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 Ляховский М.В.

Преподаватель Варлашин В.В.

« » _____ 2020 г.

Санкт-Петербург

2020

Задание

Пользуясь средствами языка C++ и библиотеки *OpenCV*, реализовать применение медианного фильтра на изображении. Ядро 7x7 с якорем в левом верхнем углу. Заполнение границ – *Border Wrap*.

Ход работы

Медианный фильтр - один из видов цифровых фильтров, широко используется в цифровой обработке сигналов и изображений для уменьшения уровня шума.

Принцип его работы заключается в сортировке значений ядра и выбор медианы на каждой итерации. Медиана поступает на выход фильтра.

Я выбрал трехканальное *RGB* изображение (рисунок 1) 320x213 пикселей.

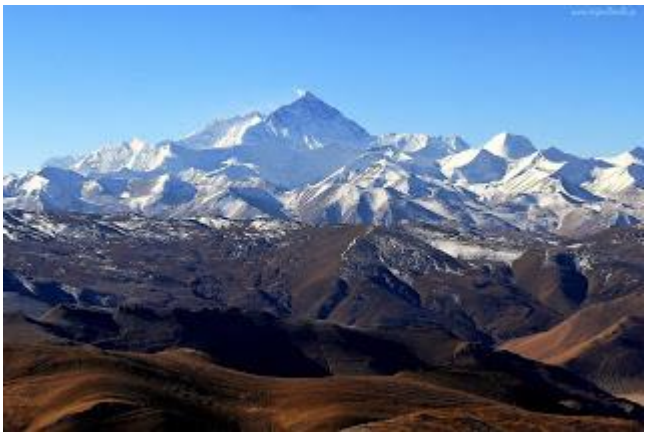

Рисунок 1 – выбранное изображение

Перед применением фильтра нужно добавить границы к изображению, так как по его периметру ядро будет выходить за границы. В моем случае: ядро 7x7, якорь находится слева сверху, значит нужно добавить по 6 пикселей только справа и снизу. При заполнении границ по принципу *border wrap*, копии изображения помещаются со всех сторон. Поэтому справа будет 6 колонок пикселей из левой части, а снизу 6 строк пикселей из верхней части. Реализация кода представлена на рисунке 2.

```
//for 7x7 core
void borderChanger(Mat *img_address, Mat *img_extended_address, BorderTypes border_type)
{
  //with 0.0 anchor position
  if(border_type == BORDER_WRAP)
  {
    //set start: x, y, width, height
    Mat ROI_left(*img_address, Rect(0, 0, 6, img_address->rows));
    Mat right_border(img_address->rows, 6, CV_8UC3, black);
    // Copy the data into new matrix
    ROI_left.copyTo(right_border);
    //add right border (h - horizontal)
    hconcat(*img_extended_address, right_border, *img_extended_address);

    Mat ROI_top(*img_extended_address, Rect(0, 0, img_extended_address->cols, 6));
    Mat bottom_border(6, img_extended_address->cols, CV_8UC3, black);
    // Copy the data into new matrix
    ROI_top.copyTo(bottom_border);
    //add bottom border (v - vertical)
    vconcat(*img_extended_address, bottom_border, *img_extended_address);
  }
  //with 3.3 anchor position
```

Рисунок 2 – реализация *border wrap*

После добавления рамок можно приступить к фильтрации. Для сортировки было написано несколько функций, представленных на рисунке 3.

```
void HeapSort(uint8_t* array, int numberOfElements, int i)
{
  int largest = i;

  int left = 2 * i + 1;
  int right = 2 * i + 2;

  if (left < numberOfElements && array[left] > array[largest])
    largest = left;

  if (right < numberOfElements && array[right] > array[largest])
    largest = right;

  if (largest != i)
  {
    swap(array[largest], array[i]);
    HeapSort(array, numberOfElements, largest);
  }
}

void Sort(uint8_t* arr, const int size)
{
  for (int i = size / 2 - 1; i >= 0; i--)
  {
    HeapSort(arr, size, i);
  }

  for (int i = size - 1; i >= 0; i--)
  {
    swap(arr[i], arr[0]);
    HeapSort(arr, i, 0);
  }
}

void Swap(int& a, int& b)
{
  int c = b;

  b = a;

  a = c;
}
```

