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: Фильтрация изображений в частотной области

Студент гр. 3331506/70401 Козлов Д. А.

Преподаватель Варлашин В. В.

« »_____2020 г.

Санкт-Петербург

2020

Задание

1. Реализовать прямое ДПФ, с возможностью вывода спектра, и обратное ДПФ. Сравнить результаты со встроенной функцией.
2. Реализовать режекторный и полосовой фильтры Баттерворта.
3. Произвести по отдельности свёртку какого-либо изображения с ядром фильтров: Собеля (по горизонтали и вертикали), усредняющего (BoxFilter), Лапласа $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$.
4. Полученные образы Фурье в результате выполнения свёртки следует обратно преобразовать в изображение.
5. Провести корреляцию (сравнение) изображений автомобильных номеров по очереди с 3-мя символами. Полученный образ Фурье обратно преобразовать в обычное изображение. Найти на нём наибольшее значение, которое принимают элементы. Отнять от этого значения небольшое число (около 0.01). Использовать полученное число в качестве порога для пороговой фильтрации от полученного изображения.

Ход работы

1. Прямое и обратное дискретные преобразования Фурье

Для реализации метода используем формулу двумерного дискретного преобразования Фурье (ДПФ):

$$F(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi(\frac{ux}{M}+\frac{vy}{N})}, \quad (1)$$

где $f(x,y)$ – цифровое изображение размерами $M \times N$. Выражение (1) должно быть вычислено для всех значений дискретных переменных u и v в диапазонах $u = 0, 1, 2, \dots, M-1$ и $v = 0, 1, 2, \dots, N-1$.

Если имеется преобразование $F(u,v)$, можно получить $f(x,y)$ при помощи обратного дискретного преобразования Фурье:

$$f(x,y) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{-i2\pi(\frac{ux}{M}+\frac{vy}{N})} \quad (2)$$

для $x = 0, 1, 2, \dots, M-1$ и $y = 0, 1, 2, \dots, N-1$.

Реализация двумерного прямого дискретного преобразования Фурье представлена на рисунке 1.

```
int Fourier2D::forwardDFT(cv::Mat& source, cv::Mat& dst)
{
    ...

    for (int u = 0; u < source.rows; u++)
    {
        for (int v = 0; v < source.cols; v++)
        {
            for (int x = 0; x < source.rows; x++)
            {
                for (int y = 0; y < source.cols; y++)
                {
                    m_tempImageReIm.at<cv::Vec2f>(u, v)[0] += source.at<float>(x, y)
                        * cos(2 * CV_PI * ((float)u * x / source.rows + (float)v * y / source.cols));
                    m_tempImageReIm.at<cv::Vec2f>(u, v)[1] -= source.at<float>(x, y)
                        * sin(2 * CV_PI * ((float)u * x / source.rows + (float)v * y / source.cols));
                }
            }
        }
    }
...
}
```

Рисунок 1 – Прямое ДПФ

Реализация двумерного обратного дискретного преобразования Фурье представлена на рисунке 2.

```
int Fourier2D::inverseDFT(cv::Mat& source, cv::Mat& dst)
{
...

    for (int u = 0; u < source.rows; u++)
    {
        for (int v = 0; v < source.cols; v++)
        {
            for (int x = 0; x < source.rows; x++)
            {
                for (int y = 0; y < source.cols; y++)
                {
                    float a1 = source.at<cv::Vec2f>(x, y)[0];
                    float a2 = cos(2 * CV_PI * ((float)u * x / source.rows + (float)v * y / source.cols));
                    float b1 = source.at<cv::Vec2f>(x, y)[1];
                    float b2 = sin(2 * CV_PI * ((float)u * x / source.rows + (float)v * y / source.cols));

                    m_tempRe.at<float>(u, v) += (a1 * a2 - b1 * b2);
                    m_tempIm.at<float>(u, v) += (a1 * b2 + a2 * b1);
                }
            }
            m_tempRe.at<float>(u, v) /= (source.rows * source.cols);
            m_tempIm.at<float>(u, v) /= (source.rows * source.cols);
        }
    }
...
}
```

Рисунок 2 – Обратное ДПФ

Для сравнения реализованного прямого ДПФ с методом библиотеки OpenCV найдем среднеквадратическую погрешность по следующей формуле:

$$S = \sqrt{\frac{\sum_{i=0}^{n-1}(x_i^{my} - x_i^{lib})^2}{n-1}}, \quad (3)$$

где n – количество пикселей изображения, x_i^{my} и x_i^{lib} – значение пикселя реализованного метода и встроенного соответственно.

Реализация функции вычисления среднеквадратической погрешности изображена на рисунке 3.

```
float compare(const cv::Mat img1, const cv::Mat img2, const int ch)
{
    if (img1.total() != img2.total()) { ... }
    float sumPow2 = 0;
    for (int i = 0; i < img1.rows; i++)
    {
        for (int j = 0; j < img2.cols; j++)
        {
            sumPow2 += (img1.at<cv::Vec2f>(i, j)[ch] - img2.at<cv::Vec2f>(i, j)[ch])
                * (img1.at<cv::Vec2f>(i, j)[ch] - img2.at<cv::Vec2f>(i, j)[ch]);
        }
    }
    return sqrt(sumPow2 / img1.total());
}
```

Рисунок 3 – Вычисление среднеквадратической погрешности

При обработке изображения размером 62×77 среднеквадратическая погрешность составила ≈ 0.07.

Для проверки обратного ДПФ найдем среднеквадратическое отклонение инвертированного изображения от исходного. Оно составило ≈ 0.7.

Исходя из малости среднеквадратических погрешностей, можно судить о правильности реализованных методов ДПФ.

