**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

отчет

**по лабораторной работе №1**

**по дисциплине «Интеллектуальные видеосистемы»**

Тема: Цифровая фильтрация изображений

Вариант 3

|  |  |  |
| --- | --- | --- |
| Студенты гр. 8571 |  | Темный Д.А.  Мальцева О.Н. |
| Преподаватель |  | Поздеев А.А. |

Санкт-Петербург

2023

**Цель работы**

Целью лабораторной работы является практическая реализация и анализ методов фильтрации цифровых изображений. В соответствии с заданием на моделирование необходимо разработать программу в виде консольного приложения, реализующую следующие функции:

1. Загрузку растрового изображения и вывод его на экран.
2. Фильтрацию изображений различными фильтрами, указанными в таблице 1.1.
3. Получение контурных препаратов изображения с помощью алгоритма DoG, детектора границ Кэнни, а также фильтров выделения контуров в соответствии с индивидуальным заданием (оператор Собела для наклонных контуров).

Таблица 1.1. Задание на фильтрацию изображений

|  |  |  |
| --- | --- | --- |
|  | **Вид фильтра** | **Размер апертуры** |
| 1 | Фильтр Гаусса | 3 |
| 2 | Фильтр Гаусса | 5 |
| 3 | Фильтр «Мозаика» | 3 |
| 4 | Фильтр апертурный коррекции | 3 |
| 5 | Медианный фильтр | 3 |

**Теоретические сведения**

Цифровые фильтры применяют для увеличения резкости изображения (двумерная апертурная коррекция) или сглаживания изображения c целью уменьшения заметности шумов; гребенчатые фильтры используют для разделения спектров сигналов яркости и цветности в системах NTSC и PAL. Цифровые фильтры могут быть линейными, нелинейными и логическими.

Процесс фильтрации заключается в том, что отсчеты фильтрованного сигнала выражают через отсчеты исходного сигнала и отсчеты усеченной по спектру (чтобы ее можно было выразить через отдельные отсчеты по Котельникову) импульсной реакции фильтра . Цифровой фильтр преобразует последовательность отсчетов входного сигнала в выходную последовательность , причем выходной сигнал в каждый момент времени представляет собой взвешенную сумму входных сигналов: Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание (1)

- алгоритм нерекурсивного (трансверсального) фильтра.

При обработке изображений используют двумерные линейные преобразования. Для двумерного случая выражение (1) можно переписать в следующем виде: Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описаниегде ;; взятые на прямоугольном окне отсчеты сигналов и импульсной реакции фильтра. Это математическая запись алгоритма, который реализуют все программы фильтрации изображений.

Различают два класса цифровых фильтров: фильтры нижних частот и фильтры верхних частот. Фильтры нижних частот применяют для сглаживания изображений, снижения влияния зернистости изображений из-за шумов, подавления помех дискретизации и ослабления муар эффекта. F - матрица свертки, маска или апертура, размером n×n. Элементы, которые охватывает маска, умножают на соответствующие им коэффициенты, результаты складывают и делят на нормирующее число k, равное сумме коэффициентов маски. Значения коэффициентов в матрице F представляют собой отсчеты двумерного импульсного отклика изображающей системы на воздействие также двумерной дельта функции. Для оптической системы – это функция рассеяния точки; для телевизионной − апертура. Параметрами апертуры являются ее диаметр, форма и закон распределения прозрачности. При цифровой фильтрации диаметр апертуры – это размерность матрицы маски, значения ее коэффициентов − отсчеты функции распределения прозрачности , а форму апертуры обычно принимают квадратной Фокусом маски называют ее центральный элемент.

Фильтры верхних частот применяют для повышения резкости изображения. Резкость изображения можно повысить, если усилить скачки яркости на границах объекта. Это так называемая апертурная коррекция.

Рангово-порядковые (медианные) фильтры относят к отдельному классу сглаживающих фильтров, которые позволяют эффективно подавлять точечные помехи и шумы, сохраняя при этом резкие перепады яркости на границах объектов. Формально медианный фильтр описывают так: Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание, где , а – нечетное число, чтобы у маски был центральный элемент (фокус).

