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 NHIỄU 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

Lĩnh vực hình ảnh kỹ thuật số luôn phải đối mặt với một thách thức dai dẳng: nhiễu 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 $(N - M)$:

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

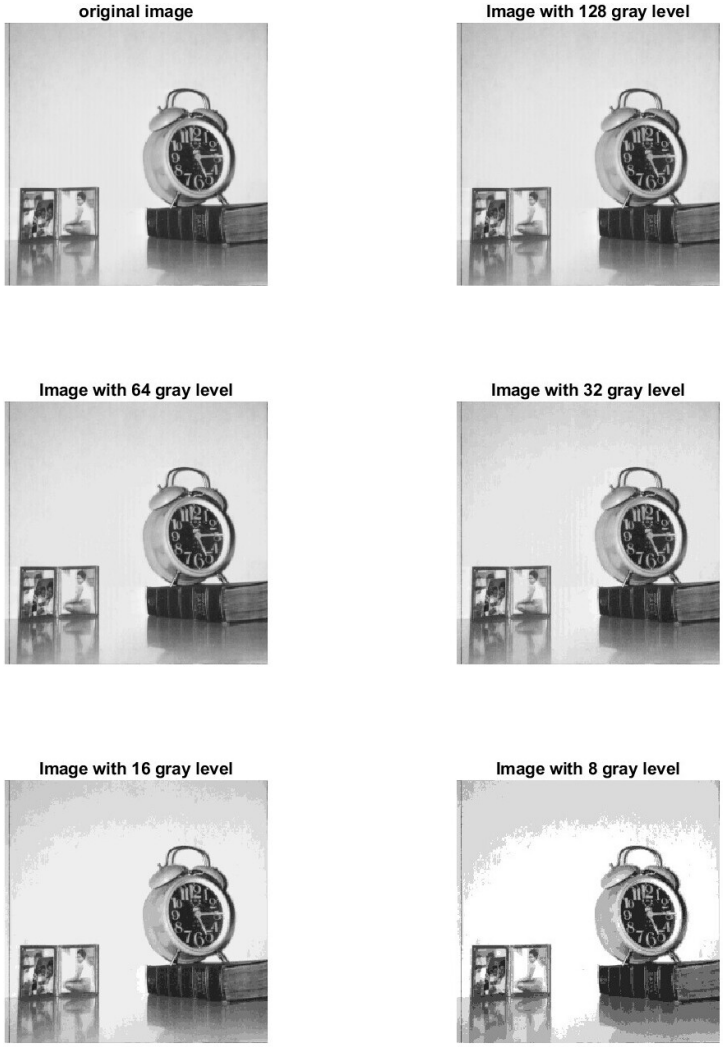
Ma trận A được gọi là ảnh số, mỗi thành phần của A được gọi là một điểm ảnh, hay pixel.

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:

$$w = \min(w_1, w_2) \tag{1}$$
$$h = \min(h_1, h_2) \tag{2}$$

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:

$$y(i,j) = \min\{x_1(i,j) + x_2(i,j), 2^N\} \tag{3}$$
$$y(i,j) = \|x_1(i,j) - x_2(i,j)\| \tag{4}$$

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:

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

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 mợt 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 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

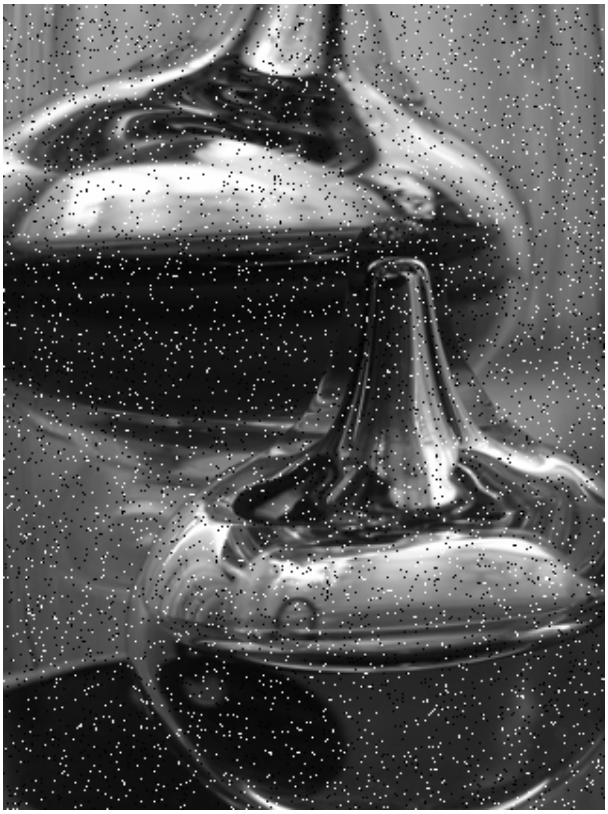
$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\bar{z})^2/2\sigma^2}$$

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

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.

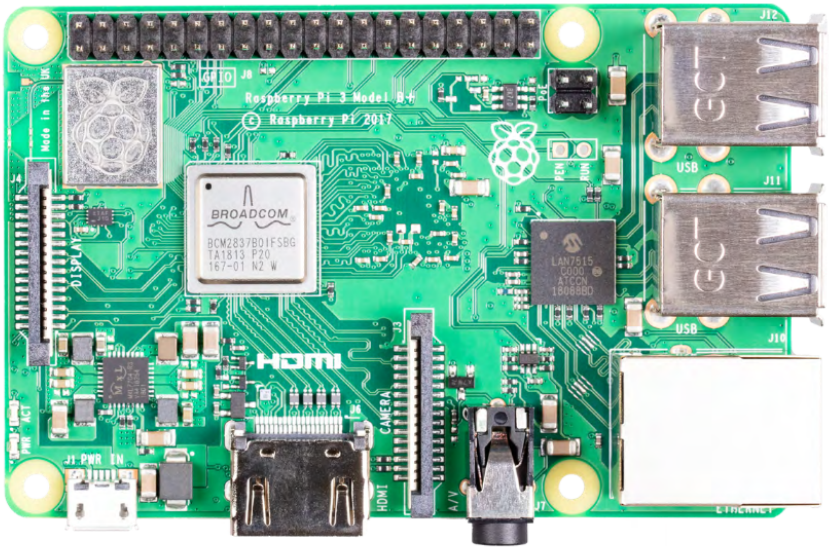
Phần mềm: Các phiên bản phần mềm và thư viện trong đề tài này là:

- MATLAB R2023a trên máy tính
- Python 3.12 trên Raspberry Pi 3, với các module chính:
  - numpy 1.26.4
  - matplotlib 3.8.0
  - opencv-python 4.9.0.80
  - scipy 1.13.0

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
Kết nối:	2.4 GHz và 5 GHz WLAN
	Bluetooth 4.2 BLE
	Gigabit Ethernet over USB 2.0
	4 cổng USB 2.0
Video và âm thanh:	1 x HDMI kích thước đầy đủ
	Cổng display MIPI DSI
	Cổng camera MIPI CSI
	Ngõ ra stero 4 cực và cổng video composite
Multimedia:	H.264, MPEG-4 decode (1080p30)
	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

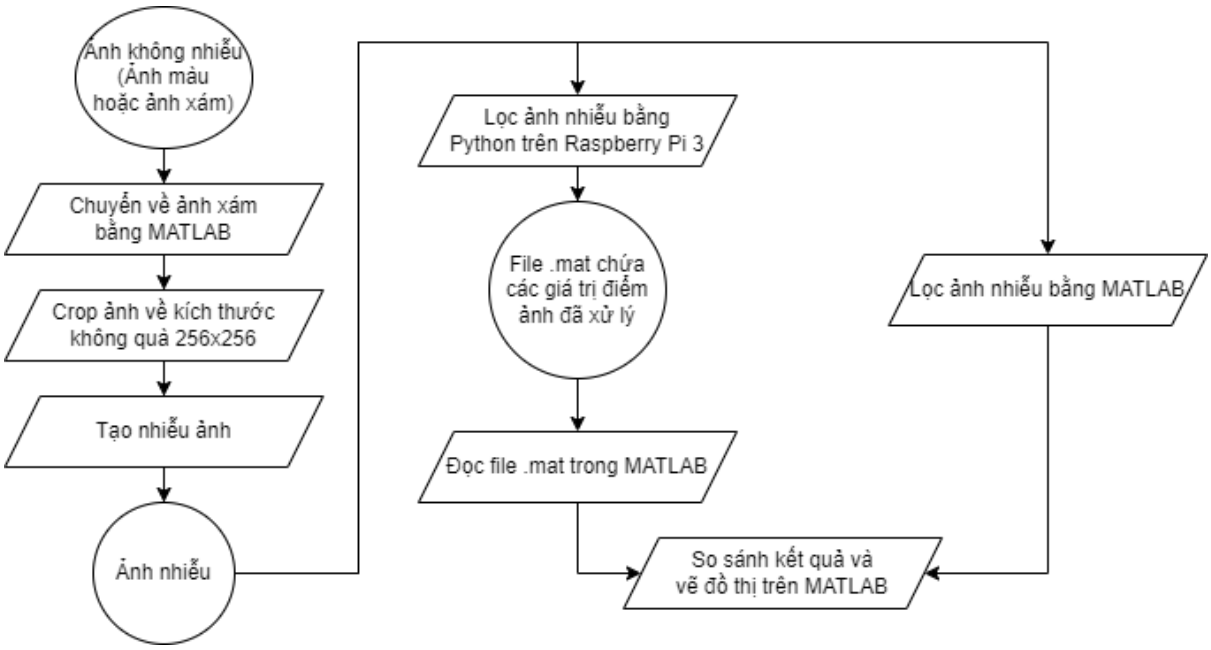


Hình 3: Bộ kit Raspberry Pi 3 Model B

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 xứ 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:

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

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.

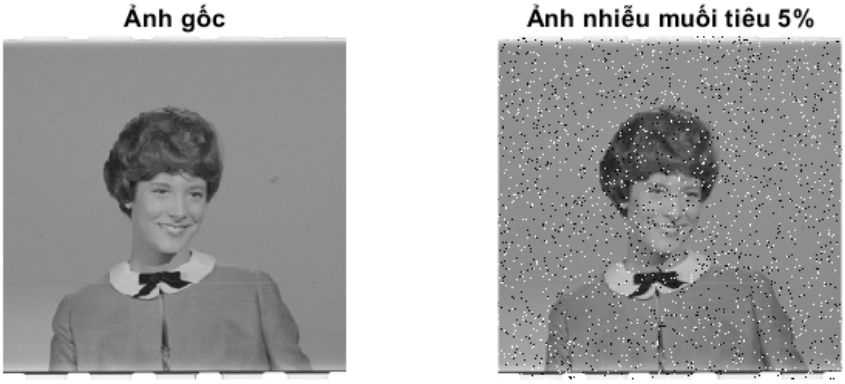
5.1 Lọc nhiễu ảnh muối tiêu

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.

```
figure, tiledlayout(1,2)

% Doc anh, chuyen sang anh xam, va crop anh neu kich thuoc lon hon 255
I = imread('dataset\4.1.03.tiff');
I = rgb2gray(I);
[rows, cols] = size(I);
size = [rows, cols];
```

```
if (rows > 256), I = imresize(I, [256 NaN]), end;
if (cols > 256), I = imresize(I, [NaN 256]), end;

% Luu anh xam da crop va hien thi anh
imwrite(I, '../images/salt_pepper_orig.bmp')
nexttile, imshow(I), title('Anh goc')

% Them nhieu muoi tieu voi mat do 0.05
J = imnoise(I, 'salt & pepper', 0.05);

% Luu anh da duoc tao nhieu va hien thi anh
imwrite(J, '../images/salt_pepper_noise.bmp')
nexttile, imshow(J), title('Anh nhieu muoi tieu 5%')
```

