

Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

Отчёт

по лабораторной работе №3

Дисциплина: Техническое зрение

Тема: Дискретное преобразование Фурье

Студент гр. 3331506/70401

Преподаватель

Паньков И.С.

Титов В.В.

«__» _____ 2020 г.

Санкт-Петербург

2020

Цель работы — применение дискретного преобразования Фурье (ДПФ) для фильтрации, свёртки и корреляции изображений.

З а д а н и е

Задание 1. Реализовать прямое ДПФ с возможностью вывода спектра и обратное ДПФ. Сравнить результаты со встроенной функцией.

Задание 2. Реализовать фильтры высоких и низких частот Баттерворта. Выбрать какое-либо изображение и в его спектре обрезать в одном случае элементы спектра с высокими частотами, в другом — с низкими. Затем выполнить обратное преобразование на основе полученных спектров.

Задание 3. Произвести по отдельности свёртку какого-либо изображения с ядром фильтра Собеля, усредняющего Фильтра и фильтра Лапласа. Необходимо вывести магнитуду Фурье-образа исходного изображения и ядра свёртки.

Задание 4. Полученные образы Фурье в результате выполнения свёртки необходимо обратно преобразовать в изображение.

Задание 5. Провести корреляцию изображений автомобильных номеров по очереди с тремя символами. Полученные образ Фурье обратно преобразовать в обычное изображение.

Краткие теоретические сведения

Прямое дискретное преобразование Фурье изображения размером $m \times n$ пикселей вычисляется по формуле

$$F_{jk} = \mathcal{F}(f) = \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} f_{jk} e^{-i \frac{2\pi j}{m}} e^{-i \frac{2\pi k}{n}},$$

где f_{jk} — интенсивность пикселя в j -ой строке и k -ом столбце;

F_{jk} — Фурье-образ изображения в j -ой строке и k -ом столбце.

Обратное дискретное преобразование Фурье этого же изображения, в свою очередь, вычисляется по формуле

$$f_{jk} = \mathcal{F}^{-1}(F) = \frac{1}{m \cdot n} \sum_{j=0}^{m-1} \sum_{k=0}^{n-1} F_{jk} e^{i \frac{2\pi j}{m}} e^{i \frac{2\pi k}{n}}.$$

Многие методы обработки изображений требует вычисления свёртки изображения с другим изображением или ядром фильтра, что при непосредственном вычислении требует временных затрат $O(m^2 n^2)$. В то же самое время произведение (или корреляция) Фурье-образов изображений требует временных затрат $O(m \cdot n)$, что проще с вычислительной точки зрения.

Стоит отметить, правда, что прямое и обратное преобразования Фурье также требуют временных затрат $O(m^2 n^2)$ при их непосредственном вычислении по описанным выше формулам. Для ускорения и упрощения вычислений эти преобразования выполняют с помощью алгоритмов быстрого преобразования Фурье, к которым относятся, например, алгоритм Винограда, алгоритм Гуда — Томаса или алгоритм Кули — Тьюки.

В случае одномерного преобразования Фурье последний алгоритм заключается в том, чтобы отобразить исходное одномерное преобразование

$$F_k = \sum_{j=0}^{n-1} f_j e^{-i \frac{2\pi k}{n} j}, \quad k = \overline{0, n-1},$$

в двумерное

$$F_{n''k'+k''} = \sum_{j''=0}^{n''-1} \sum_{j'=0}^{n'-1} f_{j'+n'j''} e^{-i \frac{j'+n'j''}{n}} e^{-i \frac{n''k'+k''}{n}}, \quad j = j' + n'j'', \quad n''k' + k'', \quad n = n' \cdot n''.$$

При этом если n' и n'' — простые числа, то вычисления упрощаются многократно, а если составные — преобразование можно разбивать дальше, упрощая тем самым вычисления. Наиболее простым является тот случай, когда исходный размер n является степенью двойки. В этом случае метод называется алгоритмом Кули — Тьюки по основанию 2. Также достаточно простым является случай, когда размер является составным числом, полученным произведением двоек, троек и пятёрок. Этот метод называется алгоритмом Кули — Тьюки по малому основанию.

Случай двумерного преобразования Фурье отличается лишь в том, что преобразование Фурье необходимо сначала применить последовательно ко всем строкам изображения, получив Фурье-образы строк, а затем — ко всем столбцам полученных Фурье-образов. Алгоритм Кули — Тьюки может быть многократно улучшен и ускорен с помощью применения различных методов, описанных в соответствующей литературе, но в данной работе будет реализован лишь классический рекурсивный алгоритм с двух-, трёх- и пятиточечной схемами преобразований Фурье.

Применение алгоритмов быстрого преобразования Фурье позволяет многократно ускорить вычисления и упростить обработку изображений, их временная сложность в случае двумерного преобразования оценивается как $O(m \cdot n \log(m \cdot n))$.

Класс FastFurierTransformer

Для выполнения прямых и обратных дискретных преобразований Фурье был разработан класс FastFurierTransformer. Этот класс предоставляет сеттеры и геттеры для доступа к изображению, спектру изображения и его (спектра) размеру, а также методы, дающие возможность проводить быстрое преобразование Фурье по двух-, трёх- и пятиточечной схемам рекурсивным методом. Также этот класс позволяет быстро выводить изображения и их спектры в нормализованном формате.

В листинге 1 представлен исходный код заголовочного файла класса FastFurierTransformer.

Листинг 1 — Исходный код файла fast_furier_transformer.h

```
#pragma once

#define _USE_MATH_DEFINES

#include "filter.h"
#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"

#include <stdint.h>
#include <math.h>

enum ComplexPart
{
    RE = 0,
    IM = 1,
    NUM_OF_COMPLEX_PARTS = 2
};

void shiftSpectrum(cv::Mat1f& spectrum, const int32_t shiftX, const int32_t shiftY);
void multiplySpectrums(cv::Mat2f& first, cv::Mat2f& second, cv::Mat2f& result, bool isCorr = false);

class FastFurierTransformer
{
public:
    FastFurierTransformer() = default;
    ~FastFurierTransformer() = default;

    void setImage(const cv::Mat1f& image);
    void setImage(const cv::Mat& image);
    cv::Mat1f getImage() const;
    void showImage(cv::String imageName = "Image");

    void setSpectrum(const cv::Mat2f& spectrum);
    void setSpectrum(const cv::Mat& spectrum);
    cv::Mat2f getSpectrum() const;
    cv::Mat1f getSpectrumMagnitude();
    void showSpectrum(cv::String imageName = "Spectrum");

    void setSpectrumSize(cv::Size2i spectrumSize);
    cv::Size2i getSpectrumSize() const;
```

```

void setImageShift(cv::Point2i imageShift);
cv::Point2i getImageShift() const;

void setFilter(Filter& filter);
Filter& getFilter();

void directFastFurierTransform();
void inverseFastFurierTransform();
void filtrateImage();

private:
void normalizeSpectrum(cv::Mat1f& spectrumMagnitude, const float min, const float max);
void shiftSpectrum(cv::Mat2f& spectrum, const int32_t shiftX, const int32_t shiftY);

void directFFT(const cv::Mat1f& image, cv::Mat2f& spectrum);
void inverseFFT(const cv::Mat2f& spectrum, cv::Mat1f& image);
void fft(cv::Vec2f *ptr, const int32_t size, const bool isInvert);

cv::Mat1f m_image;
cv::Point2i m_imageShift;
cv::Mat2f m_spectrum;
cv::Size2i m_spectrumSize;
Filter m_filter;
};

```

Прямое и обратное преобразования Фурье вычисляются с помощью методов `directFFT` и `inverseFFT`, реализация которых представлена в листингах 2 и 3 соответственно.

Листинг 2 — Исходный код метода `directFFT`

```

void FastFurierTransformer::directFFT(const Mat1f& image, Mat2f& spectrum)
{
    if (image.empty() == true)
    {
        return;
    }

    auto cols = getOptimalDFTSize(image.cols);
    auto rows = getOptimalDFTSize(image.rows);

    if (cols < m_spectrumSize.width && rows < m_spectrumSize.height)
    {
        cols = m_spectrumSize.width;
        rows = m_spectrumSize.height;
    }
    else
    {
        m_spectrumSize = Size2i(cols, rows);
    }

    auto buffer = Mat2f(Size(cols, rows), Vec2f(0, 0));
    for (auto row = 0; row < image.rows; row++)
    {
        auto imagePtr = image.ptr<float>(row);
        auto spectrumPtr = buffer.ptr<Vec2f>(row);
        for (auto col = 0; col < image.cols; col++)
        {
            spectrumPtr[col][RE] = imagePtr[col];
        }
    }
}

```

```

auto dfft = [this, &buffer](int32_t rows, int32_t cols)
{
    for (auto row = 0; row < rows; row++)
    {
        auto spcPtr = buffer.ptr<Vec2f>(row);

        fft(spcPtr, cols, false);
    }
    transpose(buffer, buffer);
    return buffer;
};
buffer = dfft(rows, cols);
buffer = dfft(cols, rows);
spectrum = buffer;
}

```

Листинг 3 — исходный код метода inverseFFT

```

void FastFurierTransformer::inverseFFT(const Mat2f& spectrum, Mat1f& image)
{
    if (spectrum.empty() == true)
    {
        return;
    }

    auto cols = getOptimalDFTSize(spectrum.cols);
    auto rows = getOptimalDFTSize(spectrum.rows);

    auto buffer = spectrum.clone();

    auto ifft = [this, &buffer](int32_t rows, int32_t cols)
    {
        for (auto row = 0; row < rows; row++)
        {
            auto imPtr = buffer.ptr<Vec2f>(row);

            fft(imPtr, cols, true);
        }
        transpose(buffer, buffer);
        return buffer;
    };
    buffer = ifft(rows, cols);
    buffer = ifft(cols, rows);

    float max = FLT_MIN;
    auto shiftX = m_imageShift.x;
    auto shiftY = m_imageShift.y;
    for (auto row = 0; row < image.rows; row++)
    {
        auto imagePtr = image.ptr<float>(row);
        auto spectrumPtr = buffer.ptr<Vec2f>(row + shiftY);
        for (auto col = 0; col < image.cols; col++)
        {
            float value = spectrumPtr[col + shiftX][RE] / (cols * rows);
            imagePtr[col] = value;
            if (value > max)
            {
                max = value;
            }
        }
    }
    m_image /= max > 1 ? max : 1;
}

```

Рекурсивный алгоритм вычисления преобразования Фурье (как прямого, так и обратного) выполняется с помощью частного метода `fft`, реализация которого представлена в листинге 4.

Листинг 4 — Реализация метода `fft`

```
void FastFurierTransformer::fft(Vec2f *ptr, const int32_t size, const bool isInverse)
{
    if (size == 1) return;

    if (size % 2 == 0)
    {
        auto step = size / 2;
        auto vec = Mat2f(1, size, Vec2f(0, 0));
        auto ptr0 = vec.ptr<Vec2f>(0);
        auto ptr1 = ptr0 + step;
        for (int i = 0; i < step; i++)
        {
            ptr0[i] = ptr[2 * i];
            ptr1[i] = ptr[2 * i + 1];
        }

        fft(ptr0, step, isInverse);
        fft(ptr1, step, isInverse);

        float ang = 2 * M_PI / size * (isInverse ? 1 : -1);
        for (auto i = 0; i < step; i++)
        {
            DFT2(ptr, ptr0, ptr1, ang, i, 0, step);
            DFT2(ptr, ptr0, ptr1, ang, i, 1, step);
        }
        return;
    }

    if (size % 3 == 0)
    {
        auto step = size / 3;
        auto vec = Mat2f(1, size, Vec2f(0, 0));
        auto ptr0 = vec.ptr<Vec2f>(0);
        auto ptr1 = ptr0 + step;
        auto ptr2 = ptr1 + step;
        for (int i = 0; i < step; i++)
        {
            ptr0[i] = ptr[3 * i];
            ptr1[i] = ptr[3 * i + 1];
            ptr2[i] = ptr[3 * i + 2];
        }

        fft(ptr0, step, isInverse);
        fft(ptr1, step, isInverse);
        fft(ptr2, step, isInverse);

        float ang = 2 * M_PI / size * (isInverse ? 1 : -1);
        for (auto i = 0; i < step; i++)
        {
            DFT3(ptr, ptr0, ptr1, ptr2, ang, i, 0, step);
            DFT3(ptr, ptr0, ptr1, ptr2, ang, i, 1, step);
            DFT3(ptr, ptr0, ptr1, ptr2, ang, i, 2, step);
        }
        return;
    }
}
```



```

if (size % 5 == 0)
{
    auto step = size / 5;
    auto vec = Mat2f(1, size, Vec2f(0, 0));
    auto ptr0 = vec.ptr<Vec2f>(0);
    auto ptr1 = ptr0 + step;
    auto ptr2 = ptr1 + step;
    auto ptr3 = ptr2 + step;
    auto ptr4 = ptr3 + step;
    for (int i = 0; i < step; i++)
    {
        ptr0[i] = ptr[5 * i];
        ptr1[i] = ptr[5 * i + 1];
        ptr2[i] = ptr[5 * i + 2];
        ptr3[i] = ptr[5 * i + 3];
        ptr4[i] = ptr[5 * i + 4];
    }

    fft(ptr0, step, isInverse);
    fft(ptr1, step, isInverse);
    fft(ptr2, step, isInverse);
    fft(ptr3, step, isInverse);
    fft(ptr4, step, isInverse);

    float ang = 2 * M_PI / size * (isInverse ? 1 : -1);
    for (auto i = 0; i < step; i++)
    {
        DFT5(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, 0, step);
        DFT5(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, 1, step);
        DFT5(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, 2, step);
        DFT5(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, 3, step);
        DFT5(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, 4, step);
    }
    return;
}
}

```

Вычисление двух-, трёх- и пятиточечной схем преобразований Фурье, а также комплексная арифметика реализована с помощью макроопределений, реализация которых представлена в листинге 5.

