数字图像处理实验二

09021227 金桥

2023 年 10 月 13 日

1 实验目标

在第一次实验作业的基础上，搭建一个可视化界面，实现对幻灯片中磁共振膝盖图像进行傅立叶变换、直方图表示、直方图均衡化、CLAHE 算法结果的展示，并体现自己的思考过程。

2 实验内容

2.1 傅里叶变换

傅里叶变换已经在第一次实验中实现，参见实验报告一。

2.2 显示直方图

2.2.1 使用 OpenCV 计算直方图

通过调用 OpenCV 提供的 `calcHist` 函数可以方便快速的实现直方图计算。具体调用参见代码。

2.2.2 自行计算直方图

计算过程如下，具体实现参见代码。

- 首先使用 `cvtColor` 函数将原图像转为灰度图
- 之后通过 `cvt2dVector` 函数将 `Mat` 格式的图像转换为 `int` 格式的二维 `vector`
- 之后遍历所有像素，统计不同灰度像素点数量
- 最后将统计结果绘图

2.3 直方图均衡

2.3.1 使用 OpenCV 进行直方图均衡

通过调用 OpenCV 的 `equalizeHist` 函数可以方便快速的实现直方图均衡。具体调用参见代码。

2.3.2 自行直方图均衡

计算过程如下，具体实现参见代码。

- 首先使用之前的计算直方图的步骤计算直方图
- 之后根据直方图数值计算累计分布函数，并进行归一化
- 根据计算出来的累计分布函数将原来的图像灰度映射到新的灰度上

2.4 CLAHE 算法

2.4.1 使用 OpenCV 实现 CLAHE

通过调用 OpenCV 提供的 `CLAHE` 类可以方便快速的实现 CLAHE 算法。具体调用参见代码。

2.4.2 自行实现 CLAHE

计算过程如下，具体实现参见代码。

- 首先将图像分块 (OpenCV 默认 8×8 分块, 与其保持一致)，图像边缘不够的进行补齐
- 对每个分块进行处理：
  - 计算出分块的直方图数值
  - 根据预设的阈值对直方图进行裁切，将高处的部分均匀补齐至所有灰度上
  - 返回处理后的直方图
- 为防止棋盘效应对图像造成割裂，采用双线性插值
  - 对每个块的像素，取离它最近的四个分块，并取出四个分块处理后对应的灰度值
  - 假设从上到下，从左到右，四个灰度值分别是 r_1, r_2, r_3, r_4
  - 假设在离这个像素最近的四个分块的中心点形成的矩形中：
    - 这个像素距离上下边界的距离是 y_1, y_2，左右边界距离 x_1, x_2，分块横纵大小 w, h
  - 代入下式计算最终像素灰度 r：

$$r = (r_1\frac{x_1}{w} + r_2\frac{x_2}{w})\frac{y_1}{h} + (r_3\frac{x_1}{w} + r_4\frac{x_2}{w})\frac{y_2}{h}$$