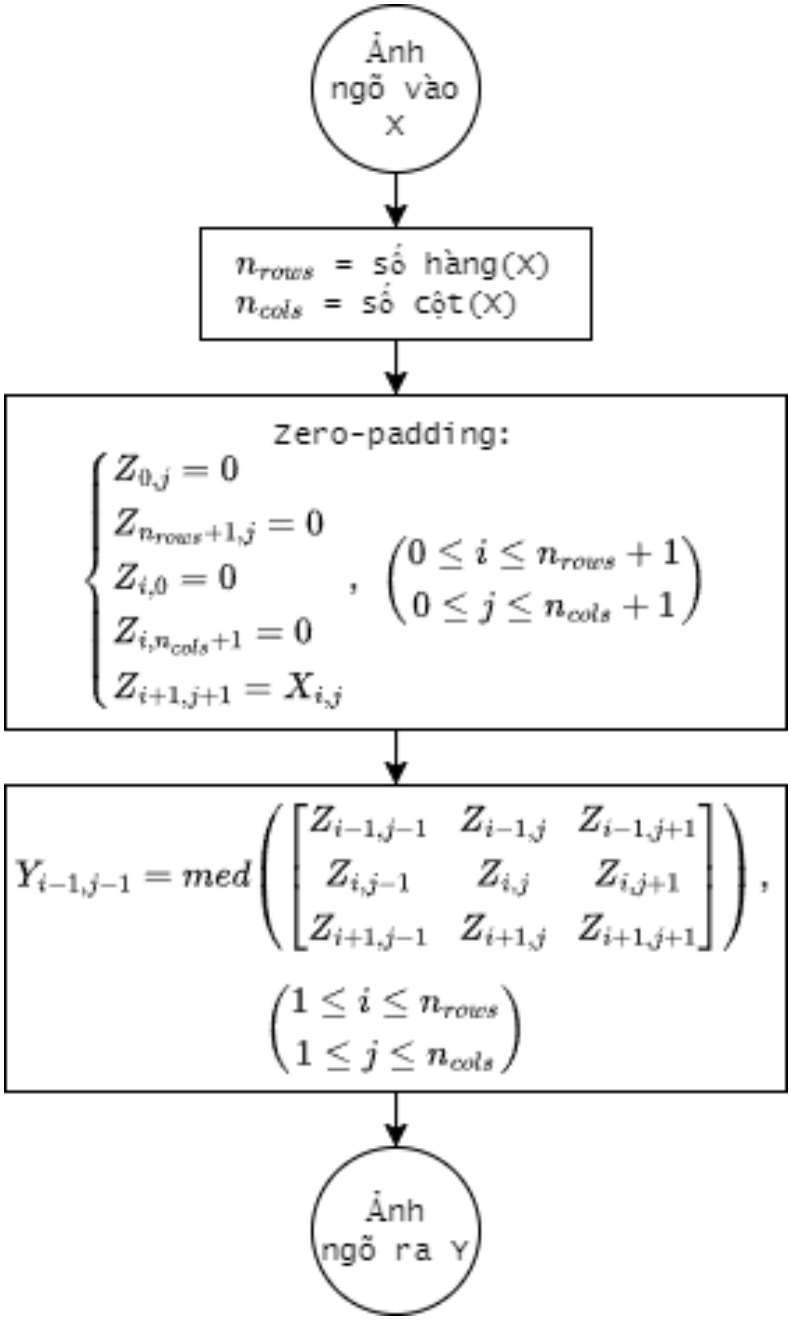


Hình 5: Tạo ảnh nhiễu muối tiêu trên MATLAB

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 3x3, ta cần padding một hàng zero cho rìa trên và dưới, và padding một 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

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import scipy.io

# Doc anh nhieu muoi tieu
I = cv2.imread('output/salt_pepper_noise.png', cv2.IMREAD_GRAYSCALE)

# Ham loc trung vi kich thuoc 3x3
def medFilter3x3(X):
```

```
  nRows, nCols = X.shape

  # Padding cho anh ngo vao
  hpad = np.zeros((1, nCols))
  X = np.vstack((hpad, X, hpad))
  vpad = np.zeros((nRows+2, 1))
  X = np.hstack((vpad, X, vpad))

  # Lay trung vi bang cach sap xep theo thu tu lon dan
  # va chon phan tu thu 5 trong 9 phan tu
  Y = np.zeros((nRows,nCols))
  for i in range(1,nRows+1):
    for j in range(1,nCols+1):
      x = [X[i-1,j-1],
           X[i-1,j],
           X[i-1,j+1],
           X[i,j-1],
           X[i,j],
           X[i,j+1],
           X[i+1,j-1],
           X[i+1,j],
           X[i+1,j+1]]
      x = sorted(x)
      Y[i-1][j-1] = x[4]
  return Y

# Loc trung vi voi bo loc 3x3
J = medFilter3x3(I)

# Luu ket qua theo dinh dang cua MATLAB
scipy.io.savemat('output/salt_pepper_denoised_rpi3.mat',
   {"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

