

BÁO CÁO XỬ LÝ ẢNH

Sinh viên: Nguyễn Tuệ Tâm (19DTCLC1 - 106190033)

Câu 1: Cải thiện chất lượng ảnh.

Ảnh **final1.bmp** đã làm mờ sử dụng bộ lọc Gauss thông thấp và thêm nhiều muối – tiêu (salt-and-pepper noise). Sinh viên hãy thực hiện ít nhất 02 phương pháp để cải thiện chất lượng ảnh đã cho.



Hình 1.1. Ảnh final1.bmp

Em sử dụng hai phương pháp lọc nhiễu ảnh là lọc trung bình và lọc trung vị.

Phương pháp 1: Lọc trung bình

a. Lý do chọn phương pháp này

- Đây là một phương pháp lọc khá đơn giản và ý tưởng dễ tiếp cận, làm tiền đề kiến thức cho những phương pháp lọc sau.

- Phương pháp này còn được gọi là bộ lọc làm mịn, nó cũng được sử dụng để làm mờ một hình ảnh, vậy nên có thể sau khi làm mờ đi những vết nhiễu sẽ thu được hình ảnh mượt hơn

b. Cách làm, giải thích

B1: Đọc ảnh, lấy thông tin các chiều của ảnh, tạo một ma trận mới có kích thước bằng với kích thước ảnh gốc. Sau đó tạo một bộ lọc trung bình có kích thước (3x3) có giá trị như sau:

1/9 1/9 1/9

1/9 1/9 1/9

1/9 1/9 1/9

B2: Chập bộ lọc với hình ảnh bằng cách dịch bộ lọc và nhân tích chập bộ lọc với ảnh. Bước này nghĩa là thay thế giá trị tại mỗi pixel bằng trung bình các giá trị pixel lân cận nhằm loại bỏ những pixel biến đổi lớn so với lân cận (nhiễu).

B3: Lưu ảnh kết quả và hiển thị, nhận xét

c. Code

```
import cv2
import numpy as np

# ham loc trung binh
def loc_trung_binh(img, mask):
    ''' Ham loc trung binh
    Args:          img: anh goc
                   mask: mat na loc trung binh
    Return:         img_res: anh da duoc chap voi bo loc
    '''
    # trich xuat kich thuoc anh
    m, n = img.shape
    # tao mot anh moi de luu ket qua co kich thuoc bang anh goc
    img_res = np.zeros([m, n])
    # chap anh voi bo loc
    for i in range(1, m - 1):
        for j in range(1, n - 1):
            temp = img[i - 1, j - 1] * mask[0, 0] \
                   + img[i - 1, j] * mask[0, 1] \
                   + img[i - 1, j + 1] * mask[0, 2] \
                   + img[i, j - 1] * mask[1, 0] \
                   + img[i, j] * mask[1, 1] \
```

```

        + img[i, j + 1]           * mask[1, 2] \
        + img[i + 1, j - 1]       * mask[2, 0] \
        + img[i + 1, j]           * mask[2, 1] \
        + img[i + 1, j + 1]       * mask[2, 2]

    img_res[i, j] = temp

# dua cac pixel ve kieu int
img_res = img_res.astype(np.uint8)
# tra ve anh ket qua
return img_res

# tao mot ma tran loc trung binh 3x3
mask3x3 = np.ones((3, 3), dtype="float") * (1.0 / (3 * 3))

# doc anh goc
image =
cv2.imread('/home/ngtuetam/workspace/XLA-cuoiky-2022/final1.bmp', 0)

# loc trung binh anh va luu ket qua
image_res = loc_trung_binh(image, mask3x3)
cv2.imwrite('loc_trung_binh.bmp', image_res)

```

d. Kết quả, nhận xét

- Ảnh kết quả tuy các vết nhiễu đã được làm mờ, tuy nhiên tổng thể hình ảnh vẫn chưa được tốt, hình ảnh chưa được cải thiện chất lượng nhiều



Hình 1.2. Ảnh kết quả sau khi lọc trung bình

Phương pháp 2: Lọc trung vị

a. Lý do chọn phương pháp này

- Phương pháp này gần như là một phương pháp kinh điển và dễ thấy khi lọc các loại nhiễu như nhiễu muối và nhiễu Gaussian.
- Đây là một bộ lọc phi tuyến. Nó thường được sử dụng để loại bỏ nhiễu muối và tiêu. Ở đây, giá trị pixel được thay thế bằng giá trị trung bình của pixel lân cận, vậy nên ý tưởng khá đơn giản để lập trình và tính toán

b. Cách làm, giải thích

B1: Đọc ảnh, lấy thông tin các chiều của ảnh, tạo một ma trận mới có kích thước bằng với kích thước ảnh gốc.

