МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ»
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806: «Вычислительная математика и программирование»

ОТЧЕТ О ВЫПОЛНЕНИИ ЛАБОРАТОРНОЙ РАБОТЫ №3

По курсу: «Цифровая обработка изображений»

Тема: «Фильтрация изображений в пространственной области»

Студент: Чернышев Д.В.
Группа: М8О-107М-22
Вариант: 8
Преподаватель: Гаврилов К.Ю.

Москва
2023

СОДЕРЖАНИЕ

1 Задание к лабораторной работе №3

Дано изображение $Img3\_08\_1.jpg$.

Часть 1

Исследование пространственных фильтров для подавления шумов на изображениях

1. Моделирование искажения изображений с помощью шумов. Проведите моделирование искажения изображения путем добавления следующих шумов: гауссовский шум, шум типа «соль и перец».
2. Составьте программу, выполняющую подавление шумов с помощью масочных фильтров следующего вида:
   - среднеарифметический;
   - среднегеометрический;
   - среднегармонический
   - медианный.
3. Проведите исследование эффективности подавления шумов обоих видов (гауссовского и «соль и перец») при различных уровнях зашумления и при использовании фильтров различного вида.

Часть 2

Использование пространственных фильтров для повышения резкости изображения.

1. Использование Лапласиана. Используя расфокусировку с помощью гауссовского фильтра получите слегка размытое изображение. Используя Лапласиан улучшите резкость изображения. Путем анализа некоторых деталей изображения (букв, символов, мелких деталей) сделайте выводы о степени улучшения изображения в результате использования Лапласиана (например, стали ли читаться буквы, насколько лучше видны отдельные детали и т.д.). Используйте две функции Matlab: $imfilter$ с маской Лапласиана и $locallapfilt$ (в

моем случае аналоги в *Python*). Сравните результаты улучшения изображения.

2. Использование маски Собела. Используя такую же расфокусировку заданного изображения, что и в п.1, улучшите резкость с помощью масочного фильтра Собела.
3. Сравните эффективность работы двух алгоритмов улучшения резкости изображения: Лапласиана и оператора Собела.

