**ЛАБОРАТОРНА РОБОТА № 7**

**СЕГМЕНТАЦІЯ ЗОБРАЖЕНЬ В СИСТЕМІ SCILAB**

**Мета роботи** – набуття навичок реалізації методів сегментації зображень у середовищі Scilab, дослідження ефективності методів виділення областей та контурів, а також використання градієнтних фільтрів.

**7.1 Короткі теоретичні відомості**

*Сегментація зображення* – це процес розбиття цифрового зображення на групи пікселів (сегменти) на основі певних критеріїв (колір, яскравість, текстура). Метою є спрощення подання зображення для подальшого аналізу.

У даній роботі розглядаються такі методи:

1. *Нарощування областей (Region Growing)* – метод, що об'єднує пікселі в групи навколо початкових точок («центрів кристалізації») на основі подібності їхніх властивостей.
2. *Порогова сегментація (Thresholding)* – як альтернатива складним методам поділу-злиття, використовується метод Оцу для автоматичного визначення порогу.
3. *Виділення контурів (Edge Detection)* – використання градієнтних фільтрів для знаходження меж об'єктів.

УВАГА! Перед початком роботи переконайтеся, що встановлено модуль обробки зображень IPCV.

У консолі Scilab виконайте:

atomsInstall('IPCV'); // Якщо не встановлено

atomsLoad('IPCV');

### **7.1.1 Метод центроїдного зв'язування (Region Growing)**

Метод центроїдного зв'язування відноситься до методу нарощування областей. На вихідному зображенні користувач заздалегідь розміщує точки (центри кристалізації), щодо яких відбуватиметься нарощування областей. Потім навколо цих точок нарощуються області шляхом додавання до кожного центру сусідніх пікселів, які за своєю яскравістю близькі до яскравості точки-центру кристалізації [1]. Пікселам кожної області надається певна мітка – ідентифікатор області (наприклад, числом). У разі злиття областей пікселі об'єднаної нової області отримують єдину мітку, що призводить до необхідності перепозначення їх та коригування таблиці міток.

Метод полягає у виборі початкових точок (seeds) і приєднанні до них сусідніх пікселів, якщо різниця їхньої яскравості не перевищує заданий поріг.

Оскільки в Scilab (і в базовому модулі IPCV) відсутня готова функція regiongrow, для виконання роботи необхідно використовувати наведену нижче функцію.

**Лістинг функції regiongrow\_scilab**

function [**g**, **NR**]=regiongrow\_scilab(**f**, **S**, **T**)

*// f - Вхідне зображення (напівтонове)*

*// S - Матриця сідів (0 і 1), того ж розміру, що й f*

*// T - Поріг (максимальна різниця яскравості)*

*// g - Результат сегментації (бінарне зображення)*

*// NR - Кількість знайдених пікселів*

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

**f** = double(**f**); *// Працюємо з числами double*

**g** = zeros(rows, cols); *// Результат*

visited = zeros(rows, cols); *// Маска відвіданих*

*// Знаходимо координати всіх точок-сідів*

[y\_seeds, x\_seeds] = find(**S** == 1);

num\_seeds = size(y\_seeds, 1);

*// Стек для обробки (LIFO)*

stack\_x = x\_seeds;

stack\_y = y\_seeds;

*// Початкова ініціалізація відвіданих точок*

for k=1:num\_seeds

visited(y\_seeds(k), x\_seeds(k)) = 1;

**g**(y\_seeds(k), x\_seeds(k)) = 1;

end

*// Напрямки для 8-зв'язності*

dx = [-1, 0, 1, -1, 1, -1, 0, 1];

dy = [-1, -1, -1, 0, 0, 1, 1, 1];

*// Середнє значення яскравості початкових регіонів*

seed\_values = **f**(**S** == 1);

reg\_mean = mean(seed\_values);

**NR** = num\_seeds;

*// Поки стек не порожній*

while ~isempty(stack\_x)

*// Витягуємо точку*

curr\_x = stack\_x($);

curr\_y = stack\_y($);

stack\_x($) = []; *// pop*

stack\_y($) = []; *// pop*

*// Перевіряємо сусідів*

for k=1:8

nx = curr\_x + dx(k);

ny = curr\_y + dy(k);

*// Перевірка меж*

if (nx > 0 & nx <= cols & ny > 0 & ny <= rows) then

if visited(ny, nx) == 0 then

*// Перевірка критерію однорідності*

if abs(**f**(ny, nx) - reg\_mean) <= **T** then

**g**(ny, nx) = 1;

visited(ny, nx) = 1;

**NR** = **NR** + 1;

*// Додаємо в стек*

stack\_x($+1) = nx;

stack\_y($+1) = ny;

*// Оновлюємо середнє (опціонально, тут спрощено)*

*// reg\_mean = (reg\_mean \* (NR-1) + f(ny, nx)) / NR;*

end

end

end

end

end

endfunction

Розглянемо приклад використання цієї функції. Для її використання необхідно змінити робочу папку на директорію з файлом цієї функції (**regiongrow\_scilab.sci**).

Як вихідне зображення було взято напівтонове зображення, представлене рис. 7.1.

*// ==========================================*

*// Приклад виконання сегментації (Scilab)*

*// ==========================================*

clc;

*// 1. Завантаження функції*

*// Ми використовуємо try-catch, щоб попередити, якщо функція не знайдена*

try

exec('regiongrow\_scilab.sci', -1);

catch

disp('Увага: Функція regiongrow\_scilab.sci не завантажилася автоматично.');

*// У наступному рядку внутрішні лапки подвоєні (''), щоб Scilab їх зрозумів*

disp('Будь ласка, виконайте exec(''шлях\_до\_файлу/regiongrow\_scilab.sci'', -1) вручну.');

end

*// 2. Завантаження зображення*

*// Вкажіть ПОВНИЙ шлях до файлу, якщо він не поруч*

image\_filename = 'face.png';

try

f = imread(image\_filename);

catch

disp('ПОМИЛКА: Не вдалося знайти файл: ' + image\_filename);

disp('Перевірте назву файлу та шлях до нього!');

abort; *// Зупинити виконання*

end

*// Перевірка на кольоровість*

if size(f, 3) == 3 then

f = rgb2gray(f);

end

f = double(f);