2. Режекторный и полосовой фильтры Баттерворта

Режекторный фильтр Баттерворта задерживает определенную полосу частот и определяется выражением:

$$H_{Notch}(u,v) = \frac{1}{1 + \left[\frac{2DW}{D^2 - D_0^2}\right]^{2n}}, \quad (4)$$

где W означает ширину полосы, D – расстояние $D(u,v)$ от центра фильтра, D_0 – частота среза, а n – порядок фильтра Баттерворта.

Полосовой фильтр Баттерворта пропускает определенную полосу частот и получается вычитанием режекторного фильтра из 1:

$$H_{Band\ pass}(u,v) = 1 - H_{Notch}(u,v). \quad (5)$$

Реализация данного метода фильтрации состоит из следующих шагов:

1) Прямое ДПФ (*forwardTransform*);
2) Вычисление спектра (*calculateSpectre*);
3) Вычисление фазы (*calculatePhase*);
4) Вычисление фильтра Баттерворта 1-го порядка по формуле (4) или (5);
5) Попиксельное перемножение спектра и фильтра;
6) Обратное ДПФ из спектра и фазы (*inverseTransformFromSpectrePhase*).

Реализация функции вычисления спектра:

```
int Fourier2D::calculateSpectre(cv::Mat& image, cv::Mat& spectre)
{
    if (image.empty()) { ... }

    std::vector<cv::Mat> ReIm;
    cv::split(image, ReIm);
    magnitude(ReIm[0], ReIm[1], spectre);
    swapSpectre(spectre);

    return 0;
}
```

Рисунок 4 – Функция вычисления спектра

Реализация функции вычисления фазы:

```
int Fourier2D::calculatePhase(cv::Mat& image, cv::Mat& phase)
{
    if (image.empty()) { ... }

    cv::Mat tempPhase(image.size(), CV_32FC1);
    for (int i = 0; i < tempPhase.rows; i++)
    {
        for (int j = 0; j < tempPhase.cols; j++)
        {
            tempPhase.at<float>(i, j) = atan2(image.at<cv::Vec2f>(i, j)[1], m_image.at<cv::Vec2f>(i, j)[0]);
        }
    }

    phase = tempPhase.clone();

    return 0;
}
```

Рисунок 5 – Функция вычисления фазы

Реализация обратного ДПФ из спектра и фазы:

```
int Fourier2D::inverseTransformFromSpectrePhase()
{
...

    swapSpectre(m_spectre);

    for (int x = 0; x < m_spectre.rows; x++)
    {
        for (int y = 0; y < m_spectre.cols; y++)
        {
            m_image.at<cv::Vec2f>(x, y)[0] = m_spectre.at<float>(x, y) * cos(m_phase.at<float>(x, y));
            m_image.at<cv::Vec2f>(x, y)[1] = m_spectre.at<float>(x, y) * sin(m_phase.at<float>(x, y));
        }
    }

    inverseTransformFromImage();

    swapSpectre(m_spectre);

    return 0;
}
```

Рисунок 6 – Обратное ДПФ из спектра и фазы

Реализация метода обработки режекторным и полосовым фильтрами Баттерворта представлен на рисунке 7.

```
int Fourier2D::filterButterworth(float D0, float W, bool isNotchFilter)
{
    if (m_source.empty()) { ... }

    forwardTransform();
    calculateSpectre(m_image, m_spectre);
    calculatePhase(m_image, m_phase);

    cv::Mat butterworth(m_spectre.size(), CV_32FC1, cv::Scalar(0));

    float x0 = butterworth.rows / 2;
    float y0 = butterworth.cols / 2;

    for (int x = 0; x < butterworth.rows; x++)
    {
        for (int y = 0; y < butterworth.cols; y++)
        {
            float D = sqrt((x - x0) * (x - x0) + (y - y0) * (y - y0));
            if (isNotchFilter)
            {
                butterworth.at<float>(x, y)
                    = 1 / (1 + (D * W / (D * D - D0 * D0)) * (D * W / (D * D - D0 * D0)));
            }
            else
            {
                butterworth.at<float>(x, y)
                    = 1 - 1 / (1 + (D * W / (D * D - D0 * D0)) * (D * W / (D * D - D0 * D0)));
            }
        }
    }

    cv::normalize(butterworth, butterworth, 0, 1, cv::NORM_MINMAX);

    for (int x = 0; x < butterworth.rows; x++)
    {
        for (int y = 0; y < butterworth.cols; y++)
        {
            m_spectre.at<float>(x, y) *= butterworth.at<float>(x, y);
        }
    }

...

    inverseTransformFromSpectrePhase();

    return 0;
}
```

Рисунок 7 – Фильтр Баттерворта

Примеры работы метода обработки изображения режекторным фильтром Баттерворта с параметрами $D = 100, W = 100$ и полосовым с параметрами $D = 100, W = 50$ представлены на рисунках ниже.

