Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления
Кафедра интеллектуальных информационных технологий
Дисциплина «Обработка изображений в интеллектуальных системах»

ОТЧЁТ
к лабораторной работе №2
на тему
«ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА ИЗОБРАЖЕНИЙ»

БГУИР 6-05-0611-03 130

Выполнил студент группы 321701
СЕМЕНЯКО Владимир Дмитриевич

_____
(дата, подпись студента)

Проверил
САЛЬНИКОВ Даниил Андреевич

_____
(дата, подпись преподавателя)

Минск 2025

1 ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

Реализовать один из фильтров на любом языке программирования (к примеру медианный фильтр) или же используя сторонние библиотеки реализовать программу, которая может применять различные фильтры к исходному изображению.

2 ВЫПОЛНЕНИЕ РАБОТЫ

Программа была реализована на языке Python с использованием cmath, math и matplotlib.pyplot для визуализации результатов.

Листинг 1 – Код программы

```
import io
import os
from dataclasses import dataclass
from typing import Callable, Optional, Tuple
import argparse
import traceback

from PIL import Image, ImageEnhance, ImageFilter, ImageOps

try:
        from pillow_heif import register_heif_opener
        register_heif_opener()
except Exception:
        pass

try:
        import tkinter as tk
        from tkinter import filedialog, messagebox
        exists_tk = True
except Exception:
        exists_tk = False

def apply_median_filter(image: Image.Image, kernel_size: int = 3) -> Image.
    Image:

        if kernel_size % 2 == 0 or kernel_size < 1:
                raise ValueError(”kernel_size must be an odd positive integer
    ”)

        original_mode = image.mode
        work_img = image.convert(”RGB”)
        width, height = work_img.size
        px = work_img.load()

        pad = kernel_size // 2
        result = Image.new(”RGB”, (width, height))
```

```
        res_px = result.load()

        for y in range(height):
                for x in range(width):
                        neighbors_r = []
                        neighbors_g = []
                        neighbors_b = []
                        for ky in range(-pad, pad + 1):
                                py = min(max(y + ky, 0), height - 1)
                                for kx in range(-pad, pad + 1):
                                        pxx = min(max(x + kx, 0), width - 1)
                                        r, g, b = px[pxx, py]
                                        neighbors_r.append(r)
                                        neighbors_g.append(g)
                                        neighbors_b.append(b)

                        neighbors_r.sort()
                        neighbors_g.sort()
                        neighbors_b.sort()
                        mid = len(neighbors_r) // 2
                        res_px[x, y] = (
                                neighbors_r[mid],
                                neighbors_g[mid],
                                neighbors_b[mid],
                        )

        return result.convert(original_mode)


def apply_grayscale(image: Image.Image, _: int = 0) -> Image.Image:
        return ImageOps.grayscale(image)


def apply_invert(image: Image.Image, _: int = 0) -> Image.Image:
        if image.mode == ”RGBA”:
                rgb, alpha = image.convert(”RGB”), image.split()[3]
                inv = ImageOps.invert(rgb).convert(”RGBA”)
                inv.putalpha(alpha)
                return inv
        if image.mode in (”1”, ”L”):
                return ImageOps.invert(image.convert(”L”))
        return ImageOps.invert(image.convert(”RGB”)).convert(image.mode)


def apply_blur(image: Image.Image, radius: int = 2) -> Image.Image:
        return image.filter(ImageFilter.GaussianBlur(radius=max(0, radius)))


def apply_sharpen(image: Image.Image, amount: int = 2) -> Image.Image:
        amount = max(1, amount)
        return ImageEnhance.Sharpness(image).enhance(amount)
```

```
def apply_edge_enhance(image: Image.Image, _: int = 0) -> Image.Image:
        return image.filter(ImageFilter.FIND_EDGES)


def apply_emboss(image: Image.Image, _: int = 0) -> Image.Image:
        return image.filter(ImageFilter.EMBOSS)


FilterFunc = Callable[[Image.Image, int], Image.Image]


@dataclass
class FilterSpec:
        name: str
        func: FilterFunc
        param_label: str
        param_default: int
        param_min: int
        param_max: int


FILTERS = [
        FilterSpec("Grayscale", apply_grayscale, "N/A", 0, 0, 0),
        FilterSpec("Invert", apply_invert, "N/A", 0, 0, 0),
        FilterSpec("Gaussian Blur", apply_blur, "Radius", 2, 0, 20),
        FilterSpec("Sharpen", apply_sharpen, "Amount", 2, 1, 8),
        FilterSpec("Edge Detect", apply_edge_enhance, "N/A", 0, 0, 0),
        FilterSpec("Emboss", apply_emboss, "N/A", 0, 0, 0),
        FilterSpec("Median (manual)", apply_median_filter, "Kernel", 3, 1, 9),
]

class App:
        def __init__(self, root: "tk.Tk") -> None:
                self.root = root
                self.root.title("Image Filters")
                self.root.geometry("1024x700")

                self.original_image: Optional[Image.Image] = None
                self.current_image: Optional[Image.Image] = None
                self.display_image_tk: Optional["tk.PhotoImage"] = None
                self.open_path: Optional[str] = None

                self._build_ui()

                self.log_path = os.path.join(os.path.dirname(__file__), "app.
   log")

        def _build_ui(self) -> None:
                toolbar = tk.Frame(self.root)
                toolbar.pack(side=tk.TOP, fill=tk.X)

                btn_open = tk.Button(toolbar, text="Open", command=self.
   on_open)
```

```
            btn_open.pack(side=tk.LEFT, padx=4, pady=4)

            btn_save = tk.Button(toolbar, text="Save As", command=self.
on_save)
            btn_save.pack(side=tk.LEFT, padx=4, pady=4)

            btn_reset = tk.Button(toolbar, text="Reset", command=self.
on_reset)
            btn_reset.pack(side=tk.LEFT, padx=4, pady=4)

            # Filters
            self.filter_var = tk.StringVar(value=FILTERS[0].name)
            lbl = tk.Label(toolbar, text="Filter:")
            lbl.pack(side=tk.LEFT, padx=(16, 4))

            self.filter_menu = tk.OptionMenu(toolbar, self.filter_var, *[f
.name for f in FILTERS], command=self.on_filter_change)
            self.filter_menu.pack(side=tk.LEFT, padx=4)

            self.param_label_var = tk.StringVar(value="Parameter")
            self.param_label = tk.Label(toolbar, textvariable=self.
param_label_var)
            self.param_label.pack(side=tk.LEFT, padx=(16, 4))

            self.param = tk.IntVar(value=FILTERS[0].param_default)
            self.param_scale = tk.Scale(toolbar, from_=FILTERS[0].
param_min, to=FILTERS[0].param_max, orient=tk.HORIZONTAL, variable=self.
param, command=self.on_param_change)
            self.param_scale.config(state=tk.DISABLED)
            self.param_scale.pack(side=tk.LEFT, padx=4)

            btn_apply = tk.Button(toolbar, text="Apply", command=self.
on_apply)
            btn_apply.pack(side=tk.LEFT, padx=8)

            btn_apply_save = tk.Button(toolbar, text="Apply & Save...",
command=self.on_apply_and_save)
            btn_apply_save.pack(side=tk.LEFT, padx=4)

            self.auto_apply = tk.BooleanVar(value=False)
            chk_auto = tk.Checkbutton(toolbar, text="Auto apply", variable
=self.auto_apply)
            chk_auto.pack(side=tk.LEFT, padx=(16, 4))

            # Canvas
            self.canvas = tk.Canvas(self.root, bg="#222")
            self.canvas.pack(fill=tk.BOTH, expand=True)
            self.canvas.bind("<Configure>", lambda _e: self.
_render_to_canvas())

    def on_open(self) -> None:
            # Use tuple of patterns (macOS Tk expects a list, semicolons
can crash Cocoa dialog)
```

```
            patterns = ("*.png", "*.jpg", "*.jpeg", "*.bmp", "*.gif", "*.
tiff", "*.heic", "*.heif")
            path = filedialog.askopenfilename(filetypes=[
                    ("Images", patterns),
                    ("All files", "*.*"),
            ])
            if not path:
                    return
            try:
                    img = Image.open(path)
                    self.original_image = img.copy()
                    self.current_image = img.copy()
                    self.open_path = path
                    self._render_to_canvas()
            except Exception as exc:  # pragma: no cover
                    self._log_exception("Open error", exc)
                    message = f"{exc}\n\Возможнон, формат не
поддерживается. Попробуйте PNG/JPG или установите pillow-heif."
                    messagebox.showerror("Open error", message)

    def on_save(self) -> None:
            if not self.current_image:
                    messagebox.showinfo("Nothing to save", "Open an image
first")
                    return
            default_name = "filtered.png"
            if self.open_path:
                    base, _ = os.path.splitext(os.path.basename(self.
open_path))
                    default_name = f"{base}_filtered.png"
            path = filedialog.asksaveasfilename(defaultextension=".png",
initialfile=default_name)
            if not path:
                    return
            try:
                    self.current_image.save(path)
                    messagebox.showinfo("Saved", f"Saved to: {path}")
            except Exception as exc:  # pragma: no cover
                    messagebox.showerror("Save error", str(exc))

    def on_reset(self) -> None:
            if self.original_image is None:
                    return
            self.current_image = self.original_image.copy()
            self._render_to_canvas()

    def on_filter_change(self, _value: Optional[str] = None) -> None:
            spec = self._current_filter_spec()
            self.param_label_var.set(spec.param_label)
