围岩图像节理数自动检测系统 V1.0 源代码

```
// pro 项目管理文件
QT += core gui
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
TARGET = **** #  应用程序名
TEMPLATE = app       #  模板类型  应用程序

SOURCES += \
      main.cpp \
      mainwindow.cpp

HEADERS += \
      mainwindow.h
FORMS += mainwindow.ui
INCLUDEPATH += D:\app\Qt\5.14.2\mingw73_64\include\OpenCV
LIBS += D:\app\Qt\5.14.2\mingw73_64\lib\libopencv_*.a

RESOURCES += \
      myimage.qrc
RC_ICONS = picture.ico
TRANSLATIONS += languages\English.ts\
                      languages\Chinese.ts

SUBDIRS += \
      MainWindow.pro

//源程序文件 main.cpp
int main(int argc, char *argv[])
{
      QApplication a(argc, argv); //  定义并创建应用程序
      QTranslator translator;
      translator.load(NULL);
      a.installTranslator(&translator);
      QPixmap pixmap(":/myImage/images/loading.gif");
      QSplashScreen splash(pixmap);
      QLabel label(&splash);
      QMovie mv(":/myImage/images/loading.gif");
      label.setMovie(&mv);
      mv.start();
      splash.show();
      splash.setCursor(Qt::BlankCursor);
      for(int i=0; i<5000; i+=mv.speed())
      {
            QCoreApplication::processEvents();
            usleep(500*static_cast<useconds_t>(mv.speed()));
      }
```

```
    MainWindow w; // 定义并创建窗口
    w.setWindowTitle(QObject::tr("围岩图像节理数自动检测系统"));
    w.show(); // 显示窗口
    splash.finish(&w);
    return a.exec(); // 应用程序运行，开始消息循环和事件处理
}

// 源程序文件 mainwindow.cpp
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    QTimer *timer_calendar;
    timer_calendar = new QTimer(this);//当前时间显示
    timer_calendar->start(1000);
    connect(timer_calendar, SIGNAL(timeout()), this, SLOT(timerUpdate()));
    connect(ui->doubleSpinBox, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_K1, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_K2, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_K3, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_d, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_d1, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->horizontalSlider_E, SIGNAL(valueChanged()),this, SLOT(verticalSliderValueChanged(int)));
    ui->horizontalSlider_E->setRange(8, 18);
    connect(ui->horizontalSlider_V, SIGNAL(valueChanged()),this, SLOT(verticalSliderValueChanged(int)));
    ui->horizontalSlider_V->setRange(10, 20);
    connect(ui->doubleSpinBox_delta, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_r, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_L, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_S, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_m, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_x, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_miu, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_p, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    connect(ui->doubleSpinBox_area, SIGNAL(valueChanged()), this, SLOT(slotDoubleSpinBox()));
    ui->pushButton_3->setDisabled(true);
    ui->pushButton_4->setDisabled(true);
    setWindowFlags(windowFlags()&~Qt::WindowMaximizeButtonHint);
    setFixedSize(this->width(),this->height());
    customMsgBox.setWindowTitle(tr("关于本软件"));
    customMsgBox.addButton(tr("好的"),QMessageBox::ActionRole);
    customMsgBox.setIconPixmap(QPixmap(":/myImage/images/about1.png"));
    customMsgBox.setText(tr("欢迎使用《围岩图像节理数自动检测系统》！本软件具有简单的围岩图像节
理数自动检测功能。\n"));
    ui->statusBar->showMessage(tr("欢迎使用围岩图像节理数自动检测系统"),2000);
    QLabel *permanent = new QLabel(this);
    permanent->setObjectName("status");
    permanent->setFrameStyle(QFrame::Box|QFrame::Sunken);
    permanent->setText("欢迎使用！ ");
```

```
    ui->statusBar->addPermanentWidget(permanent);
    ui->tabWidget->setStyleSheet("QTabWidget:pane {border-top:0px;background:    transparent; }");
}
MainWindow::~MainWindow()
{
    delete ui;
}
void MainWindow::on_action_Dock_triggered()
{
    ui->dockWidget->show();
}
void MainWindow::timerUpdate()//显示时间函数
{
    QDateTime time = QDateTime::currentDateTime();
    ui->label_time->setText(time.toString("yyyy-MM-dd hh:mm:ss"));
}
void MainWindow::on_action_Open_triggered()
{
    QStringList srcDirPathListS = QFileDialog::getOpenFileNames(this,tr("选择图片"),"/images",tr("图像文件