B2: Lần lượt di chuyển trên ảnh, đối với mỗi khu vực 3×3 , tìm trung vị của các pixel và thay thế pixel trung tâm bằng trung vị đó

B3: Lưu ảnh kết quả và hiển thị, nhận xét

c. Code

```
import cv2
import numpy as np

# ham loc trung vi
def loc_trung_vi(img):
    '''Ham loc trung vi
    Args:      img: anh goc
    Returns:   img_res: anh ket qua sau khi da qua bo loc trung
vi
    '''
    # trich xuat kich thuoc anh
    m, n = img.shape
    # tao mot anh moi de luu ket qua co kich thuoc bang anh goc
    img_res = np.zeros([m, n])
    # bat dau loc trung vi
    for i in range(1, m-1):
        for j in range(1, n-1):
            temp = [img[i-1, j-1],
                    img[i-1, j],
                    img[i-1, j + 1],
                    img[i, j-1],
                    img[i, j],
                    img[i, j + 1],
                    img[i + 1, j-1],
                    img[i + 1, j],
                    img[i + 1, j + 1]]
            # thay pixel o giua thanh pixel trung vi (co vi tri
            la 4)
            temp = sorted(temp)
            img_res[i, j]= temp[4]
    # dua cac pixel ve kieu int
    img_res = img_res.astype(np.uint8)
    # tra ve anh ket qua
    return img_res

# doc anh goc
```

```
image =  
cv2.imread('/home/ngtuetam/workspace/XLA-cuoiky-2022/final1.bm  
p', 0)  
  
# loc trung vi 3x3 va luu anh  
img_res = loc_trung_vi(image)  
cv2.imwrite('loc_trungvi.bmp', img_res)
```

d. Kết quả, nhận xét

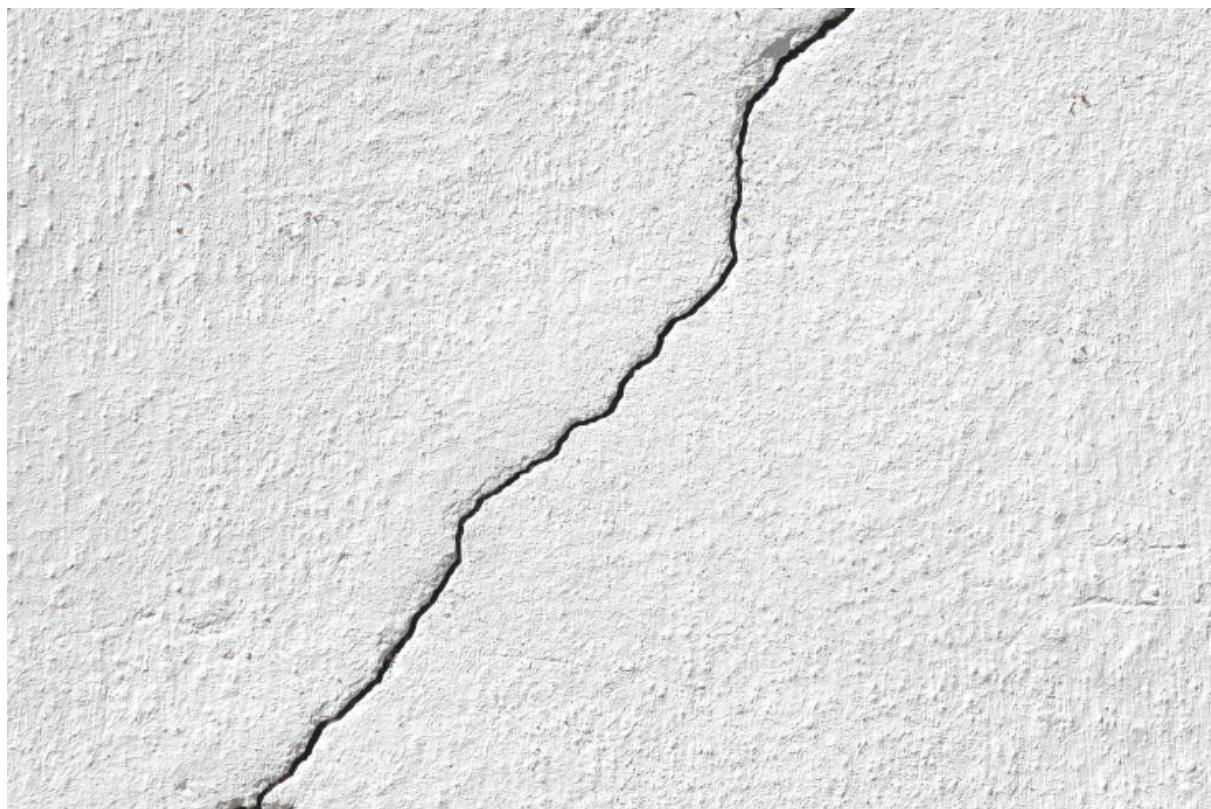
- Ảnh kết quả đã được làm sạch nhiều, các vết muối tiêu đã không còn, vậy phương pháp lọc trung vị tốt hơn phương pháp lọc trung bình trong bài tập này.



