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 №3**

**Дисциплина: Алгоритмы цифровой обработки мультимедиа**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_С. Н. Гордов

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А. А. Крамаренко

**Тема работы:** Методы фильтрации и размытия изображения.

**Ход работы:**

Задание 1. Выполнить пункты 1 и 2 алгоритма, то есть построить матрицу Гаусса. Просмотреть итоговую матрицу для размерностей 3, 5, 7.

Задание 2. Нормировать полученную матрицу Гаусса. Протестировать результаты на матрицах из предыдущего пункта.

Задание 3. Реализовать фильтр Гаусса средствами языка python.

Задание 4. Применить данный фильтр для двух разных значений среднего квадратичного отклонения и двух разных размерностей матрицы свертки, сравнить результаты для ОДНОГО изображения.

Задание 5. Реализовать размытие Гаусса встроенным методом библиотеки OpenCV, сравнить результаты с Вашей реализацией.

Задание 6 (самостоятельно) Реализовать размытие Гаусса средствами любого другого языка программирования.

В лабораторной работе необходимо было реализовать фильтр Гаусса двумя способами – средствами языка Python и встроенным методом библиотеки OpenCV.

Сперва определим шаги, из которых состоит фильтр Гаусса:

1) установление размера матрицы свертки и среднее квадратичное отклонение;

2) заполнение матрицы свертки значениями функции Гаусса с математическим ожиданием, равным координатам центра матрицы;

3) нормировка матрицы так, чтобы сумма элементов равнялась 1;

4) создание копию изображения;

5) для каждого внутреннего пикселя копии расчёт нового значения насыщенности пикселя по формуле операции свёртки и запись этого значения в пиксель нового изображения.

Для реализации фильтра Гаусса средствами языка Python была написана функция AnotherGaussianBlur(img, kernel\_size, standard\_deviation), принимающая в качестве аргументов исходное изображение, размер предполагаемой матрицы Гаусса(определяет, насколько далеко распространяются веса от центрального пикселя) и стандартное отклонение(определяет степень размытия). Последние два задаются в теле основной программы через присваивание.

Первоначально матрица свёртки (ядра) заполняется единицами с помощью функции библиотеки numpy - kernel = np.ones((kernel\_size, kernel\_size)). Далее вычисляется центральный элемент матрицы для определения центра фокуса. Эти значения необходимы нам для построения матрицы свёртки. К каждому элементу ядра применяется функция Гаусса kernel[i, j] = gauss(i, j, standard\_deviation, a, b), определенная отдельным методом в тексте программы, где одними из параметров является математическое ожидание, равное координатам центра матрицы. Функция Гаусса задаётся формулой, представленной на рисунке 1.

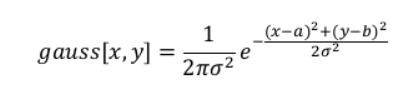


Рисунок 1 ­– Функция Гаусса для двумерной случайной величины.

Далее выполняется нормализация матрицы свёртки. Она гарантирует, что сумма всех элементов в матрице равна единице. Это означает, что при применении ядра свертки к изображению, интенсивность пикселей в выходном изображении будет оставаться пропорциональной интенсивности пикселей в исходном изображении и оно не изменит общую яркость изображения. С помощью двух циклов for сперва на каждой итерации суммируется все элементы матрицы, а затем второй цикл делит каждый элемент на полученную сумму.

Следующий шаг — это применение операции свёртки к изображению. Для этого была реализована функция под названием Convolution(img, kernel). Она принимает в качестве входных аргументов исходное изображение и полученную матрицу ядра свёртки. Операция выполняется путем “скольжения” ядра по изображению. На каждом шаге ядро умножается на соответствующую область пикселей в изображении, а затем все произведения суммируются. Результатом операции свертки является новое изображение, в котором каждый пиксель представляет собой сумму произведений ядра и соответствующих пикселей в исходном изображении. Полученная функция отображена на рисунке 2.