            self.param_scale.config(from_=spec.param_min, to=spec.
param_max)
            self.param.set(spec.param_default)
```

```
            self.param_scale.config(state=(tk.NORMAL if spec.param_max >
spec.param_min else tk.DISABLED))
            if self.auto_apply.get():
                    self.on_apply()

    def on_apply(self) -> None:
            if self.current_image is None:
                    messagebox.showinfo("No image", "Open an image first")
                    return
            spec = self._current_filter_spec()
            param_value = int(self.param.get())
            try:
                    self.current_image = spec.func(self.current_image,
param_value)
                    self._render_to_canvas()
            except Exception as exc:
                    self._log_exception("Filter error", exc)
                    messagebox.showerror("Filter error", str(exc))

    def on_apply_and_save(self) -> None:
            self.on_apply()
            self.on_save()

    def on_param_change(self, _val: str) -> None:
            if self.auto_apply.get():
                    self.on_apply()

    def _current_filter_spec(self) -> FilterSpec:
            name = self.filter_var.get()
            for f in FILTERS:
                    if f.name == name:
                            return f
            return FILTERS[0]

    def _render_to_canvas(self) -> None:
            if self.current_image is None:
                    self.canvas.delete("all")
                    return
            w = self.canvas.winfo_width() or 1
            h = self.canvas.winfo_height() or 1
            img = self.current_image.copy()
            # Fit into canvas, preserve aspect
            img.thumbnail((w - 10, h - 10))
            self.display_image_tk = _pil_to_photoimage(img)
            self.canvas.delete("all")
            cx = (w - self.display_image_tk.width()) // 2
            cy = (h - self.display_image_tk.height()) // 2
            self.canvas.create_image(cx, cy, anchor=tk.NW, image=self.
display_image_tk)

    def _log_exception(self, title: str, exc: Exception) -> None:
            stack = traceback.format_exc()
            _append_log(self.log_path, f"[{title}] {exc}\n{stack}")
```

```
def _pil_to_photoimage(image: Image.Image) -> ”tk.PhotoImage”:
        with io.BytesIO() as buffer:
                image.save(buffer, format=”PNG”)
                data = buffer.getvalue()
        return tk.PhotoImage(data=data)


def _append_log(log_path: str, text: str) -> None:
        try:
                with open(log_path, ”a”, encoding=”utf-8”) as f:
                        f.write(text + ”\n”)
        except Exception:
                pass



def main() -> None:
        parser = argparse.ArgumentParser(description=”Image Filters GUI / CLI
    ”)
        parser.add_argument(”--demo”, action=”store_true”, help=”Run headless
    demo: generate sample and apply all filters”)
        parser.add_argument(”--input”, type=str, default=None, help=”Path to
    input image for CLI mode”)
        parser.add_argument(”--outdir”, type=str, default=”./outputs”, help=”
    Directory to save outputs in CLI mode”)
        parser.add_argument(”--param”, type=int, default=None, help=”Optional
    parameter value for filters that use it”)
        args = parser.parse_args()

        if args.demo or args.input:
                _run_cli(args)
                return

        if not exists_tk:  # pragma: no cover
                raise RuntimeError(”tkinter is not available in this
    environment; use --demo or --input CLI mode”)
        root = tk.Tk()
        App(root)
        root.mainloop()


def _run_cli(args: argparse.Namespace) -> None:
        os.makedirs(args.outdir, exist_ok=True)

        if args.input:
                img = Image.open(args.input)
        else:
                w, h = 512, 320
                img = Image.new(”RGB”, (w, h))
                for y in range(h):
                        for x in range(w):
```

```
                                r = int(255 * x / max(1, w - 1))
                                g = int(255 * y / max(1, h - 1))
                                b = int(255 * (x ^ y) / 255)
                                img.putpixel((x, y), (r, g, b))

    base_name = ”input_demo” if not args.input else os.path.splitext(os.
path.basename(args.input))[0]
    input_path = os.path.join(args.outdir, f”{base_name}.png”)
    img.save(input_path)

    for spec in FILTERS:
            param_value = spec.param_default if args.param is None else
int(args.param)
            # ensure odd kernel for median when user passes even
            if spec.name.startswith(”Median”) and param_value % 2 == 0:
                    param_value = max(1, param_value - 1)
            out = spec.func(img, param_value)
            fname = f”{base_name}_{spec.name.replace(’ ’, ’_’).replace
(’(’, ’’).replace(’)’, ’’)}.png”
            out.save(os.path.join(args.outdir, fname))
```

