Санкт-Петербургский политехнический университет Петра Великого

Институт машиностроения, материалов и транспорта

Высшая школа машиностроения

Отчет

По лабораторной работы № 4

Студент гр. 3331506/90401 Д.В. Гаврилов

Преподаватель Титов В.В

Санкт-Петербург

2023

1. Написать свою функцию преобразования Фурье (прямое и обратное) используя «лобовой подход».

Для этого задания я создал класс *myDiscrFurie* в котором есть метод расчет матрицы коэффициентов и методы для расчета DTF и IDTF, основанные на матричной формы

Сам класс показан на рисунке 1.

```
class myDiscrFurie {
    const int _M;
    const complex<double> ci = complex<double>(std::polar(1.0,(2 * PI / _M))); // получение константы из методички
    bool WCalc = false;

    void calkW() // расчет матрицы комплесных экспонент
    {
        cout << endl;
        cout << endl;

        WCalc = true;

        for (int i = 0; i < _M; i++) {
            for (int j = 0 + i; j < _M; j++) {
                W(i, j) = pow(ci, i * j); // я так и не смог понять, как это делать рекурсивно
                W(j, i) = W(i, j);        // так как матрица симметрична относительно главной диагонали
            }
        }
    }
public:
    cv::Mat_<complex<double>> F;
    cv::Mat_<complex<double>> W;
    cv::Mat_<complex<double>> f;

    myDiscrFurie(double _inputArray[], int size) : _M(size)
    {
        W = cv::Mat_<complex<double>>(_M, _M);
        F = cv::Mat_<complex<double>>(_M,  1);
        f = cv::Mat_<complex<double>>(_M,  1);
    }

    void IDFT()
    {
        if (!WCalc) { calkW(); }
        f = W * F;
        f /= _M;
    }

    void DTF() {
        if (!WCalc) { calkW(); }
        F = W * f;
    }

};
```

Рис. 1 - Класс для лобового метода

2. Написать преобразование Фурье используя алгоритм Radix-2 (по основанию 2, или «Бабочка»)

Читая методичку про преобразования по основанию 2, заметил что алгоритм разбиения очень похож на алгоритм сортировки слиянием. Поэтому за скелет и взял алгоритм этой сортировки.
Рекурсивное деления на пары представленно на рисунке 2.

```
void fftRecurs(vector<complex<double>>* arr)
{
    int M = arr->size();
    if (M <= 1) { return; } // значит на уровне выше двухточеные ДПФ

    vector< complex<double>> odd(M / 2);
    vector< complex<double>> even(M / 2);

    // сам процесс дробления
    for (int i = 0; i < M / 2; i++) {
        even.at(i) = arr->at(i * 2);
        odd.at(i) = arr->at(i * 2 + 1);
    }

    fftRecurs(&even);
    fftRecurs(&odd);

    // само ДПФ
    for (int k = 0; k < M / 2; k++) {
        std::complex<double> bw = odd.at(k) * exp(std::complex<double>(0, -2 * PI * k / M)); // B * Wn
        arr->at(k) = even.at(k) + bw;
        arr->at(M / 2 + k) = even.at(k) - bw;
    }
}
```

Рис. 2 - Рекурсия

Как только мы дойдем до двухточечного ДПФ и дальше в цикле происходит расчет коэффициента и по формуле показанной ниже

$$\hat{F}(0) = \tilde{f}(0) + W_8^0 \cdot \tilde{f}(1)$$
$$\hat{F}(1) = \tilde{f}(0) + W_8^4 \cdot \tilde{f}(1)$$

Получаем выходные значение коэффициентов ряда

3. Сравнить быстродействие из функций из первой и второй части, а также с преобразованием из OpenCV

Результаты сравнения, для 8000 элементов

Лобовой метод такой долгий из-за большего количества комплексных перемножений, по сравнению с быстрым преобразованием и в моем коде используется возведение в степень, которая тоже является медленной операцией.

OpenCV функция быстрее всего, потому что, как я могу предположить, реализования более оптимальная работа с комплексными числами и может иметь заранее подсчитанные экспоненциальные коэффициенты.

4. Операции свертки и корреляции при помощи преобразовании Фурье и использование различных фильтров

Для упрощение работы я создал функцию easyFT, которая получается на первым параметром изображение и возвращает, второй и третий параметр, магнитуду образа фурье в логарифмическом масштабе и нормализованную от 0 до 1 в “красивом спектре” и изображение после обратного преобразования соответственно.