Рисунок 8 – Исходное изображение

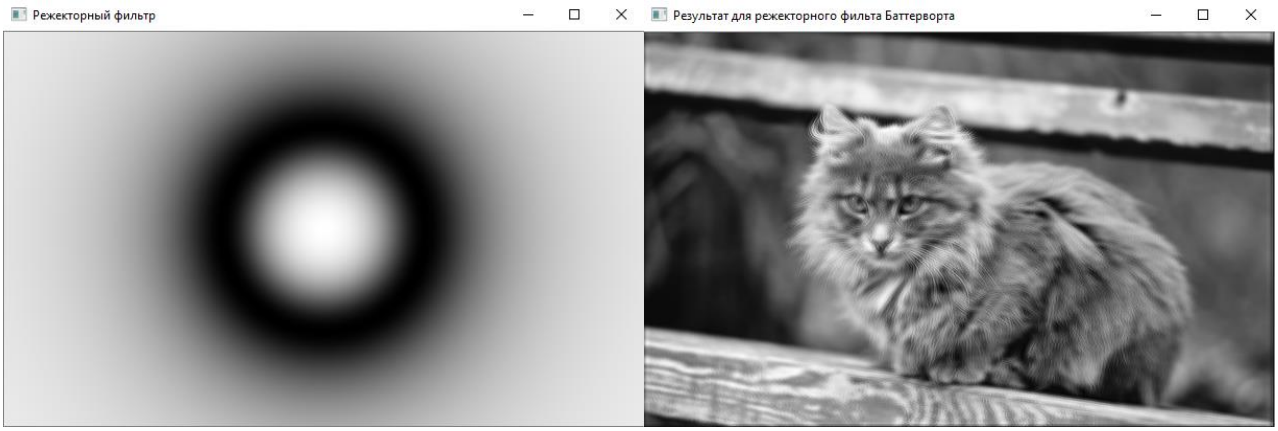

Рисунок 9 – Режекторный фильтр $D = 100, W = 100$

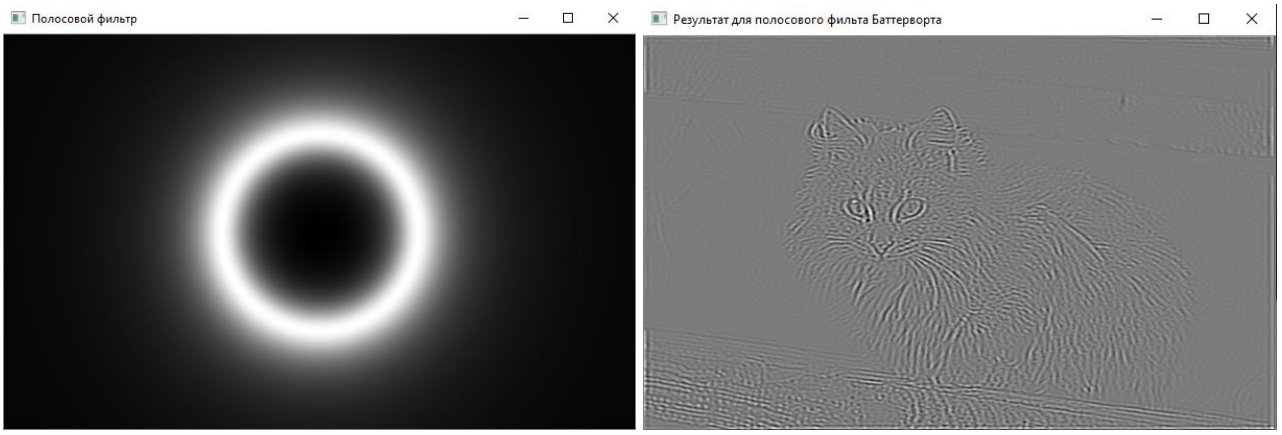

Рисунок 10 – Полосовой фильтр $D = 100, W = 50$

3, 4. Свертка с различными фильтрами

Алгоритм свертки с различными изображениями:

1) Прямое ДПФ исходного изображения;
2) Составление ядра фильтра, расширение до размера исходного изображения;
3) Прямое ДПФ фильтра;
4) Выполнение свертки (перемножение образов исходного изображения и фильтра по правилу перемножения комплексных чисел);
5) Обратное ДПФ.

Реализация метода свертки представлена на рисунке 11.

```
int Fourier2D::filterConvolution(int coreType)
{
    ...

    forwardTransform();

    cv::Mat m_core(m_sourceSize, CV_32FC1, cv::Scalar(0));
    if (coreType == 0)
    {
        m_core.at<float>(0, 0) = 1;     m_core.at<float>(0, 1) = 2;     m_core.at<float>(0, 2) = 1;
        m_core.at<float>(1, 0) = 0;     m_core.at<float>(1, 1) = 0;     m_core.at<float>(1, 2) = 0;
        m_core.at<float>(2, 0) = -1;    m_core.at<float>(2, 1) = -2;    m_core.at<float>(2, 2) = -1;
    }
    else if (coreType == 1)
    {
        m_core.at<float>(0, 0) = 1;     m_core.at<float>(0, 1) = 0;     m_core.at<float>(0, 2) = -1;
        m_core.at<float>(1, 0) = 2;     m_core.at<float>(1, 1) = 0;     m_core.at<float>(1, 2) = -2;
        m_core.at<float>(2, 0) = 1;     m_core.at<float>(2, 1) = 0;     m_core.at<float>(2, 2) = -1;
    }
    ...

    cv::Mat m_coreImage(m_core.size(), CV_32FC2, cv::Scalar(0, 0));
    forwardDFT(m_core, m_coreImage);

    for (int x = 0; x < m_image.rows; x++)
    {
        for (int y = 0; y < m_image.cols; y++)
        {
            float a1 = m_image.at<cv::Vec2f>(x, y)[0];
            float b1 = m_image.at<cv::Vec2f>(x, y)[1];
            float a2 = m_coreImage.at<cv::Vec2f>(x, y)[0];
            float b2 = m_coreImage.at<cv::Vec2f>(x, y)[1];

            m_image.at<cv::Vec2f>(x, y)[0] = a1 * a2 - b1 * b2;
            m_image.at<cv::Vec2f>(x, y)[1] = a1 * b2 + a2 * b1;
        }
    }

    ...

    inverseTransformFromImage();

    ...
}
```

Рисунок 11 – Метод свертки

Результаты работы данного метода при свертке с различными фильтрами представлены на рисунках ниже.

