



# 医学图像处理

## 医学图像处理软件项目研发报告

作者：黄锦峰

组织：东南大学计算机科学与技术学院

时间：November, 2022

版本：V 1.0



# 前言

## 编写的目的

本软件项目开发报告旨在帮助用户了解该医学图像处理软件的相关功能及其使用方法。

同时作为数字图像处理课程的实践部分，本文档较为简明地介绍了项目的开发流程，对项目开发有一定的指导作用。希望在本次项目开发中对课程中所学的知识进行了实践，提高了对知识的理解和运用能力。

## 软件简介

本软件基于 Qt 开发，系统结构较为简单，分为两层，分别是前端界面层和后端处理层。（1.2.1 系统结构中有详细的介绍）

前端界面上充分考虑了用户的使用习惯，界面简洁美观，操作简单易用。针对特定格式的医学图像文件（.raw）实现了图像的读取、灰度窗显示、灰度窗宽窗位调整、灰度反转显示、图像的几何变换、图像锐化、图像翻转、显示图像保存等功能。同时也为用户提供了操作的历史记录（操作日志），用户可以根据自己的需要进行数据存档，便于快速的进行数据的回溯（双击即可读取数据）。

后端数据处理层面，以空间换时间同时对处理的图像数据做了备份。在连续变化中具有一定的操作优势，同时降低了编程的难度，有比较好的拓展性。

## 参考资料

- Refuel C.Gonzalez & Richard E.Woods. 数字图像处理（第三版）Digital Image Processing （Third Edition）

## 联系作者

邮箱:213201953@seu.edu.cn

Github 仓库:[https://github.com/Frank2001Feng/DIP\\_09020334\\_App](https://github.com/Frank2001Feng/DIP_09020334_App)

# 目录

前言	0
<b>第1章 程序系统分析</b>	<b>1</b>
1.1 需求分析 . . . . .	1
1.1.1 图像处理功能需求 . . . . .	1
1.1.2 用户交互功能需求 . . . . .	1
1.2 系统设计分析 . . . . .	1
1.2.1 系统结构 . . . . .	1
1.2.2 类分析 . . . . .	3
1.2.2.1 <i>ImageProcess.h/cpp</i> . . . . .	3
1.2.2.2 <i>MainWindow.h/cpp</i> . . . . .	3
<b>第2章 功能模块设计</b>	<b>4</b>
2.1 读取 RAW 图像 . . . . .	4
2.1.1 读取 RAW 图像分析 . . . . .	4
2.1.2 读取 RAW 图像的核心代码 . . . . .	4
2.2 灰度窗调整 . . . . .	5
2.2.1 灰度窗调整分析 . . . . .	5
2.2.2 灰度窗调整的核心代码 . . . . .	5
2.3 灰度反转 . . . . .	6
2.3.1 灰度反转分析 . . . . .	6
2.3.2 灰度反转核心代码 . . . . .	6
2.4 图像放缩 . . . . .	6
2.4.1 图像放缩分析 . . . . .	6
2.4.2 图像放缩核心代码 . . . . .	7
2.5 图像旋转 . . . . .	8
2.5.1 图像旋转分析 . . . . .	8
2.5.2 图像旋转核心代码 . . . . .	8
2.6 图像翻转 . . . . .	10
2.6.1 图像翻转分析 . . . . .	10
2.6.2 图像翻转核心代码 . . . . .	10
2.6.2.1 水平翻转 . . . . .	10
2.6.2.2 竖直翻转 . . . . .	10
2.6.2.3 对角翻转 . . . . .	11
2.7 图像锐化/细节增强 . . . . .	11
2.7.1 图像锐化/细节增强分析 . . . . .	11
2.7.2 图像锐化/细节增强核心代码 . . . . .	12
2.8 保存图像 . . . . .	12
2.8.1 保存图像分析 . . . . .	12
2.8.2 保存图像核心代码 . . . . .	12
2.9 重置与恢复原图 . . . . .	13
2.9.1 重置与恢复原图分析 . . . . .	13

2.9.2 重置与恢复原图核心代码	13
2.10 操作日志	14
2.10.1 操作日志分析	14
2.10.2 操作日志核心代码	14
2.10.2.1 数据存档	14
2.10.2.2 双击读取	15
2.10.2.3 全部清除	15
<b>第3章 演示和测试</b>	<b>16</b>
3.1 程序界面	16
3.2 读取 RAW 图像	16
3.3 灰度窗调整	17
3.4 灰度反转	17
3.5 图像放缩与旋转	17
3.6 图像翻转	18
3.7 图像锐化/细节增强	19
3.8 图像保存	19
3.9 其余操作	20
3.9.1 重置	20
3.9.2 原始图像	20
3.9.3 操作日志	20
<b>附录 A 值得一提的操作</b>	<b>21</b>
A.1 给运行程序一个图标	21
A.2 封装成.exe 文件	21

# 第1章 程序系统分析

## 1.1 需求分析

1. 医生在读片时可能需要多次调整灰度窗，并希望及时看到该灰度窗参数下显示的图像；
2. 医生可能需要多次进行灰度反转（即灰度反转显示，再回复到正常灰度，再灰度反转显示）
3. 应考虑处理功能的撤消操作；
4. 应提供当前显示图像的保存功能（模拟生成打印检查报告）

### 1.1.1 图像处理功能需求

- 读取指定样式的图像文件 (.raw 格式)
- 根据灰度窗宽窗位参数处理图像数据生成图像
- 灰度反转
- 图像放大和缩小
- 图像旋转
- 图像翻转
- 图像锐化/细节增强

### 1.1.2 用户交互功能需求

- 打开指定样式的图像文件 (.raw 格式)
- 满足连续调节，图像画面不会无端重置
- 处理功能的撤消操作
- 对一些参数进行保存，显示操作日志
- 当前显示图像的保存功能 (.bmp 格式)

## 1.2 系统设计分析

### 1.2.1 系统结构

- 作为一个小型的图像处理软件，本软件的系统设计分析主要分为两层：前端界面层和后端处理层
- 前端界面层主要负责用户与软件的交互，包括用户界面的设计和实现，以及用户界面的事件处理、图像的显示等
- 后端处理层主要负责图像数据的处理，包括读取图像数据的存储，图像数据的变换，返回满足要求的图像数据等

