

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA



MÔN HỌC: XỬ LÝ SỐ TÍN HIỆU (EE2015)
BÁO CÁO BÀI TẬP LỚN
GIẢM NHIỀU HÌNH ẢNH BẰNG RASPBERRY PI 3

Giảng viên hướng dẫn: GS. TS. Lê Tiến Thường

Sinh viên thực hiện: Phạm Khánh

MSSV: 2011391

Lê Vĩnh Hưng

MSSV: 2010305

Nhóm P01 – Nhóm 2

Tp. Hồ Chí Minh – 2024

Mục lục

1	Giới thiệu	1
2	Mục đích đề tài	1
3	Cơ sở lý thuyết	1
3.1	Ảnh số	1
3.2	Phép toán giữa các điểm ảnh	3
3.3	Một số loại nhiễu ảnh thường gặp	4
3.3.1	Nhiều Gauss	4
3.3.2	Nhiều muối tiêu	4
3.4	Các loại bộ lọc	5
4	Công cụ yêu cầu cho bài thí nghiệm	6
4.1	Phần mềm và phần cứng	6
4.2	Giới thiệu bộ kit Raspberry Pi 3	7
5	Tiến hành thí nghiệm	8
5.1	Lọc nhiễu ảnh muối tiêu	9
5.1.1	Tạo ảnh nhiễu muối tiêu trên MATLAB	9
5.1.2	Lọc ảnh nhiễu muối tiêu bằng Python trên Raspberry Pi 3	10
5.1.3	So sánh kết quả với MATLAB	12
5.2	Lọc nhiễu ảnh Gauss	13
5.2.1	Tạo ảnh nhiễu Gauss	13
5.2.2	Lọc ảnh nhiễu Gauss bằng Python trên Raspberry Pi 3	14
5.2.3	So sánh kết quả với MATLAB	17
6	Kết luận	20
6.1	Tóm tắt những phát hiện chính	20
6.2	Mục tiêu đã đạt được	20
6.3	Tính hiệu quả của từng phương pháp	20

6.4	Hướng phát triển của đề tài	20
	Tài liệu tham khảo	21

1 Giới thiệu

Lĩnh vực hình ảnh kỹ thuật số luôn phải đối mặt với một thách thức dai dẳng: nhiều hình ảnh. Sự thay đổi không mong muốn về độ sáng hoặc màu sắc này xuất phát từ các yếu tố như hạn chế của cảm biến hoặc điều kiện ánh sáng yếu, có thể che khuất các chi tiết và làm giảm độ trung thực của hình ảnh. Do đó, đề tài này đề cập đến các kỹ thuật giảm nhiễu hình ảnh thực hiện trên phần cứng Raspberry Pi 3.

Đề tài này có ứng dụng thực tiễn cao vì hai mục đích chính. Đầu tiên, ảnh số hiện rất phổ biến trong nhiều lĩnh vực khác nhau như nhiếp ảnh, ảnh y tế, hệ thống giám sát... và nhiễu luôn là một vấn đề lớn. Thứ hai, các phần cứng nhúng ngày càng phổ biến và có giá thành rẻ hơn, cùng với sức mạnh tính toán ngày càng mạnh mẽ hơn. Điều này dẫn đến nhu cầu chỉnh sửa hoặc phát triển thêm các kỹ thuật giảm nhiễu hình ảnh, đáp ứng với sự phát triển của phần cứng.

2 Mục đích đề tài

- Tìm hiểu cơ bản kỹ thuật xử lý ảnh nói chung và kỹ thuật giảm nhiễu hình ảnh nói riêng.
- Triển khai thực tế các thuật toán giảm nhiễu hình ảnh trên phần cứng và kiểm nghiệm so với lý thuyết.

3 Cơ sở lý thuyết

3.1 Ảnh số

Ảnh $f(x, y)$ được miêu tả bằng những mẫu cách đều nhau ở dạng ma trận $(N - M)$:

$$f(x, y) \begin{bmatrix} f(0, 0) & f(0, 1) & \cdots & f(0, M - 1) \\ f(1, 0) & f(1, 1) & \cdots & f(1, M - 1) \\ \cdots & \cdots & \cdots & \cdots \\ f(N - 1, 0) & f(N - 1, 1) & \cdots & f(N - 1, M - 1) \end{bmatrix} = A$$

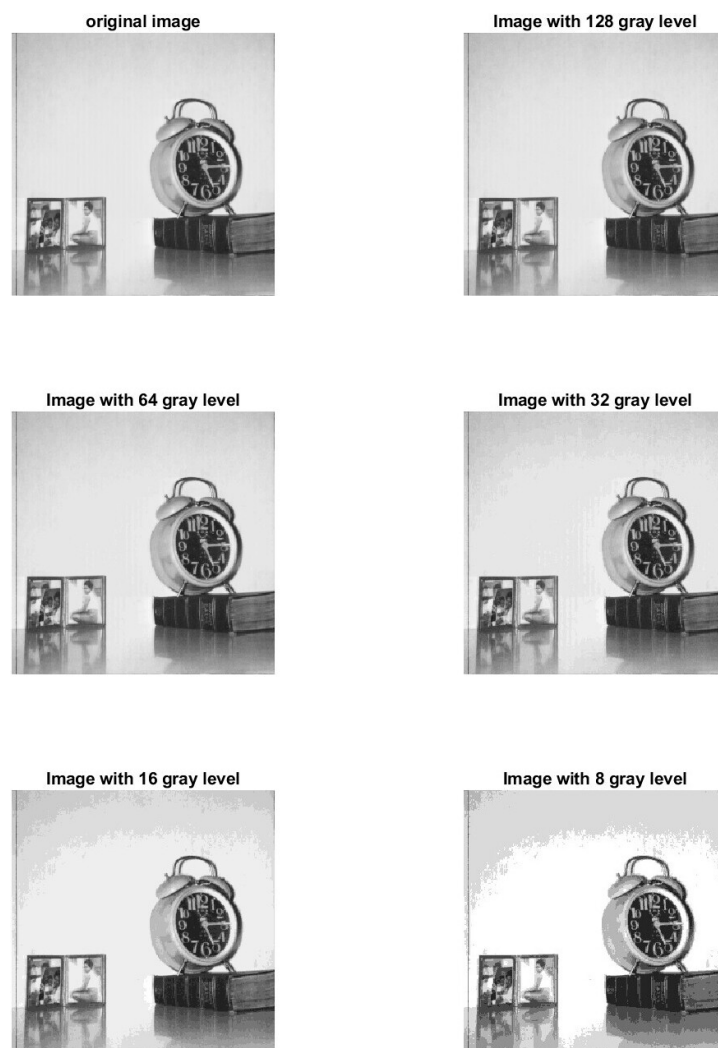
Ma trận A được gọi là ảnh số, mỗi thành phần của A được gọi là một điểm ảnh, hay pixel.

