Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**

**Дисциплина:** Методы и средства цифровой обработки информации

Выполнили студенты гр. 43501/1 А.А. Дроздовский

Руководитель Н.А. Абрамов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017г.

Санкт -Петербург

2017

**СОДЕРЖАНИЕ**

[**1.** **Цель работы** 3](#_Toc495283870)

[**2.** **Ход работы** 3](#_Toc495283871)

[**2.1.** **Оператор Собеля** 3](#_Toc495283872)

[**2.2.** **Оператор Прюитт** 3](#_Toc495283873)

[**2.3.** **Перекрёстный оператор Робертса** 3](#_Toc495283874)

[**3.** **Выводы** 3](#_Toc495283875)

[**Список используемой литературы** 3](#_Toc495283876)

[***ПРИЛОЖЕНИЕ*** 4](#_Toc495283877)

[**Текст программы** 4](#_Toc495283878)

1. **Цель работы**
2. **Ход работы**

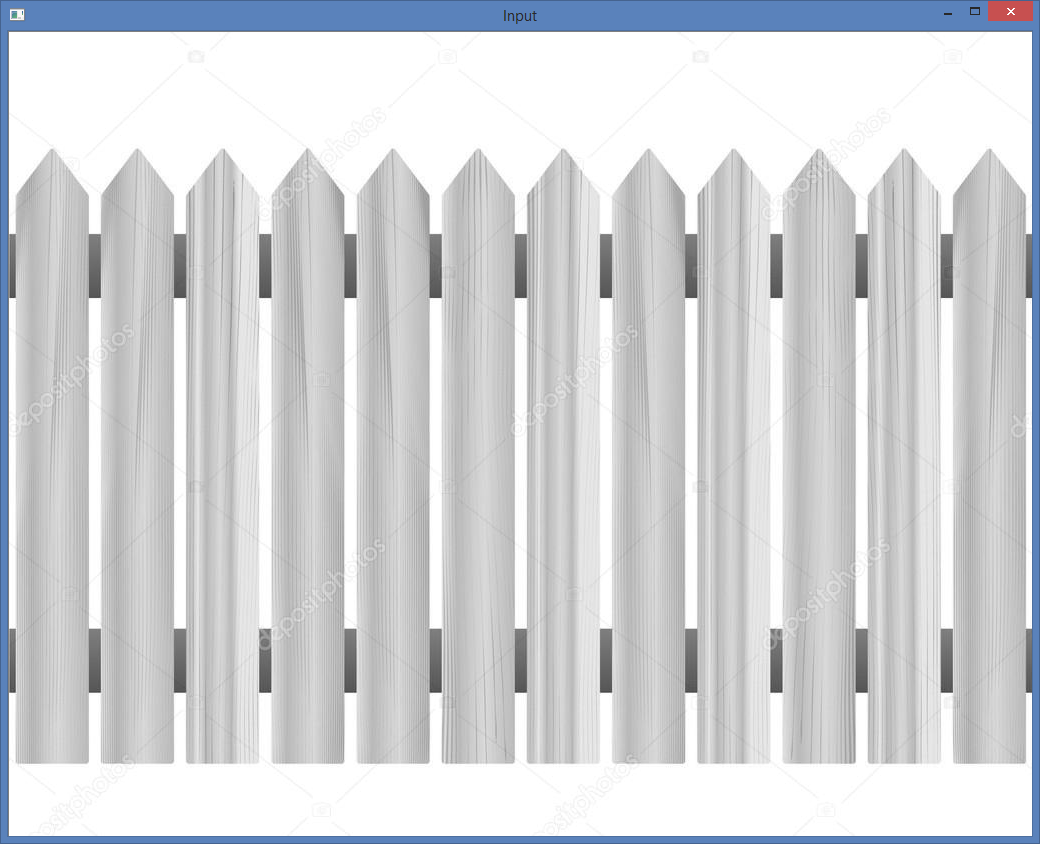


Рис. 1. Исходное изображение 1.