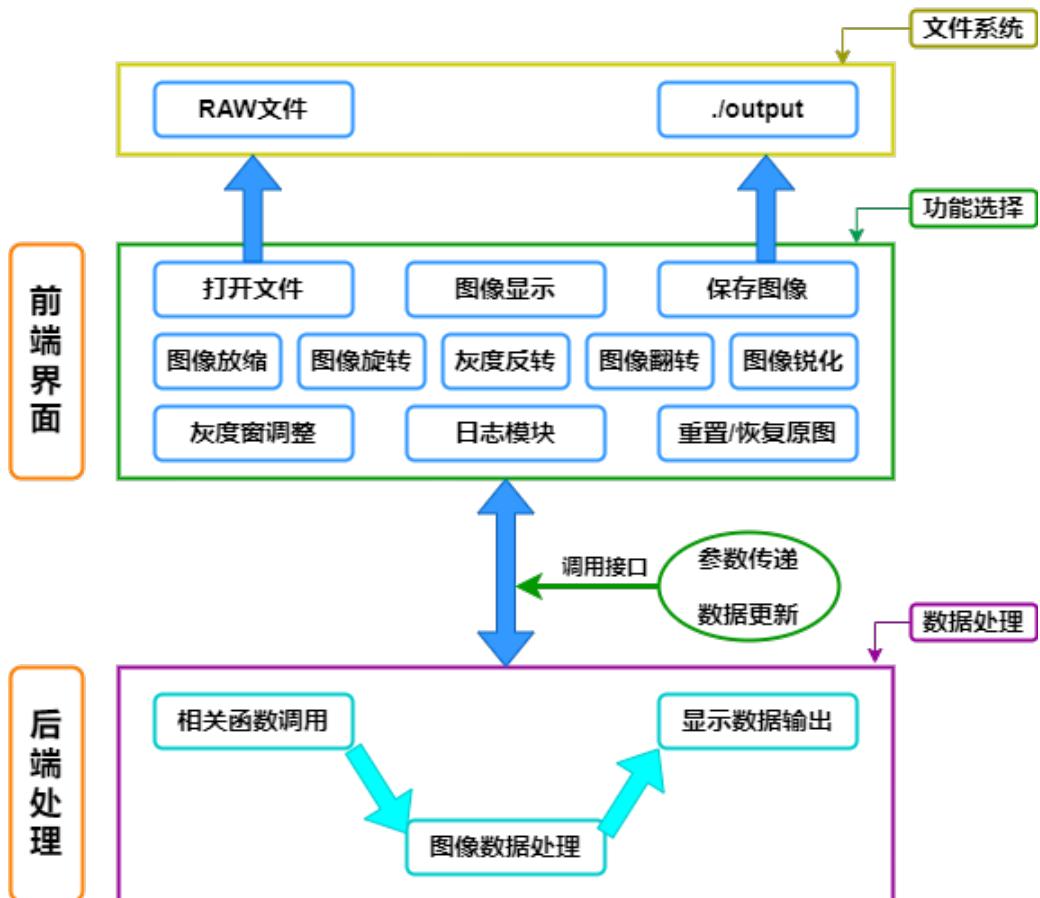


图 1.1: 系统结构图

- 前端界面中的按钮操作都连接着后端的处理函数
- 后端处理涉及多种函数，具体函数和实现见下文

### 1.2.2 类分析

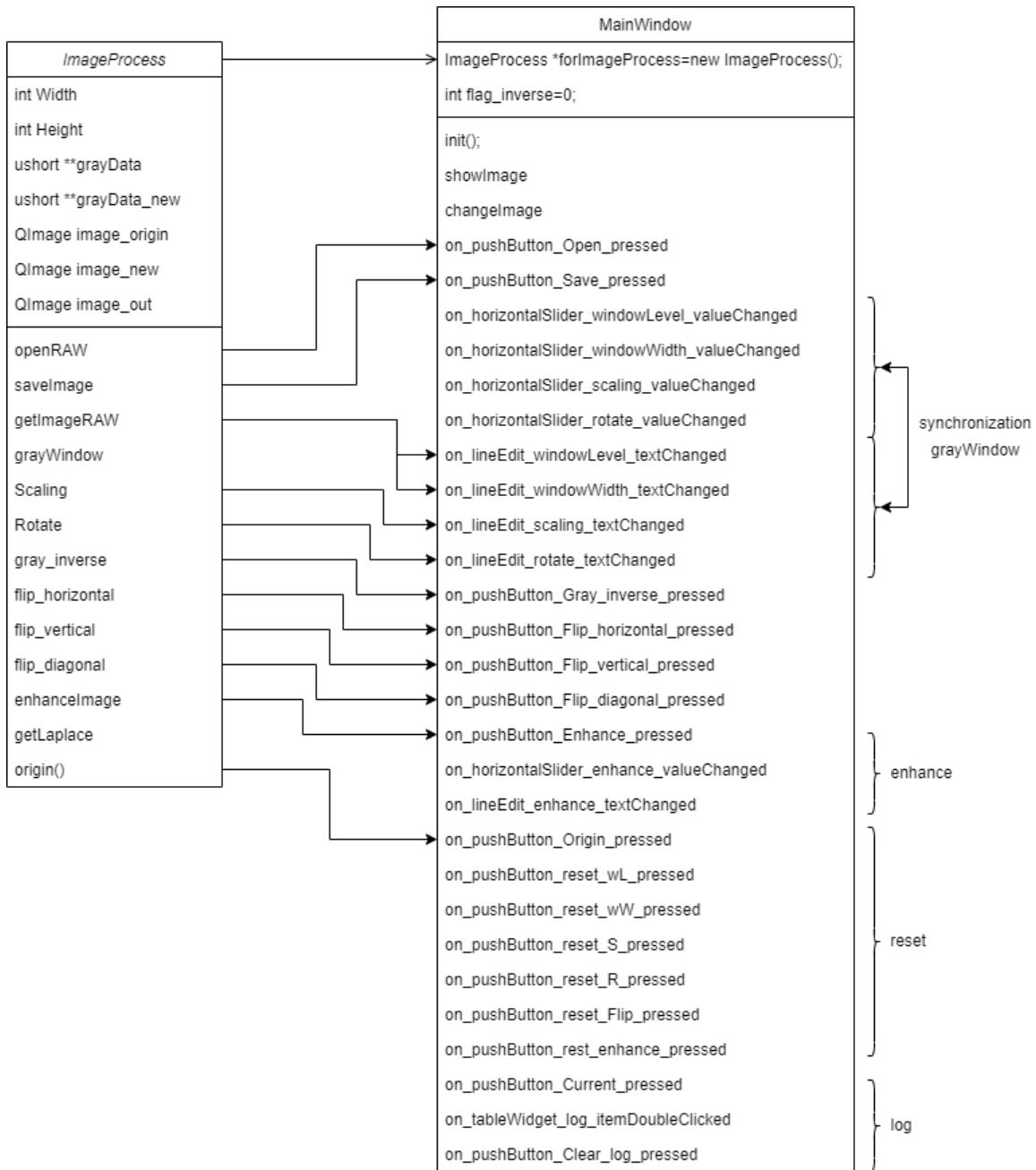


图 1.2: 类图

### 1.2.2.1 ImageProcess.h/cpp

对图像的数据进行处理，同时将处理后的图像 `image_out` 交 `MainWindow` 显示  
其中有不止一个 `QImage` 其作用是时连续调节更方便，同时还保存了初始数据，便于复原与一些撤销操作

### 1.2.2.2 MainWindow.h/cpp

创建一个 ImageProcess 对象，(这里可以进行拓展，我们完全可以处理多个图像)

有一个 `flag_inverse` 记录灰度反转操作的状态，本软件操作中没有对翻转状态做记录，在此可见其实现方法是同理的，可进行拓展。

## 第2章 功能模块设计

### 2.1 读取 RAW 图像

在此对本软件的读取的 RAW 做一些说明：

图像数据有效灰度范围 [0, 4095]，即 12 位有效灰度，每像素 2 字节（最高 4 位数据无效，有效灰度保存于低 12 位）。数据文件为自定义格式（非标准格式），文件中的数据按字节存放顺序如下图：



文件开始的 4 字节存放图像宽参数，其后 4 字节存放图像高参数，此两参数均为无符号长整型（`unsigned long`），紧随其后为按光栅扫描顺序（从左向右，逐行扫描）存放的像素值，像素值为无符号短整型（`unsigned short`）。所有多字节数据都按 intel 顺序（即低字节在前，高字节在后）存放。文件不包含其它数据。

