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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Романов В.В.

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Крамаренко А.А.

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

**Ход работы:**

Необходимо реализовать алгоритм Канни средствами языка Python и встроенными методами библиотеки OpenCV.

Сначала прочитаем полный адрес файла изображения, затем переведем его в чёрно-белый формат и выведем на экран, применяя размытие Гаусса. Реализуем операцию свёртки и определим матрицы оператора Собеля, представленные на рисунке 1.

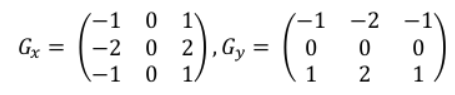


Рисунок 1 – Матрицы оператора Собеля.

После этого к изображению и к каждой из выше определенных матриц была применена операция свёртки и переопределена матрица изображения для работы с внутренними пикселями.

Далее с помощью цикла for, проходящему по каждому пикселю изображения, и формулы была найдена матрица длины вектора градиента

Для поиска матрицы значений углов градиента предварительно необходимо было определить функцию, которая находит округления угла между вектором градиента и осью абсцисс.

Округление нужно, так как нет необходимости точно знать, какова величина угла. Необходимо для направления выбрать лишь один из соседних 8 пикселей, то есть округлить величину угла до 45 градусов. На рисунке 2 представлена классическая математическая схема округления угла.

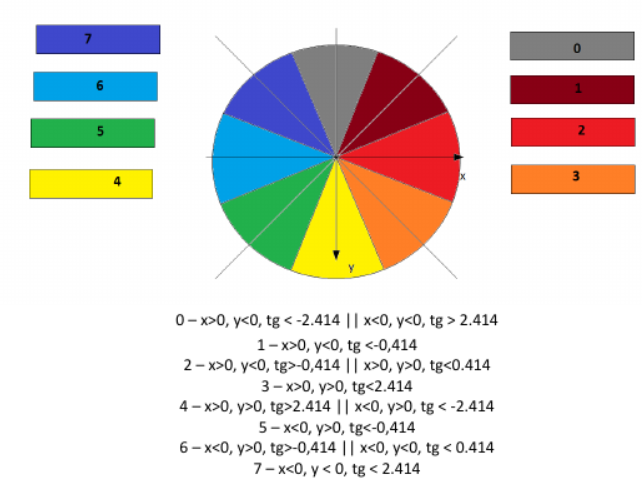


Рисунок 2 – Схема округления угла до 45 градусов.

Кроме того, в рамках данной лабораторной работы требуется подавить немаксимумы. В алгоритме Канни немаксимум – 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.

Для каждого внутреннего пикселя изображения выполнялись следующие действия: сперва значение угла градиента сохранялось в переменной angle, а значение длины градиента - в переменной gradient, затем если пиксель находится на границе изображения, то его значение в матрице img\_border устанавливалось в 0, в противном случае, для каждого пикселя определяется смещение по осям X и Y, которое зависит от значения угла градиента. Оно определялось по следующим правилам:

1) если угол градиента равен 0 или 4 (то есть градиент направлен вдоль оси X), то смещение по оси X будет равно 0, а смещение по оси Y - также равно 0;

2) если угол градиента равен 2 или 6 (то есть градиент направлен вдоль оси Y), то смещение по оси X будет равно 0, а смещение по оси Y - также равно 0;

3) если угол градиента равен 1 (то есть градиент направлен под углом 45 градусов вправо вверх), то смещение по оси X будет равно 1, а смещение по оси Y - равно -1;

4) если угол градиента равен 3 (то есть градиент направлен под углом 45 градусов вправо вниз), то смещение по оси X будет равно -1, а смещение по оси Y - также равно -1;

5) если угол градиента равен 5 (то есть градиент направлен под углом 45 градусов влево вниз), то смещение по оси X будет равно 1, а смещение по оси Y - равно -1;

6) если угол градиента равен 7 (то есть градиент направлен под углом 45 градусов влево вверх), то смещение по оси X будет равно 1, а смещение по оси Y - также равно 1;

В конце с помощью сравнения определялось, являлся ли пиксель максимальным значением градиента.

Последний шагом в алгоритме Канни является построение двойной фильтрации, которая используется 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.

