МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП’ЮТЕРНИХ НАУК

СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОПОЛОГІЙ ПРОЕКТУВАННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до курсової роботи

з дисципліни

«Методи і засоби візуалізації»

Виконав студент групи ІТ.мз-62с

Тиченко Я.М.

Перевірила Нагорний В.В.

Суми 2017

# Зміст

[Зміст 2](#_Toc502210411)

[Вступ 3](#_Toc502210412)

[1 Реалізація програми 4](#_Toc502210413)

[1.1 Згортка та лінійні фільтри 4](#_Toc502210414)

[1.2 Згладжування зображень 5](#_Toc502210415)

[1.3 Дилатація та ерозія 6](#_Toc502210416)

[1.4 Оператор Собеля 9](#_Toc502210417)

[1.5 Оператор Лапласа 11](#_Toc502210418)

[1.6 Детектор ребер Канні 12](#_Toc502210419)

[1.7 Визначення гістограм 14](#_Toc502210420)

[2 Тестування додатку 17](#_Toc502210421)

[Висновки 23](#_Toc502210422)

[Література 24](#_Toc502210423)

[Додаток А — лістинг програмного коду 25](#_Toc502210424)

# Вступ

В сучасності аналіз візуальної інформації програмними засобами має дуже багато застосувань у багатьох сферах людської діяльності. Доступ до використання мають не тільки великі корпорації або компанії, але є засоби з відкритим кодом, користуватися яким може кожен бажаючий. OpenCV є найпотужнішою із таких засобів.

OpenCV — бібліотека функцій та алгоритмів комп'ютерного зору, обробки зображень і чисельних алгоритмів загального призначення з відкритим кодом. Бібліотека надає засоби для обробки і аналізу вмісту зображень, у тому числі розпізнавання об'єктів на фотографіях (наприклад, осіб і фігур людей, тексту тощо), відстежування руху об'єктів, перетворення зображень, застосування методів машинного навчання і виявлення загальних елементів на різних зображеннях. Бібліотека розроблена Intel і нині підтримується Willow Garage та Itseez. Сирцевий код бібліотеки написаний мовою C++ і поширюється під ліцензією BSD. Біндинги підготовлені для різних мов програмування, таких як Python, Java, Ruby, Matlab, Lua та інших. Може вільно використовуватися в академічних та комерційних цілях.

Метою даної роботи є розробка консольного додатку для обробки фото, а також здобуття навичок роботи із бібліотекою функцій та алгоритмів комп'ютерного зору OpenCV.

# 1 Реалізація програми

## Згортка та лінійні фільтри

Лінійні фільтри – сімейство найпростіших фільтрів зображень з точки зору математичного опису математичного опису . Припустимо, що ми маємо зображення з пів тону, тоді будь-який лінійний фільтр визначається речовинною функцією F. Данa функція називається ядром фільтру.

I'(x,y)=\sum_i \sum_j F(i,j) \cdot I(x+i,y+j)

Рисунок 1.1 — ядро лінійного фільтру.

Як правило, ядро ​​фільтра застосовується до деякої околиці O точки, тому межі зміни індексів *i* та *j* визначаються обраної формою і розміром околиці. Дана околиця в деяких джерелах називається шаблоном або апертурою. У процесі обчислення згортки виконується прохід по пікселях всього зображення, шаблон накладається на кожен поточний піксель за допомогою поєднання пікселя з конкретною точкою шаблону - провідною позицією шаблону, після чого обчислюється згортка. Необхідно окремо звернути увагу на ситуацію, коли поточний піксель знаходиться на кордоні зображення.

Для обчислення згорток в бібліотеці OpenCV присутня функція filter2D.

*void filter2D (const Mat & src, Mat & dst, int ddepth, const Mat & kernel,*

*Point anchor = Point (-1, -1), double delta = 0,*

*int borderType = BORDER\_DEFAULT)*

Розглянемо докладніше параметри наведеної функції.

*src - вихідне зображення.*

*dst - згортка. Має таку ж кількість каналів і глибину, що і вихідне зображення.*

*ddepth - глибина результуючого зображення. Якщо на вхід функції передано від'ємне значення, то глибина збігається з глибиною вхідного зображення.*

*kernel - ядро ​​згортки, одноканальна речова матриця.*

*anchor - провідна позиція ядра. За замовчуванням приймає значення (-1, -1), яке означає, що провідна позиція розташована в центрі ядра.*

*delta - константа, яка може бути додана до значення інтенсивності після фільтрації перед безпосереднім записом результату.*

*borderType - параметр, що визначає метод доповнення кордону, щоб можна було застосовувати фільтр до граничних пикселям вихідного зображення. Приймає будь-яке значення виду BORDER\_ \* за винятком BORDER\_TRANSPARENT і BORDER\_ISOLATED.*

В нашому додатку виконання лінійного фільтру буде виконуватися у функції applyLinearFilter. Код реалізованої функції буде відображено у лістингу додатку.