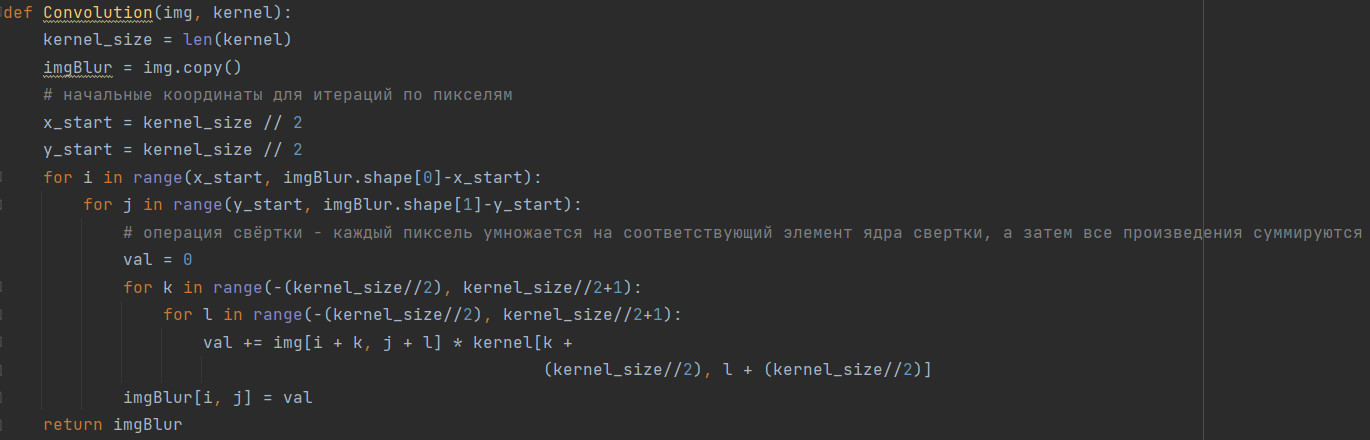


Рисунок 2 – Операция свёртки

Таким образом была выполнена реализация фильтра Гаусса средствами языка программирования. Финальным этапом является вызов соответствующей функции и создание окна для отображения результата в нём. Фрагмент кода, отвечающий за это, представлен на рисунке 3.

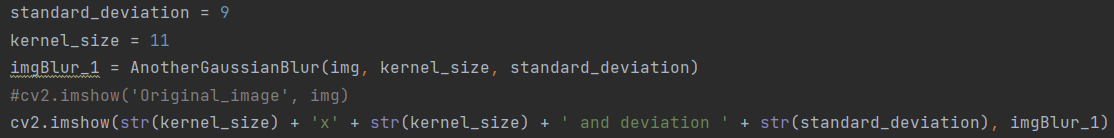


Рисунок 3 – Отображение получившегося результата.

Предварительно было заданно значение стандартного отклонения(9) и размер ядра матрицы свёртки(11). Окно с результатом вызвано метод OpenCV cv2.imshow(). Предварительно исходное изображение было прочтено в чёрно-белом формате с помощью флага cv2.IMREAD\_GRAYSCALE в методе cv2.imread(). Результат работы функции представлен на рисунке 4.

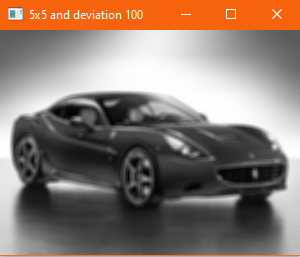


Рисунок 4 – Фильтр Гаусса с ядром 5х5 и стандартным отклонением 100.

Задача реализации фильтра Гаусса методами библиотеки OpenCV является тривиальной, поскольку он уже встроен в библиотеку и вызывается следующим образом: imgBlur\_CV2 = cv2.GaussianBlur(img,(kernel\_size, kernel\_size), standard\_deviation). Окно для отображения аналогичное. Получившееся изображение представлено на рисунке 5.