(*.jpg *.png *.bmp *.tif)"));
    if(srcDirPathListS.size()>0)
    {
        ui->tabWidget->setCurrentIndex(0);
    }

    if(srcDirPathListS.size()>=3)
    {
        srcDirPathList =srcDirPathListS;
        srcDirPathListS.clear();
        index =0;
        QString srcDirPath = srcDirPathList.at(index);
        QImage image(srcDirPath);
        global_img = image.copy();
        back_img = image.copy();
        QImage Image=ImageCenter(image,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        origin_path=srcDirPath;
        QImage images=ImageCenter(image,ui->label_other);
        ui->label_other->setPixmap(QPixmap::fromImage(images));
        ui->label_other->setAlignment(Qt::AlignCenter);
        //状态栏显示图片路径
        QLabel *label=ui->statusBar->findChild<QLabel *>("status");
        label->setText(srcDirPath);
        QString src1 = srcDirPathList.at((index+1)%srcDirPathList.size());
        QImage image1(src1);
        QImage Image1 = ImageCenter(image1,ui->label_other_1);
        ui->label_other_1->setPixmap(QPixmap::fromImage(Image1));
        ui->label_other_1->setAlignment(Qt::AlignCenter);
        QString src2 = srcDirPathList.at((index+2)%srcDirPathList.size());
        QImage image2(src2);
```

```
        QImage Image2 = ImageCenter(image2,ui->label_other_3);
        ui->label_other_3->setPixmap(QPixmap::fromImage(Image2));
        ui->label_other_3->setAlignment(Qt::AlignCenter);
        ui->pushButton_3->setDisabled(false);
        ui->pushButton_4->setDisabled(false);
        ui->label_other_1->setVisible(true);
        ui->label_other_3->setVisible(true);
    }
    else if(srcDirPathListS.size()==1)
    {
        srcDirPathList =srcDirPathListS;
        srcDirPathListS.clear();
        index =0;
        QString srcDirPath = srcDirPathList.at(index);
        QImage image(srcDirPath);
        global_img = image.copy();
        back_img = image.copy();
        QImage Image=ImageCenter(image,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        origin_path=srcDirPath;
        QImage images=ImageCenter(image,ui->label_other);
        ui->label_other->setPixmap(QPixmap::fromImage(images));
        ui->label_other->setAlignment(Qt::AlignCenter);
        //状态栏显示图片路径
        QLabel *label=ui->statusBar->findChild<QLabel *>("status");
        label->setText(srcDirPath);
        //有图片触发事件
        //isImage=true;
        //qDebug("%d",srcDirPathList.size());
        ui->pushButton_3->setDisabled(true);
        ui->pushButton_4->setDisabled(true);
        ui->label_other_3->setVisible(false);
        ui->label_other_1->setVisible(false);
    }
    else if(srcDirPathListS.size()==2)
    {
        srcDirPathList =srcDirPathListS;
        srcDirPathListS.clear();
        index =0;
        QString srcDirPath = srcDirPathList.at(index);
        QImage image(srcDirPath);
        global_img = image.copy();
        back_img = image.copy();
        QImage Image=ImageCenter(image,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        origin_path=srcDirPath;
        QImage images=ImageCenter(image,ui->label_other);
        ui->label_other->setPixmap(QPixmap::fromImage(images));
        ui->label_other->setAlignment(Qt::AlignCenter);
```

```
            //状态栏显示图片路径
            QLabel *label=ui->statusBar->findChild<QLabel *>("status");
            label->setText(srcDirPath);
            //有图片触发事件
            //isImage=true;
            //qDebug("%d",srcDirPathList.size());
            QString src1 = srcDirPathList.at((index+1)%srcDirPathList.size());
            QImage image1(src1);
            QImage Image1 = ImageCenter(image1,ui->label_other_1);
            ui->label_other_1->setPixmap(QPixmap::fromImage(Image1));
            ui->label_other_1->setAlignment(Qt::AlignCenter);
            ui->pushButton_3->setDisabled(false);
            ui->pushButton_4->setDisabled(false);
            ui->label_other_1->setVisible(true);
            ui->label_other_3->setVisible(false);
        }
}

void split(const string& s,vector<int>& sv,const char flag = ' ')
{
    sv.clear();
    istringstream iss(s);
    string temp;
