**Chương 1**

# 1.1/1.2

**% Đọc ảnh**

**I = imread('cell.tif');**

**% Hiển thị ảnh**

**figure;**

**imshow(I);**

**title('Ảnh gốc');**

**% Truy xuất giá trị tại vị trí pixel (100, 20)**

**i = 100;**

**j = 20;**

**original\_value = I(i, j); % Lấy giá trị pixel ban đầu**

**% Thực hiện phép toán: cộng và trừ 25**

**I\_add = I; % Tạo bản sao để thực hiện phép cộng**

**I\_sub = I; % Tạo bản sao để thực hiện phép trừ**

**I\_add(i, j) = I\_add(i, j) + 25; % Cộng 25 vào pixel (100, 20)**

**I\_sub(i, j) = I\_sub(i, j) - 25; % Trừ 25 từ pixel (100, 20)**

**% Đảm bảo giá trị pixel nằm trong khoảng [0, 255] cho ảnh 8-bit**

**I\_add = uint8(max(min(I\_add, 255), 0));**

**I\_sub = uint8(max(min(I\_sub, 255), 0));**

**% Hiển thị ảnh sau khi thay đổi pixel**

**figure;**

**subplot(1, 3, 1); imshow(I); title('Ảnh gốc');**

**subplot(1, 3, 2); imshow(I\_add); title('Cộng 25 tại (100, 20)');**

**subplot(1, 3, 3); imshow(I\_sub); title('Trừ 25 tại (100, 20)');**

**% Khởi tạo hai ảnh mới**

**[rows, cols] = size(I); % Lấy kích thước ảnh**

**I1 = zeros(rows, cols, 'uint8'); % Ảnh I1: Cộng 25**

**I2 = zeros(rows, cols, 'uint8'); % Ảnh I2: Trừ 25**

**% Duyệt qua từng pixel của ảnh và thực hiện phép tính**

**for i = 1:rows**

**for j = 1:cols**

**% Cộng 25 và đảm bảo giá trị không vượt quá 255**

**I1(i, j) = min(I(i, j) + 50, 255);**

**% Trừ 25 và đảm bảo giá trị không nhỏ hơn 0**

**I2(i, j) = max(I(i, j) - 50, 0);**

**end**

**end**

**% Hiển thị kết quả**

**figure;**

**subplot(1, 3, 1); imshow(I); title('Ảnh gốc');**

**subplot(1, 3, 2); imshow(I1); title('Ảnh sau khi cộng 25');**

**subplot(1, 3, 3); imshow(I2); title('Ảnh sau khi trừ 25');**

**Chương 1**

# 1.3

**% Đọc ảnh gốc "cameraman.tif"**

**I = imread('cameraman.tif');**

**% Ghi ảnh dưới định dạng JPEG và PNG**

**imwrite(I, 'Ijpg.jpg', 'jpg'); % Lưu ảnh JPEG**

**imwrite(I, 'Ipng.png', 'png'); % Lưu ảnh PNG**

**% Đọc lại hai ảnh đã lưu**

**Ijpg = imread('Ijpg.jpg'); % Đọc ảnh JPEG**

**Ipng = imread('Ipng.png'); % Đọc ảnh PNG**

**% So sánh hai ảnh với ảnh gốc**

**diff\_jpg = I - Ijpg; % Chênh lệch với ảnh JPEG**

**diff\_png = I - Ipng; % Chênh lệch với ảnh PNG**

**% Hiển thị kết quả**

**figure;**

**subplot(2, 3, 1); imshow(I); title('Ảnh gốc (cameraman.tif)');**

**subplot(2, 3, 2); imshow(Ijpg); title('Ảnh JPEG (Ijpg.jpg)');**

**subplot(2, 3, 3); imshow(Ipng); title('Ảnh PNG (Ipng.png)');**

**subplot(2, 3, 4); imshow(I); title('Ảnh gốc (cameraman.tif)');**

**subplot(2, 3, 5); imshow(diff\_jpg, []); title('Chênh lệch với JPEG');**

**subplot(2, 3, 6); imshow(diff\_png, []); title('Chênh lệch với PNG');**

**Phân tích vấn đề và lý do không giống hoàn toàn:**

Nén mất thông tin (Lossy Compression - JPEG): Định dạng JPEG sử dụng phương pháp nén mất thông tin, có nghĩa là khi ảnh được lưu dưới định dạng JPEG, một số dữ liệu ảnh sẽ bị loại bỏ để giảm kích thước file. Các chi tiết không quá quan trọng hoặc các pixel có sự thay đổi nhỏ có thể bị làm mờ hoặc mất đi hoàn toàn. Vì vậy, khi bạn so sánh ảnh gốc và ảnh JPEG, sẽ có sự khác biệt về pixel, dẫn đến chênh lệch không bằng 0.

Nén không mất thông tin (Lossless Compression - PNG): Định dạng PNG sử dụng phương pháp nén không mất thông tin, có nghĩa là ảnh PNG giữ nguyên tất cả dữ liệu của ảnh gốc mà không bị mất thông tin. Tuy nhiên, do các phương pháp nén khác nhau, mặc dù PNG không làm mất dữ liệu, ảnh PNG có thể có sự thay đổi nhỏ về cách lưu trữ và mã hóa ảnh, nhưng sự khác biệt này thường ít hơn so với JPEG.

**Giải thích về việc so sánh ảnh:**

Việc trừ từng pixel của hai ảnh để so sánh trực tiếp là hợp lý trong trường hợp bạn muốn kiểm tra sự khác biệt pixel-by-pixel. Tuy nhiên, vì JPEG có nén mất thông tin, sự khác biệt giữa ảnh gốc và ảnh JPEG sẽ là một số giá trị không bằng 0, do sự thay đổi trong quá trình nén.

Ảnh PNG có thể có sự khác biệt nhỏ khi so sánh, nhưng khác biệt này thường ít hơn so với JPEG vì PNG là định dạng nén không mất thông tin. Tuy nhiên, điều này không có nghĩa là ảnh PNG và ảnh gốc là hoàn toàn giống nhau, mà là sự khác biệt này rất nhỏ và không dễ nhận thấy bằng mắt thường.

**Kết luận:**

Chúng ta không mong đợi các ảnh giống nhau hoàn toàn khi so sánh ảnh gốc với ảnh JPEG và PNG.

Sự khác biệt không phải là lỗi mà là do định dạng nén (lossy vs. lossless) tạo ra sự thay đổi trong cách dữ liệu ảnh được lưu trữ.

Đoạn mã bạn cung cấp sẽ giúp bạn thấy rõ ràng sự khác biệt giữa các định dạng nén, đặc biệt là giữa JPEG (nén mất thông tin) và PNG (nén không mất thông tin).

**Chương 2**

# 2.1/abcd

**% Đọc ảnh vào biến r**

**r = imread('Fig0304(a)(breast\_digital\_Xray).tif');**

**% Kiểm tra nếu ảnh không phải là ảnh xám, chuyển đổi nó**

**if size(r, 3) == 3**

**r = rgb2gray(r); % Chuyển đổi ảnh màu sang ảnh xám**

**end**

**% Cấp mức xám của ảnh (L = 256 cho ảnh 8-bit)**

**L = 256;**

**% Thực hiện biến đổi âm bản: sa = L - 1 - r**

**sa = L - 1 - r;**

**% Tính histogram của ảnh gốc và ảnh âm bản**

**[counts\_r, grayLevels\_r] = imhist(r); % Histogram ảnh gốc**

**[counts\_sa, grayLevels\_sa] = imhist(sa); % Histogram ảnh âm bản**

**% Hiển thị histogram của ảnh gốc và ảnh âm bản**

**figure;**

**subplot(1, 2, 1);**

**bar(grayLevels\_r, counts\_r);**

**xlabel('Mức xám');**

**ylabel('Số lượng pixel');**

**title('Histogram ảnh gốc');**

**subplot(1, 2, 2);**

**bar(grayLevels\_sa, counts\_sa);**

**xlabel('Mức xám');**

**ylabel('Số lượng pixel');**

**title('Histogram ảnh âm bản');**

**% Chọn ngưỡng t = 127**

**t = 127;**

**% Tạo ảnh nhị phân từ ảnh xám**

**B = r >= t; % Mỗi pixel >= ngưỡng t sẽ thành 1 (trắng), còn lại thành 0 (đen)**

**% Chuyển ảnh nhị phân về kiểu uint8 để hiển thị**

**B = uint8(B) \* 255; % 1 trở thành 255, 0 giữ nguyên**

**% Hiển thị ảnh gốc và ảnh nhị phân trên một cửa sổ**

**figure;**

**subplot(2, 2, 1);**

**imshow(r);**

**title('Ảnh xám gốc');**

**subplot(2, 2, 2);**

**imshow(sa);**

**title('Ảnh âm bản');**

**subplot(2, 2, 3);**

**imshow(B);**

**title('Ảnh nhị phân với ngưỡng t = 127');**

**Phân tích sự phân bố mức xám của ảnh:**

**a. Phân tích sự phân bố mức xám của ảnh:**

**Histogram của ảnh gốc:**

Sự phân bố mức xám của ảnh gốc được thể hiện qua histogram đầu tiên. Histogram này cho biết số lượng pixel ở mỗi mức xám từ 0 đến 255. Nếu histogram có một đỉnh cao ở mức xám thấp (0-50) và đỉnh thấp ở các mức xám cao, điều này cho thấy ảnh chủ yếu có các pixel tối, hoặc ngược lại nếu đỉnh nằm ở mức xám cao.