## Згладжування зображень

Згладжування або розмиття зображення - це одна з найпростіших і часто використовуваних операцій обробки зображень. Як правило, розмиття застосовується, щоб зменшити шум або артефакти, які обумовлені вибором камери. Також згладжування грає важливу роль при необхідності зменшити дозвіл зображення і отримати піраміду зображень різного масштабу (image pyramids). Розробники OpenCV реалізували кілька функцій розмиття зображення. Далі зупинимося на деяких з них, а саме розглянемо функції blur, GaussianBlur і medianBlur:

*void blur(const Mat& src, Mat& dst, Size ksize,*

*Point anchor=Point(-1, -1),*

*int borderType=BORDER\_DEFAULT)*

*void GaussianBlur(const Mat& src, Mat& dst, Size ksize,*

*double sigmaX, double sigmaY=0,*

*int borderType=BORDER\_DEFAULT)*

*void medianBlur(const Mat& src, Mat& dst, int ksize)*

Наведемо призначення загальних параметрів зазначених функцій:

src - вихідне зображення.

dst - результуюче зображення, має такий же розмір і тип, як і вихідне зображення.

kSize - розмір ядра для розмиття.

anchor - провідний елемент ядра. За замовчуванням приймає значення (-1, -1), провідний елемент збігається з центром ядра.

borderType - спосіб доповнення кордону. Функція blur виконує розмиття за допомогою обчислення згортки вихідного зображення з ядром.

В нашому додатку згладжування буде виконуватися у функціях applyMedianBlur, applyGaussianBlur, applyBlur. Код реалізованих функцій буде відображено у лістингу додатку.

## Дилатація та ерозія

Дилатація (морфологічний розширення) - згортка зображення або виділеної області зображення з деяким ядром. Ядро може мати довільну форму і розмір [1, 2]. При цьому в ядрі виділяється єдина провідна позиція (anchor), яка поєднується з поточним пикселем при обчисленні згортки. У багатьох випадках в якості ядра вибирається квадрат або коло з провідною позицією в центрі. Ядро можна розглядати як шаблон або маску. Застосування дилатації зводиться до проходу шаблоном по всьому зображенню і застосування оператора пошуку локального максимуму до интенсивностям пікселів зображення, які накриваються шаблоном. Така операція викликає зростання світлих областей на зображенні (рис.9.3, c). На малюнку сірим кольором відзначені пікселі, які в результаті застосування дилатації будуть білими.

Ерозія (морфологічний звуження) - зворотна операція. Дія ерозії подібно дилатації, різниця лише в тому, що використовується оператор пошуку локального мінімуму (рис.9.3, d), сірим кольором залиті пікселі, які стануть чорними в результаті ерозії.

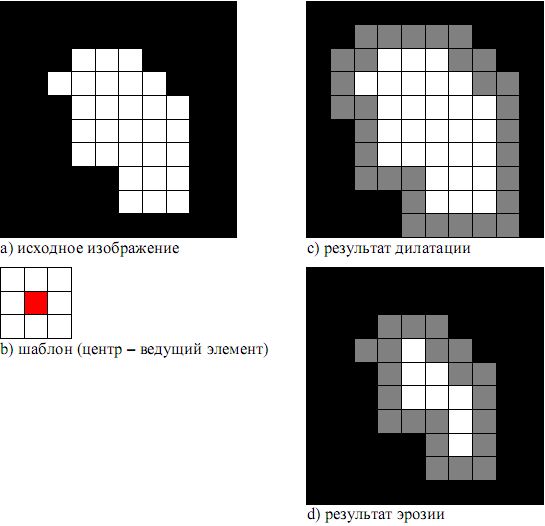


Рисунок 1.2 – Приклад дилатації та ерозії

Розглянемо прототипи відповідних функцій ерозії і дилатації, реалізованих в OpenCV.

*void dilate (const Mat & src, Mat & dst, const Mat & element,*

*Point anchor = Point (-1, -1), int iterations = 1, int borderType = BORDER\_CONSTANT,*

*const Scalar & borderValue =*

*morphologyDefaultBorderValue ())*

*void erode (const Mat & src, Mat & dst, const Mat & element,*

*Point anchor = Point (-1, -1), int iterations = 1,*

*int borderType = BORDER\_CONSTANT,*

*const Scalar & borderValue =*

*morphologyDefaultBorderValue ())*

Параметри:

src - вихідне зображення.

dst - результуюче зображення, має такий же розмір, що і вхідне зображення. Відзначимо, що результат операції може записуватися в вихідне зображення.

element - шаблон, який використовується в процесі дилатації. Якщо element = Mat (), то застосовується квадратний шаблон розміром 3x3.

anchor - позиція провідного пікселя в структурному елементі. Значення за замовчуванням (-1, -1) означає, що в якості ведучого елемента вибирається центр шаблону.

interations - кількість разів, яке застосовується дилатація / ерозія.

borderType - параметр, що визначає метод доповнення кордону, щоб можна було застосовувати дилатацію / ерозію до граничних пикселям вихідного зображення. Приймає будь-яке значення виду BORDER\_ \* за винятком BORDER\_TRANSPARENT і BORDER\_ISOLATED.

borderValue - розмір кордону в разі, якщо вона має постійний розмір. Значення за замовчуванням дорівнює morphologyDefaultBorderValue, перетворюється в -inf для дилатації і + inf для ерозії. При використанні значення за замовчуванням операція застосовується тільки до внутрішніх пикселям зображень.

В нашому додатку дилатація та єрозія буде виконуватися у функціях applyErode, applyDilate. Код реалізованих функцій буде відображено у лістингу додатку.

## Оператор Собеля

Оператор Собеля - дискретний диференціальний оператор, який обчислює наближені значення похідних різного порядку для функції яскравості пікселів [1]. Найбільш поширеним прикладом практичного використання є визначення меж (ребер) об'єктів на зображенні, тобто точок різкої зміни яскравості.

Даний оператор заснований на згортку зображення з цілочисельними фільтрами. У найпростішому випадку оператор побудований на обчисленні згорток вихідного зображення з ядрами G\_x і G\_y, що забезпечують обчислення перших похідних за напрямками, показаними на рисунку 1.3:

G_x=\begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1
\end{bmatrix},G_y=\begin{bmatrix}
-1 & -2 & 1 \\
0 & 0 & 0 \\
1 & 2 & 1
\end{bmatrix}

Рисунок 1.3 – значення ядер згорток

Даний оператор використовується для наближеного обчислення градієнта функції інтенсивності пікселів. Застосування оператора G\_x дозволяє визначити наближене значення першої похідної зміни інтенсивності в горизонтальному напрямку, G\_y - у вертикальному.

