浙江大学
ZHEJIANG UNIVERSITY

数字图像处理与机器视觉 实验报告

作业名称 HW4 图像复原

姓　　名 杨逍宇

学　　号 3220105453

电子邮箱 3220105453@zju.edu.cn

联系电话 13518290755

导　　师 蔡声泽/曹雨齐/姜伟

2025 年 4 月 23 日

1 已实现的功能简述及运行简要说明

1.1 已实现的功能简述：

(1). `blur_restoration.cpp` 实现完整的图像退化-复原流程

(2). 支持两种退化模型：运动模糊 (频域生成)、大气湍流 (5/3 次方衰减)

(3). 提供两种复原方法：维纳滤波 (Wiener)、约束最小二乘滤波 (CLS)

(4). 动态生成结果文件名 (模型 + 方法组合命名)

(5). 可视化频谱图及归一化结果输出

项目目录树如下：

```
/
├── assets  中间过程图像
├── src
│   └── blur_restoration.cpp  图像退化-复原
├── build
│   └── blur_restoration.exe  对应的可执行文件
├── docs
│   └── 实验报告 HW4.pdf  HW4 实验报告
├── CMakeLists.txt
└── README.md
```

1.2 运行说明：

- 输入图像路径：例如`../assets/imgB.jpg`
- 运行交互：

  (1). 选择退化模型 (1: 运动模糊 2: 大气湍流)

  (2). 选择复原方法 (1: 维纳滤波 2: 约束最小二乘)

  (3). 自动保存结果至`../assets/results/`

- 输出命名规则：`[model]_[method]_[degraded/restored].png`

2 开发与运行环境

本实验使用的软件和工具如下：

- 开发环境：Visual Studio Code on Ubuntu22.04
- 编程语言：C++
- 库：OpenCV 4.7.0
- 构建工具：CMake

3 算法基本思路

3.1 退化模型生成

- **运动模糊**：空域生成线状核 →FFT 移位 → 频域转换

$$H(u,v) = \mathcal{F}\left\{\frac{1}{L}\delta(x\cos\theta + y\sin\theta)\right\}$$

- **大气湍流**：频域直接生成 5/3 次方衰减核

$$H(u,v) = e^{-k(u^2+v^2)^{5/6}}$$

3.2 复原算法原理

- **维纳滤波**：本质上是一个最小均方误差估计器，即寻找最佳复原图像 $\hat{f}$ $s.t.$ $e^2 = E\{(f-\hat{f})^2\}$ 最小

$$F_{\text{hat}} = \frac{G \cdot H^*}{|H|^2 + K}$$

- **约束最小二乘**：约束最小二乘滤波法的核心是 H 对噪声敏感的问题：降低其对噪声敏感的一种方法是以平滑测度的最优复原为基础。这里我们引入拉普拉斯正则项

$$F_{\text{hat}} = \frac{H^* G}{|H|^2 + \gamma|L|^2}$$

其中 L 为拉普拉斯算子的频域响应，在本次实验中我选择使用拉普拉斯算子：

$$p(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

4 算法实现要点

4.1 Canny 边缘检测核心代码

```
// 高斯模糊（代码片段）
GaussianBlur(gray, blurred, Size(5, 5), 1.5);

// Canny边缘检测（关键参数）
double lowThreshold = 50;
double highThreshold = 120;
Canny(blurred, edges, lowThreshold, highThreshold);
```

4.2 运动模糊核生成

```
Mat generateMotionBlurKernel(Size size, int len, double angle_deg) {
    Mat kernel = Mat::zeros(size, CV_32F);
    Point center(size.width / 2, size.height / 2);
    double angle_rad = angle_deg * CV_PI / 180.0;

    int dx = static_cast<int>(round((len/2) * cos(angle_rad)));
    int dy = static_cast<int>(round((len/2) * sin(angle_rad)));
    Point end1(center.x - dx, center.y - dy);
    Point end2(center.x + dx, center.y + dy);
    line(kernel, end1, end2, Scalar(1.0/len), 1);

    fftShift(kernel);
    Mat planesH[] = { Mat_<float>(kernel.clone()), Mat::zeros(kernel.size
        (), CV_32F) };
    Mat H;
    merge(planesH, 2, H);
    dft(H, H, DFT_COMPLEX_OUTPUT);
    return H;
}
```

上述代码中对 kernel 做了一个 fftShift 变换是因为实验中直接使用核会使得图像 13,24 象限进行对调因此在这进行处理。

4.3 大气湍流传递函数生成

```
Mat generateAtmosphericTurbulenceKernel(Size size, double k) {
    Mat H_real = Mat::zeros(size, CV_32F);
    int cx = size.width / 2;
    int cy = size.height / 2;

    for (int i = 0; i < size.height; ++i) {
        for (int j = 0; j < size.width; ++j) {
            float u = j - cx;
            float v = i - cy;
            float D = sqrt(u*u + v*v);
            H_real.at<float>(i, j) = exp(-k * pow(D, 5.0/3));
        }
    }
    fftShift(H_real); // 确保原点在左上角，与DFT结果对齐
    Mat H_imag = Mat::zeros(H_real.size(), CV_32F);
    Mat planesH[] = {H_real, H_imag};
    Mat H;
    merge(planesH, 2, H);
    return H;
}
```