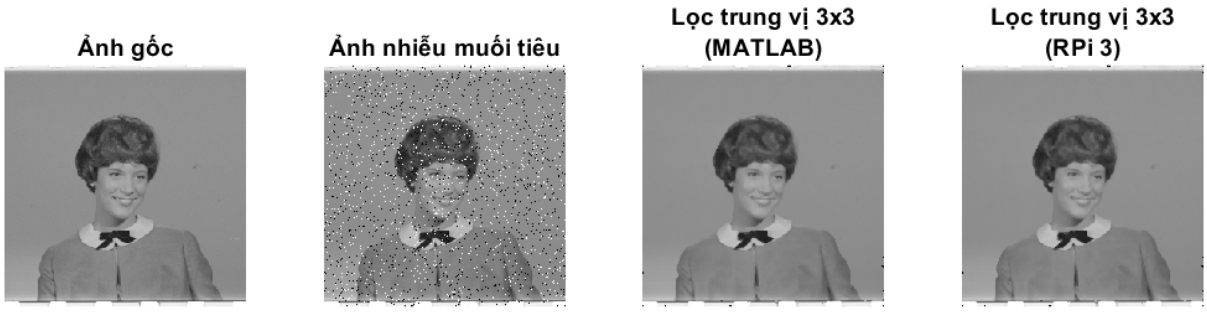
Ta lọc ảnh bằng bộ lọc trung vị kích thước 3×3 với hàm `medfilt2()` có sẵn của MATLAB. Tiếp đến, ta vẽ ảnh đã lọc từ MATLAB và từ Raspberry Pi 3, đồng thời đánh giá các kết quả này dựa trên sai số bình phương trunb bình giữa ảnh đã lọc và ảnh gốc.

```
figure
```

```
tiledlayout(1,4)

% Doc anh xam goc va hien thi
I = imread('../output/salt_pepper_orig.png');
nexttile, imshow(I), title('Anh goc')

% Doc anh nhieu muoi tieu va hien thi
J = imread('../output/salt_pepper_noise.png');
nexttile, imshow(J), title('Anh nhieu muoi tieu')

% Loc trung vi voi bo loc 3x3 bang ham co san va hien thi
K = medfilt2(J, [3 3]);
nexttile, imshow(K), title(["Loc trung vi 3x3", "(MATLAB)"])
imwrite(K, '../output/salt_pepper_denoised_matlab.png')

% Doc ket qua loc trung vi 3x3 tu Raspberry Pi 3 va hien thi
load('../output/salt_pepper_denoised_rpi3.mat')
L = uint8(rpiDenoisedImg);
nexttile, imshow(L), title(["Loc trung vi 3x3", "(RPi 3)"])
imwrite(L, '../output/salt_pepper_denoised_rpi3.png')

% Sai so binh phuong trung binh giua anh goc va anh da loc
fprintf('MSE (MATLAB): %.4f\n', mse(I, K));
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.

5.1.4 So sánh trên toàn bộ dữ liệu

Thực hiện cùng quy trình thí nghiệm này với các ảnh còn lại trong bộ dữ liệu, ta thu được kết quả sau:

Filename	MATLAB	RPi 3
4.1.01	9.2762	9.2762
4.1.02	9.8379	9.8379
4.1.03	3.8523	3.8523
4.1.04	5.6225	5.6225
4.1.05	9.2456	9.2456
4.1.06	24.7663	24.7663
4.1.07	3.1220	3.1220
4.1.08	3.9775	3.9775
4.2.01	4.3840	4.3840
4.2.03	40.0398	40.0398
4.2.05	11.7791	11.7791
4.2.06	22.3186	22.3186
4.2.07	7.6638	7.6638
5.1.09	17.4711	17.4711
5.1.10	33.3118	33.3118
5.1.11	2.9275	2.9275
5.1.12	8.4205	8.4205
5.1.13	2.3157	2.3157
5.1.14	24.9310	24.9310

Filename	MATLAB	RPi 3
5.2.08	15.6646	15.6646
5.2.09	27.0515	27.0515
5.2.10	33.6905	33.6905
5.3.01	21.9814	21.9814
5.3.02	26.8679	26.8679
7.1.01	10.5943	10.5943
7.1.02	2.5768	2.5768
7.1.03	9.7528	9.7528
7.1.04	9.4150	9.4150
7.1.05	20.9028	20.9028
7.1.06	20.9393	20.9393
7.1.07	16.7227	16.7227
7.1.08	5.2790	5.2790
7.1.09	17.2815	17.2815
7.1.10	10.4176	10.4176
7.2.01	4.5764	4.5764
boat.512	16.9074	16.9074
gray21.512	0.2504	0.2504
house	19.9191	19.9191
ruler.512	23.8984	23.8984

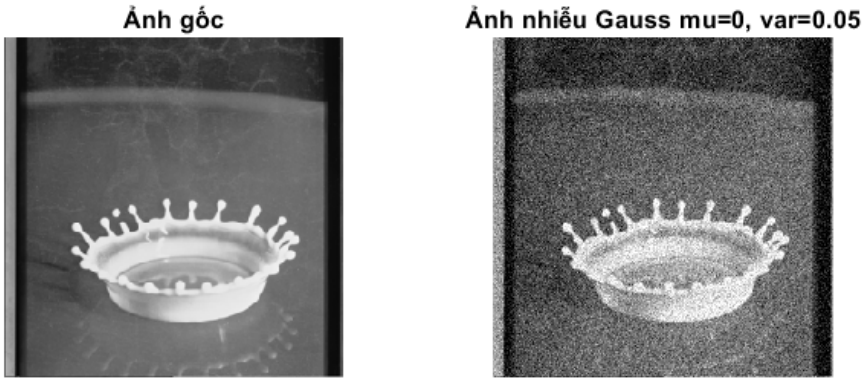
Nhận xét: Vì thuật toán lọc trung vị tương đối dễ triển khai, kết quả lọc từ MATLAB và từ Python trên Raspberry Pi 3 là giống nhau.

5.2 Lọc nhiễu ảnh Gauss

5.2.1 Tạo ảnh nhiễu Gauss 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.