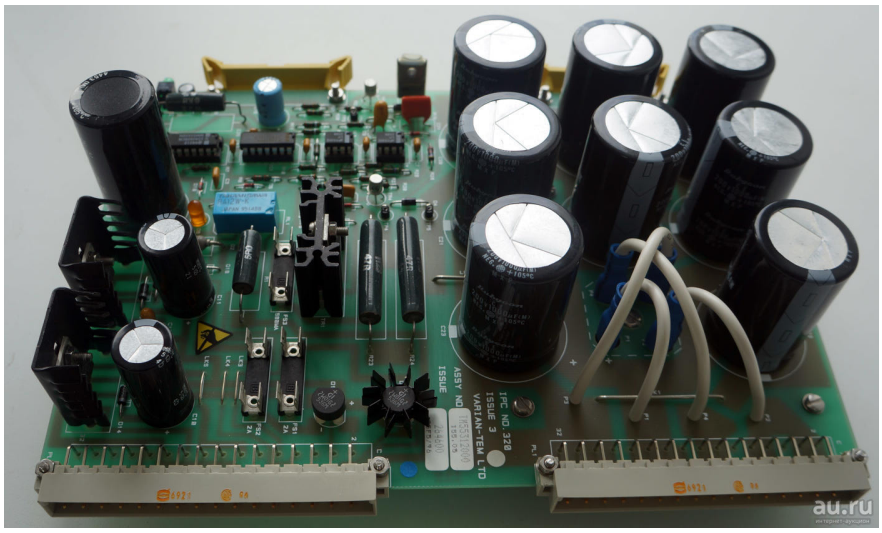


Рис. 1: Исходное изображения для обработки

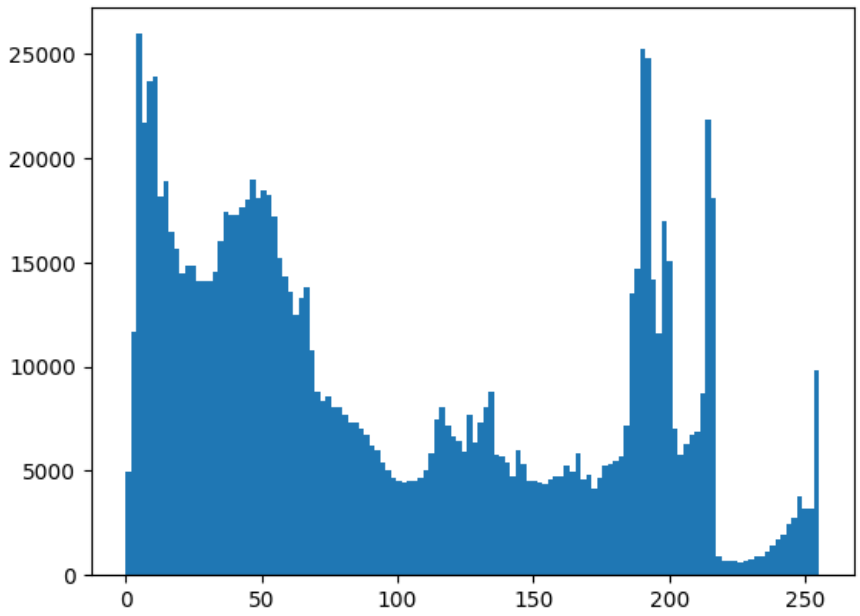


Рис. 2: Исходное изображения для обработки (гистограмма)

2 Выполнение лабораторной работы №3

Часть 1

Производим импорт необходимых библиотек которые нам понадобятся

```
import numpy as np
import PIL
import os
import cv2
import bisect
import matplotlib.image as mpimg
from skimage.util import random_noise
from scipy.ndimage import gaussian_filter
from numba import jit

from PIL import Image, ImageDraw, ImageStat
import random
import matplotlib.pyplot as plt

# from scipy import ndimage, datasets
import scipy
import scipy.stats
import random

from PIL import Image
from PIL import ImageFilter,ImageEnhance
from PIL import ImageChops
from scipy.ndimage import convolve
```

Далее определим функции зашумления фотографий

```
def sp_noise(image,prob:float=0.05):
    '''
    Add salt and pepper noise to image
```

```
    prob: Probability of the noise
    '''
    assert 0 <= prob <= 1, "Probability of the noise
    must be less
    or equal to 1 or more or equal to 0"
    output = np.zeros(image.shape,np.uint8)
    thres = 1 - prob
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            rdn = random.random()
            if rdn < prob:
                output[i][j] = 0
            elif rdn > thres:
                output[i][j] = 255
            else:
                output[i][j] = image[i][j]
    return output

def noisy(
        noise_typ:str,
        image:np.ndarray,
        prob:float=0.05) -> np.ndarray:
    """_summary_
    Args:
        noise_typ : str -  One of the following strings,
        selecting the type of noise to add:
        (*)  'gauss'     Gaussian-distributed additive noise.
        (*)  'poisson'   Poisson-distributed
        noise generated from the data.
        (*)  's_p'       Replaces random pixels
        with 0 or 1.
        (*)  'speckle'   Multiplicative noise using
        out = image + n*image, where  n is uniform noise
```

```
        with specified mean & variance.
        image : ndarray - Input image data.
        Will be converted to float.
    Returns:
        noisy: ndarray  - noised image
    """

    assert 0 <= prob <= 1, "Probability of the noise
    must be less or equal to 1 or more or equal to 0"
    if noise_typ == "gauss":
        row,col,ch= image.shape

        mean = 255/2
        var = 0.1
        sigma = var ** 0.5

        gauss = np.random.normal(mean,sigma,(row,col,ch))
        gauss = gauss.reshape(row,col,ch)
        noisy = image + gauss
        return noisy

    elif noise_typ == "s_p":
        return sp_noise(image,prob)

    elif noise_typ == "poisson":
        vals = len(np.unique(image))
        vals = 2 ** np.ceil(np.log2(vals))
        noisy = np.random.poisson(image * vals) / float(vals)
        return noisy

    elif noise_typ =="speckle":
        row,col,ch = image.shape
        gauss = np.random.randn(row,col,ch)
```

```
70          gauss = gauss.reshape(row,col,ch)
71          noisy = image + image * gauss
72          return noisy
```

Применим гауссовский шум и шум тип "соль и перец"к исходному изображения

Получим:

