

Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего
образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИТМО

Факультет систем управления и робототехники

Лабораторная работа № 6
"Обработка изображений"

по дисциплине Частотные методы

Выполнил: студент гр. R3242

Мареев П. А.

Преподаватель: Перегудин Алексей Алексеевич

Санкт-Петербург, 2025

Содержание

1	Задание 1. Фильтрация изображений с периодичностью.	3
1.1	Выводы	7
2	Задание 2. Исследование свертки.	8
2.1	Выводы	25
3	Выводы по лабораторной работе.	26

Вступление

Современные технологии обработки изображений основаны на сложных математических методах, позволяющих анализировать, преобразовывать и улучшать визуальные данные. Лабораторная работа посвящена изучению двух ключевых подходов в этой области: частотного анализа с использованием преобразования Фурье и пространственной фильтрации через операцию свертки. Эти методы находят применение в таких областях, как цифровая фотография, медицинская визуализация, компьютерное зрение и робототехника, где точность обработки изображений напрямую влияет на качество конечных результатов.

Цель работы – освоить практические навыки анализа периодических структур в изображениях с помощью Фурье-преобразования, а также исследовать влияние различных ядер свертки на визуальные характеристики изображений. В рамках первого задания рассматривается выделение и подавление гармонических шумов при помощи частотной фильтрации. Второе задание посвящено сравнению эффектов размытия, повышения резкости и выделения границ при помощи пространственных фильтров.

Работа в среде MATLAB позволит наглядно продемонстрировать связь между пространственной и частотной областями, а также проверить теоретические сведения на практике. Особое внимание уделяется интерпретации результатов: анализу спектров, сравнению методов размытия и оценке эффективности фильтрации.

1 Задание 1. Фильтрация изображений с периодичностью.

Выберем изображение для выполнения лабораторной работы из представленных на [этом гугл-диске](#), представим его на рисунке 1.



Рис. 1. Исходное изображение.

Выполним [программную](#) обработку изображения при помощи средств пакета MATLAB. Приведем изображение полученного нормализованного логарифма модуля Фурье-образа на рисунке 2.

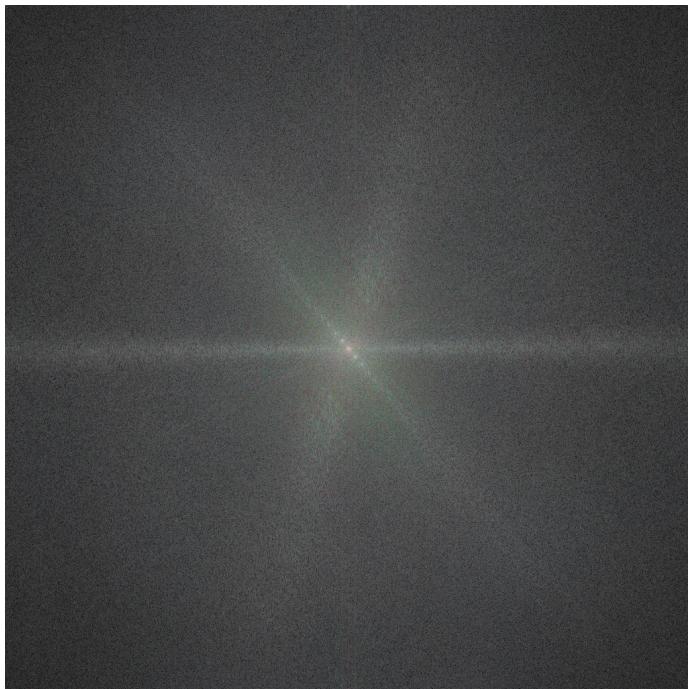


Рис. 2. Нормализованный логарифм модуля Фурье-образа.

На представленном изображении четко видны яркие пики, которые соответствуют периодическим структурам исходной картинки. Заметим, что пики расположены на диагональной прямой симметрично относительно центра.

Следовательно, для фильтрации периодичности исходной картинки необходимо избавиться от пики на полученном Фурье-образе. В удобной программе для обработки изображений сгладим все ненужные цветовые пики, отвечающие за гармоники, которые требуется подавить: покрасим все пики, кроме центрального черным цветом. Представим результаты изменений на рисунке 3.

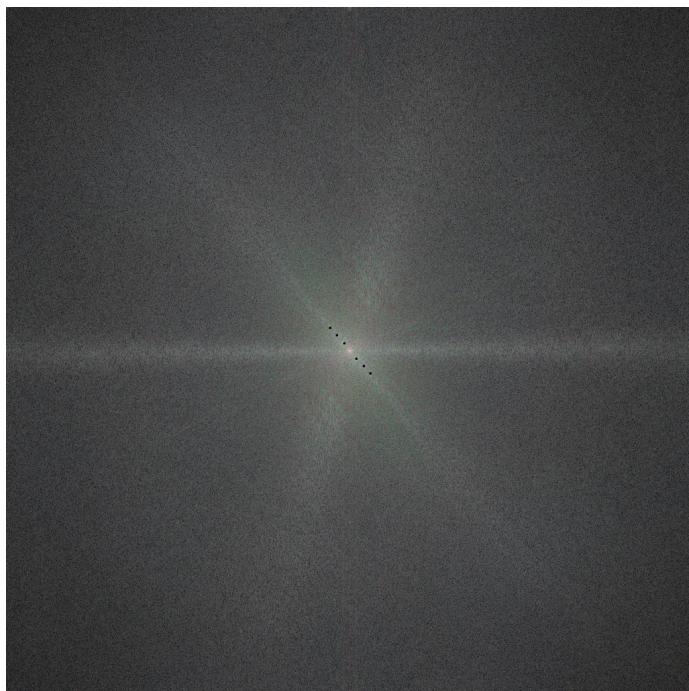


Рис. 3. Модифицированный нормализованный логарифм модуля Фурье-образа.

Восстановим исходную картинку из отредактированного образа, про- делав обратные шаги при помощи [программы](#). Приведем фильтрованное изображение на рисунке 4. Для наглядности результатов получим Фурье-образ фильтрованного изображения, а затем представим его на рисунке 5.