Сама функция представлена на рисунке 3

```
void easyFT(Mat& I, Mat& magI, Mat& out) {
    Mat padded;
    int m = getOptimalDFTSize(I.rows);
    int n = getOptimalDFTSize(I.cols);

    copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));
    Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
    Mat complexI;
    merge(planes, 2, complexI);         // Add to the expanded another plane with zeros
    dft(complexI, complexI);            // this way the result may fit in the source matrix
                                        // compute the magnitude and switch to logarithmic scale
                                        // => log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))

    split(complexI, planes);                   // planes[0] = Re(DFT(I), planes[1] = Im(DFT(I))
    magnitude(planes[0], planes[1], planes[0]);// planes[0] = magnitude

    magI = planes[0].clone();
    magI += Scalar::all(1);                    // switch to logarithmic scale
    log(magI, magI);

    // crop the spectrum, if it has an odd number of rows or columns
    magI = magI(Rect(0, 0, magI.cols & -2, magI.rows & -2));
    // rearrange the quadrants of Fourier image  so that the origin is at the image center
    int cx = magI.cols / 2;
    int cy = magI.rows / 2;
    Mat q0(magI, Rect(0, 0, cx, cy));   // Top-Left - Create a ROI per quadrant
    Mat q1(magI, Rect(cx, 0, cx, cy));  // Top-Right
    Mat q2(magI, Rect(0, cy, cx, cy));  // Bottom-Left
    Mat q3(magI, Rect(cx, cy, cx, cy)); // Bottom-Right
    Mat tmp;                           // swap quadrants (Top-Left with Bottom-Right)
    q0.copyTo(tmp);
    q3.copyTo(q0);
    tmp.copyTo(q3);
    q1.copyTo(tmp);                    // swap quadrant (Top-Right with Bottom-Left)
    q2.copyTo(q1);
    tmp.copyTo(q2);
    normalize(magI, magI, 0, 1, NORM_MINMAX); // Transform the matrix with float values into a
                                              // viewable image form (float between values 0 and 1)

    // IDFT
    cv::dft(complexI, out, DFT_INVERSE | DFT_REAL_OUTPUT);
    normalize(out, out, 0, 1, NORM_MINMAX);
}
```