Сначала были заданы пороговые границ для градиента исходя из максимального градиента по изображению. В данном коде нижняя и верхняя границы для длины градиента определялись так, чтобы отфильтровать границы изображения, которые имеют слишком маленькую или слишком большую длину градиента. Нижняя граница вычисляется как max\_gradient / bound\_path, а верхняя - как max\_gradient - max\_gradient / bound\_path. Значение bound\_path задает коэффициент, который определяет, какую часть от максимального значения длины градиента следует использовать для определения нижней и верхней границ. После, аналогично предыдущим случаям, матрица результата была проинициализирована нулями. Для каждого пикселя изображения с помощью цикла for проверялось, находится ли он на границе изображения. Если это так, то выполнялась проверка, находится ли значение длины градиента в заданном диапазоне. Если это соответствует истине, то для каждого пикселя проверяется, есть ли среди его соседей пиксель с максимальной длиной градиента. Для этого использовался двойной цикл for по переменным k и l, принимающим значения от -1 до 1. Затем проверка значение пикселя в матрице границ img\_border для пикселя, расположенного на расстоянии k по оси X и на расстоянии l по оси Y от текущего пикселя. Если значение пикселя равно 255 и значение длины градиента для этого пикселя больше или равно нижней границе, то устанавливается флаг flag в значение True. Если значение длины градиента больше верхней границы, то значение пикселя в результирующей матрице устанавливается в 255. Наконец, она выводится на экран с помощью функции cv2.imshow(). Результаты работы программы представлены ниже на рисунках 3-5.

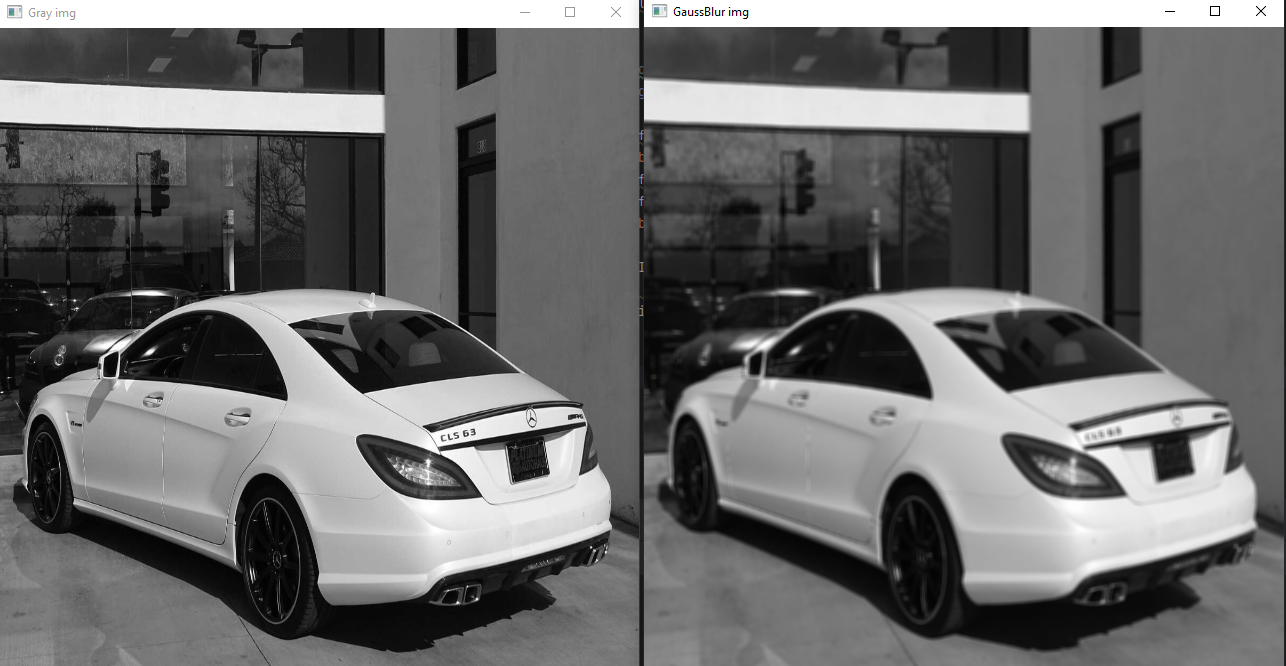


Рисунок 3 – Перевод изображения в серый оттенок и его размытие.