Рис. 2. Исходное изображение 2.



Рис. 3. Исходное изображение 3.

* 1. **Оператор Собеля**

;

;

;

, где

Image – Исходное изображение;

Gx, Gy – два изображения, на которых каждая точка содержит приближённые производные по *X* и по *Y*.

* 1. **Оператор Прюитт**
  2. **Перекрёстный оператор Робертса**

1. **Выводы**

**Список используемой литературы**

1. Sobel operator [Электронный ресурс] // Википедия : свободная энцикл., 2017. – URL: https://en.wikipedia.org/wiki/Sobel\_operator (Дата обращения: 05.10.2017).
2. Prewitt operator [Электронный ресурс] // Википедия : свободная энцикл., 2017. – URL: https://en.wikipedia.org/wiki/Prewitt\_operator (Дата обращения: 07.10.2017).
3. Roberts cross [Электронный ресурс] // Википедия : свободная энцикл., 2017. – URL: https://en.wikipedia.org/wiki/Roberts\_cross (Дата обращения: 07.10.2017).

# ***ПРИЛОЖЕНИЕ***

## **Текст программы**

#include "stdafx.h"

#include <opencv2\opencv.hpp>

#include <iostream>

#include <string>

#include <vector>

using namespace cv;

using namespace std;

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

{

/\*-----------------== Часть 1 ==-----------------\*/

namedWindow("LR1");

String imageName("LR1\_3.jpg");

if (argc > 1)

{

imageName = argv[1];

}

/// Считывание изображения в оттенках серого

Mat image = imread(imageName.c\_str(), IMREAD\_GRAYSCALE);

/// Проверка входных значений

if (image.empty())

{

cout << "Could not open or find the image" << endl;

return -1;

}

/// Вывод на экран первоначального изображения

imshow("Before", image);

int hist\_size = 256; // Число элементов в гистограмме

int hist\_width = 3; // Для автоподбора ширины изображений

float range\_0[] = { 0,256 };

const float\* ranges = { range\_0 };

int hist\_w = hist\_size\*hist\_width; int hist\_h = 400; // Ширина и высота полотна для нанесения гистограммы

int bin\_w = round((double)hist\_w / hist\_size); // Ширина элементов гистограммы

/// Построение гистограммы

Mat hist;

bool uniform = true, accumulate = false;

calcHist(&image, 1, 0, Mat(), hist, 1, &hist\_size, &ranges, uniform, accumulate);

Mat histImage(hist\_h, hist\_w, CV\_8UC3, Scalar(255, 255, 255));

Mat copy\_hist,hist\_without\_Norm; // Копии гистограммы

hist.copyTo(copy\_hist); // Для брасывания 5%

hist.copyTo(hist\_without\_Norm); // Для второй части работы

normalize(hist, hist, 0, histImage.rows, NORM\_MINMAX, -1, Mat());

for (int i = 1; i < hist\_size; i++)

rectangle(histImage, Point(i\*bin\_w, hist\_h),

Point((i + 1)\*bin\_w, hist\_h - round(hist.at<float>(i - 1))),

Scalar(100, 0, 0), 1, 8, 0.5);

/// Вывод гистограммы исходного изображения

imshow("Histogram of the initial image", histImage);

float b = 255, a = 0; // b - верхняя граница, a - нижняя граница исходной гистограммы

float d = 255, c = 0; // d - верхняя граница, c - нижняя граница результирующей гистограммы

double FivePercent = round((double)(image.cols\*image.rows)\*0.05); // 5% от общего числа пикселей исходного изображения

double sum\_for\_del = 0; // количество удаленных пикселей

for (; sum\_for\_del < FivePercent;)

if (copy\_hist.at<float>(a) > copy\_hist.at<float>(b)) // Сравнивнение последнего и первого элемент

{

sum\_for\_del += copy\_hist.at<float>(b);

for (int i = 0; i < image.cols\*image.rows; i++)

if (abs(double(image.at<unsigned char>(i)) - b)<1 )

image.at<unsigned char>(i) = b-1;

copy\_hist.at<float>(b) = 0;

b--;