У бібліотеці OpenCV підтримується обчислення перших, других, третіх і змішаних похідних функції інтенсивності пікселів з використанням розширеного оператора Собеля . Нижче наведено прототип відповідної програмної функції.

*void Sobel(const Mat&src, Mat&dst, int ddepth,*

*int xorder, int yorder, int ksize=3,*

*double scale=1, double delta=0,*

*int borderType=BORDER\_DEFAULT)*

Перерахуємо вхідні параметри функції Sobel:

src - вихідне зображення.

dst - результуюче зображення.

ddepth - глибина результуючого зображення.

xorder - порядок похідної по осі Ox.

yorder - порядок похідної по осі Oy.

ksize - розмір розширеного ядра оператора Собеля. Приймає одне зі значень 1, 3, 5 або 7. У всіх випадках ядро ​​має розмір kSize x kSize, крім ситуації, коли kSize = 1. При kSize = 1 ядра мають розмір 3x1 або 1x3, по суті застосовується фільтр Гаусса. Вказане значення можна використовувати тільки при обчисленні перших і других приватних похідних по осях Ox і Oy. За замовчуванням ядро ​​має розмір 3x3. Відзначимо, що додатково передбачено спеціальне значення параметра kSize = CV\_SCHARR = -1, який відповідає ядру розміру 3x3 фільтра Щарри (Scharr) і може давати більш точні оцінки похідних в порівнянні з оператором Собеля:

G_x=\begin{bmatrix}
-3 & 0 & 3 \\
-10 & 0 & 10 \\
-3 & 0 & 3
\end{bmatrix},G_y=\begin{bmatrix}
-3 & -10 & -3 \\
0 & 0 & 0 \\
3 & 10 & 3
\end{bmatrix}

scale - опціональний параметр, який задає коефіцієнт масштабування для обчислюваних значень похідних. За замовчуванням масштабування не застосовується.

delta - опціональний параметр зміщення інтенсивності, додається перед збереженням результату в матрицю dst.

borderType - параметр, що визначає метод доповнення кордону.

В нашому додатку оператор Собеля буде виконуватися у функції applySobel. Код реалізованої функції буде відображено у лістингу додатку.

## Оператор Лапласа

Математично оператор Лапласа представляє суму квадратів друге приватних похідних функції відображеної на рисунку 1.4 . Дискретний аналог оператора Лапласа використовується при обробці зображень, зокрема, для визначення ребер об'єктів на зображенні. Ребра формуються з безлічі пікселів, в яких оператор Лапласа набуває нульових значень, тому що нулі других похідних функції відповідають екстремальним перепадів інтенсивності.

\Delta f=\frac {\delta^2f} {\delta x^2} + \frac {\delta^2f} {\delta y^2}

Рисунок 1.4 – оператор Лапласа

Бібліотека OpenCV містить функцію Laplacian, що забезпечує обчислення оператора Лапласа.

*void Laplacian (const Mat & src, Mat & dst, int ddepth,*

*int ksize = 1, double scale = 1, double delta = 0,*

*int borderType = BORDER\_DEFAULT)*

Параметри функції:

src, dst, ddepth, scale, delta, borderType мають таке ж значення, що і при виконанні функції Sobel.

kSize - розмір апертури для обчислення другої похідної, є позитивним парним числом. При використанні значення за замовчуванням kSize = 1 застосовується апертура розміром 3x3 і ядро ​​являв собою таблицю на рисунку 1.5

L=\begin{bmatrix}
0 & 1 & 0 \\
1 & -4 & 1 \\
0 & 1 & 0
\end{bmatrix}

Рисунок 1.5 – апертура розміром 3x3

Перед безпосереднім застосуванням оператора виконується згладжування з використанням фільтра Гаусса (GaussianBlur) і перетворення вихідного зображення у відтінки сірого (cvtColor). Зауважимо, що результуючі значення оператора Лапласа записуються в матрицю, глибина якої аналогічно прикладу з фільтром Собеля відрізняється від глибини вихідного зображення, тому перш ніж можна буде виконується перетворення отриманої матриці в 8-бітну целочисленную за допомогою виклику функції convertScaleAbs.

В нашому додатку оператор Лапласа буде виконуватися у функції applyLaplacian. Код реалізованої функції буде відображено у лістингу додатку.

## 1.6 Детектор ребер Канні

Детектор ребер Канні призначений для пошуку меж об'єктів на зображенні. Детектор будується на підставі оператора Собеля і включає кілька етапів.

1. Видалення шуму на зображенні за допомогою застосування фільтра Гаусса з ядром розміру 5.

K= \frac {1} {159}L=\begin{bmatrix}
2 & 4 & 5  & 4  & 2 \\
4 & 9 & 12  & 9  & 4 \\
5 & 12 & 15  & 12  & 5 \\
4 & 9 & 12  & 9  & 4 \\
2 & 4 & 5  & 4  & 2
\end{bmatrix}

Рисунок 1.6 – фільтра Гаусса з ядром розміру 5

2. Визначення перших похідних (магнітудів та напрямків) функцій інтенсивності пікселів по горизонталі і вертикальному напряму за допомогою застосування оператора Собеля з ядрами G\_x та G\_y (див. Розділ 2.4). Направлення градієнтів обертаються до одного з можливих значень 0 ^ 0,45 ^ 0,90 ^ 0,135 ^ 0.