Рисунок 4 – Алгоритм Канни размерностью матрицы свёртки 5х5.

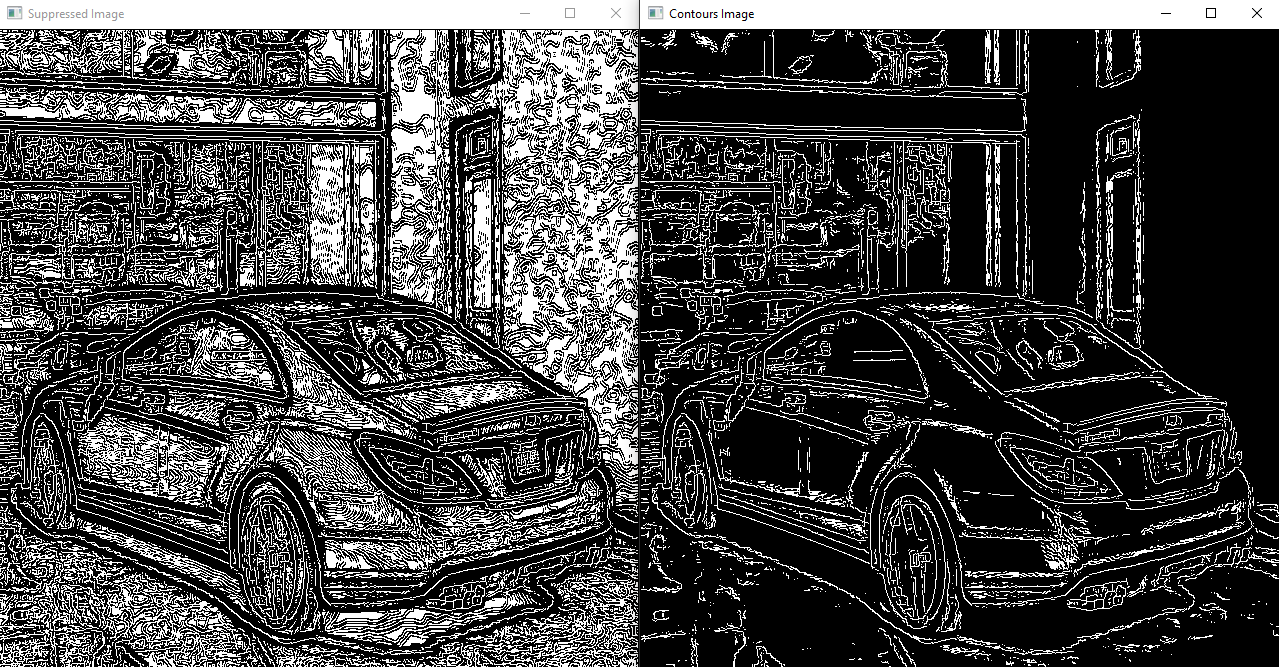


Рисунок 5 – Алгоритм Канни размерностью матрицы свёртки 9х9.

Не трудно заметить, что с увеличением параметров алгоритма, увеличиваются соответствующие матрицы и увеличивается детализированность границ на изображении. Для выполнения задания №6 программа была переписана на ЯП C++, получившийся код прикреплен в Листинге программ.

**Вывод:** освоил принципы выделения контуров объекта на изображении с помощью алгоритма Канни.

**Листинг программ**

Файл **Canny.py**

import cv2  
import numpy as np  
import math  
  
def Image(path, kernelSize = 5, sigmaX=10, sigmaY=10, sizeX=640, sizeY=640):  
 img = cv2.imread(path)  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 img = cv2.resize(img, (sizeX, sizeY))  
 cv2.imshow("Gray img", img)  
  
 imgGaussian = cv2.GaussianBlur(img, (kernelSize, kernelSize), sigmaX=sigmaX, sigmaY=sigmaY)  
 print(imgGaussian)  
 cv2.imshow("GaussBlur img", imgGaussian)  
  
 grads = Gradients(imgGaussian)  
 print(grads)  
 lengths = GradLengths(imgGaussian, grads)  
 print(lengths)  
 corners = Corners(imgGaussian, grads)  
 print(corners)  
  
 suppressed\_img = supressNotMax(lengths, corners)  
 cv2.imshow('Suppressed Image', suppressed\_img)  
  