```
figure
tiledlayout(1,2)

% Doc anh, chuyen sang anh xam, va crop anh neu kich thuoc lon hon 255
I = imread('dataset\4.2.01.tiff');
I = rgb2gray(I);
[rows, cols] = size(I);
size = [rows, cols];
if (rows > 256), I = imresize(I, [256 NaN]); end;
if (cols > 256), I = imresize(I, [NaN 256]); end;

% Luu anh xam da crop va hien thi anh
imwrite(I, '../images/gaussian_orig.bmp');
nexttile, imshow(I), title('Anh goc');

% Them nhieu Gauss voi trung binh 0, phuong sai 0.05
J = imnoise(I, 'gaussian');

% Luu anh da duoc tao nhieu va hien thi anh
imwrite(J, '../images/gaussian_noise.bmp');
nexttile, imshow(J), title('Anh nhieu Gauss mu=0, var=0.05');
```



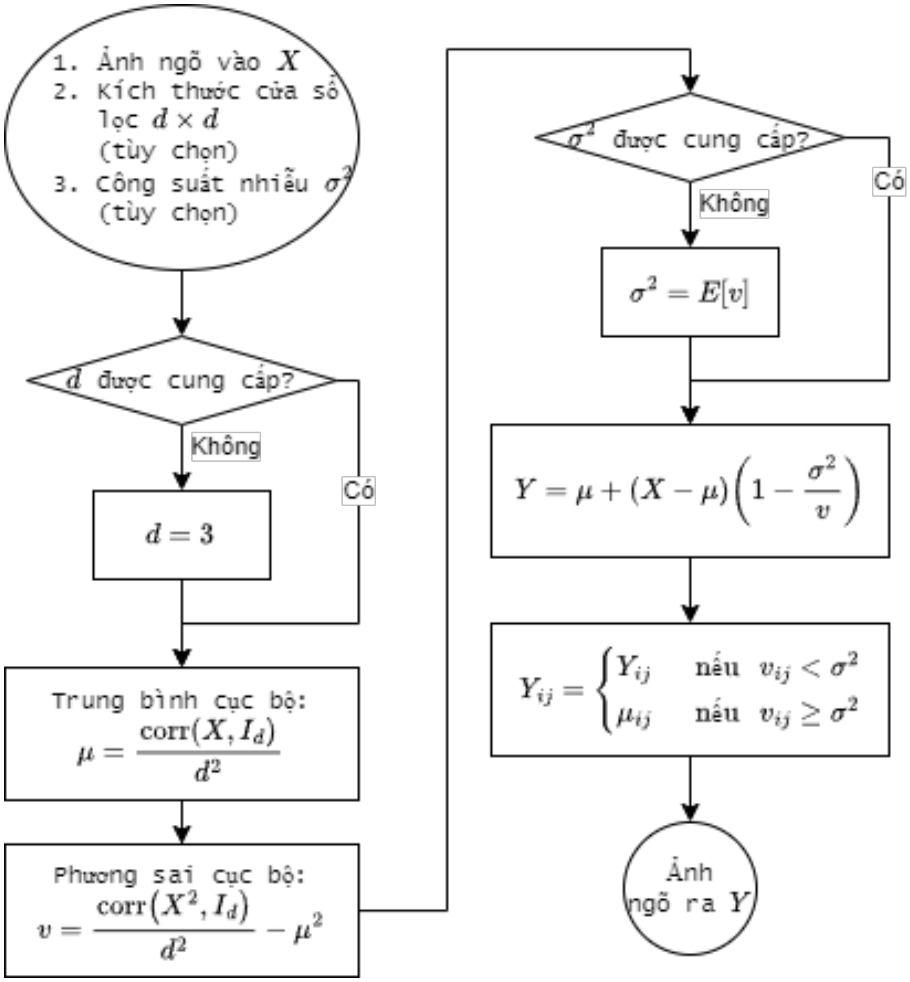
Hình 8: Ảnh nhiễu Gauss tạo từ MATLAB

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 cửa sổ 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:

```
from scipy.signal import correlate

def wiener(im, mysize=None, noise=None):
    im = np.asarray(im)
    if mysize is None:
        mysize = [3] * im.ndim
    mysize = np.asarray(mysize)
    if mysize.shape == ():
        mysize = np.repeat(mysize.item(), im.ndim)

    # Uoc luong trung binh cuc bo
    lMean = correlate(im, np.ones(mysize), 'same') / np.prod(mysize,
        axis=0)

    # Uoc luong phuong sai cuc bo
    lVar = (correlate(im ** 2, np.ones(mysize), 'same') /
            np.prod(mysize, axis=0) - lMean ** 2)

    # Uoc luong cong suat nhieu neu can
    if noise is None:
        noise = np.mean(np.ravel(lVar), axis=0)

    res = (im - lMean)
    res *= (1 - noise / lVar)
    res += lMean
    out = np.where(lVar < noise, lMean, res)

    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 ($\sigma^2 = 0.01$).