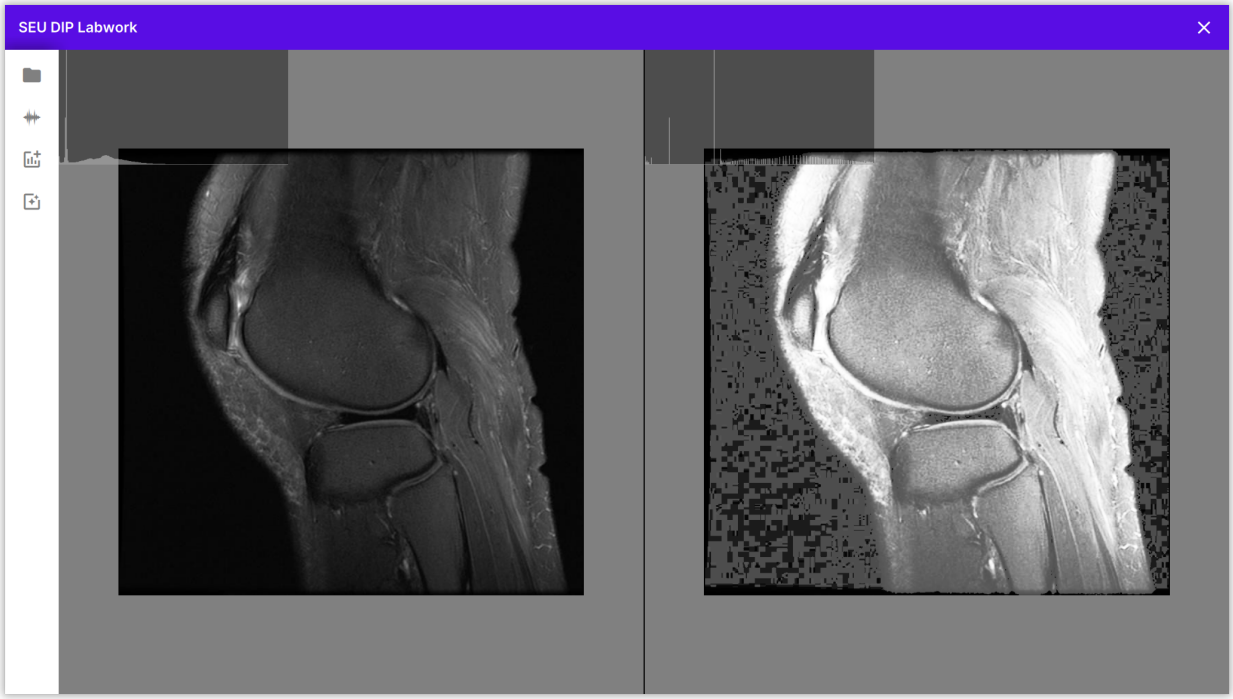
3 实验结果

3.1 显示直方图

直方图显示采用的是自行实现的算法。

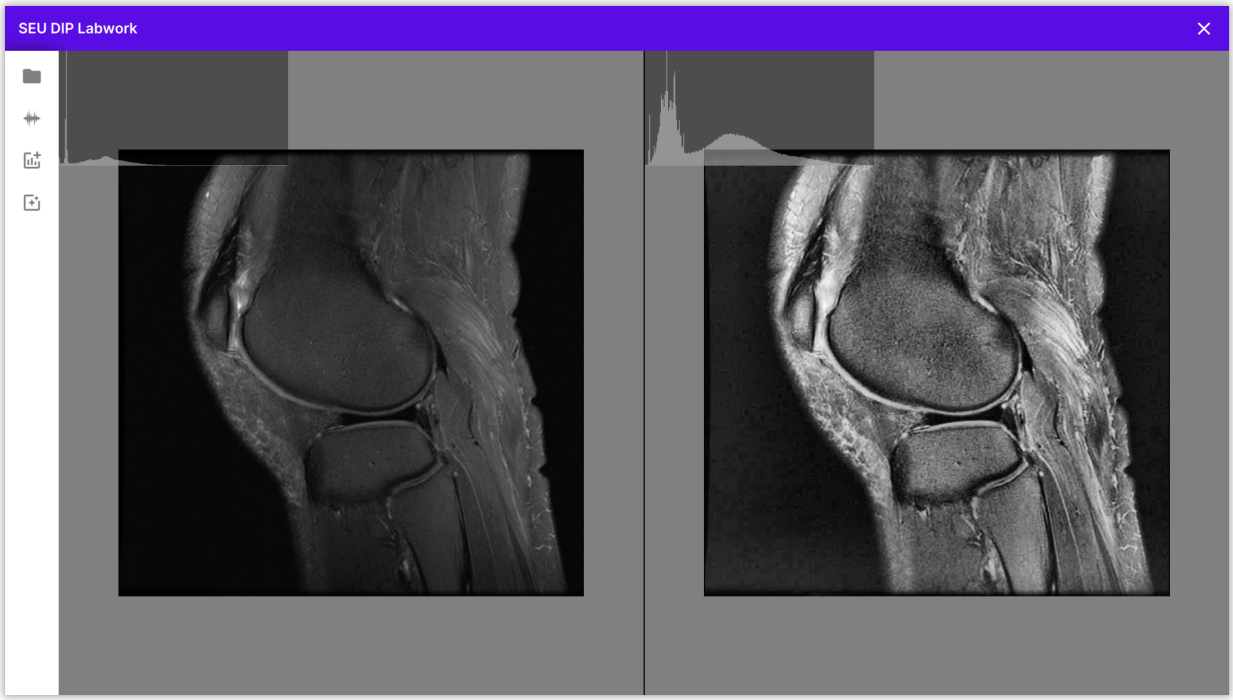
3.2 直方图均衡

下图是自行实现的直方图均衡算法。



3.3 CLAHE 算法

下图是自行实现的 CLAHE 算法。



4 思考

4.1 为什么 OpenCV 的傅里叶变换速度比较快？

作为对比，我自行实现的傅里叶变换的时间复杂度大约是 $O(m^2n^2)$ 而 OpenCV 的傅里叶变换速度极快，几乎是瞬间实现的。查阅资料后，发现主要原因有以下两个：

1. OpenCV 大量运用了并行计算，对于傅里叶变换这种大量 `for` 循环的算法具有显著的优势

2. OpenCV 的傅里叶变换采用的是快速傅里叶变换，而我采用的是离散傅里叶变换，虽然结果基本一致但在时间复杂度上差了一个量级

4.2 直方图相关算法结果细微差异原因

对于直方图相关算法，我的实现与 OpenCV 的实现产生的结果有非常细微的差别，经过分析，我认为主要原因是对于浮点数转为整数，我采用的是向下取整，而 OpenCV 采用的是四舍五入取整，因此会有非常细微的差别。

4.3 CLAHE 对比 HE 的优势

CLAHE 相对于 HE 的优势主要是它不会产生过度的增强，在改善整体图像对比度的同时不会过度增强某些区域的对比度，从而避免图像细节丢失。

我个人认为这主要得益于 CLAHE 有 `clipLimit` 的设置，可以防止过度增强对比度。除此以外，CLAHE 的分块计算也在一定程度上保留了局部图像的对比度。