4.4 维纳滤波实现

```
Mat wienerFilter(const Mat& complexDegraded, const Mat& H, double K) {
    Mat H_planes[2];
    split(H, H_planes);
    Mat H_conj_planes[] = {H_planes[0], -H_planes[1]};
    Mat H_conj;
    merge(H_conj_planes, 2, H_conj);

    Mat H_mag_abs;
    magnitude(H_planes[0], H_planes[1], H_mag_abs);
    Mat H_mag_sq = H_mag_abs.mul(H_mag_abs);

    Mat numerator;
    mulSpectrums(complexDegraded, H_conj, numerator, 0, true);
```

```
    Mat denominator;
    // Mat denom_planes[] = {H_mag_sq + K, Mat::zeros(H_mag_sq.size(), CV_32F)};
    // 这里不是复数除法，只有实数部分，但是divide是分别相除，因此两个部分都设置为H_mag_sq + K
    Mat denom_planes[] = {H_mag_sq + K, H_mag_sq + K};
    merge(denom_planes, 2, denominator);
    Mat F_hat;
    divide(numerator, denominator, F_hat);
}
```

4.5 约束最小二乘滤波实现

```
Mat constrainedLeastSquaresFilter(const Mat& complexDegraded, const Mat& H, double gamma) {
    Mat laplacianKernel = (Mat_<float>(3,3) << 0, 1, 0,
                                               1, -4, 1,
                                               0, 1, 0);
    Mat paddedLaplacian = Mat::zeros(H.size(), CV_32F);
    laplacianKernel.copyTo(paddedLaplacian(Rect(0, 0, 3, 3)));
    fftShift(paddedLaplacian);

    Mat planesL[] = {paddedLaplacian, Mat::zeros(paddedLaplacian.size(), CV_32F)};
    Mat L;
    merge(planesL, 2, L);
    dft(L, L, DFT_COMPLEX_OUTPUT);

    Mat H_planes[2], L_planes[2];
    split(H, H_planes);
    split(L, L_planes);

    // |H|² + gamma*|L|²
    Mat H_mag_sq = H_planes[0].mul(H_planes[0]) + H_planes[1].mul(H_planes[1]);
    Mat L_mag_sq = L_planes[0].mul(L_planes[0]) + L_planes[1].mul(L_planes[1]);
    Mat denominator = H_mag_sq + gamma * L_mag_sq;
```

```
    denominator = max(denominator, 1e-6f);

    // F_hat = (H* * G) / (|H|² + gamma|L|²)
    Mat numerator;
    mulSpectrums(complexDegraded, H, numerator, 0, true);

    Mat F_hat_planes[2];
    split(numerator, F_hat_planes);
    divide(F_hat_planes[0], denominator, F_hat_planes[0]);
    divide(F_hat_planes[1], denominator, F_hat_planes[1]);
```

4.6 动态命名实现

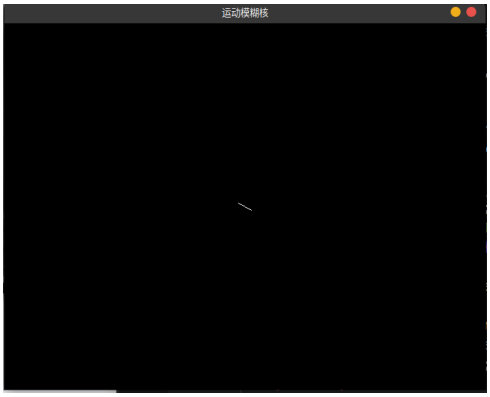
```
string modelName = (choice==1) ? "motion" : "atmospheric";
string methodName = (method==1) ? "wiener" : "clsq";
string path = modelName + "_" + methodName + "_restored.png";
```

5 实验结果及分析

5.1 不同退化模型

我对如下原始图像进行了不同方法的退化和复原方法测试:

然后我分别对原始图像进行运动模糊和大气湍流退化其中运动模糊核如下:



得到退化结果如下：

(a) 运动模糊 (b) 大气湍流

Figure 1: 不同退化模型结果

然后我们分别对两种退化模型结果使用维纳滤波和约束最小二乘滤波：

5.2 维纳滤波

使用维纳滤波效果如下：

(a) 运动模糊复原 (b) 大气湍流复原

Figure 2: 维纳滤波结果

实验中我选择了不同的 k 值，经过测试发现 k 越大针对上述图像复原效果越模糊，因此在本次实验中我选择了 $k = 0.01$

5.3 约束最小二乘滤波

使用约束最小二乘滤波效果如下：

(a) 运动模糊复原 (b) 大气湍流复原

Figure 3: 约束最小二乘滤波结果

实验中我选择了不 $\gamma = 0.001$，发现效果相对维纳滤波更加清晰。

参数影响分析

- **维纳滤波 K 值**：过小导致噪声放大，过大导致过度平滑
- **约束滤波 值**：控制平滑强度，需平衡振铃抑制与细节保留
- **运动模糊长度**：长度估计误差超过 20% 时复原质量显著下降

6 结论与心得体会

通过本实验，我们得出以下结论：

- 频域方法可有效实现复杂退化模型的构建与复原
- 维纳滤波在已知 PSF 时表现最佳，但需要准确估计噪声功率 K
- 约束最小二乘对 PSF 误差更具鲁棒性，适合实际应用场景
- 运动模糊角度估计误差对复原质量影响显著

实验中的改进思考：

- 添加 PSF 参数估计模块（如盲反卷积）
- 实现参数自适应选择算法
- 扩展更多退化模型（离焦模糊、高斯模糊等）

编码经验

- OpenCV 复数运算需注意实虚部分离与合并顺序
- 频域处理必须严格遵守 FFT 移位规范
- 数值稳定性处理是频域滤波实现的关键