Hình 1.3. Ảnh kết quả sau khi lọc trung vị

Câu 2: Phát hiện đường/cạnh.

Ảnh **final2.jpg** là ảnh vết nứt trên tường. Sinh viên hãy sử dụng ít nhất 02 phương pháp phát hiện đường (lines/edge), để phát hiện vết nứt trong hình.



Hình 2.1. Ảnh final2.jpg

Em sử dụng hai phương pháp phát hiện cạnh là Sobel và Roberts

Phương pháp 1: Kỹ thuật Sobel

a. Lý do chọn phương pháp này

- Sobel ước tính độ lớn và hướng của gradient tại mỗi pixel trong ảnh thang độ xám. Các pixel có cường độ cao tạo thành một cạnh. Trong số các thuật toán phát hiện cạnh có sẵn, kỹ thuật Sobel có hiệu quả cao nhất.
- Sobel cũng là một thuật toán dễ hình dung, đơn giản để lập trình

b. Cách làm, giải thích

B1: Chuyển đổi hình ảnh thành thang độ xám grayscale

B2: Chập hình ảnh màu xám với bộ lọc Sobel-x (horizontal mask).

1 0 -1

2 0 -2

1 0 -1

- Gọi \mathbf{A} là ảnh gốc và \mathbf{Gx} là ảnh mà tại mỗi điểm chứa các xấp xỉ đạo hàm ngang, ta có công thức như sau:

$$\mathbf{G}_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * \mathbf{A}$$

B3: Chập hình ảnh màu xám với bộ lọc Sobel-y (vertical mask)

1 2 -1
0 0 0
-1 -2 -1

- Gọi \mathbf{A} là ảnh gốc và \mathbf{Gy} là ảnh mà tại mỗi điểm chứa các xấp xỉ đạo hàm dọc, ta có công thức như sau:

$$\mathbf{G}_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

B4: Tính độ lớn gradient. Tại mỗi điểm trong ảnh, kết quả xấp xỉ đạo hàm theo chiều dọc và ngang có thể được kết hợp để tạo ra độ lớn của gradient, bằng cách sử dụng công thức sau

$$\mathbf{G} = \sqrt{\mathbf{G}_x^2 + \mathbf{G}_y^2}$$

c. Code

```
import cv2

import matplotlib.pyplot as plt

import numpy as np
```

```

def ky_thuat_sobel(img):
    '''Phat hien duong bang ky thuật Sobel

    Args:      img : anh goc

    Returns:   img_res: anh ket qua sau khi phat hien duong bang
              ky thuật Sobel

    '''

    sobel_x = np.array(
        [[1.0, 0.0, -1.0],
         [2.0, 0.0, -2.0],
         [1.0, 0.0, -1.0]])

    sobel_y = np.array(
        [[1.0, 2.0, 1.0],
         [0.0, 0.0, 0.0],
         [-1.0, -2.0, -1.0]])

    # trich xuat thong tin kich thuoc anh
    m, n = img.shape

    # tao mot anh moi de luu ket qua co kich thuoc bang anh goc
    img_res = np.zeros([m, n])

    for i in range(m - 2):
        for j in range(n - 2):
            gx = np.sum(np.multiply(sobel_x, img[i:i + 3, j:j + 3]))
            gy = np.sum(np.multiply(sobel_y, img[i:i + 3, j:j + 3]))
            img_res[i + 1, j + 1] = np.sqrt(gx ** 2 + gy ** 2)

    return img_res

