МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ВОДНОГО ГОСПОДАРСТВА ТА ПРИРОДОКОРИСТУВАННЯ

Звіт

З лабораторної роботи №8

На тему «Гістограма зображення»

З дисципліни «Основи проектування систем штучного інтелекту та розпізнавання образів»

Підготував: Коваль Ігор Леонідович

Студент навчально-наукового інституту автоматики, кібернетики та обчислювальної техніки

група КН-51м

Рівне 2022

Мета роботи: Навчитись виконувати створювати додатки з найпростішими перетвореннями зображення.

Теоретичні відомості

Гістограма зображення

Гістограма - це графік розподілу півтонів зображення, в якому по горизонтальній осі представлена яскравість, а по вертикалі - відносна кількість пікселів з даними значеннями яскравості.

Для того, щоб порахувати гістограму вручну, необхідно RGB зображення конвертувати у одноканальне 8-ми бітове зображення за допомогою наступного коду:

cvtColor(InputImage, outputImageGray, COLOR_BGR2GRAY);

де InputImage, outputImageGray це об'єкти класу Mat, що задають вихідне та результуюче зображення.

В результаті нормалізації діапазон яскравості лінійно розтягнеться на всю область. Після нормалізації зображення стане більш контрастним, так як відстань між сусідніми областями яскравості збільшиться.

Метою еквалізації є таке перетворення, щоб гістограма яскравості максимально близько відповідала рівномірному закону розподілення, тобто вихідне зображення повинно містити однакову кількість пікселів для кожного значення яскравості.

