Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О. В. Васильев

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. С. Жук

**Тема работы:** Методы выделения границ. Алгоритм Канни.

**Ход работы:**

Задание 1. Реализовать метод, который принимает в качестве строки полный адрес файла изображения, читает изображение, переводит его в черно-белый цвет и выводит его на экран применяет размытие по Гауссу и выводит полученное изображение на экран.

Задание 2. Модифицировать построенный метод так, чтобы в результате вычислялось и выводилось на экран две матрицы – матрица значений длин и матрица значений углов градиентов всех пикселей изображения.

Задание 3. Модифицировать метод так, чтобы он выполнял подавление немаксимумов и выводил полученное изображение на экран. Рассмотреть изображение, сделать выводы.

Задание 4. Модифицировать метод так, чтобы он выполнял двойную пороговую фильтрацию и выводил полученное изображение на экран.

Задание 5 (самостоятельно). Провести опыты для различных параметров размытия и различных пороговых значений градиента, определить наилучшие параметры для Вашего изображения. Показать преподавателю значения параметров и результат работы на следующем занятии.

Задание 6 (самостоятельно). Реализовать алгоритм Канни на другом языке программирования.

В лабораторной работе необходимо было реализовать алгоритм Канни средствами языка Python и встроенными методами библиотеки OpenCV.

В первую очередь, согласно заданию 1, необходимо было прочитать полный адрес файла изображения, перевести его в чёрно-белый цвет и вывести на экран, применяя размытие по Гауссу. Использовались стандартные функции, разобранные в предыдущих лабораторных работах. Фрагмент кода, выполняющий это, представлен на рисунке 1.

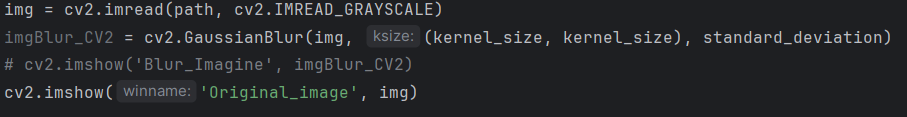


Рисунок 1 – Чтение и размытие изображения.

Далее необходимо было вывести экран две матрицы – матрицу значений длин и матрицу значений углов градиентов всех пикселей изображения, т. е. необходимо было выполнить второй шаг алгоритма Канни.

Реализована операция свертки (Рисунок 2).