 edgeImg = checkThreshAndEdge(imgGaussian, suppressed\_img, lengths, 10)  
 cv2.imshow('Contours Image', edgeImg)  
 print(edgeImg)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()  
  
def Gradients(img):  
 gradientMatrix = []  
 for x in range(1, img.shape[0]-1):  
 matrixRow = []  
 for y in range(1, img.shape[1] - 1):  
 Gx = -int(img[x - 1][y - 1]) - 2\*int(img[x][y-1]) - int(img[x + 1][y - 1]) + \  
 int(img[x - 1][y + 1]) + 2\*int(img[x][y + 1]) + int(img[x + 1][y + 1])  
 Gy = -int(img[x - 1][y - 1]) - 2\*int(img[x - 1][y]) - int(img[x - 1][y + 1]) + \  
 int(img[x + 1][y - 1]) + 2\*int(img[x + 1][y]) + int(img[x + 1][y + 1])  
 matrixRow.append((Gx, Gy))  
 gradientMatrix.append(matrixRow)  
 return gradientMatrix  
def GradLengths(img, grads):  
 res = np.zeros((img.shape[0], img.shape[1]))  
 k = 0  
 for i in range(1, img.shape[0]-1):  
 l = 0  
 for j in range(1, img.shape[1]-1):  
 res[i][j] = math.sqrt(grads[k][l][0]\*\*2+grads[k][l][1]\*\*2)  
 l = l + 1  
 k = k + 1  
 return res  
  
def Corner(grad):  
 tang = grad[1]/grad[0] if grad[0] != 0 else 999  
 if grad[0] > 0 and grad[1] < 0 and tang < -2.414 or grad[0] < 0 and grad[1] < 0 and tang > 2.414:  
 return 0  
 elif grad[0] > 0 and grad[1] < 0 and tang < -0.414:  
 return 1  
 elif grad[0] > 0 and grad[1] < 0 and tang > -0.414 or grad[0] > 0 and grad[1] > 0 and tang < 0.414:  
 return 2  
 elif grad[0] > 0 and grad[1] > 0 and tang < 2.414:  
 return 3  
 elif grad[0] > 0 and grad[1] > 0 and tang > 2.414 or grad[0] < 0 and grad[1] > 0 and tang < -2.414:  
 return 4  
 elif grad[0] < 0 and grad[1] > 0 and tang < -0.414:  
 return 5  
 elif grad[0] < 0 and grad[1] > 0 and tang > -0.414 or grad[0] < 0 and grad[1] < 0 and tang < 0.414:  
 return 6  
 elif grad[0] < 0 and grad[1] < 0 and tang < 2.414:  
 return 7  
 if (grad[0] == 0):  
 if (grad[1] > 0):  
 return 4  
 elif (grad[1] <= 0):  
 return 0  
 else:  
 if (grad[1] > 0):  
 return 2  
 elif (grad[1] <= 0):  
 return 6  
  
def Corners(img, grads):  
 corners = np.zeros((img.shape[0], img.shape[1]))  
 k = 1  
 for i in range(len(grads)):  
 l = 1  
 for j in range(len(grads[0])):  
 corners[k][l] = Corner(grads[i][j])  
 l = l + 1  
 k = k + 1  
 return corners  
  
def supressNotMax(gradsLenths, corners):  
 height, width = gradsLenths.shape  
 suppressed = np.zeros\_like(gradsLenths)  
  
 for y in range(1, height - 1):  
 for x in range(1, width - 1):  
 angle = corners[x][y]  
  
 if angle == 0 or angle == 4:  
 q = gradsLenths[x+1][y]  
 r = gradsLenths[x-1][y]  
 elif angle == 1 or angle == 5:  
 q = gradsLenths[x-1][y+1]  
 r = gradsLenths[x + 1][y - 1]  
 elif angle == 2 or angle == 6:  
 q = gradsLenths[x][y+1]  
 r = gradsLenths[x][y-1]  
 elif angle == 3 or angle == 7:  
 q = gradsLenths[x+1][y+1]  
 r = gradsLenths[x-1][y-1]  
  
 if gradsLenths[x][y] >= q and gradsLenths[x][y] >= r:  
 suppressed[x][y] = 255  
 else:  
 suppressed[x][y] = 0  
  