3. Відбір пікселей, які потенційно відносяться до ребра з використанням процедури непровіренного придушення . Пікселі, які відповідають вектору похідних по напрямках, що є локальними максимумами, вважаються потенційними кандидатами на принадлежність ребру.

4. Двойне відокремлення (гістерезис). Виділяються "сильні" та "слабкі" ребра. Пікселі, інтенсивність яких перевищує максимальний поріг, вважаються пікселями, що належать "сильним" ребрам. Приймається, що пікселі з інтенсивністю, входящій в інтервал від мінімального до максимального порогової значення, відносяться до "слабким" ребрам. Пікселі, інтенсивність яких менше меншого мінімуму, відкидаються з подальшого розгляду. Результующіе ребра містять пікселі всіх "сильних" ребер і ті пікселі "слабких" ребер, які містять принаймні один піксель "сильних" ребер.

Детектор Канні реалізується в бібліотеці OpenCV у вигляді окремої функції, прототип якого наведено далі.

*void Canny(const Mat&image, Mat&edges, double threshold1,*

*double threshold2, int apertureSize=3,*

*bool L2gradient=false)*

Функція приймає на вхід наступні параметри:

image - одноканальне 8-бітное зображення.

edges - результуюча карта ребер, представляється матрицею, розмір якої збігається з розміром вихідного зображення.

порsг1, порsг 2 - параметри алгоритму, порогове значення для відсічення.

apertureSize - розмір апертури для застосування оператора Собеля.

L2gradient - прапорець, який вказує, за якою нормою буде обчислити градієнт магнітуди. Приймає істинне значення, якщо використовується норма L\_2 (корінь квадрата з суми квадратів приватних похідних), в іншому випадку L\_1 (сума модулів приватних похідних). Як правило, норми L\_1 досить, і виявляється вона швидше в зв'язку з відсутністю викликів функції sqrt.

В нашому додатку оператор Собеля буде виконуватися у функції applyCanny. Код реалізованої функції буде відображено у лістингу додатку.

## 1.7 Визначення гістограм

Один з найбільш поширених дефектів фотографічних, сканерних та телевізійних зображень - слабкий контраст. Дефект багато в чому обумовлений обмеженістю діапазону відтворюваних яскравостей. Під контрастом розуміється різниця максимального та мінімального значення яскравостей. Контрастність зображення можна збільшити за рахунок зміни яскравості кожного елемента зображення і збільшення діапазону яскравостей. Існує декілька методів, основаних на вирахуванні гістограми.

Допустимо, що має зображення в відтінках сірого, інтенсивність пікселів якої змінюється в межах значень від a до b.

Для зображення можна побудувати гістограму зі стовпцями, що відповідають кількостям пікселів певної інтенсивності. Такого роду гістограма дозволяє представити розподіл відтінків на зображенні. В загальному випадку під гістограмою розуміється колекція цілих значень, кожна з яких визначає кількість точок, що володіють деякою властивістю або що належать певній межі. У практиці гістограми застосовуються, щоб отримати статистичну картину розподілу будь-яких даних (пікселів, векторів ознак, напрямків градієнта у всіх точках зображення тощо).

Нижче приведені прототипи доступних функцій.

*void calcHist(const Mat\* arrays, int narrays,*

*const int\* channels, const Mat&mask,*

*MatND& hist, int dims, const int\* histSize,*

*const float\*\* ranges, bool uniform=true,*

*bool accumulate=false)*

*void calcHist(const Mat\* arrays, int narrays*,

*const int\* channels, const Mat&mask,*

*SparseMat&hist, int dims,*

*const int\* histSize, const float\*\* ranges,*

*bool uniform=true, bool accumulate=false)*

Параметри:

arrays - вихідні масиви даних або зображення. Повинні мати однакову глибину (CV\_8U або CV\_32F) і розмір.

narrays - кількість вихідних масивів даних.

канали - масивний індекс каналів у кожній вхідній масиві, за якою буде розрахована гістограма.

mask - маска, на якій вважається гістограма. Опціональний параметр. Якщо маска не пуста, то вона представляється 8-бітовою матрицею того ж розміру, що і кожний вихідний масив. При побудові гістограми враховуються лише елементи масивів, які відповідають ненулевим елементам маски. Якщо маска пуста, то побудова гістограми виконується при повному наборі даних.

hist - результуюча гістограма, плотна в разі використання першого прототипу функції, розсічена - у випадку другого. Для хранения щільної гістограми використовується структура даних MatND, для розсіченої - SparseMat. MatND представляється у вигляді n-мірного масиву, SparseMat - хеш-таблицею ненульових значень.

dims - розмірність гістограми. Параметр приймає позитивні цілі значення, не перевищують CV\_MAX\_DIMS = 32.

histSize - кількість бінов по кожній розмірності гістограм.

діапазони - інтервали зміни значень по кожній розмірності гістограм. Якщо гистограма рівномірна (uniform = true), то для будь-якого розміру і досить вказати тільки нижню межу зміни (по суті значення, що відповідає першій), верхня межа буде співпадати з histSize [i] - 1.

uniform - флаг, який визначає тип діаграм (равномірна або ні).

*accumulate* - прапор, що вказує на необхідність очищення гістограм перед безпосередніми розрахунками. Використання даного флага дозволяє використовувати одну й ту ж гістограму для декількох множин масивів або оновлювати часову діаграму.

В нашому додатку робота з гістограмами буде виконуватися у функціях applyСalcHist, applyEqualizeHist. Код реалізованої функції буде відображено у лістингу додатку.