Lấy mẫu: chia mặt phẳng xy thành mắt lưới, tọa độ của mỗi mắt lưới là (x, y) , trong đó x, y là số nguyên.

Lượng tử hóa: f được gán bằng một giá trị mức xám G (thực hoặc nguyên).

Độ phân giải cho biết mức độ chi tiết của ảnh. Độ phân giải có thể phân loại thành (1) độ phân giải không gian và (2) độ phân giải mức xám.

1. Độ phân giải không gian: là chi tiết nhỏ nhất có thể nhìn thấy được trong một hình ảnh. Độ phân giải không gian phụ thuộc vào số lượng điểm ảnh. Yếu tố chính quyết định độ phân giải không gian là lấy mẫu.
2. Độ phân giải mức xám: đề cập đến sự thay đổi nhỏ nhất có thể nhận thấy được trong mức xám. Độ phân giải mức xám phụ thuộc vào số lượng mức xám.



Hình 1: Ảnh xám với các độ phân giải mức xám khác nhau

3.2 Phép toán giữa các điểm ảnh

Các phép toán số học và logic trên hai ảnh được thực hiện trên từng điểm ảnh. Các phép toán này chỉ thực hiện được nếu hai ảnh có cùng kích thước, nhưng nếu điều này không thỏa, ta vẫn có thể thực hiện các phép toán này trên vùng giao của hai ảnh. Nếu hai ảnh có kích thước $w_1 \times h_1$ và $w_2 \times h_2$ và giả sử gốc tọa độ của chúng giống nhau, vậy kết quả của phép toán trên hai ảnh này sẽ có kích thước $w \times h$, trong đó:

$$w = \min(w_1, w_2) \quad (1)$$

$$h = \min(h_1, h_2) \quad (2)$$

Phép cộng và trừ ảnh

Cho hai ảnh x_1 và x_2 có vùng giao nhau có kích thước $W \times H$, y là kết quả của phép toán giữa hai điểm ảnh, và các ảnh đều có độ phân giải mức xám N bit. Ta có các phép toán cộng và trừ giữa hai ảnh như sau:

$$y(i, j) = \min\{x_1(i, j) + x_2(i, j), 2^N\} \quad (3)$$

$$y(i, j) = \|x_1(i, j) - x_2(i, j)\| \quad (4)$$

Trong đó, $0 \leq i \leq H - 1$ và $0 \leq j \leq W - 1$.

Phép cộng ảnh có ứng dụng trong việc giảm nhiễu hình ảnh bằng cách lấy trung bình nhiều quan sát của cùng một khung hình. Phép trừ ảnh có ứng dụng trong việc phát hiện chuyển động, thông qua phân tích các điểm ảnh có giá trị khác không sau khi thực hiện phép trừ.

Phép nhân và chia ảnh với một số

Cho ảnh x là ảnh thực hiện phép toán và ảnh y là kết quả, có cùng kích thước và độ phân giải mức xám N bit. Phép nhân ảnh với một số k được biểu diễn như sau:

$$y(i, j) = kx(i, j) \quad (0 \leq i \leq H - 1, 0 \leq j \leq W - 1) \quad (5)$$

Phép chia ảnh với một số tương tự như phép nhân, trong đó $|k| < 1$.

Phép nhân và chia các điểm ảnh với một số có thể được dùng để điều chỉnh độ sáng của một ảnh. Nhân các điểm ảnh với một số lớn hơn một sẽ làm sáng ảnh, và chia các điểm ảnh với một số lớn hơn một sẽ làm tối ảnh. Nhân (hoặc chia) các điểm ảnh với một số âm sẽ làm đảo màu của ảnh.

Phép tích chập: Tích chập rời rạc hai chiều giữa hai tín hiệu $x[n_1, n_2]$ và $h[n_1, n_2]$ là:

$$y(n_1, n_2) = \sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} x(k_1, k_2)h(n_1 - k_1, n_2 - k_2)$$

Ở góc nhìn khác, phép tích chập hai chiều trong ảnh số chính là tính toán giá trị mới của điểm ảnh dưới dạng tổng có trọng số của các giá trị điểm ảnh trong một vùng lân cận nhất định xung quanh nó.

Tích chập có thể được sử dụng để thực hiện quá trình lọc tuyến tính.

3.3 Một số loại nhiễu ảnh thường gặp

3.3.1 Nhiễu Gauss

Nhiễu Gauss là nhiễu có hàm mật độ xác suất là phân phối Gauss. Nói cách khác, các giá trị mà nhiễu có thể nhận được phân bố theo Gauss.

Hàm mật độ xác suất p của một biến ngẫu nhiên Gauss được cho bởi

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\bar{z})^2/2\sigma^2}$$

Trong đó σ^2 là phương sai và \bar{z} là giá trị trung bình.

Các nguồn nhiễu Gauss chính trong ảnh số phát sinh trong quá trình thu thập. Cảm biến vốn có nhiễu do mức độ chiếu sáng và nhiệt độ của chính nó, đồng thời các mạch điện tử được kết nối với cảm biến cũng tạo ra phần nhiễu riêng của chúng.

3.3.2 Nhiễu muối tiêu

Nhiễu muối tiêu hay còn gọi là *nhiễu xung* là một dạng nhiễu đôi khi được thấy trên ảnh số. Nhiễu này có thể được gây ra bởi sự nhiễu loạn rõ rệt và đột ngột trong tín hiệu ảnh. Nhiễu này được thể hiện dưới dạng các điểm ảnh trắng và đen xuất hiện thưa thớt.