#### 2.1.1 读取 RAW 图像分析

1. 主界面点击“打开”选择特定的 RAW 文件，判断是否正确读取。（图片文件路径不能包含中文）
2. 读取前 8 个字节后确定图像的宽和高（存入 `imagepeocess.Width` 和 `imagepeocess.Height`），由此创建二维数组 (`grayData`)
3. 依次读取后续的字节，将其转换为无符号短整型，存入二维数组中

#### 2.1.2 读取 RAW 图像的核心代码

```
bool ImageProcess::openRAW()
{
    OpenFile =QFileDialog::getOpenFileName(this,"打开RAW文件","","*.RAW");
    if(OpenFile.isEmpty())
    {
        return false;
    }
    else{
        QByteArray byteData=OpenFile.toLatin1();
        char * fileData=byteData.data();
        FILE *toRead=fopen(fileData,"rb");
        //依次读入width, height
        unsigned long *ptrPara=new (unsigned long);
        fread(ptrPara,1,4,toRead);
        Width=(int)*ptrPara;
        fread(ptrPara,1,4,toRead);
        Height=(int)*ptrPara;
        // 依次读入数据
        grayData=new ushort*[Width];
        grayData_new=new ushort*[Width];
        for(int i=0;i<Width;i++)
```

```

{
    grayData[i]=new ushort[Height];
    grayData_new[i]=new ushort[Height];
}
for(int j=0;j<Height;j++)
{
    for(int i=0;i<Width;i++)
    {
        fread(ptrPara,1,2,toRead);
        grayData[i][j]=(ushort)*ptrPara;
        grayData_new[i][j]=grayData[i][j];
    }
}
fclose(toRead);
delete ptrPara;
return true;
}
return false;
}

```

## 2.2 灰度窗调整

### 2.2.1 灰度窗调整分析

- 从主界面中“窗宽”、“窗位”文本框获取窗宽和窗位的值（文本框中数值与对应滚动条一致）
- 根据灰度窗的中心值和宽度(窗宽 & 窗位)，将二维数组中的值转换为灰度值 [0,255]，  
(在取定范围内线性映射)
- 创建 QImage 对象 (img\_origin)，将二维数组中灰度值依次存入 QImage 中对应的像素位置

### 2.2.2 灰度窗调整的核心代码

```

QImage ImageProcess::getImageRAW(int windowLevel, int windowHeight)
{
    QImage newImage=QImage((int)Width,(int)Height,QImage::Format_RGB32);
    for(int j=0;j<(int)Height;j++)
    {
        for(int i=0;i<(int)Width;i++)
        {
            int res=grayWindow(windowLevel,windowWidth,grayData_new[i][j]);
            newImage.setPixelColor(i,j,QColor(res,res,res));
        }
    }
    img_origin=newImage;
    img_new=newImage;
    img_out=newImage;
    return newImage;
}

```

```

int ImageProcess::grayWindow(int windowLevel, int windowHeight, ushort grayValue)
{
    double fwindowLevel=(double)windowLevel;
    double fwindowWidth=(double)windowWidth;
    int minGray=(int)(fwindowLevel-(fwindowWidth)/2.0+0.5);
    int maxGray=(int)(fwindowLevel+(fwindowWidth)/2.0+0.5);
    if(grayValue<=minGray)
        return 0;
    if(grayValue>=maxGray)
        return 255;
    int result=(int)((grayValue-minGray)*255.0/(double)(windowWidth));
    return result;
}

```

## 2.3 灰度反转

### 2.3.1 灰度反转分析

1. 主界面中点击“灰度反转”按钮，可以连续点击，进行多次灰度反转
2. 为保证灰度反转后其余操作的连续性，对数据源头二维数组进行操作（操作 \*\*grayData\_new）

### 2.3.2 灰度反转核心代码

```

void ImageProcess::gray_inverse()
{
    for(int j=0;j<(int)Height;j++)
    {
        for(int i=0;i<(int)Width;i++)
        {
            grayData_new[i][j]=4095-grayData_new[i][j];
        }
    }
}

```

## 2.4 图像放缩

### 2.4.1 图像放缩分析

1. 从主界面中“放缩”文本框获取放缩倍数（文本框中数值与对应滚动条一致），可进行多次图像放缩，实时调节
2. 为保证图像放缩后其余操作的连续性，对灰度产生的图像（img\_origin）进行操作
3. 确定放缩后图像的宽度和高度，创建 QImage 对象（img\_new），使用双线性插值算法，将 img\_origin 中的像素点映射到新图像中

## 2.4.2 图像放缩核心代码

```

void ImageProcess::Scaling(float scaling)
{
    float dx=scaling;
    float dy=scaling;
    // 传入了放大数值,确定了变化后的参数值
    int width=img_origin.width();
    int height=img_origin.height();
    int newWidth=(int)width*dx;
    int newHeight=(int)height*dy;
    QImage newImage=QImage(newWidth,newHeight,img_origin.format());
    // 双线性插值法确定新图像的对像素的值
    /* 现在开始做反向映射处理
     * 对 newX,newY 像素而言 反向映射后为 srcX,srcY ,srcX=i+u,srcY=j+v
     */
    int i,j; //表示整数部分
    float u,v; //表示小数部分
    for(int newX=0;newX<newWidth;newX++)
    {
        for(int newY=0;newY<newHeight;newY++)
        {
            // 对每一个点进行二维插值, 可以处理彩色图像
            i=(int)newX/dx;u=(float)(newX/dx-i);
            j=(int)newY/dy;v=(float)(newY/dy-j);
            if(i<width-1&&j<height-1) //防止超界
            {
                QColor temp00=img_origin.pixel(i,j);
                QColor temp10=img_origin.pixel(i+1,j);
                QColor temp01=img_origin.pixel(i,j+1);
                QColor temp11=img_origin.pixel(i+1,j+1);
                int rr,gg,bb;
                rr=(int)((1-u)*(1-v)*temp00.red()+(u)*(1-v)*temp10.red()+(1-u)*(v)*temp01.red()+(u)*(v)
                         )*temp11.red());
                gg=(int)((1-u)*(1-v)*temp00.green()+(u)*(1-v)*temp10.green()+(1-u)*(v)*temp01.green()
                         +(u)*(v)*temp11.green());
                bb=(int)((1-u)*(1-v)*temp00.blue()+(u)*(1-v)*temp10.blue()+(1-u)*(v)*temp01.blue()+(u)
                         *(v)*temp11.blue());
                if(rr>255){ rr=255; }
                if(gg>255){ gg=255; }
                if(bb>255){ bb=255; }
                if(rr<0){ rr=0; }
                if(gg<0){ gg=0; }
                if(bb<0){ bb=0; }
                newImage.setPixelColor(newX,newY,QColor(rr,gg,bb));
            }
        }
    }
    img_new=newImage;
}

```

```
    img_out=newImage;
}
```

## 2.5 图像旋转

### 2.5.1 图像旋转分析

- 从主界面中“旋转”文本框获取旋转角度（文本框中数值与对应滚动条一致），可进行多次图像旋转，实时调节
- 为保证图像旋转后其余操作的连续性，产生的图像（img\_new）进行操作
- 确定旋转后图像的宽度和高度，创建 QImage 对象（img\_out），使用双线性插值算法，将 img\_new 中的像素点映射到新图像中