    while (getline(iss, temp, flag)) {
        sv.push_back(stoi(temp));
    }
    return;
}

//图片居中显示,图片大小与 label 大小相适应
QImage MainWindow::ImageCenter(QImage    qimage,QLabel *qLabel)
{
    QImage image;
    QSize imageSize = qimage.size();
    QSize labelSize = qLabel->size();
    double dWidthRatio = 1.0*imageSize.width() / labelSize.width();
    double dHeightRatio = 1.0*imageSize.height() / labelSize.height();

    if (dWidthRatio>dHeightRatio) { image = qimage.scaledToWidth(labelSize.width());}
    else {image = qimage.scaledToHeight(labelSize.height());}

    return image;
}


//上一张
void MainWindow::on_pushButton_3_clicked()
{
    if(srcDirPathList.size()>=3)
    {
    index=qAbs(index+srcDirPathList.size()-1);
```

```
int i = index%srcDirPathList.size();
QString srcDirPath = srcDirPathList.at(i);
QImage image(srcDirPath);

global_img = image.copy();
back_img = image.copy();

QImage Image=ImageCenter(image,ui->label_show);
ui->label_show->setPixmap(QPixmap::fromImage(Image));
ui->label_show->setAlignment(Qt::AlignCenter);
origin_path=srcDirPath;
QImage images3=ImageCenter(image,ui->label_other);
ui->label_other->setPixmap(QPixmap::fromImage(images3));
ui->label_other->setAlignment(Qt::AlignCenter);
//状态栏显示图片路径
QLabel *label=ui->statusBar->findChild<QLabel *>("status");
label->setText(srcDirPath);

QString src1 = srcDirPathList.at(qAbs(index+srcDirPathList.size()-1)%srcDirPathList.size());
QImage image1(src1);
QImage Image1 = ImageCenter(image1,ui->label_other_1);
ui->label_other_1->setPixmap(QPixmap::fromImage(Image1));
ui->label_other_1->setAlignment(Qt::AlignCenter);

QString src2 = srcDirPathList.at(qAbs(index+srcDirPathList.size()-2)%srcDirPathList.size());
QImage image2(src2);
QImage Image2 = ImageCenter(image2,ui->label_other_3);
ui->label_other_3->setPixmap(QPixmap::fromImage(Image2));
ui->label_other_3->setAlignment(Qt::AlignCenter);
}
else if(srcDirPathList.size()==2)
{
    index=qAbs(index+srcDirPathList.size()-1);
    int i = index%srcDirPathList.size();
    //qDebug("%d",i);
    QString srcDirPath = srcDirPathList.at(i);
    QImage image(srcDirPath);

    global_img = image.copy();
    back_img = image.copy();

    QImage Image=ImageCenter(image,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    origin_path=srcDirPath;
    QImage images3=ImageCenter(image,ui->label_other);
    ui->label_other->setPixmap(QPixmap::fromImage(images3));
    ui->label_other->setAlignment(Qt::AlignCenter);
    //状态栏显示图片路径
    QLabel *label=ui->statusBar->findChild<QLabel *>("status");
    label->setText(srcDirPath);
```

```
            QString src1 = srcDirPathList.at(qAbs(index+srcDirPathList.size()-1)%srcDirPathList.size());
            QImage image1(src1);
            QImage Image1 = ImageCenter(image1,ui->label_other_1);
            ui->label_other_1->setPixmap(QPixmap::fromImage(Image1));
            ui->label_other_1->setAlignment(Qt::AlignCenter);
        }
}


//下一张
void MainWindow::on_pushButton_4_clicked()
{

    if(srcDirPathList.size()>=3)
    {
    index=qAbs(index+1);
    int i = index%srcDirPathList.size();
   // qDebug("%d",i);
    QString srcDirPath = srcDirPathList.at(i);
    QImage image(srcDirPath);
    global_img = image.copy();
    back_img = image.copy();
    QImage Image=ImageCenter(image,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    origin_path=srcDirPath;
    QImage images1=ImageCenter(image,ui->label_other);
    ui->label_other->setPixmap(QPixmap::fromImage(images1));
    ui->label_other->setAlignment(Qt::AlignCenter);
    //状态栏显示图片路径
    QLabel *label=ui->statusBar->findChild<QLabel *>("status");
    label->setText(srcDirPath);
    QString src1 = srcDirPathList.at((index+1)%srcDirPathList.size());
    QImage image1(src1);
    QImage Image1 = ImageCenter(image1,ui->label_other_1);
    ui->label_other_1->setPixmap(QPixmap::fromImage(Image1));
    ui->label_other_1->setAlignment(Qt::AlignCenter);
    QString src2 = srcDirPathList.at((index+2)%srcDirPathList.size());
    QImage image2(src2);
    QImage Image2 = ImageCenter(image2,ui->label_other_3);
    ui->label_other_3->setPixmap(QPixmap::fromImage(Image2));