Листинг 5 — Реализация макроопределений для схем преобразований Фурье и комплексной арифметики

```

#define COMPLEX_MUL(complex1, complex2) \
    (COMPLEX(complex1, RE) * COMPLEX(complex2, RE) - \
     COMPLEX(complex1, IM) * COMPLEX(complex2, IM)) : \
    (COMPLEX(complex1, RE) * COMPLEX(complex2, IM) + \
     COMPLEX(complex1, IM) * COMPLEX(complex2, RE))

#define COMPLEX_EXP(angle) cosf(angle) : sinf(angle)

#define COMPLEX_VEC(ptr, index) ptr[index][RE] : ptr[index][IM]

#define COMPLEX(complex, complexPart) (complexPart == RE ? complex)

static const float k2_Re[2][2] =
{
    {1.0, 1.0},
    {1.0, -1.0}
};

```

```

static const float k2_Im[2][2] =
{
    {0.0, 0.0},
    {0.0, 0.0}
};

static const float k3_Re[3][3] =
{
    {1.0, 1.0, 1.0},
    {1.0, -0.5, -0.5},
    {1.0, -0.5, -0.5}
};
static const float k3_Im[3][3] =
{
    {0.0, 0.0, 0.0},
    {0.0, sqrtf(3) / 2, -sqrtf(3) / 2},
    {0.0, -sqrtf(3) / 2, sqrtf(3) / 2}
};

static const float k5_Re[5][5] =
{
    {1.0, 1.0, 1.0, 1.0, 1.0},
    {1.0, cosf(2 * M_PI / 5), cosf(4 * M_PI / 5), cosf(6 * M_PI / 5), cosf(8 * M_PI / 5)},
    {1.0, cosf(4 * M_PI / 5), cosf(8 * M_PI / 5), cosf(2 * M_PI / 5), cosf(6 * M_PI / 5)},
    {1.0, cosf(6 * M_PI / 5), cosf(2 * M_PI / 5), cosf(8 * M_PI / 5), cosf(4 * M_PI / 5)},
    {1.0, cosf(8 * M_PI / 5), cosf(6 * M_PI / 5), cosf(4 * M_PI / 5), cosf(2 * M_PI / 5)}
};
static const float k5_Im[5][5] =
{
    {0.0, 0.0, 0.0, 0.0, 0.0},
    {0.0, sinf(2 * M_PI / 5), sinf(4 * M_PI / 5), sinf(6 * M_PI / 5), sinf(8 * M_PI / 5)},
    {0.0, sinf(4 * M_PI / 5), sinf(8 * M_PI / 5), sinf(2 * M_PI / 5), sinf(6 * M_PI / 5)},
    {0.0, sinf(6 * M_PI / 5), sinf(2 * M_PI / 5), sinf(8 * M_PI / 5), sinf(4 * M_PI / 5)},
    {0.0, sinf(8 * M_PI / 5), sinf(6 * M_PI / 5), sinf(4 * M_PI / 5), sinf(2 * M_PI / 5)}
};

#define K2(i, j, ang) k2_Re[i][j] : (ang > 0 ? 1 : -1) * k2_Im[i][j]
#define K3(i, j, ang) k3_Re[i][j] : (ang > 0 ? 1 : -1) * k3_Im[i][j]
#define K5(i, j, ang) k5_Re[i][j] : (ang > 0 ? 1 : -1) * k5_Im[i][j]

#define DFT2_RE(ptr, ptr0, ptr1, ang, i, j, step) ptr[i + j * step][RE] = \
    COMPLEX(COMPLEX_MUL(K2(0, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr0, i), COMPLEX_EXP(ang * i * 0))), RE) + \
    COMPLEX(COMPLEX_MUL(K2(1, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr1, i), COMPLEX_EXP(ang * i * 1))), RE)

#define DFT2_IM(ptr, ptr0, ptr1, ang, i, j, step) ptr[i + j * step][IM] = \
    COMPLEX(COMPLEX_MUL(K2(0, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr0, i), COMPLEX_EXP(ang * i * 0))), IM) + \
    COMPLEX(COMPLEX_MUL(K2(1, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr1, i), COMPLEX_EXP(ang * i * 1))), IM)

#define DFT2(ptr, ptr0, ptr1, ang, i, j, step) \
    DFT2_RE(ptr, ptr0, ptr1, ang, i, j, step); \
    DFT2_IM(ptr, ptr0, ptr1, ang, i, j, step)

#define DFT3_RE(ptr, ptr0, ptr1, ptr2, ang, i, j, step) ptr[i + j * step][RE] = \
    COMPLEX(COMPLEX_MUL(K3(0, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr0, i), COMPLEX_EXP(ang * i * 0))), RE) + \
    COMPLEX(COMPLEX_MUL(K3(1, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr1, i), COMPLEX_EXP(ang * i * 1))), RE) + \
    COMPLEX(COMPLEX_MUL(K3(2, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr2, i), COMPLEX_EXP(ang * i * 2))), RE)

```

```

#define DFT3_IM(ptr, ptr0, ptr1, ptr2, ang, i, j, step) ptr[i + j * step][IM] = \
    COMPLEX(COMPLEX_MUL(K3(0, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr0, i), COMPLEX_EXP(ang * i * 0))), IM) + \
    COMPLEX(COMPLEX_MUL(K3(1, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr1, i), COMPLEX_EXP(ang * i * 1))), IM) + \
    COMPLEX(COMPLEX_MUL(K3(2, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr2, i), COMPLEX_EXP(ang * i * 2))), IM)

#define DFT3(ptr, ptr0, ptr1, ptr2, ang, i, j, step) \
    DFT3_RE(ptr, ptr0, ptr1, ptr2, ang, i, j, step); \
    DFT3_IM(ptr, ptr0, ptr1, ptr2, ang, i, j, step)

#define DFT5_RE(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, j, step) ptr[i + j * step][RE] = \
    COMPLEX(COMPLEX_MUL(K5(0, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr0, i), COMPLEX_EXP(ang * i * 0))), RE) + \
    COMPLEX(COMPLEX_MUL(K5(1, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr1, i), COMPLEX_EXP(ang * i * 1))), RE) + \
    COMPLEX(COMPLEX_MUL(K5(2, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr2, i), COMPLEX_EXP(ang * i * 2))), RE) + \
    COMPLEX(COMPLEX_MUL(K5(3, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr3, i), COMPLEX_EXP(ang * i * 3))), RE) + \
    COMPLEX(COMPLEX_MUL(K5(4, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr4, i), COMPLEX_EXP(ang * i * 4))), RE)

#define DFT5_IM(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, j, step) ptr[i + j * step][IM] = \
    COMPLEX(COMPLEX_MUL(K5(0, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr0, i), COMPLEX_EXP(ang * i * 0))), IM) + \
    COMPLEX(COMPLEX_MUL(K5(1, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr1, i), COMPLEX_EXP(ang * i * 1))), IM) + \
    COMPLEX(COMPLEX_MUL(K5(2, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr2, i), COMPLEX_EXP(ang * i * 2))), IM) + \
    COMPLEX(COMPLEX_MUL(K5(3, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr3, i), COMPLEX_EXP(ang * i * 3))), IM) + \
    COMPLEX(COMPLEX_MUL(K5(4, j, ang), \
        COMPLEX_MUL(COMPLEX_VEC(ptr4, i), COMPLEX_EXP(ang * i * 4))), IM)

#define DFT5(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, j, step) \
    DFT5_RE(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, j, step); \
    DFT5_IM(ptr, ptr0, ptr1, ptr2, ptr3, ptr4, ang, i, j, step)

```

Выполнение произведения и корреляции Фурье-образов изображений производится с помощью функции `multiplySpectrums`, реализация которой представлена в листинге 6.

Листинг 6 — Реализация функции `multiplySpectrums`

```
void multiplySpectrums(Mat2f& first, Mat2f& second, Mat2f& result, bool isCorr)
{
    auto cols = max(first.cols, second.cols);
    auto rows = max(first.rows, second.rows);

    if (first.size != second.size)
    {
        return;
    }

    auto buffer = Mat2f(Size2i(cols, rows), Vec2f(0, 0));
    auto corr = isCorr ? -1 : 1;

    for (auto row = 0; row < rows; row++)
    {
        auto ptr = buffer.ptr<Vec2f>(row);
        auto ptr1 = first.ptr<Vec2f>(row);
        auto ptr2 = second.ptr<Vec2f>(row);

        for (auto col = 0; col < cols; col++)
        {
            ptr[col][RE] = COMPLEX(COMPLEX_MUL(COMPLEX_VEC(ptr1, col),
                                                COMPLEX_VEC(ptr2, col) * corr), RE);
            ptr[col][IM] = COMPLEX(COMPLEX_MUL(COMPLEX_VEC(ptr1, col),
                                                COMPLEX_VEC(ptr2, col) * corr), IM);
        }
    }

    result = buffer;
}
```

Класс Filter

Для выполнения фильтрации изображений с помощью фильтров верхних и нижних частот, а также с помощью заграждающих (режекторных) фильтров был разработан класс `Filter`, экземпляр которого является членом класса `FastFurrierTransformer`. Этот класс предоставляет методы для настройки параметров (вида и типа фильтра, частоты среза, ширины полосы подавления для режекторных фильтров, порядка для фильтра Баттерворта) и размера фильтра.

В листинге 7 представлен исходный код заголовочного файла класса `Filter`.

Листинг 7 — Исходный код файла `filter.h`

```
#pragma once

#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"

enum class FilterType
{
    LOW_PASS = 0,
    HIGH_PASS = 1,
    REJECTOR = 2
};

enum class FilterName
{
    IDEAL = 0,
    BUTTERWORTH = 1,
    GAUSS = 2
};

class Filter
{
public:
    Filter(const FilterType filterType = FilterType::LOW_PASS,
           const FilterName filterName = FilterName::BUTTERWORTH,
           const float distance = 50, const int32_t order = 5, const float width = 10);
    ~Filter() = default;

    void setSize(const cv::Size2i size);
    cv::Size2i getSize() const;

    void setFilter(const cv::Mat1f& filter);
    cv::Mat1f getFilter() const;
    void showFilter();

    void setFilterType(const FilterType filterType, const bool isUpdate = true);
    FilterType getFilterType() const;

    void setFilterName(const FilterName filterName, const bool isUpdate = true);
    FilterName getFilterName() const;

    void setDistance(const float distance, const bool isUpdate = true);
    float getDistance() const;
```

```

void setWidth(const float width, const bool isUpdate = true);
float getWidth() const;

void setOrder(const int32_t order, const bool isUpdate = true);
int32_t getOrder() const;

void configure(const FilterType filterType, const FilterName filterName,
               const float distance = -1, const int32_t order = -1,
               const float width = -1);

private:
    void calculateFilter();

    cv::Size2i m_size;
    cv::Mat1f m_filter;
    FilterType m_filterType;
    FilterName m_filterName;
    float m_distance;
    float m_width;
    int32_t m_order;
};

```

Вычисление фильтра заданного размера осуществляется с помощью частного метода `calculateFilter`, реализация которого представлена в листинге 8.

Листинг 8 — Реализация метода `calculateFilter`

```

void Filter::calculateFilter()
{
    if (m_size.height <= 0 || m_size.width <= 0)
    {
        return;
    }

    if (m_distance <= 0)
    {
        return;
    }

    auto filter = Mat1f(m_size, 0);
    auto center = Point2i(m_size.width / 2, m_size.height / 2);
    switch (m_filterType)
    {
    case FilterType::LOW_PASS:
        switch (m_filterName)
        {
        {
            case FilterName::IDEAL:
                for (auto row = 0; row < filter.rows; row++)
                {
                    auto ptr = filter.ptr<float>(row);
                    for (auto col = 0; col < filter.cols; col++)
                    {
                        ptr[col] = hypotf(col - center.x, row - center.y) > m_distance ? 0 : 1;
                    }
                }
                break;
            case FilterName::BUTTERWORTH:
                if (m_order <= 0)
                {
                    return;
                }

```

```

    for (auto row = 0; row < filter.rows; row++)
    {
        auto ptr = filter.ptr<float>(row);
        for (auto col = 0; col < filter.cols; col++)
        {
            auto distance = hypotf(col - center.x, row - center.y);
            ptr[col] = 1 / (1 + powf(distance / m_distance, 2 * m_order));
        }
    }
    break;
case FilterName::GAUSS:
    for (auto row = 0; row < filter.rows; row++)
    {
        auto ptr = filter.ptr<float>(row);
        for (auto col = 0; col < filter.cols; col++)
        {
            auto distance = hypotf(col - center.x, row - center.y);
            ptr[col] = expf(-powf(distance, 2) / (2 * powf(m_distance, 2)));
        }
    }
    break;
default:
    break;
}
break;
case FilterType::HIGH_PASS:
    switch (m_filterName)
    {
        case FilterName::IDEAL:
            for (auto row = 0; row < filter.rows; row++)
            {
                auto ptr = filter.ptr<float>(row);
                for (auto col = 0; col < filter.cols; col++)
                {
                    ptr[col] = hypotf(col - center.x, row - center.y) > m_distance ? 1 : 0;
                }
            }
            break;
        case FilterName::BUTTERWORTH:
            if (m_order <= 0)
            {
                return;
            }
            for (auto row = 0; row < filter.rows; row++)
            {
                auto ptr = filter.ptr<float>(row);
                for (auto col = 0; col < filter.cols; col++)
                {
                    auto distance = hypotf(col - center.x, row - center.y);
                    ptr[col] = 1 / (1 + powf(m_distance / distance, 2 * m_order));
                }
            }
            break;
        case FilterName::GAUSS:
            for (auto row = 0; row < filter.rows; row++)
            {
                auto ptr = filter.ptr<float>(row);
                for (auto col = 0; col < filter.cols; col++)
                {
                    auto distance = hypotf(col - center.x, row - center.y);
                    ptr[col] = 1 - expf(-powf(distance, 2) / (2 * powf(m_distance, 2)));
                }
            }
            break;
    }
}
break;

```

```

        default:
            break;
    }
    break;
case FilterType::REJECTOR:
    if (m_width <= 0)
    {
        return;
    }
    switch (m_filterName)
    {
    case FilterName::IDEAL:
        for (auto row = 0; row < filter.rows; row++)
        {
            auto ptr = filter.ptr<float>(row);
            for (auto col = 0; col < filter.cols; col++)
            {
                ptr[col] = (hypotf(col - center.x, row - center.y) <
                    m_distance - m_width / 2 ||
                    hypotf(col - center.x, row - center.y) >
                    m_distance + m_width / 2) ? 1 : 0;
            }
        }
        break;
    case FilterName::BUTTERWORTH:
        if (m_order <= 0)
        {
            return;
        }
        for (auto row = 0; row < filter.rows; row++)
        {
            auto ptr = filter.ptr<float>(row);
            for (auto col = 0; col < filter.cols; col++)
            {
                auto distance = hypotf(col - center.x, row - center.y);
                auto value = (distance * m_width) /
                    (powf(distance, 2) - powf(m_distance, 2));
                ptr[col] = 1 / (1 + powf(value, 2 * m_order));
            }
        }
        break;
    case FilterName::GAUSS:
        for (auto row = 0; row < filter.rows; row++)
        {
            auto ptr = filter.ptr<float>(row);
            for (auto col = 0; col < filter.cols; col++)
            {
                auto distance = hypotf(col - center.x, row - center.y);
                auto value = (powf(distance, 2) - powf(m_distance, 2)) /
                    (distance * m_width);
                ptr[col] = 1 - expf(-powf(value, 2));
            }
        }
        break;
    default:
        break;
    }
    break;
default:
    break;
}
m_filter = filter;
}

```


Выполнение работы

Выполнение заданий осуществлялось с помощью программы, код которой представлен в листинге 9.