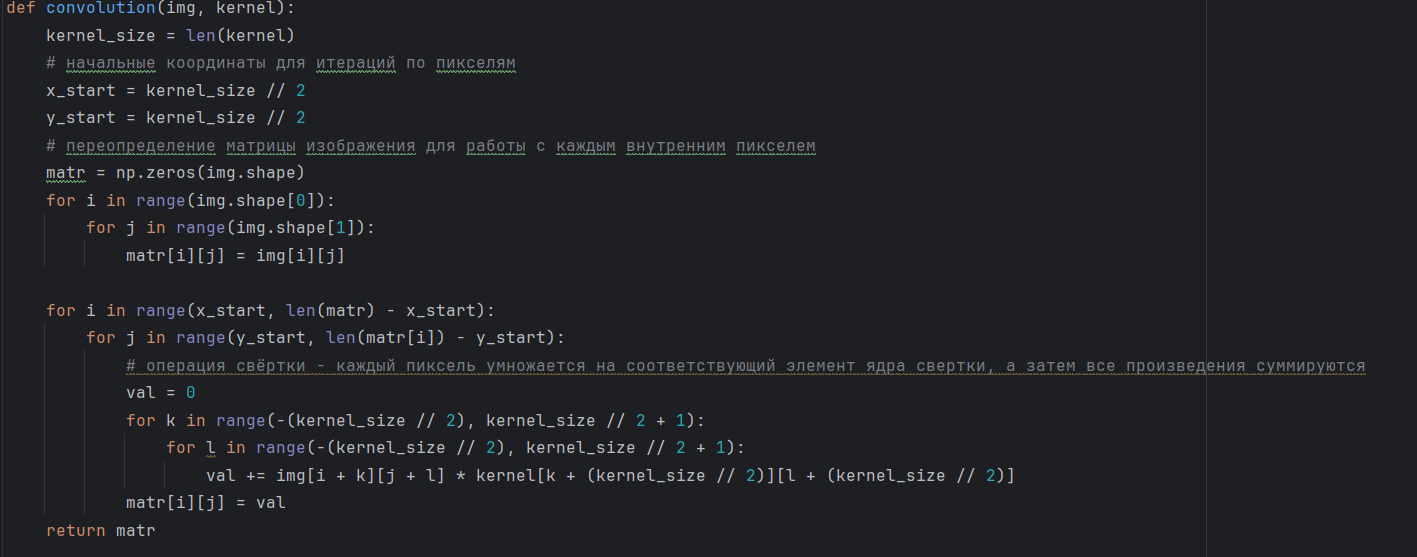


Рисунок 2 – Реализация операции свёртки.

Далее вычисляются и выводятся значения длин и матрицы углов градиента (Рисунок 3).

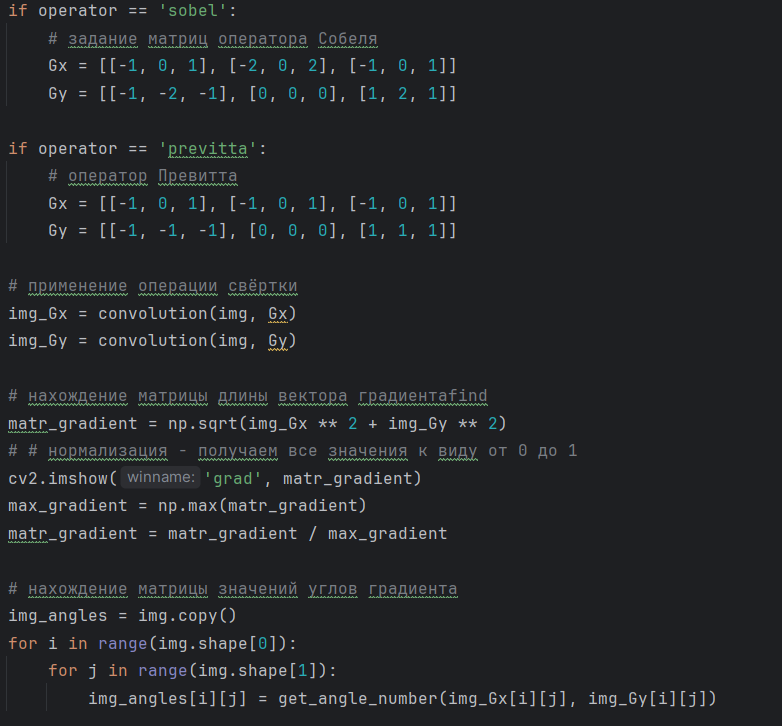


Рисунок 3 – Вычисления значений длин и матрицы углов.

В данном на рисунке 3 коде определены матрицы оператора Собеля и Превитта.

*– Матрицы оператора Собеля*

*– Матрицы оператора Превитта*

После этого к изображению и к каждой из выше определенных матриц применена операция свертки, в зависимости от выбора оператора.

Далее с помощью формулы была найдена матрица длины вектора градиента (Рисунок 4).

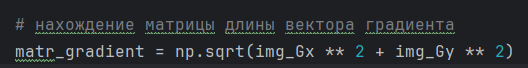


Рисунок 4 – Нахождение матрицы длины вектора градиента.

Для поиска матрицы значений углов градиента предварительно необходимо было определить функцию, которая находит округления угла между вектором градиента и осью Х. На рисунке 5 представлена её реализация.