Hình 2: Một ảnh với nhiễu muối tiêu

Hàm mật độ xác suất của nhiễu muối tiêu được cho bởi

$$p(z) = \begin{cases} P_a & z = a \\ P_b & z = b \\ 0 & \text{khác} \end{cases}$$

Trong đó $a = 0$ và $b = 255$ đối với ảnh mức xám 8 bit.

3.4 Các loại bộ lọc

Trong xử lý ảnh, một bộ lọc (hay còn gọi là kernel, ma trận tích chập, hay mặt nạ) là một ma trận có kích thước nhỏ so với kích thước ảnh cần được lọc, được ứng dụng trong việc làm mờ ảnh, sắc nét ảnh, phát hiện cạnh... Việc lọc ảnh được thực hiện bằng cách lấy tích chập giữa một bộ lọc và một ảnh. Một cách đơn giản, điểm ảnh ngõ ra là một hàm của các điểm ảnh ngõ vào lân cận (bao gồm cả chính nó).

Gọi $x(i, j)$ là ảnh gốc, $y(i, j)$ là ảnh sau khi lọc. Bộ lọc h có kích thước $W_{\text{filter}} \times H_{\text{filter}}$. S_{ij} là tập con của ngõ vào chứa các điểm lân cận xung quanh (i, j) :

$$S_{ij} = \begin{bmatrix} x[i-m, j-n] & \cdots & x[i-m, j] & \cdots & x[i-m, j+n] \\ \vdots & & \vdots & & \vdots \\ x[i, j-n] & \cdots & x[i, j] & \cdots & x[i, j+n] \\ \vdots & & \vdots & & \vdots \\ x[i+m, j-n] & \cdots & x[i+m, j] & \cdots & x[i+m, j+n] \end{bmatrix}, \quad m = \frac{H_{\text{filter}}}{2}, n = \frac{W_{\text{filter}}}{2}$$

Ta có biểu diễn tường minh của một số bộ lọc như sau:

Bộ lọc trung vị:

$$y(i, j) = \text{median}_{(s,t) \in S_{ij}} \{x(s, t)\}$$

Bộ lọc max:

$$y(i, j) = \max_{(s,t) \in S_{xy}} \{x(s, t)\}$$

Bộ lọc min:

$$y(i, j) = \min_{(s,t) \in S_{xy}} \{x(s, t)\}$$

Bộ lọc trung điểm:

$$y(i, j) = \frac{1}{2} \left(\min_{(s,t) \in S_{xy}} [x(s, t)] + \max_{(s,t) \in S_{xy}} [x(s, t)] \right)$$

4 Công cụ yêu cầu cho bài thí nghiệm

4.1 Phần mềm và phần cứng

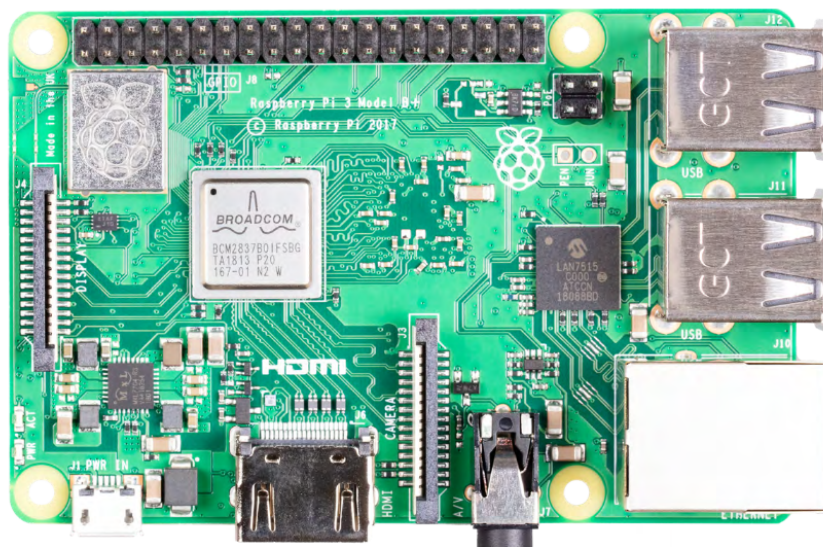
- Bộ kit Raspberry Pi 3
- Màn hình, chuột, bàn phím (tùy chọn)

Ta có thể không cần dùng màn hình, chuột, bàn phím bằng cách tiền cấu hình tên người dùng, tên miền máy chủ, kết nối mạng... trước khi cài đặt và đăng nhập và điều khiển Raspberry Pi từ xa thông qua giao thức SSH hoặc VNC.

4.2 Giới thiệu bộ kit Raspberry Pi 3

Các thông số kỹ thuật chính:

Vi xử lý:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Bộ nhớ:	1GB 2.4 GHz và 5 GHz WLAN
Kết nối:	Bluetooth 4.2 BLE Gigabit Ethernet over USB 2.0 4 cổng USB 2.0 1 x HDMI kích thước đầy đủ Cổng display MIPI DSI Cổng camera MIPI CSI Ngõ ra stereo 4 cực và cổng video composite H.264, MPEG-4 decode (1080p30)
Video và âm thanh:	
Multimedia:	H.264 encode (1080p30) OpenGL ES 1.1, 2.0 graphics
Hỗ trợ thẻ SD:	Thẻ Micro SD cho việc khởi động hệ điều hành và lưu trữ dữ liệu
Nguồn:	5V/2.5A DC qua kết nối micro USB 5V DC qua header GPIO
Nhiệt độ hoạt động:	0-50°C