### 2.5.2 图像旋转核心代码

```
void ImageProcess::Rotate(float angel)
{
    // 旋转角度 angel、旋转中心(xx,yy)
    // 角度的单位转化，转弧度，计算 sin,cos 值
    double fRotateAngel=angel*3.1415926535/180.0;
    float fsin,fco;
    fsin=(float)sin((double)fRotateAngel);
    fco=(float)cos((double)fRotateAngel);

    // 确定原图像的几何中心（矩阵中心）
    // 以几何中心为旋转中心时
    int width=img_new.width();
    int height=img_new.height();
    float a=(float)(width/2);
    float b=(float)(height/2);

    /*----下面开始确定 newImage 的大小和几何中心----*/
    // 确定一下 newImage 的中心
    float x1,y1,x2,y2,x3,y3,x4,y4;
    x1=(float)(0-a);y1=(float)(0-b);
    x2=(float)(width-a);y2=(float)(0-b);
    x3=(float)(width-a);y3=(float)(height-b);
    x4=(float)(0-a);y4=(float)(height-b);
    // 旋转后四个角的坐标
    float xt1,yt1,xt2,yt2,xt3,yt3,xt4,yt4;
    xt1=fco*x1-fsin*y1;yt1=fsin*x1+fco*y1;
    xt2=fco*x2-fsin*y2;yt2=fsin*x2+fco*y2;
    xt3=fco*x3-fsin*y3;yt3=fsin*x3+fco*y3;
    xt4=fco*x4-fsin*y4;yt4=fsin*x4+fco*y4;
    // 计算旋转后的图像真实宽度
    int newWidth,newHeight;
    if(abs(xt3-xt1)>abs(xt4-xt2))
```

```

newWidth=(int)(abs(xt3-xt1));
else
    newWidth=(int)(abs(xt4-xt2));
// 计算旋转后的图像真实高度
if(abs(yt3-yt1)>abs(yt4-yt2))
    newHeight=(int)(abs(yt3-yt1));
else
    newHeight=(int)(abs(yt4-yt2));

// 很好~ 这就定出 c、d
float c=(float)(newWidth/2);
float d=(float)(newHeight/2);

// 可以开始确定 QImage 了
QImage newImage=QImage(newWidth,newHeight,img_new.format());

/*---- newImage 的画布已经准备就绪，现在开始绘制 ~~~*/
// 逆向映射，判断、并确定像素值
int i,j; // 整数部分
float u,v; // 小数部分
for(int newX=0;newX<newWidth;newX++)
{
    for(int newY=0;newY<newHeight;newY++)
    {
        i=(int)(newX*(fcos)+newY*(fsin)+(-c*fcos-d*fsin+a));
        j=(int)(newX*(-fsin)+newY*(fcos)+(c*fsin-d*fcos+b));
        u=(float)(newX*(fcos)+newY*(fsin)+(-c*fcos-d*fsin+a)-i);
        v=(float)(newX*(-fsin)+newY*(fcos)+(c*fsin-d*fcos+b)-j);

        // 判断很关键
        if(i>=0&&i<width-2&&j>=0&&j<height-2)
        {
            // 双线性插值
            QColor temp00=img_new.pixel(i,j);
            QColor temp10=img_new.pixel(i+1,j);
            QColor temp01=img_new.pixel(i,j+1);
            QColor temp11=img_new.pixel(i+1,j+1);
            int rr,gg,bb;
            rr=(int)((1-u)*(1-v)*temp00.red()+(u)*(1-v)*temp10.red()+(1-u)*(v)*temp01.red()+(u)*(v)*temp11.red());
            gg=(int)((1-u)*(1-v)*temp00.green()+(u)*(1-v)*temp10.green()+(1-u)*(v)*temp01.green()+(u)*(v)*temp11.green());
            bb=(int)((1-u)*(1-v)*temp00.blue()+(u)*(1-v)*temp10.blue()+(1-u)*(v)*temp01.blue()+(u)*(v)*temp11.blue());
            if(rr>255){ rr=255; }
            if(gg>255){ gg=255; }
            if(bb>255){ bb=255; }
            if(rr<0){ rr=0; }
            if(gg<0){ gg=0; }
        }
    }
}

```

```

        if(bb<0){ bb=0;}
        newImage.setPixelColor(newX,newY,QColor(rr,gg,bb));
    }
    else{
        // 用白色来填充好了
        newImage.setPixelColor(newX,newY,QColor(255,255,255));
    }
}
img_out=newImage;
}

```

## 2.6 图像翻转

### 2.6.1 图像翻转分析

1. 从主界面中获取图像翻转的方向，可进行多次图像翻转，实时调节
2. 为保证图像翻转后其余操作的连续性，对数据源头二维数组进行操作（操作 `**grayData_new`）
3. 针对不同的翻转对二维数组中的值进行不同的交换

### 2.6.2 图像翻转核心代码

#### 2.6.2.1 水平翻转

```

void ImageProcess::flip_horizontal()
{
    ushort temp;
    for(int j=0;j<(int)Height;j++)
    {
        for(int i=0;i<(int)Width/2;i++)
        {
            temp=grayData_new[i][j];
            grayData_new[i][j]=grayData_new[Width-i-1][j];
            grayData_new[Width-i-1][j]=temp;
        }
    }
}

```

#### 2.6.2.2 坚直翻转

```

void ImageProcess::flip_vertical()
{
    ushort temp;
    for(int i=0;i<(int)Width;i++)
    {
        for(int j=0;j<(int)Height/2;j++)
        {

```

```

temp=grayData_new[i][j];
grayData_new[i][j]=grayData_new[i][Height-1-j];
grayData_new[i][Height-1-j]=temp;
}
}
}

```

### 2.6.2.3 对角翻转

```

void ImageProcess::flip_diagonal()
{
    flip_horizontal();
    flip_vertical();
}

```

## 2.7 图像锐化/细节增强

### 2.7.1 图像锐化/细节增强分析

1. 从主界面中获取图像锐化/细节增强的参数, 点击“锐化”按钮, 可进行多次图像锐化/细节增强
2. 为保证图像锐化/细节增强后其余操作的连续性, 对数据源头二维数组进行操作(操作 `**grayData`), 强化后的图像数据存储在 `**grayData_new` 中
3. 针对不同的锐化/细节增强对二维数组中的值进行不同的计算

本软件中的锐化方法主要运用了较为简单的空间域的锐化方法, 使用了 laplace 算子, 对图像进行处理。可以证明, 最简单的各向同性导数算子是拉普拉斯算子, 它的定义为:

$$\nabla^2 f(x, y) = \frac{\partial^2 f(x, y)}{\partial x^2} + \frac{\partial^2 f(x, y)}{\partial y^2} \quad (2.1)$$

由于任意阶导数都是线性算子, 所以拉普拉斯算子也是线性算子。为了以离散形式表示拉普拉斯算子, 我们可以用下面的离散形式: 在 x 方向上:

$$\frac{\partial^2 f(x, y)}{\partial x^2} = f(x+1, y) - 2f(x, y) + f(x-1, y) \quad (2.2)$$

类似地, 在 y 方向上:

$$\frac{\partial^2 f(x, y)}{\partial y^2} = f(x, y+1) - 2f(x, y) + f(x, y-1) \quad (2.3)$$

所以, 拉普拉斯算子的离散形式为:

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \quad (2.4)$$

类似地, 考虑对角项后可以得到更多拉普拉斯核。

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

拉普拉斯锐化图像的基本方法:

$$g(x, y) = f(x, y) + c[\nabla^2 f(x, y)] \quad (2.5)$$

本实验采用第 4 个矩阵进行处理, 此处的 c 为用户提供的参数

## 2.7.2 图像锐化/细节增强核心代码

```

void ImageProcess::enhanceImage(int grade)
{
    for(int j=0;j<(int)Height;j++)
    {
        for(int i=0;i<(int)Width;i++)
        {
            if(i==0||i==(Width)-1||j==0||j==(Height)-1)
            {
                grayData_new[i][j]=grayData[i][j];
            }
            else
            {
                ushort enhance=grade*getLaplace(i,j);
                if(enhance<5*grade*grayData[i][j])
                {
                    grayData_new[i][j]=grayData[i][j]+enhance;
                }
                else
                {
                    grayData_new[i][j]=grayData[i][j];
                }
                //grayData_new[i][j]=grayData[i][j]+grade*enhance;
            }
        }
    }
}

```

```

ushort ImageProcess::getLaplace(int i, int j)
{
    ushort SuanZi;
    SuanZi=(-grayData[i-1][j-1]-grayData[i-1][j]-grayData[i-1][j+1]
             -grayData[i][j-1]+8*grayData[i][j]-grayData[i][j+1]
             -grayData[i+1][j-1]-grayData[i+1][j]-grayData[i+1][j+1]);
    return SuanZi;
}

```

## 2.8 保存图像

### 2.8.1 保存图像分析

1. 保存的图像为当前显示的图像
2. 保存的文件格式为 bmp 格式 (图像命名为当前时间), 在应用的同一目录下建立一个 “output” 文件夹, 将保存的图像放入其中

### 2.8.2 保存图像核心代码

```

void ImageProcess::saveImage()
{
    QString fileName;
    QPixmap pix;
    QString strDir = QCoreApplication::applicationDirPath()+"\\导出图像\\";
    QDir dir(strDir);

    if(!dir.exists())
    {
        dir.mkdir(strDir);
    }
    fileName = strDir + QDateTime::currentDateTime().toString("yyyy-MM-dd-HH-mm-ss") + ".bmp";

    pix = QPixmap::fromImage(img_out);
    QString strFile = QFileDialog::getSaveFileName(this,"保存图片",fileName,"PNG (*.png);;BMP (*.bmp);;JPG (*.jpg)");
    if(!strFile.isNull())
    {
        pix.save(strFile);
    }
}

```

## 2.9 重置与恢复原图

### 2.9.1 重置与恢复原图分析

1. 因为采用的是实时监测参数变化图像的方法，所以特地设置了重置和恢复原图的按钮，方便快速调节，同时这也体现了一定的撤销功能
2. 为了给用户提供便利，给每一个可调节的参数都设置了一个重置按钮，点击重置按钮可以将参数重置为默认值
3. 重置：将图像恢复到原始状态
4. 恢复原图：将图像恢复到最初打开的状态

### 2.9.2 重置与恢复原图核心代码

```

//重置
void on_pushButton_reset_wL_pressed();
void on_pushButton_reset_ww_pressed();
void on_pushButton_reset_S_pressed();
void on_pushButton_reset_R_pressed();
void on_pushButton_reset_Flip_pressed();
//恢复原图
void on_pushButton_Origin_pressed();

```

```

void MainWindow::on_pushButton_reset_wL_pressed()

```

```

{
    ui->lineEdit_windowLevel->setText(QString::number(2048));
}

void MainWindow::on_pushButton_reset_wW_pressed()
{
    ui->lineEdit_windowWidth->setText(QString::number(4096));
}

void MainWindow::on_pushButton_reset_S_pressed()
{
    ui->lineEdit_scaling->setText(QString::number((float)10/10,'f',1));
}

void MainWindow::on_pushButton_reset_R_pressed()
{
    ui->lineEdit_rote->setText(QString::number(0));
}

void MainWindow::on_pushButton_reset_Flip_pressed()
{
    forImageProcess->origin();
    forImageProcess->enhanceImage(ui->lineEdit_enhance->text().toInt());
    changeImage();
}

```

## 2.10 操作日志

### 2.10.1 操作日志分析

1. 为满足用户的撤销需求可对操作进行记录，同时也为用户提供了操作的历史记录，便于选择合适的操作图像进行观察
2. 日志记录的数据包括：序号、窗宽、窗位、放缩、旋转中心、锐化、反转。
3. 操作数据的存档采用手动存档的方式（因为数据是实时检测成像，此处存档进行了简化）点击存“数据存档”按钮即可
4. 对存档的数据进行了编号和显示，双击即可读取数据，方便用户选择
5. 存档数据可以进行“全部清除”操作。点击“全部清除”按钮即可
6. 操作日志可以记录更多信息且有拓展性，但是由于时间原因，此处只实现了基本功能

### 2.10.2 操作日志核心代码

#### 2.10.2.1 数据存档

```

void MainWindow::on_pushButton_Current_pressed()
{
    int rowcount=ui->tableView_log->rowCount();
    ui->tableView_log->insertRow(rowcount); //新增行
}

```

```

ui->tableWidget_log->setItem(rowcount,0,new QTableWidgetItem(QString::number(rowcount)));
ui->tableWidget_log->setItem(rowcount,1,new QTableWidgetItem(ui->lineEdit_windowLevel->text()));
ui->tableWidget_log->setItem(rowcount,2,new QTableWidgetItem(ui->lineEdit_windowWidth->text()));
ui->tableWidget_log->setItem(rowcount,3,new QTableWidgetItem(ui->lineEdit_scaling->text()));
ui->tableWidget_log->setItem(rowcount,4,new QTableWidgetItem(ui->lineEdit_rrote->text()));
ui->tableWidget_log->setItem(rowcount,5,new QTableWidgetItem(ui->lineEdit_enhance->text()));
ui->tableWidget_log->setItem(rowcount,6,new QTableWidgetItem(QString::number(flag_inverse)));
}

```

## 2.10.2.2 双击读取

```

void MainWindow::on_tableWidget_log_itemDoubleClicked(QTableWidgetItem *item)
{
    int row=item->row();

    flag_inverse=ui->tableWidget_log->item(row,6)->text().toInt();

    ui->lineEdit_windowLevel->setText(ui->tableWidget_log->item(row,1)->text());
    ui->lineEdit_windowWidth->setText(ui->tableWidget_log->item(row,2)->text());
    ui->lineEdit_scaling->setText(ui->tableWidget_log->item(row,3)->text());
    ui->lineEdit_rrote->setText(ui->tableWidget_log->item(row,4)->text());
    ui->lineEdit_enhance->setText(ui->tableWidget_log->item(row,5)->text());
    on_pushButton_Enhance_pressed();
}

```

## 2.10.2.3 全部清除

```

void MainWindow::on_pushButton_Clear_log_pressed()
{
    int row_all=ui->tableWidget_log->rowCount();
    for(int i=0;i<=row_all;++i){
        ui->tableWidget_log->removeRow(0); //每次都把第一行删除
    }
}