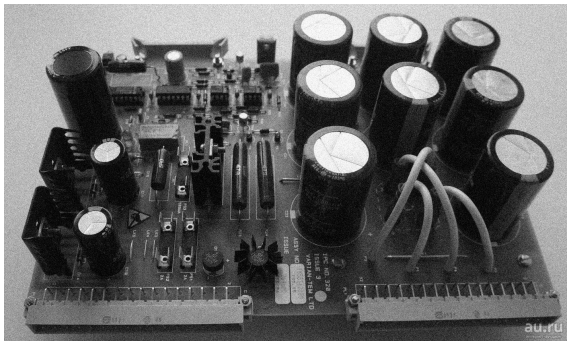


Рис. 3: Применение гауссовского шума

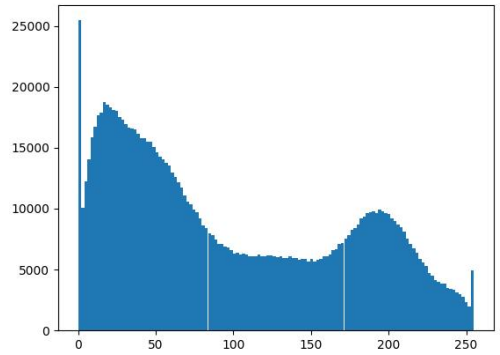


Рис. 4: Применение гауссовского шума (гистограмма)

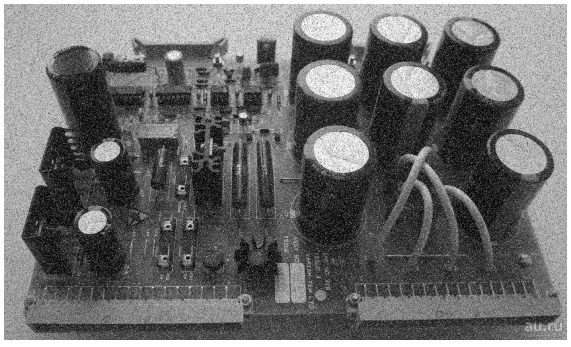


Рис. 5: Применение S&P шума

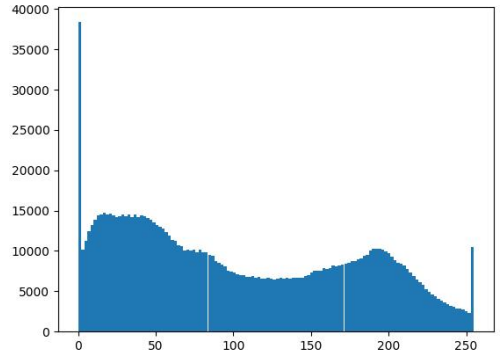


Рис. 6: Применение S&P шума (гистограмма)

Необходимо определить функции фильтров

```
def arithmetic_mean_filter(image, mask_size):
    height, width = image.shape
    result = np.zeros((height, width), dtype=np.uint8)

    half_mask_size = mask_size // 2

    for i in range(half_mask_size, height - half_mask_size):
        for j in range(half_mask_size, width - half_mask_size):
            neighbors = image[
                i - half_mask_size:i + half_mask_size + 1,
                j - half_mask_size:j + half_mask_size + 1
                ]
            result[i, j] = np.mean(neighbors)

    return result

def geometric_mean_filter(image, mask_size):
    height, width = image.shape
    result = np.zeros((height, width), dtype=np.uint8)

    half_mask_size = mask_size // 2

    for i in range(half_mask_size, height - half_mask_size):
     for j in range(half_mask_size, width - half_mask_size):
       neighbors = image[
            i - half_mask_size:i + half_mask_size + 1,
            j - half_mask_size:j + half_mask_size + 1]
        result[i, j] = np.prod(neighbors) **
                    (1.0 / (mask_size * mask_size))

    return result

```

```
def harmonic_mean_filter(image, mask_size):
    height, width = image.shape
    result = np.zeros((height, width), dtype=np.uint8)

    half_mask_size = mask_size // 2

    for i in range(half_mask_size, height - half_mask_size):
        for j in range(half_mask_size, width - half_mask_size):
            neighbors = image[
            i - half_mask_size:i + half_mask_size + 1,
            j - half_mask_size:j + half_mask_size + 1
            ]
            result[i, j] = (mask_size * mask_size) /
                        np.sum(1.0 / (neighbors + 1e-8))

    return result

def median_filter(image, mask_size):
    height, width = image.shape
    result = np.zeros((height, width), dtype=np.uint8)

    half_mask_size = mask_size // 2

    for i in range(half_mask_size, height - half_mask_size):
        for j in range(half_mask_size, width - half_mask_size):
            neighbors = image[
                i - half_mask_size:i + half_mask_size + 1,
                j - half_mask_size:j + half_mask_size + 1
                ]
            result[i, j] = np.median(neighbors)

    return result
```