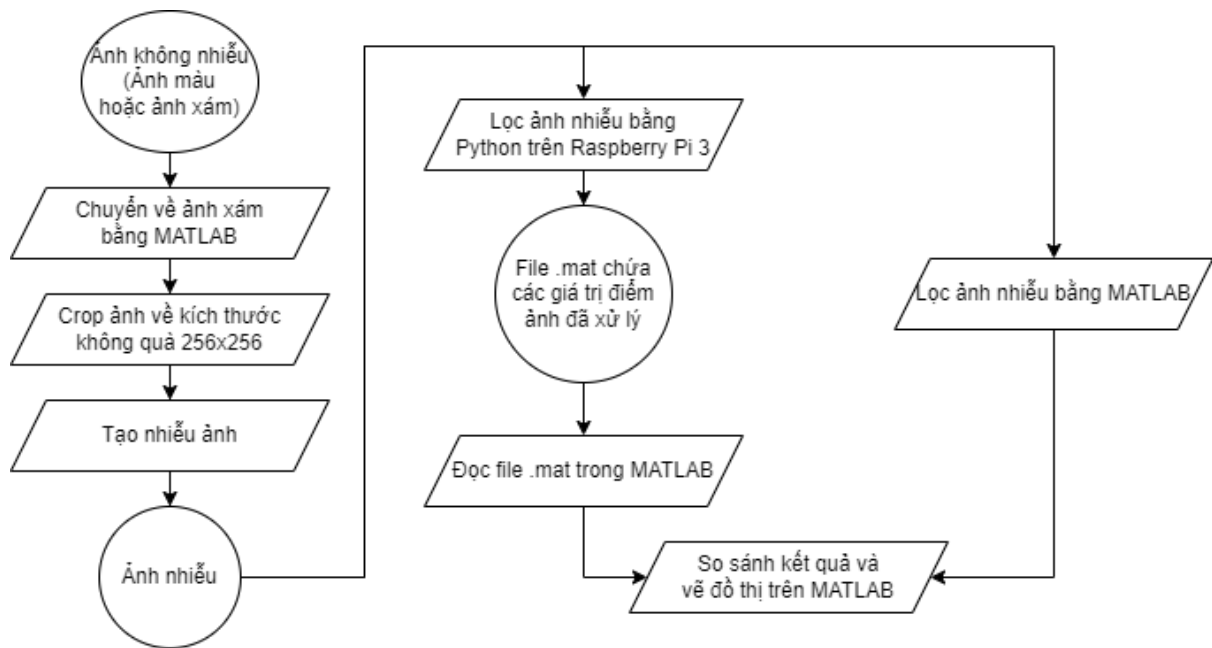


Hình 3: Bộ kit Raspberry Pi 3 Model B

5 Tiến hành thí nghiệm

Các bước thí nghiệm chung:

1. Thêm nhiễu cho ảnh bằng MATLAB
2. Xử lý nhiễu bằng MATLAB trên máy tính và xử lý nhiễu bằng Python trên Raspberry Pi 3
3. Lấy kết quả MATLAB làm chuẩn để đánh giá kết quả giảm nhiễu ảnh trên Raspberry Pi.



Hình 4: Quy trình thí nghiệm

Để đánh giá hiệu quả của bộ lọc, ta tính sai số bình phương trung bình (mean square error, MSE) giữa ảnh gốc và ảnh đã lọc:

$$MSE = \frac{1}{WH} \sum_{i=1}^H \sum_{j=1}^W (y_{ij} - x_{ij})^2 \quad (6)$$

Trong đó, W và H là chiều rộng và chiều cao ảnh, x là ảnh gốc và y là ảnh đã lọc.

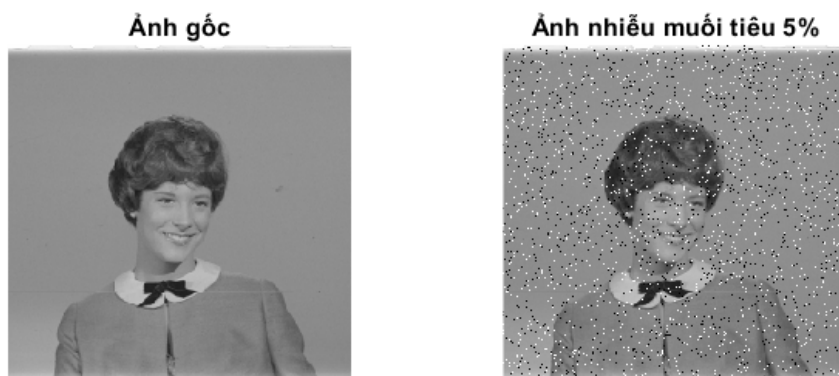
Ta tìm MSE bằng hàm `mse` có sẵn của MATLAB.

5.1 Lọc nhiễu ảnh muối tiêu

5.1.1 Tạo ảnh nhiễu muối tiêu trên MATLAB

Ta đọc một ảnh gốc (ảnh màu hoặc ảnh xám), chuyển sang ảnh xám nếu cần, crop ảnh xuống 256×256 để thuận tiện cho việc xử lý. Cuối cùng, ta tạo nhiễu muối tiêu cho ảnh bằng hàm `imnoise` có sẵn của MATLAB.

```
1 figure, tiledlayout(1,2)
2
3 % Đọc ảnh, chuyển sang ảnh xám, và crop ảnh nếu kích thước lớn hơn 255
4 I = imread('dataset\4.1.03.tiff');
5 I = rgb2gray(I);
6 [rows, cols] = size(I);
7 size = [rows, cols];
8 if (rows > 256), I = imresize(I, [256 NaN]), end;
9 if (cols > 256), I = imresize(I, [NaN 256]), end;
10
11 % Lưu ảnh xám đã crop và hiển thị ảnh
12 imwrite(I, '../images/salt_pepper_orig.bmp')
13 nexttile, imshow(I), title('Ảnh gốc')
14
15 % Thêm nhiễu muối tiêu với mật độ 0.05
16 J = imnoise(I, 'salt & pepper', 0.05);
17
18 % Lưu ảnh đã được tạo nhiễu và hiển thị ảnh
19 imwrite(J, '../images/salt_pepper_noise.bmp')
20 nexttile, imshow(J), title('Ảnh nhiễu muối tiêu 5%')
```