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import scipy.io

# Doc anh nhieu Gauss
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,
19                   "J_var0p01": J_var0p01})
```

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

5.2.3 So sánh kết quả với MATLAB

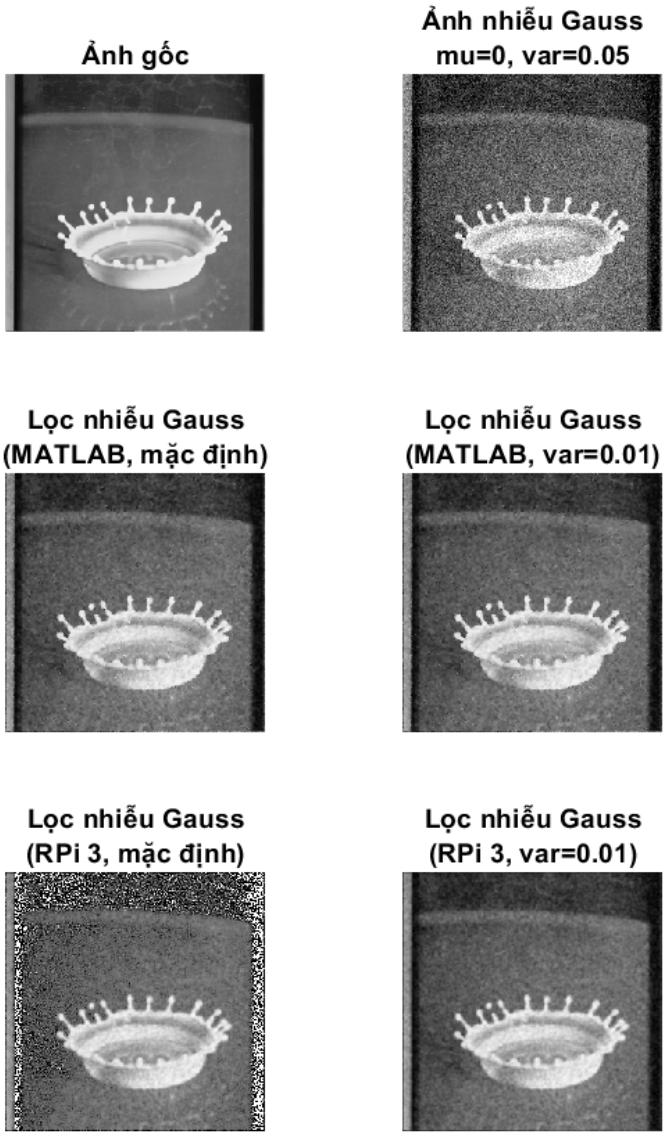
Ta lọc ảnh bằng bộ lọc Wiener với hàm `wiener2()` có sẵn của MATLAB. Nếu không cung cấp thêm tham số kích thước bộ lọc và công suất nhiễu, mặc định MATLAB sẽ dùng cửa sổ lọc có kích thước 3×3, và ước lượng công suất nhiễu trước khi thực hiện quá trình lọc.

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')
```

```
L1 = uint8(J1_rpi3);
L2 = uint8(J2_rpi3);
nexttile, imshow(L1), title(["Loc nhieu Gauss", "(RPi 3, mac dinh)"])
imwrite(L1, '../output/gaussian_denoised_rpi3_default.png')
nexttile, imshow(L2), title(["Loc nhieu Gauss", "(RPi 3, var=0.01)"])
imwrite(L2, '../output/gaussian_denoised_rpi3_var_0p01.png')

% Cong suat nhieu uoc luong va sai so trung binh binh phuong
fprintf("Cong suat nhieu uoc luong (truong hop khong cung cap truoc tham
    so nhieu):\n");
fprintf('MATLAB: %.4f\n', K_noise);
fprintf('Raspberry Pi 3: %.4f\n', J_default_noise);
fprintf('\n');

fprintf('Sai so trung binh binh phuong giua anh goc va anh da loc:\n');
fprintf('MSE (MATLAB): %.4f\n', mse(I, K));
fprintf('MSE (RPi 3, default): %.4f\n', mse(I, L1));
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 ($39.0874 < 41.6963$).

5.2.4 So sánh trên toàn bộ dữ liệu

Thực hiện cùng quy trình thí nghiệm này với các ảnh còn lại trong bộ dữ liệu, ta thu được kết quả sau:

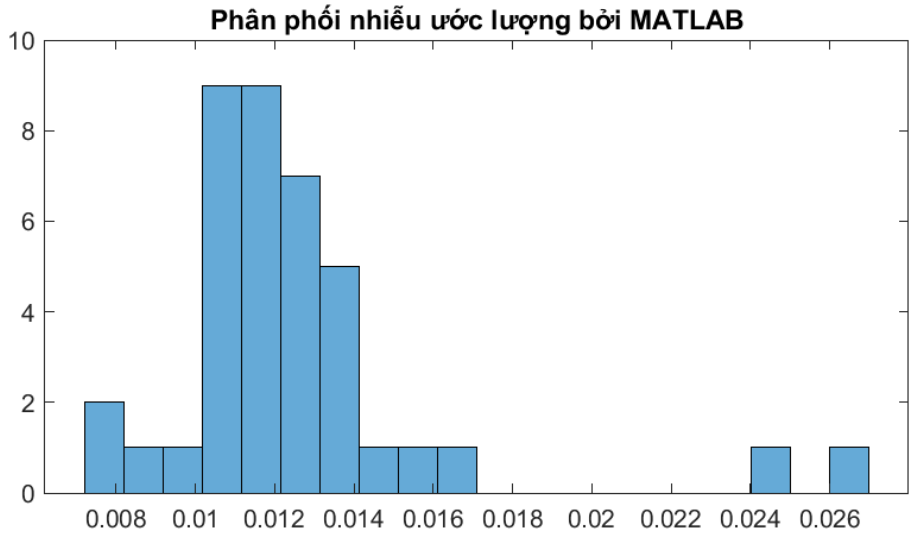
Filename	**Mặc định**				**Noise = 0.01**	
	MATLAB		**RPi3**		**MATLAB**	**RPi3**
	Noise	**MSE**	**Noise**	**MSE**	**MSE**	**MSE**
4.1.01	0.0093	35.2312	-4949.0979	73.4121	34.3646	32.5127
4.1.02	0.0078	26.6458	-2057.6920	51.8060	24.8055	24.9675
4.1.03	0.0120	40.9004	-19870.1450	47.5380	43.2635	41.0366
4.1.04	0.0118	41.7485	-14358.2962	68.7183	43.6294	41.5368
4.1.05	0.0119	43.3974	-20863.6106	64.4206	45.4044	43.3415
4.1.06	0.0151	50.2411	-20531.3964	82.0751	52.6044	53.7265
4.1.07	0.0115	38.3446	-32041.4723	46.9728	40.3967	37.0458
4.1.08	0.0119	40.5311	-29350.5579	52.3743	42.9729	40.1785
4.2.01	0.0105	41.0319	-12996.8919	66.2440	41.6784	39.5524
4.2.03	0.0141	58.9022	-17811.8769	73.1133	60.0894	62.0433
4.2.05	0.0139	47.8020	-33459.0209	61.9616	50.0903	51.1122
