маСанкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт о лабораторной работе №3**

**Дисциплина:** Методы и средства цифровой обработки информации

Выполнил студент гр. 13541/1 А.А. Дроздовский

(подпись)

Руководитель Н.А. Абрамов

(подпись)

« » 2017 г.

Санкт – Петербург

2017

**СОДЕРЖАНИЕ**

[**1.** **Цель работы** 3](#_Toc496848237)

[**2.** **Ход работы** 3](#_Toc496848238)

[**2.1.** **Фильтр Гаусса** 3](#_Toc496848239)

[**2.2.** **Оператор Лапласиана** 4](#_Toc496848240)

[**2.3.** **Повышение резкости** 7](#_Toc496848241)

[**3.** **Выводы** 9](#_Toc496848242)

[***ПРИЛОЖЕНИЕ*** 11](#_Toc496848243)

[Текст программы 11](#_Toc496848244)

1. **Цель работы**

Изучить методы нахождения границ на изображении с помощью оператора Лапласиана;

изучить метод сглаживания изображения на примере фильтра Гаусса;

изучить метод повышения резкости изображения.

1. **Ход работы**
   1. **Фильтр Гаусса**

;

, где

– выходное изображение после обработки;

– исходное изображение;

– матрица свертки.

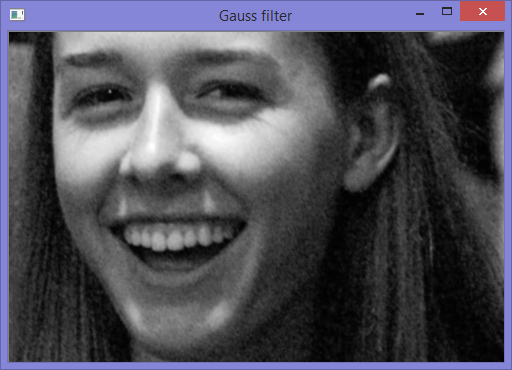
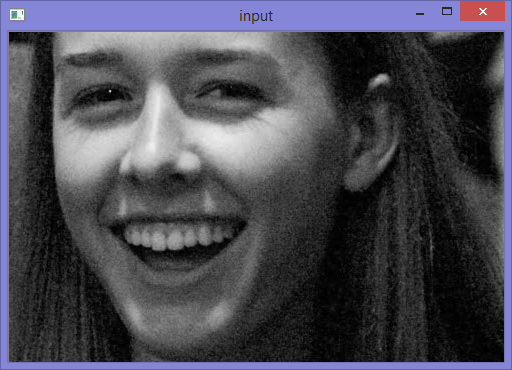


Рис. 2.1. Результат работы фильтра Гаусса. (Изображение 1)

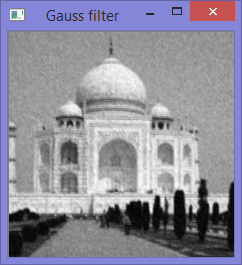
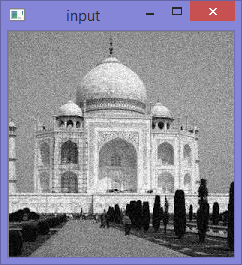


Рис. 2.2. Результат работы фильтра Гаусса. (Изображение 2)



Рис. 2.3. Результат работы фильтра Гаусса. (Изображение 3)

* 1. **Оператор Лапласиана**

;

, где

– выходное изображение после обработки;

– исходное изображение после сглаживания;

– матрица свертки.