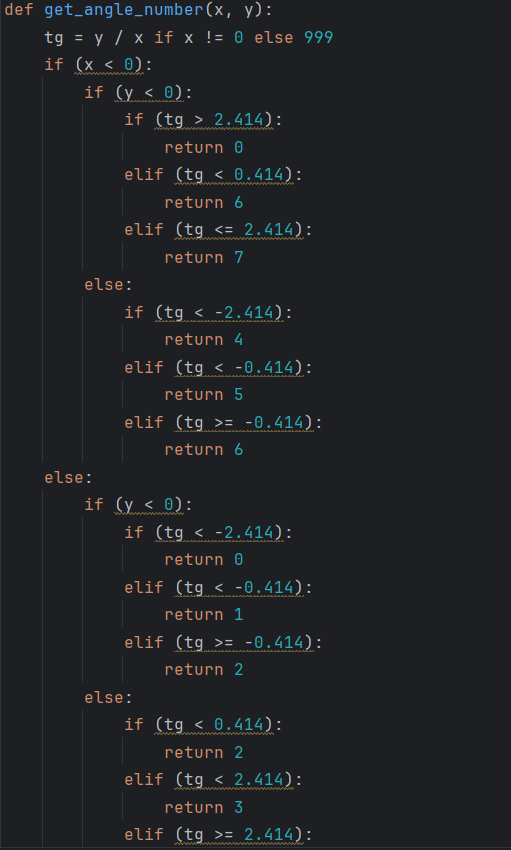


Рисунок 5 – Функция нахождения округления угла между вектором градиента и осью Х.

Округление нужно, так как нет необходимости точно знать, какова величина угла 10 или 15 градусов, например. По факту, необходимо для направления выбрать лишь один из соседних 8 пикселей. То есть необходимо округлить величину угла до 45 градусов.

Далее нормализуем матрицу градиентов, то есть приведем все значения к диапазону от 0 до 1 (Рисунок 6).

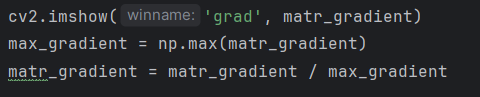


Рисунок 6 – нормализация матрицы градиентов.

После этого по заданию необходимо найти значения углов градиента. Инициализировав перед этим изображение, с помощью циклов находятся значения углов градиента и выводятся матрица значений длин и матрица значений углов градиента (Рисунок 7).

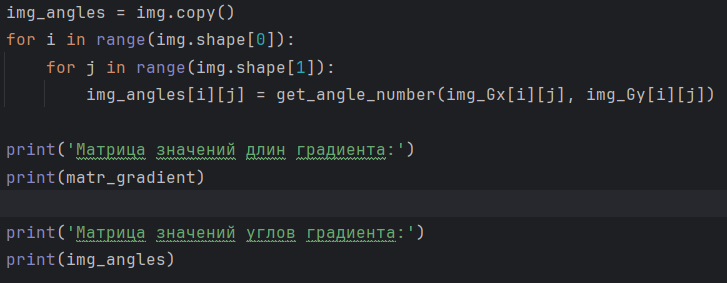


Рисунок 7 – вычисление матрицы значений углов и вывод

В задании 3 требуется подавить немаксимумы. В алгоритме Канни немаксимум (non-maximum suppression) — это процесс, который используется для уменьшения ширины границ, обнаруженных на изображении. Он заключается в том, что для каждого пикселя на границе изображения проверяется, является ли он локальным максимумом в направлении градиента (направление градиента перпендикулярно границам). Если пиксель не является локальным максимумом, то его значение устанавливается в 0. Это позволяет сохранить только те пиксели на границе изображения, которые имеют максимальную длину градиента в направлении градиента. Говоря другими словами: границей будет считаться пиксель, градиент которого максимален в сравнении с пикселями по направлению наибольшего роста функции; если значение градиента выше, чем у пикселей слева и справа, то данный пиксель – это граница, иначе – не граница.

На рисунке 8 отображена реализация данного метода.