**Фильтрация изображения**

Исходное изображение представлено на рис. 1.



Рис. 1 – Исходное изображение

Коэффициенты апертуры для фильтра Гаусса:

- размером 3x3 (рис. 2):

- размером 5x5 (рис. 3):



Рис. 2 – Результат обработки фильтром Гаусса, апертура размером 3x3



Рис. 3 – Результат обработки фильтром Гаусса, апертура размером 5x5

Коэффициенты апертуры для фильтра «Мозаика» (рис. 4):



Рис. 4 – Результат обработки фильтром «Мозаика»

Коэффициенты апертуры для фильтра апертурной коррекции при глубине коррекции = 20% (рис. 5):



Рис. 5 – Результат обработки фильтром апертурной коррекции

Результат обработки медианным фильтром представлен на рис. 6.



Рис. 6 – Результат обработки медианным фильтром

**Получение контура изображения**

Результат разности гауссиан представлена на рис. 7.



Рис. 7 – Разность гауссиан (яркость увеличена в 14 раз)

Результат обработки исходного изображения фильтром Кэнни представлен на рис. 8.

Изображение выглядит как черный, вода, черно-белый, темнота

Автоматически созданное описание

Рис. 8 – Результат обработки фильтром Кэнни

Был выбран оператор Собела для наклонных контуров (рис. 9). Коэффициенты апертуры:



Рис. 9 – Результат обработки оператором Собела для наклонных контуров

**Вывод**

В ходе работы была исследован способ фильтрации шумов с помощью свертки и применения различных фильтров: Гаусса с различным размером окна апертуры, «Мозаика», апертурной коррекции и медианного фильтра.

Фильтр апертурной коррекции показал самую высокую контрастность и резкость полученного изображения по сравнению с прочими результатами, а медианный фильтр – наибольшим сглаживанием без сильного размытия границ объектов.

Также были рассмотрены способы получения контурного препарата: разность гауссиан (модуль разности изображений после обработки фильтром Гаусса с размером апертуры 3x3 и 5x5), фильтр Кэнни и оператор Собела для наклонных контуров.

Результат разности гауссиан обладает наименьшей яркостью контуров, так как для них используется разница между мало различающимися нормировками пикселов. Детектор границ Кэнни предоставляет тонкие контрастные линии, однако обладает малой чувствительностью, из-за чего на изображении присутствуют крупные черные области без контуров. Оператор Собела показывает результат с наиболее различимыми объектами и видимой текстурой в местах, где предыдущие два способа отображали только минимальную яркость.

**Листинг Кода**

**main.cpp (фильтрация изображений):**

#include <iostream>

#include <string>

#include <opencv2/opencv.hpp>

#include <gauss\_filter.h>

#include <mosaic\_filter.h>

#include <median\_filter.h>

#include <aperture\_correction\_filter.h>

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

    std::ios\_base::sync\_with\_stdio(false);

    std::cin.tie(NULL);

    std::string filename = "../kitten.png";

    std::string output\_folder = "output";

    int gauss\_aperture\_size = 5;

    int mosaic\_aperture\_size = 3;

    int aper\_correction\_aperture\_size = 3;

    int median\_aperture\_size = 3;

    float aper\_correction\_percentage = 20;

    std::cout << "Input filter id for showing only one pic (1-4, 0 for skip): ";

    int filterId = 1;

    std::cin >> filterId;

    cv::Mat image = cv::imread(filename, cv::IMREAD\_GRAYSCALE);

    switch (filterId)

    {

        case 1:

        {

            std::cout << "Gaussian filter\n";

            std::cout << "Input aperture size (3, 5, 7 ...): ";

            std::cin >> gauss\_aperture\_size;

            cv::Mat gauss\_img;

            gauss(image, gauss\_img, gauss\_aperture\_size);

            show\_img("gaussian output "+filename, gauss\_img);

            break;

        }

        case 2:

        {

            std::cout << "Mosaic filter\n";

            cv::Mat mosaic\_img;

            mosaic(image, mosaic\_img, mosaic\_aperture\_size);

            show\_img("mosaic output "+filename, mosaic\_img);

            break;