Листинг 9 — Исходный код файла main.cpp

```
#include <iostream>

#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"

#include "fast_furrier_transformer.h"
#include "filter.h"

#define KERNEL_COLS 3
#define KERNEL_ROWS 3

using namespace std;
using namespace cv;

int main(int argc, char *argv[])
{
    /***** Задание 1. Прямое и обратное преобразования Фурье *****/

    Mat1f image_1 = imread("src/images/lena.bmp", IMREAD_GRAYSCALE);
    image_1 /= static_cast<float>(0xFF);

    // Реализованный класс FastFurrierTransformer
    auto fft_1 = FastFurrierTransformer();
    fft_1.setImage(image_1.clone());
    fft_1.directFastFurrierTransform();
    fft_1.inverseFastFurrierTransform();

    fft_1.showSpectrum();
    fft_1.showImage();

    // Встроенные функции
    auto cols = getOptimalDFTSize(image_1.cols);
    auto rows = getOptimalDFTSize(image_1.rows);

    auto openCVSpectrum_1 = Mat2f(Size2i(cols, rows), Vec2f(0, 0));
    auto openCVImage_1 = Mat1f(Size2i(cols, rows), 0);
    auto openCVImageROI_1 = Mat1f(image_1.clone());
    openCVImageROI_1.copyTo(openCVImage_1(Rect(0, 0, image_1.cols, image_1.rows)));

    dft(openCVImage_1, openCVSpectrum_1, DFT_COMPLEX_OUTPUT);
    idft(openCVSpectrum_1, openCVImage_1, DFT_REAL_OUTPUT);
    Mat1f openCVSpectrumComplex_1[NUM_OF_COMPLEX_PARTS];
    split(openCVSpectrum_1, openCVSpectrumComplex_1);
    auto openCVSpectrumMagnitude_1 = Mat1f();
    magnitude(openCVSpectrumComplex_1[RE], openCVSpectrumComplex_1[IM],
              openCVSpectrumMagnitude_1);
    shiftSpectrum(openCVSpectrumMagnitude_1, cols / 2, rows / 2);
    openCVSpectrumMagnitude_1 += Scalar::all(1);
    log(openCVSpectrumMagnitude_1, openCVSpectrumMagnitude_1);
    normalize(openCVSpectrumMagnitude_1, openCVSpectrumMagnitude_1, 0, 1, NORM_MINMAX);

    imshow("Spectrum [OpenCV]", openCVSpectrumMagnitude_1);
    imshow("Image [OpenCV]", openCVImageROI_1);

    while (waitKey() != 27);
}
```

```

// Сравнение спектров и изображений
auto diffSpectrum = Mat1f();
auto diffImage = Mat1f();

auto fftSpectrumMagnitude_1 = fft_1.getSpectrumMagnitude();
auto fftImage_1 = fft_1.getImage();

absdiff(fftSpectrumMagnitude_1, openCVSpectrumMagnitude_1, diffSpectrum);
imshow("Spectrum Difference", diffSpectrum);

absdiff(fftImage_1, openCVImageROI_1, diffImage);
imshow("Image Difference", diffImage);

while (waitKey() != 27);

/***** Конеч задание 1 *****/

/***** Задание 2. Фильтры верхних и нижних частот *****/

Mat1f image_2 = imread("src/images/lena.bmp", IMREAD_GRAYSCALE);
image_2 /= static_cast<float>(0xFF);

// 1) Фильтр верхних частот
auto fft_2_hpf = FastFurierTransformer();
fft_2_hpf.setImage(image_2.clone());

auto hpf = Filter();
hpf.configure(FilterType::HIGH_PASS, FilterName::BUTTERWORTH, 50, 5);

fft_2_hpf.setFilter(hpf);
fft_2_hpf.filtrateImage();

fft_2_hpf.showSpectrum("Spectrum with High Pass Butterworth Filter");
fft_2_hpf.showImage("Image with High Pass Butterworth Filter");

while (waitKey() != 27);

// 2) Фильтр нижних частот
auto fft_2_lpf = FastFurierTransformer();
fft_2_lpf.setImage(image_2.clone());

auto lpf = Filter();
lpf.configure(FilterType::LOW_PASS, FilterName::BUTTERWORTH, 50, 5);

fft_2_lpf.setFilter(lpf);
fft_2_lpf.filtrateImage();

fft_2_lpf.showSpectrum("Spectrum with Low Pass Butterworth Filter");
fft_2_lpf.showImage("Image with Low Pass Butterworth Filter");

while (waitKey() != 27);

/***** Конеч задание 2 *****/

/** Задание 3. Фурье-образы свёртки изображения с фильтрами */

Mat1f image_3 = imread("src/images/lena.bmp", IMREAD_GRAYSCALE);
image_3 /= static_cast<float>(0xFF);

cols = getOptimalDFTSize(image_3.cols + KERNEL_COLS - 1);
rows = getOptimalDFTSize(image_3.rows + KERNEL_ROWS - 1);

```

```

// 1) Фильтр Собеля

// а) Горизонтальный оператор Собеля
float sobelH[KERNEL_COLS][KERNEL_ROWS] =
{
    { 1.0,  2.0,  1.0},
    { 0.0,  0.0,  0.0},
    {-1.0, -2.0, -1.0}
};
auto sobelOperatorH = Mat1f(KERNEL_COLS, KERNEL_ROWS, *sobelH);

auto fft_sobelH = FastFurierTransformer();
fft_sobelH.setImage(image_3.clone());
fft_sobelH.setSpectrumSize(Size2i(cols, rows));
fft_sobelH.directFastFurierTransform();

auto fft_sobelFilterH = FastFurierTransformer();
fft_sobelFilterH.setImage(sobelOperatorH);
fft_sobelFilterH.setSpectrumSize(Size2i(cols, rows));
fft_sobelFilterH.directFastFurierTransform();

fft_sobelH.showSpectrum("Image Spectrum");
fft_sobelFilterH.showSpectrum("Horizontal Sobel Operator Spectrum");

auto sobelSpectrumH = fft_sobelH.getSpectrum();
auto sobelOperatorSpectrumH = fft_sobelFilterH.getSpectrum();
multiplySpectrums(sobelSpectrumH, sobelOperatorSpectrumH, sobelSpectrumH);

fft_sobelH.setSpectrum(sobelSpectrumH);
fft_sobelH.showSpectrum("Horizontal Sobel Operator Spectrum Result");

while (waitKey() != 27);

// 6) Вертикальный оператор Собеля
float sobelV[KERNEL_COLS][KERNEL_ROWS] =
{
    { 1.0,  0.0, -1.0},
    { 2.0,  0.0, -2.0},
    { 1.0,  0.0, -1.0}
};
auto sobelOperatorV = Mat1f(KERNEL_COLS, KERNEL_ROWS, *sobelV);

auto fft_sobelV = FastFurierTransformer();
fft_sobelV.setImage(image_3.clone());
fft_sobelV.setSpectrumSize(Size2i(cols, rows));
fft_sobelV.directFastFurierTransform();

auto fft_sobelFilterV = FastFurierTransformer();
fft_sobelFilterV.setImage(sobelOperatorV);
fft_sobelFilterV.setSpectrumSize(Size2i(cols, rows));
fft_sobelFilterV.directFastFurierTransform();

fft_sobelV.showSpectrum("Image Spectrum");
fft_sobelFilterV.showSpectrum("Vertical Sobel Operator Spectrum");

auto sobelSpectrumV = fft_sobelV.getSpectrum();
auto sobelOperatorSpectrumV = fft_sobelFilterV.getSpectrum();
multiplySpectrums(sobelSpectrumV, sobelOperatorSpectrumV, sobelSpectrumV);

fft_sobelV.setSpectrum(sobelSpectrumV);
fft_sobelV.showSpectrum("Vertical Sobel Operator Spectrum Result");

while (waitKey() != 27);

```

```

// в) Свёртка горизонтального и вертикального операторов Собеля
auto fft_sobel = FastFurierTransformer();
fft_sobel.setImage(image_3.clone());
fft_sobel.setSpectrumSize(Size2i(cols, rows));
fft_sobel.directFastFurierTransform();

auto sobelSpectrum = fft_sobel.getSpectrum();
auto sobelFilterSpectrum = Mat2f();
multiplySpectrums(sobelOperatorSpectrumH, sobelOperatorSpectrumV, sobelFilterSpectrum);

auto fft_sobelFilter = FastFurierTransformer();
fft_sobelFilter.setSpectrum(sobelFilterSpectrum);

fft_sobel.showSpectrum("Image Spectrum");
fft_sobelFilter.showSpectrum("Sobel Filter Spectrum");

multiplySpectrums(sobelSpectrum, sobelFilterSpectrum, sobelSpectrum);

fft_sobel.setSpectrum(sobelSpectrum);
fft_sobel.showSpectrum("Sobel Filter Spectrum Result");

while (waitKey() != 27);

// 2) Усредняющий фильтр (Box Filter)
float box[KERNEL_COLS][KERNEL_ROWS] =
{
    {1.0, 1.0, 1.0},
    {1.0, 1.0, 1.0},
    {1.0, 1.0, 1.0}
};

auto boxFilterKernel = Mat1f(KERNEL_COLS, KERNEL_ROWS, *box);

auto fft_box = FastFurierTransformer();
fft_box.setImage(image_3.clone());
fft_box.setSpectrumSize(Size2i(cols, rows));
fft_box.directFastFurierTransform();

auto fft_boxFilter = FastFurierTransformer();
fft_boxFilter.setImage(boxFilterKernel);
fft_boxFilter.setSpectrumSize(Size2i(cols, rows));
fft_boxFilter.directFastFurierTransform();

fft_box.showSpectrum("Image Spectrum");
fft_boxFilter.showSpectrum("Box Filter Spectrum");

auto boxSpectrum = fft_box.getSpectrum();
auto boxFilterSpectrum = fft_boxFilter.getSpectrum();
multiplySpectrums(boxSpectrum, boxFilterSpectrum, boxSpectrum);

fft_box.setSpectrum(boxSpectrum);
fft_box.showSpectrum("Box Filter Spectrum Result");

while (waitKey() != 27);

// 3) Фильтр Лапласа
float laplace[KERNEL_COLS][KERNEL_ROWS] =
{
    {0.0, 1.0, 0.0},
    {1.0, -4.0, 1.0},
    {0.0, 1.0, 0.0}
};

auto laplaceFilterKernel = Mat1f(KERNEL_COLS, KERNEL_ROWS, *laplace);

```

```

auto fft_laplace = FastFurierTransformer();
fft_laplace.setImage(image_3.clone());
fft_laplace.setSpectrumSize(Size2i(cols, rows));
fft_laplace.directFastFurierTransform();

auto fft_laplaceFilter = FastFurierTransformer();
fft_laplaceFilter.setImage(laplaceFilterKernel);
fft_laplaceFilter.setSpectrumSize(Size2i(cols, rows));
fft_laplaceFilter.directFastFurierTransform();

fft_laplace.showSpectrum("Image Spectrum");
fft_laplaceFilter.showSpectrum("Laplace Filter Spectrum");

auto laplaceSpectrum = fft_laplace.getSpectrum();
auto laplaceFilterSpectrum = fft_laplaceFilter.getSpectrum();
multiplySpectrums(laplaceSpectrum, laplaceFilterSpectrum, laplaceSpectrum);

fft_laplace.setSpectrum(laplaceSpectrum);
fft_laplace.showSpectrum("Laplace Filter Spectrum Result");

while (waitKey() != 27);

/**** Задание 4. Результаты свёртки изображения с фильтрами ****/

// 1) Фильтр Собеля

// а) Горизонтальный оператор Собеля
fft_sobelH.setImageShift(Point2i(KERNEL_COLS / 2, KERNEL_ROWS / 2));
fft_sobelH.inverseFastFurierTransform();
fft_sobelH.showSpectrum("Horizontal Sobel Operator Spectrum Result");
fft_sobelH.showImage("Horizontal Sobel Operator Image Result");

while (waitKey() != 27);

// Сравнение горизонтального оператора Собеля со встроенным фильтром
auto fftSobelHImage = fft_sobelH.getImage();
auto openCVSobelHImage = Mat1f();
Sobel(image_3, openCVSobelHImage, CV_32FC1, 0, 1, 3, 1, 0, BORDER_CONSTANT);
imshow("Horizontal Sobel Operator Image Result [OpenCV]", openCVSobelHImage);

auto diffSobelHFilter = Mat1f();
absdiff(fftSobelHImage, openCVSobelHImage, diffSobelHFilter);
imshow("Horizontal Sobel Operator Image Difference", diffSobelHFilter);

while (waitKey() != 27);

// 6) Вертикальный оператор Собеля
fft_sobelV.setImageShift(Point2i(KERNEL_COLS / 2, KERNEL_ROWS / 2));
fft_sobelV.inverseFastFurierTransform();
fft_sobelV.showSpectrum("Vertical Sobel Operator Spectrum Result");
fft_sobelV.showImage("Vertical Sobel Operator Image Result");

while (waitKey() != 27);

// Сравнение вертикального оператора Собеля со встроенным фильтром
auto fftSobelVImage = fft_sobelV.getImage();
auto openCVSobelVImage = Mat1f();
Sobel(image_3, openCVSobelVImage, CV_32FC1, 1, 0, 3, 1, 0, BORDER_CONSTANT);
imshow("Vertical Sobel Operator Image Result [OpenCV]", openCVSobelVImage);

auto diffSobelVFilter = Mat1f();
absdiff(fftSobelVImage, openCVSobelVImage, diffSobelVFilter);
imshow("Vertical Sobel Operator Image Difference", diffSobelVFilter);

while (waitKey() != 27);