Рис. 3 - easyTF

Результат использования фильтров показан на рисунке 4

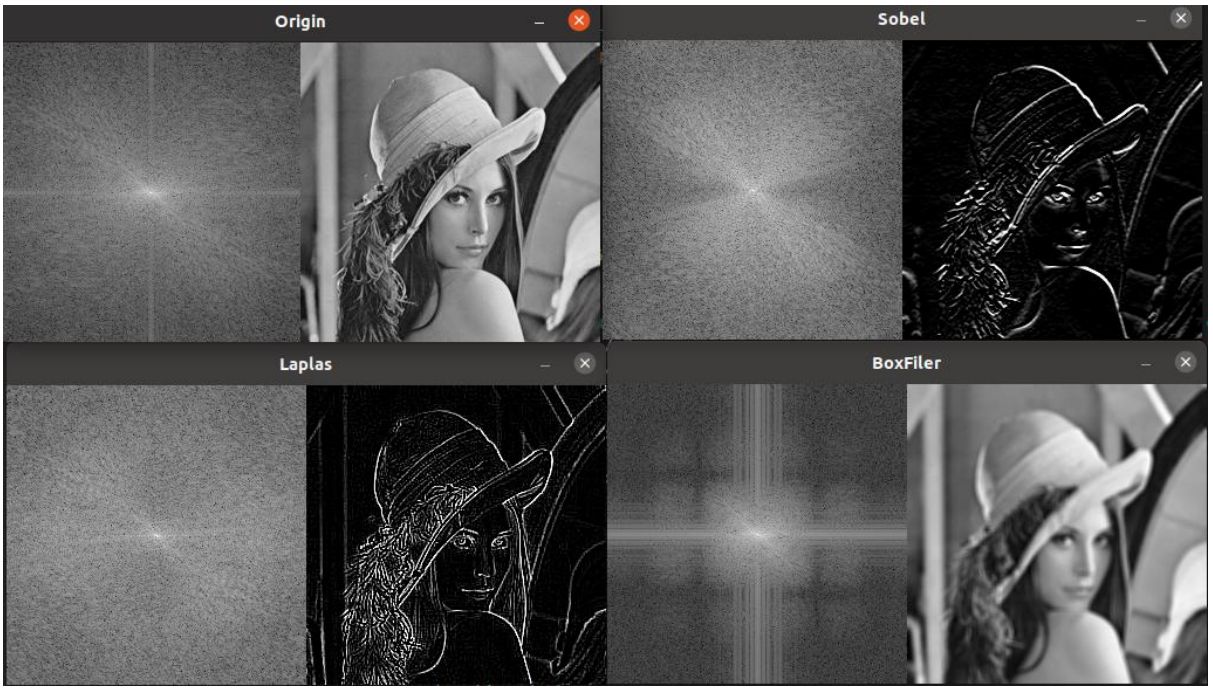
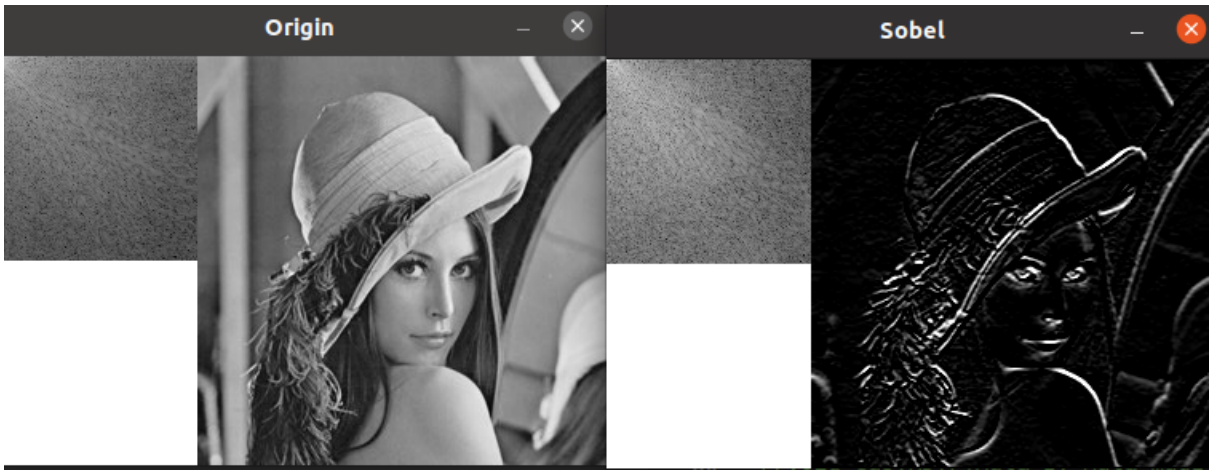


Рис. 4 - результаты

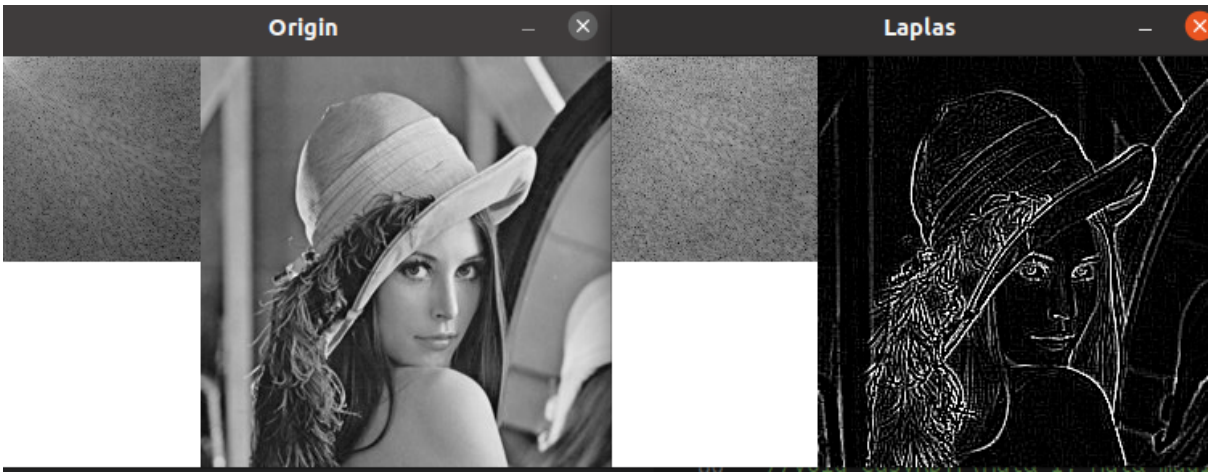
Мне почему-то удобнее сравнивать изображения не приведенные в красивый спектр и обрезанные по одному квадранту, из-за свойства симметрии в преобразования.