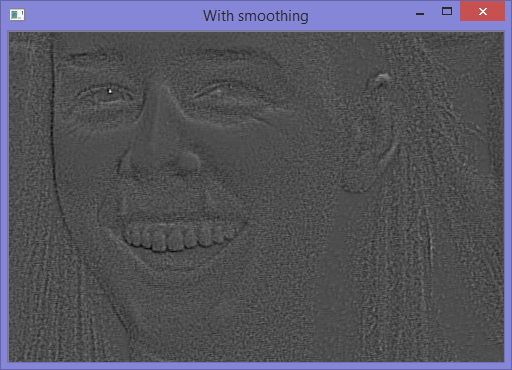
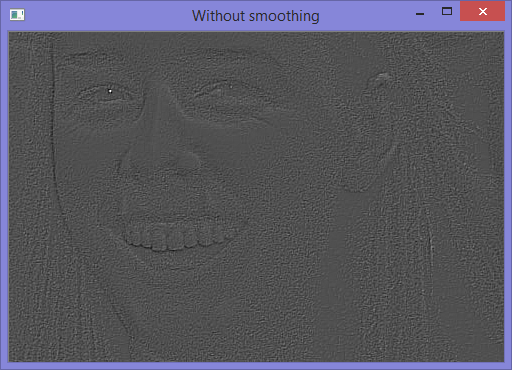


Рис. 2.4. Результат работы оператора Лапласиана. (Изображение 1)

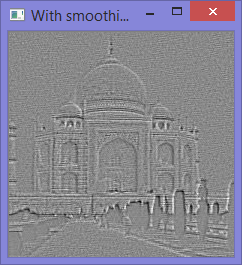
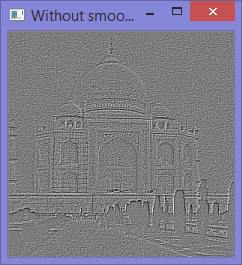


Рис. 2.5. Результат работы оператора Лапласиана. (Изображение 2)

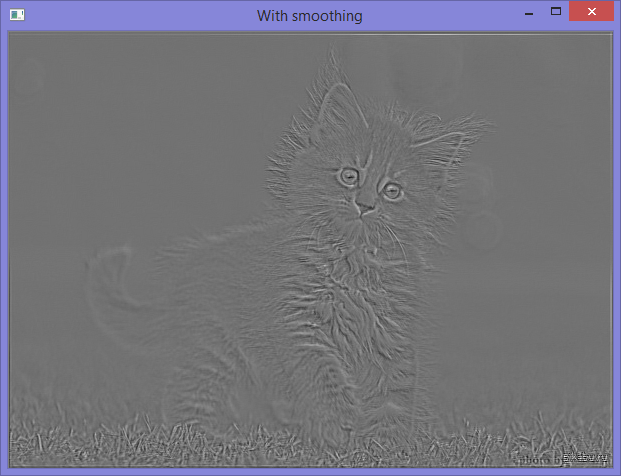
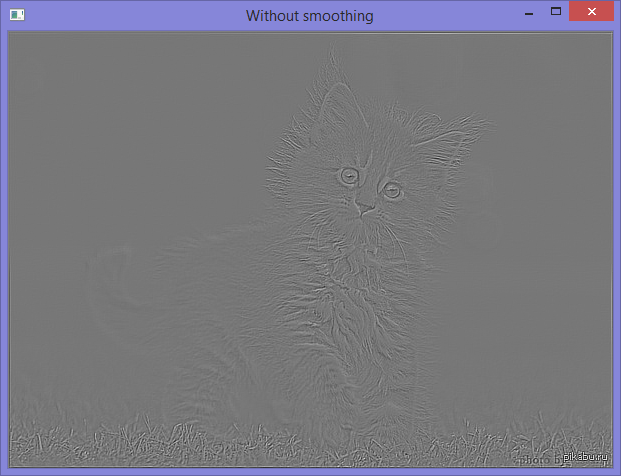


Рис. 2.5. Результат работы оператора Лапласиана. (Изображение 3)

Ниже представлено сравнение изображений с найденным границами, с использованием различных операторов. Для удобства сравнения, матрица, полученная с применением оператора Лапласиана была преобразована к виду :

|  |  |
| --- | --- |
| Рис. 2.6. Лапласиан. | Рис. 2.7.Робертс. |
| Рис. 2.8. Прюитт. | Рис. 2.9. Собель. |
| Рис. 2.10. Лапласиан. | Рис. 2.11. Робертс. |
| Рис. 2.12. Прюитт. | Рис. 2.13. Собель. |

* 1. **Повышение резкости**

Для повышения резкости на 30% матрицу изображения с границами нормализовать в диапазон от 1 до 1,3. Для нахождения границ подходит любой из ранее изученных операторов. В данной работе применяются оператор Лапласиана и оператор Прюитта.

