hợp các masks màu đỏ (combine_red_masks)**

Tạo mask cho màu đỏ bằng cách kết hợp hai khoảng giới hạn màu đỏ (lower_red1 đến upper_red1 và lower_red2 đến upper_red2) thông qua toán tử bitwise_or.

```
def combine_red_masks(hsv, lower_red1, upper_red1, lower_red2, upper_red2):
    mask_red1 = cv2.inRange(hsv, lower_red1, upper_red1)
    mask_red2 = cv2.inRange(hsv, lower_red2, upper_red2)
    mask_red = cv2.bitwise_or(mask_red1, mask_red2)
    return mask_red
```

- **Định nghĩa hàm xử lý màu (process_color)**

Chuyển đổi không gian màu BGR sang HSV.

Tùy thuộc vào tên màu, tạo mask cho màu tương ứng.

Áp dụng mask lên ảnh gốc bằng phép bitwise_and.

Loại bỏ nhiễu từ ảnh kết quả và lưu lại ảnh.

```
def process_color(image, lower_bound, upper_bound, color_name):
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    if color_name == 'red':
        mask_color = combine_red_masks(hsv, lower_bound[0], upper_bound[0], lower_bound[1], upper_bound[1])
    else:
        mask_color = cv2.inRange(hsv, lower_bound, upper_bound)
    color_result = cv2.bitwise_and(image, image, mask=mask_color)
    processed_image = remove_noise(color_result)
    cv2.imwrite(f'{color_name}_stars.jpg', processed_image)
```

b) Định nghĩa hàm áp dụng ngưỡng (apply_threshold)

- Chuyển đổi ảnh sang ảnh thang độ xám (Grayscale): sử dụng hàm cv2.cvtColor. Điều này giúp làm việc với ảnh một cách dễ dàng trong không gian màu xám.
- Áp dụng ngưỡng để tách biệt sao và nền với giá trị ngưỡng là 220 để tạo ra ảnh ngưỡng từ ảnh thang độ xám bằng hàm cv2.threshold. Các pixel có giá trị lớn hơn ngưỡng được giữ lại và được đặt thành 255, trong khi các pixel có giá trị nhỏ hơn ngưỡng được đặt thành 0. Đặc điểm cv2.THRESH_TOZERO_INV được sử dụng để đảo ngược giá trị ngưỡng, đặt các pixel vượt qua ngưỡng thành 0.
- Loại bỏ nhiễu trên ảnh ngưỡng: remove_noise để loại bỏ nhiễu từ ảnh ngưỡng.
- Lưu ảnh kết quả cuối cùng: "Output_1B.jpg".

```
def apply_threshold(image, threshold_value):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    _, threshold_img = cv2.threshold(gray, threshold_value, 255, cv2.THRESH_TOZERO_INV)
    return remove_noise(threshold_img)
```

c) Định nghĩa hàm áp dụng ngưỡng kép (apply_dual_threshold)

- Áp dụng ngưỡng để tách biệt sao và nền: tiếp tục xử lý ảnh đã được ngưỡng trong bước trước. Chúng ta sử dụng ngưỡng hai lần liên tiếp để tạo ra ảnh tách biệt sao và nền.
- Đầu tiên, chúng ta sử dụng hàm cv2.threshold để áp dụng ngưỡng lên ảnh ngưỡng ban đầu với ngưỡng là 175 và đặc điểm cv2.THRESH_TOZERO.
- Điều này giúp chuyển các pixel có giá trị dưới ngưỡng thành 0, trong khi giữ lại các pixel có giá trị cao hơn ngưỡng.
- Tiếp theo, chúng ta lại áp dụng ngưỡng lần thứ hai lên ảnh đã xử lý trước đó, với ngưỡng là 185 và đặc điểm cv2.THRESH_TOZERO_INV. Điều này đảo ngược giá trị ngưỡng, chuyển các pixel có giá trị cao hơn ngưỡng thành 0.
- Loại bỏ nhiễu trên ảnh tách biệt sao: hàm remove_noise để loại bỏ nhiễu từ ảnh tách biệt sao.
- Lưu ảnh kết quả cuối cùng: "Output_1C.jpg".

```
def apply_dual_threshold(image, lower_threshold, upper_threshold):
    _, thresholded = cv2.threshold(image, lower_threshold, 255, cv2.THRESH_TOZERO)
    _, thresholded_inv = cv2.threshold(thresholded, upper_threshold, 255, cv2.THRESH_TOZERO_INV)
    return remove_noise(thresholded_inv)
```

Hàm main:

- Load ảnh từ file.
- Định nghĩa các khoảng màu cho các màu khác nhau.
- Với mỗi màu, gọi hàm process_color để xử lý và lưu ảnh kết quả.
- Áp dụng ngưỡng cho ảnh xám và lưu ảnh.
- Áp dụng ngưỡng kép để tách sao từ nền cho màu đỏ và lưu ảnh.