Результат роботи програми

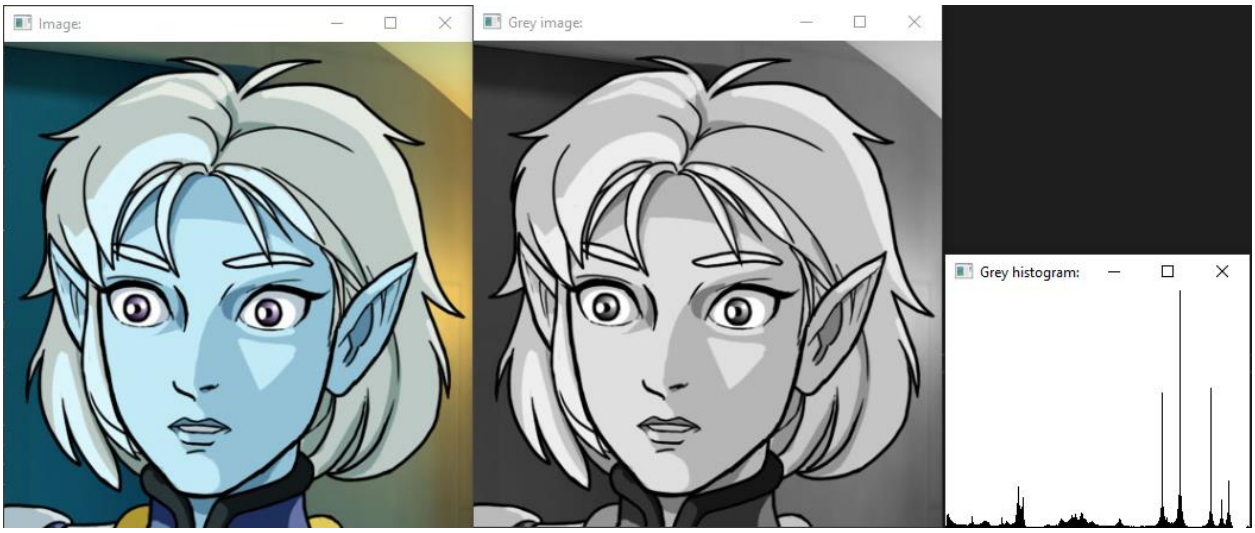

Рис. 1. Вихідне та чорно-біле зображення, гістограма чорнобілого зображення

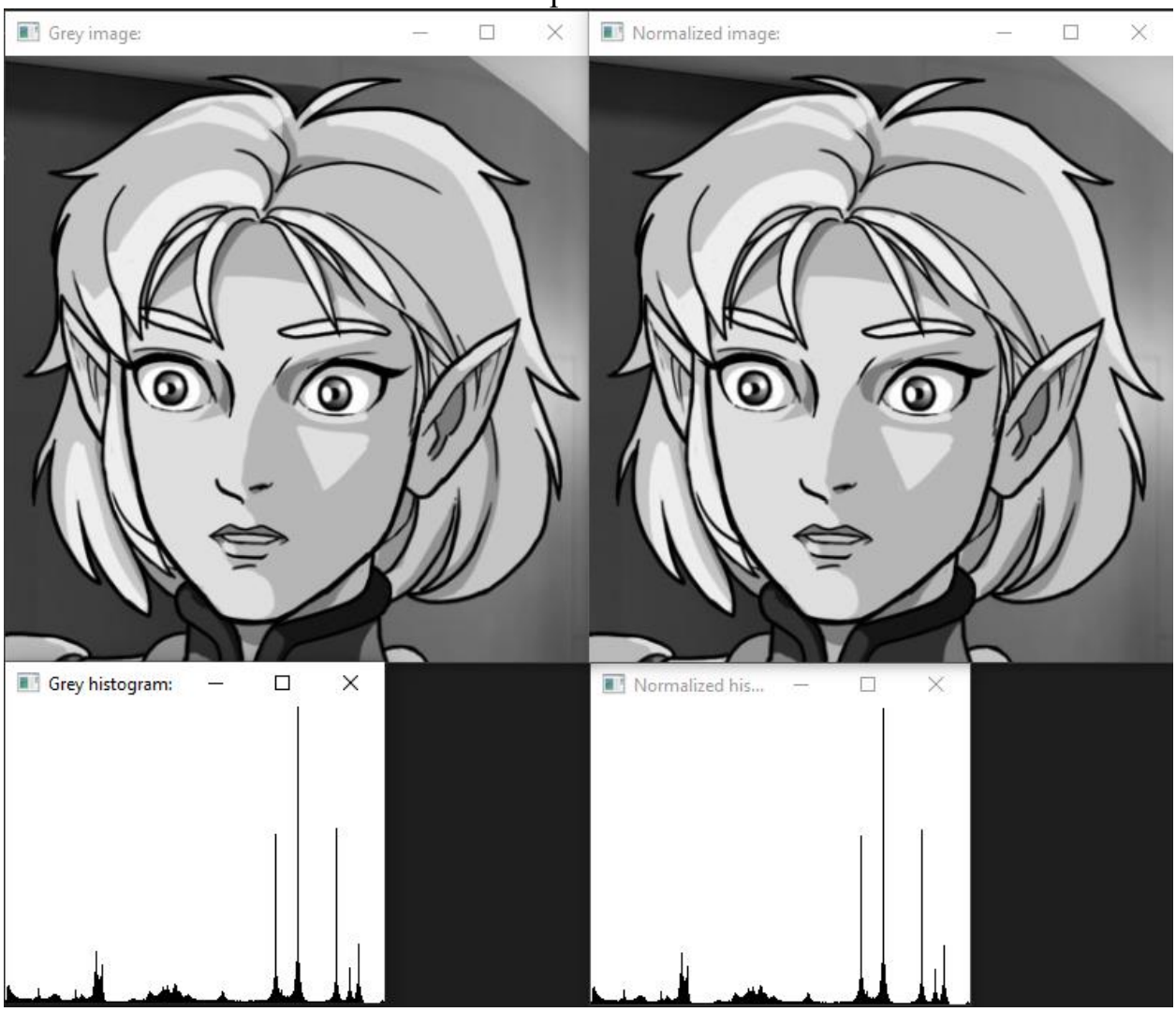

Рис. 2. Чорно-біле(зліва) та нормалізоване(справа) зображення, їх гістограми

Видно, що нормалізація не дала явних результатів, адже вихідне зображення вже мало невелику кількість засвітлених(255 по яскравості) та чорних(0 по яскравості) пікселів.