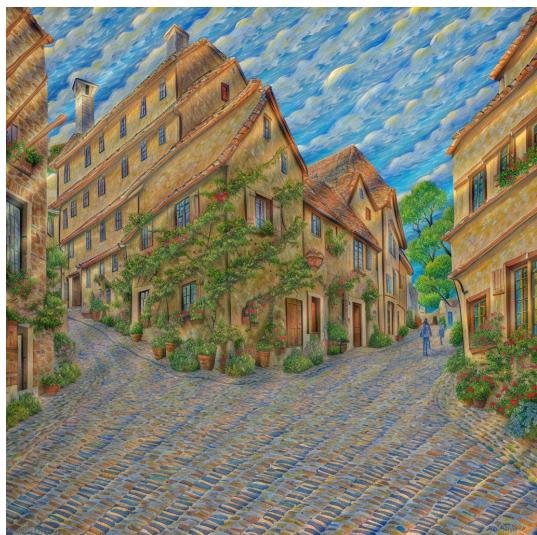


Рис. 4. Изображение, полученное после фильтрации.

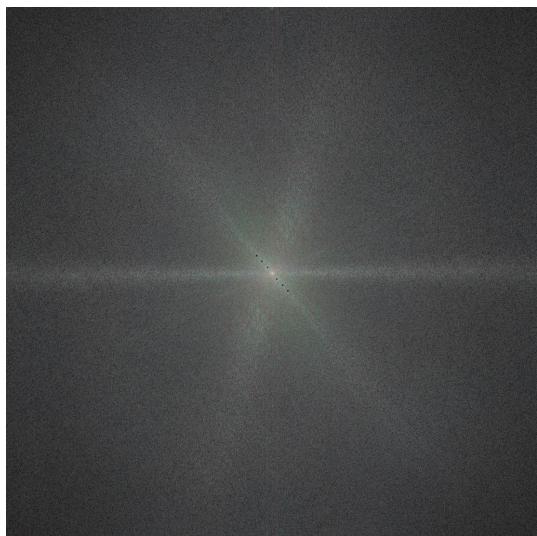


Рис. 5. Фурье-образ фильтрованного изображения.

Проанализируем полученные результаты.

Удалось успешно выполнить фильтрацию, изображения с периодичностью: на рисунке 1 отчетливо видны гармоники, создающие периодическую структуру изображения, при этом на рисунке 4 они практически незаметны. Шумы изначального рисунка исчезли, он стал однороднее, артефактов в процессе преобразований не возникло.

Кроме того, корректность операций подтверждается сравнением спектров изображений: на рисунке 2 видны пики, которые влияют на периодичность изображения, а на рисунках 3 и 5 они затемнены, что указывает на подавление периодичности фильтрованного изображения.

1.1 Выводы.

Сделаем выводы на основе результатов выполнения данного задания.

Фильтрация в частотной области на основе преобразования Фурье позволяет устраниТЬ периодические шумы, например, полосы, за счет подавления соответствующих гармоник в спектре.

Эффективность и корректность фильтрации может быть исследована при помощи соответствующих Фурье-образов для изображений.

Таким образом, многомерное преобразование Фурье является эффективным инструментом фильтрации изображений с периодичностью.

2 Задание 2. Исследование свертки.

Приведем изображение, используемое в рамках данного задания, на рисунке 6.

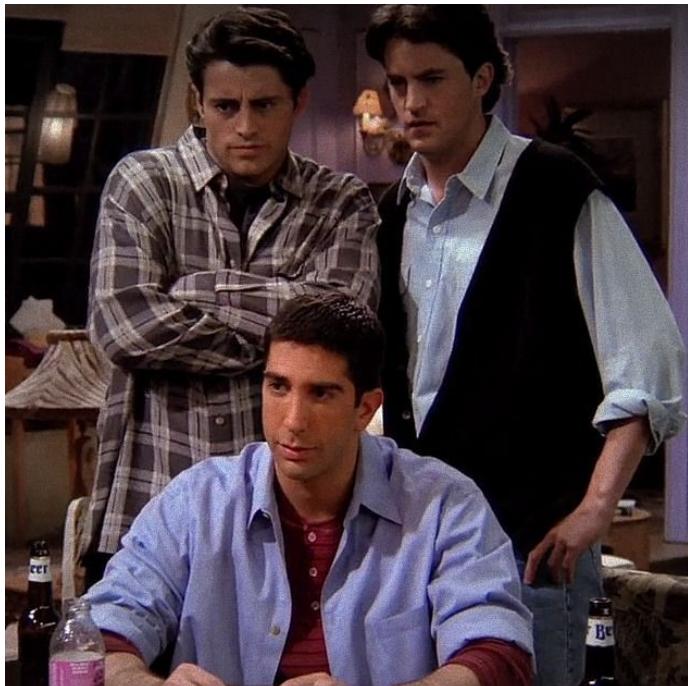


Рис. 6. Исходное изображение.

При выполнении задания используем программу, написанную при помощи средств пакета MATLAB. Преобразуем рассматриваемое изображение в черно-белое при помощи команды `im2gray` и приведем полученное изображение на рисунке 7.

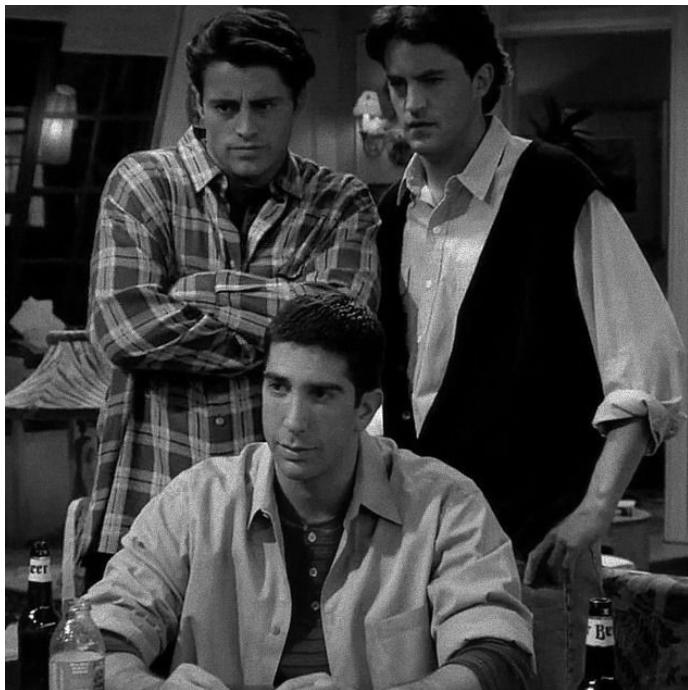


Рис. 7. Черно-белое изображение.