# Тестування додатку

Стартова сторінка та вибір зображення для операцій над ним показано на рисунку 2.1.

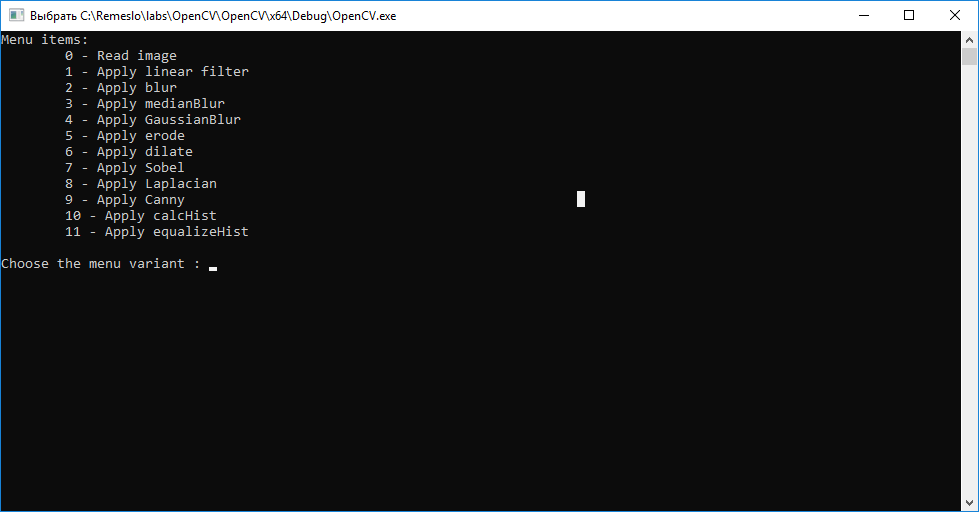


Рисунок 2.1 – фільтра Гаусса з ядром розміру 5

Після вдалого завантаження картинки, протестуємо лінійний фільтр, результат на рисунку 2.2. Зліва показане зображення без дій на ним, справа — результат роботи функції.

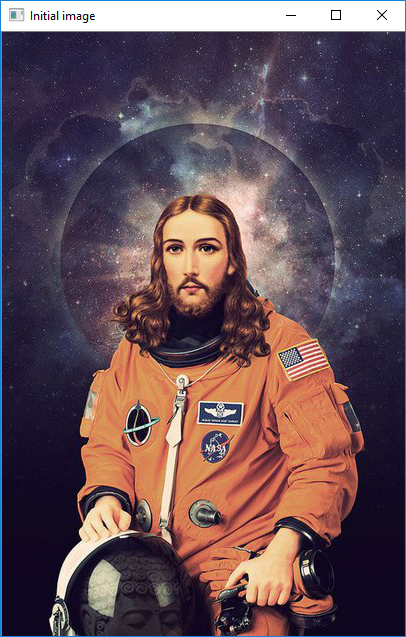
 

Рисунок 2.2 – результат роботи лінійного фільтру

Результат роботи згладжування різними функціями показано на рисунках 2.3, 2.4, та 2.5.



Рисунок 2.3 – звичайне згладжування



Рисунок 2.4 – медіанне згладжування



Рисунок 2.5 – Гаусовське згладжування

Результат роботи еродії та дилатації функціями показано на рисунках 2.6, 2.7.

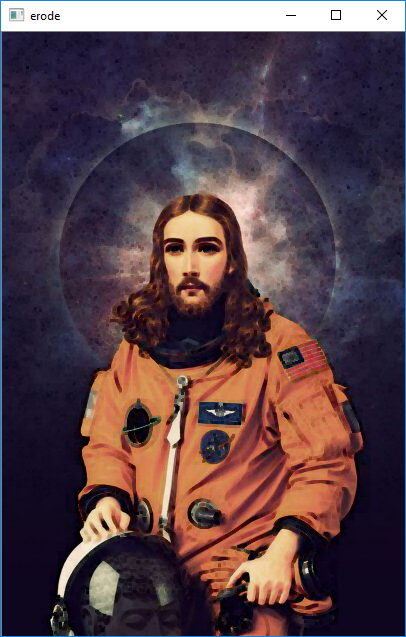


Рисунок 2.6 – результат функції еродії



Рисунок 2.7 – результат функції дилатації

Результати роботи функцій операторів Лапласа, Собеля та Канні показано на рисунках 2.8, 2.9, 2.10 відповідно



Рисунок 2.8 – результат функції оператора Собеля

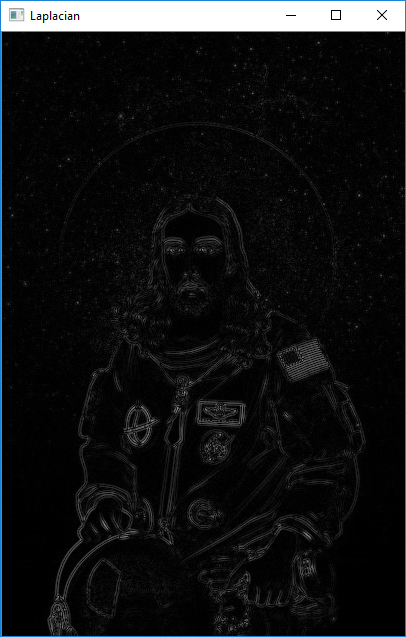


Рисунок 2.9 – результат функції оператора Лапласа



Рисунок 2.10 – результат функції оператора Канни

Результат визначення гістограм показано на рисунках 2.11 та 2.12.

