



Федеральное государственное автономное образовательное учреждение высшего образования «Национальный Исследовательский Университет ИТМО»

**ЛАБОРАТОРНАЯ РАБОТА №6
ПРЕДМЕТ «ЧАСТОТНЫЕ МЕТОДЫ»
ТЕМА «ОБРАБОТКА ИЗОБРАЖЕНИЙ»**

Лектор: Перегудин А. А.
Практик: Пашенко А. В.
Студент: Румянцев А. А.
Поток: ЧАСТ.МЕТ. 1.3

Факультет: СУиР
Группа: R3241

Санкт-Петербург
2024

Содержание

1 Задание 1. Фильтрация изображений с периодичностью	2
1.1 Листинг для задания 1	4
2 Задание 2. Размытие изображения	7

1 Задание 1. Фильтрация изображений с периодичностью

Скачаем изображение под номером 4 с данного [гугл-диска](#).



Рис. 1: Изображение, выбранное с представленного гугл-диска

Далее проделаем следующие шаги:

1. Загрузим это изображение в `python` с помощью библиотеки `pillow`.
2. Преобразуем полученный массив к вещественному типу с помощью инструментов библиотеки `numpy`. Поделим все значения на 255 – максимальное значение яркости цвета.
3. Найдем двумерный Фурье-образ массива и сдвинем его в центр с помощью команд `fftshift(fft2(my_image))`. Так как изображение цветное, выполним преобразование для каждого из цветовых каналов по отдельности.
4. Разделим полученные образы каждого канала на массивы модулей и аргументов с помощью функций библиотеки `numpy abs` и `angle`.
5. Для удобства работы, найдем логарифм от каждого массива модулей и нормализуем его значения в диапазон от 0 до 1. Чтобы избежать неопределённости в логарифме, предварительно прибавим ко всем значениям 1. Для этого понадобится команда `pr.log1p()`. Для обратной операции `pr.expm1()`.
6. Объединим преобразованные каналы. Сохраним полученный массив (нормализованный логарифм модуля Фурье-образа) как изображение командой `save`, вызываемой от объекта `Image`.
7. Проанализируем полученное на предыдущем шаге изображение. Найдем пики, соответствующие периодичности на исходной картинке.
8. В программе редактирования изображений исправим полученный Фурье-образ: сгладим все ненужные цветовые пики, отвечающие за гармоники, от которых мы хотим избавиться.

9. Восстановим картинку из отредактированного образа, проделав обратные шаги.

Цветной нормализованный логарифм модуля Фурье-образа данного изображения представлен на следующем рисунке.