**Histogram của ảnh âm bản:**

Histogram của ảnh âm bản có thể khác biệt rõ rệt so với ảnh gốc. Do ảnh âm bản chuyển các pixel sáng thành tối và ngược lại, histogram của ảnh âm bản sẽ có phân bố ngược lại so với histogram của ảnh gốc.

**Đánh giá phân tích:**

Nếu histogram của ảnh gốc có một đỉnh rõ ràng ở các mức xám trung bình hoặc tối, điều này có thể chỉ ra rằng ảnh có các vùng tối chiếm ưu thế.

Nếu histogram của ảnh gốc là phân bố đều, điều đó có thể chỉ ra rằng ảnh có sự phân bố mức xám khá đều và không có vùng tối hoặc sáng chiếm ưu thế.

Sự phân bố mức xám của ảnh gốc có thể phản ánh độ sáng tổng thể của ảnh, trong khi sự phân bố của ảnh âm bản phản ánh sự tương phản ngược lại.

**Ảnh đã phân tích tốt chưa?**

Ảnh đã được phân tích tốt vì bạn đã tính toán và hiển thị histogram của ảnh gốc và ảnh âm bản, giúp bạn thấy được sự phân bố mức xám của ảnh.

**Đánh giá sự phân bố:**

Cần kiểm tra xem histogram của ảnh gốc có sự phân bố rõ ràng không, liệu ảnh có độ sáng hoặc tối chiếm ưu thế hay không.

Nếu histogram của ảnh gốc là đều, tức là các mức xám phân bố đều từ 0 đến 255, thì phân tích có thể xem là tốt, thể hiện ảnh có sự phân bổ mức xám đồng đều.

**Tóm lại:**

Mã của bạn thực hiện đúng yêu cầu và giúp phân tích sự phân bố mức xám của ảnh. Việc phân tích mức xám qua histogram là một phương pháp chính xác để hiểu về độ sáng, độ tương phản và đặc điểm của ảnh.