```

```

// в) Свёртка горизонтального и вертикального операторов Собеля
fft_sobel.setImageShift(Point2i(2 * (KERNEL_COLS / 2), 2 * (KERNEL_ROWS / 2)));
fft_sobel.inverseFastFurierTransform();
fft_sobel.showSpectrum("Sobel Filter Spectrum Result");
fft_sobel.showImage("Sobel Filter Image Result");

while (waitKey() != 27);

// Сравнение свёртки операторов Собеля со встроенным фильтром
auto fftSobelImage = fft_sobel.getImage();
auto openCVSobelImage = Mat1f();
Sobel(image_3, openCVSobelImage, CV_32FC1, 1, 1, 3, 1, 0, BORDER_CONSTANT);
imshow("Sobel Filter Image Result [OpenCV]", openCVSobelImage);

auto diffSobelFilter = Mat1f();
absdiff(fftSobelImage, openCVSobelImage, diffSobelFilter);
imshow("Sobel Filter Image Difference", diffSobelFilter);

while (waitKey() != 27);

// 2) Усредняющий фильтр (Box Filter)
fft_box.setImageShift(Point2i(KERNEL_COLS / 2, KERNEL_ROWS / 2));
fft_box.inverseFastFurierTransform();
fft_box.showSpectrum("Box Filter Spectrum Result");
fft_box.showImage("Box Filter Image Result");

while (waitKey() != 27);

// Сравнение усредняющего фильтра со встроенным фильтром
auto fftBoxImage = fft_box.getImage();
auto openCVBoxImage = Mat1f();
boxFilter(image_3, openCVBoxImage, CV_32FC1, Size(3, 3),
          Point(-1, -1), true, BORDER_CONSTANT);
imshow("Box Filter Image Result [OpenCV]", openCVBoxImage);

auto diffBoxFilter = Mat1f();
absdiff(fftBoxImage, openCVBoxImage, diffBoxFilter);
imshow("Box Filter Image Difference", diffBoxFilter);

while (waitKey() != 27);

// 3) Фильтр Лапласа
fft_laplace.setImageShift(Point2i(KERNEL_COLS / 2, KERNEL_ROWS / 2));
fft_laplace.inverseFastFurierTransform();
fft_laplace.showSpectrum("Laplace Filter Spectrum Result");
fft_laplace.showImage("Laplace Filter Image Result");

while (waitKey() != 27);

// Сравнение фильтра Лапласа со встроенным фильтром
auto fftLaplaceImage = fft_laplace.getImage();
auto openCVLaplaceImage = Mat1f();
Laplacian(image_3, openCVLaplaceImage, CV_32FC1, 1, 1, 0, BORDER_CONSTANT);
imshow("Laplace Filter Image Result [OpenCV]", openCVLaplaceImage);

auto diffLaplaceFilter = Mat1f();
absdiff(fftLaplaceImage, openCVLaplaceImage, diffLaplaceFilter);
imshow("Laplace Filter Difference", diffLaplaceFilter);

while (waitKey() != 27);

/***** Конеч задания 4 *****/

```

```

/***** Задание 5. Корреляция изображений *****/

Mat1f image_5 = imread("src/images/car_number.bmp", IMREAD_GRAYSCALE);
image_5 /= static_cast<float>(0xFF);

// Корреляция с символом '6'
Mat1f symbol_1 = imread("src/images/number_six.bmp", IMREAD_GRAYSCALE);
symbol_1 /= static_cast<float>(0xFF);

cols = getOptimalDFTSize(image_5.cols + symbol_1.cols - 1);
rows = getOptimalDFTSize(image_5.rows + symbol_1.rows - 1);

auto fft_carNumber_1 = FastFurierTransformer();
fft_carNumber_1.setImage(image_5);
fft_carNumber_1.setSpectrumSize(Size2i(cols, rows));
fft_carNumber_1.directFastFurierTransform();

auto fft_symbol_1 = FastFurierTransformer();
fft_symbol_1.setImage(symbol_1);
fft_symbol_1.setSpectrumSize(Size2i(cols, rows));
fft_symbol_1.directFastFurierTransform();

fft_carNumber_1.showImage("Car Number");
fft_symbol_1.showImage("Symbol '6' Image");
fft_symbol_1.showSpectrum("Symbol '6' Spectrum");

while (waitKey() != 27);

auto carNumberSpectrum_1 = fft_carNumber_1.getSpectrum();
auto symbolSpectrum_1 = fft_symbol_1.getSpectrum();
multiplySpectrums(carNumberSpectrum_1, symbolSpectrum_1, carNumberSpectrum_1, true);

fft_carNumber_1.setSpectrum(carNumberSpectrum_1);
fft_carNumber_1.inverseFastFurierTransform();

auto carNumberImage_1 = fft_carNumber_1.getImage();
threshold(carNumberImage_1, carNumberImage_1, 0.95, 1.0, THRESH_BINARY);
fft_carNumber_1.setImage(carNumberImage_1);

fft_carNumber_1.showSpectrum("Correlation with Symbol '6' Spectrum");
fft_carNumber_1.showImage("Correlation with Symbol '6' Image");

while (waitKey() != 27);

// Корреляция с символом '9'
Mat1f symbol_2 = imread("src/images/number_nine.bmp", IMREAD_GRAYSCALE);
symbol_2 /= static_cast<float>(0xFF);

cols = getOptimalDFTSize(image_5.cols + symbol_2.cols - 1);
rows = getOptimalDFTSize(image_5.rows + symbol_2.rows - 1);

auto fft_carNumber_2 = FastFurierTransformer();
fft_carNumber_2.setImage(image_5);
fft_carNumber_2.setSpectrumSize(Size2i(cols, rows));
fft_carNumber_2.directFastFurierTransform();

auto fft_symbol_2 = FastFurierTransformer();
fft_symbol_2.setImage(symbol_2);
fft_symbol_2.setSpectrumSize(Size2i(cols, rows));
fft_symbol_2.directFastFurierTransform();

fft_carNumber_2.showImage("Car Number");
fft_symbol_2.showImage("Symbol '9' Image");
fft_symbol_2.showSpectrum("Symbol '9' Spectrum");

while (waitKey() != 27);

```

```

auto carNumberSpectrum_2 = fft_carNumber_2.getSpectrum();
auto symbolSpectrum_2 = fft_symbol_2.getSpectrum();
multiplySpectrums(carNumberSpectrum_2, symbolSpectrum_2, carNumberSpectrum_2, true);

fft_carNumber_2.setSpectrum(carNumberSpectrum_2);
fft_carNumber_2.inverseFastFurierTransform();

auto carNumberImage_2 = fft_carNumber_2.getImage();
threshold(carNumberImage_2, carNumberImage_2, 0.95, 1.0, THRESH_BINARY);
fft_carNumber_2.setImage(carNumberImage_2);

fft_carNumber_2.showSpectrum("Correlation with Symbol '9' Spectrum");
fft_carNumber_2.showImage("Correlation with Symbol '9' Image");

while (waitKey() != 27);

// Корреляция с символом '0'
Mat1f symbol_3 = imread("src/images/letter_o.bmp", IMREAD_GRAYSCALE);
symbol_3 /= static_cast<float>(0xFF);

cols = getOptimalDFTSize(image_5.cols + symbol_3.cols - 1);
rows = getOptimalDFTSize(image_5.rows + symbol_3.rows - 1);

auto fft_carNumber_3 = FastFurierTransformer();
fft_carNumber_3.setImage(image_5);
fft_carNumber_3.setSpectrumSize(Size2i(cols, rows));
fft_carNumber_3.directFastFurierTransform();

auto fft_symbol_3 = FastFurierTransformer();
fft_symbol_3.setImage(symbol_3);
fft_symbol_3.setSpectrumSize(Size2i(cols, rows));
fft_symbol_3.directFastFurierTransform();

fft_carNumber_3.showImage("Car Number");
fft_symbol_3.showImage("Symbol '0' Image");
fft_symbol_3.showSpectrum("Symbol '0' Spectrum");

while (waitKey() != 27);

auto carNumberSpectrum_3 = fft_carNumber_3.getSpectrum();
auto symbolSpectrum_3 = fft_symbol_3.getSpectrum();
multiplySpectrums(carNumberSpectrum_3, symbolSpectrum_3, carNumberSpectrum_3, true);

fft_carNumber_3.setSpectrum(carNumberSpectrum_3);
fft_carNumber_3.inverseFastFurierTransform();

auto carNumberImage_3 = fft_carNumber_3.getImage();
threshold(carNumberImage_3, carNumberImage_3, 0.95, 1.0, THRESH_BINARY);
fft_carNumber_3.setImage(carNumberImage_3);

fft_carNumber_3.showSpectrum("Correlation with Symbol '0' Spectrum");
fft_carNumber_3.showImage("Correlation with Symbol '0' Image");

while (waitKey() != 27);

// Корреляция с символом 'E'
Mat1f symbol_4 = imread("src/images/letter_e.bmp", IMREAD_GRAYSCALE);
symbol_4 /= static_cast<float>(0xFF);

cols = getOptimalDFTSize(image_5.cols + symbol_4.cols - 1);
rows = getOptimalDFTSize(image_5.rows + symbol_4.rows - 1);

auto fft_carNumber_4 = FastFurierTransformer();
fft_carNumber_4.setImage(image_5);
fft_carNumber_4.setSpectrumSize(Size2i(cols, rows));
fft_carNumber_4.directFastFurierTransform();

```



```

auto fft_symbol_4 = FastFurierTransformer();
fft_symbol_4.setImage(symbol_4);
fft_symbol_4.setSpectrumSize(Size2i(cols, rows));
fft_symbol_4.directFastFurierTransform();

fft_carNumber_4.showImage("Car Number");
fft_symbol_4.showImage("Symbol 'E' Image");
fft_symbol_4.showSpectrum("Symbol 'E' Spectrum");

while (waitKey() != 27);

auto carNumberSpectrum_4 = fft_carNumber_4.getSpectrum();
auto symbolSpectrum_4 = fft_symbol_4.getSpectrum();
multiplySpectrums(carNumberSpectrum_4, symbolSpectrum_4, carNumberSpectrum_4, true);

fft_carNumber_4.setSpectrum(carNumberSpectrum_4);
fft_carNumber_4.inverseFastFurierTransform();

auto carNumberImage_4 = fft_carNumber_4.getImage();
threshold(carNumberImage_4, carNumberImage_4, 0.95, 1.0, THRESH_BINARY);
fft_carNumber_4.setImage(carNumberImage_4);

fft_carNumber_4.showSpectrum("Correlation with Symbol 'E' Spectrum");
fft_carNumber_4.showImage("Correlation with Symbol 'E' Image");

while (waitKey() != 27);

/***** Конеч задание 5 *****/

return 0;
}

```

Задание 1. Прямое и обратное преобразования Фурье

Прямое и обратное дискретные преобразования Фурье применялись к изображению `lena.bmp`, которое представлено на рисунке 1.



Рисунок 1 — Изображение `lena.bmp`

Результаты применения прямого и обратного преобразований Фурье с помощью реализованного класса и с помощью встроенных функций представлены на рисунках 2 и 3 соответственно.