*// 3. Створення маски*

[rows, cols] = size(f);

Mask = zeros(rows, cols);

*// 4. Вибір точок кристалізації*

*// Координати (y, x) перевіряються на вихід за межі*

if rows > 100 & cols > 120 then

Mask(38, 79) = 1;

Mask(93, 64) = 1;

Mask(92, 117) = 1;

Mask(100, 64) = 1;

else

disp('Зображення замале для координат з прикладу. Точку встановлено в центрі.');

Mask(round(rows/2), round(cols/2)) = 1;

end

*// 5. Виклик функції*

T = 30; *// Поріг*

disp('Початок сегментації...');

[g, NR] = regiongrow\_scilab(f, Mask, T);

disp('Готово. Кількість пікселів: ' + string(NR));

*// 6. Відображення*

*// Використовуємо scf() (set current figure)*

scf(0); clf();

imshow(uint8(f));

title('Оригінал');

scf(1); clf();

imshow(g);

title('Результат сегментації');

scf(2); clf();

*// Робимо точки жирнішими для візуалізації*

se = imcreatese('rect', 3, 3);

Mask\_vis = imdilate(Mask, se);

imshow(Mask\_vis \* 255);

title('Точки кристалізації');

|  |  |
| --- | --- |
|  | |
| Рисунок 7.1 – Оригінальне зображення | |
|  |  |
| Рисунок 7.2 – Результат сегментації | Рисунок 7.3 – Точки кристалізації |

### **7.1.2 Метод поділу-злиття**

Як видно з назви, метод підрозділяється на два етапу: поділ зображення на ряд областей та подальше злиття частини з них.

Поділ можна уявити в вигляді квадродерева (рис. 7.4) [1].

|  |  |
| --- | --- |
|  |  |
| Рисунок 7.4 – Поділ зображення на частини | Рисунок 7.5 – Відповідне квадродерево |

Як видно з рис. 7.4-7.5, вихідне зображення *R* ділиться на частини. Поділ здійснюється в відповідність з правилом *P.*

Алгоритм поділу в загалом вигляді може бути записаний наступним чином:

* будь-яка квадратна область *Rj* , для якої *P(Rj) = FALSE* , поділяється на чотири квадратні блоки, що не перетинаються, - чверті;
* будь-які дві сусідні області *Rj* і *Rk,* для яких *P( Rj A \cup B* *Rk) = TRUE* об'єднуються в одну;
* процедура завершується, якщо неможливо виконати ні одну операцію злиття чи поділу.

У середовищі Scilab відсутня вбудована функція квадродерева. Тому було реалізовано алгоритм рекурсивного розбиття зображення з використанням стекової структури даних. Критерієм розбиття виступала різниця між максимальною та мінімальною яскравістю сегмента.

Розглянемо приклад квадратичного розкладання зображення.

*// ======================================================*

*// Quadtree Decomposition*

*// ======================================================*

clc;

*// 1. Завантаження зображення*

*// Спробуємо знайти стандартне зображення або використовуємо ваше*

image\_filename = 'cameraman.png';

*// Перевірка наявності файлу*

*// Якщо файлу немає, генеруємо тестове зображення (квадрати)*

if fileinfo(image\_filename) == [] then

disp('Файл не знайдено, генеруємо тестове зображення...');

img = zeros(512, 512);

img(100:300, 100:300) = 0.5; *// Світліший квадрат*

img(350:450, 350:450) = 0.8; *// Ще світліший*

img = img + rand(512, 512) \* 0.1; *// Додаємо шум*

img = uint8(img \* 255);

else

img = imread(image\_filename);

end

if size(img, 3) == 3 then

img = rgb2gray(img);

end

*// Нормалізуємо до діапазону [0, 1]*

I = double(img) / 255;

[H, W] = size(I);

*// ======================================================*

*// 2. Алгоритм Квадродерева (реалізація через стек)*

*// ======================================================*

*// Параметри розбиття*

Threshold = 0.2; *// Поріг чутливості. Чим менше, тим більше квадратів.*

MinDim = 4; *// Мінімальний розмір блоку*

*// Матриця для результату (сітка)*

blocks = zeros(H, W);

*// Стек для блоків: кожен рядок [r, c, size]*

*// r, c - координати лівого верхнього кута*

stack = list();

stack($+1) = [1, 1, min(H, W)]; *// Починаємо з повного зображення*

disp('Початок розбиття квадродерева...');

count = 0;

while length(stack) > 0

*// Дістаємо останній елемент (pop)*

curr = stack($);

stack($) = null();

r = curr(1);

c = curr(2);

dim = curr(3);

*// Вирізаємо поточний блок*

sub\_img = I(r : r+dim-1, c : c+dim-1);

*// Критерій розбиття: різниця між макс і мін яскравістю*

val\_min = min(sub\_img);

val\_max = max(sub\_img);

diff = val\_max - val\_min;

*// Якщо блок неоднорідний І розмір > мінімального -> Розбиваємо*

if (diff > Threshold) & (dim > MinDim) then

half = dim / 2;

*// Додаємо 4 дочірні блоки в стек*

stack($+1) = [r, c, half]; *// Верхній лівий*

stack($+1) = [r, c+half, half]; *// Верхній правий*

stack($+1) = [r+half, c, half]; *// Нижній лівий*

stack($+1) = [r+half, c+half, half]; *// Нижній правий*

else

*// Якщо розбивати не треба -> Малюємо межі цього блоку*

*// Малюємо рамку (ставимо 1)*

*// Верхня і нижня лінії*

blocks(r, c : c+dim-1) = 1;

blocks(r+dim-1, c : c+dim-1) = 1;

*// Ліва і права лінії*

blocks(r : r+dim-1, c) = 1;

blocks(r : r+dim-1, c+dim-1) = 1;

end

count = count + 1;

end

disp('Розбиття завершено.');