}

else

{

for (int i = 0; i < image.cols\*image.rows; i++)

if (abs(double(image.at<unsigned char>(i)) - a)<1)

image.at<unsigned char>(i) = a+1;

sum\_for\_del += copy\_hist.at<float>(a);

copy\_hist.at<float>(a) = 0;

a++;

} Mat histImage2(hist\_h, hist\_w, CV\_8UC3, Scalar(255, 255, 255));

normalize(copy\_hist, copy\_hist, 0, histImage2.rows, NORM\_MINMAX, -1, Mat());

for (int i = 1; i < hist\_size; i++)

rectangle(histImage2, Point(i\*bin\_w, hist\_h),

Point((i + 1)\*bin\_w, hist\_h - round(copy\_hist.at<float>(i - 1))),

Scalar(100, 0, 0), 1, 8, 0.5);

/// Вывод гистограммы после отбрасывания 5%

imshow("Histogram after 5%", histImage2);

/// Замена пикселей в исходном изображение

for (int i = 0; i < image.rows\*image.cols; i++)

{

image.at<unsigned char>(i) = (double(image.at<unsigned char>(i)) - a)\*((d - c) / (b - a)) + c;

}

/// Вывод полученного изображения

imshow("After", image);

/// Построение результирующей гистограммы

Mat hist\_rez; // Результирующая гистограмма

calcHist(&image, 1, 0, Mat(), hist\_rez, 1, &hist\_size, &ranges, uniform, accumulate);

Mat histImage3(hist\_h, hist\_w, CV\_8UC3, Scalar(255, 255, 255));

normalize(hist\_rez, hist\_rez, 0, histImage3.rows, NORM\_MINMAX, -1, Mat());

for (int i = 1; i < hist\_size; i++)

rectangle(histImage3, Point(i\*bin\_w, hist\_h),

Point((i + 1)\*bin\_w, hist\_h - round(hist\_rez.at<float>(i - 1))),

Scalar(100, 0, 0), 1, 8, 0.5);

/// Вывод результирующей гистограммы

imshow("Resulting histogram", histImage3);

/\*-----------------== Часть 2 ==-----------------\*/

Mat image2 = imread(imageName.c\_str(), IMREAD\_GRAYSCALE);

vector<float> CDF;

CDF.push\_back(hist\_without\_Norm.at<float>(0));

for (int i = 1; i < hist\_without\_Norm.rows\*hist\_without\_Norm.cols; i++)

{

CDF.push\_back(CDF.at(i - 1) + hist\_without\_Norm.at<float>(i) );

}

/// Замена исходных пикселей

for (int i = 0; i < image2.rows\*image2.cols; i++)

{

image2.at<unsigned char>(i) = round(((CDF.at(double(image2.at<unsigned char>(i)))-CDF.at(0))/ (image2.rows\*image2.cols -1))\*255);

}

/// Вывод полученного изображения

imshow("After Histogram Equalization", image2);

/// Построение результирующей гистограммы после эквализации

Mat hist\_equalization; // Результирующая гистограмма

calcHist(&image2, 1, 0, Mat(), hist\_equalization, 1, &hist\_size, &ranges, uniform, accumulate);

Mat histImage4(hist\_h, hist\_w, CV\_8UC3, Scalar(255, 255, 255));

normalize(hist\_equalization, hist\_equalization, 0, histImage4.rows, NORM\_MINMAX, -1, Mat());

for (int i = 1; i < hist\_size; i++)

rectangle(histImage4, Point(i\*bin\_w, hist\_h),

Point((i + 1)\*bin\_w, hist\_h - round(hist\_equalization.at<float>(i - 1))),

Scalar(100, 0, 0), 1, 8, 0.5);

/// Вывод результирующей гистограммы

imshow("Resulting histogram befor equalization", histImage4);

waitKey(0);

return 0;

}