**Chương 2**

# 2.1/abcd Không dùng hàm có sẵn

**% Đọc ảnh vào biến r**

**r = imread('Fig0304(a)(breast\_digital\_Xray).tif');**

**%Câu 2.1ab=============================================**

**% Cấp mức xám của ảnh (L = 256 cho ảnh 8-bit)**

**L = 256;**

**% Thực hiện biến đổi âm bản: sa = L - 1 - r**

**sa = L - 1 - r;**

**% Tính histogram của ảnh gốc và ảnh âm bản**

**[counts\_r, grayLevels\_r] = imhist(r); % Histogram ảnh gốc**

**[counts\_sa, grayLevels\_sa] = imhist(sa); % Histogram ảnh âm bản**

**% Hiển thị histogram của ảnh gốc và ảnh âm bản**

**figure;**

**subplot(1, 2, 1);**

**bar(grayLevels\_r, counts\_r);**

**xlabel('Mức xám');**

**ylabel('Số lượng pixel');**

**title('Histogram ảnh gốc');**

**subplot(1, 2, 2);**

**bar(grayLevels\_sa, counts\_sa);**

**xlabel('Mức xám');**

**ylabel('Số lượng pixel');**

**title('Histogram ảnh âm bản');**

**%Câu 2.1cd==============================================**

**% Chọn ngưỡng t = 127**

**t = 127;**

**% Tạo ảnh nhị phân từ ảnh xám**

**B = r >= t; % Mỗi pixel >= ngưỡng t sẽ thành 1 (trắng), còn lại thành 0 (đen)**

**% Chuyển ảnh nhị phân về kiểu uint8 để hiển thị**

**% Duyệt từng pixel để tạo ảnh nhị phân**

**for i = 1:size(r, 1) % Duyệt qua từng hàng**

**for j = 1:size(r, 2) % Duyệt qua từng cột**

**if r(i, j) >= t**

**B(i, j) = 255; % Pixel >= ngưỡng t chuyển thành 255 (trắng)**

**else**

**B(i, j) = 0; % Pixel < ngưỡng t chuyển thành 0 (đen)**

**end**

**end**

**end**

**% Hiển thị ảnh gốc và ảnh nhị phân trên một cửa sổ**

**figure;**

**subplot(2, 2, 1);**

**imshow(r);**

**title('Ảnh xám gốc');**

**subplot(2, 2, 2);**

**imshow(sa);**

**title('Ảnh âm bản');**

**subplot(2, 2, 3);**

**imshow(B);**

**title('Ảnh nhị phân với ngưỡng t = 127');**

**Chương 2**

# 2.2/abc

**% Đọc ảnh vào biến I**

**I = imread('Fig0122(a)(fractal-iris).tif');**

**%2.2a=================================================**

**% Tạo ảnh i3 từ bit plane thứ 3 của ảnh I**

**bit\_plane\_3 = 3;**

**I3 = bitget(I, bit\_plane\_3);**

**%2.2b=================================================**

**% Tạo ảnh i6 từ bit plane thứ 6 của ảnh I**

**bit\_plane\_6 = 6;**

**I6 = bitget(I, bit\_plane\_6);**

**% Truy xuất bit plane thứ 7 và thứ 8**

**bit\_plane\_7 = bitget(I, 7); % Lấy bit thứ 7**

**bit\_plane\_8 = bitget(I, 8); % Lấy bit thứ 8**

**% Tạo ảnh i78 bằng cách kết hợp bit thứ 7 và 8**

**i78 = bitset(zeros(size(I)), 7, bit\_plane\_7); % Đặt bit thứ 7 của ảnh mới**

**i78 = bitset(i78, 8, bit\_plane\_8); % Đặt bit thứ 8 của ảnh mới**

**% Hiển thị ảnh gốc và ảnh từ bit plane thứ 3**

**figure;**

**subplot(2, 2, 1); imshow(I);**

**title('Ảnh gốc');**

**subplot(2, 2, 2); imshow(I3, []); % Hiển thị ảnh i3 (bit plane thứ 3)**

**title('Bit plane thứ 3');**

**% Hiển thị ảnh từ bit plane thứ 6**

**subplot(2, 2, 3); imshow(I6, []); % Hiển thị ảnh i6 (bit plane thứ 6)**

**title('Bit plane thứ 6');**

**% Hiển thị ảnh từ bit plane thứ 7 và 8**

**subplot(2, 2, 4); imshow(i78, []); % Hiển thị ảnh i78 (bit plane thứ 7 và 8)**

**title('Ảnh i78 (Bit plane thứ 7 và 8)');**

**Chương 2**

# 2.2/abc Không dùng hàm có sẵn

**% Đọc ảnh vào biến I**

**I = imread('Fig0122(a)(fractal-iris).tif');**

**% Kích thước ảnh**

**[rows, cols] = size(I);**

**% 2.2a: Tạo ảnh i3 từ bit plane thứ 3 của ảnh I**

**bit\_plane\_3 = 3; % Bit thứ 3**

**I3 = zeros(rows, cols); % Khởi tạo ma trận ảnh I3**

**for i = 1:rows**

**for j = 1:cols**

**% Lấy giá trị bit thứ 3 của mỗi pixel**

**I3(i, j) = bitand(I(i, j), 2^(bit\_plane\_3 - 1)) > 0;**

**end**

**end**

**% 2.2b: Tạo ảnh i6 từ bit plane thứ 6 của ảnh I**

**bit\_plane\_6 = 6; % Bit thứ 6**

**I6 = zeros(rows, cols); % Khởi tạo ma trận ảnh I6**

**for i = 1:rows**

**for j = 1:cols**

**% Lấy giá trị bit thứ 6 của mỗi pixel**

**I6(i, j) = bitand(I(i, j), 2^(bit\_plane\_6 - 1)) > 0;**

**end**

**end**

**% Truy xuất bit plane thứ 7 và thứ 8**

**bit\_plane\_7 = 7;**

**bit\_plane\_8 = 8;**

**bit7 = zeros(rows, cols); % Ảnh chứa bit thứ 7**

**bit8 = zeros(rows, cols); % Ảnh chứa bit thứ 8**

**for i = 1:rows**

**for j = 1:cols**

**% Lấy giá trị bit thứ 7 và 8**

**bit7(i, j) = bitand(I(i, j), 2^(bit\_plane\_7 - 1)) > 0;**

**bit8(i, j) = bitand(I(i, j), 2^(bit\_plane\_8 - 1)) > 0;**

**end**

**end**

**% Tạo ảnh i78 bằng cách kết hợp bit thứ 7 và 8**

**i78 = zeros(rows, cols, 'uint8'); % Khởi tạo ảnh i78**

**for i = 1:rows**

**for j = 1:cols**

**% Đặt bit thứ 7 và thứ 8**

**i78(i, j) = bit7(i, j) \* 2^(bit\_plane\_7 - 1) + bit8(i, j) \* 2^(bit\_plane\_8 - 1);**

**end**

**end**

**% Hiển thị ảnh**

**figure;**

**subplot(2, 2, 1); imshow(I);**

**title('Ảnh gốc');**

**subplot(2, 2, 2); imshow(I3, []);**

**title('Bit plane thứ 3');**

**subplot(2, 2, 3); imshow(I6, []);**

**title('Bit plane thứ 6');**

**subplot(2, 2, 4); imshow(i78, []);**

**title('Ảnh i78 (Bit plane thứ 7 và 8)');**

**Chương 2**

# 2.3/abc

**% Đọc ảnh**

**I = imread('Fig0316(4)(bottom\_left).tif');**

**% Hiển thị ảnh gốc**

**figure;**

**imshow(I);**

**title('Ảnh gốc');**

**% Tính toán histogram (lược đồ xám)**

**[counts, grayLevels] = imhist(I);**

**% Hiển thị histogram**

**figure;**

**bar(grayLevels, counts);**

**xlabel('Mức xám');**

**ylabel('Số lượng pixel');**

**title('Lược đồ xám của ảnh gốc');**

**% Cân bằng lược đồ xám của ảnh**

**I\_eq = histeq(I);**

**% Hiển thị ảnh gốc và ảnh đã cân bằng**

**figure;**

**subplot(1, 2, 1);**

**imshow(I);**

**title('Ảnh gốc');**

**subplot(1, 2, 2);**

**imshow(I\_eq);**

**title('Ảnh sau khi cân bằng mức xám');**

**% Hiển thị histogram của ảnh gốc và ảnh đã cân bằng**

**figure;**

**subplot(1, 2, 1);**

**imhist(I);**

**title('Histogram của ảnh gốc');**

**subplot(1, 2, 2);**

**imhist(I\_eq);**

**title('Histogram của ảnh sau khi cân bằng');**

**Chương 2**

# 2.3/abc Không dùng hàm có sẵn

**% Đọc ảnh xám**

**i = imread('Fig0316(4)(bottom\_left).tif');**

**% Kiểm tra nếu ảnh là RGB, chuyển sang ảnh xám**

**if size(i, 3) == 3**

**i = rgb2gray(i);**

**end**

**% Kích thước ảnh**

**[N, M] = size(i); %N số dòng, M số cột**

**L = 256; % Số mức xám**

**% Bước 1: Tính histogram gốc**

**hist\_orig = zeros(1, L); %Số lần xuất hiện của 1 mức xám cụ thể**

**for x = 1:N**

**for y = 1:M**

**hist\_orig(i(x, y) + 1) = hist\_orig(i(x, y) + 1) + 1;**

**end**

**end**

**% Bước 2: Tính hàm tích lũy xác suất (CDF)**

**cdf = zeros(1, L); %Tổng mức xám tích lũy từ 0 đến k**

**cdf(1) = hist\_orig(1);**

**for k = 2:L**

**cdf(k) = cdf(k-1) + hist\_orig(k);**

**end**

**cdf = cdf / (N \* M); % Chuẩn hóa CDF, sau khi chuẩn hóa ta có tỷ lệ xuất hiện của 1 mức xám cụ thể**

**% Bước 3: Ánh xạ mức xám mới**

**s = round(cdf \* (L - 1)); % Ánh xạ CDF vào khoảng 0–255, Hàm round được sử dụng để làm tròn kết quả về số nguyên**

**% Bước 4: Tạo ảnh sau cân bằng**

**i\_equalized = zeros(N, M, 'uint8');**

**for x = 1:N**

**for y = 1:M**

**i\_equalized(x, y) = s(i(x, y) + 1);**

**end**

**end**

**% Tính histogram sau cân bằng**

**hist\_equalized = zeros(1, L);**

**for x = 1:N**

**for y = 1:M**

**hist\_equalized(i\_equalized(x, y) + 1) = hist\_equalized(i\_equalized(x, y) + 1) + 1;**

**end**

**end**

**% Hiển thị kết quả**

**figure;**

**subplot(2, 2, 1); imshow(i); title('Ảnh gốc');**

**subplot(2, 2, 2); bar(0:L-1, hist\_orig); title('Histogram gốc');**

**subplot(2, 2, 3); imshow(i\_equalized); title('Ảnh sau cân bằng');**

**subplot(2, 2, 4); bar(0:L-1, hist\_equalized); title('Histogram