В ходе выполнения работы на вход программы было подано изображение. На Рисунке 1 представлено исходное изображение.

Рисунок 1 – Изображение для оценки работы программы

На Рисунке 2 показано, как можно выбрать режим для обработки изображения: Grayscale, Invert, Gaussian Blur, Sharpen, Edge Detect, Emboss, Median

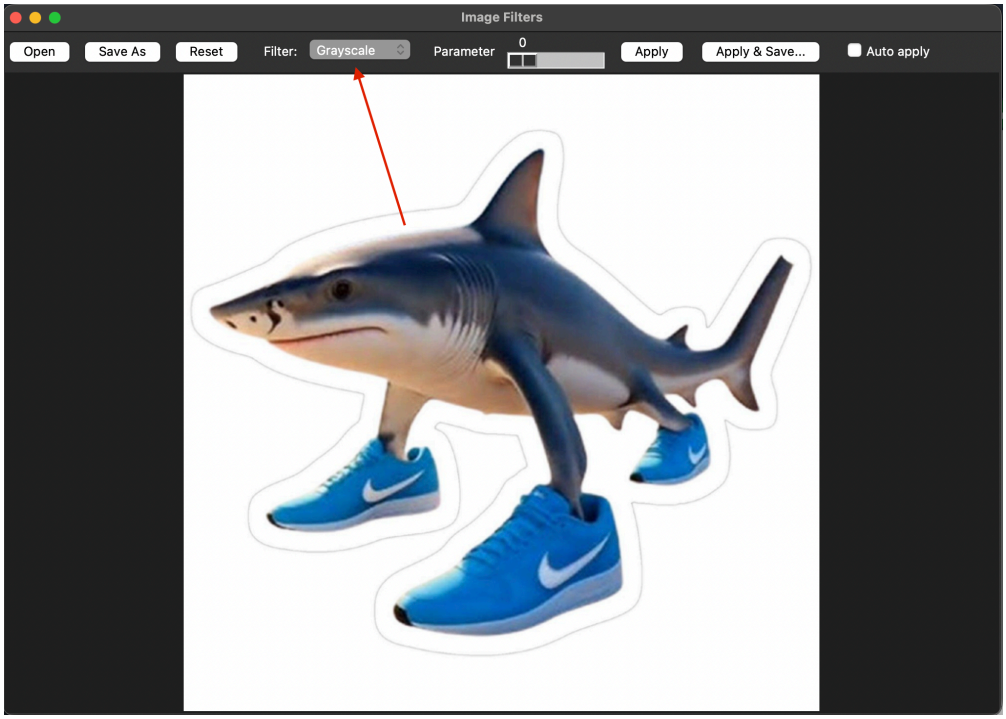

Рисунок 2 – Меню для выбора режима фильтра

ВЫВОД

В ходе выполнения лабораторной работы были изучены основные методы предварительной обработки изображений, направленные на улучшение качества изображений и подготовку их к дальнейшему анализу. Были реализованы и протестированы различные фильтры, включая медианный, средний (усредняющий), гауссов, фильтр Собеля и фильтр Лапласа.