```

```

# doc anh goc

image =
cv2.imread('/home/ngtuetam/workspace/XLA-cuoiky-2022/final2.jpg',
cv2.IMREAD_GRAYSCALE)

# phat hien canh bang ky thuật sobel

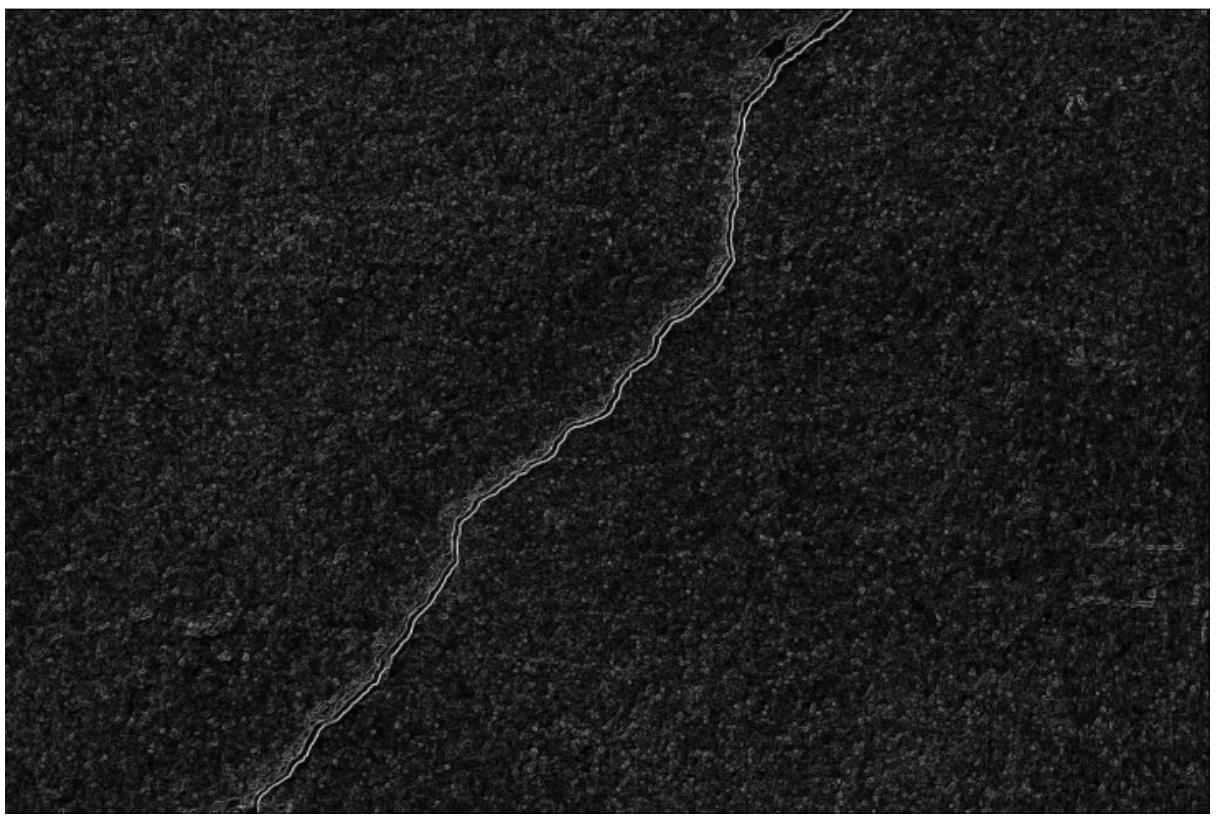
img_res = ky_thuat_sobel(image)

plt.imsave('anh_sobel.jpg', img_res,
cmap=plt.get_cmap('gray'))

```

d. Kết quả, nhận xét

- Ảnh kết quả cho thấy kỹ thuật phát hiện khá tốt, nổi bật được các đường nứt, tuy nhiên vì nền sần sùi nên vẫn còn các chấm li ti nhỏ chưa được giải quyết



Hình 2.2. Ảnh được phát hiện qua kỹ thuật Sobel

Phương pháp 2: Kỹ thuật Roberts

a. Lý do chọn phương pháp này

- Thuật toán này khá tương tự với Sobel vì đều cùng là Gradient Based, tức là các toán tử dựa trên tính toán các dẫn xuất bậc nhất trong một hình ảnh kỹ thuật số
- Thuật toán đơn giản, chính xác, được áp dụng nhiều và dễ lập trình, không tốn quá nhiều độ phức tạp thời gian.

b. Cách làm, giải thích

B1: Chuyển đổi hình ảnh thành thang độ xám grayscale

B2: Khởi tạo cặp ma trận của Roberts (được gọi là các kernel)

$$\begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix} \quad \text{và} \quad \begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix}$$

B3: Lần lượt chập hình ảnh xám với hai kernel vừa được khởi tạo.