sau cân bằng');**

**Chương 2**

# 2.7

**% Đọc các ảnh đầu vào**

**pillsetc = imread('pillsetc.png');**

**tape = imread('tape.png');**

**coins = imread('coins.png');**

**eight = imread('eight.tif');**

**% Chuyển đổi ảnh sang ảnh nhị phân với ngưỡng cố định**

**pillsetc\_1 = im2bw(pillsetc, 0.1); % Tách ngưỡng tự động với ngưỡng 0.1**

**tape\_1 = im2bw(tape, 0.1); % Tách ngưỡng tự động với ngưỡng 0.1**

**coins\_1 = im2bw(coins, 0.1); % Tách ngưỡng tự động với ngưỡng 0.1**

**eight\_1 = im2bw(eight, 0.1); % Tách ngưỡng tự động với ngưỡng 0.1**

**% Hiển thị các ảnh gốc và ảnh đã tách ngưỡng**

**figure;**

**subplot(4, 2, 1), imshow(pillsetc);**

**title('Ảnh gốc: pillsetc');**

**subplot(4, 2, 2), imshow(pillsetc\_1);**

**title('Ảnh nhị phân: pillsetc');**

**subplot(4, 2, 3), imshow(tape);**

**title('Ảnh gốc: tape');**

**subplot(4, 2, 4), imshow(tape\_1);**

**title('Ảnh nhị phân: tape');**

**subplot(4, 2, 5), imshow(coins);**

**title('Ảnh gốc: coins');**

**subplot(4, 2, 6), imshow(coins\_1);**

**title('Ảnh nhị phân: coins');**

**subplot(4, 2, 7), imshow(eight);**

**title('Ảnh gốc: eight');**

**subplot(4, 2, 8), imshow(eight\_1);**

**title('Ảnh nhị phân: eight');**

**Chương 2**

# 2.7 Không dùng hàm có sẵn

**Chương 2**

# 2.8

**% Đọc ảnh f**

**f = imread('Fig0122(a)(fractal-iris).tif');**

**% a. Tạo ảnh h bằng cách đặt 4 plane bit thấp của ảnh f bằng 0**

**h = bitset(f, 1, 0); % Đặt bit 1 bằng 0**

**h = bitset(h, 2, 0); % Đặt bit 2 bằng 0**

**h = bitset(h, 3, 0); % Đặt bit 3 bằng 0**

**h = bitset(h, 4, 0); % Đặt bit 4 bằng 0**

**% b. Tạo ảnh g từ phép trừ f - h sử dụng vòng lặp**

**[M, N] = size(f); % Kích thước ảnh**

**g = zeros(M, N, 'uint8'); % Khởi tạo ảnh g với kích thước giống f và h**

**for x = 1:M**

**for y = 1:N**

**g(x, y) = f(x, y) - h(x, y); % Phép trừ từng pixel**

**end**

**end**

**% c. Tạo ảnh i từ việc cân bằng mức xám của ảnh g**

**i = histeq(g); % Cân bằng histogram của ảnh g**

**% Hiển thị các ảnh**

**figure;**

**subplot(2, 2, 1);**

**imshow(f);**

**title('Ảnh gốc f');**

**subplot(2, 2, 2);**

**imshow(h);**

**title('Ảnh h (4 plane bit thấp = 0)');**

**subplot(2, 2, 3);**

**imshow(g);**

**title('Ảnh g (f - h)');**

**subplot(2, 2, 4);**

**imshow(i);**

**title('Ảnh i (Cân bằng mức xám của g)');**

**Chương 2**

# 2.8 Không dùng hàm có sãn

**f = imread('Fig0122(a)(fractal-iris).tif'); % Đọc ảnh từ file và lưu vào biến f**

**% Nếu ảnh là ảnh màu, chuyển nó thành ảnh xám**

**if size(f, 3) == 3**

**f = rgb2gray(f); % Chuyển ảnh màu thành ảnh xám (grayscale)**

**end**

**% Tạo ảnh h bằng cách làm giảm độ phân giải của ảnh f xuống 4 bit thấp**

**h = floor(f / 16) \* 16; % Chia giá trị pixel của f cho 16, sau đó nhân lại với 16 để làm tròn xuống mức bậc 16 (chỉ giữ 4 bit thấp)**

**% Tạo ảnh g từ phép trừ giữa ảnh f và ảnh h**

**g = f - h; % g là phần còn lại sau khi làm giảm độ phân giải của ảnh f**

**% Cân bằng mức xám của ảnh g**

**% 1. Tính histogram (tần suất) của ảnh g**

**n = zeros(1, 256); % Mảng để chứa tần suất các mức xám của ảnh g (0-255)**

**[M, N] = size(g); % Lấy kích thước của ảnh g (M là chiều cao, N là chiều rộng)**

**% Tính histogram của ảnh g**

**for row = 1:M**

**for col = 1:N**

**level = g(row, col); % Lấy giá trị pixel tại vị trí (row, col)**

**n(level + 1) = n(level + 1) + 1; % Tăng tần suất của mức xám này**

**end**

**end**

**% 2. Tính CDF (Cumulative Distribution Function) của ảnh g**

**cdf = zeros(1, 256); % Mảng để lưu CDF**

**cdf(1) = n(1); % CDF ban đầu bằng tần suất của mức xám 0**

**for i = 2:256**

**cdf(i) = cdf(i - 1) + n(i); % Cập nhật CDF bằng tổng dồn các tần suất**

**end**

**% 3. Chuẩn hóa CDF để có giá trị pixel mới trong khoảng 0-255**

**% Tạo CDF chuẩn hóa**

**cdf\_normalized = round((cdf - min(cdf)) / (M \* N - min(cdf)) \* 255); % Chuẩn hóa CDF và chuyển sang giá trị từ 0 đến 255**

**% 4. Áp dụng CDF chuẩn hóa vào ảnh g để tạo ảnh mới i**

**i = cdf\_normalized(g + 1); % Thay thế giá trị pixel của ảnh g bằng giá trị từ CDF chuẩn hóa**

**% Hiển thị kết quả**

**figure;**

**% Hiển thị các ảnh f, h, g và i**

**subplot(2, 2, 1), imshow(f), title('Ảnh f');**

**subplot(2, 2, 2), imshow(h), title('Ảnh h');**

**subplot(2, 2, 3), imshow(g), title('Ảnh g');**

**subplot(2, 2, 4), imshow(i), title('Ảnh i (cân bằng mức xám)');**

====================================================================

**Chương 2**

# 2.9

**i = imread('Fig0333(a)(test\_pattern\_blurring\_orig).tif'); % Đọc ảnh từ file và lưu vào biến i**

**% Nếu ảnh là ảnh màu, chuyển thành ảnh xám (grayscale)**

**if size(i, 3) == 3**

**i = rgb2gray(i); % Chuyển ảnh màu thành ảnh xám**

**end**

**% Xây dựng một bộ lọc w (bộ lọc 3x3 với mỗi phần tử bằng 1/9)**

**w = ones(3, 3) / 9; % Bộ lọc trung bình 3x3 (các phần tử đều có giá trị bằng 1/9)**

**% Áp dụng lọc tuyến tính (dùng Correlation hoặc Convolution)**

**g = imfilter(i, w, 'corr', 'replicate', 'same'); % Lọc với Correlation, biên 'replicate', kích thước 'same'**

**% Hiển thị ảnh gốc và ảnh sau khi lọc**

**figure;**

**subplot(1, 2, 1), imshow(i), title('Ảnh gốc');**

**subplot(1, 2, 2), imshow(g), title('Ảnh sau khi lọc');**

**Cách 2:**

**% 1. Đọc ảnh f**

**i = imread('Fig0333(a)(test\_pattern\_blurring\_orig).tif'); % Đọc ảnh i**

**% Nếu ảnh là ảnh màu, chuyển thành ảnh xám**

**if size(i, 3) == 3**

**i = rgb2gray(i); % Chuyển ảnh màu thành ảnh xám**

**end**

**% 2. Xây dựng bộ lọc w (bộ lọc 3x3 với mỗi phần tử = 1/9)**

**w = ones(3, 3) / 9; % Bộ lọc trung bình 3x3**

**% 3. Kiểm tra tất cả các trường hợp với các giá trị khác nhau của filtering\_mode, boundary\_options, size\_options**

**filtering\_modes = {'corr', 'conv'}; % Các giá trị filtering\_mode**

**boundary\_options = {'replicate', 'symmetric', 'circular'}; % Các giá trị boundary\_options**

**size\_options = {'full', 'same'}; % Các giá trị size\_options**

**% Lặp qua các trường hợp**

**for fm = 1:length(filtering\_modes)**

**for bo = 1:length(boundary\_options)**

**for so = 1:length(size\_options)**

**% Áp dụng lọc với các tham số hiện tại**

**g = imfilter(i, w, filtering\_modes{fm}, boundary\_options{bo}, size\_options{so});**

**% Hiển thị kết quả**

**figure;**

**subplot(1, 2, 1), imshow(i), title('Ảnh gốc');**

**subplot(1, 2, 2), imshow(g), title(['Lọc: ' filtering\_modes{fm} ', Biên: ' boundary\_options{bo} ', Kích thước: ' size\_options{so}]);**

**end**

**end**

**end**

====================================================================

**Chương 2**

# 2.9 Không dùng hàm có sẵn

% 1. Đọc ảnh f

i = imread('Fig0333(a)(test\_pattern\_blurring\_orig).tif'); % Đọc ảnh i

% Nếu ảnh là ảnh màu, chuyển thành ảnh xám

if size(i, 3) == 3

i = rgb2gray(i); % Chuyển ảnh màu thành ảnh xám

end

% 2. Xây dựng bộ lọc w (bộ lọc 3x3 với mỗi phần tử = 1/9)

w = ones(3, 3) / 9; % Bộ lọc trung bình 3x3, mỗi phần tử có giá trị 1/9

% 3. Kích thước ảnh

[M, N] = size(i); % Kích thước ảnh đầu vào

g = zeros(M, N); % Khởi tạo ảnh đầu ra, ảnh sẽ được lưu tại đây

% 4. Áp dụng lọc thủ công (dùng Correlation hoặc Convolution)

% Lặp qua tất cả các pixel của ảnh i

for row = 1:M

for col = 1:N

% Lấy một vùng 3x3 xung quanh pixel (row, col)

% Đảm bảo không vượt quá biên của ảnh

row\_start = max(row-1, 1);

row\_end = min(row+1, M);

col\_start = max(col-1, 1);

col\_end = min(col+1, N);

% Cắt ra vùng con (sub-matrix) 3x3

region = i(row\_start:row\_end, col\_start:col\_end);