*// ======================================================*

*// 3. Відображення результатів*

*// ======================================================*

*// Початкове зображення*

scf(0); clf();

imshow(I);

title('Початкове зображення');

*// Результат розкладання (сітка)*

scf(1); clf();

imshow(blocks);

title('Результат квадродерева');

*// Комбіноване*

result\_combined = I + blocks;

result\_combined(result\_combined > 1) = 1;

scf(2); clf();

imshow(result\_combined);

title('Накладання сітки на оригінал');

Початкове зображення представлено на рис. 7.6, розкладене зображення – на рис. 7.7 та 7.8.

|  |  |
| --- | --- |
|  |  |
| Рисунок 7.6 – Початкове зображення | Рисунок 7.6 – Результат квадродерева |
|  | |
| Рисунок 7.8 – Накладання сітки на оригінал | |

Оскільки в Scilab (і в модулі IPCV) повна реалізація методу поділу-злиття відсутня, нам потрібно реалізувати спрощену версію цього алгоритму самостійно.

*// ======================================================*

*// Split & Merge*

*// ======================================================*

clc;

*// 1. Завантаження зображення*

*// ------------------------------------------------------*

image\_filename = 'fig.png';

if fileinfo(image\_filename) == [] then

disp('ПОМИЛКА: Файл ' + image\_filename + ' не знайдено.');

disp('Переконайтеся, що файл знаходиться у папці: ' + pwd());

abort;

end

f = imread(image\_filename);

*// Якщо зображення кольорове - переводимо у відтінки сірого*

if size(f, 3) == 3 then

f = rgb2gray(f);

end

*// ------------------------------------------------------*

*// 2. Функція splitmerge\_scilab*

*// ------------------------------------------------------*

function **g**=splitmerge\_scilab(**f**, **mindim**, **matOj**, **minMean**, **maxMean**)

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

**f** = double(**f**); *// Працюємо з double*

**g** = zeros(rows, cols); *// Результат - чорний фон*

*// Кількість блоків по вертикалі та горизонталі*

num\_blocks\_r = floor(rows / **mindim**);

num\_blocks\_c = floor(cols / **mindim**);

*// --- ЦИКЛ 1: По рядках ---*

for r = 1:num\_blocks\_r

*// --- ЦИКЛ 2: По стовпцях ---*

for c = 1:num\_blocks\_c

*// Розрахунок координат*

r\_start = (r-1)\***mindim** + 1;

r\_end = r\_start + **mindim** - 1;

c\_start = (c-1)\***mindim** + 1;

c\_end = c\_start + **mindim** - 1;

*// Витягуємо поточний блок*

block = **f**(r\_start:r\_end, c\_start:c\_end);

*// Статистика блоку*

m\_val = mean(block);

s\_val = stdev(block);

*// --- УМОВА IF ---*

*// Перевіряємо, чи підходить блок під критерії об'єкта*

if (m\_val >= **minMean**) & (m\_val <= **maxMean**) & (s\_val > **matOj**) then

**g**(r\_start:r\_end, c\_start:c\_end) = 1;

end

*// КІНЕЦЬ IF*

end

*// КІНЕЦЬ ЦИКЛУ 2 (по стовпцях)*

end

*// КІНЕЦЬ ЦИКЛУ 1 (по рядках)*

endfunction

*// КІНЕЦЬ ФУНКЦІЇ*

*// ------------------------------------------------------*

*// 3. Виконання*

*// ------------------------------------------------------*

disp('Обробка зображення ' + image\_filename + ' ...');

*// Випадок А: mindim = 2*

disp('Розрахунок для mindim = 2 ...');

g2 = splitmerge\_scilab(f, 2, 10, 0, 125);

*// Випадок Б: mindim = 8*

disp('Розрахунок для mindim = 8 ...');

g8 = splitmerge\_scilab(f, 8, 10, 0, 125);

*// ------------------------------------------------------*

*// 4. Відображення*

*// ------------------------------------------------------*

scf(0); clf();

imshow(f);

title('Оригінал');

scf(1); clf();

imshow(g2);

title('Splitmerge: mindim = 2');

scf(2); clf();

imshow(g8);

title('Splitmerge: mindim = 8');

disp('Завдання завершено.');

Розглянемо приклад використання цієї функції. Нехай на рис. 7.9 необхідно виділити менш яскраву область, що оточує центральну більш яскраву область. Область, що цікавить, має кілька характеристик, які можуть допомогти при сегментації.

Замість складного рекурсивного квадродерева, функція реалізує блочну декомпозицію: зображення розбивається на квадрати розміром mindim × mindim. Для кожного квадрата обчислюється середнє значення (mean) та стандартне відхилення (stdev). Предикат (умова відбору): блок вважається частиною об'єкта, якщо його середня яскравість потрапляє в діапазон [0,125] (це відсікає яскравий центр), а стандартне відхилення >10 (це відсікає ідеально гладкий чорний фон, якщо він без шуму, або просто перевіряє наявність текстури).

В результаті при mindim = 2 межі виділеного об'єкта більш детальні, при mindim = 8 межі стають "блоковими", грубими, що демонструє принцип роботи методу split-merge (робота з блоками).

|  |  |
| --- | --- |
|  |  |
| Рисунок 7.9 – Вхідне зображення | Рисунок 7.10 – Фрагментоване зображення з блоками мінімального розміру 2x2 |
|  | |
| Рисунок 7.11 – фрагментоване зображення з блоками мінімального розміру 8x8 | |

Як видно з рисунку 7.11, для коректного сегментування кращим вибором був мінімальних розмірів блоків 8x8.

**7.1.3 Градієнтні фільтри**

Для знаходження меж об'єктів зображення часто використовують градієнтні фільтри. У спрощеній формі градієнтний фільтр представляє маску розміром 3x3, яка ковзним вікном проходить по всіх пікселях зображення. Цей процес відображено рис. 7.12 [3].