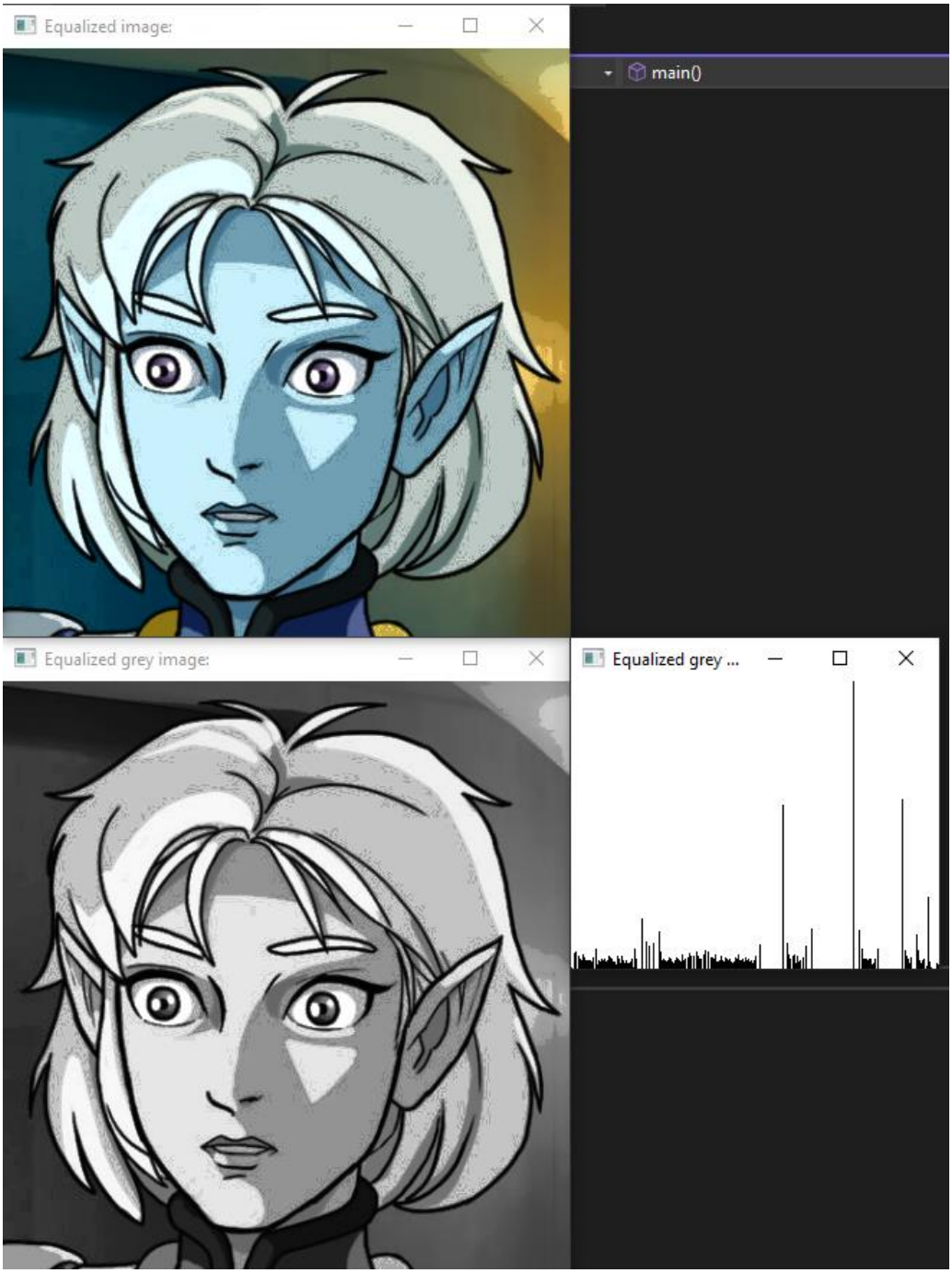

Рис. 3. Еквалізоване(зверху) та еквалізоване чорно-біле(знизу) зображення, гістограма еквалізованого чорно-білого зображення.