    ui->label_other_3->setAlignment(Qt::AlignCenter);
    }
    else if(srcDirPathList.size()==2)
    {
        index=qAbs(index+1);
        int i = index%srcDirPathList.size();
        QString srcDirPath = srcDirPathList.at(i);
        QImage image(srcDirPath);
        global_img = image.copy();
        back_img = image.copy();
```

```
            QImage Image=ImageCenter(image,ui->label_show);
            ui->label_show->setPixmap(QPixmap::fromImage(Image));
            ui->label_show->setAlignment(Qt::AlignCenter);
            origin_path=srcDirPath;
            QImage images1=ImageCenter(image,ui->label_other);
            ui->label_other->setPixmap(QPixmap::fromImage(images1));
            ui->label_other->setAlignment(Qt::AlignCenter);
            //状态栏显示图片路径
            QLabel *label=ui->statusBar->findChild<QLabel *>("status");
            label->setText(srcDirPath);
            QString src1 = srcDirPathList.at((index+1)%srcDirPathList.size());
            QImage image1(src1);
            QImage Image1 = ImageCenter(image1,ui->label_other_1);
            ui->label_other_1->setPixmap(QPixmap::fromImage(Image1));
            ui->label_other_1->setAlignment(Qt::AlignCenter);
        }
}


//灰度化
QImage MainWindow::gray(QImage image)
{
    QImage newImage =image.convertToFormat(QImage::Format_ARGB32);
    QColor oldColor;
        for(int y = 0; y < newImage.height(); y++)
        {
            for(int x = 0; x < newImage.width(); x++)
            {
                oldColor = QColor(image.pixel(x,y));
                int average = (oldColor.red() + oldColor.green() + oldColor.blue()) / 3;
                newImage.setPixel(x, y, qRgb(average, average, average));
            }
        }
        return newImage;
}


//灰度化
void MainWindow::on_pushButton_gray_clicked()
{
    QImage image = global_img.copy();
    back_img = image.copy();
    QImage images=gray(image);
    QImage Image=ImageCenter(images,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    global_img = images.copy();
}


//均值滤波
```

```
QImage MainWindow::junzhi(QImage image)
{
    int kernel [3][3] = {{1,1,1},{1,1,1},{1,1,1}};
        int sizeKernel = 3;
        int sumKernel = 9;
        QColor color;
         for(int x = sizeKernel/2;x<image.width() - sizeKernel/2;x++)
        {
            for(int y= sizeKernel/2;y<image.height() - sizeKernel/2;y++)
             {
                    int r = 0;
                    int g = 0;
                    int b = 0;
                    for(int i = -sizeKernel/2;i<=sizeKernel/2;i++)
                    {
                        for(int j = -sizeKernel/2;j<=sizeKernel/2;j++)
                         {
                          color = QColor(image.pixel(x+i,y+j));
                          r += color.red()*kernel[sizeKernel/2+i][sizeKernel/2+j];
                          g += color.green()*kernel[sizeKernel/2+i][sizeKernel/2+j];
                          b += color.blue()*kernel[sizeKernel/2+i][sizeKernel/2+j];
                         }
                    }
                    r = qBound(0,r/sumKernel,255);
                    g = qBound(0,g/sumKernel,255);
                    b = qBound(0,b/sumKernel,255);
                    image.setPixel(x,y,qRgb( r,g,b));
             }
        }
         return image;
}

//均值滤波
void MainWindow::on_pushButton_junzhi_clicked()
{
    QImage image= global_img.copy();
    back_img = image.copy();
    QImage images=junzhi(image);
    QImage Image=ImageCenter(images,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("均值滤波成功！"));
    global_img = images.copy();
}

//亮度调节
void MainWindow::on_horizontalSlider_valueChanged(int value)
{
    QImage image = global_img.copy();
    back_img = image.copy();
```

```
    int red, green, blue;
        int pixels = image.width() * image.height();
        unsigned int *data = (unsigned int *)image.bits();

        for (int i = 0; i < pixels; ++i)
        {
            red= qRed(data[i])+ value;
            red = (red < 0x00) ? 0x00 : (red > 0xff) ? 0xff : red;
            green= qGreen(data[i]) + value;
            green = (green < 0x00) ? 0x00 : (green > 0xff) ? 0xff : green;