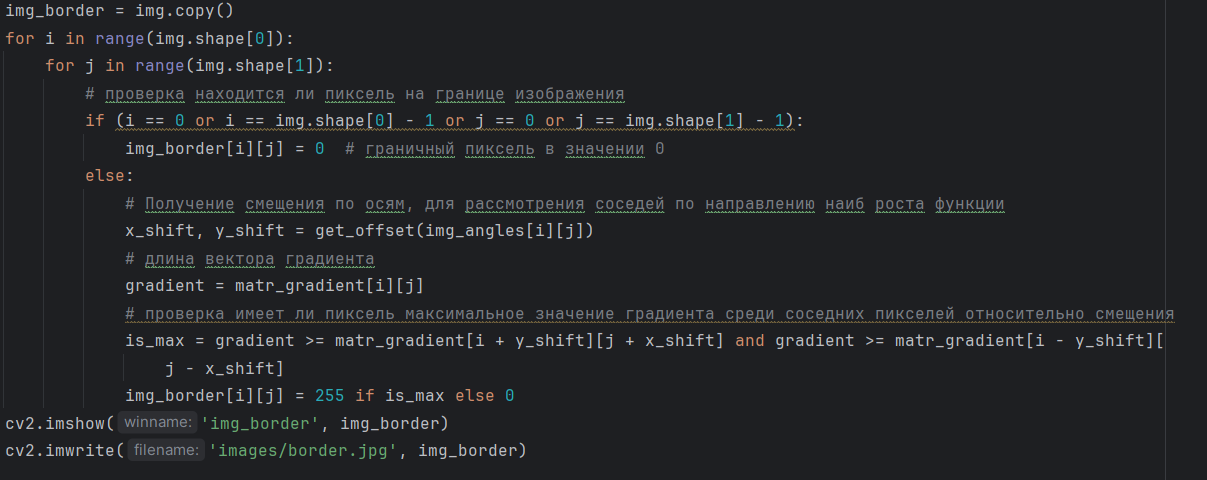


Рисунок 8 – Подавление немаксимумов.

Последний шагом в алгоритме Канни является построение двойной фильтрации, что соответствует заданию 4 лабораторной работы. Она используется в алгоритме для определения границ на изображении. Процесс заключается в том, что для каждого пикселя на границе изображения сравнивается величина его градиента с двумя пороговыми значениями. Это позволяет отфильтровать шумы и сохранить только те границы, которые имеют максимальную длину градиента и являются значимыми.

Реализация предоставлена на рисунке 9.

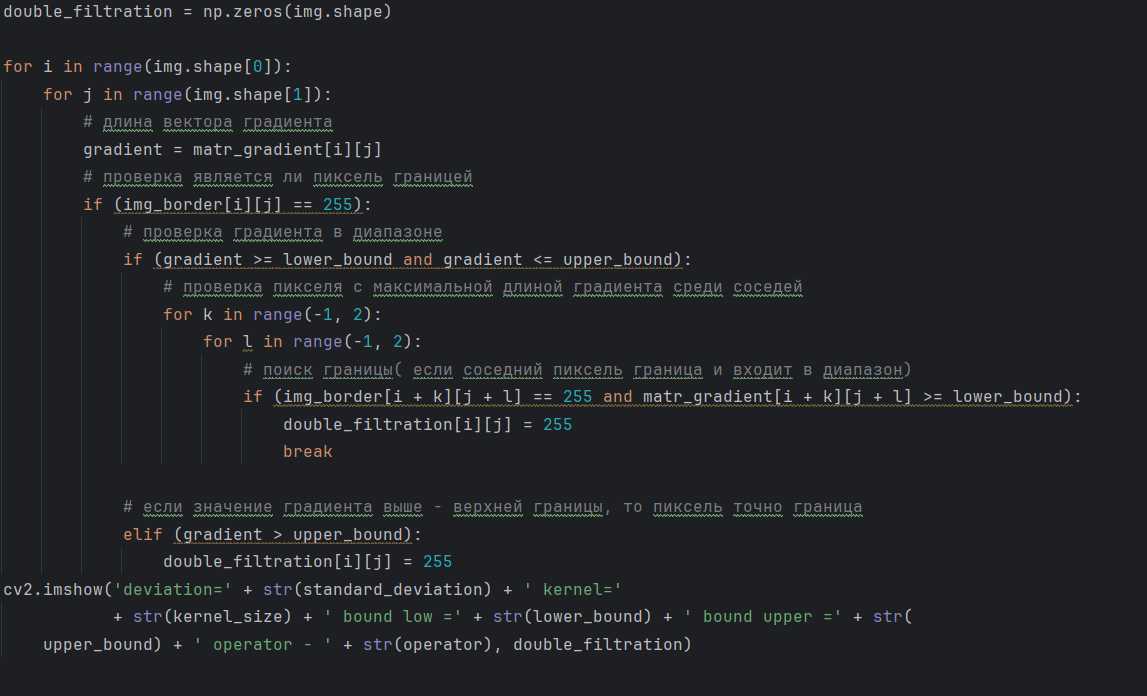


Рисунок 9 – Двойная пороговая фильтрация.