        }

        case 3:

        {

            std::cout << "Aperture correction filter\n";

            std::cout << "Input percentage (20-30 max): ";

            std::cin >> aper\_correction\_percentage;

            cv::Mat aper\_cor\_img;

            aperture\_correction(image, aper\_cor\_img, aper\_correction\_aperture\_size, aper\_correction\_percentage);

            show\_img("aperture correction output "+filename, aper\_cor\_img);

            break;

        }

        case 4:

        {

            std::cout << "Median filter\n";

            cv::Mat median\_img;

            median\_f(image, median\_img, median\_aperture\_size);

            show\_img("median output "+filename, median\_img);

            break;

        }

        default:

            break;

    }

    std::cout << "\nSave all? (0/1) ";

    int confirm = 0;

    std::cin >> confirm;

    if (confirm)

    {

        std::cout << "\nShow all? (0/1) ";

        std::cin >> confirm;

        std::string create\_folder = "mkdir " + output\_folder;

        system(create\_folder.c\_str());

        output\_folder += "/";

        cv::Mat gauss\_img;

        gauss(image, gauss\_img, 3);

        cv::imwrite(output\_folder + "gauss3\_img.jpg", gauss\_img);

        gauss(image, gauss\_img, 5);

        cv::imwrite(output\_folder + "gauss5\_img.jpg", gauss\_img);

        cv::Mat mosaic\_img;

        mosaic(image, mosaic\_img, mosaic\_aperture\_size);

        cv::imwrite(output\_folder + "mosaic\_img.jpg", mosaic\_img);

        cv::Mat aper\_cor\_img;

        aperture\_correction(image, aper\_cor\_img, aper\_correction\_aperture\_size, aper\_correction\_percentage);

        cv::imwrite(output\_folder + "aper\_cor\_img.jpg", aper\_cor\_img);

        cv::Mat median\_img;

        median\_f(image, median\_img, median\_aperture\_size);

        cv::imwrite(output\_folder + "median\_img.jpg", median\_img);

        if (confirm)

        {

            show\_img("gaussian output "+filename, gauss\_img);

            show\_img("mosaic output "+filename, mosaic\_img);

            show\_img("aperture correction output "+filename, aper\_cor\_img);

            show\_img("median output "+filename, median\_img);

        }

    }

    std::cout << "\n";

    show\_img("input "+filename, image);

    if (confirm != 0)

        cv::waitKey(0);

}

**gauss\_filter.h (функция фильтра Гаусса):**

#include <vector>

#include <opencv2/opencv.hpp>

#include <utils/utils.h>

void gauss(const cv::Mat &input\_img, cv::Mat &output\_img, int aperture\_size = 3) {

    std::cout << "\nGaussian filter with ";

    std::cout << aperture\_size << " aperture\n";

    output\_img = cv::Mat::zeros(input\_img.size(), CV\_8U);

    double k = 0; // коэффициент нормировки

    std::vector<std::vector<float>> Fk;

    int frameWidth = aperture\_size/2;

    int index = 4;

    for (int i = 0; i <= frameWidth; i++)

    {

        std::vector<float> \*row = new std::vector<float>;

        for (int j = 0; j <= frameWidth; j++)

        {

            row->push\_back(pow(index, j+i));

        }

        for (int j = frameWidth; j > 0; j--)

        {

            row->push\_back(pow(index, j+i-1));

        }

        Fk.push\_back(\*row);

    }

    for (int i = frameWidth; i > 0; i--)

    {

        std::vector<float> \*row = new std::vector<float>;

        for (int j = 0; j <= frameWidth; j++)

        {

            row->push\_back(pow(index, j+i-1));

        }

        for (int j = frameWidth; j > 0; j--)

        {

            row->push\_back(pow(index, j+i-2));

        }

        Fk.push\_back(\*row);

    }

    std::cout << "Aperture:" << std::endl;

    for (auto i: Fk)

    {

        for (auto j: i)

        {

            k += j;

            std::cout << int(j) << " ";