4.2.06	0.0140	47.7164	-19410.9745	80.2701	50.0511	50.1180
4.2.07	0.0122	41.9256	-16943.4129	67.7821	44.3010	42.4176
5.1.09	0.0111	47.7691	-16796.5722	55.9229	48.9364	46.7720
5.1.10	0.0157	58.4775	-21107.4482	78.2653	60.2243	62.2025
5.1.11	0.0109	40.4260	-38016.1132	48.0063	41.4247	40.9802
5.1.12	0.0128	45.3810	-36891.2660	63.1316	46.6671	49.0570
5.1.13	0.0244	111.5673	-50761.9816	117.1397	105.4671	116.2954
5.1.14	0.0129	51.0761	-12304.1432	74.2564	53.0954	52.5074

Filename	Mặc định				Noise = 0.01	
	MATLAB		RPi3		MATLAB	RPi3
	Noise	MSE	Noise	MSE	MSE	MSE
5.2.08	0.0126	46.9440	-16293.7235	61.2985	48.6607	47.5992
5.2.09	0.0167	57.8253	-33225.4572	67.1652	58.4314	62.7763
5.2.10	0.0139	56.6113	-15226.0315	81.3532	57.9642	59.3643
5.3.01	0.0125	43.6389	-10769.8362	65.7852	45.0282	46.3524
5.3.02	0.0125	51.5868	-7596.2661	73.6383	53.2986	52.7383
7.1.01	0.0108	46.0228	-11980.7130	55.3913	46.9575	44.1631
7.1.02	0.0115	39.7870	-30876.5094	41.7975	41.7456	39.2065
7.1.03	0.0110	44.2193	-17975.5355	52.1784	45.4040	42.7549
7.1.04	0.0104	44.5998	-14420.1253	52.3886	45.1665	42.0101
7.1.05	0.0117	50.1191	-12179.5757	66.0205	51.6857	50.0475
7.1.06	0.0113	49.9953	-8977.6643	67.0978	51.0894	49.2660
7.1.07	0.0111	49.9211	-11985.2352	55.9454	51.1771	48.5177
7.1.08	0.0104	43.1070	-16489.4790	46.8678	43.7457	41.0177
7.1.09	0.0113	49.7544	-16676.8104	64.2958	51.0697	49.2057
7.1.10	0.0108	45.2453	-14688.9014	53.0281	46.3296	42.9850
7.2.01	0.0073	24.0014	-1706.1240	70.5824	21.1649	19.9642
boat.512	0.0131	47.6931	-18492.6913	64.2792	49.8632	50.1183
gray21.512	0.0092	39.9605	-21425.8306	67.2078	38.8942	37.1617
house	0.0140	49.9374	-27705.0110	64.0971	52.1837	52.2799
ruler.512	0.0269	139.0348	-47894.1489	147.9325	119.3403	146.8871

Nhận xét:

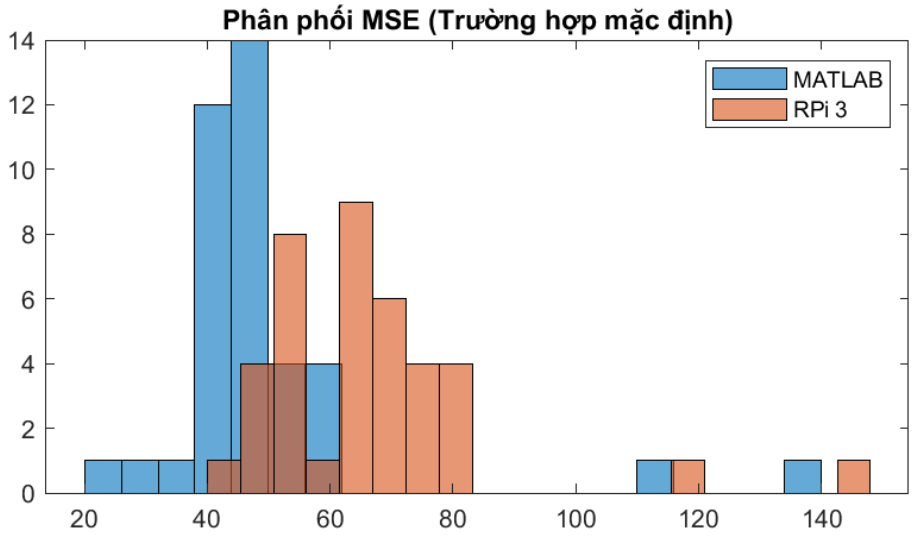
Trường hợp tự ước lượng công suất nhiễu bằng phần mềm:

MATLAB ước lượng tốt công suất nhiễu (Hình 11), 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.



Hình 11: Phân phối nhiễu ước lượng bởi MATLAB

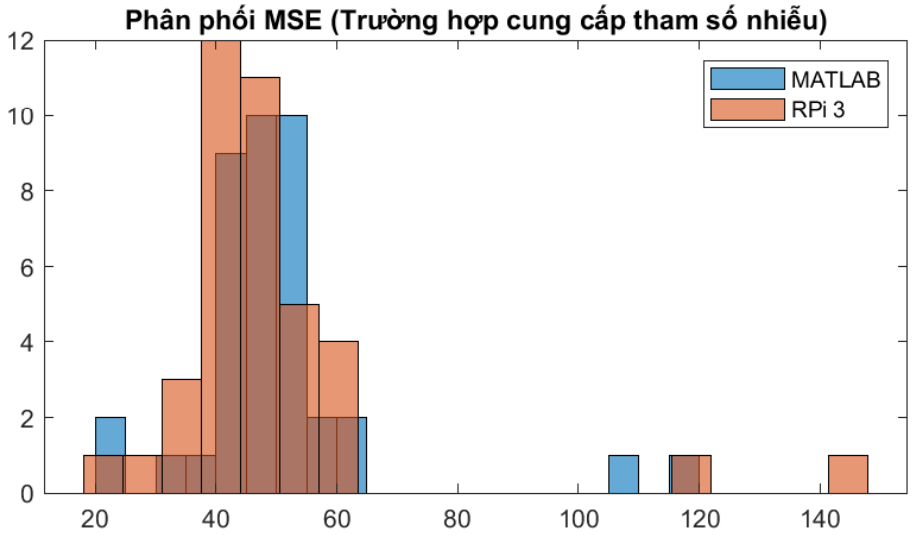
Về mặt sai số, kết quả lọc từ MATLAB cũng có xu hướng cho sai số bình phương trung bình thấp hơn so với kết quả từ Python trên kit Raspberry Pi 3 (Hình 12).



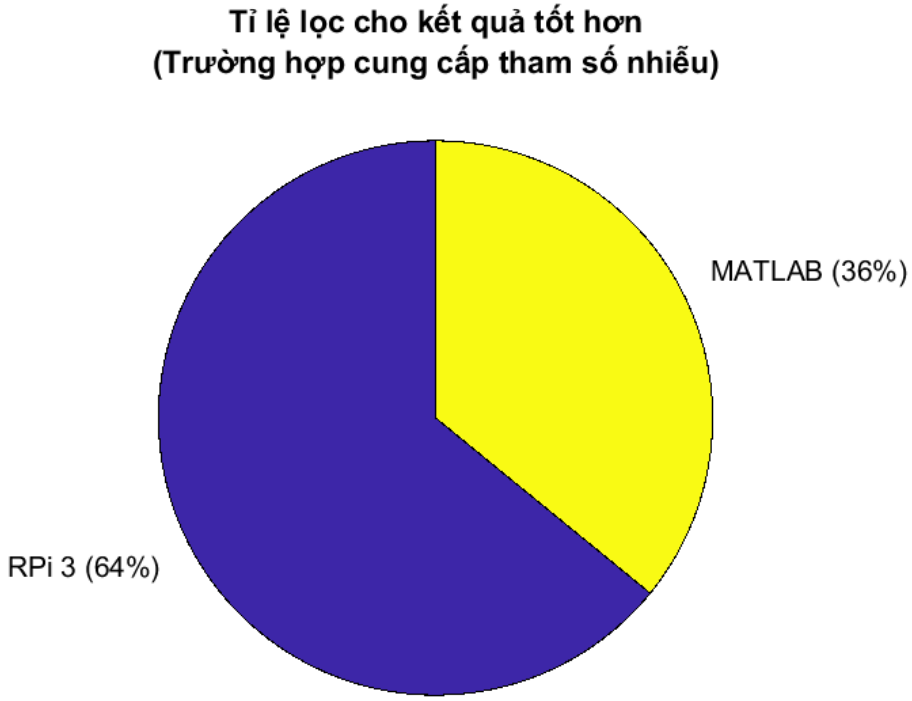
Hình 12: Phân phối MSE (Trường hợp mặc định)