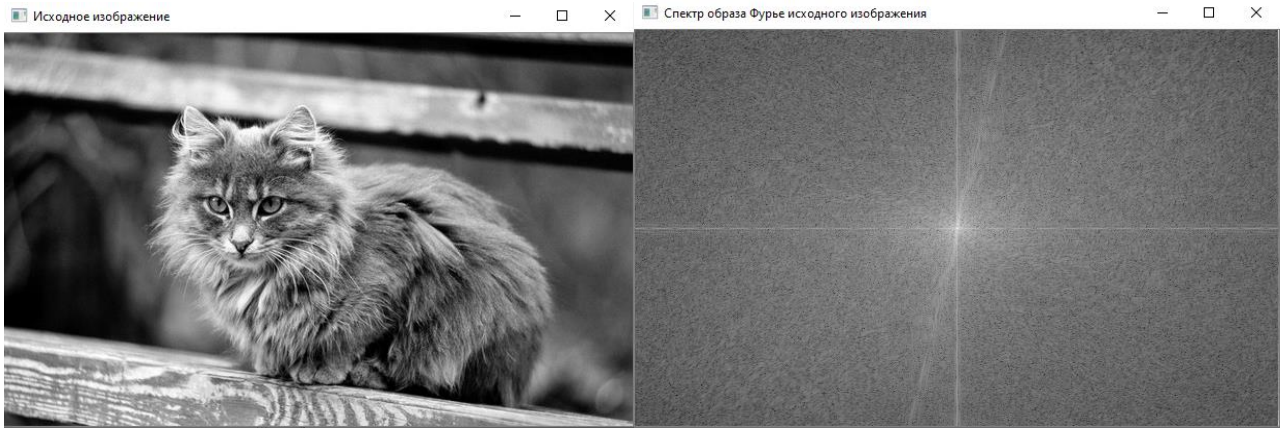

Рисунок 12 – Исходное изображение и его спектр

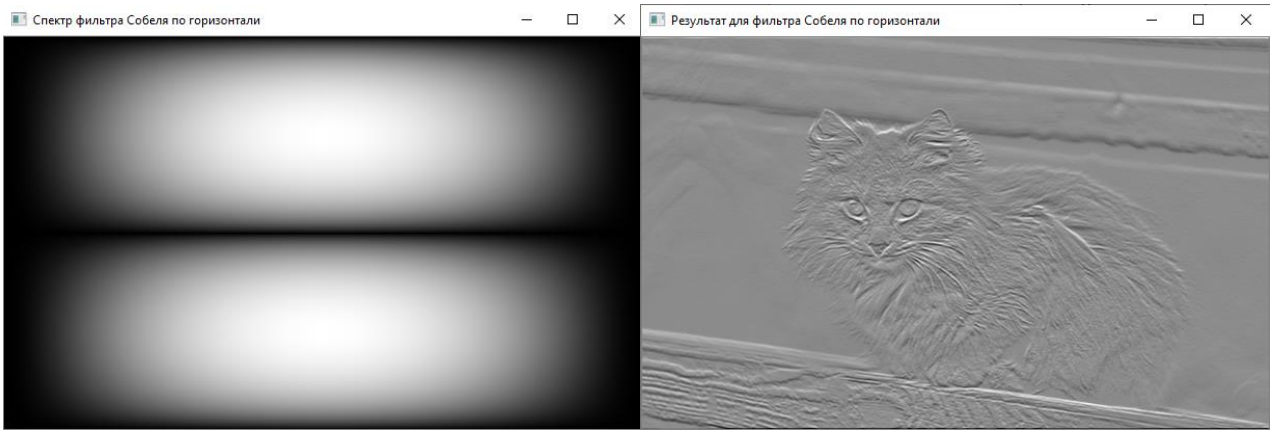

Рисунок 13 – Фильтр Собеля по горизонтали

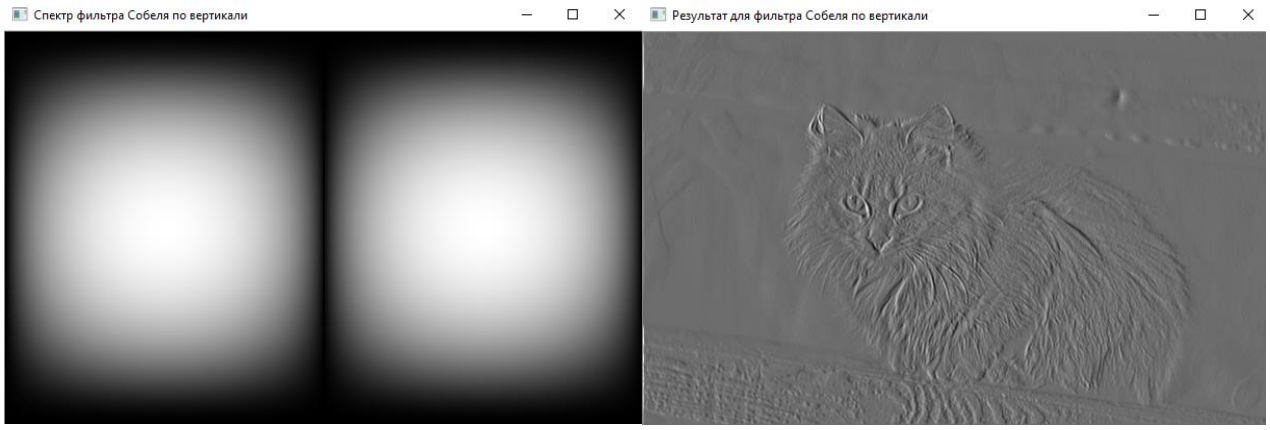

Рисунок 14 - Фильтр Собеля по вертикали

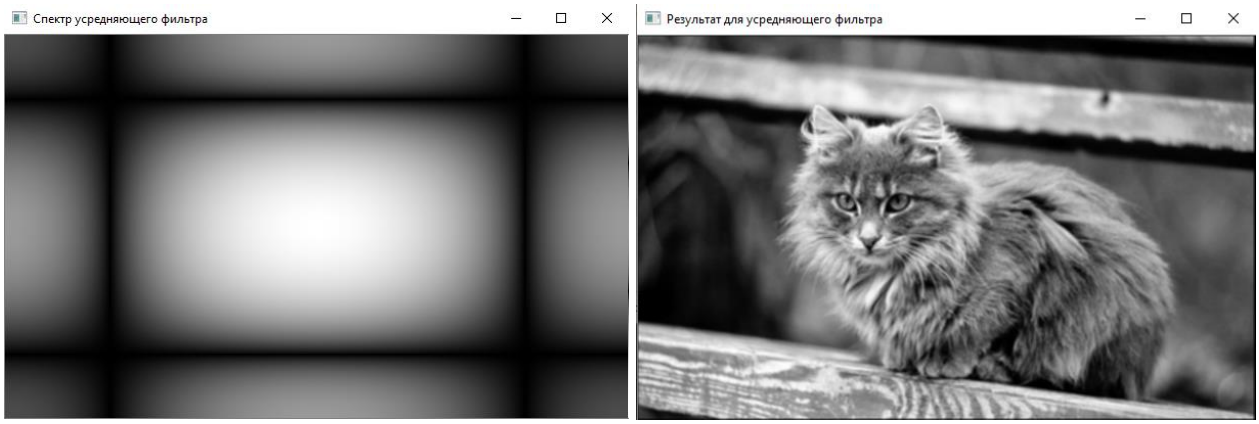

Рисунок 15 – Усредняющий фильтр

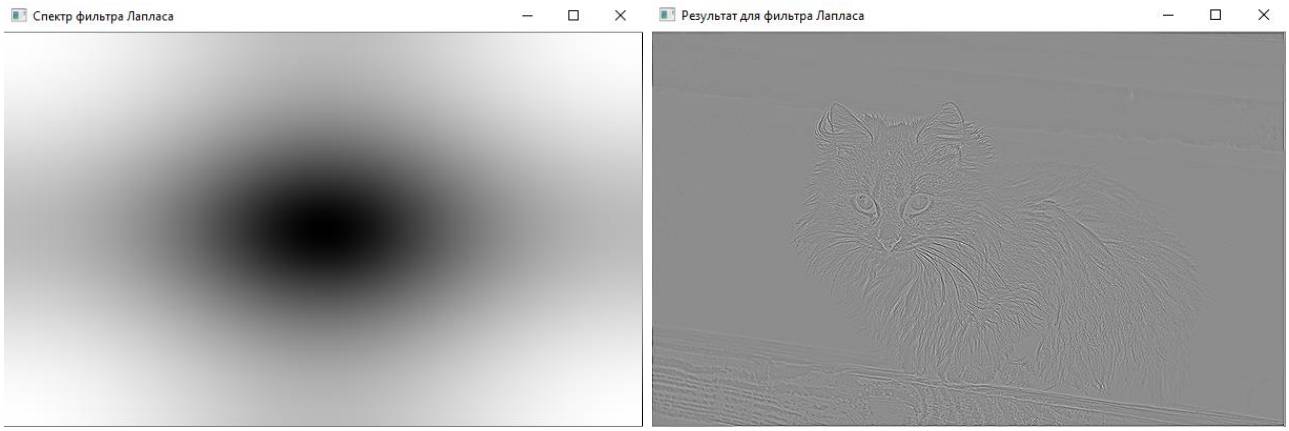

Рисунок 16 – Фильтр Лапласа

5. Корреляция

Алгоритм метода корреляции изображений:

1) Приведение изображений к одному размеру;
2) Прямое ДПФ;
3) Корреляция при помощи встроенной функции *mulSpectrums*();
4) Обратное ДПФ результата корреляции;
5) Найти на полученном изображении максимальное значение при помощи встроенной функции *minMaxLoc*();
6) Провести пороговую фильтрацию встроенной функцией *threshold*() со значением порога чуть меньшим, чем максимальное, найденное в предыдущем шаге.

Реализация метода корреляции представлена на рисунке 17.

```
int Fourier2D::correlation(cv::Mat img, std::string name)
{
    ...

    expandCanvas(img, m_source.size());
    forwardDFT(m_source, m_image);
    cv::Mat m_imgF;
    forwardDFT(img, m_imgF);

    m_result.convertTo(m_result, CV_32FC2);
    cv::mulSpectrums(m_image, m_imgF, m_result, 0, 1);

    inverseDFT(m_result, m_result);
    normalize(m_result, m_result, 0, 1, cv::NORM_MINMAX);

    ...

    double thresh;
    cv::minMaxLoc(m_result, NULL, &thresh);
    thresh -= 0.02;
    cv::threshold(m_result, m_result, thresh, 1, cv::THRESH_BINARY);
    m_result.convertTo(m_result, CV_8UC1, 255);

    return 0;
}
```

Рисунок 17 – Метод корреляции

Проведем сравнение изображения автомобильных номеров по очереди с 3 символами. Результаты сравнения представлены на рисунках ниже.