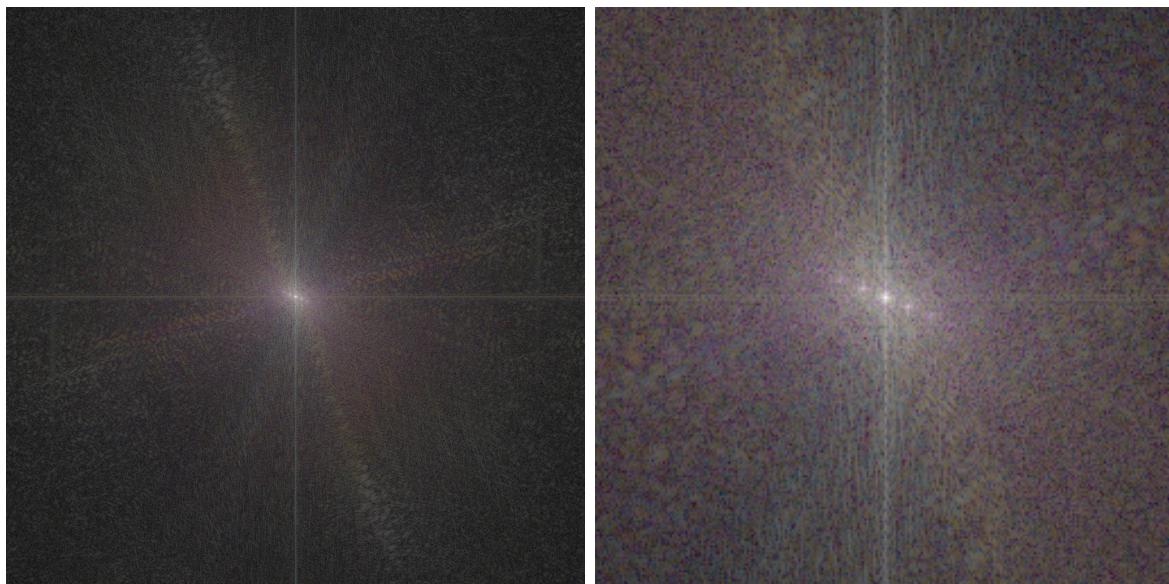


Рис. 2: Нормализованный логарифм модуля Фурье-образа

Видим на рис. 2b около центра цветовые пики во второй и четвертой четвертях системы координат, если задать ее в центре изображения. Нам необходимо сгладить эти цветовые пики, отвечающие за синусоидальный шум на картинке. Сделаем это в редакторе изображений `paint` – наложим «пластиры» из наиболее близких к пикам незасветленных областей с пикселями на места гармонического шума. Результат исправления расположен ниже.

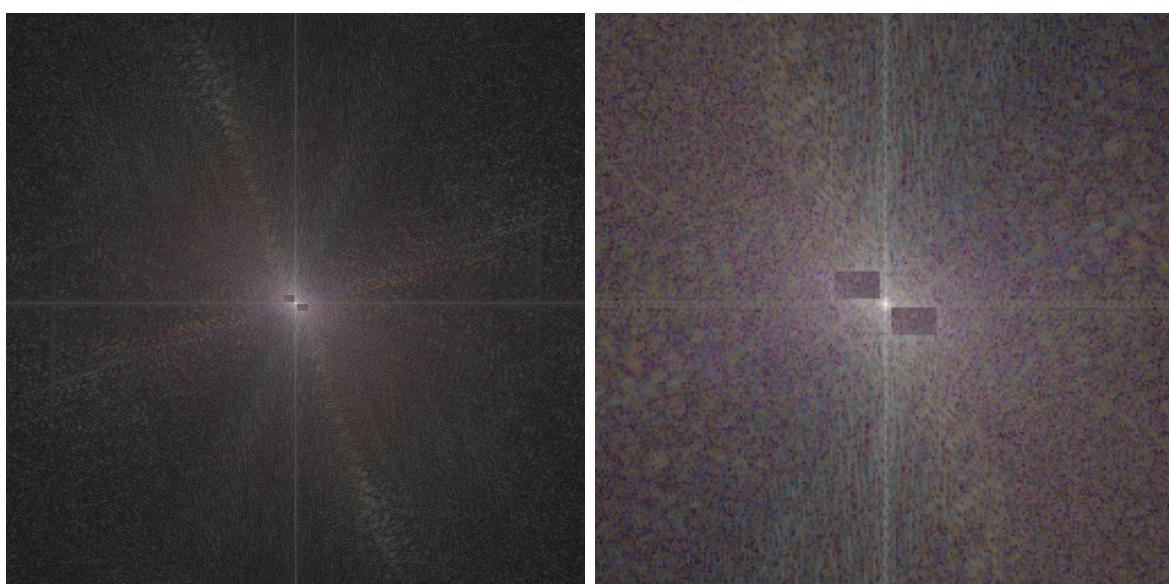


Рис. 3: Исправленный нормализованный логарифм модуля Фурье-образа

Ниже представлен сравнительный рисунок – оригинальное изображение и фильтрованное. Изображение без синусоидального шума (или с минимальным) выглядит более «мягко». Море и небо стали более естественными.



Рис. 4: Сравнение исходного и фильтрованного изображений

Для восстановления изображения понадобилась формула ниже:

$$\hat{X} = |\hat{X}|_{new} \cdot e^{i\angle\hat{X}},$$

где \hat{X} – Фурье-образ изображения, $|\hat{X}|_{new}$ фильтрованный массив модулей, $e^{i\angle\hat{X}}$ комплексное число с аргументом $\angle\hat{X}$, представляющим массив углов Фурье-образа. $|\hat{X}|_{new}$ отвечает за новые амплитуды частотных компонент, а $e^{i\angle\hat{X}}$ за фазы исходных амплитуд. Без массива углов мы бы не смогли привести изображение к изначальной «структуре».