% Nếu cần, pad vùng con bằng biên 'replicate'

if row\_start == 1 || row\_end == M || col\_start == 1 || col\_end == N

region = padarray(region, [1, 1], 'replicate');

end

% Thực hiện phép Correlation (hoặc Convolution) bằng tích giữa vùng con và bộ lọc

g(row, col) = sum(sum(region .\* w)); % Tính tổng các tích giữa bộ lọc và vùng con

end

end

% 5. Hiển thị ảnh g (kết quả lọc)

figure;

subplot(1, 2, 1), imshow(i), title('Ảnh gốc');

subplot(1, 2, 2), imshow(uint8(g)), title('Ảnh sau khi lọc');

====================================================================

**Chương 2**

# 2.10

% a. Đọc ảnh và gây nhiễu muỗi - tiêu (salt & pepper noise)

i = imread('coins.png'); % Đọc ảnh i

% Kiểm tra xem ảnh đã là ảnh xám chưa

if size(i, 3) == 3

i = rgb2gray(i); % Nếu ảnh là ảnh màu, chuyển thành ảnh xám

end

% Gây nhiễu muỗi - tiêu với tỷ lệ 3%

noisy\_img\_saltpepper = imnoise(i, 'salt & pepper', 0.03); % Gây nhiễu muỗi - tiêu 3%

% Hiển thị ảnh gốc và ảnh đã gây nhiễu

figure;

subplot(1, 2, 1), imshow(i), title('Ảnh gốc');

subplot(1, 2, 2), imshow(noisy\_img\_saltpepper), title('Ảnh sau khi gây nhiễu muỗi - tiêu');

% b. Lọc trung vị với kích thước lân cận 3x3

filtered\_img\_saltpepper = medfilt2(noisy\_img\_saltpepper, [3 3]);

% Hiển thị ảnh sau khi lọc trung vị

figure;

imshow(filtered\_img\_saltpepper);

title('Ảnh sau khi lọc trung vị (muỗi - tiêu)');

% c. Gây nhiễu Gaussian 2% và áp dụng lọc trung vị

noisy\_img\_gaussian = imnoise(i, 'gaussian', 0, 0.02); % Gây nhiễu Gaussian với độ lệch chuẩn 0.02

% Lọc trung vị ảnh đã gây nhiễu Gaussian

filtered\_img\_gaussian = medfilt2(noisy\_img\_gaussian, [3 3]);

% Hiển thị ảnh gây nhiễu Gaussian và ảnh sau khi lọc

figure;

subplot(1, 2, 1), imshow(noisy\_img\_gaussian), title('Ảnh sau khi gây nhiễu Gaussian');

subplot(1, 2, 2), imshow(filtered\_img\_gaussian), title('Ảnh sau khi lọc trung vị (Gaussian)');

====================================================================

**Chương 2**

# 2.10 Không dùng hàm có sẵn

% a. Gây nhiễu muỗi - tiêu (salt and pepper noise) thành công

i = imread('coins.png'); % Đọc ảnh

% Kiểm tra xem ảnh đã là ảnh xám chưa

if size(i, 3) == 3

i = rgb2gray(i); % Nếu ảnh là ảnh màu, chuyển thành ảnh xám

end

% Gây nhiễu muỗi - tiêu với tỷ lệ 3%

[m, n] = size(i);

noisy\_img\_saltpepper = i; % Bắt đầu với ảnh gốc

% Xác định tỷ lệ nhiễu

noise\_ratio = 0.03;

% Số lượng pixel bị nhiễu

num\_noise = round(noise\_ratio \* m \* n);

% Chọn ngẫu nhiên các pixel để thay đổi thành 0 (muỗi) hoặc 255 (tiêu)

salt\_pepper\_pixels = rand(1, num\_noise); % Tạo số ngẫu nhiên để xác định muỗi và tiêu

% Gây nhiễu

% Chọn các pixel muỗi (0) và tiêu (255)

noisy\_img\_saltpepper(randperm(m\*n, num\_noise)) = 255 \* (salt\_pepper\_pixels > 0.5); % Muỗi-tiêu

% Hiển thị ảnh gốc và ảnh đã gây nhiễu

figure;

subplot(1, 2, 1), imshow(i), title('Ảnh gốc');

subplot(1, 2, 2), imshow(noisy\_img\_saltpepper), title('Ảnh sau khi gây nhiễu muỗi - tiêu');

**Bài 2.10 ko hàm có sẳn tiếp theo lần 1**

% b. Lọc trung vị với kích thước lân cận 3x3 (thực hiện thành công)

% Khởi tạo ảnh đầu ra

filtered\_img\_saltpepper = noisy\_img\_saltpepper;

% Kích thước của mặt nạ

mask\_size = 3;

pad\_size = floor(mask\_size / 2); % Kích thước phần tử padding

noisy\_img\_padded = padarray(noisy\_img\_saltpepper, [pad\_size, pad\_size], 'replicate');

% Duyệt qua tất cả các pixel trong ảnh

for row = 1:m

for col = 1:n

% Lấy vùng lân cận 3x3 xung quanh pixel (row, col)

region = noisy\_img\_padded(row:row+mask\_size-1, col:col+mask\_size-1);

% Tính trung vị của vùng lân cận

filtered\_img\_saltpepper(row, col) = median(region(:));

end

end

% Hiển thị ảnh sau khi lọc trung vị

figure;

imshow(filtered\_img\_saltpepper);

title('Ảnh sau khi lọc trung vị (muỗi - tiêu)');

% c. Tạo nhiễu Gaussian thành công và áp dụng lọc trung vị

% Độ lệch chuẩn của nhiễu Gaussian

std\_dev = 0.02; % Độ lệch chuẩn của nhiễu Gaussian

mean\_val = 0; % Giá trị trung bình của nhiễu Gaussian

% Chuyển ảnh gốc sang kiểu double để có thể cộng với nhiễu Gaussian

**Bài 2.10 ko hàm có sẳn tiếp theo lần 2**

i\_double = double(i);

% Gây nhiễu Gaussian (thành công)

noise\_gaussian = std\_dev \* randn(m, n); % Tạo nhiễu Gaussian với phân phối chuẩn

noisy\_img\_gaussian = i\_double + noise\_gaussian \* 255; % Cộng nhiễu Gaussian vào ảnh (phóng đại giá trị pixel)

% Đảm bảo giá trị pixel trong phạm vi [0, 255]

noisy\_img\_gaussian(noisy\_img\_gaussian > 255) = 255;

noisy\_img\_gaussian(noisy\_img\_gaussian < 0) = 0;

% Chuyển ảnh về kiểu uint8

noisy\_img\_gaussian = uint8(noisy\_img\_gaussian);

% Lọc trung vị ảnh đã gây nhiễu Gaussian (thực hiện thành công)

filtered\_img\_gaussian = noisy\_img\_gaussian;

noisy\_img\_padded = padarray(noisy\_img\_gaussian, [pad\_size, pad\_size], 'replicate');

% Duyệt qua tất cả các pixel trong ảnh

for row = 1:m

for col = 1:n

% Lấy vùng lân cận 3x3 xung quanh pixel (row, col)

region = noisy\_img\_padded(row:row+mask\_size-1, col:col+mask\_size-1);

% Tính trung vị của vùng lân cận

filtered\_img\_gaussian(row, col) = median(region(:));

end

end

% Hiển thị ảnh gây nhiễu Gaussian và ảnh sau khi lọc

figure;

subplot(1, 2, 1), imshow(noisy\_img\_gaussian, []), title('Ảnh sau khi gây nhiễu Gaussian');

subplot(1, 2, 2), imshow(filtered\_img\_gaussian, []), title('Ảnh sau khi lọc trung vị (Gaussian)');

====================================================================

**Chương 2**

# 2.11

% a. Đọc ảnh

i = imread('Fig0335(a)(ckt\_board\_saltpep\_prob\_pt05).tif'); % Đọc ảnh vào biến i

% Nếu ảnh là ảnh màu, chuyển thành ảnh xám

if size(i, 3) == 3

i = rgb2gray(i); % Chuyển ảnh màu thành ảnh xám

end

% b. Lọc tuyến tính (sử dụng bộ lọc trung bình 3x3)

h = fspecial('average', [3 3]); % Tạo bộ lọc trung bình 3x3

g\_linear = conv2(double(i), h, 'same'); % Áp dụng phép convolution để lọc tuyến tính

% Chuyển kết quả về kiểu uint8

g\_linear = uint8(g\_linear);

% c. Lọc phi tuyến (lọc trung vị 3x3)

g\_median = medfilt2(i, [3 3]); % Áp dụng lọc trung vị với lân cận 3x3

% d. Hiển thị kết quả

figure;