2.1.2 Các bước xử lý bài tập 2

- Đọc ảnh đầu vào: "input2.png" bằng cách sử dụng OpenCV (cv2)
- Chuyển đổi ảnh thành thang độ xám (Grayscale): cv2.cvtColor, chúng ta chuyển đổi ảnh màu gốc thành ảnh thang độ xám.
- Tạo ảnh nhị phân (Binary Image) bằng ngưỡng hóa

```
binary_image = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY_INV)[1]
```

- Chia ảnh thành 4 phần:

Hàm chia_anh_lam_bon chia ảnh thành 4 phần bằng cách cắt ảnh thành 2 phần theo chiều dọc và sau đó cắt từng phần thành 2 theo chiều ngang. Các phần này được gán cho các biến top_left, top_right, bottom_left, và bottom_right.

```
def chia_anh_lam_bon(img):
    height, width, _ = img.shape
    chia2partdoc = height // 2
    chia2partngang = width // 2
    top_left = img[0:chia2partdoc, 0:chia2partngang]
    top_right = img[0:chia2partdoc, chia2partngang:width]
    bottom_left = img[chia2partdoc:height, 0:chia2partngang]
    bottom_right = img[chia2partdoc:height, chia2partngang:width]
    return top_left, top_right, bottom_left, bottom_right
```

- Erosion và Dilation cho từng phần của ảnh:

```
def erosion_and_dilation(img,erosion_kernel, dilation_kernel):
    kernel1 = cv2.getStructuringElement(cv2.MORPH_RECT, erosion_kernel)
    kernel2 = cv2.getStructuringElement(cv2.MORPH_RECT, dilation_kernel)
    erosion_image = cv2.erode(img, kernel1, iterations = 1)
    dilation_image = cv2.dilate(erosion_image, kernel2, iterations = 1)
    return dilation_image
```

```
img_tl = erosion_and_dilation(top_left, (3, 1), (1, 4))
img_tr = erosion_and_dilation(top_right, (3, 1), (1, 4))
img_bl = erosion_and_dilation(bottom_left, (3, 1), (3, 3))
```

Chỉ xử lý cho 3 góc đầu thôi vì góc phải bottom_right bị nhiễu quá nhiều nên phải xử lý riêng phần bottom_right.

Hàm process_bottom_right_part được gọi để xử lý đặc biệt cho phần bottom-right. Trong trường hợp ảnh này, thực hiện erosion và dilation cho các phần khác nhau của bottom-right, sau đó ghép chúng lại.

Chia tiếp bottom_right thành 2 mảng là 1/3 mảng đầu và 2/3 mảng cuối theo height của nó.

Khi chia xong thì tiếp tục thấy ảnh width của 2/3 mảng phía dưới khác nhau nên chia tiếp thành 1/2 và 1/2 theo. Sau đó xử lý từng bức ảnh một theo đúng kernel của nó.

```
def process_bottom_right_part(part):
    height_part, width_part, _ = part.shape
    chia_nguyen = height_part // 3

    part1 = part[:chia_nguyen, :]
    part2 = part[chia_nguyen:, :]
    eroi_part1 = erosion_and_dilation(part1, (4, 4), (3, 3))

    height_p2, width_p2, _ = part2.shape
    phan_mo = width_p2 // 2
    phanmonhat = erosion_and_dilation(part2[:, :phan_mo], (6, 6), (2, 6))
    anhcuoicung = erosion_and_dilation(part2[:, phan_mo:], (4, 4), (2, 5))

    part2_after = part2
    part2_after[:, :phan_mo] = phanmonhat
    part2_after[:, phan_mo:] = anhcuoicung

    return np.vstack((eroi_part1, part2_after))
```