B4: Tính độ lớn Gradient

- Gọi $I(x,y)$ là một điểm của hình ảnh gốc, $G_x(x,y)$ là một điểm trong hình ảnh được tạo bằng cách chập ảnh gốc với kernel đầu tiên, $G_y(x,y)$ là một điểm trong hình ảnh được tạo bằng cách chập ảnh gốc với kernel thứ hai. Độ lớn Gradient có thể được tính như sau

$$\nabla I(x, y) = G(x, y) = \sqrt{G_x^2 + G_y^2}.$$

c. Code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def ky_thuat_roberts(img):
    '''Phat hien duong bang ky thuật Roberts
    Args:      img : anh goc
    Returns:   img_res: anh ket qua sau khi phat hien duong bang
    ky thuật Roberts
    '''
    roberts_v = np.array( [[1, 0],
                           [0, -1]] )
    roberts_h = np.array([[0, 1],
                           [-1, 0]])

    # trich xuat thong tin kich thuoc anh
    m, n = img.shape
```

```

# tao mot anh moi de luu ket qua co kich thuoc bang anh goc
img_res = np.zeros([m, n])

for i in range(m - 1):
    for j in range(n - 1):
        gx = np.sum(np.multiply(roberts_v, img[i:i + 2, j:j + 2]))
        gy = np.sum(np.multiply(roberts_h, img[i:i + 2, j:j + 2]))
        img_res[i + 1, j + 1] = np.sqrt(gx ** 2 + gy ** 2)
return img_res

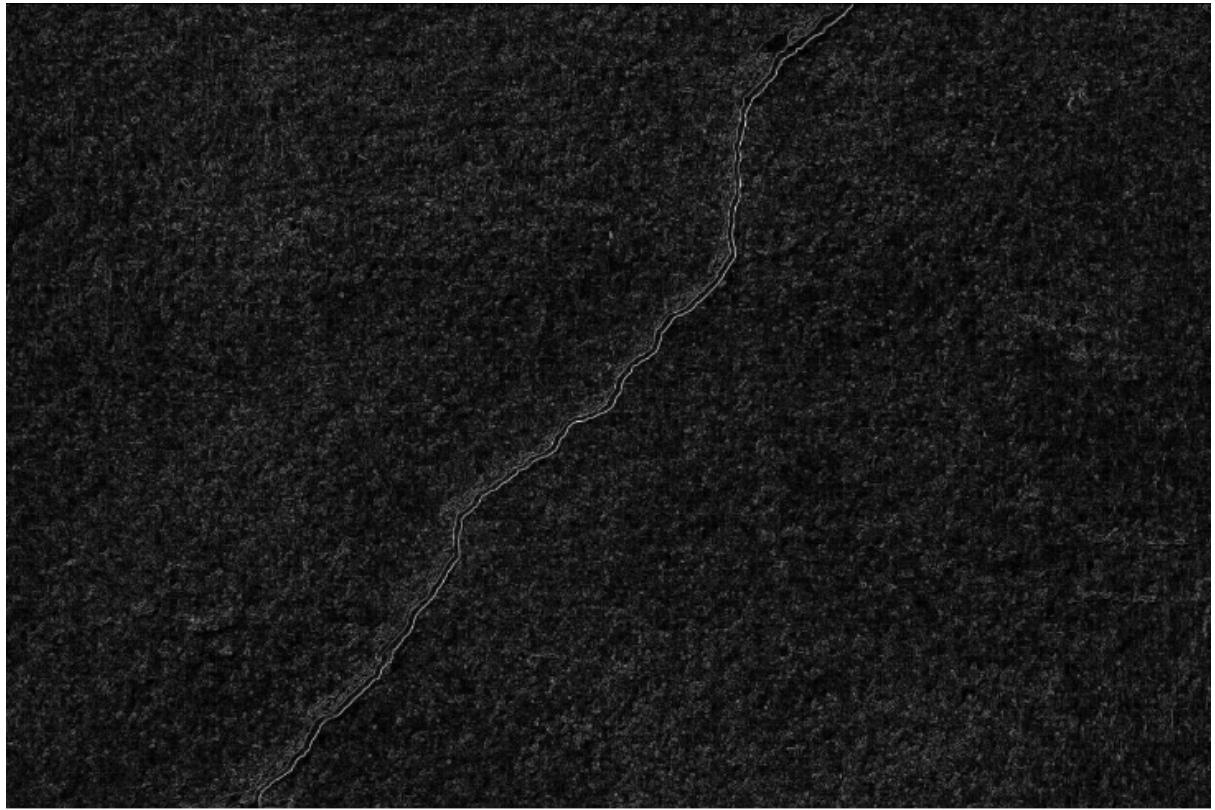
# doc anh goc
image =
cv2.imread('/home/ngtuetam/workspace/XLA-cuoiky-2022/final2.jpg',
cv2.IMREAD_GRAYSCALE)