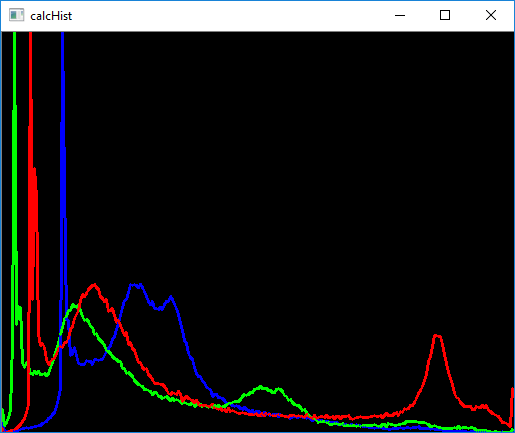


Рисунок 2.11 – гістограми за допомогою функції calcHist

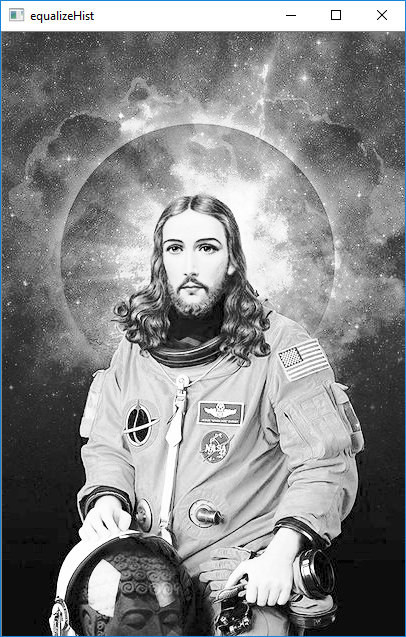


Рисунок 2.11 – наведення чіткості за допомогою гістограми та функції equalizeHist

# Висновки

В ході виконання курсової роботи роботи було розглянуто основні можливості OpenCV для роботи із зображеннями, деякі методи було використано та розглянута їх математична модель та принцип дії.

Також, було виконане практичне завдання з використанням OpenCV та С++, результатом якого є невеличка консольна програма для роботи із зображенням. Більшість параметрів для функцій роботи із зображенням вказані за замовчуванням, задля більшої зручності.

В цілому, OpenCV - це приголомшливий опенсорсний проект. Його можна використовувати безкоштовно для розробки додатків, які можуть знайти застосування в дуже багатьох сфера використання.

# Література

1. Разработка мультимедийных приложений с использованием библиотек OpenCV [Електронний ресурс] – Режим доступу до ресурсу https://www.intuit.ru/studies/courses/10622/1106/lecture/18030?page=7
2. Приклади об'єктно-орієнтованого проектування / Е. Гамма, Р. Хелм, Р. Джонсон, Дж. Вліссідес. – Санкт-Петербург: Питер, 2001. – 344 с.
3. Паттерны проектирования.Структурные паттерны [Електронний ресурс] – Режим доступу до ресурсу: <http://faq.linemedia.ru/patternyi-pro/>
4. OpenCV By Example [Електронний ресурс] – Режим доступу до ресурсу: <https://addyosmani.com/resources/essentialjsdesignpatterns/book/#OpenCVin>Nodejs
5. OpenCV with Python By Example [Електронний ресурс] – Режим доступу до ресурсу: http:/[crovefed.ru](http://faq.linemedia.ru/patternyi-pro/)
6. OpenCV 3.0 Computer Vision with Java [Електронний ресурс] – Режим доступу до ресурсу: <http://aldo.com/VC/>
7. OpenCV Computer Vision with Python [Електронний ресурс] – Режим доступу до ресурсу: <http://borodiro.com/>
8. Programming Computer Vision [Електронний ресурс] – Режим доступу до ресурсу: <http://yajamal.info/>

# Додаток А — лістинг програмного коду

#include <stdio.h>

#include <opencv2/opencv.hpp>

using namespace cv;

const int kMenuTabs = 12;

//номера вариантов при свитче

const char\* menu[] =

{

"0 - Read image",

"1 - Apply linear filter",

"2 - Apply blur",

"3 - Apply medianBlur",

"4 - Apply GaussianBlur",

"5 - Apply erode",

"6 - Apply dilate",

"7 - Apply Sobel",

"8 - Apply Laplacian",

"9 - Apply Canny",

"10 - Apply calcHist",

"11 - Apply equalizeHist"

};

//названия вариантов

const char\* winNames[] =

{

"Initial image",

"filter2d",

"blur",

"medianBlur",

"GaussianBlur",

"erode",

"dilate",

"Sobel",

"Laplacian",

"Canny",

"calcHist",

"equalizeHist"

};

const int maxFileNameLen = 1000;

const int escCode = 27;

void printMenu();

void chooseMenuTab(int &activeMenuTab, Mat &srcImg);

void loadImage(Mat &srcImg);

int applyOperation(const Mat &src, const int operationIdx);

int applyBlur(const Mat &src, Mat &dst);

int applyMedianBlur(const Mat &src, Mat &dst);

int applyLinearFilter(const Mat &src, Mat &dst);

int applyGaussianBlur(const Mat &src, Mat &dst);

int applyErode(const Mat &src, Mat &dst);

int applyDilate(const Mat &src, Mat &dst);

int applySobel(const Mat &src, Mat &dst);

int applyLaplacian(const Mat &src, Mat &dst);

int applyCanny(const Mat &src, Mat &dst);

int applyСalcHist(const Mat &src, Mat &dst);

int applyEqualizeHist(const Mat &src, Mat &dst);

int main(int argc, char\*\* argv)