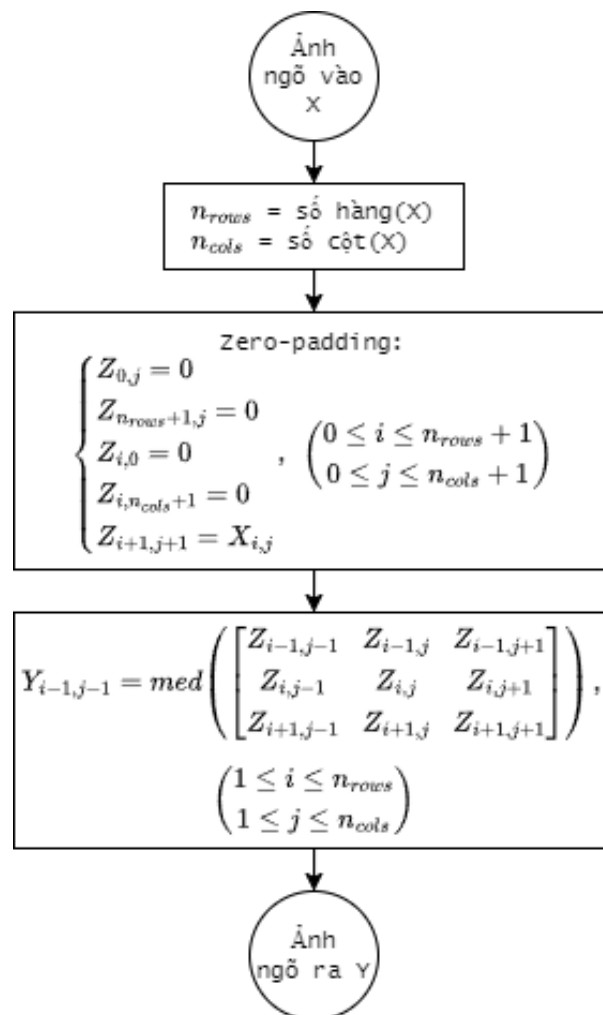


Hình 5: Tạo ảnh nhiễu muối tiêu trên MATLAB

5.1.2 Lọc ảnh nhiễu muối tiêu bằng Python trên Raspberry Pi 3

Các bước triển khai hàm lọc trung vị kích thước 3×3 trên Python như sau (Hình 6):

- Padding cho ngõ vào: Để không bỏ sót các điểm ảnh ở rìa, ta cần padding cho ảnh ngõ vào. Vì bộ lọc có kích thước 3×3 , ta cần padding 1 hàng zero cho rìa trên và dưới, và padding 1 cột zero cho rìa trái và phải.
- Tìm trung vị: Cho vòng lặp chạy qua từng điểm ảnh. Ở mỗi vòng lặp, xếp các điểm ảnh lân cận vào một mảng, sắp xếp mảng theo thứ tự tăng dần (hoặc giảm dần), và điểm ảnh ngõ ra sẽ là phần tử ở giữa của mảng đã sắp xếp.



Hình 6: Thuật toán lọc trung vị

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import scipy.io
5
```

```

6 # Doc anh nhieu muoi tieu
7 I = cv2.imread('output/salt_pepper_noise.png', cv2.IMREAD_GRAYSCALE)
8
9 # Ham loc trung vi kich thuoc 3x3
10 def medFilter3x3(X):
11     nRows, nCols = X.shape
12
13     # Padding cho anh ngo vao
14     hpad = np.zeros((1, nCols))
15     X = np.vstack((hpad, X, hpad))
16     vpad = np.zeros((nRows+2, 1))
17     X = np.hstack((vpad, X, vpad))
18
19     # Lay trung vi bang cach sap xep theo thu tu lon dan
20     # va chon phan tu thu 5 trong 9 phan tu
21     Y = np.zeros((nRows,nCols))
22     for i in range(1,nRows+1):
23         for j in range(1,nCols+1):
24             x = [X[i-1,j-1],
25                 X[i-1,j],
26                 X[i-1,j+1],
27                 X[i,j-1],
28                 X[i,j],
29                 X[i,j+1],
30                 X[i+1,j-1],
31                 X[i+1,j],
32                 X[i+1,j+1]]
33             x = sorted(x)
34             Y[i-1][j-1] = x[4]
35     return Y
36
37 # Loc trung vi voi bo loc 3x3
38 J = medFilter3x3(I)
39
40 # Luu ket qua theo dinh dang cua MATLAB
41 scipy.io.savemat('output/salt_pepper_denoised_rpi3.mat',
42                 {"rpiDenoisedImg": J})

```

Kết quả chạy chương trình trên Raspberry Pi 3: Chương trình đọc ảnh nhiễu ngõ vào và xuất file salt_pepper_denoised_rpi3.mat thành công để đưa vào MATLAB.

5.1.3 So sánh kết quả với MATLAB

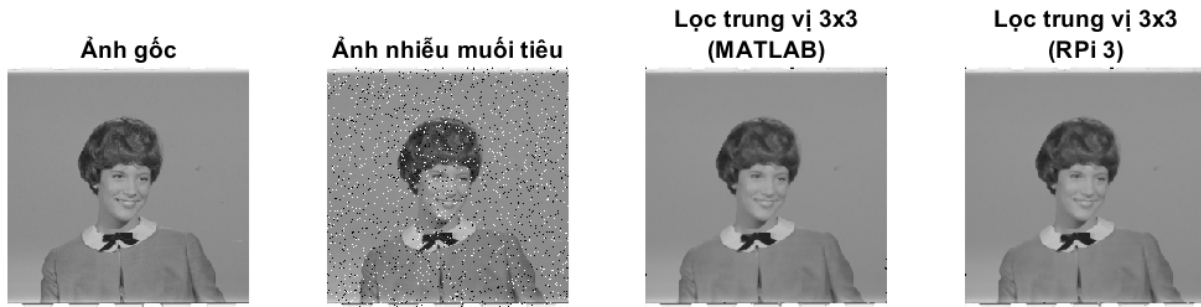
Ta lọc ảnh bằng bộ lọc trung vị kích thước 3×3 với hàm `medfilt2()` có sẵn của MATLAB. Tiếp đến, ta vẽ ảnh đã lọc từ MATLAB và từ Raspberry Pi 3, đồng thời đánh giá các kết quả này dựa trên sai số bình phương trung bình giữa ảnh đã lọc và ảnh gốc.