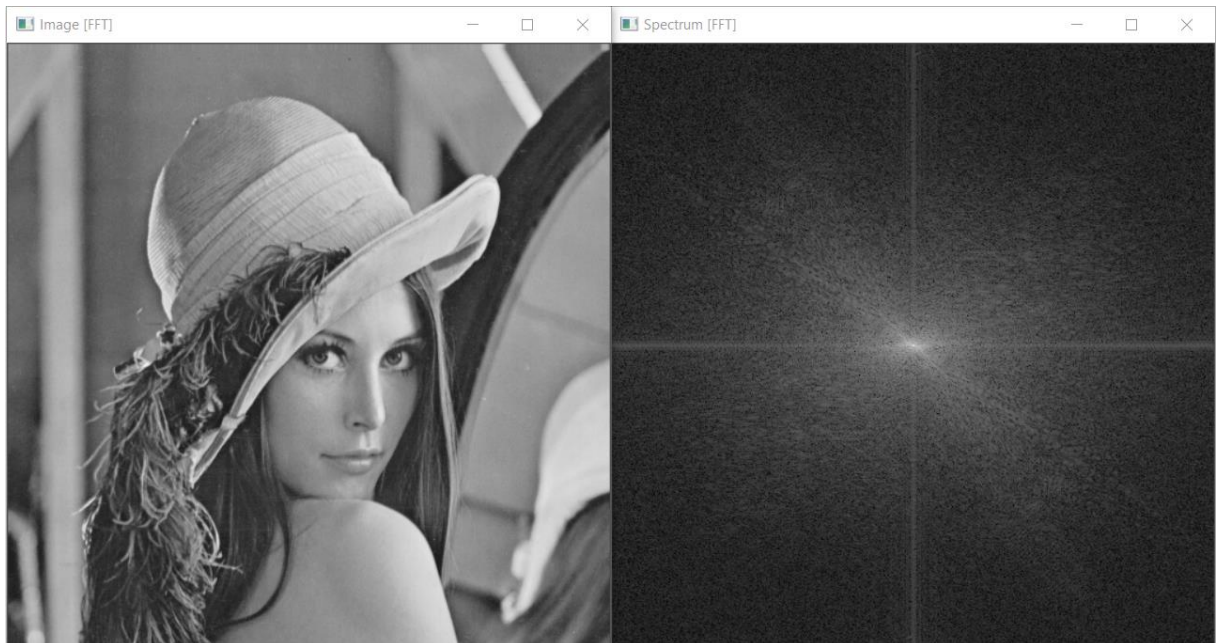


Рисунок 2 — Изображение (слева) и его спектр (справа), полученные путём преобразований Фурье с помощью методов класса `FastFurierTransformer`

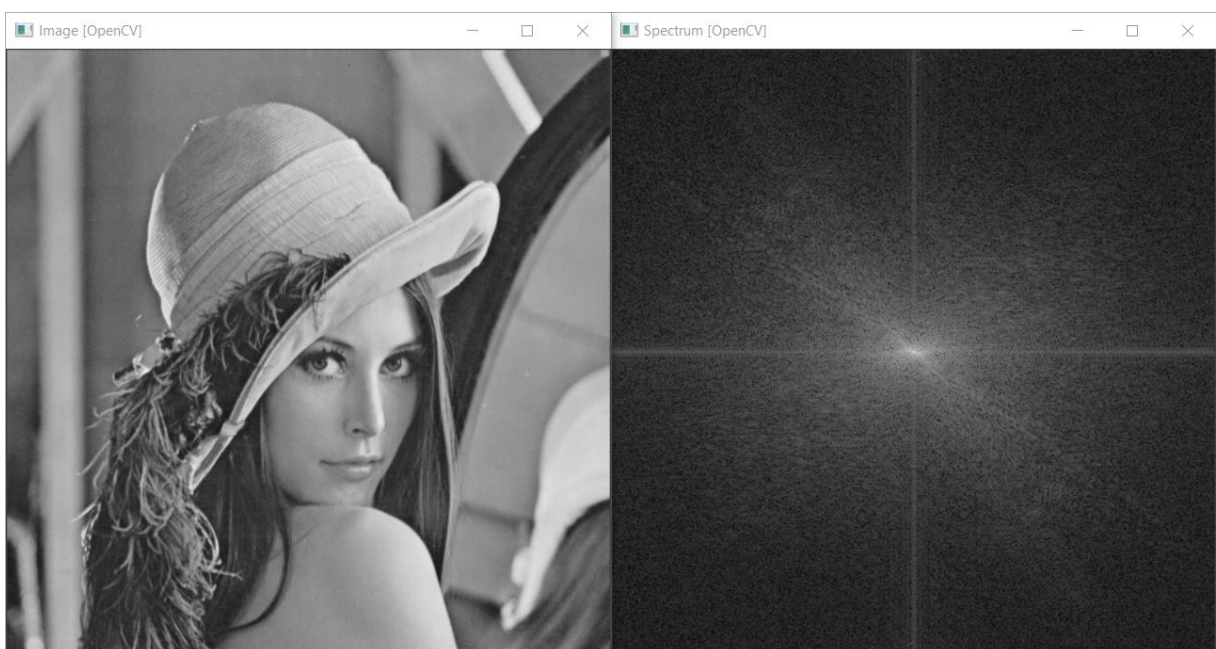


Рисунок 3 — Изображение (слева) и его спектр (справа), полученные путём преобразований Фурье с помощью встроенных функций `dft` и `idft`

Время работы методов разработанного класса составило величину порядка 4 секунд в режиме отладки и 400 миллисекунд в релизной версии, в то время как встроенные функции выполняют преобразование за время, не превышающее величину в 3 миллисекунды.

Также сравним результаты работы реализованного класса и библиотечных функций, рассчитав абсолютную разницу между изображениями и их спектрами, полученными разными методами, которая представлена на рисунке 4.

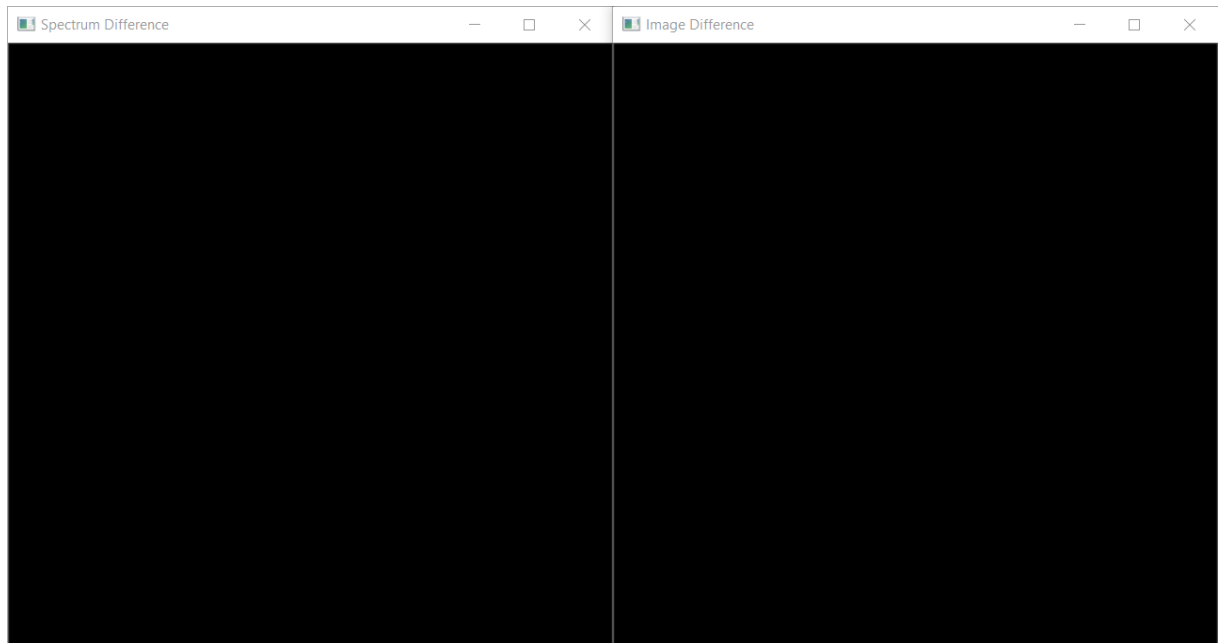


Рисунок 4 — Абсолютная разница изображений (слева) и их спектров (справа)

Задание 2. Фильтры верхних и нижних частот

В работе применялись фильтры верхних и нижних частот Баттерворта порядка $n = 5$ с частотой среза $D_0 = 50$. Фильтрации подвергалось то же самое изображение, что и в предыдущем пункте. Результаты работы фильтров верхних и нижних частот представлены на рисунках 5 и 6 соответственно.

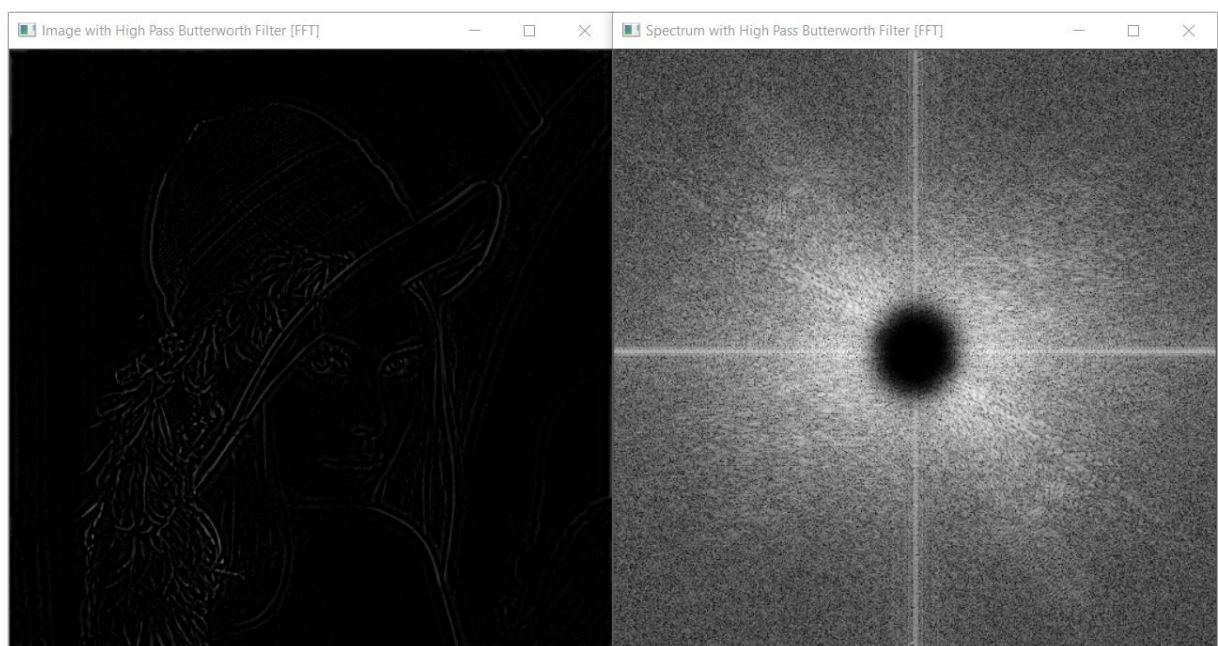


Рисунок 5 — Изображение (слева) и его спектр (справа), подвергнутые фильтрации с помощью фильтра верхних частот Баттерворта

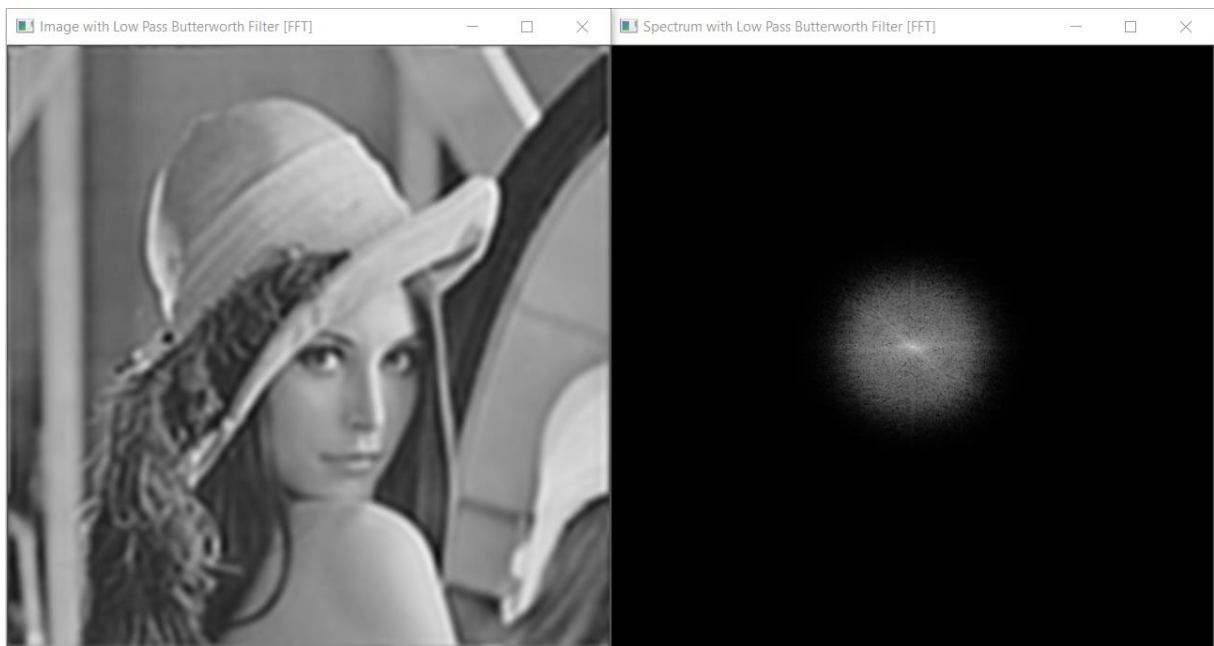


Рисунок 6 — Изображение (слева) и его спектр (справа), подвергнутые фильтрации с помощью фильтра нижних частот Баттерворта

Задание 3. Фурье-образы свёртки изображения с фильтрами

Вычислим свёртку изображения, которое использовалось в предыдущем пункте, с фильтром Собеля, усредняющим фильтром (Box Filter) и фильтром Лапласа. Для этого сначала применим прямое преобразование Фурье к ядрам этих фильтров, а затем произведём поэлементное умножение Фурье-образов каждого из фильтров с Фурье-образом исходного изображения.

Результаты произведения спектра изображения со спектрами горизонтального и вертикального операторов Собеля, а также свёртки этих операторов представлены на рисунках 7, 10 и 11 соответственно.