Код програми

```
#include <opencv.hpp>
#include <core.hpp>
#include <highgui.hpp>
#include <algorithm>

void drawHistogram(cv::Mat image, std::string windowName);
cv::Mat equalizeIntensity(const cv::Mat& inputImage);
cv::Mat normalize(const cv::Mat& inputImage);

int main()
{
    cv::Mat image = cv::imread("1.png");

    cv::namedWindow("Image:", 1);
    cv::imshow("Image:", image);


    cv::Mat GreyImage;
    cv::cvtColor(image, GreyImage, cv::COLOR_BGR2GRAY);
    cv::namedWindow("Grey image:", 1);
    cv::imshow("Grey image:", GreyImage);
    cv::namedWindow("Grey histogram:", 1);
    drawHistogram(GreyImage, "Grey histogram:");

    auto normalized = normalize(GreyImage);
    cv::namedWindow("Normalized image:", 1);
    cv::imshow("Normalized image:", normalized);
    cv::namedWindow("Normalized histogram:", 1);
    drawHistogram(normalized, "Normalized histogram:");

    auto equalized = equalizeIntensity(image);
    cv::namedWindow("Equalized image:", 1);
    cv::imshow("Equalized image:", equalized);
    cv::namedWindow("Equalized histogram:", 1);
    drawHistogram(equalized, "Equalized histogram:");

    cv::Mat equalizedGrey;
    cvtColor(equalized, equalizedGrey, cv::COLOR_BGR2GRAY);
    cv::namedWindow("Equalized grey image:", 1);
    cv::imshow("Equalized grey image:", equalizedGrey);
    cv::namedWindow("Equalized grey histogram:", 1);
    drawHistogram(equalizedGrey, "Equalized grey histogram:");


    cv::waitKey();

    return 0;
}

void drawHistogram(cv::Mat image, std::string windowName)
{
    int h = image.size().height;
    int w = image.size().width;
    std::vector<int> hist(256, 0);
    // обчислення значень гістограми
    for (int y = 0; y < h; y++)
    {
        for (int x = 0; x < w; x++)
        {
            int color = image.at <unsigned char>(y, x);
            hist[color]++;
        }
    }
```

```
    //рисування графіка гістограми
    cv::Mat imgHistogram(cv::Size(256, 200), CV_8UC1);
    rectangle(imgHistogram, cv::Point(0, 0),
    cv::Point(256, 200), CV_RGB(255, 255, 255), -1);
    int max_value = *(max_element(hist.begin(), hist.end()));
    //рисуємо вертикальні лінії на графіку гістограми для кожного значення
яскравості
    for (int i = 0; i < 256; ++i)
    {
        int val = hist[i];
        int nor = cvRound(val * 200 / max_value);
        line(imgHistogram, cv::Point(i, 200),
        cv::Point(i, 200 - nor), CV_RGB(0, 0, 0));
    }
    imshow(windowName, imgHistogram);
}

cv::Mat equalizeIntensity(const cv::Mat& inputImage)
{
    if (inputImage.channels() >= 3)
    {
        cv::Mat ycrcb;
        cvtColor(inputImage, ycrcb, cv::COLOR_BGR2YCrCb);
        std::vector<cv::Mat> channels;
        split(ycrcb, channels);
        cv::equalizeHist(channels[0], channels[0]);
        cv::Mat result;
        merge(channels, ycrcb);
        cvtColor(ycrcb, result, cv::COLOR_YCrCb2BGR);
        return result;
    }
    return cv::Mat();
}

cv::Mat normalize(const cv::Mat& inputImage)
{

    int h = inputImage.size().height;
    int w = inputImage.size().width;
    std::vector<int> hist(256, 0);

    uint minColor = std::numeric_limits<uint>::max();
    uint maxColor = std::numeric_limits<uint>::min();
    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            int color = inputImage.at <unsigned char>(y, x);

            if (color < minColor) {
                minColor = color;
            }
            if (color > maxColor) {
                maxColor = color;
            }
        }
    }


    auto norm = [minColor, maxColor](const int& original) {
        return std::clamp(static_cast<uint>(255 * (static_cast<float>(original -
minColor) / static_cast<float>(maxColor - minColor))), 0u, 255u);
    };

    cv::Mat normalized = inputImage;
    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
```

```
            int originalColor = inputImage.at <unsigned char>(y, x);
            normalized.at<unsigned char>(y, x) = norm(originalColor);
        }
    }

    return normalized;
}
```