Рисунок 18 – Поиск символа ‘С’

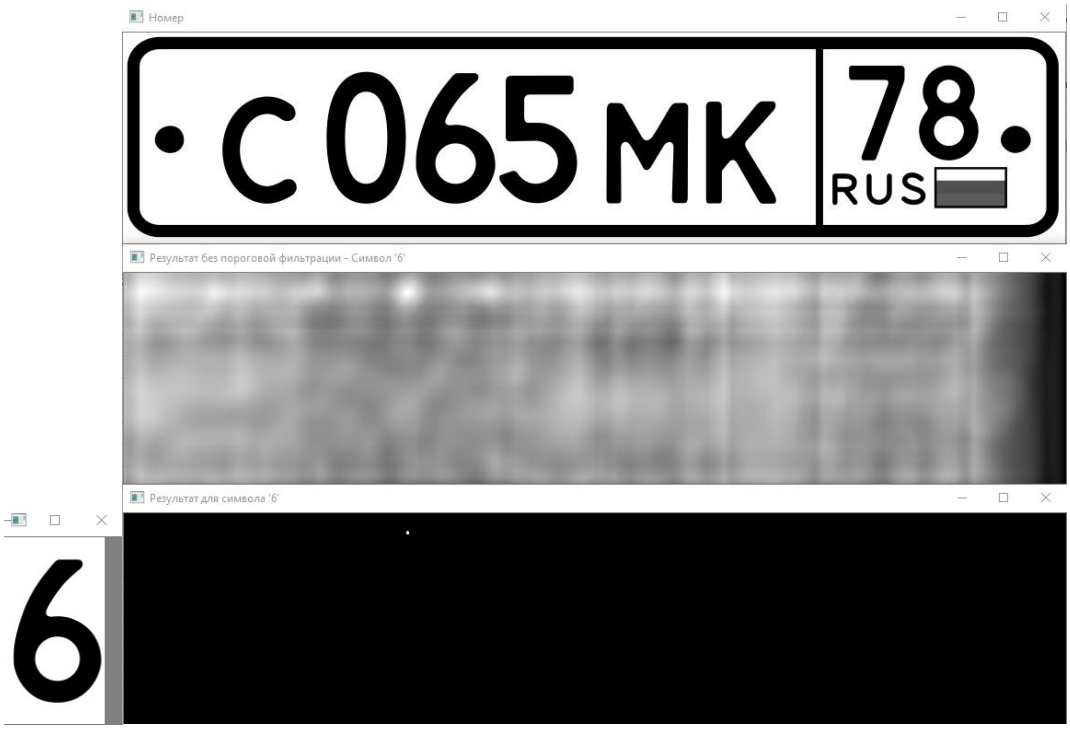

Рисунок 19 - Поиск символа ‘6’

Рисунок 20 - Поиск символа ‘78’

Вывод

В результате выполнения лабораторной работы были реализованы прямое и обратное дискретные преобразования Фурье, различные методы фильтрации изображений в частотной области, рассмотрены некоторые возможности библиотеки OpenCV в области частотной фильтрации изображений.

Дополнительное задание

В качестве дополнительного задания необходимо реализовать генератор изображений, который работает следующим образом: пользователь кликает по окну, отвечающему за амплитуду образа Фурье, тем самым изменяя её, затем производится обратное преобразование, и в новое окно выводится результат. Для упрощения новая амплитуда равна 1.

Для реализации данного генератора изображений необходимо учесть свойство симметрии прямого и обратного ДПФ – Фурье-образ действительной функции $f(x, y)$ будет симметрично сопряженным:

$$F^*(u, v) = F(-u, -v),$$

это означает, что спектр имеет центральную симметрию:

$$|F(u, v)| = |F(-u, -v)|.$$

Клики мышки по окну реализованы при помощи встроенной функции *setMouseCallback*().

Результаты работы программы для различных спектров изображены на рисунках ниже.