- Cuối cùng, ảnh được tái tạo bằng cách ghép lại các phần đã xử lý vào vị trí tương ứng trong ảnh gốc (binary_image theo chiều mà nó tách ra.

```
bottom_right_processed = process_bottom_right_part(bottom_right)

height, width, _ = binary_image.shape
chia2partdoc = height // 2
chia2partngang = width // 2
binary_image[0:chia2partdoc, 0:chia2partngang] = img_tl
binary_image[0:chia2partdoc, chia2partngang:width] = img_tr
binary_image[chia2partdoc:height, 0:chia2partngang] = img_bl
binary_image[chia2partdoc:height, chia2partngang:width] = bottom_right_processed
```

- Gaussian Blur và Canny Edge Detection

Ảnh nhị phân sau khi đã được xử lý được làm mờ bằng Gaussian Blur với kernel kích thước (5, 5), sau đó thực hiện Canny Edge Detection với ngưỡng dưới là 30 và ngưỡng trên là 150.

```
blurred = cv2.GaussianBlur(binary_image, (5, 5), 0)

edged = cv2.Canny(blurred, 30, 150)
```

- Sử dụng cv2.findContours để tìm contours trên ảnh Canny Edge Detection.

```
contours, _ = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

- Vẽ hình chữ nhật bao quanh các đối tượng

x, y, w, h = cv2.boundingRect(contour): Hàm này trả về tọa độ và kích thước của hình chữ nhật bao quanh contour. Cụ thể:

- (x, y): Tọa độ (x, y) của góc trái trên của hình chữ nhật.
- (w, h): Chiều rộng và chiều cao của hình chữ nhật.

1000 < w * h < 5500 and 23 < w < 200 and h < 100: Kiểm tra các điều kiện về kích thước của hình chữ nhật:

1000 < w * h < 5500: Diện tích của hình chữ nhật nằm trong khoảng từ 1000 đến 5500 pixel.

23 < w < 200: Chiều rộng của hình chữ nhật nằm trong khoảng từ 23 đến 200 pixel

h < 100: Chiều cao của hình chữ nhật nhỏ hơn 100 pixel.

cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2): Nếu hình chữ nhật thỏa mãn các điều kiện, vẽ hình chữ nhật lên ảnh gốc (image). Cụ thể:

image: Ảnh gốc.

(x, y): Tọa độ (x, y) của góc trái trên của hình chữ nhật.

(x + w, y + h): Tọa độ (x+w, y+h) của góc phải dưới của hình chữ nhật.

(0, 255, 0): Màu của hình chữ nhật (ở đây là màu xanh lá cây).

2: Độ dày của đường vẽ.

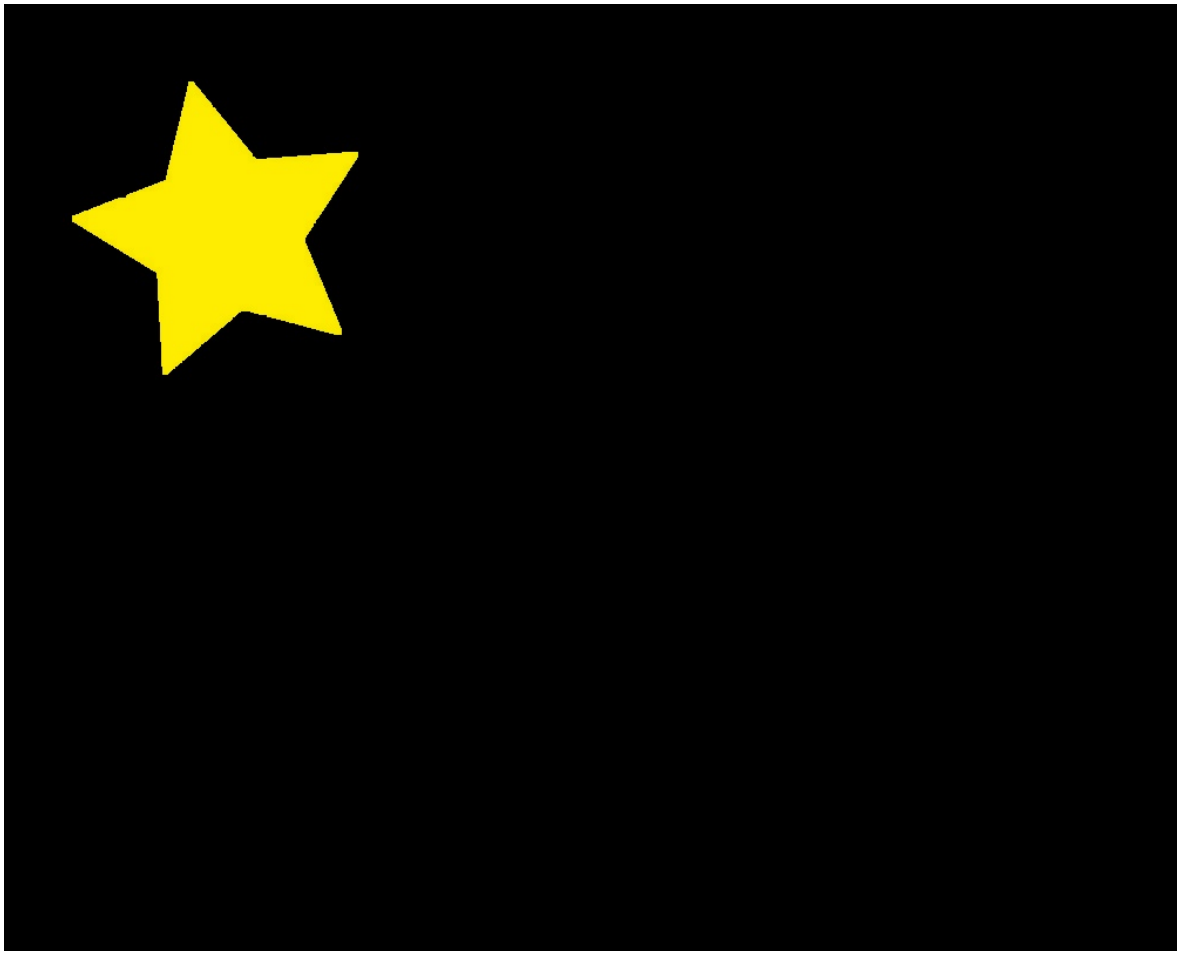
```
for contour in contours:
    x, y, w, h = cv2.boundingRect(contour)
    if 1000 < w * h < 5500 and 23 < w < 200 and h < 100:
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