Последним заданием является опыты с изображением. На рисунке 10 представлено результат работы алгоритма Канни со стандартным отклонением равным 10, размером матрицы свёртки равным 3.

Изображение выглядит как текст, снимок экрана, черно-белый, черный

Автоматически созданное описание

Рисунок 10 – Алгоритм Канни со стандартным отклонением равным 10, размером матрицы свёртки равным 3.

Изменим параметры на новые. Стандартное отклонение выставим на 100, а размер матрицы свёртки на 11 (рисунок 11).

Изображение выглядит как текст, снимок экрана, черный, черно-белый

Автоматически созданное описание

Рисунок 11 – Алгоритм Канни со стандартным отклонением равным 100, размером матрицы свёртки равным 11.

Повторим изменение. Отклонение- 6, размер- 5 (рисунок 12).



Рисунок 12 – Алгоритм Канни со стандартным отклонением равным 6, размером матрицы свёртки равным 5.

Не трудно заметить, что с увеличением параметров алгоритма, увеличиваются соответствующие матрицы и увеличивается детализированность границ на изображении.

**Листинг программ**

Файл main.py

import cv2  
import numpy as np  
import copy  
  
  
# реализация операции свёртки  
def convolution(img, kernel):  
 kernel\_size = len(kernel)  
 # начальные координаты для итераций по пикселям  
 x\_start = kernel\_size // 2  
 y\_start = kernel\_size // 2  
 # переопределение матрицы изображения для работы с каждым внутренним пикселем  
 matr = np.zeros(img.shape)  
 for i in range(img.shape[0]):  
 for j in range(img.shape[1]):  
 matr[i][j] = img[i][j]  
  
 for i in range(x\_start, len(matr) - x\_start):  
 for j in range(y\_start, len(matr[i]) - y\_start):  
 # операция свёртки - каждый пиксель умножается на соответствующий элемент ядра свертки, а затем все произведения суммируются  
 val = 0  
 for k in range(-(kernel\_size // 2), kernel\_size // 2 + 1):  
 for l in range(-(kernel\_size // 2), kernel\_size // 2 + 1):  
 val += img[i + k][j + l] \* kernel[k + (kernel\_size // 2)][l + (kernel\_size // 2)]  
 matr[i][j] = val  
 return matr  
  
  
# нахождение округления угла между вектором градиента и осью Х  
def get\_angle\_number(x, y):  
 tg = y / x if x != 0 else 999  
 if (x < 0):  
 if (y < 0):  
 if (tg > 2.414):  
 return 0  
 elif (tg < 0.414):  
 return 6  
 elif (tg <= 2.414):  
 return 7  
 else:  
 if (tg < -2.414):  
 return 4  
 elif (tg < -0.414):  
 return 5  
 elif (tg >= -0.414):  
 return 6  
 else:  
 if (y < 0):  
 if (tg < -2.414):  
 return 0  
 elif (tg < -0.414):  
 return 1  
 elif (tg >= -0.414):  
 return 2  
 else:  
 if (tg < 0.414):  
 return 2  
 elif (tg < 2.414):  
 return 3  
 elif (tg >= 2.414):  
 return 4  
  
  
# Получение значений для смещения по осям  
# на вход номер блока угла  
def get\_offset(angle):  
 x\_shift = 0  
 y\_shift = 0  
 # смещение по оси абсцисс  
 if (angle == 0 or angle == 4):  
 x\_shift = 0  
 elif (angle > 0 and angle < 4):  
 x\_shift = 1  
 else:  
 x\_shift = -1  
 # смещение по оси ординат  
 if (angle == 2 or angle == 6):  
 y\_shift = 0  
 elif (angle > 2 and angle < 6):  
 y\_shift = -1  
 else:  
 y\_shift = 1  
 return x\_shift, y\_shift  
  
  
def main(path, standard\_deviation, kernel\_size, lower\_bound, upper\_bound, operator):  
 # Задание 1 - чтение строки полного адреса изображения и размытие Гаусса  
 img = cv2.imread(path, cv2.IMREAD\_GRAYSCALE)  
 imgBlur\_CV2 = cv2.GaussianBlur(img, (kernel\_size, kernel\_size), standard\_deviation)  
 # cv2.imshow('Blur\_Imagine', imgBlur\_CV2)  
 cv2.imshow('Original\_image', img)  
  