        }

        std::cout << std::endl;

    }

    std::cout << "\nNorm koeff: " << k << "\n";

    for (int i = frameWidth; i < input\_img.cols - frameWidth; i++)

    {

        for (int j = frameWidth; j < input\_img.rows - frameWidth; j++)

        {

            uchar pix\_value = input\_img.at<uchar>(j, i);

            // далее производим свертку

            float Rez = 0;

            for (int ii = -frameWidth; ii <= frameWidth; ii++)

                for (int jj = -frameWidth; jj <= frameWidth; jj++)

                {

                    uchar blurred = input\_img.at<uchar>(j + jj, i + ii);

                    Rez += Fk[ii + frameWidth][jj + frameWidth] \* blurred;

                }

            uchar blurred = ((Rez / k) > 255) ? 255 : (((Rez / k) < 0) ? 0 : (Rez / k)); // осуществляем нормировку

            output\_img.at<uchar>(j, i) = blurred;

        }

        if (( ( i + 1 ) % ( (input\_img.cols - frameWidth) / 10 ) ) == 0)

            progressbar(input\_img.cols - frameWidth, i);

    }

}

**mosaic\_filter.h (функция фильтра «Мозаика»):**

#include <vector>

#include <opencv2/opencv.hpp>

#include <utils/utils.h>

void mosaic(const cv::Mat &input\_img, cv::Mat &output\_img, int aperture\_size = 3) {

    std::cout << "\nMosaic filter\n";

    output\_img = cv::Mat::zeros(input\_img.size(), CV\_8U);

    double k = 0; // коэффициент нормировки

    std::vector<std::vector<float>> Fk;

    int frameWidth = aperture\_size/2;

    int index = 4;

    float power;

    for (int i = 0; i <= frameWidth; i++)

    {

        std::vector<float> \*row = new std::vector<float>;

        for (int j = 0; j <= frameWidth; j++)

        {

            power = pow(index, j+i);

            row->push\_back(power);

        }

        for (int j = frameWidth; j > 0; j--)

        {

            power = pow(index, j+i-1);

            row->push\_back(power);

        }

        Fk.push\_back(\*row);

    }

    for (int i = frameWidth; i > 0; i--)

    {

        std::vector<float> \*row = new std::vector<float>;

        for (int j = 0; j <= frameWidth; j++)

        {

            power = pow(index, j+i-1);

            row->push\_back(power);

        }

        for (int j = frameWidth; j > 0; j--)

        {

            power = pow(index, j+i-2);

            row->push\_back(power);

        }

        Fk.push\_back(\*row);

    }

    std::cout << "Aperture:" << std::endl;

    for (auto i: Fk)

    {

        for (auto j: i)

        {

            k += j;

            std::cout << int(j) << " ";

        }

        std::cout << std::endl;

    }

    for (int i = frameWidth; i < input\_img.cols - frameWidth; i += aperture\_size)

    {

        for (int j = frameWidth; j < input\_img.rows - frameWidth; j += aperture\_size)

        {

            uchar pix\_value = input\_img.at<uchar>(j, i);

            // далее производим свертку

            float Rez = 0;

            for (int ii = -frameWidth; ii <= frameWidth; ii++)

                for (int jj = -frameWidth; jj <= frameWidth; jj++)

                {

                    uchar blurred = input\_img.at<uchar>(j + jj, i + ii);

                    Rez += Fk[ii + frameWidth][jj + frameWidth] \* blurred;

                }

            uchar blurred = Rez / k; // осуществляем нормировку

            for (int ii = -frameWidth; ii <= frameWidth; ii++)

                for (int jj = -frameWidth; jj <= frameWidth; jj++)

                {

                    output\_img.at<uchar>(j + jj, i + ii) = blurred;

                }

        }

        if (( ( i + 1 ) % ( (input\_img.cols - frameWidth) / 10 ) ) == 0)

            progressbar(input\_img.cols - frameWidth, i);