```
1 figure
2 tiledlayout(1,4)
3
4 % Doc anh xam goc va hien thi
5 I = imread('../output/salt_pepper_orig.png');
6 nexttile, imshow(I), title('Anh goc')
7
8 % Doc anh nhieu muoi tieu va hien thi
9 J = imread('../output/salt_pepper_noise.png');
10 nexttile, imshow(J), title('Anh nhieu muoi tieu')
11
12 % Loc trung vi voi bo loc 3x3 bang ham co san va hien thi
13 K = medfilt2(J, [3 3]);
14 nexttile, imshow(K), title(["Loc trung vi 3x3", "(MATLAB)"])
15 imwrite(K, '../output/salt_pepper_denoised_matlab.png')
16
17 % Doc ket qua loc trung vi 3x3 tu Raspberry Pi 3 va hien thi
18 load('../output/salt_pepper_denoised_rpi3.mat')
19 L = uint8(rpiDenoisedImg);
20 nexttile, imshow(L), title(["Loc trung vi 3x3", "(RPI 3)"])
21 imwrite(L, '../output/salt_pepper_denoised_rpi3.png')
22
23 % Sai so binh phuong trung binh giua anh goc va anh da loc
24 fprintf('MSE (MATLAB): %.4f\n', mse(I, K));
25 fprintf('MSE (RPI 3): %.4f\n', mse(I, L));
```

Kết quả chạy:

MSE (MATLAB): 3.8137

MSE (RPI 3): 3.8137



Hình 7: So sánh kết quả lọc trung vị từ Raspberry Pi 3 so với MATLAB

Nhận xét: Kết quả lọc trung vị từ Python trên Raspberry Pi hiệu quả tương đương MATLAB. Sai số bình phương trung bình giữa ảnh gốc và ảnh đã lọc trên Raspberry Pi 3 và trên MATLAB là giống nhau.

5.2 Lọc nhiễu ảnh Gauss

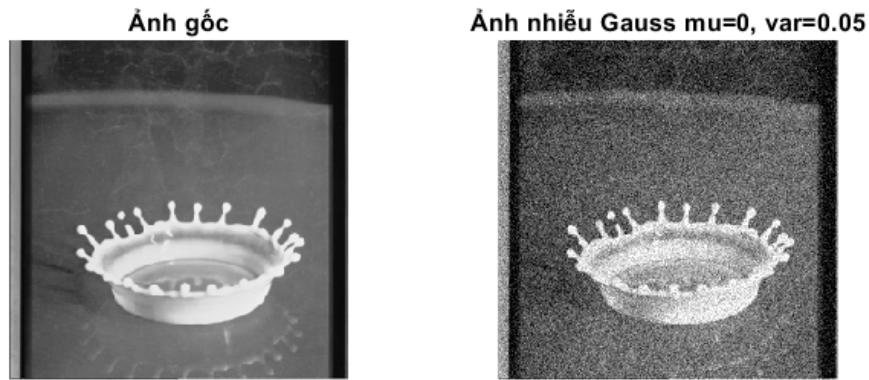
5.2.1 Tạo ảnh nhiễu Gauss

Ta đọc một ảnh gốc (ảnh màu hoặc ảnh xám), chuyển sang ảnh xám nếu cần, crop ảnh xuống 256×256 để thuận tiện cho việc xử lý. Cuối cùng, ta tạo nhiễu muối tiêu cho ảnh bằng hàm `imnoise` có sẵn của MATLAB.