Применяем фильтры к двум зашумленным изображениям

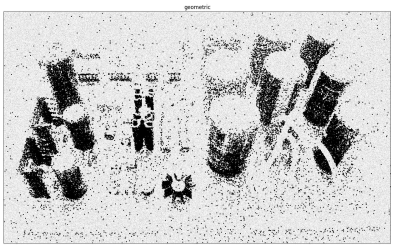
S&P

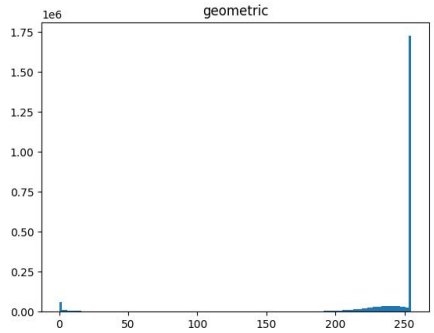
```
sp_image = cv2.imread(SP_PATH_FILE, cv2.IMREAD_GRAYSCALE)

mask_size = 3

sp_filtered = dict()
sp_filtered_PIL = dict()
sp_filtered_path = dict()
sp_filtered_hist_path = dict()

# Среднегеометрический фильтр
sp_filtered["geometric"] = geometric_mean_filter(
    sp_image,
    mask_size
    )
# Среднееарифмитический фильтр
sp_filtered["arithmetic"] = arithmetic_mean_filter(
    sp_image,
    mask_size
    )
# Среднегармонический фильтр
sp_filtered["harmonic"] = harmonic_mean_filter(
    sp_image,
    mask_size
    )
# Медианный  фильтр
sp_filtered["median"] = median_filter(
    sp_image,
    mask_size
    )
```

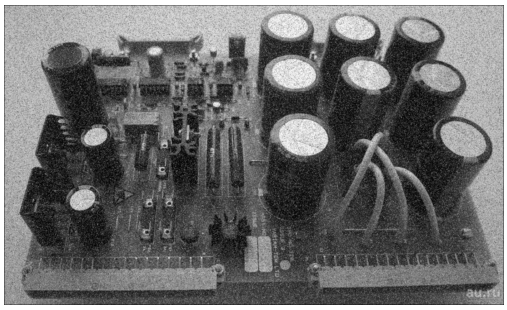
Gauss

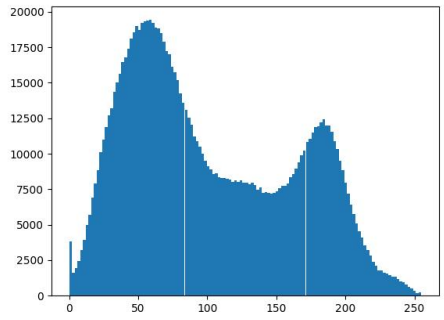
```
gauss_image = cv2.imread(GAUSS_PATH_FILE, cv2.IMREAD_GRAYSCALE)
mask_size = 3

gauss_filtered = dict()
gauss_filtered_PIL = dict()
gauss_filtered_path = dict()
gauss_filtered_hist_path = dict()

gauss_filtered["geometric"] = geometric_mean_filter(
    gauss_image,
    mask_size
    )
# Среднееарифмитический фильтр
gauss_filtered["arithmetic"] = arithmetic_mean_filter(
    gauss_image,
    mask_size
    )
# Среднегармонический фильтр
gauss_filtered["harmonic"] = harmonic_mean_filter(
    gauss_image,
    mask_size
    )
# Медианный  фильтр
gauss_filtered["median"] = median_filter(
    gauss_image,
    mask_size
    )
```