```

# 第3章 演示和测试

## 3.1 程序界面

左侧右侧为操作，右侧为显示区。

操作区的操作主要为按钮操作，文本框输入和滑动条操作。

“窗宽”、“窗位”、“放缩”、“旋转”都是实时显示的，只要改变对应文本框数值即可（推动滑动条亦然）。

“锐化”：需要点击“锐化”按钮，才会对图像进行锐化操作。

所有操作之间不会产生冲突（不会无端重置），即可以同时进行多个操作，进行连续的变化。随时可以进行数据存档，便于选择合适的操作图像进行观察

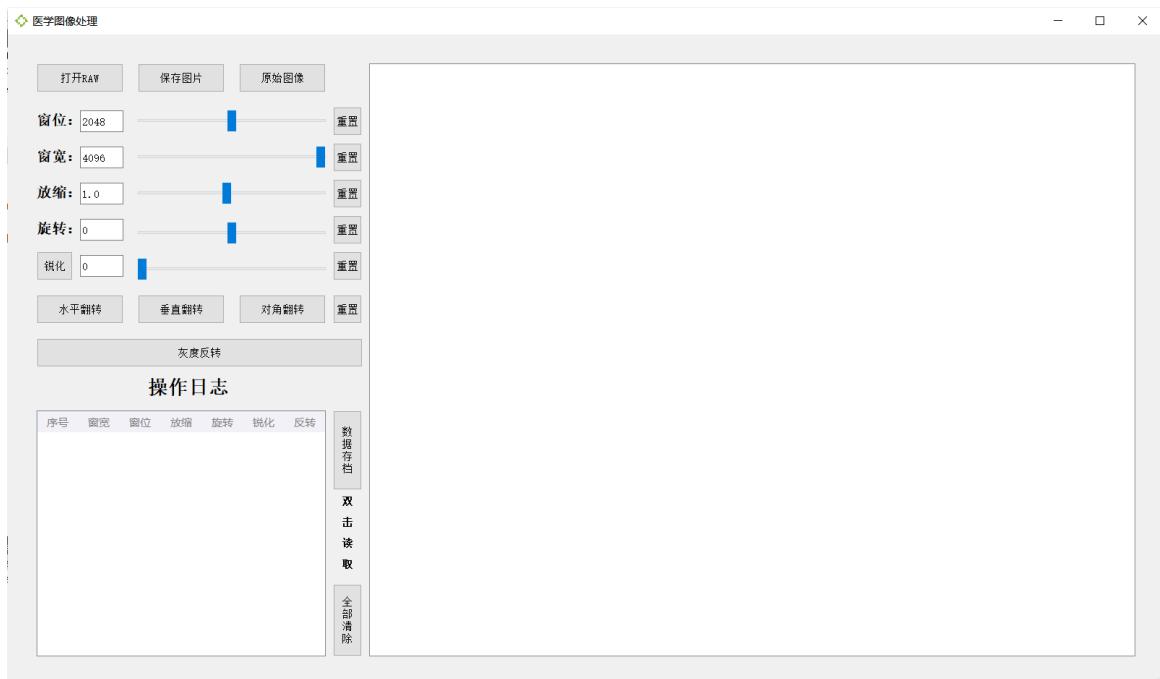
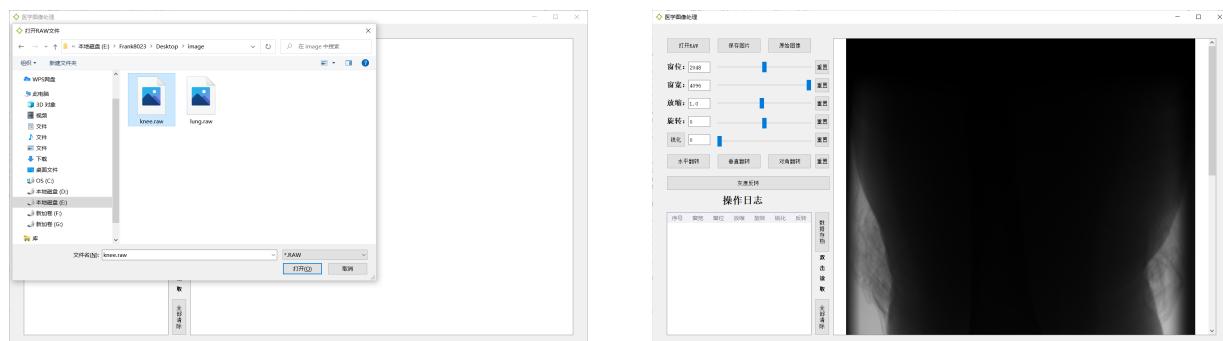