, где

– выходное и исходное изображения;

– нормализованная матрица границ.

|  |  |
| --- | --- |
| Рис. 2.14. Размытое изображение 1. | |
| Рис. 2.15. | Рис. 2.16. |
| Рис. 2.17. | Рис. 2.18. |
| Рис. 2.19. | |
| Рис. 2.20. | Рис. 2.21. |
| Рис. 2.22. | Рис. 2.23. |

1. **Выводы**

Сглаживающие фильтры используются в основном для снижения шума. Чем больше окно фильтрации, тем меньше средняя интенсивность шума. Побочным эффектом фильтра является размытие деталей изображения. Существенным недостатком фильтра усреднения является то, что все пиксели в маске фильтра на любом расстоянии от обрабатываемой точки оказывают одинаковое влияние на результат. Матрицу, которую использует фильтр Гаусса обеспечивает более эффективное снижение шума, т.к. влияние каждого пикселя уменьшается с расстоянием.

Оператор Лапласиана имеет маску размером 3х3. При использование данного оператора для нахождения границ без предварительного сглаживания (применения фильтра Гаусса) в большинстве случаев не удается получить желаемого результата, поэтому приходится предварительно обрабатывать изображение, что ведет к снижению быстродействия алгоритма. В сравнение с операторами, рассмотренными в прошлой лабораторной работе, преимуществ не выявлено.

В обоих примерах видно значительное улучшения качества изображения. Изменяя диапазон нормированной матрицы можно увеличивать или уменьшать качество обработки. При использование различных операторов для получения изображения с границами существенной разницы не наблюдается.

# ***ПРИЛОЖЕНИЕ***

## Текст программы

#include "stdafx.h"

using namespace std;

using namespace cv;

#define To\_Float static\_cast<float>

template <typename \_Tp\_out, typename \_Tp>

void Matrix\_multiplication(Mat\_<\_Tp\_out> &out, Mat\_<\_Tp> image\_with\_frame, int K[9]);

template <typename \_Tp>

Mat\_<uchar> Frame(Mat\_<\_Tp> input);

template <typename \_Tp>

Mat\_<uchar> Convert(Mat\_<\_Tp> input);

template<typename \_Tp>

Mat\_<uchar> Formulation(Mat\_<\_Tp> img, String S, int Kx[9], int Ky[9]);

template <typename \_Tp>

void Matrix\_addition(Mat\_<\_Tp> &G, Mat\_<\_Tp> Gx, Mat\_<\_Tp> Gy);

int main()

{

int Laplacian[9] = { -1,-1,-1,-1,8,-1,-1,-1,-1 };

int Gauss\_filter[9] = { 1,2,1,2,4,2,1,2,1 };

int Sobel\_operator\_x[9] = { 1,0,-1,2,0,-2,1,0,-1 };

int Sobel\_operator\_y[9] = { 1,2,1,0,0,0,-1,-2,-1 };

int Prewitt\_operator\_x[9] = { -1,0,1,-1,0,1,-1,0,1 };

int Prewitt\_operator\_y[9] = { 1,1,1,0,0,0,-1,-1,-1 };

int Roberts\_operator\_x[9] = { 1,0,0,0,-1,0,0,0,0 };

int Roberts\_operator\_y[9] = { 0,1,0,-1,0,0,0,0,0 };

namedWindow("LR3");

String imageName("LR3\_2.jpg");

Mat\_<uchar> image\_input // Исходное изображение в оттенках серого

= imread(imageName.c\_str(), IMREAD\_GRAYSCALE);