Рисунок 3 – реализация сортировки

Непосредственно для фильтрации нужно копировать значения из пикселей, которые попадают в ядро на текущей итерации. Реализация представлена на рисунке 4.

```
73  void takePart(Mat *roi_address, uint8_t *core_address, uint8_t channel)
74  {
75    uint8_t index = 0;
76    Vec3b pixel(0, 0, 0);
77
78    //vertical
79    for (uint16_t j = 0; j < 7; j++)
80    {
81      //horizontal
82      for (uint16_t i = 0; i < 7; i++)
83      {
84        pixel = roi_address->at<Vec3b>(j, i);
85        core_address[index] = pixel.val[channel];
86        index++;
87      }
88    }
89  }
```

Рисунок 4 – копирование в ядро

Фильтр работает отдельно по каждому каналу. В основном цикле внутри двух циклов для движения по вертикали и горизонтали вызывается фильтр, передаются координаты текущего положения ядра. Функция фильтра вызывает функцию для копирования пикселей на текущем шаге в ядро, затем вызывает функцию сортировки и результат записывает в соответствующий пиксель в новом изображении. Затем те же действия повторяются для второго и третьего канала. Реализация представлена на рисунке 5.

```
213   //vertical
214   for(core_position.y = 0; core_position.y < img.rows; core_position.y++)
215   {
216     //horizontal
217     for(core_position.x = 0; core_position.x < img.cols; core_position.x++)
218     {
219       roi_rectangle.x = core_position.x;
220       roi_rectangle.y = core_position.y;
221       ROI = img_extended(roi_rectangle);
222
223       filter(&new_img, &ROI, core);
224     }
225   }
```

```
91  void filter(Mat *new_img_address, Mat *roi_address, uint8_t *core_address)
92  {
93    //red channel
94    takePart(roi_address, core_address, 0);
95    Sort(core_address, 49);
96    new_img_address->at<Vec3b>(core_position.y, core_position.x).val[0] = core_address[24];
97
98    //green channel
99    takePart(roi_address, core_address, 1);
100   Sort(core_address, 49);
101   new_img_address->at<Vec3b>(core_position.y, core_position.x).val[1] = core_address[24];
102
103   //blue channel
104   takePart(roi_address, core_address, 2);
105   Sort(core_address, 49);
106   new_img_address->at<Vec3b>(core_position.y, core_position.x).val[2] = core_address[24];
107 }
```

Рисунок 5 – реализация фильтра

Сравнение с методом из *OpenCV*

Метод медианного фильтра в *OpenCV* реализован на заполнении границ по принципу *border replicate*. Якорь в середине. Поэтому для корректного сравнения я добавил такую же реализацию у себя.

```
131   //with 3.3 anchor position
132   else if(border_type == BORDER_REPLICATE)
133   {
134     //take first row
135     Mat ROI = img_address->row(0);
136     //copy to empty matrix
137     Mat ROI_up = ROI;
138     //add first row twice
139     vconcat(ROI_up, ROI, ROI_up);
140     vconcat(ROI_up, ROI, ROI_up);
141
142     //take last row
143     ROI = img_address->row(img_address->rows-1);
144     //copy to empty matrix
145     Mat ROI_down = ROI;
146     //add last row twice
147     vconcat(ROI, ROI_down, ROI_down);
148     vconcat(ROI, ROI_down, ROI_down);
149
150     //add first and last rows to origin img
151     vconcat(ROI_up, *img_extended_address, *img_extended_address);
152     vconcat(*img_extended_address, ROI_down, *img_extended_address);
153
154     //take first column of extended img
155     ROI = img_extended_address->col(0);
156     //copy to empty img
157     Mat ROI_left = ROI;
158     //add twice
159     hconcat(ROI_left, ROI, ROI_left);
160     hconcat(ROI_left, ROI, ROI_left);
161
162     //take last column of extended img
163     ROI = img_extended_address->col(img_extended_address->cols-1);
164     //copy to empty img
165     Mat ROI_right = ROI;
166     //add twice
167     hconcat(ROI_right, ROI, ROI_right);
168     hconcat(ROI_right, ROI, ROI_right);
169
170     hconcat(*img_extended_address, ROI_right, *img_extended_address);
171     hconcat(ROI_left, *img_extended_address, *img_extended_address);
172   }
173 }
```

Рисунок 6 – реализация *border replicate*

Сравнение времени работы представлено в таблице 1.