```

1 figure
2 tiledlayout(1,2)
3
4 % Đọc ảnh, chuyển sang ảnh xám, và crop ảnh nếu kích thước lớn hơn 255
5 I = imread('dataset\4.2.01.tiff');
6 I = rgb2gray(I);
7 [rows, cols] = size(I);
8 size = [rows, cols];
9 if (rows > 256), I = imresize(I, [256 NaN]); end;
10 if (cols > 256), I = imresize(I, [NaN 256]); end;
11
12 % Lưu ảnh xám đã crop và hiển thị ảnh
13 imwrite(I, '../images/gaussian_orig.bmp');
14 nexttile, imshow(I), title('Ảnh gốc');
15
16 % Thêm nhiễu Gauss với trung bình 0, phương sai 0.05
17 J = imnoise(I, 'gaussian');
18
19 % Lưu ảnh đã được tạo nhiễu và hiển thị ảnh
20 imwrite(J, '../images/gaussian_noise.bmp');
21 nexttile, imshow(J), title('Ảnh nhiễu Gauss mu=0, var=0.05');

```

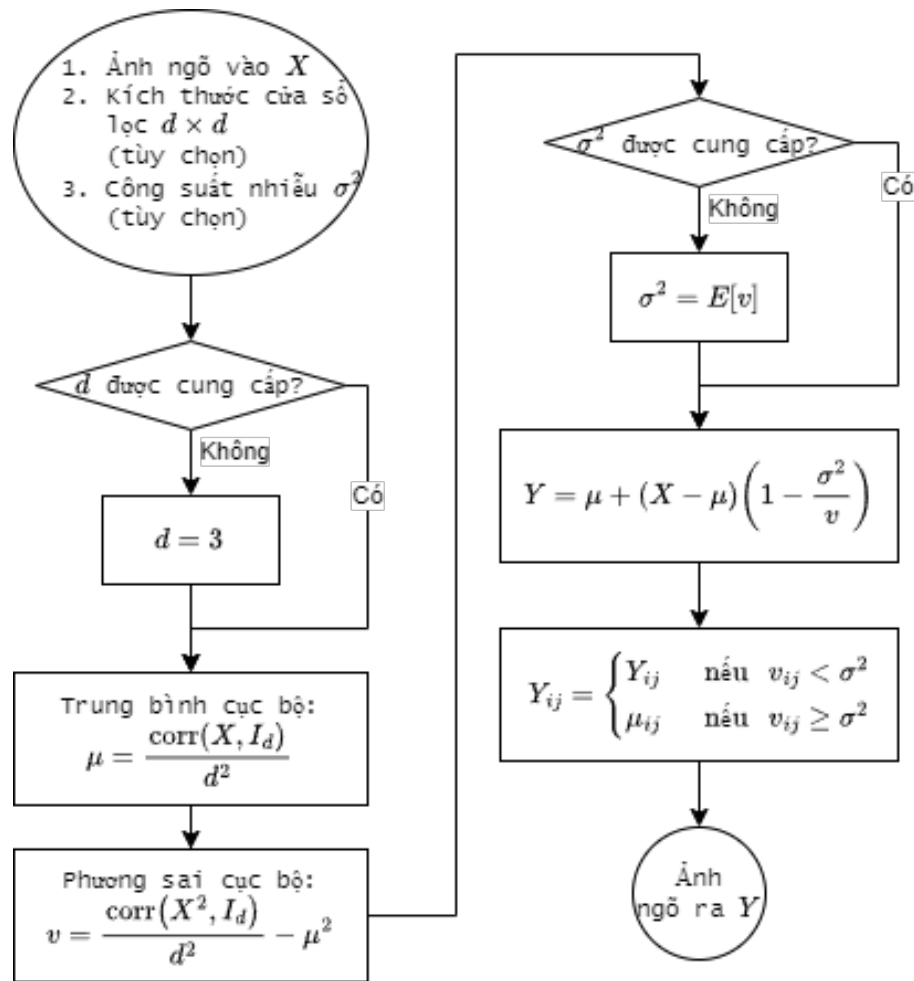



Hình 8: Ảnh nhiễu Gauss tạo từ MATLAB

Nếu không cung cấp thêm tham số, mặc định MATLAB sẽ tạo nhiễu Gauss trắng với trung bình $\mu = 0$ và phương sai hay công suất nhiễu $\sigma^2 = 0.01$.

5.2.2 Lọc ảnh nhiễu Gauss bằng Python trên Raspberry Pi 3

Với Python trên Raspberry Pi 3, ta lọc ảnh bằng bộ lọc Wiener với hàm `wiener()` của thư viện `scipy.signal`. Wiener là bộ lọc thông thấp cho một hình ảnh có giá trị cường độ bị suy hao bởi nhiễu cộng có công suất hằng. Với mỗi điểm ảnh, bộ lọc sẽ tìm trung bình và độ lệch chuẩn cục bộ trong một lân cận có kích thước được định sẵn. Thuật toán của hàm `wiener()` được mô tả ở hình 9.



Hình 9: Thuật toán của hàm `wiener()`

Nếu không cung cấp thêm tham số kích thước bộ lọc và công suất nhiễu, mặc định hàm `scipy.signal.wiener()` sẽ:

- Chọn kích thước của số lọc là 3×3 ;
- Ước lượng công suất nhiễu.

Tuy nhiên, cho tới phiên bản Scipy 1.13, hàm `wiener()` không trả về giá trị được ước lượng của công suất. Do đó ta tiến hành chỉnh sửa lại mã nguồn của hàm này như sau:

```

1 from scipy.signal import correlate
2
3 def wiener(im, mysize=None, noise=None):
4     im = np.asarray(im)
5     if mysize is None:
6         mysize = [3] * im.ndim
7     mysize = np.asarray(mysize)
8     if mysize.shape == ():
9         mysize = np.repeat(mysize.item(), im.ndim)

```

```

10
11 # Uoc luong trung binh cuc bo
12 lMean = correlate(im, np.ones(mysize), 'same') / np.prod(mysize,
13                    axis=0)
14
15 # Uoc luong phuong sai cuc bo
16 lVar = (correlate(im ** 2, np.ones(mysize), 'same') /
17         np.prod(mysize, axis=0) - lMean ** 2)
18
19 # Uoc luong cong suat nhieu neu can
20 if noise is None:
21     noise = np.mean(np.ravel(lVar), axis=0)
22
23 res = (im - lMean)
24 res *= (1 - noise / lVar)
25 res += lMean
26 out = np.where(lVar < noise, lMean, res)
27
28 return [out, noise]

```

Ta tiến hành thí nghiệm với hai trường hợp:

1. (Mặc định) Không cung cấp tham số công suất nhiễu để phần mềm tự ước lượng giá trị này.
2. Cung cấp trước tham số nhiễu ($\sigma^2 = 0.01$).

```

1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import scipy.io
5
6 # Doc anh nhieu Gauss
7 I = cv2.imread('output/gaussian_noise.png', cv2.IMREAD_GRAYSCALE)
8
9 # Loc nhieu Gauss bang ham wiener() va luu ket qua
10 # Truong hop mac dinh (tu uoc luong cong suat nhieu)
11 [J_default, J_default_noise] = wiener(I, (3,3))
12 # Cung cap tham so cong suat nhieu var=0.01
13 [J_var0p01, J_var0p01_noise] = wiener(I, (3,3), 0.01)
14
15 # Luu ket qua loc theo dinh dang cua MATLAB
16 scipy.io.savemat('output/gaussian_denoised_rpi3.mat',
17                  {"J_default": J_default,
18                   "J_default_noise": J_default_noise,

```

Kết quả chạy chương trình trên Raspberry Pi 3: Chương trình đọc ảnh nhiễu ngẫu nhiên vào và xuất file `output/gaussian_denoised_rpi3.mat` thành công để đưa vào MATLAB.

5.2.3 So sánh kết quả với MATLAB

Ta lọc ảnh bằng bộ lọc Wiener với hàm `wiener2()` có sẵn của MATLAB. Nếu không cung cấp thêm tham số kích thước bộ lọc và công suất nhiễu, mặc định MATLAB sẽ dùng cửa sổ lọc có kích thước 3×3 , và ước lượng công suất nhiễu trước khi thực hiện quá trình lọc.

Tiếp đến, ta vẽ ảnh đã lọc từ MATLAB và từ Raspberry Pi 3, đồng thời đánh giá các kết quả này dựa trên sai số bình phương trung bình giữa ảnh đã lọc và ảnh gốc.