Дополнительно при помощи использованной ранее программы получим изображение логарифма модуля образа исходной картинки и приведем его на рисунке 8.

В этом задании необходимо сравнить два подхода к выполнению фильтрации: использование свертки изображения с ядром и применение преобразования Фурье. Предлагается сравнивать результаты обоих способов при помощи анализа полученных после преобразований изображений.

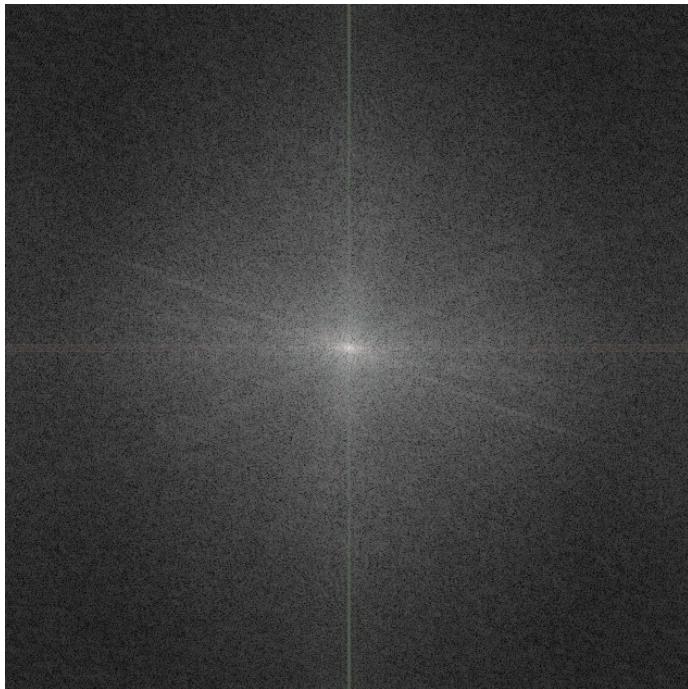


Рис. 8. Нормализованный логарифм модуля Фурье-образа.

Определим ядра, используемые для исследования свертки. Выберем множество показательных нечетных значений N для последующего конструирования ядер.

$$N = \{7, 15, 25\}. \quad (1)$$

- Ядра размытия *по Гауссу* для трех значений $N \geq 3$ из множества (1):

$$\sigma = \frac{N-1}{6}, \quad A_{ij} = \exp \left(-\frac{(i - \frac{N+1}{2})^2 + (j - \frac{N+1}{2})^2}{2\sigma^2} \right) \quad (2)$$

$$i, j \in \{1, \dots, N\}, \quad K_\sigma = \frac{A}{\sum_{i,j} A_{ij}}, \text{ где } A - \text{ вся матрица.}$$

- Ядра *блочного размытия* для тех значений N :

$$A_{ij} = 1 \quad i, j \in \{1, \dots, N\}, \quad K_{\square} = \frac{A}{\sum_{i,j} A_{ij}}. \quad (3)$$

- Ядро *увеличения резкости*:

$$K_* = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (4)$$

- Ядро *выделения краев*:

$$K_{\nabla} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (5)$$

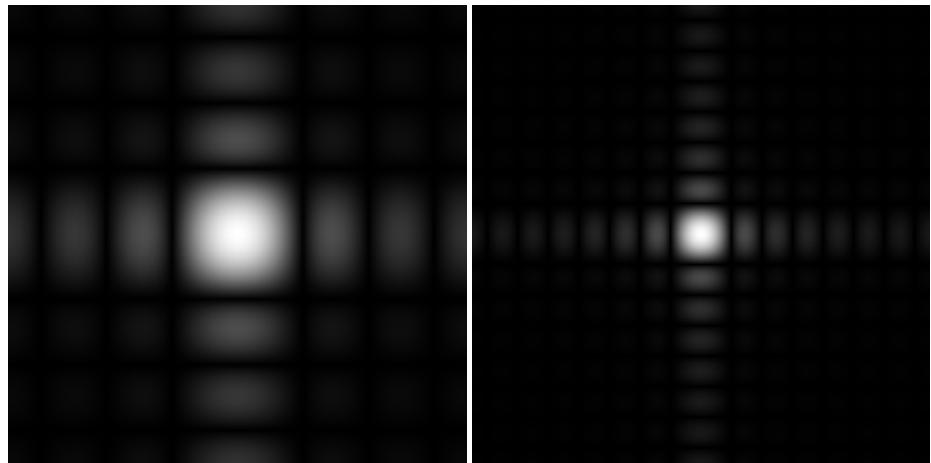
- Ядро, которое дает *интересный результат и не похоже на предыдущие*¹:

$$K = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix} \quad (6)$$

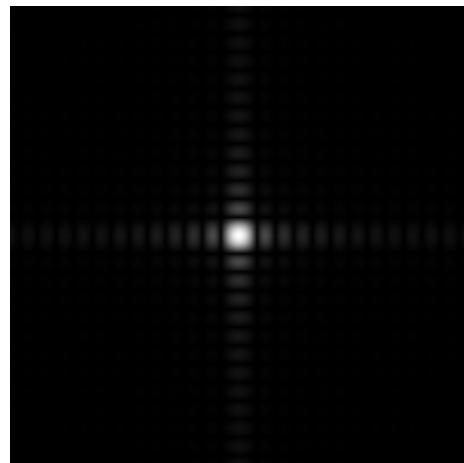
Последовательно для каждого ядра выполним свертку исходного изображения при помощи команды `conv2`. Затем найдем Фурье-образ от исходного изображения и от каждого из применяемых ядер, заполнив пропуски нулями при помощи методов из программного пакета MATLAB. Поэлементно перемножим Фурье-образ изображения с Фурье-образами каждого из ядер. Выполним обратное преобразование Фурье от полученных произведений, а после сравним изображения, полученные с помощью свертки и преобразования Фурье.

Перед выполнением задания приведем изображения логарифма модуля Фурье-образа каждого ядра, которые были получены при помощи [программы](#).

¹Выбранное из [источника](#) ядро есть ядро *тиснения*.