# phat hien canh bang ky thuật Roberts
img_res = ky_thuat_roberts(image)
plt.imsave('anh_roberts.jpg', img_res,
cmap=plt.get_cmap('gray'))

```

d. Kết quả, nhận xét

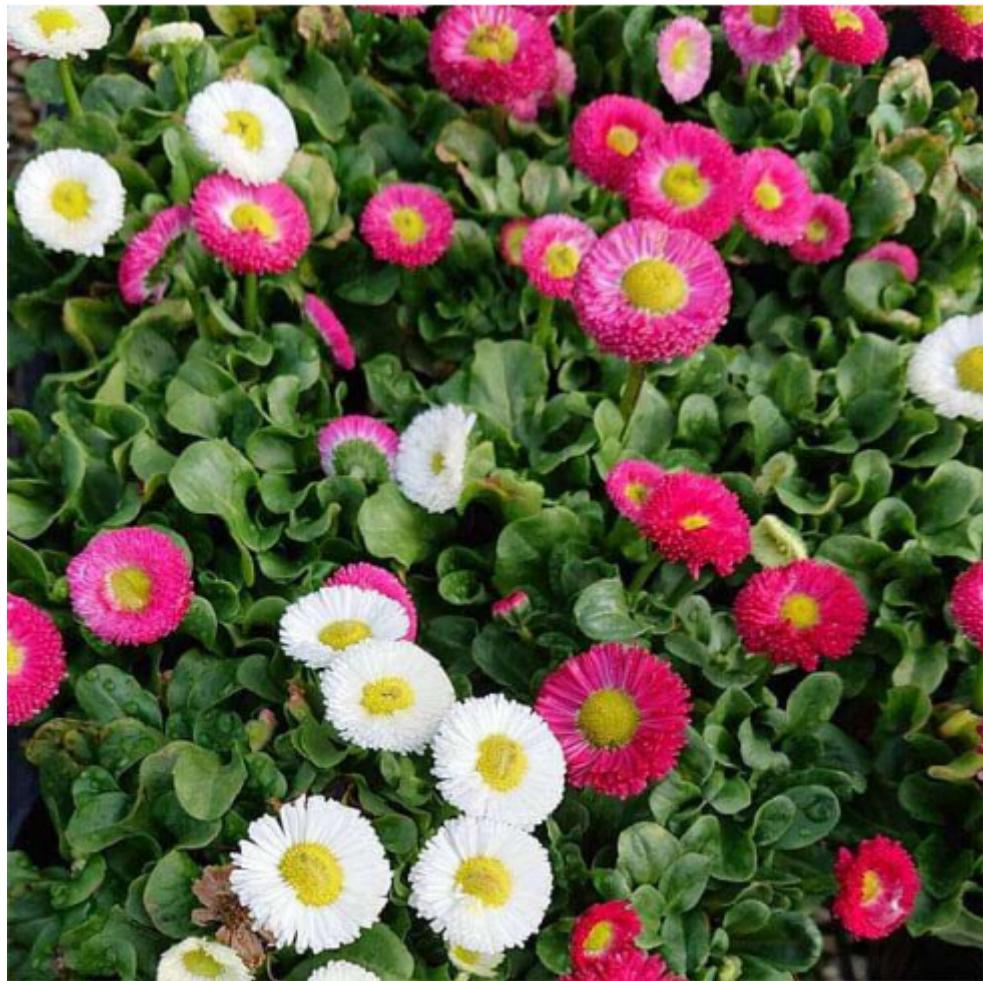
- Ảnh được phát hiện tốt các cạnh và nét, các điểm hướng chéo được giữ nguyên. Tuy nhiên so với kỹ thuật Sobel, các điểm trắng không nổi bật hơn, hình ảnh không dễ nhìn như Sobel, ngoài ra Roberts vẫn giống với kỹ thuật Sobel, ảnh kết quả vẫn bị nhạt cảm với nhiều.



Hình 2.2. Ảnh được phát hiện qua kỹ thuật Roberts

Câu 3: Phân vùng ảnh

Thực hiện phân vùng ảnh với ảnh **final3.jpg**. Đơn giản nhất, sinh viên có thể thực hiện phân vùng ảnh dựa trên thông tin màu sắc của các điểm ảnh (pixels) trong ảnh đã cho.



Hình 3.1. Ảnh final3.jpg

Em sử dụng phương pháp K-means để phân vùng ảnh

a. Lý do chọn phương pháp này

- K-Means là phương pháp thường được dùng nhất trong các nhiệm vụ phân đoạn ảnh. Em cũng được học K-Means qua các môn học ở trường, vậy nên áp dụng kiến thức vào để vận dụng làm bài tập là một cách để nắm vững bài học hơn.
- Không cần sử dụng các thuật toán phức tạp của AI hay Deep learning, với K-Means ta dễ dàng nắm được ý tưởng và triển khai thuật toán một cách đơn giản, thuật toán không mất quá nhiều thời gian để thực hiện

b. Cách làm, giải thích

B1: Chuẩn bị dữ liệu cho KMeans. Ảnh có 3 chiều RGB, để chạy thuật toán K Means ta cần định hình lại hình ảnh thành một mảng hai chiều có kích thước Mx3 (với M là số pixel trong ảnh).

B2: Chạy thuật toán KMeans

- (1) Xác định số cụm mong muốn: gọi là K
- (2) Gán ngẫu nhiên các điểm dữ liệu cho K cụm bất kỳ

- (3) Sau đó tính tâm của K cụm đã có.
- (4) Tính khoảng cách của các điểm dữ liệu đến trung tâm (centroid) của mỗi cụm.
- (5) Dựa vào khoảng cách của từng điểm dữ liệu từ cụm, gán lại các điểm dữ liệu cho các cụm gần nhất.
- (6) Tính centroid của cụm mới
- (7) Lặp lại các bước 4, 5 và 6 cho đến khi các điểm dữ liệu không thay đổi cụm hoặc cho đến khi đạt được số lần lặp được chỉ định.

B3: Áp dụng các giá trị trọng tâm (cũng là R,G,B) cho tất cả các pixel, sao cho hình ảnh thu được sẽ có số lượng màu được chỉ định.

c. Code