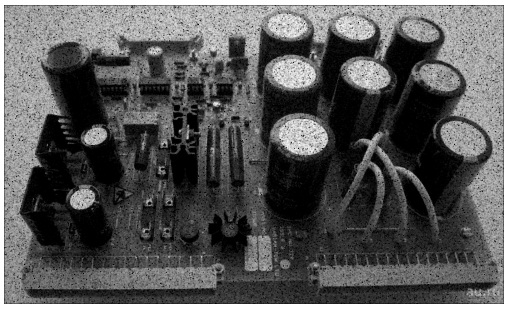
Таблица 1: Фильтрация S&P-зашумленного изображения разными фильтрами

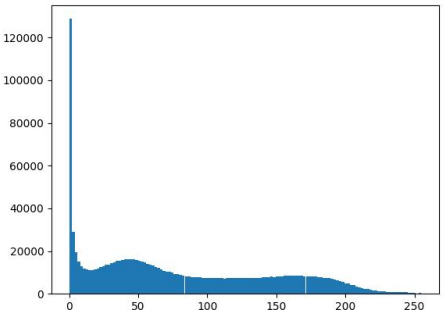


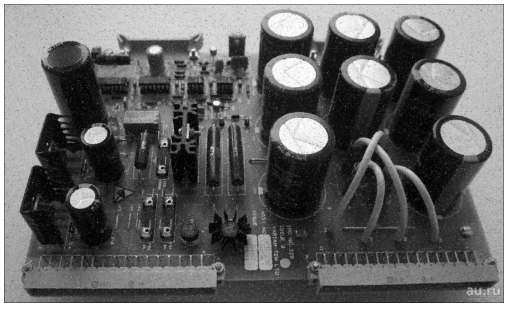












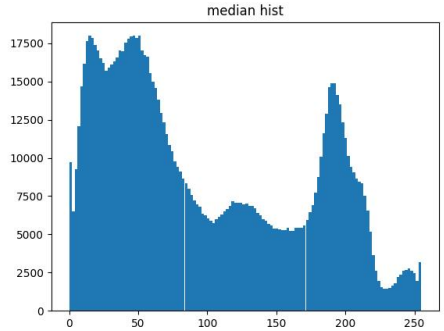
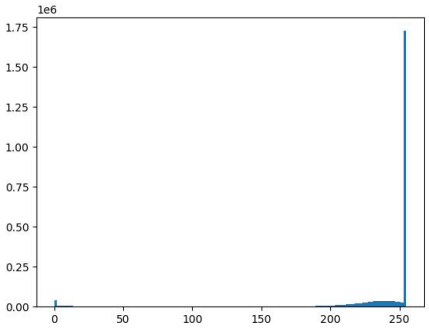
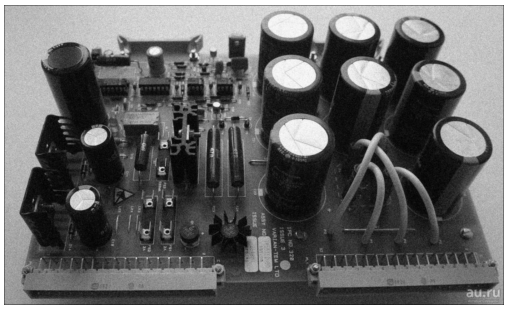
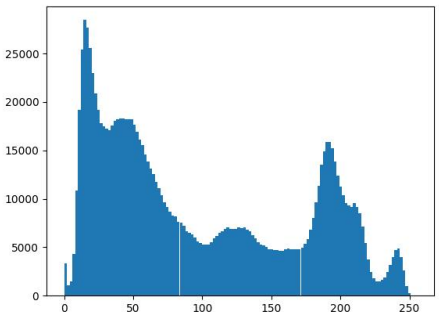
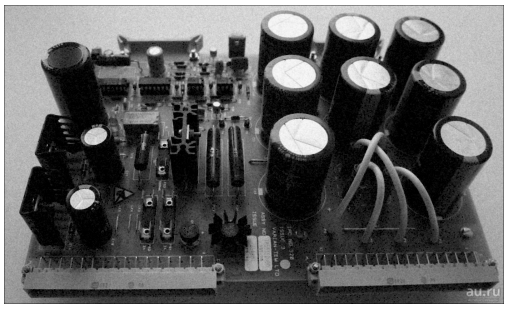