    }

}

**aperture\_correction\_filter.h (функция фильтра апертурной коррекции):**

#include <vector>

#include <opencv2/opencv.hpp>

#include <utils/utils.h>

void aperture\_correction(const cv::Mat &input\_img, cv::Mat &output\_img, int aperture\_size, float percentage) {

    std::cout << "\nAperture correction filter\n";

    output\_img = cv::Mat::zeros(input\_img.size(), CV\_8U);

    double k = 0; // коэффициент нормировки

    std::vector<std::vector<float>> Fk;

    int frameWidth = aperture\_size/2;

    std::cout << "\n\n" << frameWidth << "\n\n";

    int index = ceil((100.0/percentage - 1) + 8);

    for (int i = 0; i < frameWidth; i++)

    {

        std::vector<float> \*row = new std::vector<float>;

        for (int j = 0; j < aperture\_size; j++)

        {

            row->push\_back(-1);

        }

        Fk.push\_back(\*row);

    }

    std::vector<float> \*row = new std::vector<float>;

    for (int i = 0; i < aperture\_size; i++)

    {

        if (i == frameWidth)

            row->push\_back(index);

        else

            row->push\_back(-1);

    }

    Fk.push\_back(\*row);

    for (int i = frameWidth; i > 0; i--)

    {

        std::vector<float> \*row = new std::vector<float>;

        for (int j = 0; j < aperture\_size; j++)

        {

            row->push\_back(-1);

        }

        Fk.push\_back(\*row);

    }

    std::cout << "Aperture:" << std::endl;

    for (auto i: Fk)

    {

        for (auto j: i)

        {

            k += j;

            std::cout << int(j) << " ";

        }

        std::cout << std::endl;

    }

    for (int i = frameWidth; i < input\_img.cols - frameWidth; i++)

    {

        for (int j = frameWidth; j < input\_img.rows - frameWidth; j++)

        {

            uchar pix\_value = input\_img.at<uchar>(j, i);

            // далее производим свертку

            float Rez = 0;

            for (int ii = -frameWidth; ii <= frameWidth; ii++)

                for (int jj = -frameWidth; jj <= frameWidth; jj++)

                {

                    uchar blurred = input\_img.at<uchar>(j + jj, i + ii);

                    Rez += Fk[ii + frameWidth][jj + frameWidth] \* blurred;

                }

            uchar blurred = ((Rez / k) > 255) ? 255 : (((Rez / k) < 0) ? 0 : (Rez / k)); // осуществляем нормировку

            output\_img.at<uchar>(j, i) = blurred;

        }

        if (( ( i + 1 ) % ( (input\_img.cols - frameWidth) / 10 ) ) == 0)

            progressbar(input\_img.cols - frameWidth, i);

    }

}

**median\_filter.h (функция медианного фильтра):**

#include <vector>

#include <numeric>

#include <opencv2/opencv.hpp>

#include <utils/utils.h>

void median\_f(const cv::Mat &input\_img, cv::Mat &output\_img, int aperture\_size) {

    std::cout << "\nMedian filter\n";

    output\_img = cv::Mat::zeros(input\_img.size(), CV\_8U);

    std::vector<float> Fk;

    int frameWidth = aperture\_size/2;

    for (int i = frameWidth; i < input\_img.cols - frameWidth; i++)

    {

        for (int j = frameWidth; j < input\_img.rows - frameWidth; j++)

        {

            uchar pix\_value = input\_img.at<uchar>(j, i);

            // далее производим свертку

            float Rez = 0;

            for (int ii = -frameWidth; ii <= frameWidth; ii++)

                for (int jj = -frameWidth; jj <= frameWidth; jj++)

                {

                    Fk.push\_back(input\_img.at<uchar>(j + jj, i + ii));

                }

            std::sort(Fk.begin(), Fk.end());

            uchar blurred = Fk[aperture\_size\*aperture\_size/2]; // осуществляем нормировку

            Fk.clear();

            output\_img.at<uchar>(j, i) = blurred;

        }

        if (( ( i + 1 ) % ( (input\_img.cols - frameWidth) / 10 ) ) == 0)

            progressbar(input\_img.cols - frameWidth, i);