% Hiển thị ảnh gốc và các ảnh sau khi lọc

subplot(1, 3, 1), imshow(i), title('Ảnh gốc');

subplot(1, 3, 2), imshow(g\_linear), title('Lọc tuyến tính');

subplot(1, 3, 3), imshow(g\_median), title('Lọc phi tuyến (Trung vị)');

====================================================================

**Chương 2**

# 2.11 Không dùng hàm có sẵn

% a. Đọc ảnh

i = imread('Fig0335(a)(ckt\_board\_saltpep\_prob\_pt05).tif'); % Đọc ảnh vào biến i

% Nếu ảnh là ảnh màu, chuyển thành ảnh xám

if size(i, 3) == 3

i = rgb2gray(i); % Chuyển ảnh màu thành ảnh xám

end

% Chuyển ảnh về kiểu double để xử lý toán học

i = double(i);

% b. Lọc tuyến tính (Bộ lọc trung bình 3x3)

[M, N] = size(i);

w = ones(3, 3) / 9; % Bộ lọc trung bình 3x3

% Tạo ảnh đầu ra cho lọc tuyến tính

g\_linear = zeros(M, N);

% Duyệt qua từng pixel của ảnh

for row = 2:M-1

for col = 2:N-1

% Lấy một vùng 3x3 xung quanh pixel (row, col)

region = i(row-1:row+1, col-1:col+1);

% Áp dụng bộ lọc trung bình và tính giá trị pixel mới

g\_linear(row, col) = sum(sum(region .\* w));

end

end

% Chuyển kết quả về lại kiểu uint8

g\_linear = uint8(g\_linear);

% c. Lọc phi tuyến (Lọc trung vị 3x3)

g\_median = zeros(M, N);

% Duyệt qua từng pixel của ảnh

for row = 2:M-1

for col = 2:N-1

% Lấy một vùng 3x3 xung quanh pixel (row, col)

region = i(row-1:row+1, col-1:col+1);

% Tính giá trị trung vị của vùng 3x3

g\_median(row, col) = median(region(:));

end

end

% Chuyển kết quả về lại kiểu uint8

g\_median = uint8(g\_median);

% d. Hiển thị kết quả

figure;

% Hiển thị ảnh gốc và các ảnh sau khi lọc

subplot(1, 3, 1), imshow(uint8(i)), title('Ảnh gốc');

subplot(1, 3, 2), imshow(g\_linear), title('Lọc tuyến tính');

subplot(1, 3, 3), imshow(g\_median), title('Lọc phi tuyến (Trung vị)');

**Lý thuyết**

**a. Lọc tuyến tính**

Lọc tuyến tính là quá trình áp dụng một bộ lọc (mặt nạ) vào ảnh theo cách mà giá trị của mỗi pixel trong ảnh sau lọc là một hàm tuyến tính của các pixel trong vùng lân cận của nó. Ví dụ, bộ lọc trung bình là một loại lọc tuyến tính phổ biến, trong đó mỗi pixel trong vùng lân cận sẽ có giá trị trung bình của các pixel xung quanh nó.

Trong mã mẫu bạn đưa ra, bộ lọc trung bình 3x3 được áp dụng lên ảnh bằng hàm fspecial('average', [3 3]) để tạo bộ lọc trung bình và sau đó sử dụng conv2() để thực hiện phép toán convolution trên ảnh. Kết quả là một ảnh mờ (blurred), vì bộ lọc trung bình làm mờ các nhiễu có tần số cao trong ảnh, nhưng lại không xử lý tốt được các nhiễu điểm (salt-and-pepper noise).

**b. Lọc phi tuyến**

Lọc phi tuyến là loại lọc mà quá trình xử lý không tuân theo một phép toán tuyến tính. Một trong những kỹ thuật lọc phi tuyến phổ biến là lọc trung vị (median filter). Trong lọc trung vị, giá trị của mỗi pixel sau khi lọc sẽ là giá trị trung vị trong vùng lân cận của nó thay vì là giá trị trung bình.

Lọc trung vị rất hiệu quả trong việc loại bỏ nhiễu điểm (salt-and-pepper noise) vì nó thay thế các giá trị cực trị (rất sáng hoặc rất tối) trong vùng lân cận bằng giá trị trung bình trung vị, giúp giữ lại các chi tiết của ảnh trong khi loại bỏ các điểm nhiễu.

|  |  |  |
| --- | --- | --- |
| Tiêu chí | Lọc tuyến tính (Bộ lọc trung bình) | Lọc phi tuyến (Lọc trung vị) |
| Khả năng loại bỏ nhiễu | Tốt cho các nhiễu đồng đều, nhưng không hiệu quả với nhiễu điểm (salt-and-pepper noise). | Rất tốt trong việc loại bỏ nhiễu điểm (salt-and-pepper noise), nhưng có thể làm mất các chi tiết mịn trong ảnh. |
| Ảnh sau khi lọc | Làm mờ toàn bộ ảnh, giảm nhiễu nhưng có thể làm mất chi tiết. | Giữ được chi tiết hơn so với lọc tuyến tính, nhưng có thể làm mờ vùng có biên rõ rệt. |
| Ứng dụng | Thường được dùng để làm mờ ảnh hoặc giảm nhiễu đồng đều. | Thường được dùng để loại bỏ nhiễu điểm mà không làm mờ các chi tiết quan trọng trong ảnh. |

**Nhận xét**

Lọc tuyến tính có thể làm mờ ảnh và giảm nhiễu tốt với các loại nhiễu đồng đều (Gaussian noise), nhưng lại không hiệu quả khi xử lý nhiễu điểm (salt-and-pepper noise). Lọc này cũng làm mất một số chi tiết trong ảnh.

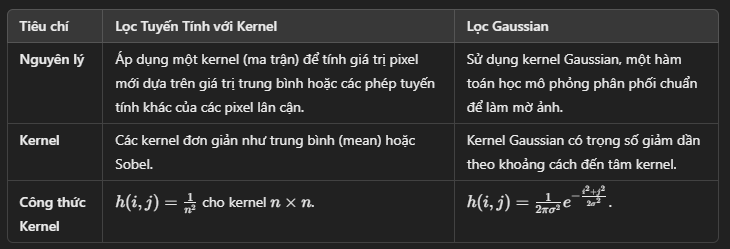
Lọc phi tuyến (lọc trung vị) lại đặc biệt hiệu quả trong việc loại bỏ nhiễu điểm mà không làm mờ quá nhiều chi tiết ảnh. Tuy nhiên, nếu có quá nhiều chi tiết nhỏ hoặc biên sắc nét trong ảnh, phương pháp này có thể làm mờ các biên và chi tiết đó.

Tóm lại, việc chọn lựa lọc tuyến tính hay phi tuyến phụ thuộc vào loại nhiễu bạn đang xử lý và yêu cầu về chi tiết ảnh sau khi lọc.

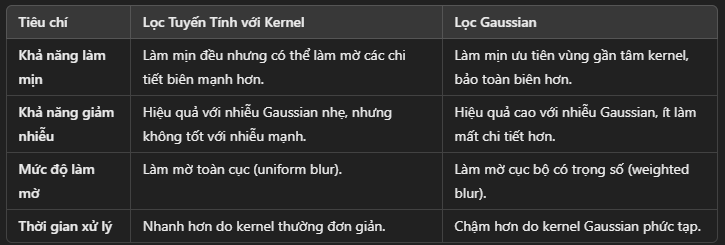
**So sánh Lọc Tuyến Tính sử dụng Kernel và Lọc Gaussian**

Lọc tuyến tính với kernel (như lọc trung bình) và lọc Gaussian đều là các phương pháp xử lý ảnh dựa trên việc làm mịn (smoothing) hoặc giảm nhiễu. Tuy nhiên, chúng có sự khác biệt về nguyên lý, hiệu quả, và ứng dụng.