图 3.1: 程序界面

## 3.2 读取 RAW 图像

注意：选取的图像规定为 RAW 格式，文件路径不得包含中文。



(a) 点击打开选择文件

(b) 确定显示

图 3.2: 读取 RAW 图像

### 3.3 灰度窗调整

灰度窗调整是指调整图像的窗宽和窗位，使得图像的灰度值分布在合适的范围内，使得图像的细节更加清晰。



图 3.3: 灰度窗宽窗位调整

### 3.4 灰度反转

knee.RAW 为例，设置的窗位和窗宽为 [250,500] 后由交为合适的图像显示点击灰度反转后，为仍然为合适的图像显示，此处对应的窗位值也做了改变

4095 – 窗位值

可以多次点击灰度反转，或在此基础上进行参数调整。这里同时使用了数据存档的功能，可以随时选择合适的图像进行观察

灰度反转后的图像如下：

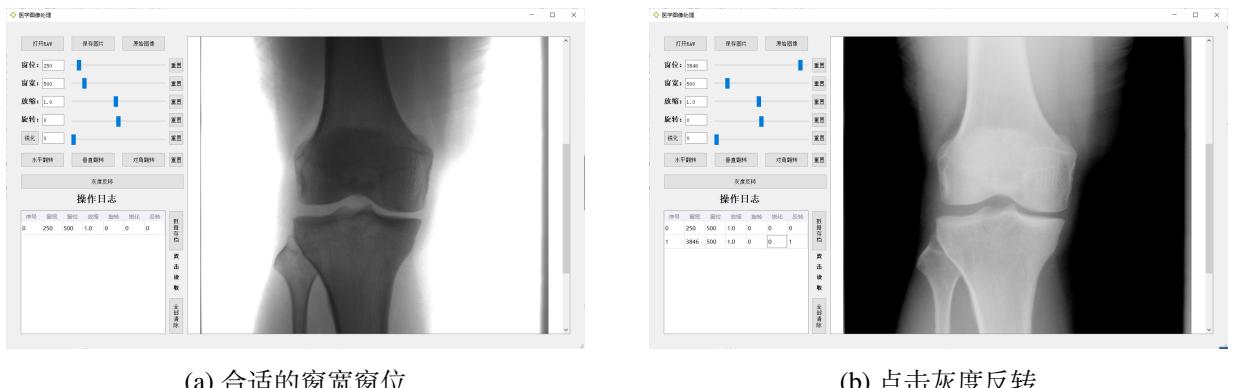


图 3.4: 灰度反转

### 3.5 图像放缩与旋转

在任何状态下，都可以进行图像的放缩和旋转，放缩和旋转的参数都是实时显示的，可以随时进行调整、进行数据存档。

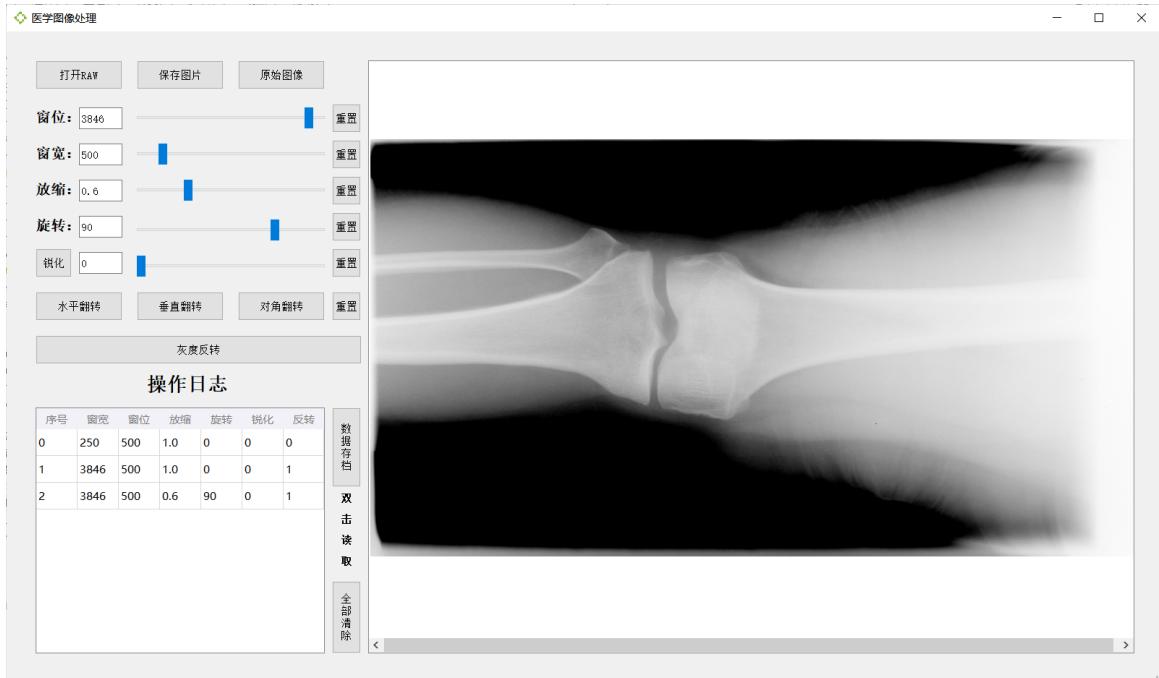


图 3.5: 图像放缩与旋转

## 3.6 图像翻转

在任何状态下，都可以进行图像的翻转，点击相应反转按钮即可。

此处的翻转仅供临时的观察，在进行其他操作时并不会体现，数据存档也没有记录这样的设计可能有一点不全面，但实际操作中也是合理的满足需求的。

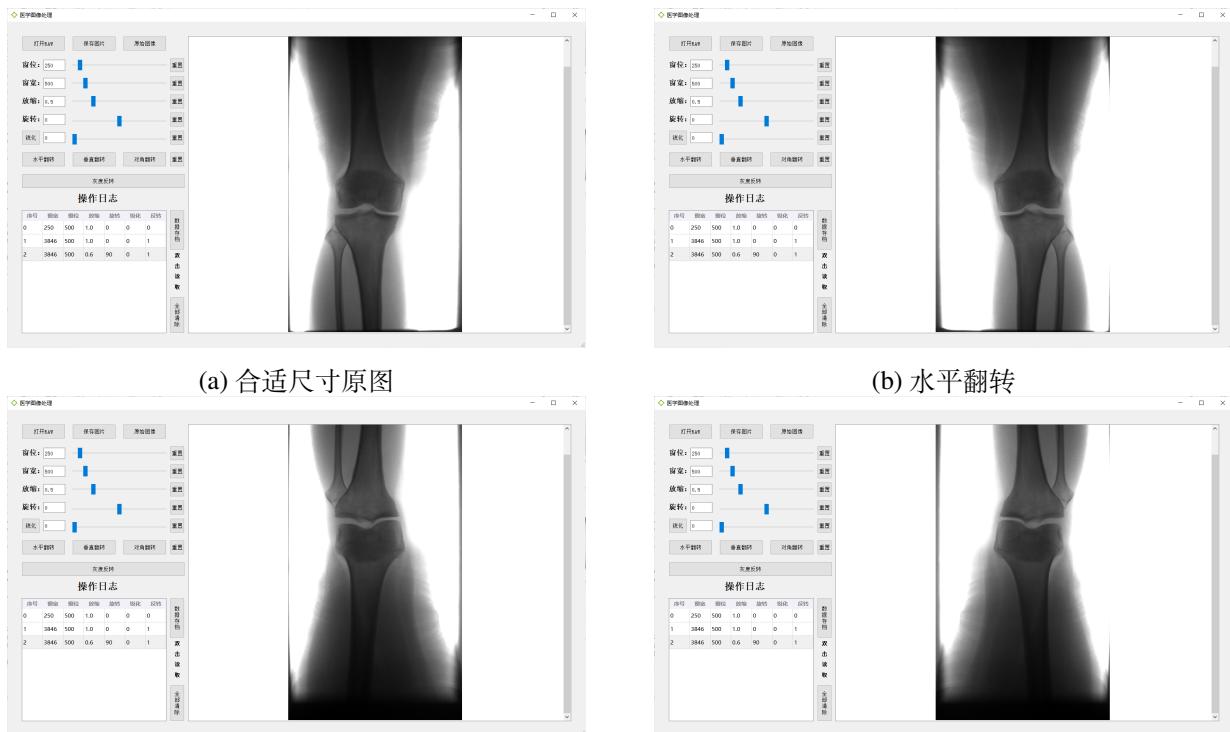


图 3.6: 图像翻转

## 3.7 图像锐化/细节增强

在文本框中输入锐化/细节增强的参数或拉动滚动条后，点击“锐化”按钮即可对图像进行锐化。锐化强度为0~10级。

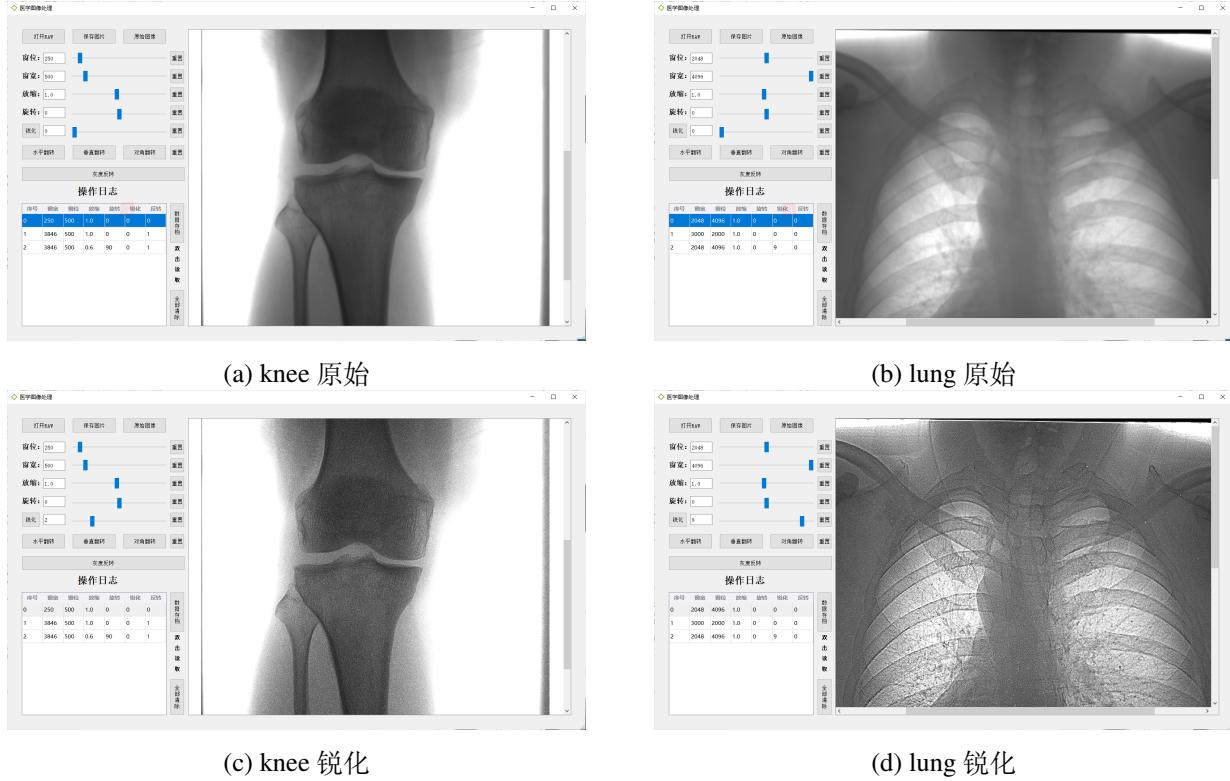


图 3.7: 图像锐化/细节增强

## 3.8 图像保存

点击“图像保存”按钮即可对图像进行保存，默认保存为 bmp 格式，保存路径为当前路径下的“output”文件夹中。

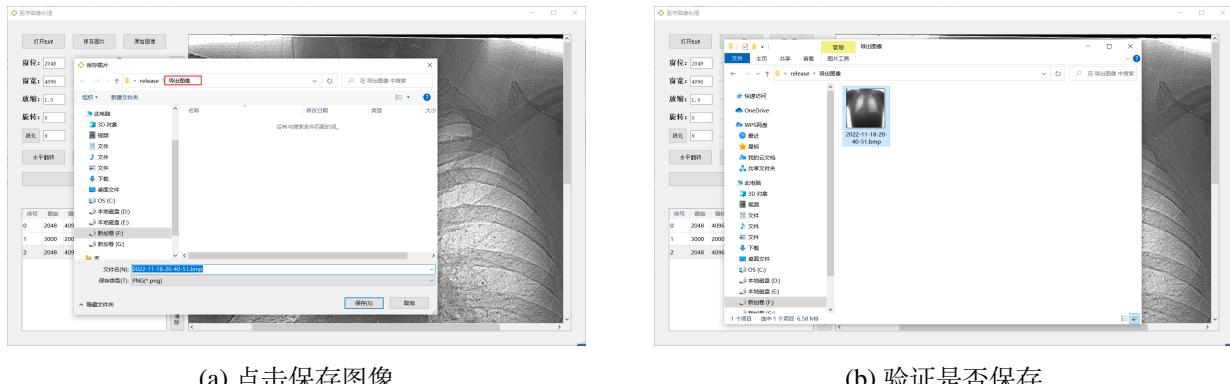


图 3.8: 图像保存