При использовании Собеля, мы получили более выраженные края

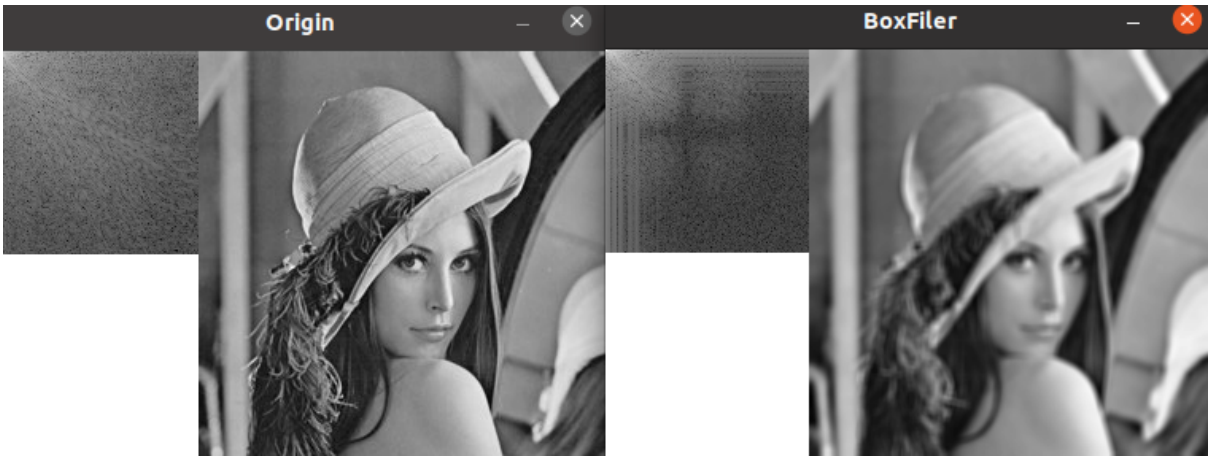


Из-за этого поменялась частотная область, высокие частоты стали больше, это понятно из-за более светлой правой нижней части изображения в частотной области

Аналогичная ситуация с Лапласом, но там из-за более резких перепадов интенсивности, описанная выше область стала еще светлее



Обратная ситуация с Box Filter, перепады интенсивности стали меньше и поэтому зоны низких частот стали ярче



5. Работа с изображением в частотной области

LowPassFilter

```
void lowPassFilter(Mat& I, Mat& out) {
    Mat padded;
    int m = getOptimalDFTSize(I.rows);
    int n = getOptimalDFTSize(I.cols);

    copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));
    Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
    Mat complexI;
    merge(planes, 2, complexI);         // Add to the expanded another plane with zeros
    dft(complexI, complexI);

    Vec2f cmplxValue;
    double cutOff = 350;
    double D; // radius
    double centerI = complexI.rows / 2;
    double centerJ = complexI.cols / 2;
    double H;
    for (int i = 0; i < complexI.rows; i++) {
        for (int j = 0; j < complexI.cols; j++) {
            D = sqrt(pow(i-centerI,2) + pow(j-centerJ,2));
            H = D <= cutOff ? 0 : 1;
            cmplxValue = complexI.at<Vec2f>(i, j);
            cmplxValue.val[0] *= H;
            cmplxValue.val[1] *= H;
            complexI.at<Vec2f>(i, j) = cmplxValue;
        }
    }

    // IDFT
    cv::dft(complexI, out, DFT_INVERSE | DFT_REAL_OUTPUT);
    normalize(out, out, 0, 1, NORM_MINMAX);
}
```

Первая часть работы этой функции аналогичная с easyTF, только в отличии от нее, я не перевожу в логарифмический масштаб и не нормализую изображение, а работаю с тем что получаю на выходе функции openCV dtf().