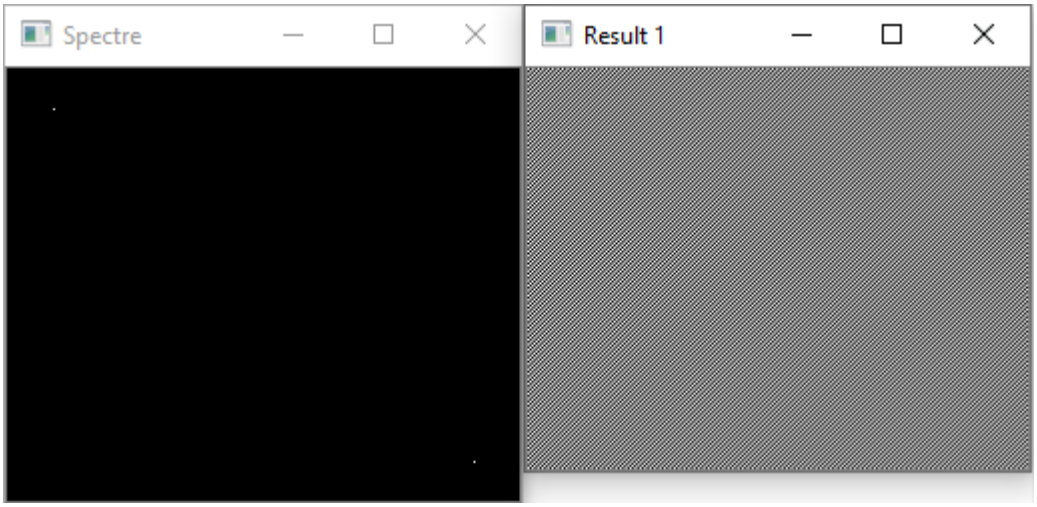

Рисунок 21 – Точка вдали от центра

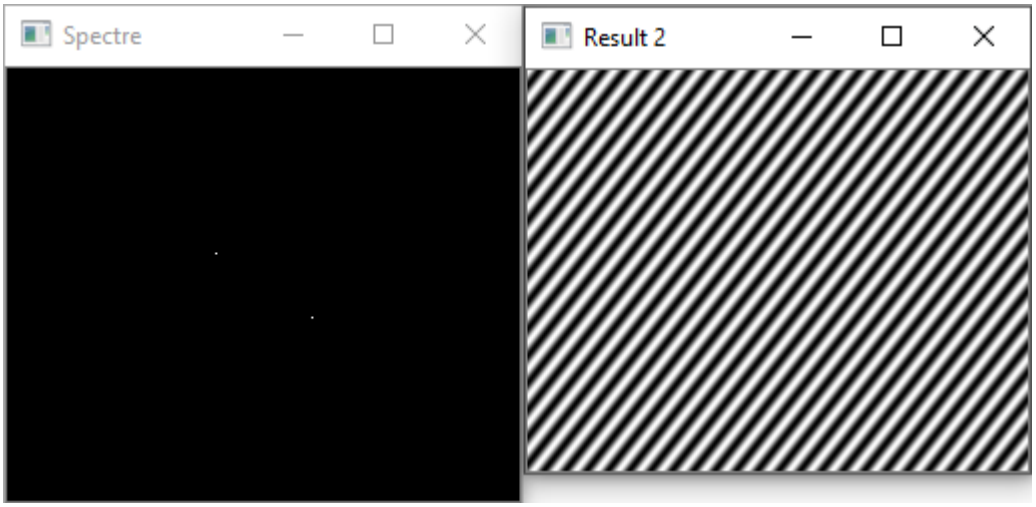

Рисунок 22 – Точка близко к центру

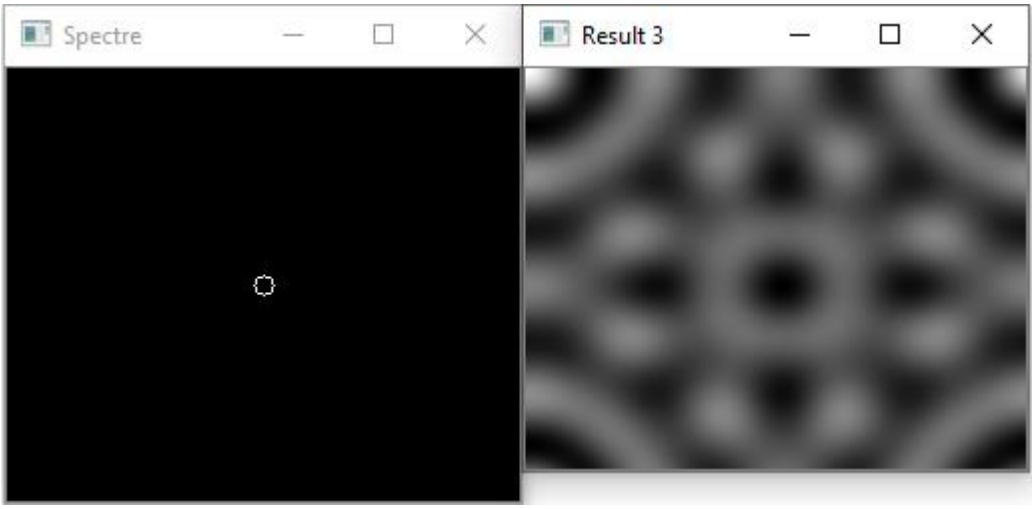

Рисунок 23 – Окружность радиуса 5 пикселей

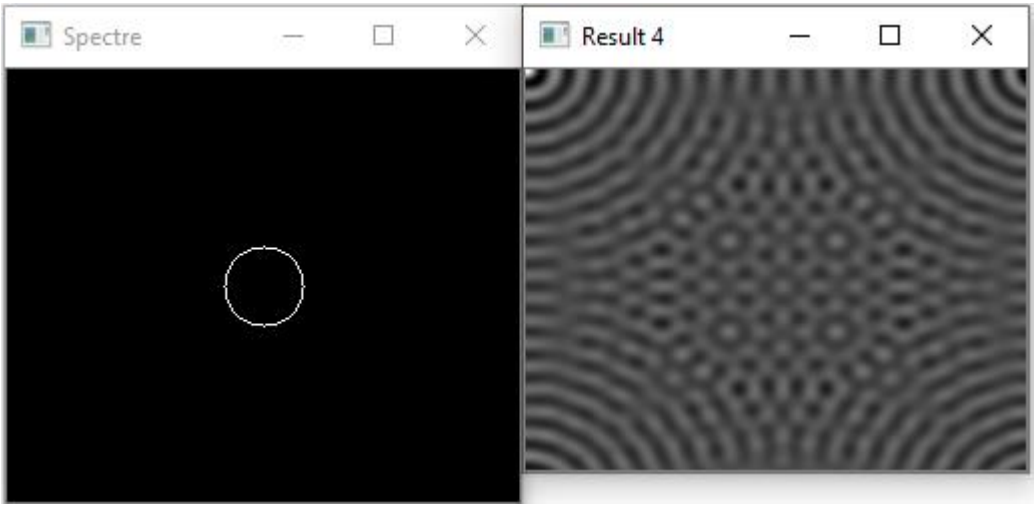

Рисунок 24 – Окружность радиуса 20 пикселей

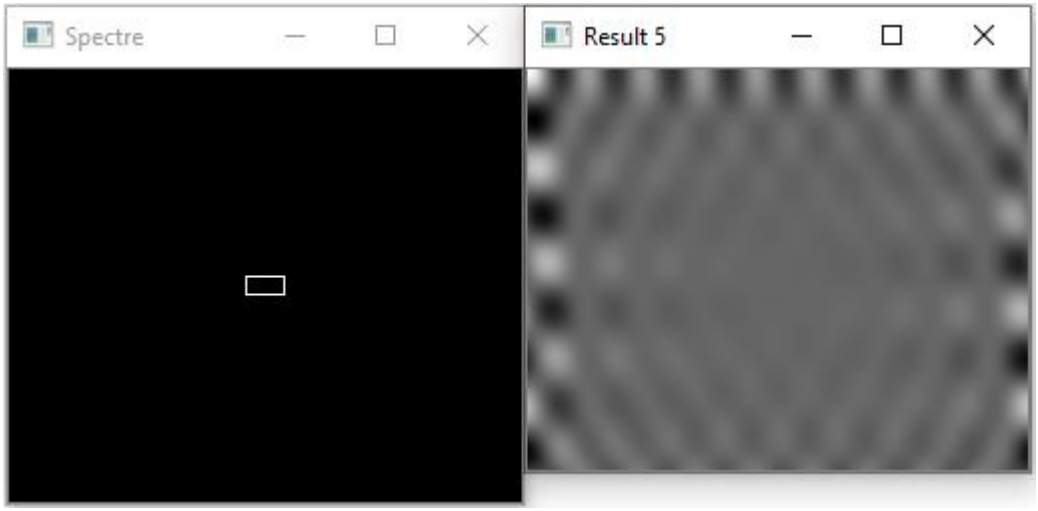