```

import numpy as np
import matplotlib.pyplot as plt
import cv2

def k_means(image, K):
    '''Thuật toán K-Means
    Args:      image: hình ảnh gốc
               K: số cụm
    Returns:   img_res : ảnh kết quả đã được phân đoạn màu sắc
    '''

    # vì cv2 đọc ảnh theo thứ tự BGR nên ta chuyển về RGB
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # reshape hình ảnh thành 2 chiều có 3 giá trị RGB
    img_vectorized = image.reshape((-1, 3))      # (360000x3)
    # chuyển về kiểu float
    img_vectorized = np.float32(img_vectorized)

    # kmeans dùng lại khi đạt 100 vòng lặp hoặc do chính xác =
    # 85% (epsilon = 0.85)
    criteria = (cv2.TERM_CRITERIA_EPS +
    cv2.TERM_CRITERIA_MAX_ITER, 100, 0.85)

    # chọn khởi tạo centers theo kiểu ngẫu nhiên (random) cho
    # k-means

```

```

        retval, labels, centers = cv2.kmeans(img_vectorized, K,
None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

        # convert center ve gia tri 8-bit
centers = np.uint8(centers)

        # áp dụng các giá trị trọng tâm (cũng là R,G,B) cho tất cả
các pixel,
        # sao cho hình ảnh thu được sẽ có số lượng màu được chỉ
định.

result_data = centers[labels.flatten()]

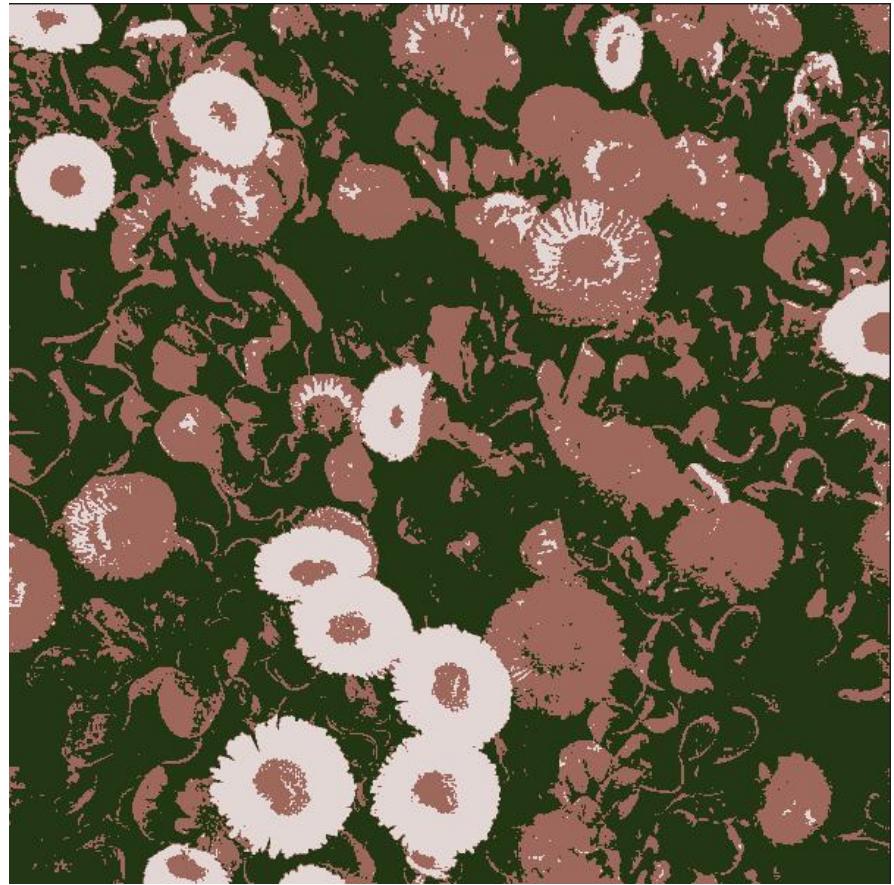
        # reshape data into the original image dimensions
img_res = result_data.reshape((image.shape))
return img_res