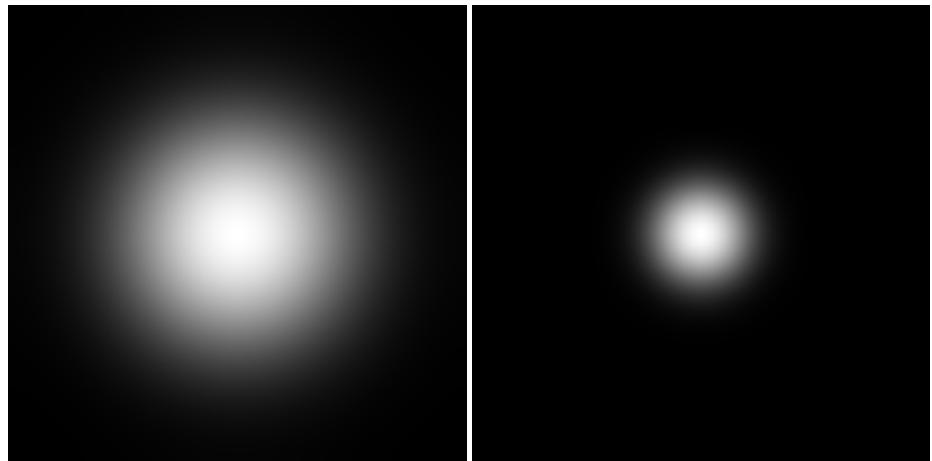


(a) Ядро блочного размытия для $N = 7$. (b) Ядро блочного размытия для $N = 15$.

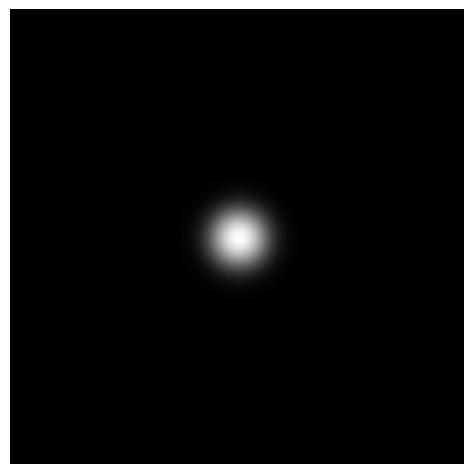


(c) Ядро блочного размытия для $N = 25$.

Рис. 9. Изображения логарифма модуля Фурье-образа ядер (3) блочного размытия.

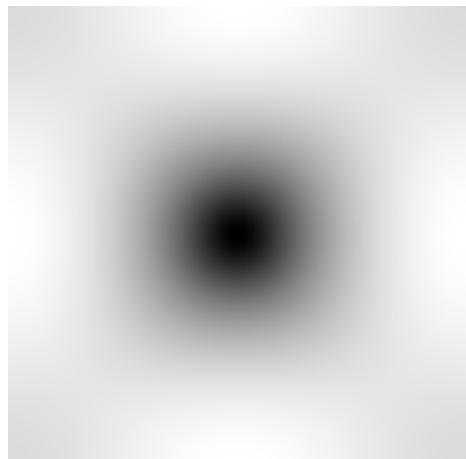


(a) Ядро размытия по Гауссу для $N = 7$. (b) Ядро размытия по Гауссу для $N = 15$.

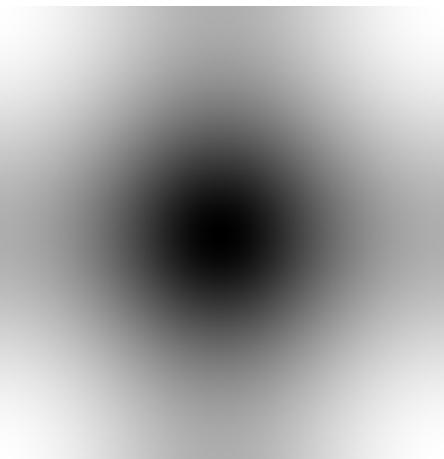


(c) Ядро размытия по Гауссу для $N = 25$.

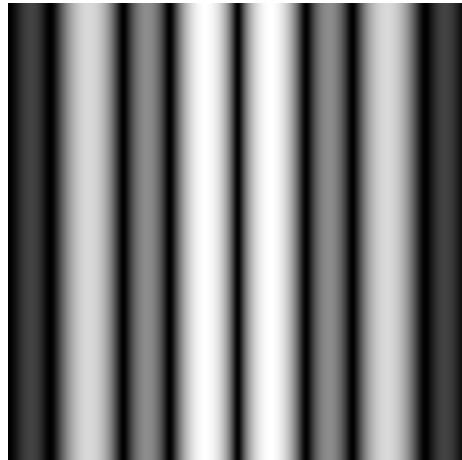
Рис. 10. Изображения логарифма модуля Фурье-образа ядер (2) размытия по Гауссу.



(a) Ядро увеличения резкости.



(b) Ядро выделения краев.



(c) Ядро тиснения.

Рис. 11. Изображения логарифма модуля Фурье-образа ядер (4), (5) и (6).



Рис. 12. Размытие по Гауссу при помощи свертки при $N = 7$.



Рис. 13. Блочное размытие при помощи свертки при $N = 7$.



Рис. 14. Размытие по Гауссу при помощи свертки при $N = 15$.



Рис. 15. Блочное размытие при помощи свертки при $N = 15$.



Рис. 16. Размытие по Гауссу при помощи свертки при $N = 25$.

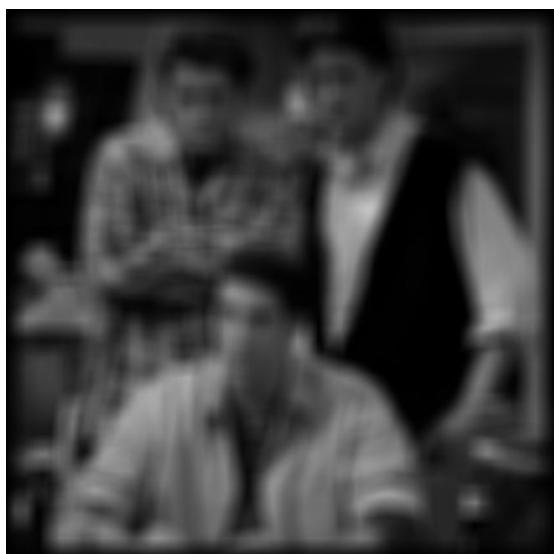


Рис. 17. Блочное размытие при помощи свертки при $N = 25$.



Рис. 18. Размытие по Гауссу при помощи преобразования Фурье при $N = 7$.



Рис. 19. Блочное размытие при помощи преобразования Фурье при $N = 7$.