{

Mat srcImg; // устанавливаем картиночку

char ans;

int activeMenuTab = -1;

do

{

// выбираем пункт меню

chooseMenuTab(activeMenuTab, srcImg);

// выполняем операцию, согласно пункту

applyOperation(srcImg, activeMenuTab);

// вопрос на выход

printf("U wanna to continue? ESC - exit\n");

// ждем ответа на кнопку

ans = waitKey();

} while (ans != escCode);

destroyAllWindows(); // все закрываем

srcImg.release(); // освобождаем память на картиночке

return 0;

}

void printMenu()

{

int i = 0;

printf("Menu items:\n");

for (i; i < kMenuTabs; i++)

{

printf("\t%s\n", menu[i]);

}

printf("\n");

}

void loadImage(Mat &srcImg)

{

char fileName[maxFileNameLen];

do

{

printf("Input full file name: ");

scanf\_s("%s", &fileName, 40);

srcImg = imread(fileName, 1);

} while (srcImg.data == 0);

printf("The image was succesfully read\n\n");

}

void chooseMenuTab(int &activeMenuTab, Mat &srcImg)

{

int tabIdx;

while (true)

{

// напечатать номера меню

printMenu();

// выпрашиваем номер меню

printf("Choose the menu variant : ");

scanf\_s("%d", &tabIdx);

if (tabIdx == 0)

{

// читаем картиночку

loadImage(srcImg);

}

else if (tabIdx >= 1 && tabIdx < kMenuTabs && srcImg.data == 0)

{

// читаем картиночку

printf("Need image, write it fuulll name!\n");

loadImage(srcImg);

}

else if (tabIdx >= 1 && tabIdx < kMenuTabs)

{

activeMenuTab = tabIdx;

break;

}

}

}

int applyOperation(const Mat &src, const int operationIdx)

{

char key = -1;

Mat dst;

switch (operationIdx)

{

case 1:

{

// "1 - Apply linear filter"

applyLinearFilter(src, dst);

break;

}

case 2:

{

// "2 - Apply blur(...)"

applyBlur(src, dst);

break;

}

case 3:

{

// "3 - Apply medianBlur(...)"

applyMedianBlur(src, dst);

break;

}

case 4:

{

// "4 - Apply GaussianBlur(...)"

applyGaussianBlur(src, dst);

break;

}

case 5:

{

applyErode(src, dst);

break;

}

case 6:

{

// "6 - Apply dilate(...)"

applyDilate(src, dst);

break;

}

case 7:

{

// "7 - Apply Sobel(...)"

applySobel(src, dst);

break;

}

case 8:

{

// "8 - Apply Laplacian(...)"

applyLaplacian(src, dst);

break;

}

case 9:

{

// "9 - Apply Canny(...)"

applyCanny(src, dst);

break;

}

case 10:

{

// "10 - Apply calcHist(...)"

applyСalcHist(src, dst);

break;

}

case 11:

{

// "11 - Apply equalizeHist(...)"

applyEqualizeHist(src, dst);

break;

}

}

// показать стартовую картиночку

namedWindow(winNames[0], 1);

imshow(winNames[0], src);

// показать обработанную картиночку

namedWindow(winNames[operationIdx]);

imshow(winNames[operationIdx], dst);

return 0;

}

//линейный фильтр с констанами на матрице, ядре (центре изображения) ============ вариант 1

int applyLinearFilter(const Mat &src, Mat &dst)

{

Size kernelSize(3, 3);

const float kernelData[] = { -0.1f, 0.2f, -0.1f,

0.2f, 3.0f, 0.2f,

-0.1f, 0.2f, -0.1f };

const Mat kernel(3, 3, CV\_32FC1, (float \*)kernelData);

filter2D(src, dst, -1, kernel);

return 0;

}

//простой блюр ============ вариант 2

int applyBlur(const Mat &src, Mat &dst)

{

int imgHeight, imgWidth;

Size kernelSize(3, 3);

imgHeight = src.size().height;

imgWidth = src.size().width;

do

{

printf("Set Kernel sizes: \n");

printf("\tHeight: ");

scanf\_s("%d", &kernelSize.height);

printf("\twidth: ");

scanf\_s("%d", &kernelSize.width);

} while (kernelSize.width < 3 && kernelSize.width > imgWidth && kernelSize.height < 3 && kernelSize.height > imgHeight);

blur(src, dst, kernelSize);

return 0;

}

//медианный блюр, на входе кернел непарный, больше 3 (вообще больше 1, но лучше больше 3) ============ вариант 3

int applyMedianBlur(const Mat &src, Mat &dst)

{

int kSize = 3, minDim = -1;

minDim = min(src.size().height, src.size().width);

do

{

printf("Set kernel (apertur size) , it should be > 3 and odd : ");

scanf\_s("%d", &kSize);

} while (kSize < 3 && kSize > minDim && kSize % 2 == 0);

medianBlur(src, dst, kSize);

return 0;

}

//Гаусовский блюр, на входе кернел непарный, больше 3 (вообще больше 1, но лучше больше 3) ============ вариант 4

int applyGaussianBlur(const Mat &src, Mat &dst)

{

int imgHeight, imgWidth;

Size kernelSize(3, 3);

imgHeight = src.size().height;

imgWidth = src.size().width;

do

{

printf("Set Kernel sizes: \n");

printf("\tHeight : ");

scanf\_s("%d", &kernelSize.height);

printf("\twidth: ");

scanf\_s("%d", &kernelSize.width);

} while (kernelSize.width < 3 && kernelSize.width > imgWidth && kernelSize.height < 3 && kernelSize.height > imgHeight);

GaussianBlur(src, dst, kernelSize, 5, 5); //тут 2 последних значения ядра больше 0 и нечетные

return 0;