    }

}

**main.h (препарация изображений):**

#include <iostream>

#include <string>

#include <opencv2/opencv.hpp>

#include <sobel\_outline.h>

#include <laplas\_outline.h>

#include <lab1/include/gauss\_filter.h>

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

    std::string filename = "../kitten.png";

    std::string output\_folder = "output";

    cv::Mat image = cv::imread(filename, cv::IMREAD\_GRAYSCALE);

    cv::Mat gauss3\_img;

    cv::Mat gauss5\_img;

    cv::Mat gauss\_diff;

    gauss(image, gauss3\_img, 3);

    gauss(image, gauss5\_img, 5);

    cv::absdiff(gauss3\_img, gauss5\_img, gauss\_diff);

    cv::Mat canny\_img;

    float thr1 = 50;

    float thr2 = 200;

    cv::Canny(image, canny\_img, thr1, thr2);

    cv::Mat sobel\_diagonal\_img;

    sobel\_diagonal(image, sobel\_diagonal\_img);

    show\_img("diagonal sobel output "+filename, sobel\_diagonal\_img);

    break;

    int confirm = 0;

    std::cout << "\nSave all? (0/1) ";

    std::cin >> confirm;

    if (confirm)

    {

        std::cout << "\nShow all? (0/1) ";

        std::cin >> confirm;

        std::string create\_folder = "mkdir " + output\_folder;

        system(create\_folder.c\_str());

        output\_folder += "/";

        cv::imwrite(output\_folder + "gauss\_diff.jpg", gauss\_diff\*14);

        cv::imwrite(output\_folder + "canny\_img.jpg", canny\_img);

        cv::Mat sobel\_diagonal\_img;

        sobel\_diagonal(image, sobel\_diagonal\_img);

        cv::imwrite(output\_folder + "sobel\_diagonal\_img.jpg", sobel\_diagonal\_img);

        if (confirm)

        {

            show\_img("gauss dif "+filename, gauss\_diff);

            show\_img("canny  "+filename, canny\_img);

            show\_img("diagonal sobel output "+filename, sobel\_diagonal\_img);

        }

    }

    std::cout << "\n";

    show\_img("input "+filename, image);

    if (confirm != 0)

        cv::waitKey(0);

}

**sobel\_outline.h (функция оператора собела):**

#include <vector>

#include <opencv2/opencv.hpp>

#include <utils/utils.h>

#include <base\_outline.h>

void sobel\_diagonal(const cv::Mat &input\_img, cv::Mat &output\_img)

{

    std::cout << "\nSobel diagonal filter\n";

    output\_img = cv::Mat::zeros(input\_img.size(), CV\_8U);

    std::vector<std::vector<float>> Fk;

    Fk = {

        {-2, -1, 0},

        {-1, 0, 1},

        {0, 1, 2}

    };

    base\_outline(input\_img, output\_img, Fk);

}

**base\_outline.h (функция применения контурных операторов):**

# pragma once

#include <vector>

#include <opencv2/opencv.hpp>

#include <utils/utils.h>

void base\_outline(const cv::Mat &input\_img, cv::Mat &output\_img, std::vector<std::vector<float>> &Fk)

{

    int frameWidth = Fk.size()/2;

    std::cout << "Aperture:" << std::endl;

    for (auto i: Fk)

    {

        for (auto j: i)

        {

            std::cout << int(j) << " ";

        }

        std::cout << std::endl;

    }

    for (int i = frameWidth; i < input\_img.cols - frameWidth; i++)

    {

        for (int j = frameWidth; j < input\_img.rows - frameWidth; j++)

        {

            uchar pix\_value = input\_img.at<uchar>(j, i);

            // далее производим свертку

            float Rez = 0;

            for (int ii = -frameWidth; ii <= frameWidth; ii++)

                for (int jj = -frameWidth; jj <= frameWidth; jj++)

                {

                    uchar blurred = input\_img.at<uchar>(j + jj, i + ii);

                    Rez += Fk[ii + frameWidth][jj + frameWidth] \* blurred;

