Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения, материалов и транспорта

Высшая школа автоматизации и робототехники

Отчёт

по лабораторной работе №2

Дисциплина: Техническое зрение

Тема: Медианный фильтр

Студент гр. 3331506/70401 Патрушева А.И.

Преподаватель Варлашин В.В.

« »\_\_\_\_\_\_\_\_2020 г.

Санкт-Петербург

2020

**Задание**

Методами OpenCV реализовать медианный фильтр с ядром 7x7 и якорной точкой в центре, и обработку границ *borderReflect101*.

**Алгоритм работы**

1. Расширение исходного изображения;
2. Создание новых границ с помощью метода *borderReflect101*;
3. Реализация функции поиска медианы в массиве;
4. Проход ядром 7x7 по изображению медианным фильтром;
5. Вывод исходного и полученного изображений.

**Описание класса**

Для реализации медианного фильтра был создан класс *median\_filter,* его методы представлены ниже:

int medianBlur(Mat input, Mat output);

//Увеличение исходного изображения и создание новых границ

int borderReflect101(Mat input, Mat enlargedImage);

//размытие с применением поиска медианы в области, охваченной ядром

void medianFilter(Mat input, Mat output);

//пирамидальная сортировка

void heapsort(uint32\_t\* arr, int32\_t size, int32\_t count);

//поиск медианы

uint32\_t getMedian(uint32\_t\* arr, uint32\_t size);

Для применения медианного фильтра необходимо вызвать функцию *medianBlur()*, принимающую два параметра: исходное изображение и изображение, на котором сохранится результат фильтрации.

**Расширение исходного изображения и обработка краев**

Для обработки краев используется функция *borderReflect101().* Так как якорная точка находится в центре, то сначала в данной функции создается новое изображение на 6 пикселей больше исходного по ширине и высоте, чтобы охватить все точки изображения ядром фильтра 7x7. Исходное изображение копируется в центр расширенного.

Далее созданные края обрабатываются методом *borderReflect101.* Упрощенный пример его работы:

**gfedcb | abcdefgh | gfedcba**,

где ***abcdefgh*** – исходное изображение, gfedcb – левый край после обработки, gfedcba – правый край после обработки.

Реализация обработки правого края

int32\_t rightSideCols = input.cols - 2 ;

//заполнение правого края

for (int i = 3; i < enlargedImage.rows - 3; i++)

{

rightSideCols = input.cols - 2;

for (int j = enlargedImage.cols - 3; j < enlargedImage.cols; j++)

{

for (int channel = 0; channel < 3; channel++)

{

enlargedImage.at<Vec<uint8\_t, 3>>(i, j)[channel] = input.at<Vec<uint8\_t, 3>>(i - 3, rightSideCols)[channel];

}

rightSideCols--;

}

}

Верхний край будет заполняться по тому же принципу, только отражение будет происходить уже по каждому столбцу расширенного изображения, включая ранее созданные правый и левый края.

**Реализация функции поиска медианы в массиве**

Функция medianFilter(), вызываемая в конце основной функции medianBlur(), совершает проход ядром по изображению. Значения пикселей, попавшие в область видимости ядра, записываются в одномерный массив и передаются в функцию *getMedian()* для получения медианы*.* А после, полученное значение записывается вякорную точку на новом изображении.

Функция *getMedian()* вызывает функцию сортировки, а потом просто берет центральный элемент в нечетном массиве.

Для сортировки массива применяется пирамидальная сортировка, использующая бинарное сортирующее дерево, изначально размером с количество элементов. Заполняем его по уровням слева направо, где первый – корень. Цель - поднять наибольший элемент в корень дерева (вершину пирамиды), меняя местами элементы в левой или правой ветви. Найдя наибольший, его меняют местами с элементом с последнего уровня. Повторяем сравнение. Когда найдем второй по величине элемент (появится в корне), меняем его с последним элементом третьего уровня. Каждый раз находя следующий по величине элемент записывает его в отсортированный массив. Повторяем действия, только в последующие разы меняем не с предпоследним элементом, а каждый раз с индексом на 1 меньше.

Для пирамидальной сортировки среднее время поиска медианы big O равен O(n).

**Сравнение библиотечной функции с реализованной**

Для подсчета времени работы фильтра используется функция *clock()*, измеряющая время в миллисекундах*.*

Для сравнения полученных результатов функция *absdiff(),* которая вычитает из полученного изображения эталонное(полученное библиотечной функцией). Чем чернее результат вычитания, тем ближе реализован медианный фильтр к библиотечному. Переменная *nonzeroPixels* показывает количество точек, значение которых после вычитания отличны от нуля.

for (int i = 0; i < result.rows; i++)

{

for (int j = 0; j < result.cols; j++)

{

for (int channel = 0; channel < 3; channel++)

{

if (result.at<Vec<uint8\_t, 3>>(i, j)[0] != 0)

{

nonzeroPixels++;

}

}

}

}

По трем каналам изображения, размером 365x544 получилось 11 772 точек. Отношение к общему количеству пикселей равно 5,9%, причем изображение после вычитания выглядит абсолютно черным. Из этого можно сделать вывод, что отклонение от библиотечной функции незначительно.