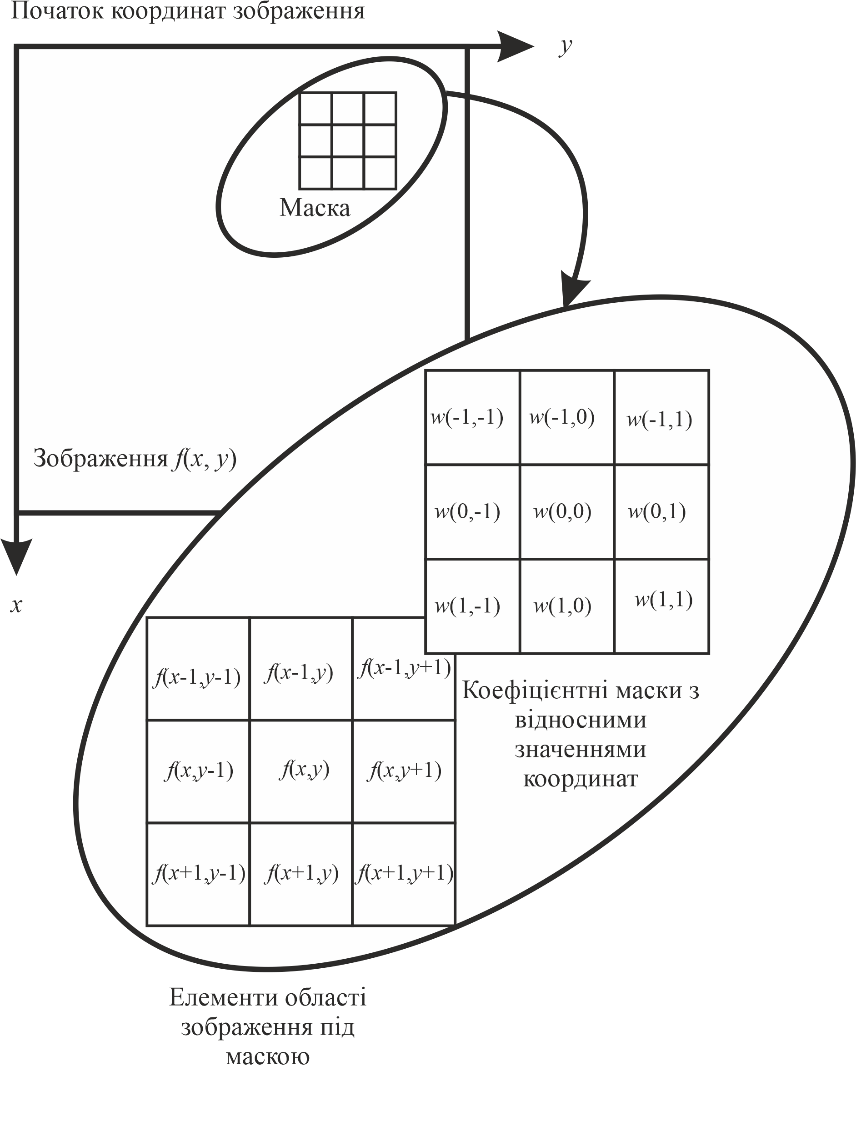


Рисунок 7.12 – Схема роботи градієнтного фільтра

У процесі такого проходу матриця пікселів розміром 3x3 (центральний піксел та 8 прилеглих до нього) поелементно множиться на маску фільтра. Якщо матриця пікселі виходить за межі зображення, її виходять за межі зображення пікселі вважаються рівними нулю.

Отримана на кожному кроці матриця підсумовується за такою формулою [3]:

image

де *R* - відповідний *Gx* або *GY* при використанні градієнтного матриці фільтра, призначеного для осі *OX* або *OY.* Кінцевий піксель обробленого зображення виходить за такою формулою:

image

Для виділення меж використовуються оператори, що апроксимують першу похідну яскравості (градієнт).

У даної лабораторної роботі розглядаються фільтри Робертса, Собеля і Превітта, маски яких мають такий вигляд [4]:

а) Робертса :

по осі OX по осі OY

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | -1 | 0 |
| 0 | 0 | 1 |

|  |  |  |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 0 | -1 |
| 0 | 1 | 0 |

б) Собеля:

по осі OY по осі OX

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

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

в) Превітта:

за осі OX по осі OY

|  |  |  |
| --- | --- | --- |
| -1 | -1 | -1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| -1 | 0 | 1 |
| -1 | 0 | 1 |
| -1 | 0 | 1 |

У модулі IPCV для Scilab реалізована функція edge(img, 'method'), де method може бути 'sobel', 'prewitt', 'canny' та ін.

*// ======================================================*

*// Edge Detection*

*// ======================================================*

clc;

*// 1. Читання вихідного зображення*

image\_filename = 'lena.bmp'; *// Або lena.png, lena.jpg*

*// Перевірка наявності файлу*

if fileinfo(image\_filename) == [] then

disp('Увага: Файл ' + image\_filename + ' не знайдено.');

disp('Спробуємо завантажити стандартне зображення');

*// Спроба завантажити стандартну Лену з бібліотеки (якщо є) або просто створити шлях*

image\_filename = 'lena.png';

if fileinfo(image\_filename) == [] then

disp('ПОМИЛКА: Зображення не знайдено. Будь ласка, додайте файл у папку.');

abort;

end

end

I = imread(image\_filename);

*// 2. Попередня обробка*

*// Функція edge працює коректно лише з напівтоновими зображеннями (Grayscale)*

if size(I, 3) == 3 then

I = rgb2gray(I);

end

*// 3. Виділення кордонів*

*// Метод 'sobel', поріг 0.09 (як у прикладі)*

*// Примітка: у Scilab поріг може працювати трохи інакше залежно від версії IPCV.*

*// Якщо зображення вийде чорним, спробуйте прибрати 0.09, щоб Scilab обрав поріг сам:*

*// BW = edge(I, 'sobel');*

thresh = 0.58;

try

BW = edge(I, 'sobel', thresh);

catch

disp('Помилка виконання edge. Спробуємо без явного порогу...');