```

1 fig = figure("Position", [0 0 800 500]);
2 tiledlayout(2,3)
3
4 % Doc anh xam goc va hien thi
5 I = imread('../output/gaussian_orig.png');
6 nexttile, imshow(I), title('Anh goc')
7
8 % Doc anh nhieu Gauss va hien thi
9 J = imread('../output/gaussian_noise.png');
10 nexttile, imshow(J), title(["Anh nhieu Gauss", "mu=0, var=0.05"])
11
12 % Loc nhieu Gauss bang ham wiener2 co san va hien thi
13 [K, noise] = wiener2(J);
14 nexttile, imshow(K), title(["Loc nhieu Gauss", "(MATLAB)"])
15 imwrite(K, '../output/gaussian_denoised_matlab.png')
16
17 % Doc ket qua loc nhieu Gauss tu Raspberry Pi 3 va hien thi
18 load('../output/gaussian_denoised_rpi3.mat')
19 L1 = uint8(J1_rpi3);
20 L2 = uint8(J2_rpi3);
21 nexttile, imshow(L1), title(["Loc nhieu Gauss", "(RPI 3, mac dinh)"])
22 imwrite(L1, '../output/gaussian_denoised_rpi3_default.png')
23 nexttile, imshow(L2), title(["Loc nhieu Gauss", "(RPI 3, var=0.01)"])
24 imwrite(L2, '../output/gaussian_denoised_rpi3_var_0p01.png')
25
26 % Cong suat nhieu uoc luong va sai so trung binh binh phuong
27 fprintf("Cong suat nhieu uoc luong (truong hop khong cung cap truoc tham
    so nhieu):\n");
28 fprintf('MATLAB: %.4f\n', K_noise);

```

```

29 fprintf('Raspberry Pi 3: %.4f\n', J_default_noise);
30 fprintf('\n');
31
32 fprintf('Sai so trung binh binh phuong giua anh goc va anh da loc:\n');
33 fprintf('MSE (MATLAB): %.4f\n', mse(I, K));
34 fprintf('MSE (RPI 3, default): %.4f\n', mse(I, L1));
35 fprintf('MSE (RPI 3, var=0.01): %.4f\n', mse(I, L2));

```

Cong suat nhieu uoc luong (truong hop khong cung cap tham so nhieu)

MATLAB: 0.0105

Raspberry Pi 3: -13035.2815

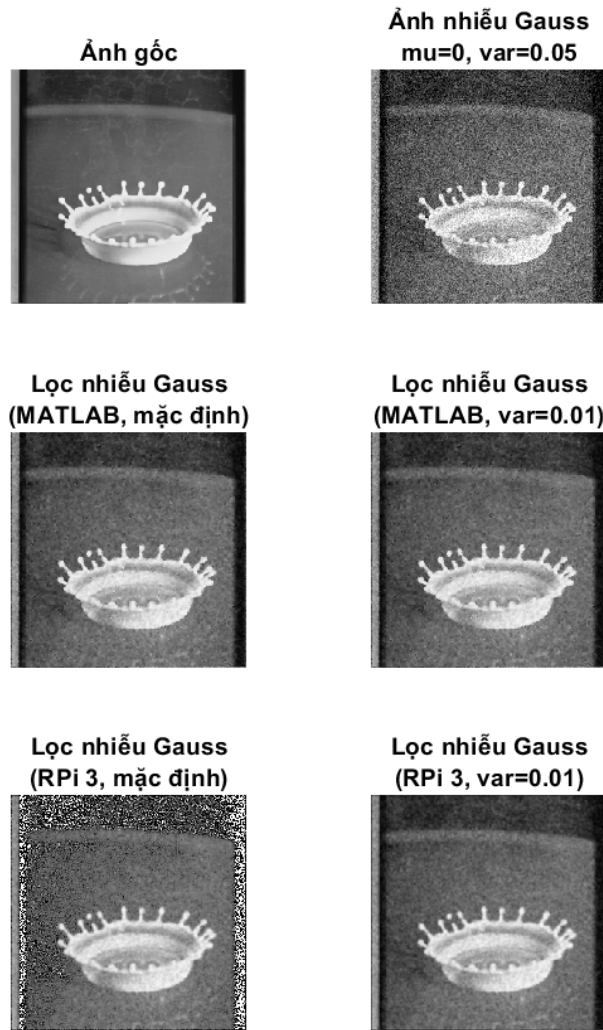
Sai so binh phuong trung binh giua anh goc va anh da loc:

MSE (MATLAB, mac dinh): 41.1119

MSE (MATLAB, var=0.01): 41.6963

MSE (RPI 3, mac dinh): 66.0795

MSE (RPI 3, var=0.01): 39.0874



Hình 10: So sánh kết quả lọc nhiễu Gauss từ Raspberry Pi 3 so với MATLAB

Nhận xét:

- **Trường hợp tự ước lượng công suất nhiễu bằng phần mềm:** MATLAB ước lượng tốt công suất nhiễu (0.0105 so với 0.010), do đó cho kết quả lọc tương đối tốt. Trong khi đó, thư viện scipy của Python trên Raspberry Pi 3 ước lượng công suất nhiễu chệch rất nhiều so với công suất nhiễu thực tế (-13035.2815 so với 0.01), do đó cho kết quả lọc tệ hơn. Điều này thể hiện ở MSE của MATLAB nhỏ hơn đáng kể ($41.1119 < 66.0795$).
- **Trường hợp cung cấp trước tham số công suất nhiễu:** Khi cung cấp trước tham số công suất nhiễu $\sigma^2 = 0.01$, bộ kit lại cho kết quả lọc tốt hơn MATLAB. Cụ thể, sai số trung bình bình phương từ kết quả lọc của Raspberry Pi 3 nhỏ hơn so với của MATLAB ($39.0874 < 41.6963$).

6 Kết luận

6.1 Tóm tắt những phát hiện chính

6.2 Mục tiêu đã đạt được

6.3 Tính hiệu quả của từng phương pháp

6.4 Hướng phát triển của đề tài

Tài liệu tham khảo

- [1] Subramania Jayaraman, S. Esakkirajan, and T. Veerakumar. *Digital image processing*. Vol. 7014. Tata McGraw Hill Education New Delhi, 2009.