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. Phân phối MSE cho Raspberry Pi 3 có xu hướng nhỏ hơn so với của MATLAB (Hình 13). Trong 39 ảnh được dùng để so sánh giữa MATLAB và Python trên Raspberry Pi 3, có đến gần 2/3 tổng số ảnh mà bộ kit cho kết quả lọc tốt hơn (Hình 14)



Hình 13: Phân phối MSE (Trường hợp cung cấp tham số nhiễu)



Hình 14: Tỉ lệ lọc cho kết quả tốt hơn (Trường hợp cung cấp tham số nhiễu)

6 Kết luận

Đề tài đã thành công lọc được hai loại nhiễu cơ bản trong xử lý ảnh là nhiễu muối tiêu và nhiễu Gauss trên Raspberry Pi 3. Kết quả cho thấy rằng phương pháp giảm nhiễu trên Raspberry Pi 3 có thể đạt được sai số thấp và kết quả gần với ảnh gốc. Trong phần lớn trường hợp, kết quả từ Raspberry Pi 3 không đạt được hiệu suất tốt như trên MATLAB.

Sự khác biệt trong kết quả giữa Raspberry Pi 3 và MATLAB có thể do nhiều yếu tố, bao gồm sự khác biệt trong thuật toán, cấu trúc phần cứng, hoặc việc triển khai thuật toán trên hai nền tảng khác nhau. Cải thiện có thể được thực hiện bằng cách tối ưu hóa thuật toán hoặc sử dụng phần cứng mạnh mẽ hơn để xử lý ảnh.

Đề tài này mở ra nhiều hướng nghiên cứu tiếp theo:

- Tối ưu hóa các phương pháp giảm nhiễu để tăng hiệu suất và độ chính xác trên Raspberry Pi 3.
- Nghiên cứu và áp dụng các phương pháp giảm nhiễu nhằm cải thiện kết quả.
- Khảo sát và áp dụng các công nghệ phần cứng mới cho tác vụ giảm nhiễu hình ảnh.
- Nghiên cứu các phương pháp tự động điều chỉnh tham số để tối ưu hóa việc giảm nhiễu.

Tổng kết lại, việc giảm nhiễu hình ảnh bằng Raspberry Pi 3 là một đề tài quan trọng và có tiềm năng. Mặc dù đã đạt được một số kết quả tích cực, việc tiếp tục nghiên cứu và phát triển là cần thiết để cải thiện hiệu suất và độ chính xác của phương pháp trên nền tảng này.

Tài liệu tham khảo

[1] Bộ môn Viễn Thông. *Tài liệu Thí nghiệm Xử lý số tín hiệu*. Trường Đại học Bách khoa - ĐHQG TPHCM, 2015.

[2] Ajay Kumar Boyat and Brijendra Kumar Joshi. "A review paper: noise models in digital image processing". In: *Signal and Image Processing An International Journal* (2015).

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

[4] William K Pratt. "Generalized Wiener filtering computation techniques". In: *IEEE Transactions on Computers* 100.7 (1972), pp. 636–641.

[5] Signal and Image Processing Institute - University of Southern California. *The USC-SIPI Image Database*. URL: `https://sipi.usc.edu/database/database.php`.

[6] *Tài liệu tham khảo hàm wiener() của module SciPy*. URL: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.wiener.html`.