BW = edge(I, 'sobel');

end

*// 4. Виведення результатів на екран*

scf(0); clf();

imshow(I);

title('Вихідне зображення');

scf(1); clf();

imshow(BW);

title('Виділення кордонів (Sobel, thresh=' + string(thresh) + ')');

disp('Завдання 3 виконано.');

Результати представлені на рис.7.13.

|  |  |
| --- | --- |
|  |  |
| Рисунок 7.13 – Вхідне зображення | Рисунок 7.14 – Виділення кордонів методом Собеля, thresh = 0.58; |

**7.1.4 Алгоритм детектора меж Канні**

*Детектор Канні* (Canny Edge Detector), розроблений Джоном Канні у 1986 році, вважається одним із найефективніших алгоритмів виявлення меж. На відміну від простих градієнтних фільтрів (Собель, Превітт), які просто реагують на різкі зміни яскравості, метод Канні є багатоступеневим алгоритмом. Викнонуютьс ятакі етапи::

1. згладжування (Gaussian Smoothing): зображення спочатку розмивається фільтром Гауса для видалення шумів. Це запобігає помилковому визначенню точок шуму як країв;
2. пошук градієнтів: використовується оператор Собеля для знаходження інтенсивності та напрямку градієнта в кожній точці;
3. потоншення ліній (Non-maximum Suppression): алгоритм аналізує знайдений градієнт і залишає лише ті пікселі, які є локальними максимумами в напрямку градієнта. Це робить лінії контурів тонкими (шириною в 1 піксель).
4. подвійна порогова фільтрація (Hysteresis Thresholding): використовуються два пороги — верхній (*Thigh*) і нижній (*Tlow*):

* пікселі з градієнтом > *Thigh* вважаються «сильними» межами (точно контур);
* пікселі з градієнтом < *Tlow* відкидаються;
* пікселі між порогами (*Tlow* < *G* < *Thigh*) вважаються «слабкими» і залишаються лише тоді, коли вони з'єднані із "сильними" межами.

Цей метод має вагомі переваги:

* низький рівень помилок,
* гарна локалізація (межі знаходяться там, де вони є насправді)

та отримання чітких тонких ліній.

**7.2 Практична робота**

**Завдання № 1**

1. Використовуючи власно обране зображення людини, за допомогою наведеної у розділі 7.1.1 функції здійсніть його сегментацію методом центроїдного зв'язування (Region Growing). Необхідно виділити особу людини, досягнувши виділення максимально можливої області.
2. За результатами виконання п.1 виведіть на екран відповідний фрагмент напівтонового зображення, використовуючи результат сегментації як маску.
3. Отримані зображення та програмний код помістіть у звіт.

**Завдання № 2**

1. Використовуючи програмний код для квадродерева, здійсніть квадратичне розкладання, власно обраного зображення.
2. Намагайтеся вибрати поріг, що забезпечує найбільш точне виділення основної частини зображення.
3. Використайте функцію imotsu з пакету IPCV для автоматичного визначення оптимального порогу бінаризації зображення (метод Оцу).
4. Виконайте сегментацію зображення за отриманим порогом за допомогою функції im2bw.
5. Порівняйте результати, отримані методом нарощування областей (з Завдання №1) та глобальним методом Оцу. Проаналізуйте, який метод краще впорався з виділенням об'єкта (особи) і чому.
6. Програмний код та отримані зображення (порівняння) помістіть у звіт.

**Завдання № 3**

1. Використовуючи функцію splitmerge, застосуйте метод поділу-злиття на отриманому для цього завдання зображенні.
2. Постарайтеся вибрати параметри matOj, minMean, maxMean таким чином, щоб досягти максимально можливого виділення основного об'єкта зображення.
3. Програмний код і отримані зображення помістіть в звіт.

**Завдання № 4**

1. Не використовуючи стандартну функцію edge (для розрахунку градієнтів), створіть програмну реалізацію фільтрів **Roberts**, **Sobel**, **Prewitt** за допомогою масок алгоритму.
2. Виконання функцій перевірте на заздалегідь отриманих для цього завдання зображеннях.
3. Лістинг програми та отримані зображення помістіть у звіт. У звіті зробіть висновки про фільтри Собеля та Превітта, порівнявши їхню ефективність роботи за отриманими результатами.

**Завдання № 5. Дослідження детектора Канні**

1. Застосуйте детектор Канні (edge з параметром 'canny') до досліджуваного зображення.
2. Експериментально підберіть значення порогів (параметр thresh), щоб отримати максимально деталізований, але чистий контур.
3. Програмно виведіть в одному вікні для порівняння результат роботи фільтра **Собеля** (з Завдання 3) та фільтра **Канні**.
4. Зробіть висновок про різницю в якості контурів (товщина ліній, наявність розривів, шум).

**7.3 Приклад виконання**

*// ==========================================================*

*// ЛАБОРАТОРНА РОБОТА № 7*

*// ==========================================================*

clc;

funcprot(0); *// Вимкнути попередження*

*// 1. ЗАВАНТАЖЕННЯ БІБЛІОТЕКИ*

try

atomsLoad('IPCV');

catch

disp('Увага: Модуль IPCV не завантажено.');

end

*// Завантаження допоміжних функцій*

exec('regiongrow\_scilab.sci', -1);

exec('splitmerge\_scilab.sci', -1);

*// Перевірка наявності зображення*

img\_name = 'lena.bmp';

if fileinfo(img\_name) == [] then img\_name = 'lena.png'; end

if fileinfo(img\_name) == [] then disp('ПОМИЛКА: Файл зображення не знайдено!'); abort; end

I\_orig = imread(img\_name);

if size(I\_orig, 3) == 3 then I\_gray = rgb2gray(I\_orig); else I\_gray = I\_orig; end

disp('Зображення завантажено: ' + img\_name);

*// ==========================================================*

*// ЗАВДАННЯ №1: Region Growing*

*// ==========================================================*

disp('--- Завдання 1: Region Growing ---');

[h, w] = size(I\_gray);

Mask = zeros(h, w);