            blue= qBlue(data[i]) + value;
            blue =    (blue    < 0x00) ? 0x00 : (blue    > 0xff) ? 0xff : blue ;
            data[i] = qRgba(red, green, blue, qAlpha(data[i]));
        }
        QImage Image=ImageCenter(image,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        ui->label_light->setText(QString::number(value));
        global_img = image.copy();
}

//边缘检测
QImage MainWindow::bianyuan(QImage image)
{
    QImage newImage =image.convertToFormat(QImage::Format_ARGB32);
    QColor color0; QColor color1; QColor color2; QColor color3;
    int    r = 0; int g = 0; int b = 0; int rgb = 0; int r1 = 0; int g1 = 0; int b1 = 0; int rgb1 = 0; int a = 0;
    for( int y = 0; y < image.height() - 1; y++)
    {
        for(int x = 0; x < image.width() - 1; x++)
        {
            color0 =     QColor ( image.pixel(x,y));
            color1 =     QColor ( image.pixel(x + 1,y));
            color2 =     QColor ( image.pixel(x,y + 1));
            color3 =     QColor ( image.pixel(x + 1,y + 1));
            r = abs(color0.red() - color3.red());
            g = abs(color0.green() - color3.green());
            b = abs(color0.blue() - color3.blue());
            rgb = r + g + b;
            r1 = abs(color1.red() - color2.red());
            g1= abs(color1.green() - color2.green());
            b1 = abs(color1.blue() - color2.blue());
            rgb1 = r1 + g1 + b1;
            a = rgb + rgb1;
            a = a>255?255:a;
            newImage.setPixel(x,y,qRgb(a,a,a));
        }
    }
    return newImage;
}
```

```
//边缘检测
void MainWindow::on_pushButton_junzhi_2_clicked()
{
    QImage image= global_img.copy();
    back_img = image.copy();
    QImage newImage =bianyuan(image);
    QImage Image=ImageCenter(newImage,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    Mat m_img = QImage2cvMat(newImage);
    global_img = newImage.copy();
}

void MainWindow::on_horizontalSlider_2_valueChanged(int value1)
{
    // 检查是否有打开的图像
    if (ui->label_show->pixmap() == nullptr)
    {
        QMessageBox::warning(this, "提示", "请先打开图片！");
        return;
    }
    // 获取原始图像
    QImage images(origin_path);
    Mat image = QImage2cvMat(images);
    // 计算缩放比例
    double scaleFactor = static_cast<double>(value1) / 300.0;
    // 缩放图像
    Mat scaledImage;
    cv::resize(image, scaledImage, cv::Size(), scaleFactor, scaleFactor);
    // 将 OpenCV 图像转换为 Qt 图像
    QImage scaledQImage = cvMat2QImage(scaledImage);

    // 在标签上显示缩放后的图像
    ui->label_show->setPixmap(QPixmap::fromImage(scaledQImage));
    ui->label_show->setAlignment(Qt::AlignCenter);
}

//保存
void MainWindow::on_pushButton_save_clicked()
{
        if(ui->checkBox->isChecked()){//要加水印

            if (ui->label_show->pixmap() == nullptr)
                {
                    QMessageBox::warning(nullptr, "提示", "请先打开图片！", QMessageBox::Yes |
QMessageBox::Yes);
                    return;
                }

                QPixmap originalPixmap = *ui->label_show->pixmap();
```

```
                QImage originalImage = originalPixmap.toImage();

                if (ui->checkBox->isChecked())
                {
                    // Create a QPainter to draw the watermark text on the image
                    QPainter painter(&originalImage);
                    QFont font("Arial", 10); // Customize the font and size
                    font.setBold(true); // Make the text bold
                    painter.setFont(font);
                    painter.setPen(Qt::black); // Customize the text color
                    QString watermarkText = "[@禁止商用]";

                    // Calculate the position to center the text on the image
                    int textX = (originalImage.width() - painter.fontMetrics().width(watermarkText)) / 2;
                    int textY = (originalImage.height() - painter.fontMetrics().height()) / 2;

                    // Draw the text above the image
                    painter.drawText(textX, textY, watermarkText);
                }

                QString filename = QFileDialog::getSaveFileName(this,
                                                                tr("保存图片"),