Рис. 20. Размытие по Гауссу при помощи преобразования Фурье при $N = 15$.



Рис. 21. Блочное размытие при помощи преобразования Фурье при $N = 15$.



Рис. 22. Размытие по Гауссу при помощи преобразования Фурье при $N = 25$.



Рис. 23. Блочное размытие при помощи преобразования Фурье при $N = 25$.

Применим к изображению на рисунке 7 размытие по Гауссу при помощи программы, а также программным путем выполним блочное размытие. Сравним имеющиеся результаты использования свертки, иначе говоря, непосредственного применения фильтра.

При $N = 7$ изображение на рисунке 12 почти не отличается от исходного, а на рисунке 13 заметно умеренное размытие, сохраняющее лишь основные контуры.

При $N = 15$ на рисунке 14 видно плавное размытие с небольшими отличиями от исходного изображения, изображение на рисунке 15 отдаленно напоминает мозаику по структуре.

При $N = 25$ на рисунке 16 представлено сравнительно сильное размытие, сохраняющее общие формы исходной картинки, но теряющее мелкие детали. Изображение на рисунке 17 стало еще больше похоже на мозаику, размытие выражено более явно: исходная картина лишилась всех деталей, остались только основные контуры.

Опишем теперь применение преобразования Фурье, то есть использование фильтра в частотной области.

При $N = 7$ изображение на рисунке 18 совсем не отличимо от исходного, а на рисунке 19 уже остаются лишь очертания объектов исходного изображения.

При $N = 15$ на рисунке 20 заметно умеренное размытие с сохранением большинства деталей, а на рисунке 21 отчетливо видна мозаичная структура.

При $N = 25$ на рисунке 22 видно значительно размытие, которое передает главные контуры исходной картинки. На рисунке 23 утрачены детали, но формы крупных объектов различимы, а само изображение словно состоит из пикселей.

То есть, при использовании свертки при одинаковых N блочное размытие более агрессивно, чем размытие по Гауссу.

Кроме того, при сравнении подходов к фильтрации можно заметить, что размытие сильнее проявляется при использовании преобразования Фурье, чем в случае применения свертки.

Таким образом, ядро размытия по Гауссу создает плавное размытие, сохраняя естественность, тогда как блочное приводит к «мозаичному» эффекту с заметными артефактами. С увеличением N разница в качестве становится более выраженной: Гауссово размытие становится плавнее, а блочное приобретает характер мозаики.

Иначе говоря, Гауссово ядро предпочтительнее для задач, требующих естественности и плавности, а блочное ядро подходит для упрощенных вычислений, где допустимы приближенные результаты.

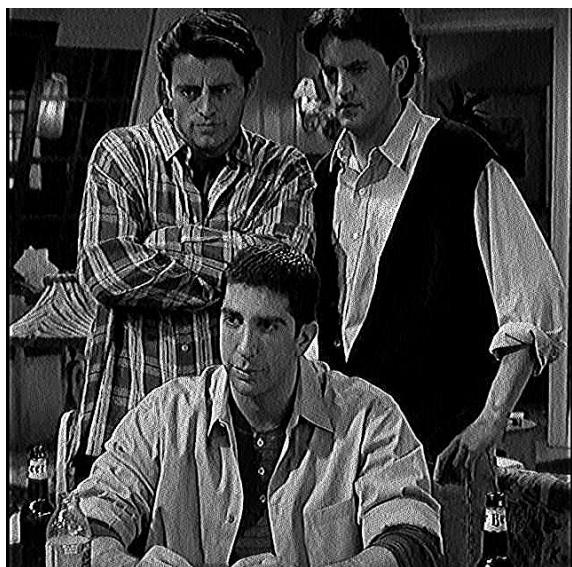


Рис. 24. Увеличение резкости при помощи свертки.

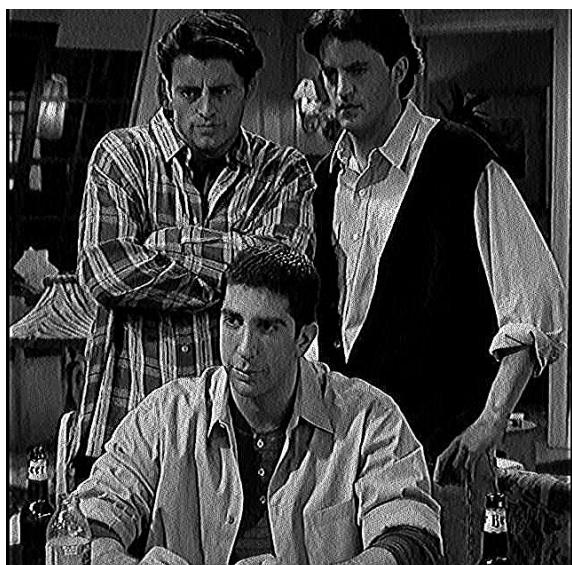


Рис. 25. Увеличение резкости при помощи преобразования Фурье.

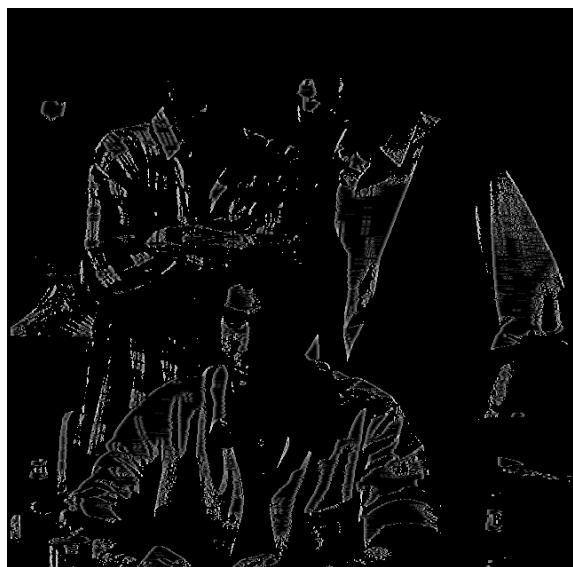


Рис. 26. Выделение краев при помощи свертки.