## 3.9 其余操作

### 3.9.1 重置

点击对应的“重置”按钮对应行的数据进行重置，重置为以下默认值。

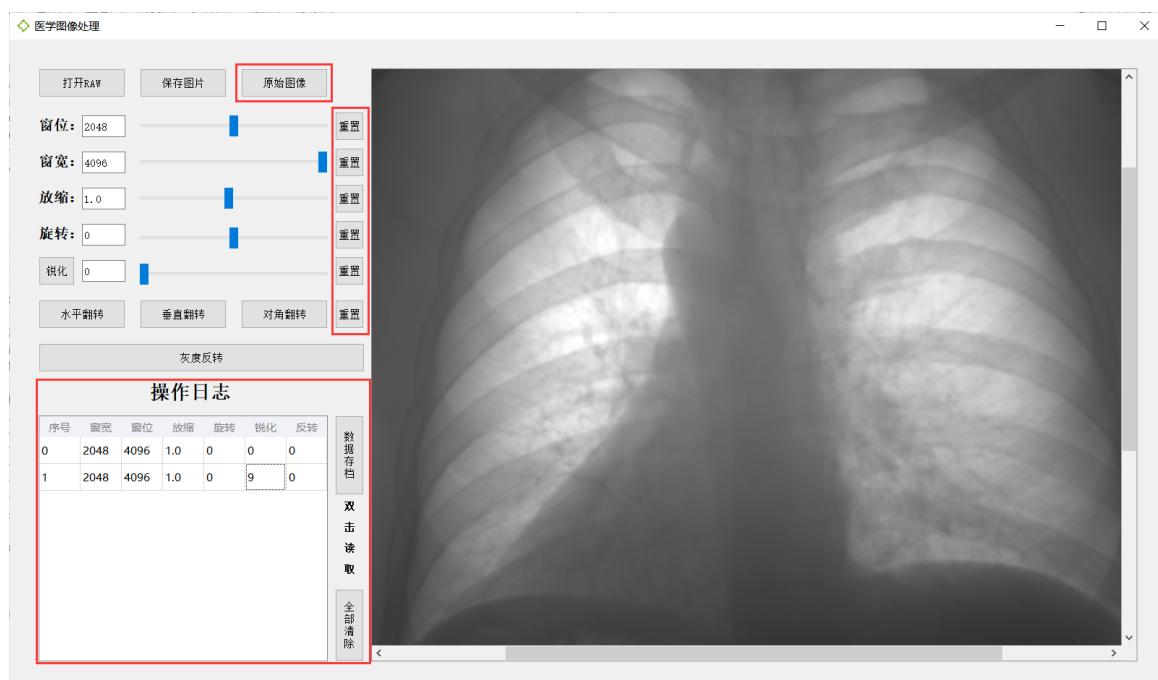
- 窗位：2048
- 窗宽：4096
- 放缩：1
- 旋转：0
- 锐化：0
- 翻转：无

### 3.9.2 原始图像

点击“原始图像”按钮，可以将图像恢复到原始状态，即重置所有操作。

### 3.9.3 操作日志

在操作日志面板中，可以查看所有存档数据的记录，包括：点击“数据存档”按钮当前数据进行存档，包括窗位、窗宽、放缩、旋转、锐化、反转。



## 附录 A 值得一提的操作

### A.1 给运行程序一个图标

- 在项目添加 images 文件
- images 文件中包含
  - logo.ico 图标
  - logo.rc 文件)

```
IDI_ICON1 ICON DISCARDABLE "logo.ico"
```

- 在.pro 中添加

```
DISTFILES += \
    images/logo.ico \
    images/logo.rc

RC_FILE += images/logo.rc
```

### A.2 封装成.exe 文件

参考QT 软件打包为一个单独可执行.exe 文件流程

1. 选择“Release”版本，设置路径；编译或者“run”；
2. 把 release 文件夹下的 demo.exe 拷贝到任意一个新文件夹里 test。
3. 打开 Qt5.12.10MinGW7.3.0.064 – bit)
4. 输入命令：cd (空格) /d(空格) demo.exe 路径；
5. 输入命令：windeployqt + demo.exe, 回车；
6. 查看 test 文件夹下是否多了很多文件，点击 demo.exe 是否可运行；

完成之后可以看到文件夹中多了很多程序执行的依赖文件，这里生成的 demo.exe 已经可以运行了。但是这样可执行程序有很多依赖选项，如果需要把程序给其他人使用就需要发整个文件夹，很麻烦。接下来我们再次把整个文件夹打包为一个单一的可执行.exe 文件

下载使用 Enigma virtual box。官方链接：<https://enigmaprotector.com/en/downloads.html>

对文件进一步进行压缩。