*// Ставимо точку по центру*

cy = round(h / 2); cx = round(w / 2);

if cy > 0 & cx > 0 then Mask(cy, cx) = 1; end

if (cy + 10 <= h) & (cx + 10 <= w) then Mask(cy + 10, cx + 10) = 1; end

Threshold = 25;

[Res\_RegGrow, count] = regiongrow\_scilab(I\_gray, Mask, Threshold);

scf(0); clf();

subplot(1,3,1); imshow(I\_orig); title('Оригінал');

subplot(1,3,2); imshow(Mask\*255); title('Точки (Seeds)');

subplot(1,3,3); imshow(Res\_RegGrow); title('Region Growing');

*// ======================================================*

*// ЗАВДАННЯ №2: Метод Оцу vs Region Growing*

*// ======================================================*

disp('--- Завдання 2: Метод Оцу vs Region Growing ---');

*// Розрахунок порогу Оцу*

level = 0.5;

try

level = imotsu(I\_gray);

disp('Поріг Оцу: ' + string(level));

catch

level = mean(double(I\_gray)) / 255;

end

*// Бінаризація*

try

BW\_Otsu = im2bw(I\_gray, level);

catch

BW\_Otsu = double(I\_gray)/255 > level;

end

scf(1); clf();

subplot(1, 3, 1); imshow(I\_gray); title('Оригінал');

subplot(1, 3, 2); imshow(Res\_RegGrow); title('Region Growing');

subplot(1, 3, 3); imshow(BW\_Otsu); title('Otsu Threshold');

*// ==========================================================*

*// ЗАВДАННЯ №3: Виділення контурів функцією EDGE*

*// ==========================================================*

disp('--- Завдання 3: Функція edge (Sobel) ---');

thresh\_edge = 0.55;

try

BW\_Edge = edge(I\_gray, 'sobel', thresh\_edge);

catch

BW\_Edge = edge(I\_gray, 'sobel');

end

scf(2); clf();

*// Палітра для коректного відображення (0-чорний, 1-білий)*

f = gcf();

t = (0:255)' / 255;

cmap = [t t t];

f.color\_map = cmap;

imshow(uint8(bool2s(BW\_Edge)\*255));

title('Завдання 3: Sobel (thresh=' + string(thresh\_edge) + ')');

*// ==========================================================*

*// ЗАВДАННЯ №4: Ручні фільтри (Sobel, Prewitt, Roberts)*

*// ==========================================================*

disp('--- Завдання 4: Ручні фільтри ---');

I\_d = double(I\_gray); *// Працюємо з double*

*// ----------------------------------------------------------*

*// А) Фільтр СОБЕЛЯ (Sobel)*

*// ----------------------------------------------------------*

*// Маски 3x3. Коефіцієнт 2 дає згладжування.*

Sx = [-1 0 1; -2 0 2; -1 0 1];

Sy = [-1 -2 -1; 0 0 0; 1 2 1];

Gx\_s = imfilter(I\_d, Sx);

Gy\_s = imfilter(I\_d, Sy);

G\_sobel = sqrt(Gx\_s.^2 + Gy\_s.^2);

*// Нормалізація (щоб картинка була від 0 до 1)*

max\_s = max(G\_sobel); if max\_s == 0 then max\_s = 1; end

G\_sobel = G\_sobel / max\_s;

*// ----------------------------------------------------------*

*// Б) Фільтр ПРЕВІТТА (Prewitt)*

*// ----------------------------------------------------------*

*// Маски 3x3. Схожий на Соболя, але без згладжування (одиниці замість двійок).*

Px = [-1 0 1; -1 0 1; -1 0 1];

Py = [-1 -1 -1; 0 0 0; 1 1 1];

Gx\_p = imfilter(I\_d, Px);

Gy\_p = imfilter(I\_d, Py);

G\_prewitt = sqrt(Gx\_p.^2 + Gy\_p.^2);

max\_p = max(G\_prewitt); if max\_p == 0 then max\_p = 1; end

G\_prewitt = G\_prewitt / max\_p;

*// ----------------------------------------------------------*

*// В) Фільтр РОБЕРТСА (Roberts)*

*// ----------------------------------------------------------*

*// Маски 2x2. Діагональні перепади. Найчутливіший до шуму.*

Rx = [1 0; 0 -1];

Ry = [0 1; -1 0];

Gx\_r = imfilter(I\_d, Rx);

Gy\_r = imfilter(I\_d, Ry);

G\_roberts = sqrt(Gx\_r.^2 + Gy\_r.^2);

max\_r = max(G\_roberts); if max\_r == 0 then max\_r = 1; end

G\_roberts = G\_roberts / max\_r;

*// ----------------------------------------------------------*

*// ВІДОБРАЖЕННЯ РЕЗУЛЬТАТІВ (Порівняння)*

*// ----------------------------------------------------------*

scf(3); clf();

f3 = gcf(); f3.color\_map = cmap; *// Та сама ч/б палітра*

*// Виводимо всі 4 картинки разом*

subplot(2,2,1);

imshow(I\_orig);

title('Оригінал');

subplot(2,2,2);

imshow(G\_sobel);

title('Sobel (Ручний)');

subplot(2,2,3);

imshow(G\_prewitt);

title('Prewitt (Ручний)');

subplot(2,2,4);

imshow(G\_roberts);

title('Roberts (Ручний)');

disp('Всі завдання (1-4) успішно завершено.');

*// ==========================================================*

*// ЗАВДАННЯ №5: Детектор Канні (Canny Edge Detector)*

*// ==========================================================*

disp('--- Завдання 5: Детектор Канні ---');

*// 1. Підготовка (використовуємо I\_gray з попередніх кроків)*

*// Нормалізація важлива для коректної роботи порогів*

I\_norm = double(I\_gray) / 255;

*// 2. Фільтр СОБЕЛЯ (для порівняння)*

*// Ми вже робили це, але повторимо для наочності поруч*

BW\_Sobel = edge(I\_norm, 'sobel');