                }

            output\_img.at<uchar>(j, i) = (Rez > 255) ? 255 : ((Rez < 0) ? 0 : Rez);

        }

        if (( ( i + 1 ) % ( (input\_img.cols - frameWidth) / 10 ) ) == 0)

            progressbar(input\_img.cols - frameWidth, i);

    }

}

**utils.h (вспомогательные функции вывода):**

#pragma once

#include <string>

#include <iostream>

#include <iomanip>

#include <opencv2/opencv.hpp>

int progressbar(float total\_items, int iter, int bar\_len = 40)

{

    int filled\_len = int(float(bar\_len \* iter) / total\_items);

    float percents = 100.0 \* float(iter) / total\_items;

    std::string bar = "";

    int i = 0;

    while ( i++ < filled\_len)

    {

        bar += "=";

    }

    i = 0;

    while ( i++ < (bar\_len - filled\_len))

    {

        bar += "-";

    }

    std::cout << std::fixed << std::setprecision(3);

    std::cout << '[' << bar << "]\t" << percents << '\r';

    return (iter+1);

}

void show\_img(std::string window\_name, cv::Mat img)

{

    int width\_coeff = img.rows/img.cols;

    int height\_coeff = img.cols/img.rows;

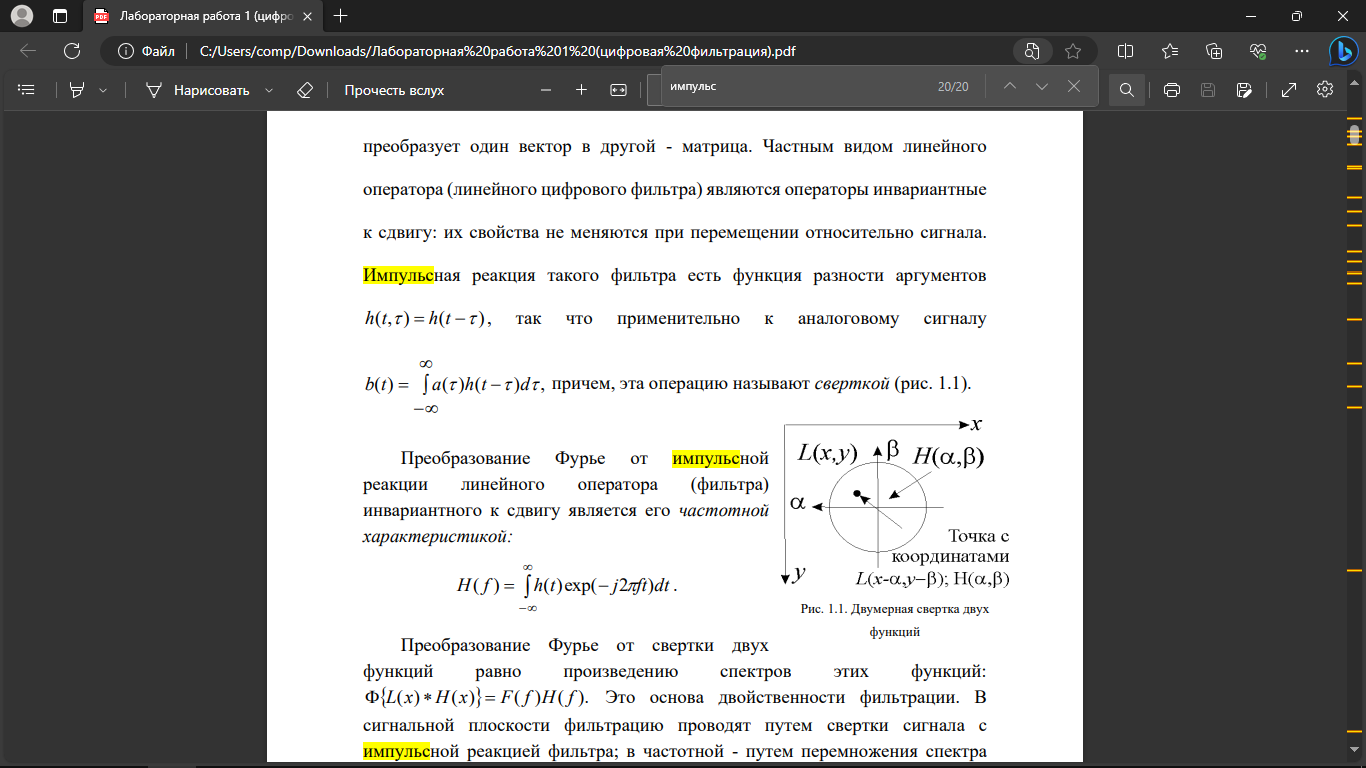
    cv::namedWindow(window\_name, cv::WINDOW\_NORMAL);