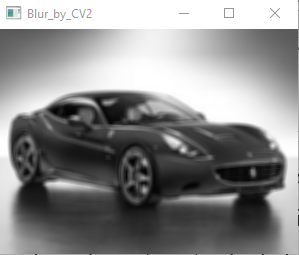


Рисунок 5 – Результат работы встроенного метода с аналогичными параметрами.

Исходя из получившихся изображений можно сделать вывод, что и метод, реализованный средствами языка, и метод, реализованный библиотекой, выдают аналогичное выходное изображение. На рисунка 6 и 7 отображены изображения с другими значениями среднестатистического отклонения и размера матрицы.

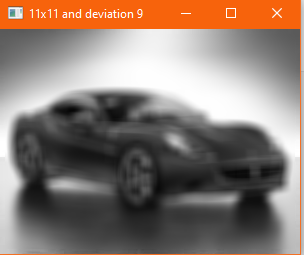


Рисунок 6 – Фильтр Гаусса с ядром 11х11 и стандартным отклонением 9.

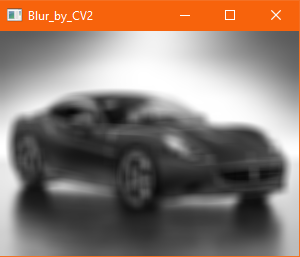


Рисунок 7 – Результат работы встроенного метода с аналогичными параметрами.

Для выполнения последнего задания лабораторной работы программа была переписана на язык C++. Функции фильтра Гаусса, реализованные на языке Python, были переписаны на синтаксис языка С++. Получившийся код прикреплен в Листинге.

**Листинг программ**

Файл GaussianBlur.py

import cv2  
import numpy as np  
  
def main():  
  
 # Задание 1 - построение матрицы Гаусса  
 # чтение изображения в черно-белом формате  
 img = cv2.imread( r'..\media\2.jpg', cv2.IMREAD\_GRAYSCALE)  
  
 standard\_deviation = 9  
 kernel\_size = 11  
 imgBlur\_1 = AnotherGaussianBlur(img, kernel\_size, standard\_deviation)  
 #cv2.imshow('Original\_image', img)  
 cv2.imshow(str(kernel\_size) + 'x' + str(kernel\_size) + ' and deviation ' + str(standard\_deviation), imgBlur\_1)  
  
 # # Задание 4 - Применение фильтра с другими параметрами  
 # standard\_deviation = 50  
 # kernel\_size = 11  
 # imgBlur\_2 = AnotherGaussianBlur(img, kernel\_size, standard\_deviation)  
 # cv2.imshow(str(kernel\_size) + 'x' + str(kernel\_size) + ' and deviation ' + str(standard\_deviation), imgBlur\_2)  
  
 # Задание 5 - Реализация размытие Гаусса встроенным методом OpenCV  
 imgBlur\_CV2 = cv2.GaussianBlur(  
 img, (kernel\_size, kernel\_size), standard\_deviation)  
 cv2.imshow('Blur\_by\_CV2', imgBlur\_CV2)  
 cv2.waitKey(0)  
  
  
# Задание 3 - Реализация фильтра Гаусса средствами языка Python  
def AnotherGaussianBlur(img, kernel\_size, standard\_deviation):  
 kernel = np.ones((kernel\_size, kernel\_size)) # первоначальное ядро свёртки  
 a = b = (kernel\_size+1) // 2 # вычисление центрального элемента матрицы (определения пикселя в фокусе)  
  
 # построение матрицы свёртки  
 for i in range(kernel\_size):  
 for j in range(kernel\_size):  
 kernel[i, j] = gauss(i, j, standard\_deviation, a, b) # вычисление функции Гаусса  
 #print(kernel)  
  
 # Задание 2 - Нормализация матрицы ядра свёртки  
 sum = 0  
 for i in range(kernel\_size):  
 for j in range(kernel\_size):  
 sum += kernel[i, j]  
 for i in range(kernel\_size):  
 for j in range(kernel\_size):  
 kernel[i, j] /= sum  
 #print(kernel)  
  