Рисунок 25 – Прямоугольник шириной 20 пикселей и высотой 10 пикселей

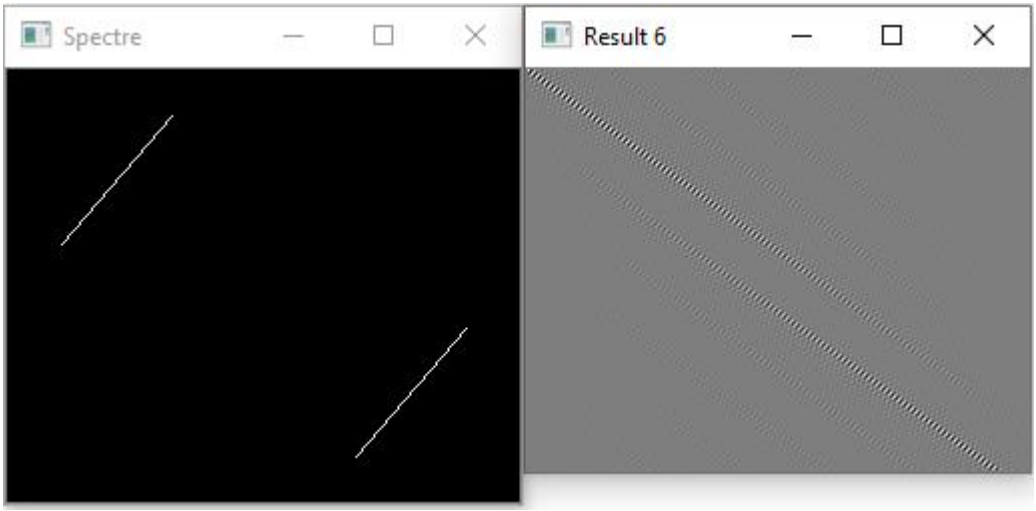

Рисунок 26 – Прямая вдали от центра

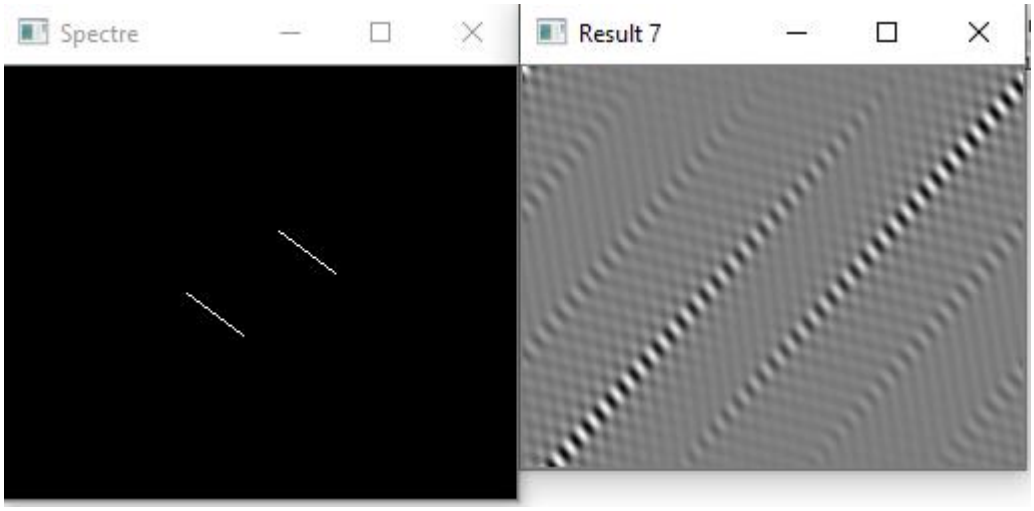

Рисунок 27 – Прямая близко к центру

Список использованной литературы

1. **Гонсалес Р., Вудс Р.** Цифровая обработка изображений / Москва: Техносфера, 2012. – 1104 с.
2. OpenCV modules. [Электронный ресурс]. URL: http://docs.opencv.org/trunk/