 return suppressed  
def checkThreshAndEdge(img, filteredImg, gradientsLength, boundPath1=10, boundPath2 = 25):  
 maxGradient = np.max(gradientsLength)  
 print(maxGradient)  
 lower\_bound = maxGradient / boundPath1  
 upper\_bound = maxGradient / boundPath2  
 img\_border\_filter = np.zeros(img.shape)  
  
 for i in range(0, img.shape[0]):  
 for j in range(0, img.shape[1]):  
 gradient = gradientsLength[i][j]  
 if (filteredImg[i][j] == 255):  
 if (gradient >= lower\_bound and gradient <= upper\_bound):  
 print(1)  
 flag = False  
 *# проверим соседние пиксели текусщего пикселя* for k in range(-1, 2):  
 for l in range(-1, 2):  
 if (flag):  
 break  
 if (filteredImg[i + k][j + l] == 255 and filteredImg[i + k][j + l] >= lower\_bound):  
 flag = True  
 break  
 if (flag):  
 filteredImg[i][j] = 255  
 elif (gradient > upper\_bound):  
 img\_border\_filter[i][j] = 255  
 return img\_border\_filter  
  
Image("1.jpg")

Файл **Canny.cpp**

#include <opencv2/opencv.hpp>

#include <iostream>

using namespace cv;

using namespace std;

void gradientLength(Mat& src, Mat& dst) {

Mat img\_Gx, img\_Gy;

Sobel(src, img\_Gx, CV\_32FC1, 1, 0);

Sobel(src, img\_Gy, CV\_32FC1, 0, 1);

magnitude(img\_Gx, img\_Gy, dst);

}

void gradientAngle(Mat& src, Mat& dst) {

Mat img\_Gx, img\_Gy;

Sobel(src, img\_Gx, CV\_32FC1, 1, 0);

Sobel(src, img\_Gy, CV\_32FC1, 0, 1);

phase(img\_Gx, img\_Gy, dst);

}

void nonMaximumSuppression(Mat& matr\_gradient, Mat& img\_angles) {

Mat img\_border = matr\_gradient.clone();

for (int i = 1; i < matr\_gradient.rows - 1; i++) {

for (int j = 1; j < matr\_gradient.cols - 1; j++) {

float angle = img\_angles.at<float>(i, j);

float gradient = matr\_gradient.at<float>(i, j);

if (i == 0 || i == matr\_gradient.rows - 1 || j == 0 || j == matr\_gradient.cols - 1) {

img\_border.at<float>(i, j) = 0;

}

else {

int x\_shift = 0;

int y\_shift = 0;

if (angle == 0 || angle == 4) {

x\_shift = 0;

}

else if (angle > 0 && angle < 4) {

x\_shift = 1;

}

else {

x\_shift = -1;

}

if (angle == 2 || angle == 6) {

y\_shift = 0;

}

else if (angle > 2 && angle < 6) {

y\_shift = -1;

}

else {

y\_shift = 1;

}

bool is\_max = gradient >= matr\_gradient.at<float>(i + y\_shift, j + x\_shift) && gradient >= matr\_gradient.at<float>(i - y\_shift, j - x\_shift);

img\_border.at<float>(i, j) = is\_max ? gradient : 0;

}

}

}

imshow("img\_border", img\_border);

}

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

Mat imgCanny;

setlocale(LC\_ALL, "Russian");

std::cout << "Введите имя файла изображения: ";

std::string img\_addr;

std::cin >> img\_addr;

std::cout << "Ищем " + img\_addr << std::endl;

Mat src = imread(img\_addr, IMREAD\_GRAYSCALE);

Mat gradLength(src.size(), CV\_32FC1);

Mat gradAngle(src.size(), CV\_32FC1);

Mat doubleFiltered(src.size(), CV\_8UC1);

gradientLength(src, gradLength);

gradientAngle(src, gradAngle);

nonMaximumSuppression(gradLength, gradAngle);

Canny(src, imgCanny, 50, 150);

imshow("Original\_Image", src);

imshow("Matr\_Gradient", gradLength / 255.0f);

imshow("Matr\_Angle", gradAngle / CV\_PI / 2.0f);

imshow("Double\_Filtration", imgCanny);

waitKey(0);

return 0;

}