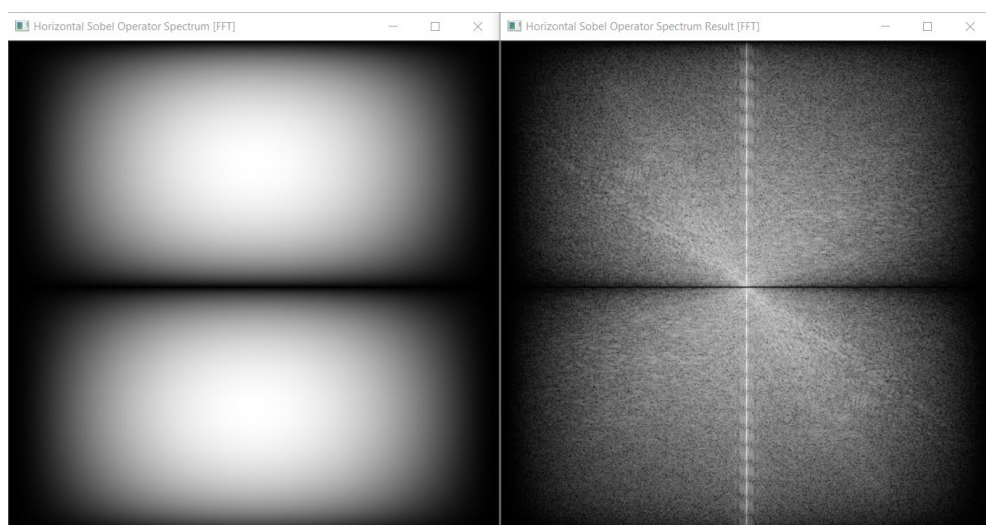


Рисунок 7 — Спектры ядра горизонтального оператора Собеля (слева) и его свёртки с исходным изображением (справа)

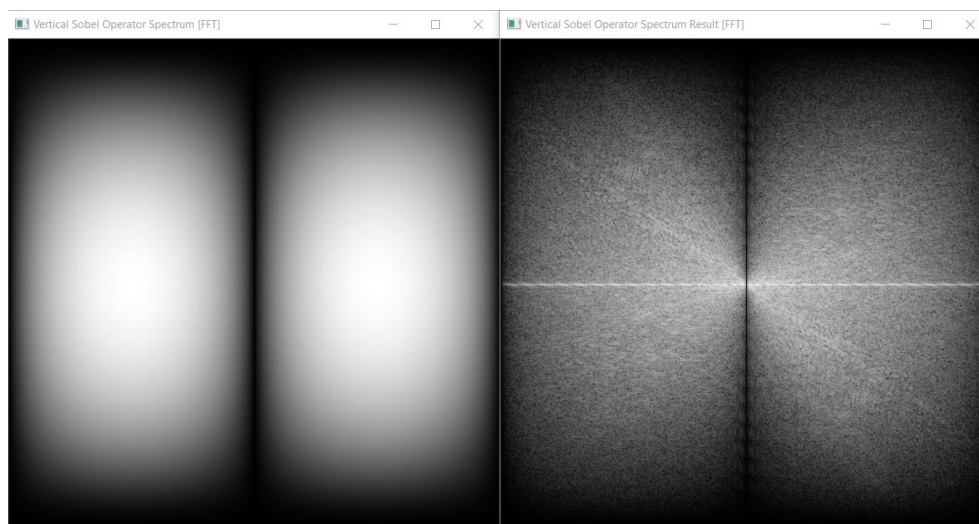


Рисунок 8 — Спектры ядра горизонтального оператора Собеля (слева) и его свёртки с исходным изображением (справа)

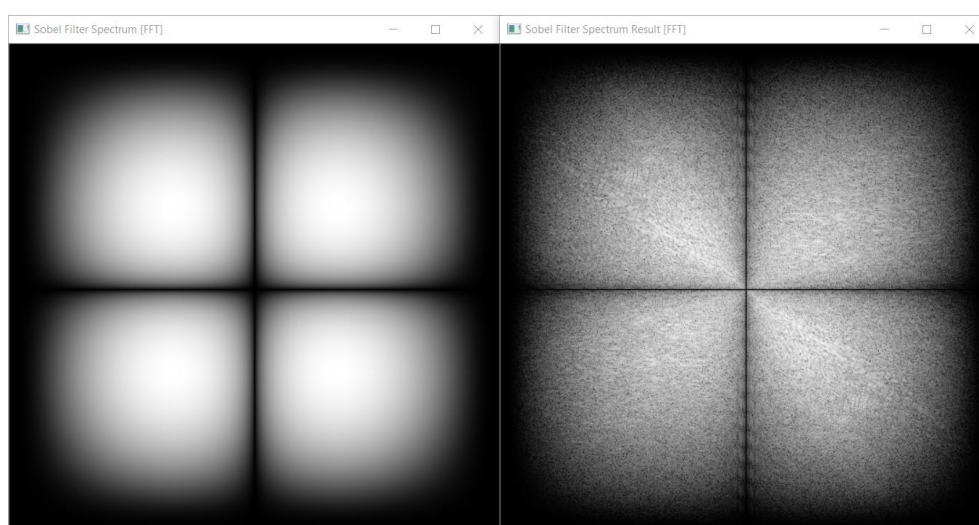


Рисунок 9 — Спектры ядра свёртки горизонтального и вертикального операторов Собеля (слева) и его свёртки с исходным изображением (справа)

Результат произведения спектра изображения со спектром ядра усредняющего фильтра представлен на рисунке 10.

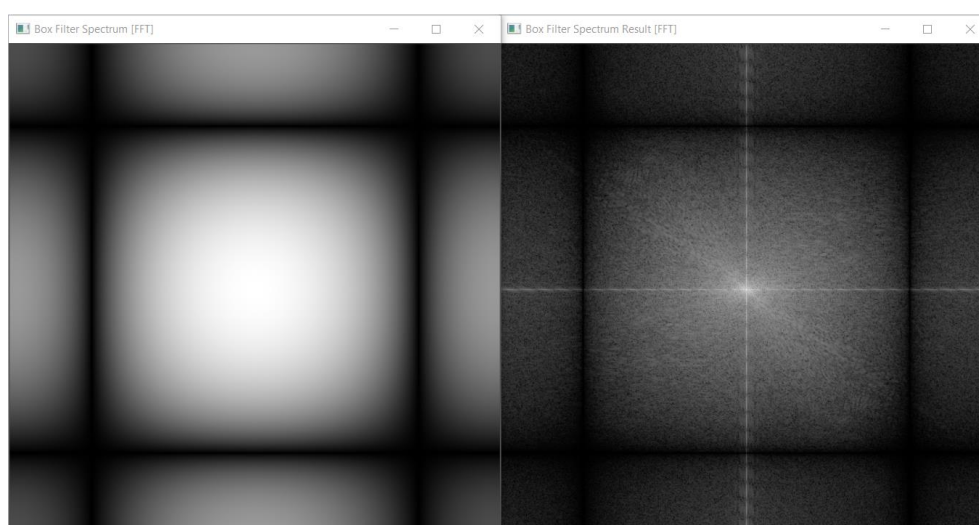


Рисунок 10 — Спектры ядра усредняющего фильтра (слева) и его свёртки с исходным изображением (справа)

Результат произведения спектра изображения со спектром ядра фильтра Лапласа представлен на рисунке 11.

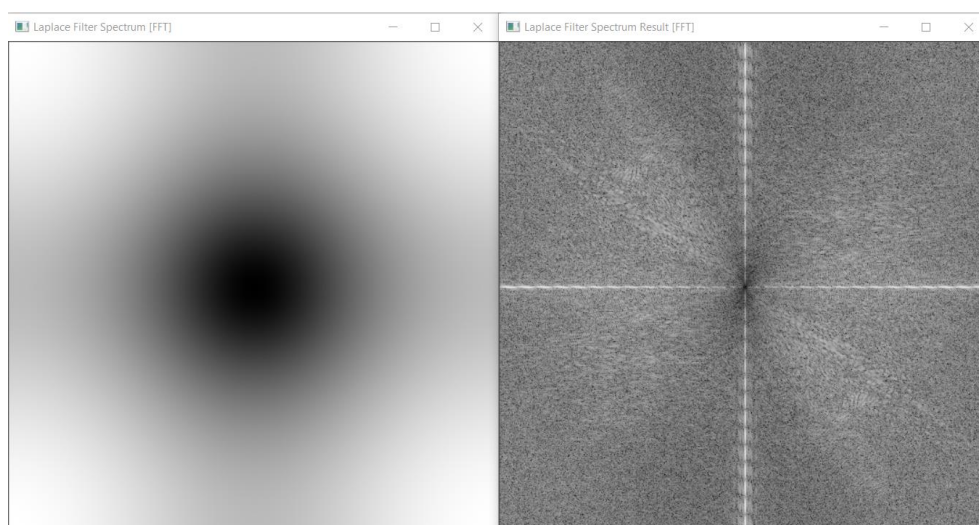


Рисунок 11 — Спектры ядра фильтра Лапласа (слева) и его свёртки с исходным изображением (справа)

Задание 4. Результаты свёртки изображения с фильтрами

Применим к полученным Фурье-образам свёртки изображения с фильтрами обратное преобразование Фурье и получим результат свёртки.

Результаты применения свёртки изображения с горизонтальным и вертикальным операторами Собеля, а также со свёрткой этих операторов представлены на рисунках 12, 15 и 16 соответственно.

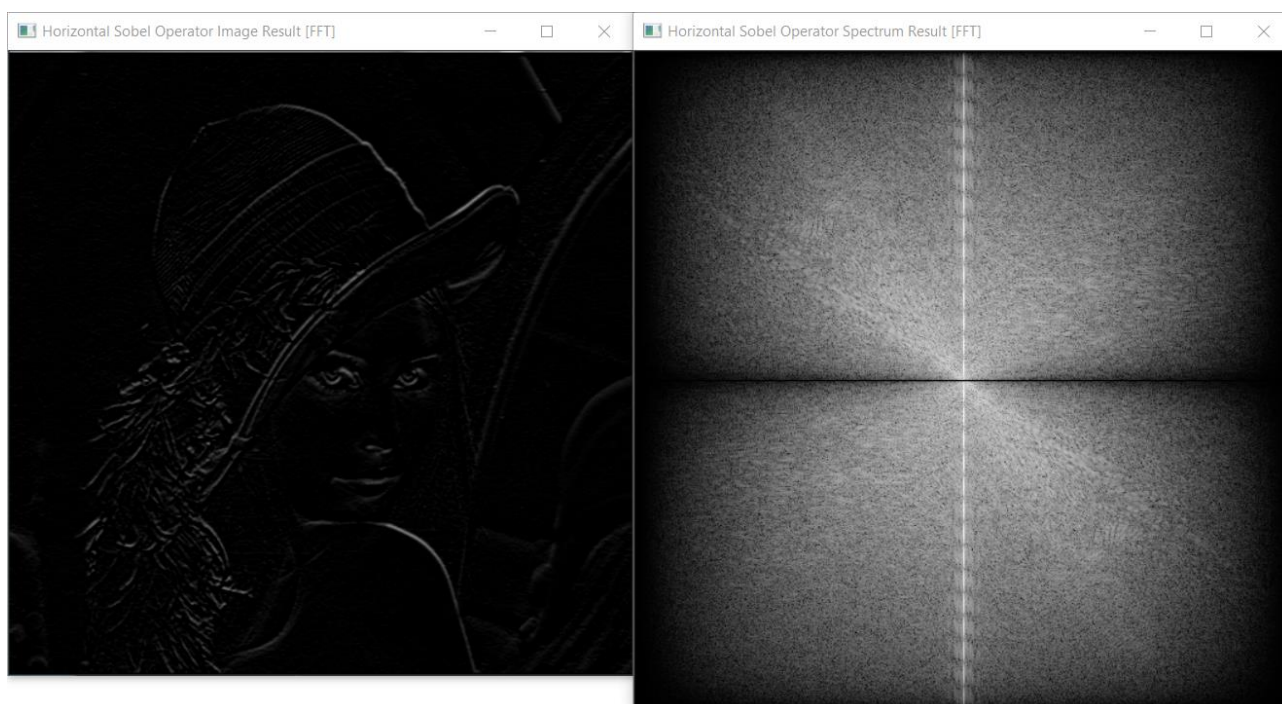


Рисунок 12 — Результат свёртки изображения с горизонтальным оператором Собеля (слева) и её (свёртки) спектр (справа)

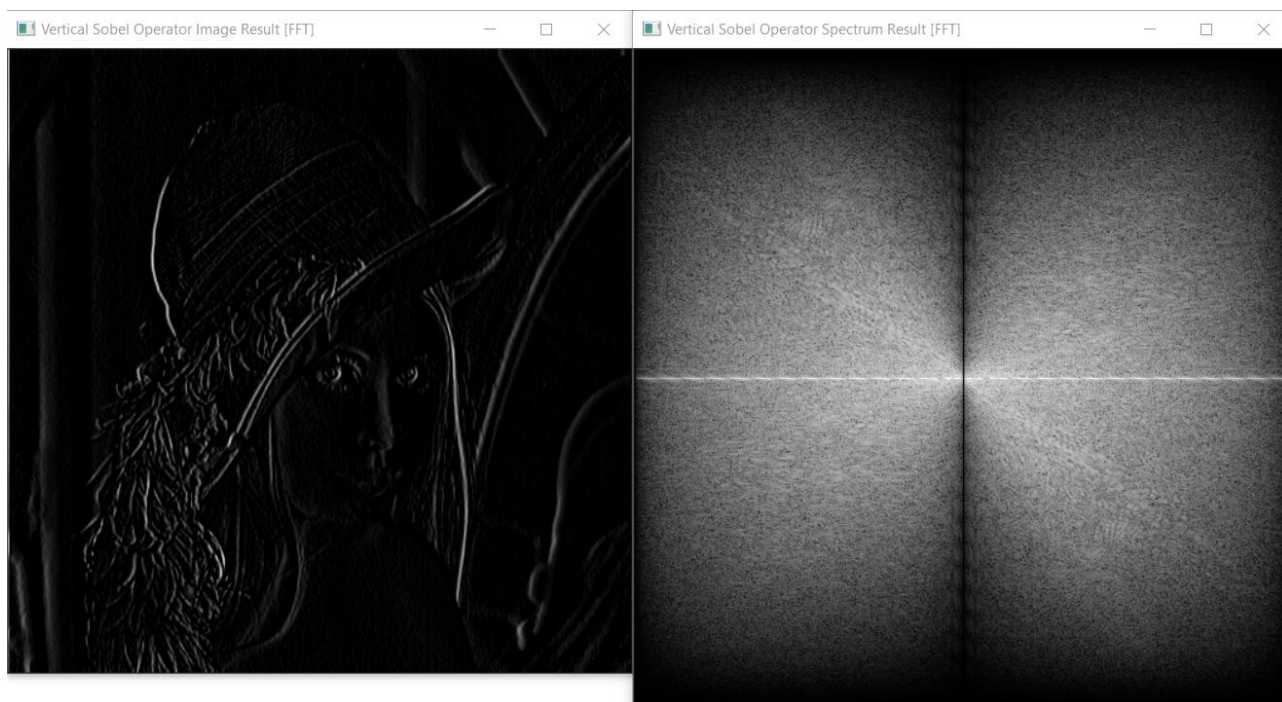


Рисунок 13 — Результат свёртки изображения с вертикальным оператором Собеля (слева) и её (свёртки) спектр (справа)