Mat\_<short> image\_1 // Изображение для нахождения границ

= Mat\_<short>(image\_input.rows, image\_input.cols);

Mat\_<float> image\_2 // Изображение для повышения резкости

= Mat\_<float>(image\_input.rows, image\_input.cols);

// Вывод исходного изображения на экран

imshow("input", image\_input);

/\*-----------=== Часть 1 ===-----------\*/

// Применение Лапласиана без сглаживания

Mat\_<uchar> image\_with\_frame // Изображение с рамкой

= Frame(image\_input);

Matrix\_multiplication(image\_1, image\_with\_frame, Laplacian);

imshow("Without smoothing", Convert(image\_1));

// Применение фильтра Гауcса 3х3

// для сглаживания изображения

Matrix\_multiplication(image\_1, image\_with\_frame, Gauss\_filter);

normalize(image\_1, image\_1, // Линейное растяжение

-32767,32767, NORM\_MINMAX, -1, Mat());

for (int i = 0; i < image\_1.rows\*image\_1.cols; i++)

image\_1.at<short>(i) = round(float(image\_1.at<short>(i)) / 16);

imshow("Gauss filter", Convert(image\_1));

// Применение Лапласиана для выделения границ

image\_with\_frame // Изображение после фильтра Гаусса с рамкой

= Frame(Convert(image\_1));

Matrix\_multiplication(image\_1, image\_with\_frame, Laplacian);

imshow("With smoothing", Convert(image\_1));

/\*-----------=== Часть 2 ===-----------\*/

// Повышение резкости

normalize(image\_1, image\_1, 10, // от 1 до 1.3

13, NORM\_MINMAX, -1, Mat());

for (int i = 0; i < image\_2.rows\*image\_2.cols; i++)

image\_2.at<float>(i) = // input \* Kоэф\_резкости

round(To\_Float(image\_input.at<uchar>(i))\*To\_Float(image\_1.at<short>(i)) / 10);

imshow("High Boost Filter", Convert(image\_2));

waitKey(0);

return 0;

}

/\*\*

\* Преобразование Mat\_<typename> в Mat\_<uchar>

\*

\* Исходная матрица изображения нормализуется

в формат оттенков пикселей от 0 до 255.

Затем выводится в качестве выходного параметра.

\* @input Mat\_<typename> исходное изображение

\* @return Mat\_<uchar>

\*/

template <typename \_Tp>

Mat\_<uchar> Convert(Mat\_<\_Tp> input)

{

Mat\_<uchar> output(input.rows, input.cols);

normalize(input, input, 0, // Линейное растяжение

255, NORM\_MINMAX, -1, Mat());

output = input;

return output;

}

/\*\*

\* Умножение пикселя на маску

\*

\* Исходное изображение с рамкой

умножется на маску К

\* @img\_wf Mat\_<typename> исходное изображение c рамкой

\* @K - маска 3х3

\* @out Mat\_<typename> выходная матрица

\*/

template <typename \_Tp\_out, typename \_Tp>

void Matrix\_multiplication(Mat\_<\_Tp\_out> &out, Mat\_<\_Tp> img\_wf, int K[9])

{

for (int i = 1; i < img\_wf.rows - 1; i++)

for (int j = 1; j < img\_wf.cols - 1; j++)

out.at<\_Tp\_out>(i - 1, j - 1) = round(

K[0] \* To\_Float(img\_wf.at<\_Tp>(i - 1, j - 1)) + K[1] \* To\_Float(img\_wf.at<\_Tp>(i - 1, j)) + K[2] \* To\_Float(img\_wf.at<\_Tp>(i - 1, j + 1))

+ K[3] \* To\_Float(img\_wf.at<\_Tp>(i - 1, j - 1)) + K[4] \* To\_Float(img\_wf.at<\_Tp>(i, j)) + K[5] \* To\_Float(img\_wf.at<\_Tp>(i - 1, j + 1))