*// 3. Фільтр КАННІ (Canny)*

*// edge(I, 'canny', [low\_thresh, high\_thresh])*

*// Підібрані параметри: нижній поріг 0.05, верхній 0.1*

*// Можна змінювати їх для регулювання деталізації*

thresh\_canny = [0.05, 0.2];

try

BW\_Canny = edge(I\_norm, 'canny', thresh\_canny);

catch

*// Якщо параметри не приймаються, запускаємо авто-режим*

disp('Автоматичні налаштування для Canny...');

BW\_Canny = edge(I\_norm, 'canny');

end

*// 4. ВІДОБРАЖЕННЯ ПОРІВНЯННЯ*

scf(4); clf();

*// Налаштування палітри (0-Чорний, 1-Білий)*

f = gcf();

t = (0:255)' / 255;

cmap = [t t t];

f.color\_map = cmap;

*// Ліворуч - Собель*

subplot(1, 2, 1);

*// bool2s перетворює True/False в 1/0, \*255 робить білим*

imshow(uint8(bool2s(BW\_Sobel) \* 255));

title('Фільтр Собеля (Sobel)');

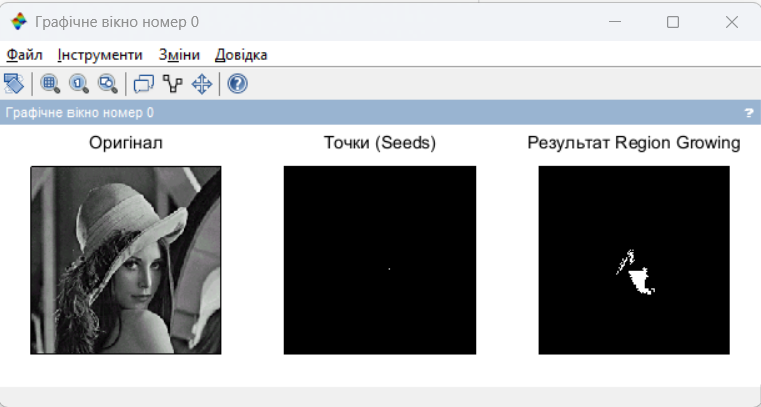
*// Праворуч - Канні*

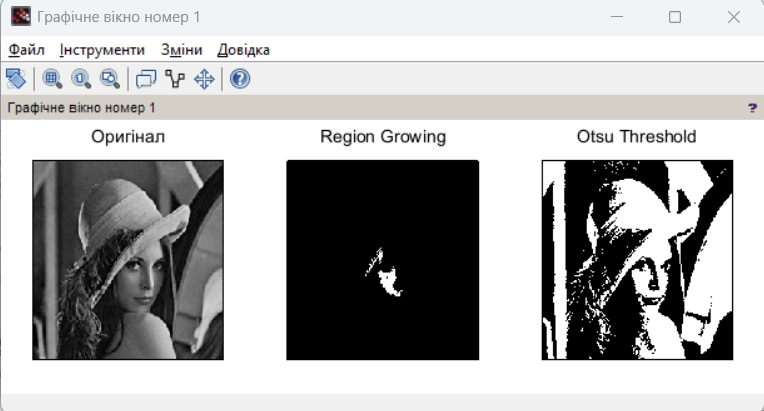
subplot(1, 2, 2);

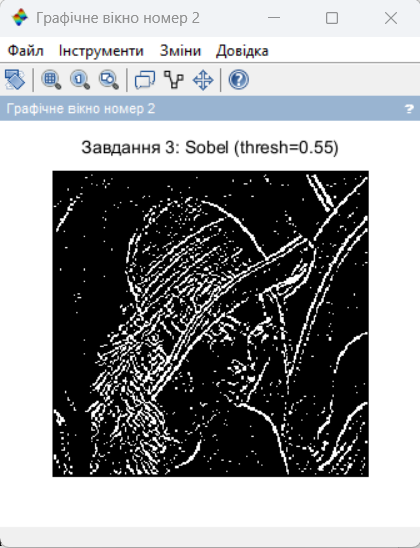
imshow(uint8(bool2s(BW\_Canny) \* 255));

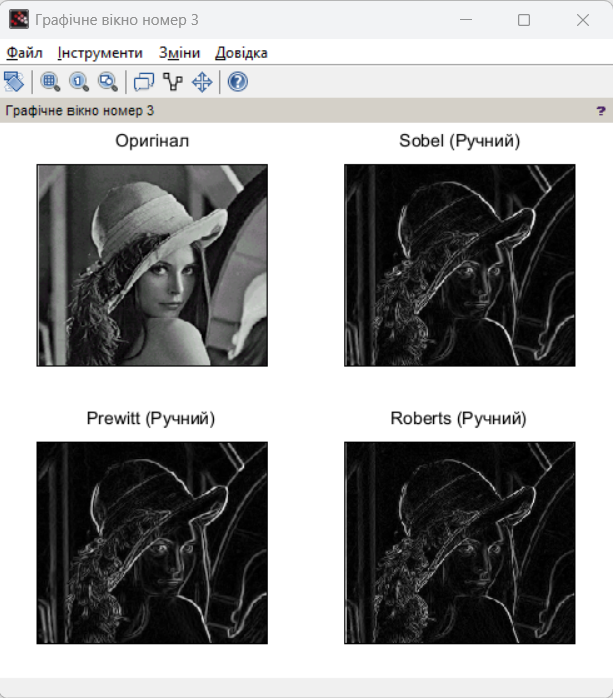
title('Фільтр Канні (Canny)');

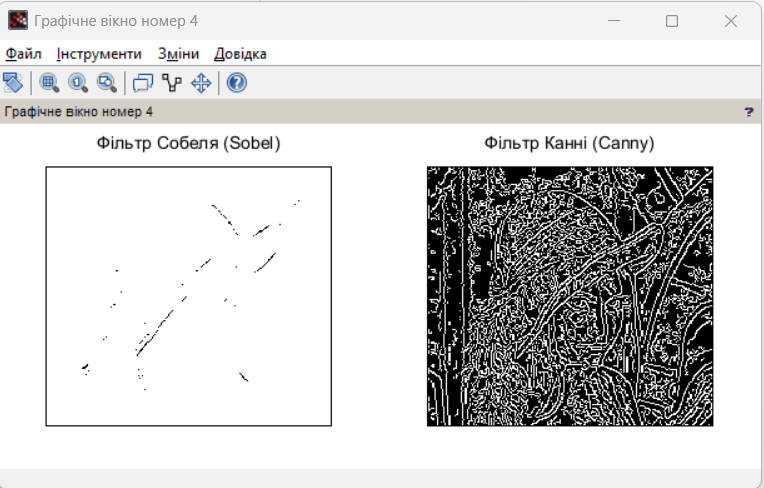
disp('Завдання 5 виконано. Порівняйте товщину ліній.');











**7.4 Індивідуальне завдання**

Номер варіанта визначається за останньою цифрою номера студента у списку групи (або за вказівкою викладача). Якщо остання цифра 0, обирається варіант 10.