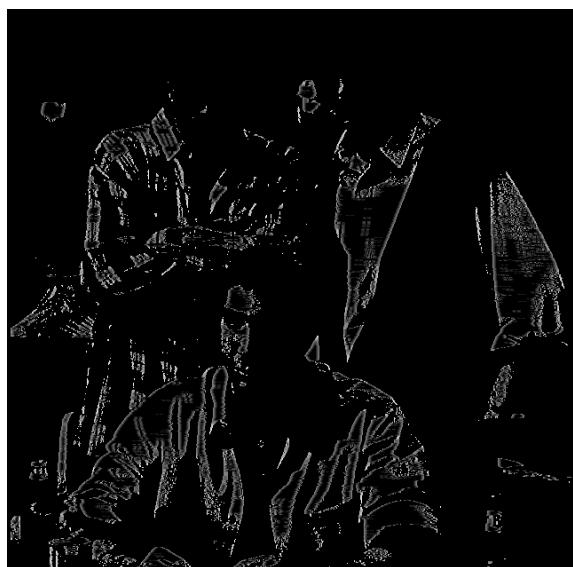


Рис. 27. Выделение краев при помощи преобразования Фурье.

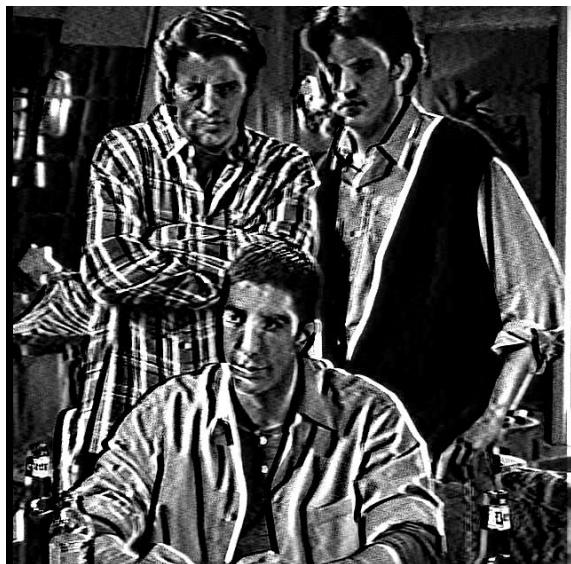


Рис. 28. Наложение эффекта тиснения при помощи свертки.

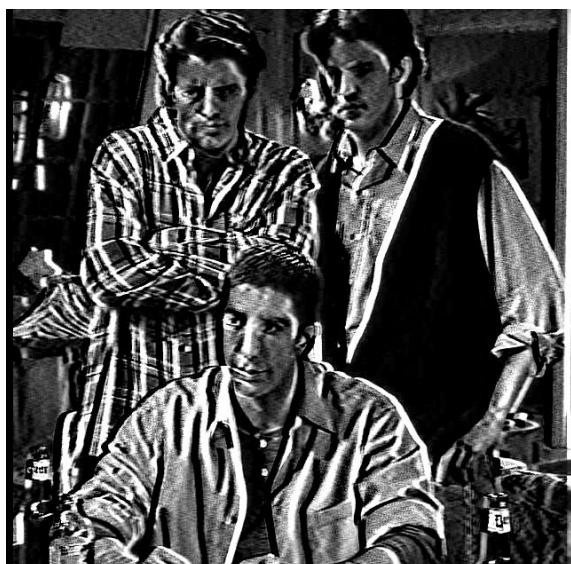


Рис. 29. Наложение эффекта тиснения при помощи преобразования Фурье.

Теперь увеличим резкость при помощи [программы](#), выделим края [программным](#) путем и наложим эффект тиснения при помощи [программы](#).

Проанализируем полученные графические результаты.

На рисунке [24](#) края объектов стали более четкими, мелкие детали были усилены. На рисунке [25](#) заметна большая разница светлого и темного, а значит, резкость возросла.

Можно отметить, что результаты использования свертки и преобразования Фурье практически неотличимы.

На рисунке [26](#) контуры объектов выделены белыми линиями на темном фоне. На рисунке [27](#) границы фигур явно выражены, изображение превращено в схематичный рисунок.

То есть, выделение краев было выполнено успешно и корректно: результаты обоих подходов идентичны.

На рисунке [28](#) приведено наложение эффекта тиснения: создается иллюзия объема за счет имитации направленного освещения. На рисунке [29](#) объекты выглядят рельефными, как при гравировке, подчеркнуты перепады яркости.

Следовательно, применение свертки практически эквивалентно использованию преобразования Фурье.

2.1 Выводы.

Сформулируем общие наблюдения по итогам выполнения данного задания.

Разные ядра свертки позволяют достигать специфических визуальных эффектов – от размытия (по Гауссу, блочное) до усиления деталей (увеличение резкости, выделение краев), демонстрируя гибкость пространственной фильтрации.

Теорема о свертке подтверждается на практике: результаты пространственной свертки и умножения в частотной области совпадают, что упрощает анализ сложных операций.

Ядра размытия подавляют высокие частоты (центр спектра), а ядра выделения краев усиливают их (периферия), что видно на изображении логарифма модуля Фурье-образа.

Свертка эффективна для малых ядер, тогда как использование преобразования Фурье предпочтительно для больших ядер или изображений за счет сокращения сложности вычислений.

Гауссово размытие широко используется в регулировании, а ядра резкости – в улучшении детализации, что подчеркивает их утилитарную ценность.

3 Выводы по лабораторной работе.

Сформулируем общие выводы, которые можно сделать на основании результатов выполнения лабораторной работы:

1. Фильтрация в частотной области при помощи преобразования Фурье эффективно устраниет периодические шумы за счет подавления соответствующих гармоник в спектре.
2. Сравнение способов размытия показало, что ядро размытия по Гауссу обеспечивает плавное сглаживание с сохранением естественности, тогда как блочное размытие создает «мозаичный» эффект с заметными квадратными артефактами.
3. Увеличение резкости усиливает детали изображения, подчеркивая контуры объектов. Результаты свертки и Фурье-фильтрации визуально неразличимы.
4. Выделение краев преобразует изображение в схематичный контурный рисунок, выделяя границы объектов. Оба подхода к фильтрации демонстрируют идентичные результаты.
5. Эффект тиснения имитирует объем за счет направленного освещения. Свертка с ядром и частотная фильтрация дают схожие результаты, подчеркивая перепады яркости.
6. С увеличением N Гауссово размытие сохраняет плавность, тогда как блочное становится более агрессивным, превращая изображение в «пикельную мозаику».
7. Пространственная свертка удобна для локальных операций с малыми ядрами, обеспечивая скорость и минимальные искажения.
8. Частотная фильтрация используется для глобальных преобразований и работы с периодическими шумами, но требует аккуратной настройки.
9. Для естественного сглаживания предпочтительно Гауссово ядро, а для задач с большими ядрами или сложными фильтрами эффективнее использовать преобразование Фурье.
10. Логарифм модуля Фурье-образа ядер помогает исследовать их влияние на исходное изображение.