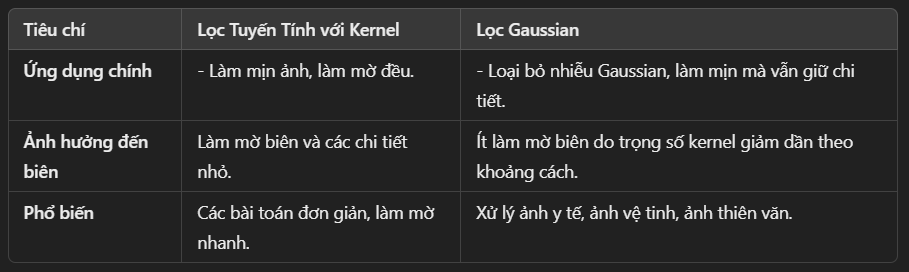
**1. Nguyên lý hoạt động**



**2. Hiệu quả và đặc điểm**

****

**3. Ứng dụng**

****

# 3.1 Phát hiện biên Canny

clc;

clear;

close all;

% Đọc ảnh xám từ tệp 'trui.png'

A = imread('trui.png');

% Hiển thị ảnh gốc ở vị trí (1,1) trên lưới 3x3

subplot(3,3,1), imshow(A,[]);

title('Ảnh gốc');

% Tạo hai bộ lọc Gaussian với các kích thước và độ lệch chuẩn khác nhau

h1 = fspecial('gaussian', [15 15], 6); % Kernel 15x15, sigma = 6 (làm mờ nhẹ)

h2 = fspecial('gaussian', [30 30], 12); % Kernel 30x30, sigma = 12 (làm mờ mạnh hơn)

% Áp dụng bộ lọc Gaussian h1 và hiển thị kết quả ở vị trí (2,1)

subplot(3,3,4), imshow(imfilter(A, h1), []);

title('Ảnh làm mờ (Gaussian 15x15, sigma=6)');

% Áp dụng bộ lọc Gaussian h2 và hiển thị kết quả ở vị trí (3,1)

subplot(3,3,7), imshow(imfilter(A, h2), []);

title('Ảnh làm mờ (Gaussian 30x30, sigma=12)');

% Phát hiện biên bằng phương pháp LoG (Laplacian of Gaussian) trên ảnh gốc

[bw, thresh] = edge(A, 'log');

subplot(3,3,2), imshow(bw, []);

title('Biên LoG (ảnh gốc)');

% Phát hiện biên bằng phương pháp Canny trên ảnh gốc

[bw, thresh] = edge(A, 'canny');

subplot(3,3,3), imshow(bw, []);

title('Biên Canny (ảnh gốc)');

% Phát hiện biên bằng LoG trên ảnh đã làm mờ bằng Gaussian h1

[bw, thresh] = edge(imfilter(A, h1), 'log');

subplot(3,3,5), imshow(bw, []);

title('Biên LoG (Gaussian 15x15, sigma=6)');

% Phát hiện biên bằng Canny trên ảnh đã làm mờ bằng Gaussian h1

[bw, thresh] = edge(imfilter(A, h1), 'canny');

subplot(3,3,6), imshow(bw, []);

title('Biên Canny (Gaussian 15x15, sigma=6)');

% Phát hiện biên bằng LoG trên ảnh đã làm mờ bằng Gaussian h2

[bw, thresh] = edge(imfilter(A, h2), 'log');

subplot(3,3,8), imshow(bw, []);

title('Biên LoG (Gaussian 30x30, sigma=12)');

% Phát hiện biên bằng Canny trên ảnh đã làm mờ bằng Gaussian h2

[bw, thresh] = edge(imfilter(A, h2), 'canny');

subplot(3,3,9), imshow(bw, []);

title('Biên Canny (Gaussian 30x30, sigma=12)');

# 3.2 Phát hiện biên (ngang. +45 độ, dọc, -45 độ)

Hàm conv2 trong MATLAB được sử dụng để thực hiện tích chập 2D giữa một ảnh và một kernel (bộ lọc/mặt nạ). Đây là một phép toán quan trọng trong xử lý ảnh, đặc biệt là trong phát hiện biên, làm mờ ảnh, làm sắc nét, v.v.

1. Cú pháp của conv2

matlab

Sao chépChỉnh sửa

C = conv2(A, B, shape)

Trong đó:

* A: Ma trận đầu vào (thường là ảnh xám).
* B: Bộ lọc (kernel/mặt nạ).
* shape: Cách xử lý biên, có 3 tùy chọn chính:
  + 'full' (mặc định) → Kết quả có kích thước lớn hơn ảnh gốc.
  + 'same' → Kết quả có cùng kích thước với ảnh gốc.
  + 'valid' → Kết quả có kích thước nhỏ hơn (loại bỏ biên ngoài).

**NGANG dùng hàm CONV2**

clc;

clear;

close all;

% Đọc ảnh xám

A = imread('trui.png');

A = double(A); % Chuyển ảnh sang kiểu double để xử lý chính xác hơn

% Định nghĩa mặt nạ tùy chỉnh

custom\_mask = [-1 -1 -1; 2 2 2; -1 -1 -1];

% Áp dụng mặt nạ bằng phép tích chập

custom\_edge = conv2(A, custom\_mask, 'same');

% Hiển thị kết quả

figure;

subplot(1,2,1), imshow(A, []); title('Ảnh gốc');

subplot(1,2,2), imshow(abs(custom\_edge), []); title('Biên ngang (Mặt nạ tùy chỉnh)');

**NGANG KHÔNG dùng hàm CONV2**

clc;

clear;

close all;

% Đọc ảnh xám

A = imread('trui.png');

A = double(A); % Chuyển ảnh sang kiểu double để xử lý chính xác hơn

% Định nghĩa mặt nạ phát hiện biên ngang

custom\_mask = [-1 -1 -1; 2 2 2; -1 -1 -1];

% Kích thước ảnh

[rows, cols] = size(A);

% Tạo ảnh kết quả có cùng kích thước

edge\_custom = zeros(rows, cols);

% Duyệt từng pixel (trừ viền ảnh)

for i = 2:rows-1

for j = 2:cols-1

% Trích xuất cửa sổ 3x3

window = A(i-1:i+1, j-1:j+1);

% Nhân từng phần tử với mặt nạ rồi tính tổng

edge\_custom(i, j) = sum(sum(window .\* custom\_mask));

end

end

% Hiển thị kết quả

figure;

subplot(1,2,1), imshow(A, []); title('Ảnh gốc');

subplot(1,2,2), imshow(abs(edge\_custom), []); title('Phát hiện biên (Mặt nạ tùy chỉnh)');

**+45 độ không dùng CONV2**

clc;

clear;

close all;

% Đọc ảnh xám

A = imread('trui.png');

A = double(A); % Chuyển ảnh sang kiểu double để xử lý chính xác hơn

% Định nghĩa mặt nạ phát hiện biên +45 độ

custom\_mask = [-1 -1 2; -1 2 -1; 2 -1 -1];

% Kích thước ảnh

[rows, cols] = size(A);

% Tạo ảnh kết quả có cùng kích thước

edge\_custom = zeros(rows, cols);

% Duyệt từng pixel (trừ viền ảnh)

for i = 2:rows-1

for j = 2:cols-1

% Trích xuất cửa sổ 3x3

window = A(i-1:i+1, j-1:j+1);

% Nhân từng phần tử với mặt nạ rồi tính tổng

edge\_custom(i, j) = sum(sum(window .\* custom\_mask));

end

end

% Hiển thị kết quả

figure;

subplot(1,2,1), imshow(A, []); title('Ảnh gốc');

subplot(1,2,2), imshow(abs(edge\_custom), []); title('Phát hiện biên (Mặt nạ tùy chỉnh)');

**ĐỐI VỚI DỌC VÀ -45 ĐỘ THÌ ĐỔI CÁI**

**custom\_mask THÀNH MA TRẬN MẶT NẠ TƯƠNG ỨNG LÀ ĐƯỢC**

# 3.3 Phát hiện biên bằng phương pháp ‘sobel’, ‘prewitt’, ‘roberts’

**SOBEL (Không dùng hàm có sẵn)**

clc;

clear;

close all;

% Đọc ảnh xám

A = imread('trui.png');

A = double(A); % Chuyển sang kiểu double để tính toán chính xác

% Định nghĩa mặt nạ Sobel

sobel\_x = [-1 0 1; -2 0 2; -1 0 1]; % Biên theo chiều dọc

sobel\_y = [-1 -2 -1; 0 0 0; 1 2 1]; % Biên theo chiều ngang

% Kích thước ảnh

[rows, cols] = size(A);

% Khởi tạo ảnh biên

Gx = zeros(rows, cols);

Gy = zeros(rows, cols);

G = zeros(rows, cols);

% Tính toán tích chập thủ công

for i = 2:rows-1

for j = 2:cols-1

% Cửa sổ 3x3 quanh pixel (i, j)

window = A(i-1:i+1, j-1:j+1);