Таблица 1 – сравнение времени работы методов

Конфигурация решения	Ручная	*OpenCV*
Debug	1401 мс	9,5 мс
Release	493 мс	8,9 мс

Пример работы фильтра показан на рисунке 6.

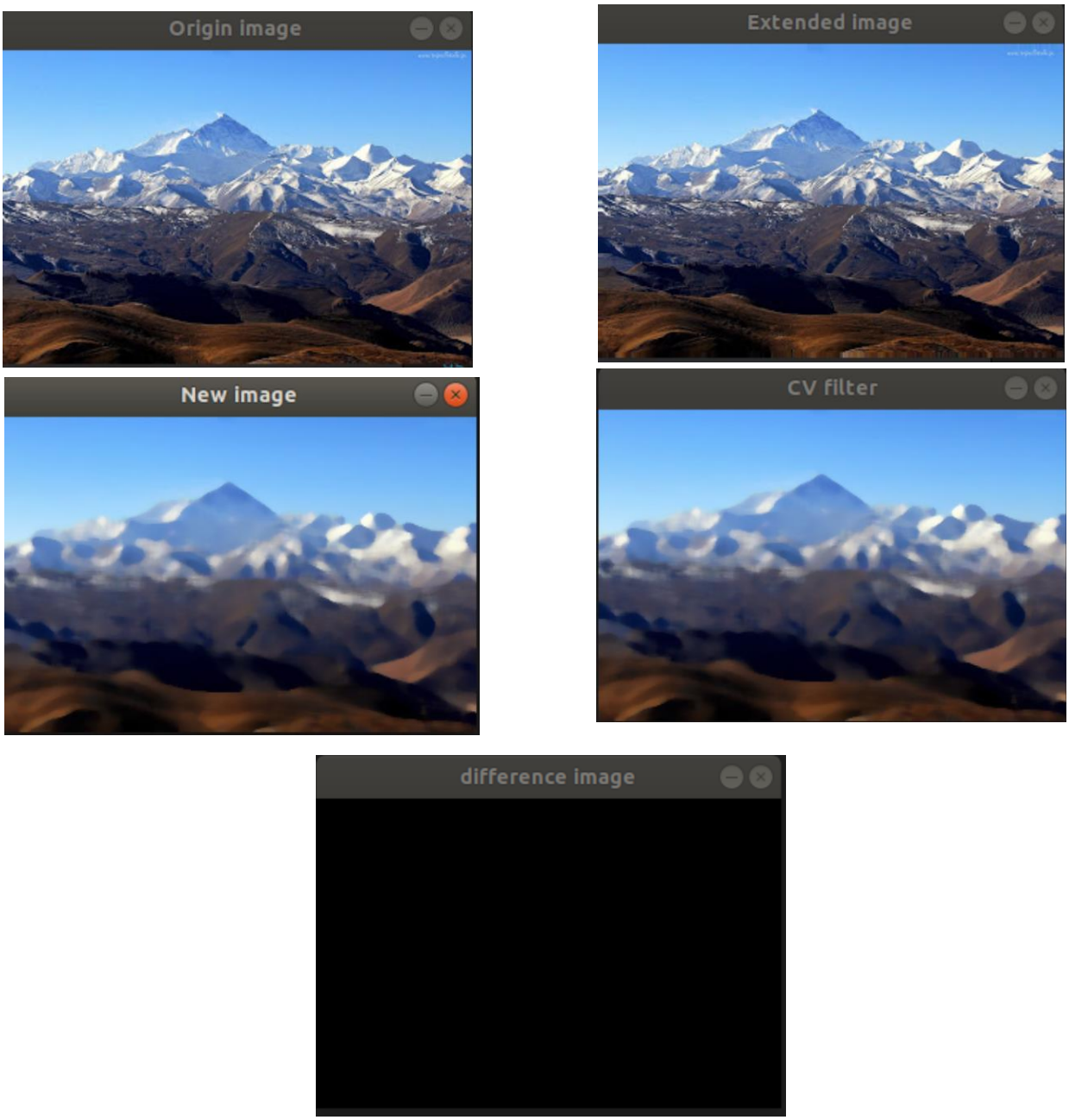

Рисунок 6 – пример работы фильтров