    cv::resizeWindow(window\_name, img.cols > 1000\*width\_coeff ? img.cols/2 : img.cols, img.rows > 1000\*height\_coeff ? img.rows/2 : img.rows);

    cv::imshow(window\_name, img);

}

**Вопросы**

1. Поясните разницу между линейными, нелинейными и логическими цифровыми фильтрами. Приведите примеры фильтров каждого класса. Линейные преобразования характерны тем, что для них справедлив принцип суперпозиции. Цифровой фильтр считают линейным, если: сумма любого числа входных сигналов преобразуется в сумму откликов на отдельные слагаемые. Пример: операторы инвариантные к сдвигу.
2. В чем отличие БИХ и КИХ фильтров? КИХ – фильтр с конечной памятью, БИХ – с бесконечной, выходная переменная у таких фильтров в данный момент времени зависит не только от значений входных переменных, но и от значений выходной переменной в предыдущие моменты времени.
3. Что такое импульсная реакция фильтра?  - импульсная реакция линейного цифрового фильтра, свертка. Отсчеты импульсной реакции фильтра можно рассчитать через обратное преобразование Фурье
4. Сравните операцию свертки в сигнальной и спектральной области.

- Сигнальная свертка работает с одномерными сигналами и перемещает ядро по входному сигналу.

- Спектральная свертка работает с двумерными сигналами (изображениями) в частотной области и выполняет умножение Фурье-преобразований.

- Обе операции позволяют обрабатывать и анализировать данные, но применяются в разных контекстах и для разных типов данных.

- Спектральная свертка часто используется в обработке изображений, в то время как сигнальная свертка применяется в аудиообработке и анализе временных рядов.

1. Перечислите и поясните основные шаги синтеза цифрового фильтра

- Необходимо определить характеристики, которые должны иметь цифровой фильтр, такие как частотные характеристики (например, частоту среза и полосу пропускания), амплитудные характеристики (например, уровень подавления), тип фильтра (например, ФНЧ, ФВЧ) и тип ограничений (например Баттерворта, Чебышёва и т. д.).

- На основе требований необходимо спроектировать аналоговый фильтр, который имеет желаемые характеристики. Это может включать в себя выбор типа аналогового фильтра и его параметров, таких как порядок фильтра и коэффициенты передачи.

- Аналоговый фильтр затем преобразуется в цифровой фильтр с использованием методов дискретизации. Это может включать в себя выбор метода дискретизации и определение частотной дискретизации (дискретизации времени).

- Для уменьшения искажений фильтрации, цифровой фильтр может быть умножен на оконную функцию. Эта оконная функция может управлять формой и свойствами фильтра во временной и частотной областях.

- На основе преобразования аналогового фильтра и оконной функции, рассчитываются коэффициенты цифрового фильтра. Эти коэффициенты определяют, как фильтр будет обрабатывать входные данные.

- Фильтр реализуется в виде программного кода или аппаратного устройства, который выполняет операции фильтрации на входных данных.

- После реализации фильтра проводится тестирование его характеристик на тестовых данных. Если необходимо, коэффициенты фильтра могут быть отрегулированы, чтобы удовлетворить требования.

- В зависимости от конкретных требований и ограничений ресурсов можно провести оптимизацию фильтра, чтобы снизить его вычислительную нагрузку или использование памяти.

- Важно документировать характеристики и реализацию фильтра для последующего обслуживания и понимания его работы.