% Áp dụng mặt nạ Sobel theo X và Y

Gx(i, j) = sum(sum(window .\* sobel\_x));

Gy(i, j) = sum(sum(window .\* sobel\_y));

% Tổng hợp biên

G(i, j) = sqrt(Gx(i, j)^2 + Gy(i, j)^2);

end

end

% Hiển thị kết quả

figure;

subplot(1,3,1), imshow(A, []); title('Ảnh gốc');

subplot(1,3,2), imshow(abs(Gx), []); title('Biên theo X (Sobel)');

subplot(1,3,3), imshow(abs(G), []); title('Biên tổng hợp (Sobel)');

**PREWITT**

clc;

clear;

close all;

% Đọc ảnh xám

A = imread('trui.png');

A = double(A); % Chuyển sang kiểu double để xử lý chính xác hơn

% Định nghĩa mặt nạ Prewitt

prewitt\_x = [-1 0 1; -1 0 1; -1 0 1]; % Phát hiện biên dọc

prewitt\_y = [-1 -1 -1; 0 0 0; 1 1 1]; % Phát hiện biên ngang

% Kích thước ảnh

[rows, cols] = size(A);

% Khởi tạo ảnh biên

Gx = zeros(rows, cols);

Gy = zeros(rows, cols);

G = zeros(rows, cols);

% Tính toán tích chập thủ công

for i = 2:rows-1

for j = 2:cols-1

% Cửa sổ 3x3 quanh pixel (i, j)

window = A(i-1:i+1, j-1:j+1);

% Áp dụng mặt nạ Prewitt theo X và Y

Gx(i, j) = sum(sum(window .\* prewitt\_x));

Gy(i, j) = sum(sum(window .\* prewitt\_y));

% Tổng hợp biên

G(i, j) = sqrt(Gx(i, j)^2 + Gy(i, j)^2);

end

end

% Hiển thị kết quả

figure;

subplot(1,3,1), imshow(A, []); title('Ảnh gốc');

subplot(1,3,2), imshow(abs(Gx), []); title('Biên theo X (Prewitt)');

subplot(1,3,3), imshow(abs(G), []); title('Biên tổng hợp (Prewitt)');

**ROBERTS**

clc;

clear;

close all;

% Đọc ảnh xám

A = imread('trui.png');

A = double(A); % Chuyển sang kiểu double để xử lý chính xác hơn

% Định nghĩa mặt nạ Roberts

roberts\_x = [1 0; 0 -1]; % Phát hiện biên theo hướng chéo 1

roberts\_y = [0 1; -1 0]; % Phát hiện biên theo hướng chéo 2

% Kích thước ảnh

[rows, cols] = size(A);

% Khởi tạo ảnh biên

Gx = zeros(rows, cols);

Gy = zeros(rows, cols);

G = zeros(rows, cols);

% Tính toán tích chập thủ công (bỏ qua viền ngoài)

for i = 1:rows-1

for j = 1:cols-1

% Cửa sổ 2x2 quanh pixel (i, j)

window = A(i:i+1, j:j+1);

% Áp dụng mặt nạ Roberts theo X và Y

Gx(i, j) = sum(sum(window .\* roberts\_x));

Gy(i, j) = sum(sum(window .\* roberts\_y));

% Tổng hợp biên

G(i, j) = sqrt(Gx(i, j)^2 + Gy(i, j)^2);

end

end

% Hiển thị kết quả

figure;

subplot(1,3,1), imshow(A, []); title('Ảnh gốc');

subplot(1,3,2), imshow(abs(Gx), []); title('Biên theo X (Roberts)');

subplot(1,3,3), imshow(abs(G), []); title('Biên tổng hợp (Roberts)');

**Hàm** **SOBEL (Không dùng CONV2)**  
clc;

clear;

close all;

% Đọc ảnh xám

A = imread('trui.png');

% Gọi hàm Sobel

G = sobel\_edge(A);

% Hiển thị kết quả

figure;

subplot(1,2,1), imshow(A, []); title('Ảnh gốc');

subplot(1,2,2), imshow(G, []); title('Biên ảnh (Sobel)');

function G = sobel\_edge(A)

% Chuyển ảnh sang kiểu double để tính toán chính xác

A = double(A);

% Định nghĩa mặt nạ Sobel

sobel\_x = [-1 0 1; -2 0 2; -1 0 1]; % Phát hiện biên dọc

sobel\_y = [-1 -2 -1; 0 0 0; 1 2 1]; % Phát hiện biên ngang

% Kích thước ảnh

[rows, cols] = size(A);

% Khởi tạo ảnh biên

Gx = zeros(rows, cols);

Gy = zeros(rows, cols);

G = zeros(rows, cols);

% Tính toán tích chập thủ công (bỏ qua viền ngoài)

for i = 2:rows-1

for j = 2:cols-1

% Cửa sổ 3x3 quanh pixel (i, j)

window = A(i-1:i+1, j-1:j+1);

% Áp dụng mặt nạ Sobel theo X và Y

Gx(i, j) = sum(sum(window .\* sobel\_x));

Gy(i, j) = sum(sum(window .\* sobel\_y));

% Tổng hợp biên

G(i, j) = sqrt(Gx(i, j)^2 + Gy(i, j)^2);

end

end

end

**Hàm** **PREWITT (Không dùng CONV2)**

clc;

clear;

close all;

% Đọc ảnh xám

A = imread('trui.png');

% Gọi hàm Prewitt

G = prewitt\_edge(A);

% Hiển thị kết quả

figure;

subplot(1,2,1), imshow(A, []); title('Ảnh gốc');

subplot(1,2,2), imshow(G, []); title('Biên ảnh (Prewitt)');

function G = prewitt\_edge(A)

% Chuyển ảnh sang kiểu double để tính toán chính xác

A = double(A);

% Định nghĩa mặt nạ Prewitt

prewitt\_x = [-1 0 1; -1 0 1; -1 0 1]; % Phát hiện biên dọc

prewitt\_y = [-1 -1 -1; 0 0 0; 1 1 1]; % Phát hiện biên ngang

% Kích thước ảnh

[rows, cols] = size(A);

% Khởi tạo ảnh biên

Gx = zeros(rows, cols);

Gy = zeros(rows, cols);

G = zeros(rows, cols);

% Tính toán tích chập thủ công (bỏ qua viền ngoài)

for i = 2:rows-1

for j = 2:cols-1

% Cửa sổ 3x3 quanh pixel (i, j)

window = A(i-1:i+1, j-1:j+1);

% Áp dụng mặt nạ Prewitt theo X và Y

Gx(i, j) = sum(sum(window .\* prewitt\_x));

Gy(i, j) = sum(sum(window .\* prewitt\_y));

% Tổng hợp biên

G(i, j) = sqrt(Gx(i, j)^2 + Gy(i, j)^2);

end

end

end

**Hàm** **ROBERTS (Không dùng CONV2)**

clc;

clear;

close all;

% Đọc ảnh xám

A = imread('trui.png');

% Gọi hàm Roberts

G = roberts\_edge(A);

% Hiển thị kết quả

figure;

subplot(1,2,1), imshow(A, []); title('Ảnh gốc');

subplot(1,2,2), imshow(G, []); title('Biên ảnh (Roberts)');

function G = roberts\_edge(A)

% Chuyển ảnh sang kiểu double để tính toán chính xác

A = double(A);

% Định nghĩa mặt nạ Roberts

roberts\_x = [1 0; 0 -1]; % Phát hiện biên theo đường chéo chính

roberts\_y = [0 1; -1 0]; % Phát hiện biên theo đường chéo phụ

% Kích thước ảnh

[rows, cols] = size(A);

% Khởi tạo ảnh biên

Gx = zeros(rows, cols);

Gy = zeros(rows, cols);

G = zeros(rows, cols);

% Tính toán tích chập thủ công (bỏ qua viền ngoài)

for i = 1:rows-1

for j = 1:cols-1

% Cửa sổ 2x2 quanh pixel (i, j)

window = A(i:i+1, j:j+1);

% Áp dụng mặt nạ Roberts theo X và Y

Gx(i, j) = sum(sum(window .\* roberts\_x));

Gy(i, j) = sum(sum(window .\* roberts\_y));

% Tổng hợp biên

G(i, j) = sqrt(Gx(i, j)^2 + Gy(i, j)^2);

end

end

end