Приложение

```

1 % Загрузка изображения
2 img = imread('13.png'); % Путь к изображению
3
4 % Преобразование в вещественный тип
5 img_double = double(img) / 255;
6
7 % Получение сдвинутого Фурье-образа
8 fft_img = fftshift(fft2(img_double));
9
10 % Разделение образа на массивы модулей и аргументов
11 magnitude = abs(fft_img);
12 phase = angle(fft_img);
13
14 % Логарифмирование и нормализация
15 log_magnitude = log(magnitude + 1); % Устранение неопределенности
16 log_norm = mat2gray(log_magnitude); % Нормализация в [0, 1]
17
18 % Сохранение Фурье-образа
19 imwrite(log_norm, 'fourier_spec.png');

```

Листинг 1. Программа, используемая для получения нормализованного логарифма Фурье-образа.

```

1 % Загрузка изображения
2 image2 = imread('fourier_spec_mod.png');
3
4 % Преобразование в вещественный тип
5 double_image2 = double(image2) / 255;
6 abs_2 = abs(double_image2);
7
8 % Нормализация и возвведение в степень
9 norm_coef = max(log_magnitude(:));
10 log_abs_2 = abs_2 * norm_coef;
11 rec_abs = exp(log_abs_2) - 1;
12 rec = rec_abs .* exp(1i * phase);
13
14 % Обратное преобразование
15 ifft_image2 = ifft2(ifftshift(rec));
16 ifft_image2 = real(ifft_image2);
17
18 % Сохранение фильтрованного изображения
19 imwrite(ifft_image2, '13_filtered.png');

```

Листинг 2. Программа, используемая для восстановления фильтрованного изображения.

```
1 % Загрузка и обработка изображения
2 image = imread('friends.jpeg');
3 img = im2gray(image);
4
5 % Выбор значений N
6 n_1 = 7;
7 n_2 = 15;
8 n_3 = 25;
9
10 % Вычисление значений sigma
11 sigma_1 = (n_1 - 1)/6;
12 sigma_2 = (n_2 - 1)/6;
13 sigma_3 = (n_3 - 1)/6;
14
15 % Построение ядер размытия по Гауссу
16 gkern_1 = zeros(n_1, n_1);
17 gkern_2 = zeros(n_2, n_2);
18 gkern_3 = zeros(n_3, n_3);
19
20 % Ядро размытия по Гауссу для n_1 и sigma_1
21 s1 = 0;
22 for i = 1 : n_1
23     for j = 1 : n_1
24         gkern_1(i, j) = exp(-((i - (n_1 + 1) / 2)^2 + (j - (n_1 +
25             1) / 2)^2) / (2*sigma_1^2));
26         s1 = s1 + gkern_1(i, j);
27     end
28 end
29 gkern_1 = gkern_1 / s1;
30
31 % Ядро размытия по Гауссу для n_2 и sigma_2
32 s2 = 0;
33 for i = 1 : n_2
34     for j = 1 : n_2
35         gkern_2(i, j) = exp(-((i - (n_2 + 1) / 2)^2 + (j - (n_2 +
36             1) / 2)^2) / (2*sigma_2^2));
37         s2 = s2 + gkern_2(i, j);
38     end
39 gkern_2 = gkern_2 / s2;
40
41 % Ядро размытия по Гауссу для n_3 и sigma_3
42 s3 = 0;
43 for i = 1 : n_3
44     for j = 1 : n_3
45         gkern_3(i, j) = exp(-((i - (n_3 + 1) / 2)^2 + (j - (n_3 +
46             1) / 2)^2) / (2*sigma_3^2));
47         s3 = s3 + gkern_3(i, j);
48     end
```

```

48 end
49 gkern_3 = gkern_3 / s3;
50
51 % Свертка и запись результата
52 gauss_res_1_conv = conv2(img, gkern_1 / 255);
53 imwrite(gauss_res_1_conv, 'p_n_1_conv_gauss.jpg');
54 gauss_res_2_conv = conv2(img, gkern_2 / 255);
55 imwrite(gauss_res_2_conv, 'p_n_2_conv_gauss.jpg');
56 gauss_res_3_conv = conv2(img, gkern_3 / 255);
57 imwrite(gauss_res_3_conv, 'p_n_3_conv_gauss.jpg');
58
59 % Определение размера матрицы на основе изображения
60 [row, col] = size(img);
61 gkern_1 = gkern_1 / 255;
62 gkern_2 = gkern_2 / 255;
63 gkern_3 = gkern_3 / 255;
64
65 % Фурье-образ изображения и ядра
66 img_1_f = fft2(img, row + size(gkern_1, 1) - 1, col + size(gkern_1,
    2) - 1);
67 kern_1_g = fft2(gkern_1, row + size(gkern_1, 1) - 1, col + size(
    gkern_1, 2) - 1);
68 img_2_f = fft2(img, row + size(gkern_2, 1) - 1, col + size(gkern_2,
    2) - 1);
69 kern_2_g = fft2(gkern_2, row + size(gkern_2, 1) - 1, col + size(
    gkern_2, 2) - 1);
70 img_3_f = fft2(img, row + size(gkern_3, 1) - 1, col + size(gkern_3,
    2) - 1);
71 kern_3_g = fft2(gkern_3, row + size(gkern_3, 1) - 1, col + size(
    gkern_3, 2) - 1);
72
73 % Поэлементное умножение и запись результата
74 res_g_1 = ifft2(img_1_f .* kern_1_g);
75 res_g_2 = ifft2(img_2_f .* kern_2_g);
76 res_g_3 = ifft2(img_3_f .* kern_3_g);
77 imwrite(res_g_1, 'p_n_1_fourier_gauss.jpg');
78 imwrite(res_g_2, 'p_n_2_fourier_gauss.jpg');
79 imwrite(res_g_3, 'p_n_3_fourier_gauss.jpg');