Так как нам нужно получить более размытое изображение, нам нужно убрать высокие частоты, в моем случае сделать 0 все значения из центра изображения в частотной области, что и реализовано внутри двух циклов for

HighPassFilter

```
void hightPassFilter(Mat& I, Mat& out) {
    Mat padded;
    int m = getOptimalDFTSize(I.rows);
    int n = getOptimalDFTSize(I.cols);

    copyMakeBorder(I, padded, 0, m - I.rows, 0, n - I.cols, BORDER_CONSTANT, Scalar::all(0));
    Mat planes[] = { Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F) };
    Mat complexI;
    merge(planes, 2, complexI);         // Add to the expanded another plane with zeros
    dft(complexI, complexI);

    Vec2f cmplxValue;
    double cutOff = 350;
    double D; // radius
    double centerI = complexI.rows / 2;
    double centerJ = complexI.cols / 2;
    double H;
    for (int i = 0; i < complexI.rows; i++) {
        for (int j = 0; j < complexI.cols; j++) {
            D = sqrt(pow(i - centerI, 2) + pow(j - centerJ, 2));
            H = D >= cutOff ? 0 : 1;
            cmplxValue = complexI.at<Vec2f>(i, j);
            cmplxValue.val[0] *= H;
            cmplxValue.val[1] *= H;
            complexI.at<Vec2f>(i, j) = cmplxValue;
        }
    }

    // IDFT
    cv::dft(complexI, out, DFT_INVERSE | DFT_REAL_OUTPUT);
    normalize(out, out, 0, 1, NORM_MINMAX);
}
```

Логика такая же как и в LowPassFilter, только теперь мы обнуляем значения, которые по краям изображения, чтобы получить более резкую картинку.

6. Корреляция изображения при помощи преобразования Фурье

Обычно корреляция производится проходом шаблона по всему изображения и на каждом переходе сравнение пикселей, что весьма ресурсно затратная задача.

Образ фурье полученный из изображения хранит информацию всего изображения в частотной области. Получается если перемножить два образа мы выделим схожие зоны и после обратного преобразования получим выделяющиеся места, которые и являются зоной нашего интереса, т.е. место нахождения фрагмента совпадающего с нашим шаблоном.

Код алгоритма корреляции

```cpp
int main(void) {
    Mat A = imread("/home/dima/Desktop/lab4/img/num1.jpg", IMREAD_GRAYSCALE);
    Mat B = imread("/home/dima/Desktop/lab4/img/template.jpg", IMREAD_GRAYSCALE);

    int dft_M = getOptimalDFTSize(A.rows);
    int dft_N = getOptimalDFTSize(A.cols);

    Mat dft_A = Mat::zeros(dft_M, dft_N, CV_32F);
    Mat dft_B = Mat::zeros(dft_M, dft_N, CV_32F);

    Mat dft_A_part = dft_A(Rect(0, 0, A.cols, A.rows));
    Mat dft_B_part = dft_B(Rect(0, 0, B.cols, B.rows));
    // это пороговое преобразование
    A.convertTo(dft_A_part, dft_A_part.type(), 1, -mean(A)[0]);
    B.convertTo(dft_B_part, dft_B_part.type(), 1, -mean(B)[0]);

    dft(dft_A, dft_A, 0, A.rows);
    dft(dft_B, dft_B, 0, B.rows);

    mulSpectrums(dft_A, dft_B, dft_A, 0, true);
    idft(dft_A, dft_A, DFT_SCALE, A.rows + B.rows - 1);

    Mat corr = dft_A(Rect(0, 0, A.cols , A.rows));
    normalize(corr, corr, 0, 1, NORM_MINMAX, corr.type());
    pow(corr, 3.0, corr);

    imshow("Correlation", corr);
    waitKey(-1);

    return 0;
}
```

Я получаю два изображения равных размеров, на одном исходная картинка(она с приставкой A) и шаблон(приставка B), провожу пороговое преобразование, для наложения исходных изображений на новые, с удобными размерами, и сама пороговая фильтрация для получения более “чистого образа Фурье”, получаю резкие изменения интенсивности. Потом перемножаю спектры, нормализую и возвожу в степень чтобы было видно изображение хорошо.