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**

**Отчёт**

**по лабораторной работе №4**

**Дисциплина: АЛГОРИТМЫ ЦИФРОВОЙ ОБРАБОТКИ МУЛЬТИМЕДИА**

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

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. А. Корнилов

Направление подготовки 02.03.02 Фундаментальная информатика и

информационные технологии

Направленность (профиль) Математическое и программное обеспечение

компьютерных технологий

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

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

**Цель:** Реализация выделения границ объектов с помощью алгоритма Канни.

**Ход работы:**

1. Был реализован 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.

Для этого сначала создается изображение переводится в grayscale, затем применяется встроенная в opencv реализация шума Гаусса

Код:

def preprocessImage(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))  
 print(img)  
 cv2.imshow("GrayScale image",img)  
  
 imgGaussian = cv2.GaussianBlur(img,(kernelSize,kernelSize),sigmaX=sigmaX,sigmaY=sigmaY)  
 print(imgGaussian)  
 cv2.imshow("Gaussian image",imgGaussian)

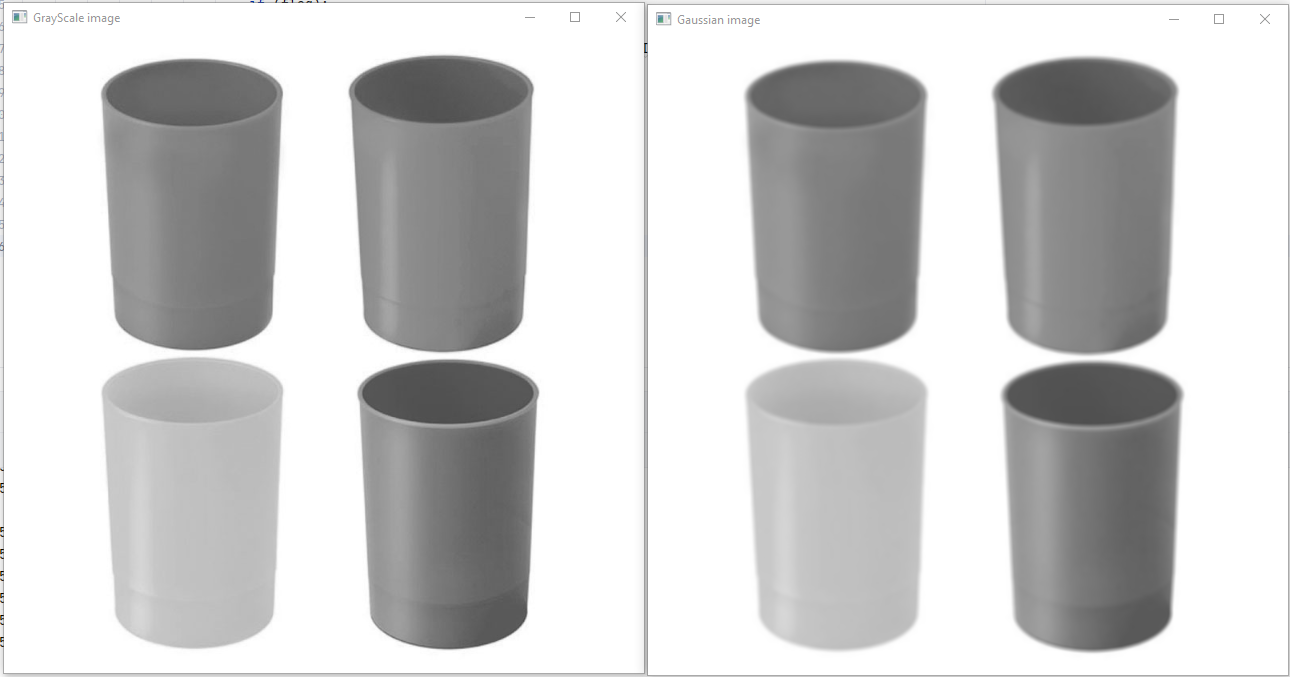


Рисунок 1 – Результат считывания и предобработки изображения

1. 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.

Для этого сначала вычисляется градиент по X и по Y с помощью оператора Собеля. Результаты вычисления записываются в матрицу градиентов. Затем вычисляется матрица длин градиентов с помощью евклидовой метрики. Для вычисления углов используется метод округления на основании знаков градиентов и величины их отношения.

Код:

def calcGradients(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 caclGradLengths(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 calcCorner(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 calcCorners(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] = calcCorner(grads[i][j])  
 l = l + 1  
 k = k + 1  
 return corners

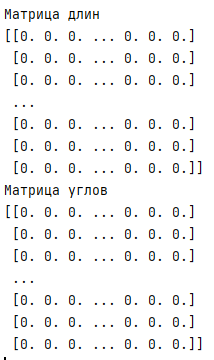


Рисунок 2 – Пример вывода матрицы длин

1. Был модифицирован метод так, чтобы он выполнял подавление немаксимумов и выводил полученное изображение на экран.

Для этого сначала для каждого элемента матрицы длин градиентов определяется соответствующий угол. Затем берутся соответствующие значения пикселей в качестве границ. Если текущая длина градиента больше этих границ, то он закрашивается белым, иначе черным.

Код:

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



Рисунок 3 – Пример изображения после подавления не максимумов

1. Был модифицировать метод так, чтобы он выполнял двойную пороговую фильтрацию и выводил полученное изображение на экран.

Для этого сначала вычисляется максимальная длина градиента, затем вычисляются граничные значения. Если длина превосходит их, то проверяются все соседние пиксели. Если какой-то из них также является граничным, то текущий пиксель остается в итоговом контуре.