+ K[6] \* To\_Float(img\_wf.at<\_Tp>(i + 1, j + 1)) + K[7] \* To\_Float(img\_wf.at<\_Tp>(i + 1, j)) + K[8] \* To\_Float(img\_wf.at<\_Tp>(i + 1, j + 1)));

}

/\*\*

\* Создание изображения с рамкой

\*

\* К исходному изображению добавляется по

1 пикселю с каждой стороны. Каждый из которых

заполняется соседним оттенком

\* @input Mat\_<typename> исходное изображение

\* @return Mat\_<uchar>

\*/

template <typename \_Tp>

Mat\_<uchar> Frame(Mat\_<\_Tp> input)

{

Mat\_<\_Tp> output(input.rows + 2, input.cols + 2);

// Копия изображения с белой рамкой:

for (int i = 1; i <= input.rows; i++)

for (int j = 1; j <= input.cols; j++)

output.at<\_Tp>(i, j) = input.at<\_Tp>(i - 1, j - 1);

// Заполнение белой рамки соседними цветами:

// Заполнение вехний и нижний границы

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

{

output.at<\_Tp>(0, j) = output.at<\_Tp>(1, j);

output.at<\_Tp>(output.rows - 1, j) = output.at<\_Tp>(output.rows - 2, j);

}

// Заполнение левой и правой границы

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

{

output.at<\_Tp>(i, 0) = output.at<\_Tp>(i, 1);

output.at<\_Tp>(i, output.cols - 1) = output.at<\_Tp>(i, output.cols - 2);

}

return Convert(output);

}

/\*\*

\* Нахождение границ в изображение

\*

\* С помощью оператора Кх находятся границы

по вертикали. С помощью оператора Ку находятся

границы по горизонтали. Далее с помощью функции

"Matrix\_addition" получаем выходное изображение,

содержащее в себе границы исходного изображения.

\* @img - Mat\_<typename> исходное изображение

\* @S - String Название оператора

\* @Kx - int[9] оператор Х

\* @Ky - int[9] оператор Y

\* @return Mat\_<uchar> изображение с границами

\*/

template<typename \_Tp>

Mat\_<uchar> Formulation(Mat\_<\_Tp> img, String S, int Kx[9], int Ky[9])

{

Mat\_<\_Tp> Gx(img.rows, img.cols);

Mat\_<\_Tp> Gy(img.rows, img.cols);

Mat\_<\_Tp> G(img.rows, img.cols);

Matrix\_multiplication(Gx, Frame(img), Kx);

Matrix\_multiplication(Gy, Frame(img), Ky);

Matrix\_addition(G, Gx, Gy);

normalize(Gx, Gx, -32767, // Линейное растяжение

32767, NORM\_MINMAX, -1, Mat());

normalize(Gy, Gy, -32767,

32767, NORM\_MINMAX, -1, Mat());

normalize(G, G, -32767,

32767, NORM\_MINMAX, -1, Mat());

// Вывод полученных изображений на экран

imshow(S + " : Gx", Gx);

imshow(S + " : Gy", Gy);

imshow(S + " : G", G);

return Convert(G);

}

/\*\*

\* Сложение матриц

\*

\* Сложение матриц, содержащих

границы по оси Х и по оси Y

по формуле sqrt(Gx^2+Gy^2)

\* @Gx - Mat\_<typename> Границы по вертикали

\* @Gy - Mat\_<typename> Границы по горизонтали

\* @G - Mat\_<typename> Выходное изображение с границами

\*/

template <typename \_Tp>

void Matrix\_addition(Mat\_<\_Tp> &G, Mat\_<\_Tp> Gx, Mat\_<\_Tp> Gy)

{

for (int i = 0; i < G.rows \* G.cols; i++)

G.at<short>(i) = sqrt((To\_Float(Gx.at<short>(i)))\*(To\_Float(Gx.at<short>(i)))

+ (To\_Float(Gy.at<short>(i)))\*(To\_Float(Gy.at<short>(i))));

}