}

//Эродия, шаблон стандартный 3х3 ============ вариант 5

int applyErode(const Mat &src, Mat &dst)

{

Mat element;

element = Mat();

erode(src, dst, element); // (шаблон стандартный 3х3)

return 0;

}

//Дилитация, шаблон стандартный 3х3 ============ вариант 6

int applyDilate(const Mat &src, Mat &dst)

{

Mat element;

element = Mat();

dilate(src, dst, element); // (шаблон стандартный 3х3)

return 0;

}

//СобелЪ ============ вариант 7

int applySobel(const Mat &src, Mat &dst)

{

int ddepth = CV\_16S;

double alpha = 0.5, beta = 0.5;

Mat img, grayImg, xGrad, yGrad,

xGradAbs, yGradAbs, grad;

// сглаживание помощью фильтра Гаусса

GaussianBlur(src, img, Size(3, 3),

0, 0, BORDER\_DEFAULT);

// преобразование в оттенки серого

cvtColor(img, grayImg, CV\_RGB2GRAY);

// вычисление производных по двум направлениям

Sobel(grayImg, xGrad, ddepth, 1, 0); // по Ox

Sobel(grayImg, yGrad, ddepth, 0, 1); // по Oy

// преобразование градиентов в 8-битные

convertScaleAbs(xGrad, xGradAbs);

convertScaleAbs(yGrad, yGradAbs);

// поэлементное вычисление взвешенной

// суммы двух массивов

addWeighted(xGradAbs, alpha, yGradAbs, beta, 0, dst);

return 0;

}

//Лаплас ============ вариант 8

int applyLaplacian(const Mat &src, Mat &dst)

{

Mat img, grayImg, laplacianImg, laplacianImgAbs;

int ddepth = CV\_16S;

GaussianBlur(src, img, Size(3, 3),

0, 0, BORDER\_DEFAULT);

// преобразование в оттенки серого

cvtColor(img, grayImg, CV\_RGB2GRAY);

// применение оператора Лапласа

Laplacian(grayImg, laplacianImg, ddepth);

convertScaleAbs(laplacianImg, dst);

return 0;

}

//Канни ============ вариант 9

int applyCanny(const Mat &src, Mat &dst)

{

Mat img, grayImg, edgesImg;

double lowThreshold = 70, uppThreshold = 260;

// удаление шумов

blur(src, img, Size(3, 3));

// преобразование в оттенки серого

cvtColor(img, grayImg, CV\_RGB2GRAY);

// применение детектора Канни

Canny(grayImg, dst, lowThreshold, uppThreshold);

return 0;

}

//Вычисление гистограмм ============ вариант 10

int applyСalcHist(const Mat &src, Mat &dst)

{

Mat img, bgrChannels[3], bHist, gHist, rHist, histImg;

int kBins = 256; // количество бинов гистограммы

// интервал изменения значений бинов

float range[] = { 0.0f, 256.0f };

const float\* histRange = { range };

// равномерное распределение интервала по бинам

bool uniform = true;

// запрет очищения перед вычислением гистограммы

bool accumulate = false;

// размеры для отображения гистограммы

int histWidth = 512, histHeight = 400;

// количество пикселей на бин

int binWidth = cvRound((double)histWidth / kBins);

int i, kChannels = 3;

Scalar colors[] = { Scalar(255, 0, 0),

Scalar(0, 255, 0), Scalar(0, 0, 255) };

// выделение каналов изображения

split(src, bgrChannels);

// вычисление гистограммы для каждого канала

calcHist(&bgrChannels[0], 1, 0, Mat(), bHist, 1,

&kBins, &histRange, uniform, accumulate);

calcHist(&bgrChannels[1], 1, 0, Mat(), gHist, 1,

&kBins, &histRange, uniform, accumulate);

calcHist(&bgrChannels[2], 1, 0, Mat(), rHist, 1,

&kBins, &histRange, uniform, accumulate);

// построение гистограммы

dst = Mat(histHeight, histWidth, CV\_8UC3,

Scalar(0, 0, 0));

// нормализация гистограмм в соответствии с размерами

// окна для отображения

normalize(bHist, bHist, 0, dst.rows,

NORM\_MINMAX, -1, Mat());

normalize(gHist, gHist, 0, dst.rows,

NORM\_MINMAX, -1, Mat());

normalize(rHist, rHist, 0, dst.rows,

NORM\_MINMAX, -1, Mat());

// отрисовка ломаных

for (i = 1; i < kBins; i++)

{

line(dst, Point(binWidth \* (i - 1),

histHeight - cvRound(bHist.at<float>(i - 1))),

Point(binWidth \* i,

histHeight - cvRound(bHist.at<float>(i))),

colors[0], 2, 8, 0);

line(dst, Point(binWidth \* (i - 1),

histHeight - cvRound(gHist.at<float>(i - 1))),

Point(binWidth \* i,

histHeight - cvRound(gHist.at<float>(i))),

colors[1], 2, 8, 0);

line(dst, Point(binWidth \* (i - 1),

histHeight - cvRound(rHist.at<float>(i - 1))),

Point(binWidth \* i,

histHeight - cvRound(rHist.at<float>(i))),

colors[2], 2, 8, 0);

}

return 0;

}

//Выравнивание гистограмм ============ вариант 11

int applyEqualizeHist(const Mat &src, Mat &dst)

{

Mat img, grayImg, equalizedImg;

double lowThreshold = 70, uppThreshold = 260;

cvtColor(src, grayImg, CV\_RGB2GRAY);

// выравнивание гистограммы

equalizeHist(grayImg, dst);

return 0;

}