- Lưu ảnh kết quả cuối cùng: "Ouput2.jpg".

2.2Kết quả thực hiện

2.2.1 Kết quả bài tập 1

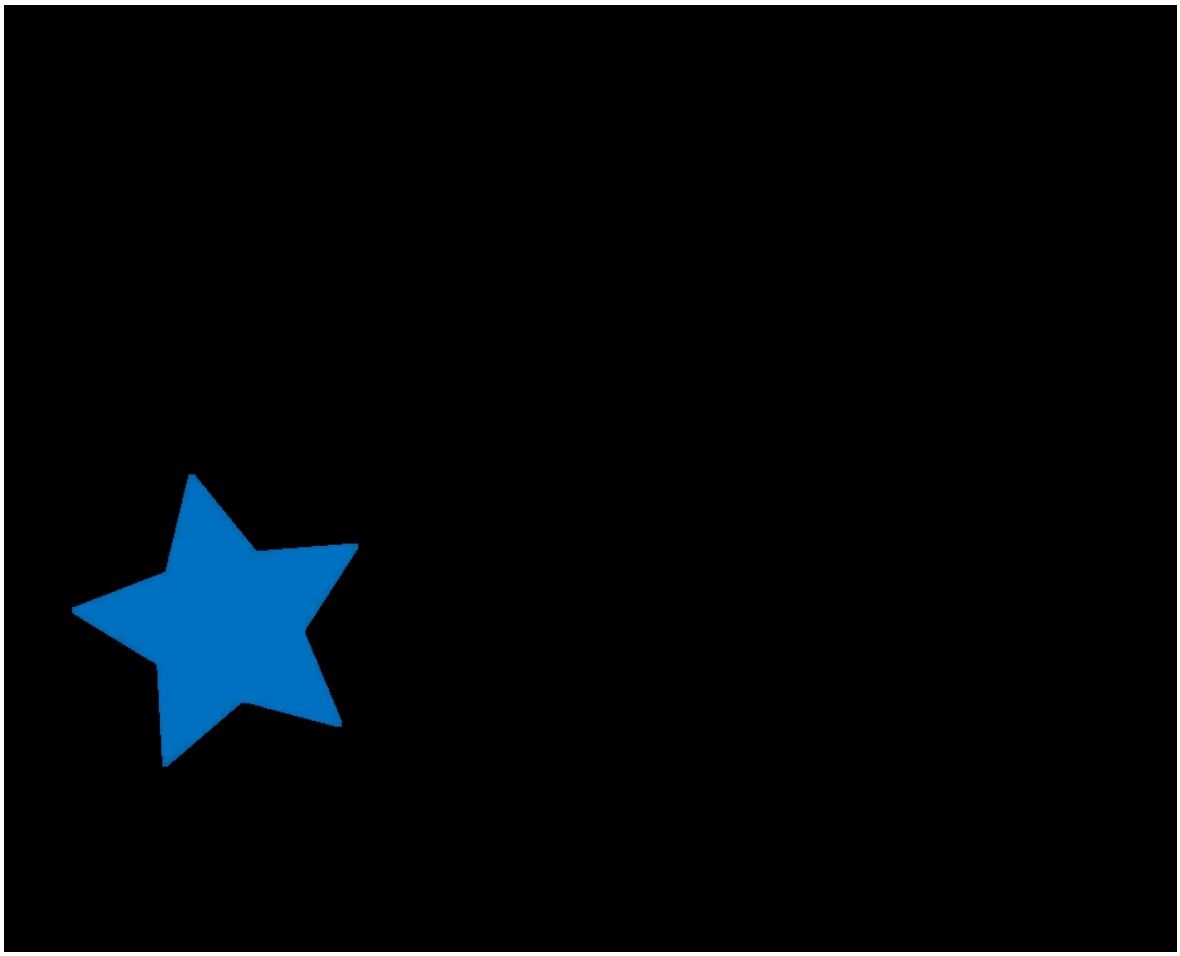
a) Kết quả bài 1a



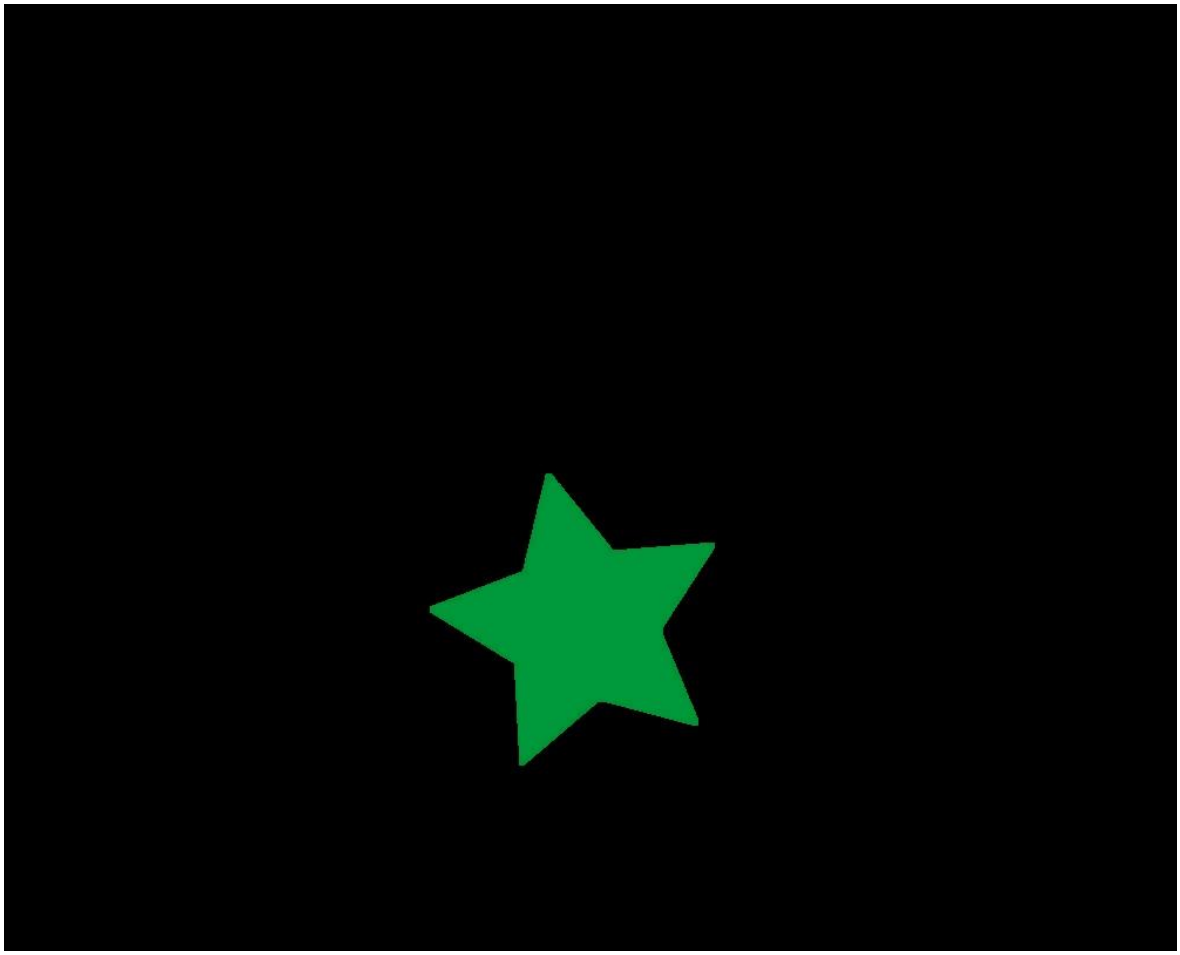
Hình 13: yellow_stars

Hình 14: orange_stars