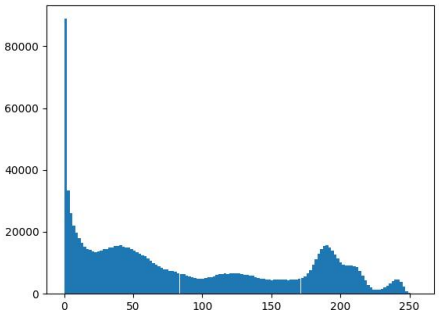
Таблица 2: Фильтрация гауссово-зашумленного изображения разными фильтрами

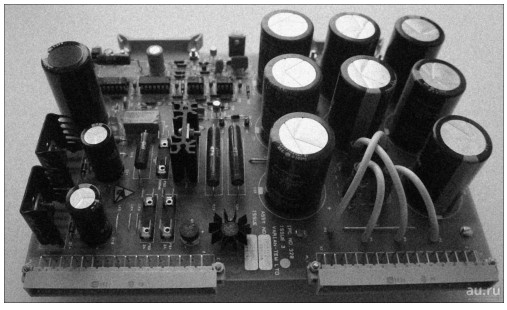


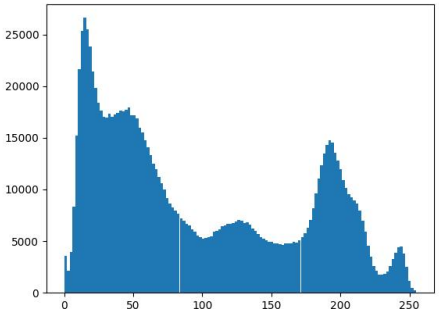












Часть 2

Применяем гауссовский фильтра для расфокусировки. Для начала зададим функции применяющие маску Лапласиана и Собеля

```
def laplacian_sharpen(image,A:int=4):
    # Convert the image to a numpy array
    image_array = np.array(image, dtype=np.float32)
    # Define the Laplacian kernel
    laplacian_kernel = np.array(
        [
            [1, 1, 1],
            [1, -A, 1],
            [1, 1, 1]
            ]
        )
    # Apply the Laplacian filter
    filtered_array = convolve(
        image_array,
        laplacian_kernel
        )
    # Normalize the filtered array
    filtered_array = (filtered_array - np.min(filtered_array)) \
                / (
                    np.max(filtered_array) - np.min(filtered_array)
                    )
    # Convert the filtered array back to image
    sharpened_image = Image.fromarray(
        (filtered_array * 255).astype(np.uint8)
        )
    # Show the result image
    return sharpened_image

def sobel_sharpen(image):
```

```
    # Apply the Sobel operator
    sobel_x = image.filter(ImageFilter.FIND_EDGES)
    sobel_y = image.transpose(Image.TRANSPOSE).filter(
        ImageFilter.FIND_EDGES
        ).transpose(Image.TRANSPOSE)
    # Combine the horizontal and vertical edges
    sobel_combined = ImageChops.add(
        sobel_x,
        sobel_y
        )
    # Enhance the edges for sharpening effect
    enhanced_image = ImageEnhance.Contrast(sobel_combined).enhance(2.0
    # Blend the enhanced edges with the original image
    sharpened_image = Image.blend(image, enhanced_image, 0.5)
    return sharpened_image
```

```
img = cv2.imread(SOURCE_PIC)
noise_img_sp = gaussian_filter(img, sigma=3)
im = Image.fromarray((noise_img_sp * 255).astype(np.uint8))
Gauss_PATH_FILE = os.path.join(
     OUTPUT_DIR_2 ,
     f'Gauss_{SOURCE_PIC}'
     )
im.save(Gauss_PATH_FILE)
plt.imshow(noise_img_sp)
```

Применяем

```
# convolute with proper kernels
A = 8
laplacian = laplacian_sharpen(Image.fromarray(noise_img_sp),A=A)
sobel = sobel_sharpen(Image.fromarray(noise_img_sp))
```

Результаты:

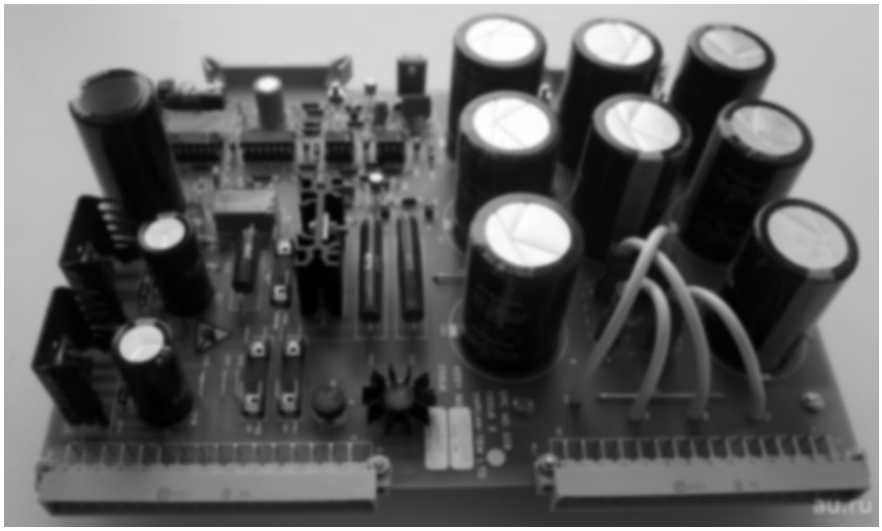


Рис. 7: Расфокусированное изображение

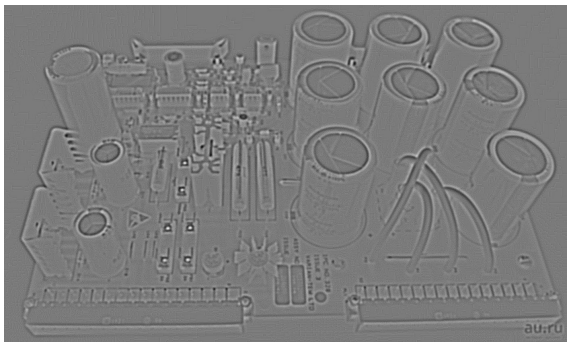


Рис. 8: Применение Лапласиана

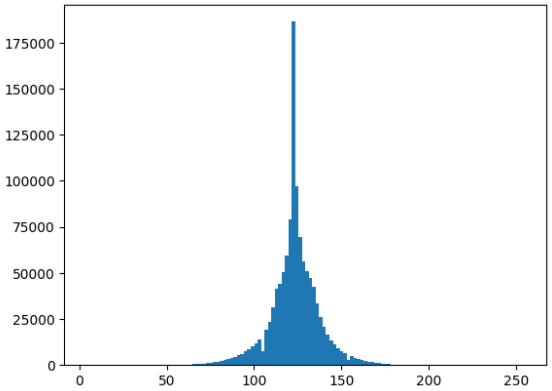


Рис. 9: Применение Лапласиана (гистограмма)

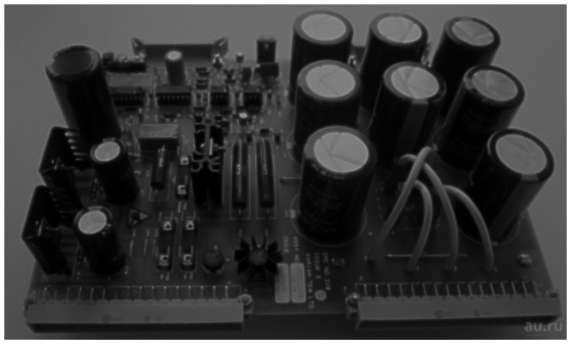


Рис. 10: Применение *Sobel X*

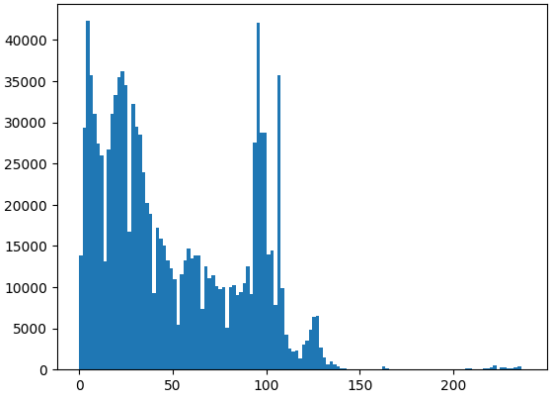


Рис. 11: Применение *Sobel X* (гистограмма)

3 Выводы

По результатам выполнения лабораторной работы №3 было продемострировано действие различных фильтров на зашумленные изображения, получены и проанализированы диаграммы исходных и отфильтрованных изображений.