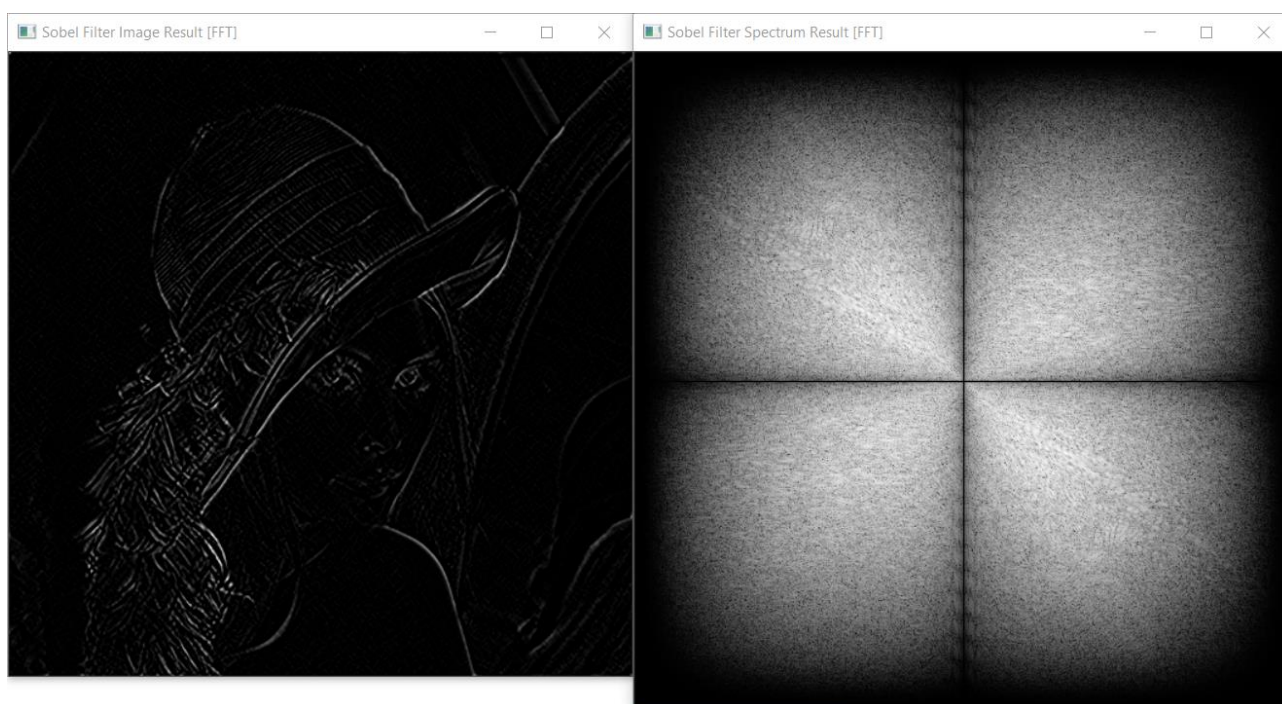


Рисунок 14 — Результат свёртки изображения со свёрткой горизонтального и вертикального операторов Собеля (слева) и её (свёртки) спектр (справа)

Результаты применения свёртки изображения с усредняющим фильтром представлены на рисунке 15.

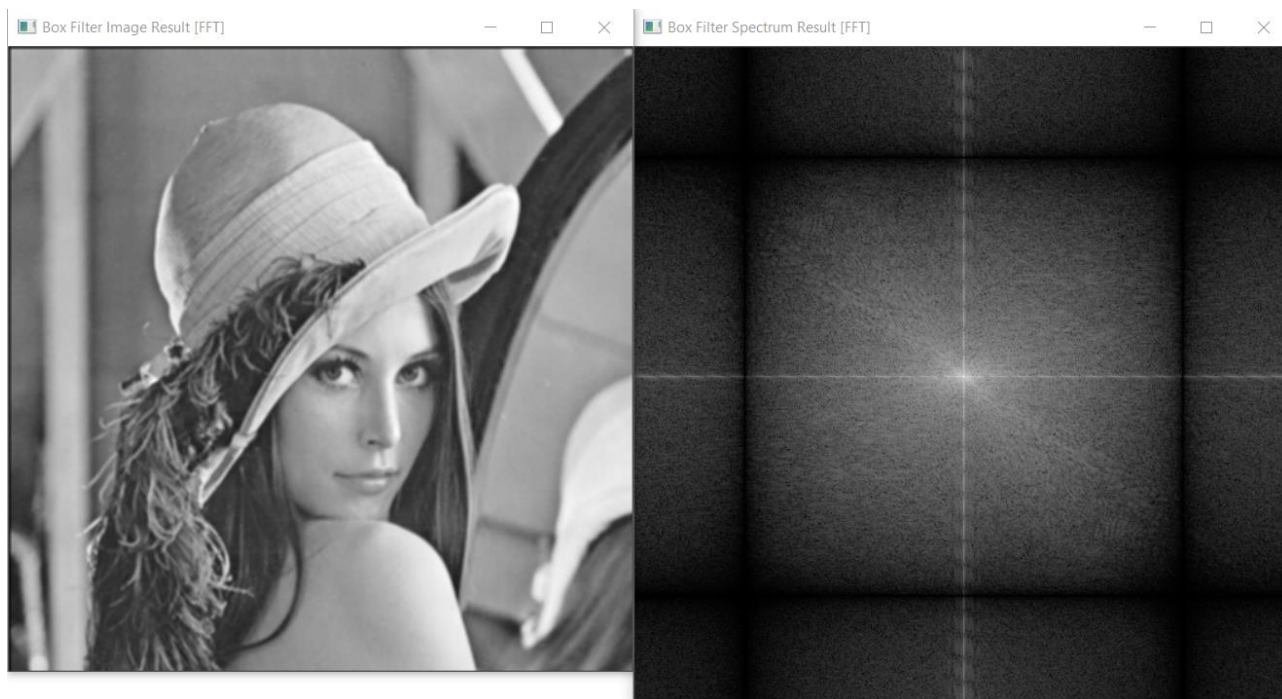


Рисунок 15 — Результат свёртки изображения с усредняющим фильтром (слева) и её (свёртки) спектр (справа)

Результаты применения свёртки изображения с фильтром Лапласа представлены на рисунке 16.

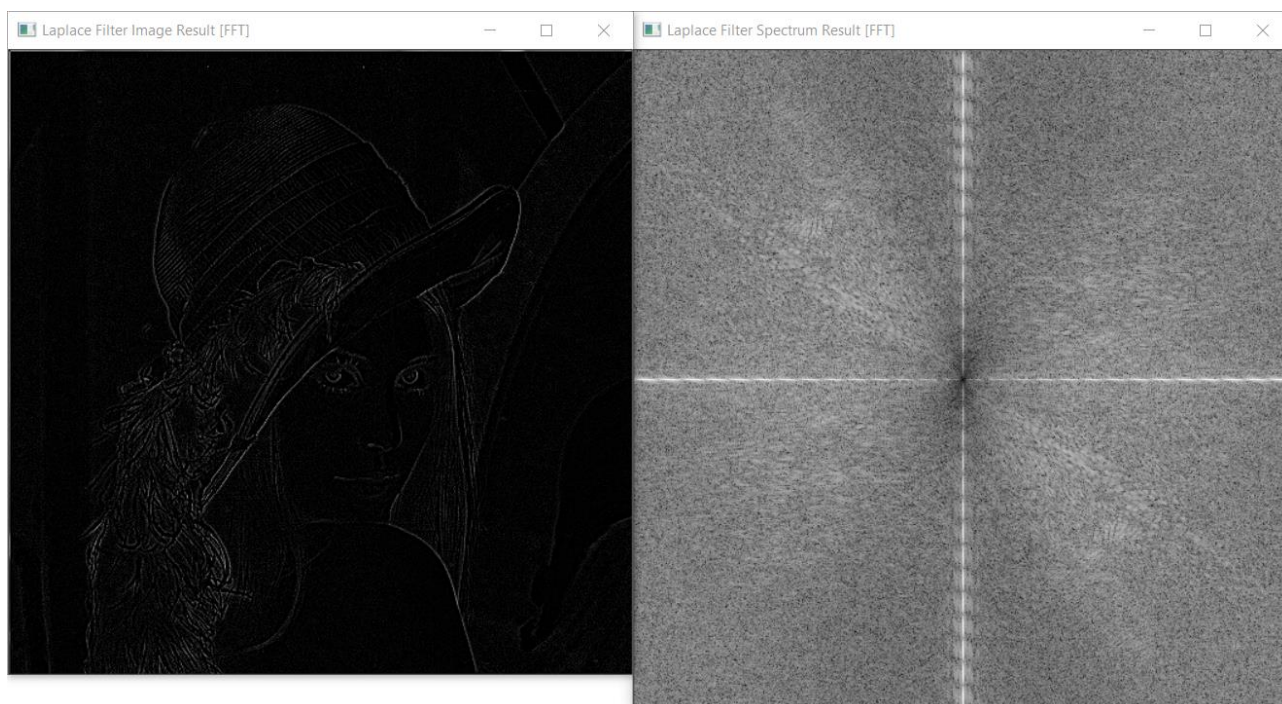


Рисунок 16 — Результат свёртки изображения с фильтром Лапласа (слева) и её (свёртки) спектр (справа)

Выполним свёртку исходного изображения с ядрами фильтров с помощью библиотечных функций и рассчитаем их (свёрток) абсолютную разницу с ранее полученными свёртками. Результаты сравнения представлены на рисунке 17.

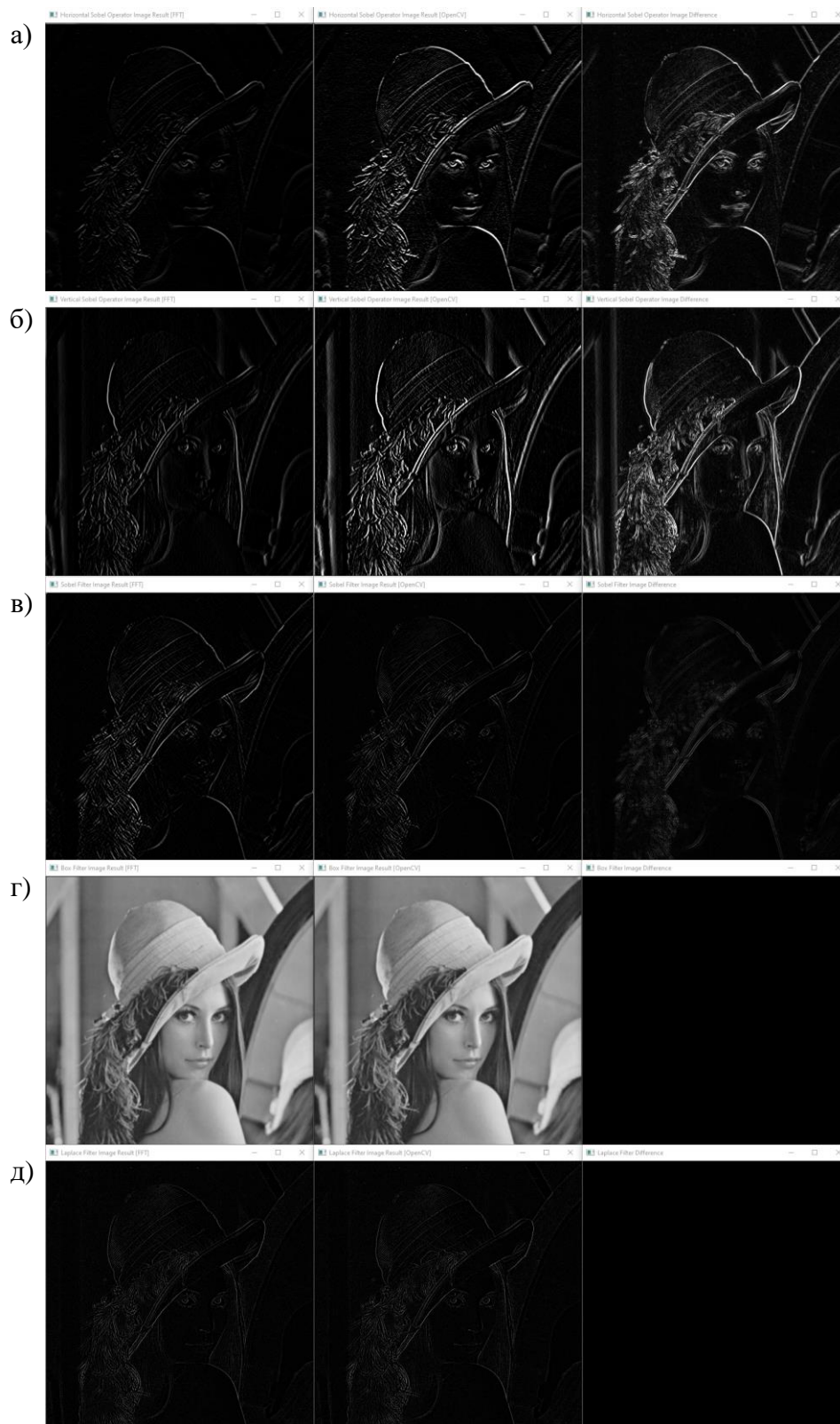


Рисунок 17 — Сравнение результатов применения фильтров с помощью произведения Фурье-образов (слева), библиотечных функций (в центре) и их абсолютная разность (справа):
а — горизонтальный оператор Собеля, б — вертикальный оператор Собеля,
в — свёртка операторов Собеля, г — усредняющий фильтр, д — фильтр Лапласа

Задание 5. Корреляция изображений

Произведём корреляцию изображения автомобильных знаков, представленного на рисунке 18, с символами '6', '9', 'О' и 'Е' для обнаружения данных символов на изображении. Для этого найдём Фурье-образы изображения и символов, затем произведём поэлементное умножение Фурье-образа исходного изображения на комплексно-сопряжённые Фурье-образы полученных изображений. К полученным Фурье-образу применим сначала пороговую фильтрацию по уровню 0,95 от максимального значения, а после этого выполним обратное преобразование Фурье. Результаты проведения корреляции исходного изображения с символами '6', '9', 'О' и 'Е' представлены на рисунках 19 — 22.