 # Задание 2 - вычисление и вывод матрицы значений длин и матрицы значений углов градиентов  
 if operator == 'sobel':  
 # задание матриц оператора Собеля  
 Gx = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]  
 Gy = [[-1, -2, -1], [0, 0, 0], [1, 2, 1]]  
  
 if operator == 'previtta':  
 # оператор Превитта  
 Gx = [[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]]  
 Gy = [[-1, -1, -1], [0, 0, 0], [1, 1, 1]]  
  
 # применение операции свёртки  
 img\_Gx = convolution(img, Gx)  
 img\_Gy = convolution(img, Gy)  
  
 # нахождение матрицы длины вектора градиента  
 matr\_gradient = np.sqrt(img\_Gx \*\* 2 + img\_Gy \*\* 2)  
  
 # # нормализация - получаем все значения к виду от 0 до 1  
 cv2.imshow('grad', matr\_gradient)  
 max\_gradient = np.max(matr\_gradient)  
 matr\_gradient = matr\_gradient / max\_gradient  
  
 # нахождение матрицы значений углов градиента  
 img\_angles = img.copy()  
 for i in range(img.shape[0]):  
 for j in range(img.shape[1]):  
 img\_angles[i][j] = get\_angle\_number(img\_Gx[i][j], img\_Gy[i][j])  
  
 print('Матрица значений длин градиента:')  
 print(matr\_gradient)  
  
 print('Матрица значений углов градиента:')  
 print(img\_angles)  
  
 # Задание 3 - подавление немаксимумов  
  
 # инициализация массива границ изображения  
 img\_border = img.copy()  
 for i in range(img.shape[0]):  
 for j in range(img.shape[1]):  
 # проверка находится ли пиксель на границе изображения  
 if (i == 0 or i == img.shape[0] - 1 or j == 0 or j == img.shape[1] - 1):  
 img\_border[i][j] = 0 # граничный пиксель в значении 0  
 else:  
 # Получение смещения по осям, для рассмотрения соседей по направлению наиб роста функции  
 x\_shift, y\_shift = get\_offset(img\_angles[i][j])  
 # длина вектора градиента  
 gradient = matr\_gradient[i][j]  
 # проверка имеет ли пиксель максимальное значение градиента среди соседних пикселей относительно смещения  
 is\_max = gradient >= matr\_gradient[i + y\_shift][j + x\_shift] and gradient >= matr\_gradient[i - y\_shift][  
 j - x\_shift]  
 img\_border[i][j] = 255 if is\_max else 0  
 cv2.imshow('img\_border', img\_border)  
 cv2.imwrite('images/border.jpg', img\_border)  
  
 # Задание 4 - двойная пороговая фильтрация  
  
 # инициализация массива результата  
 double\_filtration = np.zeros(img.shape)  
  
 for i in range(img.shape[0]):  
 for j in range(img.shape[1]):  
 # длина вектора градиента  
 gradient = matr\_gradient[i][j]  
 # проверка является ли пиксель границей  
 if (img\_border[i][j] == 255):  
 # проверка градиента в диапазоне  
 if (gradient >= lower\_bound and gradient <= upper\_bound):  
 # проверка пикселя с максимальной длиной градиента среди соседей  
 for k in range(-1, 2):  
 for l in range(-1, 2):  
 # поиск границы( если соседний пиксель граница и входит в диапазон)  
 if (img\_border[i + k][j + l] == 255 and matr\_gradient[i + k][j + l] >= lower\_bound):  
 double\_filtration[i][j] = 255  
 break  
  