```

Листинг 3. Программа, используемая для размытия по Гауссу.

```

1 % Загрузка и обработка изображения
2 img = im2gray(imread('friends.jpeg'));
3
4 % Выбор значений N
5 n_1 = 7;
6 n_2 = 15;
7 n_3 = 25;
8
9 % Построение ядер блочного размытия;
10 bkern_1 = ones(n_1) / n_1^2;
11 bkern_2 = ones(n_2) / n_2^2;
12 bkern_3 = ones(n_3) / n_3^2;
13
14 % Свертка и запись результата
15 block_res_1_conv = conv2(img, bkern_1 / 255);
16 imwrite(block_res_1_conv, 'p_n_1_conv_block.jpg');
17 block_res_2_conv = conv2(img, bkern_2 / 255);
18 imwrite(block_res_2_conv, 'p_n_2_conv_block.jpg');
19 block_res_3_conv = conv2(img, bkern_3 / 255);
20 imwrite(block_res_3_conv, 'p_n_3_conv_block.jpg');
21
22 % Определение размера матрицы на основе изображения
23 [row, col] = size(img);
24 bkern_1 = bkern_1 / 255;
25 bkern_2 = bkern_2 / 255;
26 bkern_3 = bkern_3 / 255;
27
28 % Фурье-образ изображения и ядра
29 img_1_b = fft2(img, row + size(bkern_1, 1) - 1, col + size(bkern_1,
    2) - 1);
30 kern_1_b = fft2(bkern_1, row + size(bkern_1, 1) - 1, col + size(
    bkern_1, 2) - 1);
31 img_2_b = fft2(img, row + size(bkern_2, 1) - 1, col + size(bkern_2,
    2) - 1);
32 kern_2_b = fft2(bkern_2, row + size(bkern_2, 1) - 1, col + size(
    bkern_2, 2) - 1);
33 img_3_b = fft2(img, row + size(bkern_3, 1) - 1, col + size(bkern_3,
    2) - 1);
34 kern_3_b = fft2(bkern_3, row + size(bkern_3, 1) - 1, col + size(
    bkern_3, 2) - 1);
35
36 % Поэлементное умножение и запись результата
37 res_b_1 = ifft2(img_1_b .* kern_1_b);
38 res_b_2 = ifft2(img_2_b .* kern_2_b);
39 res_b_3 = ifft2(img_3_b .* kern_3_b);
40 imwrite(res_b_1, 'p_n_1_fourier_block.jpg');
41 imwrite(res_b_2, 'p_n_2_fourier_block.jpg');
42 imwrite(res_b_3, 'p_n_3_fourier_block.jpg');

```

Листинг 4. Программа, используемая для блочного размытия.

```

1 % Загрузка и подготовка изображения
2 image = imread('friends.jpeg');
3 img = im2gray(image);
4
5 % Ядро увеличения резкости
6 K = [[0, -1, 0], [-1, 5, -1], [0, -1, 0]];
7 K = K / 255;
8
9 % Свертка и запись результата
10 res_1 = conv2(img, K);
11 imwrite(res_1, 'sharpness_conv.jpg')
12
13 % Определение размера матрицы на основе изображения
14 [row, col] = size(img);
15
16 % Фурье-образ изображения и ядра
17 img_f = fft2(img, row + size(K, 1) - 1, col + size(K, 2) - 1);
18 kern_f = fft2(K, row + size(K, 1) - 1, col + size(K, 2) - 1);
19
20 % Поэлементное и запись результата
21 res_2 = ifft2(img_f .* kern_f);
22 imwrite(res_2, 'sharpness_fourier.jpg');

```

Листинг 5. Программа, используемая для увеличения резкости.

```

1 % Загрузка и подготовка изображения
2 img = im2gray(imread('friends.jpeg')) / 255;
3
4 % Ядро выделения краев
5 K = [[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]];
6
7 % Свертка, нормализация и запись результата
8 res_1 = conv2(img, K);
9 norm_coef = max(res_1(:));
10 res_1 = res_1 / norm_coef;
11 imwrite(res_1, 'edge_highlighting_conv.png');
12
13 % Определение размера матрицы на основе изображения
14 [row, col] = size(img);
15
16 % Фурье-образ изображения и ядра
17 img_f = fft2(img, row + size(K, 1) - 1, col + size(K, 2) - 1);
18 kern_f = fft2(K, row + size(K, 1) - 1, col + size(K, 2) - 1);
19
20 % Поэлементное умножение, нормализация и запись результата
21 res_2 = ifft2(img_f .* kern_f);
22 norm_coef = max(res_2(:));
23 res_2 = res_2 / norm_coef;
24 imwrite(res_1, 'edge_highlighting_fourier.png');

```

Листинг 6. Программа, используемая для выделения краев.

```

1 % Загрузка и подготовка изображения
2 image = imread('friends.jpeg');
3 img = im2gray(image);
4
5 % Ядро тиснения
6 K = [[-2, -1, 0], [-1, 1, 1], [0, 1, 2]];
7 K = K / 255;
8
9 % Свертка и запись результата
10 res_1 = conv2(img, K);
11 imwrite(res_1, 'optional_conv.jpg')
12
13 % Определение размера матрицы на основе изображения
14 [row, col] = size(img);
15
16 % Фурье-образ изображения и ядра
17 img_f = fft2(img, row + size(K, 1) - 1, col + size(K, 2) - 1);
18 kern_f = fft2(K, row + size(K, 1) - 1, col + size(K, 2) - 1);
19
20 % Поэлементное умножение и запись результата
21 res_2 = ifft2(img_f .* kern_f);
22 imwrite(res_2, 'optional_fourier.jpg');
```

Листинг 7. Программа, используемая для применения произвольного ядра.

```

1 % Произвольное ядро
2 kernel = [[-2, -1, 0], [-1, 1, 1], [0, 1, 2]];
3
4 % Дополнение ядра нулями для улучшения визуализации
5 padded_size = 255; % Размер спектра изображения
6 padded_kernel = padarray(kernel, [padded_size - size(kernel, 1),
7                                         padded_size - size(kernel, 2)], 'post');
8
9 % Вычисление сдвинутого Фурье-образа
10 fft_kernel = fftshift(fft2(padded_kernel));
11
12 % Модуль Фурье-образа и логарифм
13 magnitude = abs(fft_kernel);
14 log_magnitude = log(magnitude + 1);
15
16 % Нормализация в диапазон [0, 1]
17 log_norm = mat2gray(log_magnitude);
18
19 % Сохранение изображения
20 imwrite(log_norm, 'optional_kernel.png');
```

Листинг 8. Программа, используемая для построения изображения логарифма модуля образа ядра.