Рисунок 18 — Изображение автомобильных номеров car_number.bmp

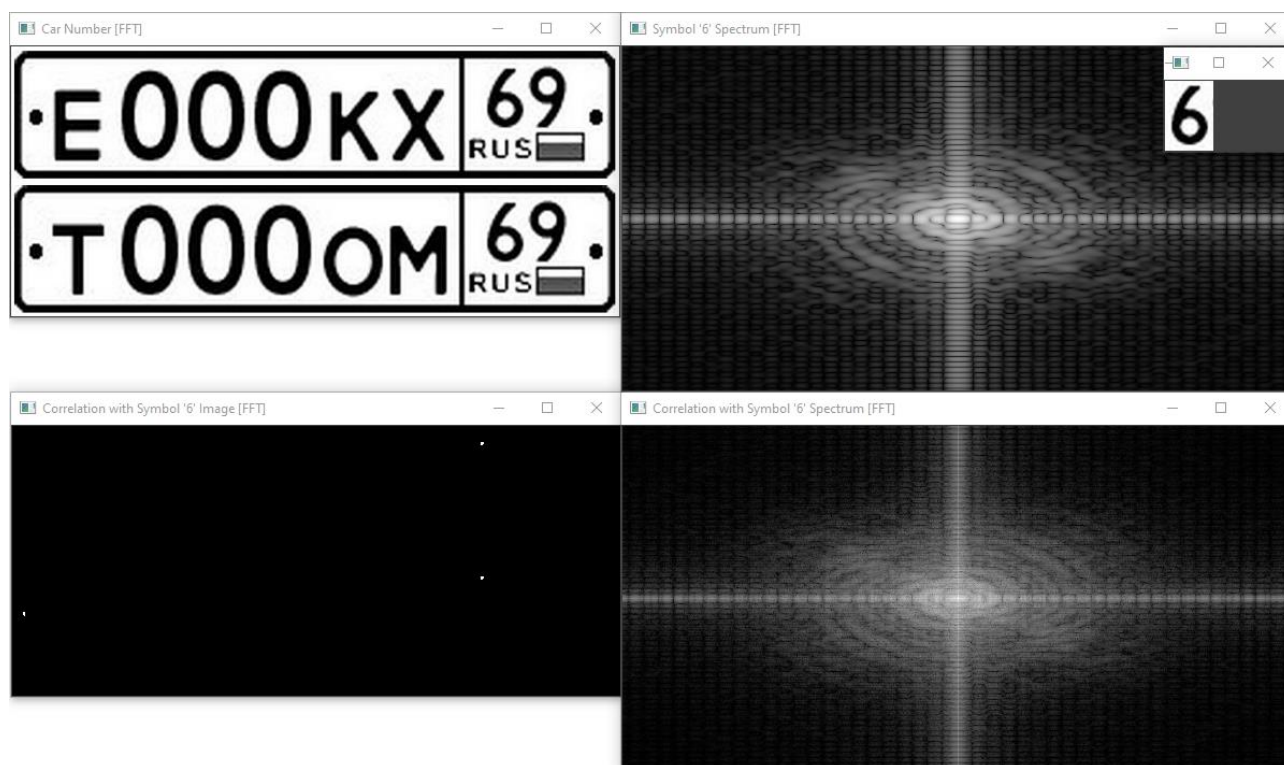


Рисунок 19 — Исходное изображение (слева сверху), символ '6' и его спектр (справа сверху), результат корреляции изображения с этим символом (слева снизу) и его спектр (справа внизу)

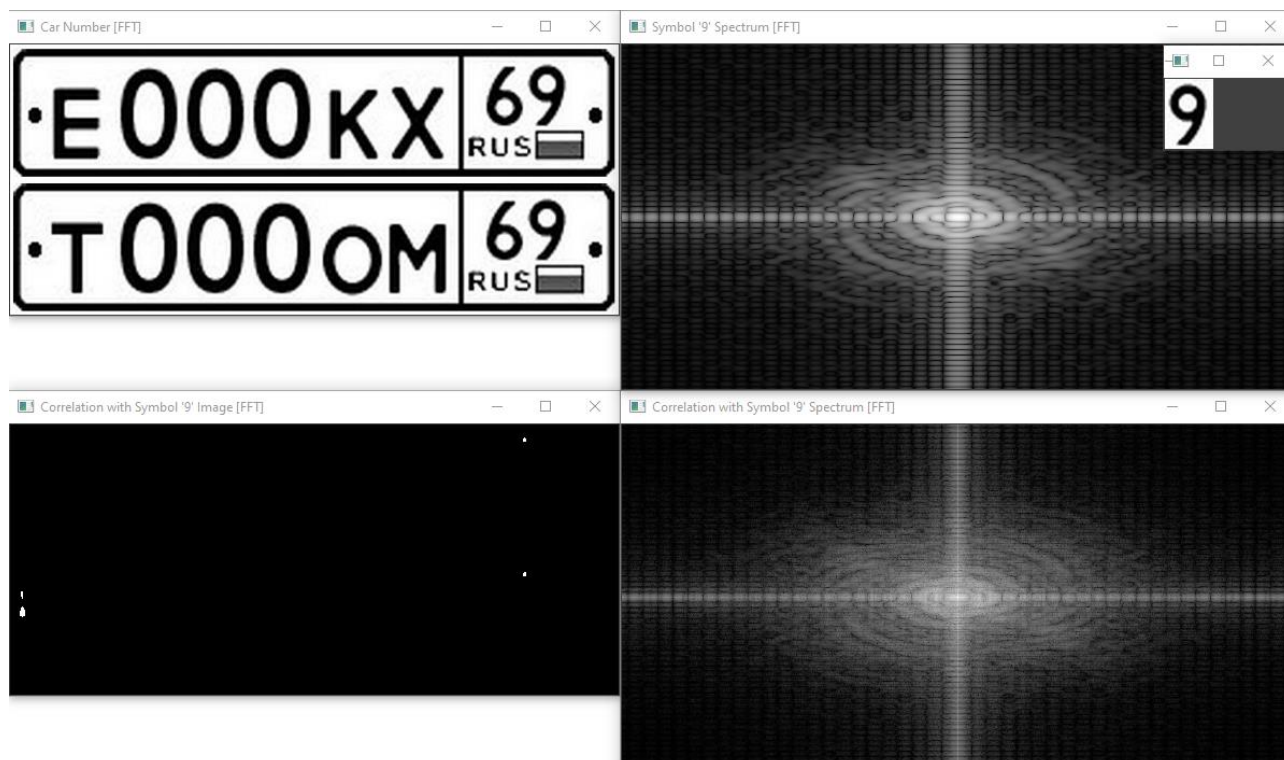


Рисунок 20 — Исходное изображение (слева сверху), символ '9' и его спектр (справа сверху), результат корреляции изображения с этим символом (слева снизу) и его спектр (справа внизу)

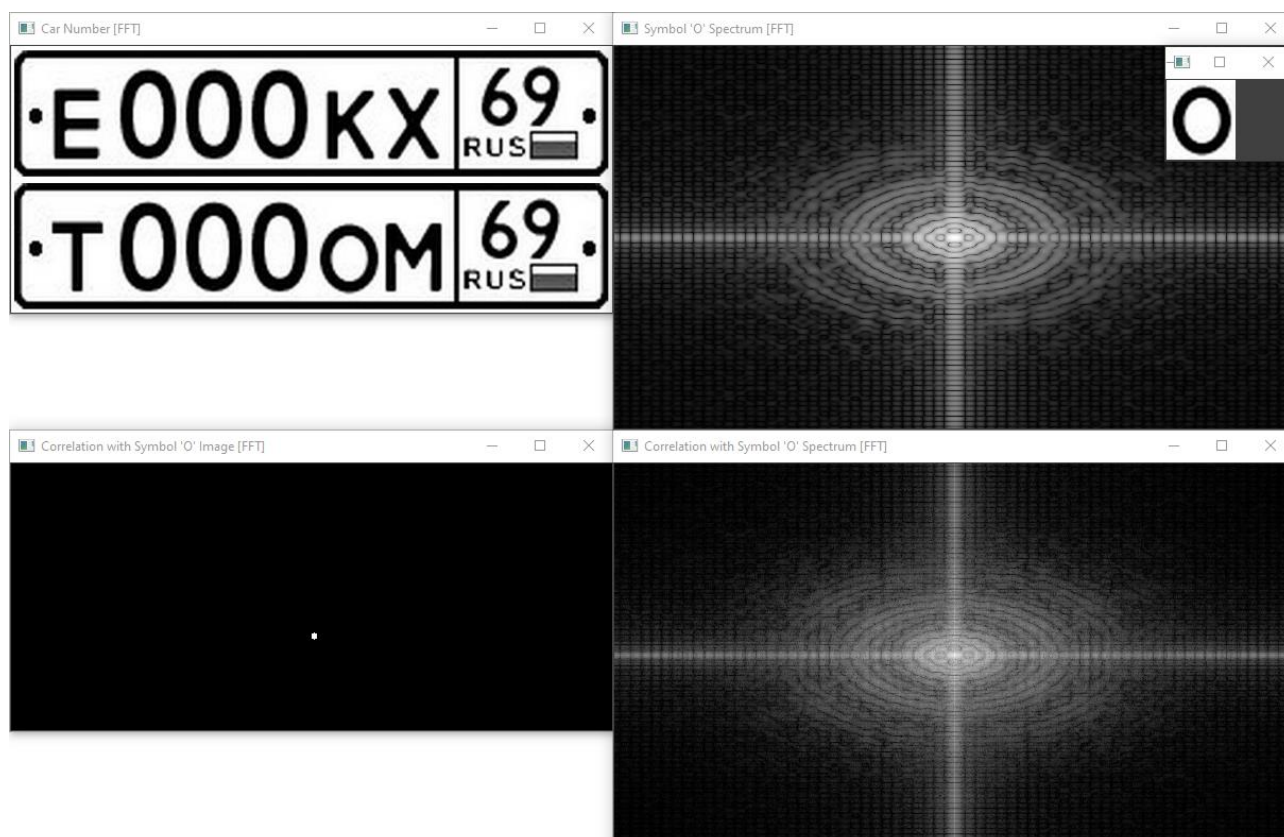


Рисунок 21 — Исходное изображение (слева сверху), символ 'O' и его спектр (справа сверху), результат корреляции изображения с этим символом (слева снизу) и его спектр (справа внизу)

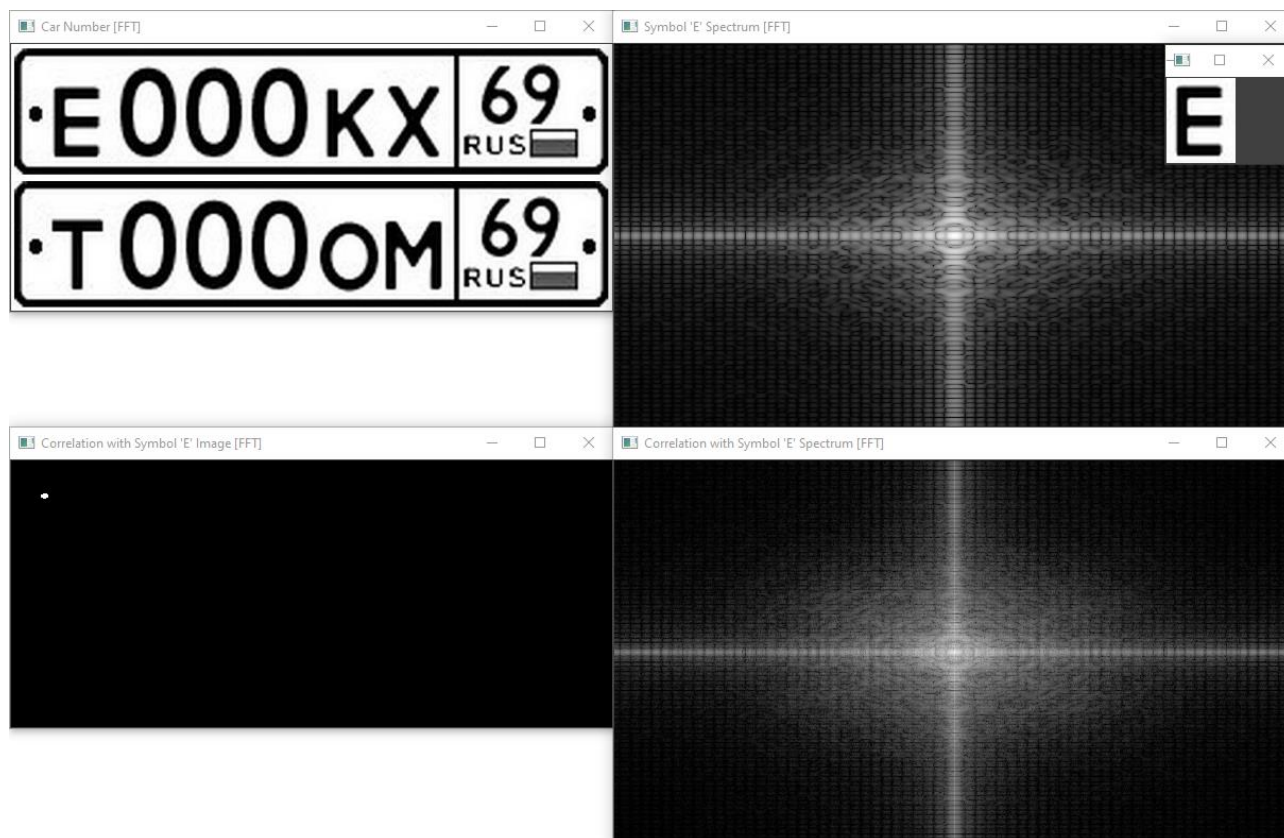


Рисунок 22 — Исходное изображение (слева сверху), символ 'Е' и его спектр (справа сверху), результат корреляции изображения с этим символом (слева снизу) и его спектр (справа внизу)