Код:

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]  
 *# является лок максимумом* 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

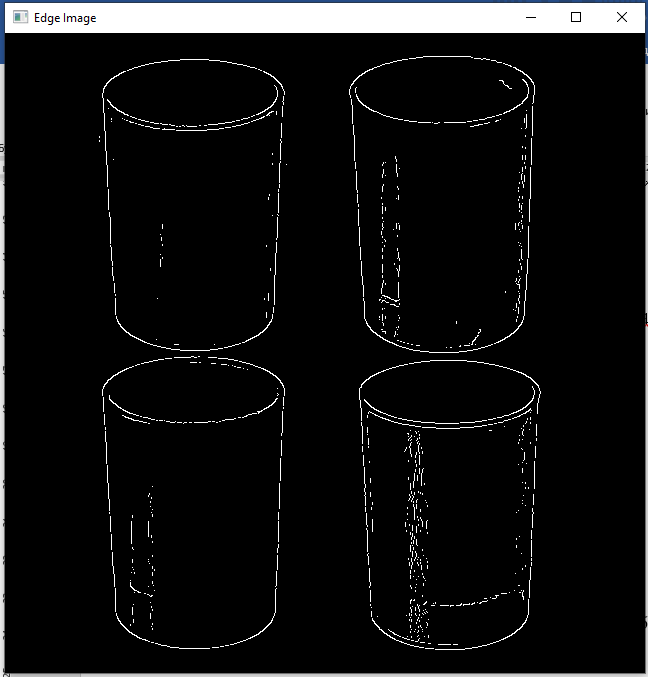


Рисунок 4 – Итоговые контуры

1. Реализован алгоритм средствами языка Kotlin. Для этого используется модуль scrimage, содержащий функции для работы с изображениями

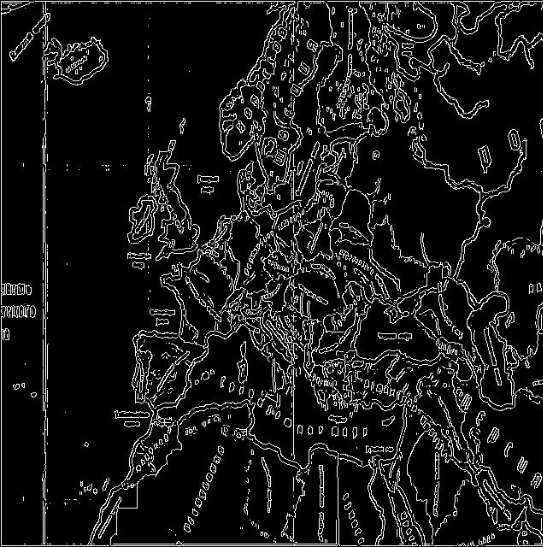


Рисунок 5 – Пример результата работы

**Вывод:** Был реализован алгоритм поиска контуров объектов средствами языков Python и Kotlin.

**1. Принцип операции размытия изображения и ее применение**

Размытие изображения — это операция, которая снижает резкость и детализацию изображения, сглаживая его пиксели. Этот процесс помогает уменьшить шум и улучшить визуальное восприятие, а также подготовить изображение для дальнейших операций обработки, таких как детекция краев или сегментация. Размытие может быть полезно в различных приложениях, включая:

* Устранение шумов в изображениях.
* Подготовка изображений для анализа.
* Улучшение качества при отображении изображений.

**2. Операция свертки матрицы**

Свертка — это математическая операция, которая комбинирует два массива (например, изображение и ядро) для создания нового массива. В контексте обработки изображений свертка применяется для изменения значений пикселей, основываясь на значениях окружающих пикселей. Операция свертки включает:

* Применение ядра (матрицы свертки) к каждому пикселю изображения.
* Вычисление новой интенсивности пикселя на основе значений окружающих пикселей и весов из ядра.

**3. Матрица свертки в размытии по Гауссу**

Матрица свертки для размытия по Гауссу (ядро Гаусса) строится на основе функции Гаусса. Она имеет следующие характеристики:

* Центральный пиксель имеет максимальное значение, и значения уменьшаются по мере удаления от центра.
* Значения в матрице нормализованы так, чтобы сумма всех элементов была равна 1.
* Размер ядра часто выбирается нечетным (например, 3x3, 5x5, 7x7).

**4. Алгоритм размытия по Гауссу**

Алгоритм размытия по Гауссу включает следующие шаги:

1. Определить размер ядра и значение стандартного отклонения (sigma).
2. Создать матрицу Гаусса на основе заданного размера и sigma.
3. Применить свертку к изображению, используя созданное ядро Гаусса.
4. Для каждого пикселя изображения:
   * Учитывать значения пикселей в области вокруг него в соответствии с весами из ядра.
   * Вычислить новое значение пикселя, суммируя взвешенные значения.

**5. Параметры размытия по Гауссу**

Некоторые параметры размытия по Гауссу включают:

* **Размер ядра**: Определяет, сколько пикселей вокруг центрального пикселя будет учитываться.
* **Стандартное отклонение (sigma)**: Указывает, насколько "широким" будет размытие. Большие значения sigma приводят к более сильному размытия.
* **Граница обработки**: Как обрабатывать пиксели на краях изображения (например, отразить или заполнить).

**6. Фильтрация по Гауссу с использованием OpenCV**

В OpenCV фильтрация по Гауссу выполняется с помощью функции **cv2.GaussianBlur()**.