 # применение операции свёртки к изображению  
 imgBlur = Convolution(img, kernel)  
 return imgBlur  
  
  
# реализация операции свёртки  
def Convolution(img, kernel):  
 kernel\_size = len(kernel)  
 imgBlur = img.copy()  
 # начальные координаты для итераций по пикселям  
 x\_start = kernel\_size // 2  
 y\_start = kernel\_size // 2  
 for i in range(x\_start, imgBlur.shape[0]-x\_start):  
 for j in range(y\_start, imgBlur.shape[1]-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)]  
 imgBlur[i, j] = val  
 return imgBlur  
  
  
# реализация функции Гаусса  
def gauss(x, y, omega, a, b):  
 omegaIn2 = 2 \* omega \*\* 2  
 m1 = 1/(np.pi \* omegaIn2)  
 m2 = np.exp(-((x-a) \*\* 2 + (y-b) \*\* 2)/omegaIn2)  
 return m1\*m2  
  
  
main()

Файл GaussianBlur.cpp

#include <iostream>  
#include <vector>  
#include <cmath>  
using namespace std;  
  
  
// Реализация функции Гаусса  
double gauss(double x, double y, double omega, double a, double b) {  
 double omegaIn2 = 2 \* pow(omega, 2);  
 double m1 = 1 / (3.14 \* omegaIn2);  
 double m2 = exp(-(pow((x - a), 2) + pow((y - b), 2)) / omegaIn2);  
 return m1 \* m2;  
}  
  
// Реализация фильтра Гаусса  
vector<vector<double>> gaussianBlur(vector<vector<double>>& img, int kernel\_size, double standard\_deviation) {  
 int a = (kernel\_size + 1) / 2;  
 int b = (kernel\_size + 1) / 2;  
  
 // построение матрицы ядра свертки  
 vector<vector<double>> kernel(kernel\_size, vector<double>(kernel\_size));  
 for (int i = 0; i < kernel\_size; ++i) {  
 for (int j = 0; j < kernel\_size; ++j) {  
 kernel[i][j] = gauss(i, j, standard\_deviation, a, b);  
 }  
 }  
  
 // нормализация матрицы ядра свертки  
 double sum = 0;  
 for (int i = 0; i < kernel\_size; ++i) {  
 for (int j = 0; j < kernel\_size; ++j) {  
 sum += kernel[i][j];  
 }  
 }  
 for (int i = 0; i < kernel\_size; ++i) {  
 for (int j = 0; j < kernel\_size; ++j) {  
 kernel[i][j] /= sum;  
 }  
 }  
  
 // применение операции свертки  
 int img\_height = img.size();  
 int img\_width = img[0].size();  
 vector<vector<double>> img\_blurred(img\_height, vector<double>(img\_width));  
 for (int i = a; i < img\_height - a; ++i) {  
 for (int j = b; j < img\_width - b; ++j) {  
 double val = 0;  
 for (int k = -a; k <= a; ++k) {  
 for (int l = -b; l <= b; ++l) {  
 val += img[i + k][j + l] \* kernel[k + a][l + b];  
 }  
 }  
 img\_blurred[i][j] = val;  
 }  
 }  
  
 return img\_blurred;  
}  
  
int main() {  
  
 vector<vector<double>> img = { {1.0, 2.0, 3.0, 4.0, 5.0},  
 {6.0, 7.0, 8.0, 9.0, 10.0},  
 {11.0, 12.0, 13.0, 14.0, 15.0},  
 {16.0, 17.0, 18.0, 19.0, 20.0},  
 {21.0, 22.0, 23.0, 24.0, 25.0} };  
  
 int kernel\_size = 5;  
 double standard\_deviation = 1.0;  
  
 vector<vector<double>> img\_blurred = gaussianBlur(img, kernel\_size, standard\_deviation);  
  
 // вывод размытого изображения  
 for (const auto& row : img\_blurred) {  
 for (const auto& pixel : row) {  
 cout << pixel << " ";  
 }  
 cout << endl;  
 }  
  
 return 0;  
}