# doc anh goc
image =
cv2.imread('/home/ngtuetam/workspace/XLA-cuoiky-2022/cau03/fin
al3.jpg')
# ap dung k-means
img_res = k_means(image,10)
# luu anh ket qua
plt.imsave('kmeans_10.jpg', img_res,
cmap=plt.get_cmap('gray'))

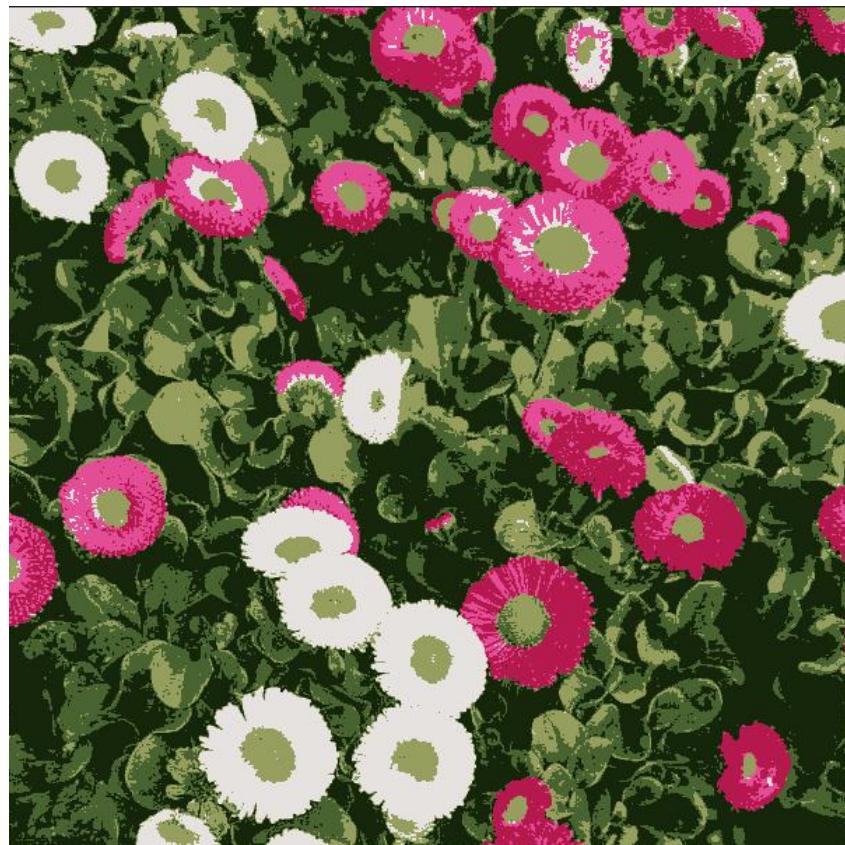
```

d. Kết quả, kết luận

- Thực hiện thuật toán K-Means với K=3, K=6 và K=10



Hình 3.2. K-Means với K=3



Hình 3.3. K-Means với K=6



Hình 3.4. K-Means với K=10

- Nhận xét: thuật toán chạy tốt trên ảnh đã cho. Với K=3, ảnh được chia thành 3 màu trắng, nâu và đen. Với K=6, ảnh đã rõ hơn với nhiều màu sắc hơn, tuy nhiên với K=10 thì mới đạt kết quả đẹp nhất, các nhụy hoa không còn màu xanh như K=6 mà hiển thị được màu vàng gần giống với ảnh gốc