**Варіант 1, 6**

1. **Region Growing:** Використовуючи функцію regiongrow\_scilab, виділіть на зображенні lena.png **ліве око** (необхідно підібрати координати стартової точки Mask та поріг Threshold так, щоб виділилося око, але не все обличчя).
2. **Ручні фільтри:** Реалізуйте виділення контурів методом **Собеля (Sobel)**, але виведіть на екран **тільки вертикальні межі** (використовуйте тільки маску $S\_x$ без об'єднання з $S\_y$).
3. **Фільтр Канні:** Дослідіть вплив верхнього порогу. Зафіксуйте нижній поріг на рівні 0.05, а верхній змініть у діапазоні від 0.1 до 0.4.
4. **Аналіз:** Порівняйте результат ручного фільтра Собеля з повним фільтром та поясніть, як зміна верхнього порогу в методі Канні впливає на кількість знайдених контурів.

**Варіант 2, 7**

1. **Region Growing:** Виділіть на зображенні **фон** (суцільну область зліва або справа від Лени). Встановіть стартову точку (seed) у кутку зображення. Підберіть поріг так, щоб фон не "затік" на капелюх чи обличчя.
2. **Ручні фільтри:** Реалізуйте виділення контурів методом **Превітта (Prewitt)**, але виведіть на екран **тільки горизонтальні межі** (використовуйте тільки маску $P\_y$).
3. **Фільтр Канні:** Порівняйте "зв'язність" ліній. Знайдіть ділянку зображення (наприклад, контур плеча або капелюха), де звичайний фільтр Собеля дає розриви (пунктир), а фільтр Канні створює суцільну лінію.
4. **Аналіз:** Поясніть, чому фільтр Канні краще з'єднує лінії (завдяки гістерезису), та які деталі зникли при використанні тільки горизонтальної маски Превітта.

**Варіант 3, 8**

1. **Region Growing:** Виділіть **світлу смугу на капелюсі** (стрічку). Це вимагатиме встановлення точки на стрічці та досить вузького діапазону порогу (Threshold), щоб не захопити решту капелюха.
2. **Ручні фільтри:** Реалізуйте фільтр **Робертса (Roberts)**. Оскільки він чутливий до шумів, спробуйте попередньо додати до зображення слабкий шум (функція imnoise або rand), і порівняйте результат роботи фільтра Робертса та Собеля на ньому.
3. **Фільтр Канні:** Застосуйте фільтр Канні до того ж зашумленого зображення.
4. **Аналіз:** Який з трьох фільтрів (Робертс, Собель, Канні) найкраще впорався з шумом і чому?

**Варіант 4, 9**

1. **Region Growing:** Виділіть **волосся** (текстурну область). Спробуйте використати декілька стартових точок (seeds) одночасно (наприклад, 3-4 точки в різних частинах зачіски), щоб покрити більшу область.
2. **Ручні фільтри:** Використайте фільтр **Собеля**. Змініть код так, щоб результат був **інвертованим** (білий фон, чорні лінії) без використання функції edge (тільки математичними операціями з матрицями $S\_x, S\_y$ та інверсією значень).
3. **Фільтр Канні:** Дослідіть товщину ліній. Зробіть "зум" (відобразіть лише частину матриці зображення) на чіткому контурі та порівняйте ширину лінії в пікселях у методу Собеля та методу Канні.
4. **Аналіз:** Як впливає нормалізація на відображення тонких ліній та який метод дає тонший контур?

**Варіант 5, 10**

1. **Region Growing:** Виділіть найтемнішу ділянку (наприклад, **тінь** під полями капелюха або чорну область фону, якщо є).
2. **Ручні фільтри:** Реалізуйте **діагональний** пошук контурів. Для цього створіть та застосуйте власну маску (на базі Превітта або Собеля), яка реагує на зміни під кутом 45 градусів (наприклад, маска, де ненульові елементи розташовані по діагоналі: [0 1; -1 0]).
3. **Фільтр Канні:** Спробуйте налаштувати пороги фільтра Канні так, щоб він ігнорував дрібну текстуру (пір'я, волосся), але залишав тільки **головний силует** (контур обличчя та капелюха).
4. **Аналіз:** Наведіть матрицю (маску), яку ви використали для діагонального пошуку, та значення порогів для фільтра Канні.

**7.5 Зміст звіту**

1. Титульний аркуш.
2. Тема та мета роботи.
3. Лістинг функції regiongrow\_scilab.
4. Код виконання практичних завдань (скрипт).
5. Скриншоти отриманих результатів (початкове зображення, маска точок, результати сегментації, результати фільтрації).
6. Висновки щодо ефективності різних методів сегментації та фільтрів.
7. Відповіді на контрольні питання.

**7.6 Контрольні питання**

1. Що таке сегментація зображення і які її основні завдання?
2. Поясніть принцип роботи методу нарощування областей (Region Growing).
3. Яку роль відіграє поріг (Threshold) у методах сегментації?
4. У чому відмінність між операторами Собеля та Превітта?
5. Чому фільтр Канні (Canny) вважається більш ефективним, ніж звичайні градієнтні фільтри?
6. Як впливає шум на роботу градієнтних фільтрів?