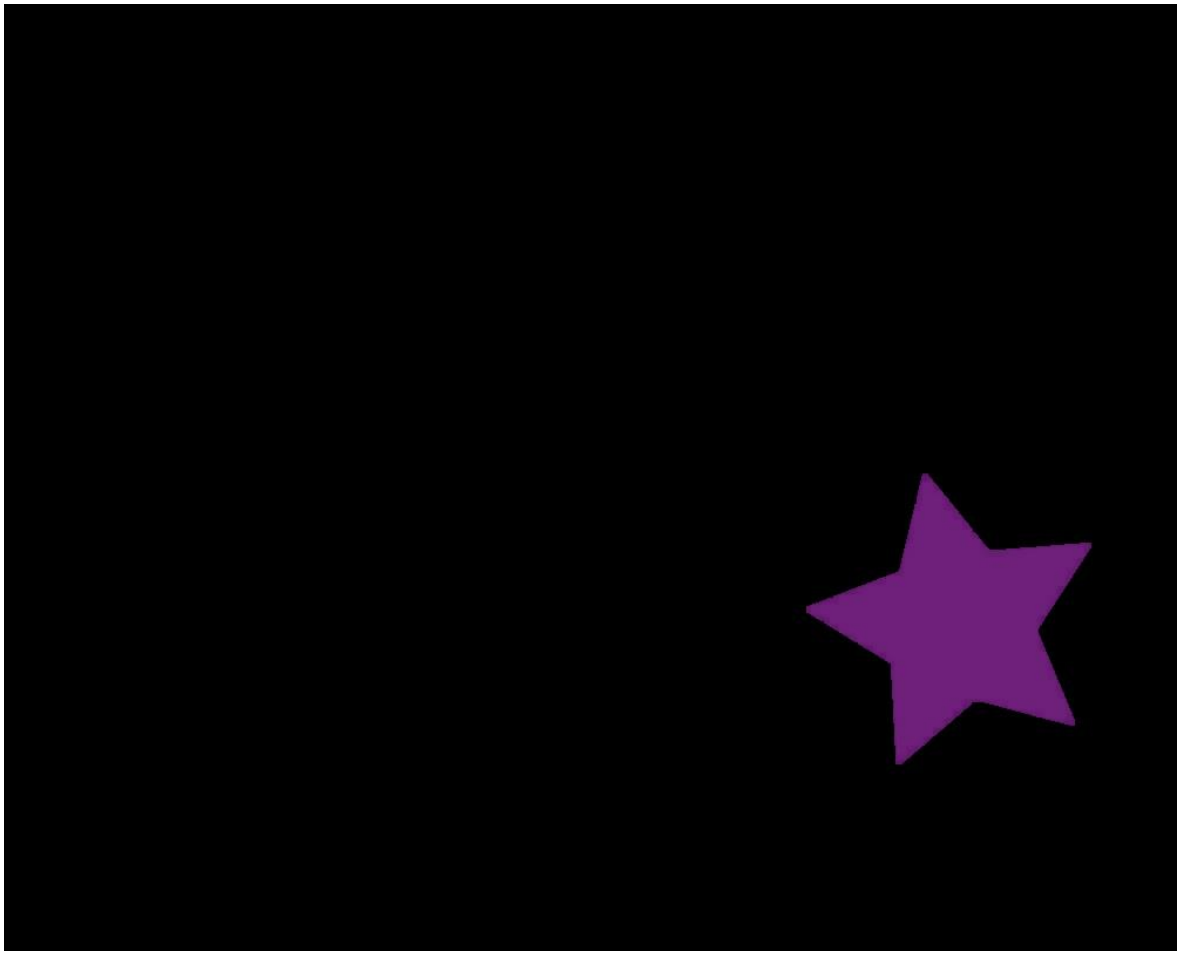
Hình 15: red_stars



Hình 16: blue_stars



Hình 17: green_stars

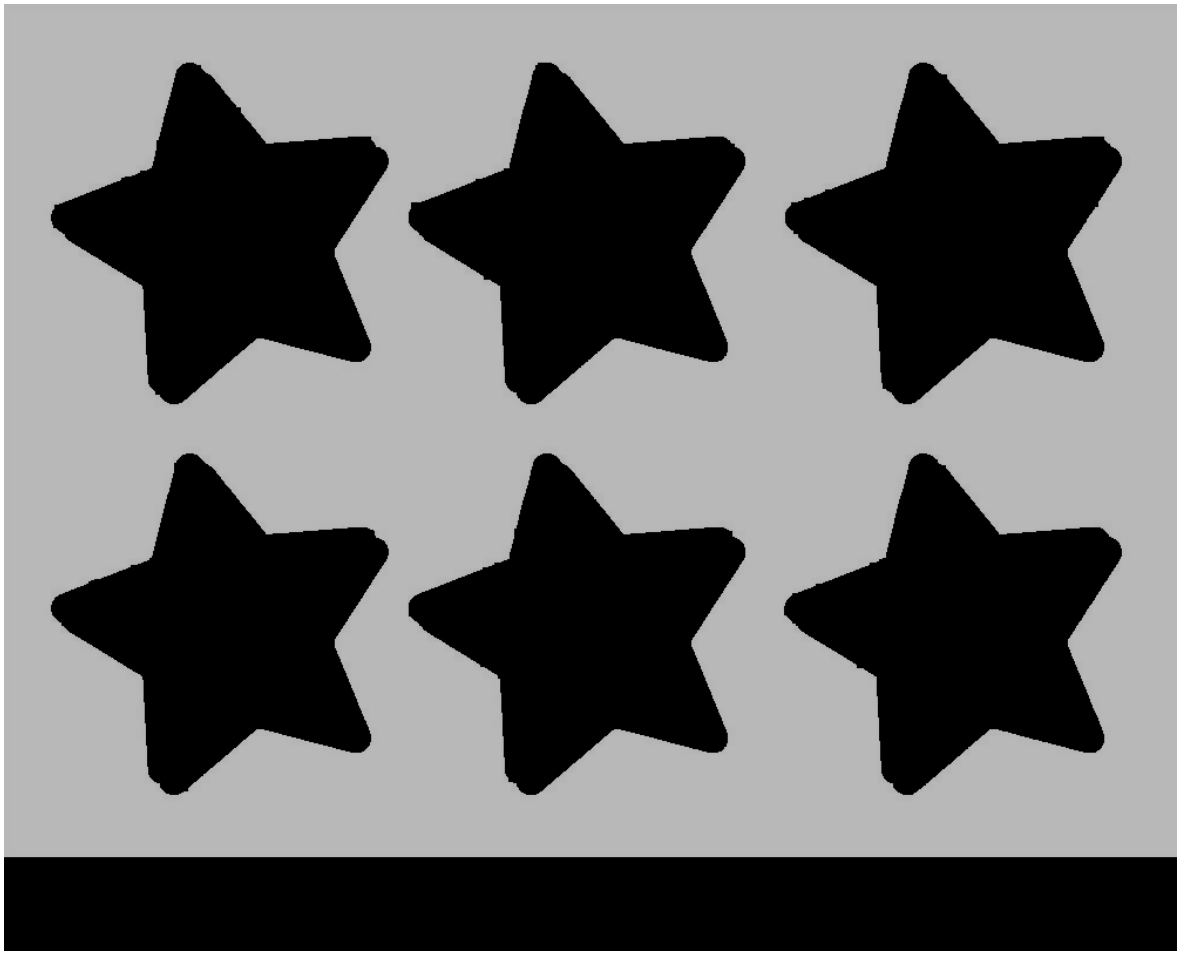


Hình 18: purple_stars

b) Kết quả bài 1b

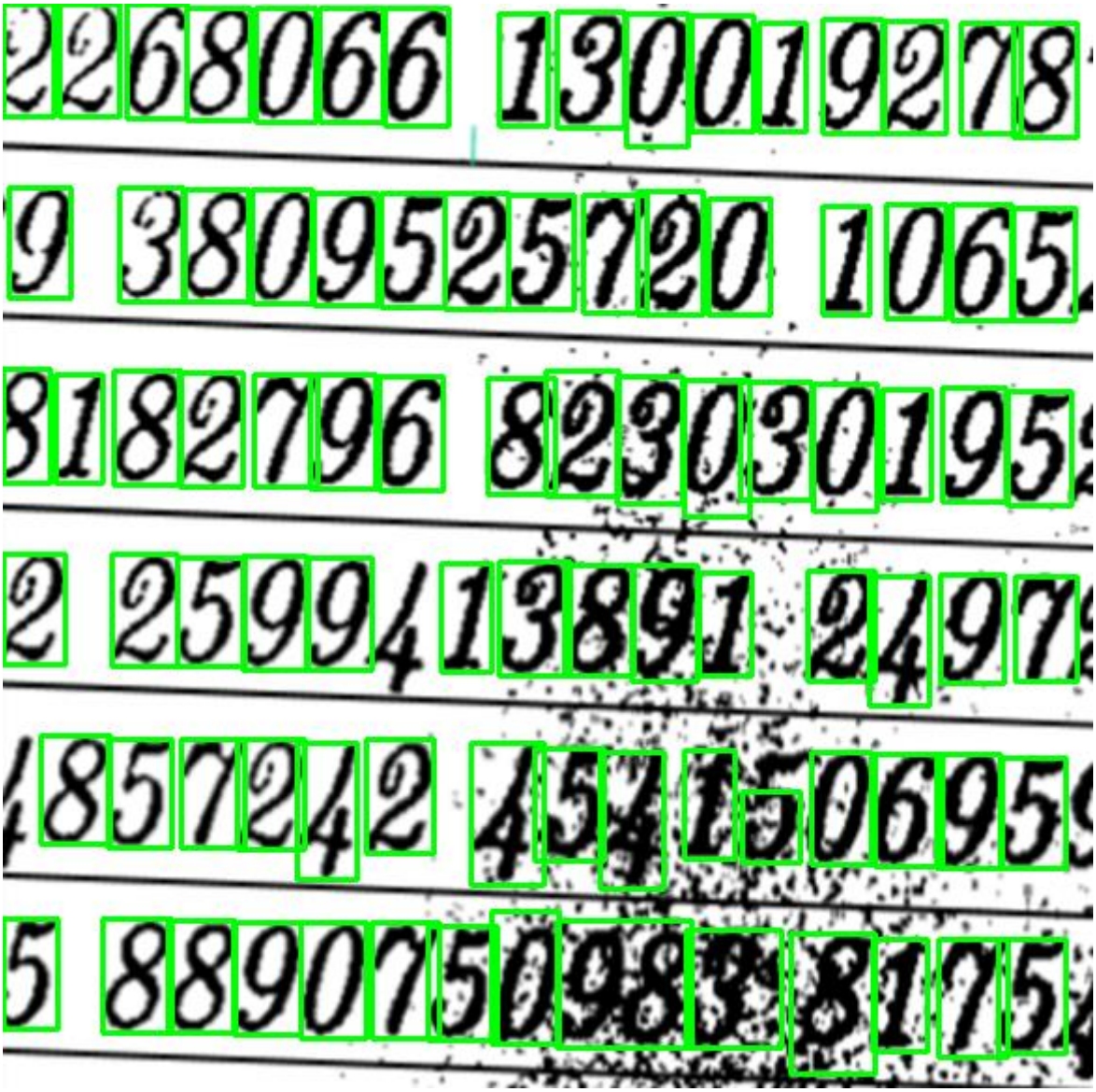
Hình 19: Ảnh đầu ra với tất cả ngôi sao viền đen

c) Kết quả bài 1c



Hình 20: Ảnh đầu ra với tất cả ngôi sao màu đen

2.2.2 Kết quả bài tập 2



Hình 21: Ảnh đầu ra kết quả của bài tập 2

TÀI LIỆU THAM KHẢO

Tiếng Anh

[1] "Digital Image Processing" by Rafael C. Gonzalez and Richard E. Woods, 2007.

[2] "Computer Vision: Algorithms and Applications" by Richard Szeliski, 2010.

[3] "Image Processing, Analysis, and Machine Vision" by Milan Sonka, Vaclav Hlavac, and Roger Boyle, 1999.

[4] "Computer Vision: Principles, Algorithms, Applications, Learning" by Richard E. Harvey, Janice C. Harvey, and Michael A. Pratt, N/A.

[5] "Digital Image Processing Using MATLAB" by Gonzalez, Woods, and Eddins, 2009.