 # если значение градиента выше - верхней границы, то пиксель точно граница  
 elif (gradient > upper\_bound):  
 double\_filtration[i][j] = 255  
 cv2.imshow('deviation=' + str(standard\_deviation) + ' kernel='  
 + str(kernel\_size) + ' bound low =' + str(lower\_bound) + ' bound upper =' + str(  
 upper\_bound) + ' operator - ' + str(operator), double\_filtration)  
  
  
main('images/image.jpg', 10, 3, 0.2, 0.4, 'sobel')  
main('images/image.jpg', 100, 3, 0.15, 0.85, 'sobel')  
# main('image.jpg', 20, 3, 7, 'previtta')  
cv2.waitKey(0)

Файл findBorders.java

package lab4;  
  
import org.bytedeco.opencv.opencv\_core.\*;  
  
import static org.bytedeco.opencv.global.opencv\_core.\*;  
import static org.bytedeco.opencv.global.opencv\_imgcodecs.\*;  
import static org.bytedeco.opencv.global.opencv\_imgproc.\*;  
import static org.bytedeco.opencv.global.opencv\_highgui.\*;  
  
public class findBorders {  
  
 public static int[][] convolution(Mat img, int[][] kernel) {  
 int kernelSize = kernel.length;  
 int xStart = kernelSize / 2;  
 int yStart = kernelSize / 2;  
 int[][] matr = new int[img.rows()][img.rows()];  
  
  
 for (int i = 0; i < img.rows(); i++) {  
 for (int j = 0; j < img.cols(); j++) {  
 matr[i][j] = img.ptr(i, j).get();  
 }  
 }  
 for (int i = xStart; i < img.rows() - xStart; i++) {  
 for (int j = yStart; j < img.cols() - yStart; j++) {  
 int val = 0;  
  
 for (int k = -(kernelSize / 2); k <= kernelSize / 2; k++) {  
 for (int l = -(kernelSize / 2); l <= kernelSize / 2; l++) {  
 int imgValue = img.ptr(i + k, j + l).get();  
 if (imgValue < 0) {  
 imgValue = 128 + (128 % imgValue);  
 }  
 int res = imgValue \* kernel[k + (kernelSize / 2)][l + (kernelSize / 2)];  
 val += res;  
 }  
 }  
  
 matr[i][j] = val;  
 }  
 }  
 return matr;  
 }  
  
 public static int getAngleNumber(double x, double y) {  
 double tg = (x != 0) ? y / x : 999;  
 if (x < 0) {  
 if (y < 0) {  
 if (tg > 2.414) return 0;  
 else if (tg < 0.414) return 6;  
 else return 7;  
 } else {  
 if (tg < -2.414) return 4;  
 else if (tg < -0.414) return 5;  
 else return 6;  
 }  
 } else {  
 if (y < 0) {  
 if (tg < -2.414) return 0;  
 else if (tg < -0.414) return 1;  
 else return 2;  
 } else {  
 if (tg < 0.414) return 2;  
 else if (tg < 2.414) return 3;  
 else return 4;  
 }  
 }  
 }  
  
 public static int[] getOffset(int angle) {  
 int xShift = 0;  
 int yShift = 0;  
  
 if (angle == 0 || angle == 4) {  
 xShift = 0;  
 } else if (angle > 0 && angle < 4) {  
 xShift = 1;  
 } else {  
 xShift = -1;  
 }  
  
 if (angle == 2 || angle == 6) {  
 yShift = 0;  
 } else if (angle > 2 && angle < 6) {  
 yShift = -1;  
 } else {  
 yShift = 1;  
 }  
  
 return new int[]{xShift, yShift};  
 }  
  
 public static void main(String[] args) {  
 double standardDeviation = 20;  
 int kernelSize = 3;  
 int boundPath = 6;  
  
 // Load the image  
 Mat img = imread("images/test\_512.jpg", IMREAD\_GRAYSCALE);  
 Mat imgBlurCV2 = new Mat();  
 GaussianBlur(img, imgBlurCV2, new Size(kernelSize, kernelSize), standardDeviation);  
 imshow("blur", imgBlurCV2);  
  
 // Create the kernel  
 int[][] Gx = {  
 {-1, 0, 1},  
 {-2, 0, 2},  
 {-1, 0, 1}  
 };  
 int[][] Gy = {  
 {-1, -2, -1},  
 {0, 0, 0},  
 {1, 2, 1}  
 };  
  