1.1 Листинг для задания 1

Для реализации программ, необходимых для выполнения задания 1, были использованы язык программирования `python` с библиотеками `pillow` и `numpy`.

```
from PIL import Image
import numpy as np

# Method to read image along the specified path
def read_img(path: str):
    return Image.open(path)

# Method to save image along the specified path
def save_img(img: Image, path: str):
    img.save(path)

# Method to convert an array to an image
def convert_arr_to_img(arr):
    return Image.fromarray(arr)

# Method to normalize data (ex. logarithm of abs(image))
```

```
# Returns min and max to inverse the normalization
def normalize(data):
    min_ = np.min(data)
    max_ = np.max(data)
    nz = (data - min_) / (max_ - min_)
    return nz, min_, max_

# Method to remove normalization from data
# Requires min and max to inverse normalization
def denormalize(nz, min_, max_):
    return nz * (max_ - min_) + min_

# Method to split rgb image array into 3 channels
def rgb_channels(img: Image):
    img_arr = np.array(img)
    r = img_arr[:, :, 0]
    g = img_arr[:, :, 1]
    b = img_arr[:, :, 2]
    return r, g, b

# Method to split rgb image angles array into 3 for each channel
def rgb_angles(angles):
    ar = angles[:, :, 0]
    ag = angles[:, :, 1]
    ab = angles[:, :, 2]
    return ar, ag, ab

# Method to apply fft2 to a channel
# Uses special algorithm for convenience
def fft2_channel(channel):
    channel = channel.astype(np.float64) / 255
    fft_ = np.fft.fftshift(np.fft.fft2(channel))
    abs_ = np.abs(fft_)
    angle = np.angle(fft_)
    log = np.log1p(abs_)
    nz, min_, max_ = normalize(log)
    return nz, min_, max_, angle

# Method to apply ifft2 to a channel
# Inverts all the steps of the special algorithm
def ifft2_channel(channel, angle, min_, max_):
    channel = channel.astype(np.float64) / 255
    denor = denormalize(channel, min_, max_)
    abs_ = np.expm1(denor)
    col_res = abs_ * np.exp(1j * angle)
    ifft_ = np.fft.ifft2(np.fft.ifftshift(col_res))
    return np.real(ifft_)

# Method to apply fft2 to an image. Processes three channels at once
def fft2(img: Image):
    r, g, b = rgb_channels(img)

    nzr, min_r, max_r, ar = fft2_channel(r)
    nzg, min_g, max_g, ag = fft2_channel(g)
    nzb, min_b, max_b, ab = fft2_channel(b)

    res = np.stack((nzr, nzg, nzb), axis=2)
    res = (res * 255).astype(np.uint8)

    ang = np.stack((ar, ag, ab), axis=2)
```

```

nz_min_max = [(min_r, max_r), (min_g, max_g), (min_b, max_b)]
return res, ang, nz_min_max

# Method to apply ifft2 to an image. Processes three channels at once
def ifft2(img: Image, angles, nz_min_max):
    r, g, b = rgb_channels(img)
    ar, ag, ab = rgb_angles(angles)

    nz_r_min, nz_r_max = nz_min_max[0][0], nz_min_max[0][1]
    nz_g_min, nz_g_max = nz_min_max[1][0], nz_min_max[1][1]
    nz_b_min, nz_b_max = nz_min_max[2][0], nz_min_max[2][1]

    new_r = ifft2_channel(r, ar, nz_r_min, nz_r_max)
    new_g = ifft2_channel(g, ag, nz_g_min, nz_g_max)
    new_b = ifft2_channel(b, ab, nz_b_min, nz_b_max)

    res = np.stack((new_r, new_g, new_b), axis=2)
    res = normalize(res)[0] * 255
    return res.astype(np.uint8)

# Helper method for external use of fft2
def fft2_2img(img: Image):
    res, ang, nz_min_max = fft2(img)
    return convert_arr_to_img(res), ang, nz_min_max

# Helper method for external use of ifft2
def ifft2_2img(img: Image, angles, nz_min_max):
    return convert_arr_to_img(ifft2(img, angles, nz_min_max))

```

Листинг 1: Программные методы, необходимые для задания 1

Все вышеперечисленные наработки используются в программе, представленной ниже.

```

import img_utils as iu

# Specify the image source
src = 'fm_lab6/src'
img_path = f'{src}/4.png'

# Specify where to render the result
render_to = 'fm_lab6/renderers/task1'
rimg_path = f'{render_to}/fft2.png'
reimg_path = f'{render_to}/new4.png'

# Specify the corrected image source
# Parameter "corrected" if that source does exist
corr_im_path = f'{src}/corr_fft2.png'
corrected = True

# Reading image, applying fft2 and saving the result
img = iu.read_img(img_path)
ans, ang, nzmm = iu.fft2_2img(img)
iu.save_img(ans, rimg_path)

# Reading corr. im., applying ifft2 to recover im. and saving the res.
if corrected:
    img2 = iu.read_img(corr_im_path)
    ans2 = iu.ifft2_2img(img2, ang, nzmm)
    iu.save_img(ans2, reimg_path)

```

Листинг 2: Реализация задания 1

2 Задание 2. Размытие изображения