 // Perform convolution  
 int[][] imgGx = convolution(img, Gx);  
 int[][] imgGy = convolution(img, Gy);  
  
 // Calculate gradient length matrix  
 Mat matrGradient = new Mat(img.size(), CV\_64F);  
 for (int i = 0; i < img.rows(); i++) {  
 for (int j = 0; j < img.cols(); j++) {  
 int gx = imgGx[i][j];  
 int gy = imgGy[i][j];  
 double magnitude = Math.sqrt(gx \* gx + gy \* gy);  
 matrGradient.ptr(i, j).putDouble(magnitude);  
 }  
 }  
 //нормирование  
 double maxGradient = 0;  
 for (int i = 0; i < img.rows(); i++) {  
 for (int j = 0; j < img.cols(); j++) {  
 if (matrGradient.ptr(i, j).getDouble() > maxGradient) {  
 maxGradient = matrGradient.ptr(i, j).getDouble();  
 }  
 }  
 }  
 for (int i = 0; i < img.rows(); i++) {  
 for (int j = 0; j < img.cols(); j++) {  
 double newVal = matrGradient.ptr(i, j).getDouble()/maxGradient;  
 matrGradient.ptr(i, j).putDouble(newVal);  
 }  
 }  
  
  
 // Calculate gradient angle matrix  
 Mat imgAngles = new Mat(img.size(), img.type());  
 for (int i = 0; i < img.rows(); i++) {  
 for (int j = 0; j < img.cols(); j++) {  
 int angle = getAngleNumber(imgGx[i][j], imgGy[i][j]);  
 imgAngles.ptr(i, j).put((byte) angle);  
 }  
 }  
  
  
 Mat imgBorder = new Mat(img.size(), CV\_32S);  
 for (int i = 0; i < img.rows(); i++) {  
 for (int j = 0; j < img.cols(); j++) {  
 if (i == 0 || i == img.rows() - 1 || j == 0 || j == img.cols() - 1) {  
 imgBorder.ptr(i, j).putInt(0);  
 } else {  
 int[] offset = getOffset(imgAngles.ptr(i, j).get());  
 double gradient = matrGradient.ptr(i, j).getDouble();  
 boolean isMax = gradient >= matrGradient.ptr(i + offset[1], j + offset[0]).getDouble() &&  
 gradient >= matrGradient.ptr(i - offset[1], j - offset[0]).getDouble();  
  
 imgBorder.ptr(i, j).putInt((byte) (isMax ? 255 : 0));  
 }  
 }  
 }  
  
 //Double thresholding  
 Mat doubleThresholded = new Mat(img.size(), img.type());  
 maxGradient = 0;  
 for (int i = 0; i < img.rows(); i++) {  
 for (int j = 0; j < img.cols(); j++) {  
 if (matrGradient.ptr(i, j).getDouble() > maxGradient) {  
 maxGradient = matrGradient.ptr(i, j).getDouble();  
 }  
 }  
 }  
 double lowerBound = maxGradient / boundPath;  
 double upperBound = maxGradient - (maxGradient / boundPath);  
  
 for (int i = 0; i < img.rows(); i++) {  
 for (int j = 0; j < img.cols(); j++) {  
 double gradient = matrGradient.ptr(i, j).getDouble();  
 if (imgBorder.ptr(i, j).get() == (byte) 255) {  
 if (gradient >= lowerBound && gradient <= upperBound) {  
 boolean hasStrongNeighbor = false;  
 for (int k = -1; k <= 1; k++) {  
 for (int l = -1; l <= 1; l++) {  
 if (matrGradient.ptr(i + k, j + l).getDouble() >= lowerBound) {  
 hasStrongNeighbor = true;  
 break;  
 }  
 }  
 if (hasStrongNeighbor) {  
 break;  
 }  
 }  
 if (hasStrongNeighbor) {  
 doubleThresholded.ptr(i, j).put((byte) 255);  
 }  
 } else if (gradient > upperBound) {  
 doubleThresholded.ptr(i, j).put((byte) 255);  
 }  
 } else {  
 doubleThresholded.ptr(i, j).put((byte) 0);  
 }  
 }  
 }  
  
 imshow("result", doubleThresholded);  
 waitKey(0);  
 }  
}