"/myImage/images/signed_images.png",
                                                                tr("*.png;; *.jpg;; *.bmp;; *.tif;;
*.GIF"));

                if (filename.isEmpty())
                {
                    return;
                }

                if (originalImage.save(filename))
                {
                    ui->statusBar->showMessage("图片保存成功！");
                }
                else
                {
                    QMessageBox::information(this, tr("图片保存成功！"), tr("图片保存失败！"));
                }
            }
            else //不加水印
            {
                if(ui->label_show->pixmap()!=nullptr){
                    QString filename = QFileDialog::getSaveFileName(this,
                    tr("保存图片"),
                    "/myImage/images",
                    tr("*.png;; *.jpg;; *.bmp;; *.tif;; *.GIF")); //选择路径
                if (filename.isEmpty())
                {
                    return;
```

```
            }
            else
            {
                if (!(ui->label_show->pixmap()->toImage().save(filename))) //保存图像
                {

                    QMessageBox::information(this,
                        tr("图片保存成功！"),
                        tr("图片保存失败！"));
                    return;
                }
                ui->statusBar->showMessage("图片保存成功！");
            }

        }else{
            QMessageBox::warning(nullptr, "提示", "请先打开图片！", QMessageBox::Yes |
QMessageBox::Yes);
        }
        }
}

//显示原图按钮
void MainWindow::on_pushButton_origin_clicked()
{
    if(origin_path!=nullptr){
        QImage image(origin_path);
        QImage Image=ImageCenter(image,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
    }else{
        QMessageBox::warning(nullptr, "提示", "请先打开图片！", QMessageBox::Yes |
QMessageBox::Yes);
    }
}

QImage MainWindow::gamma(QImage image){
    double d=1.2;
    QColor color;
    int height = image.height();
    int width = image.width();
    for (int i=0;i<width;i++){
        for(int j=0;j<height;j++){
            color = QColor(image.pixel(i,j));
            double r = color.red();
            double g = color.green();
            double b = color.blue();
            int R = qBound(0,(int)qPow(r,d),255);
            int G = qBound(0,(int)qPow(g,d),255);
            int B = qBound(0,(int)qPow(b,d),255);
            image.setPixel(i,j,qRgb(R,G,B));
        }
```

```
    }
    return image;
}

//伽马变换按钮
void MainWindow::on_pushButton_gamma_clicked()
{
        QImage image=global_img.copy();
        back_img = image.copy();
        image=gamma(image);
        QImage Image=ImageCenter(image,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        global_img = image.copy();
}

//二值化滑动条
void MainWindow::on_horizontalSlider_erzhi_valueChanged(int value)
{
    if(origin_path!=nullptr){
    QImage image(origin_path);
    QImage images=gray(image);
    int height=images.height();
    int width=images.width();
            int bt;
            QColor oldColor;
            for (int i = 0; i < height; ++i)
            {
                for(int j=0;j<width;++j){
                    oldColor = QColor(images.pixel(j,i));
                    bt = oldColor.red();
                if(bt<value){
                    bt=0;
                }else{
                    bt=255;
                }
                images.setPixel(j,i, qRgb(bt, bt, bt));

                }
            }
    QImage Image=ImageCenter(images,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    ui->label_yuzhi->setText(QString::number(value));
    }
    else{
        QMessageBox::warning(nullptr,"提示", "请先选择一张图片！",QMessageBox::Yes |
QMessageBox::Yes);
    }
}
```

```
//调整对比度
QImage MainWindow::AdjustContrast(QImage image, int value)
{
    int pixels = image.width() * image.height();
    unsigned int *data = (unsigned int *)image.bits();

    int red, green, blue, nRed, nGreen, nBlue;

    if (value > 0 && value < 256)
    {
        float param = 1 / (1 - value / 256.0) - 1;

        for (int i = 0; i < pixels; ++i)
        {
            nRed = qRed(data[i]);
            nGreen = qGreen(data[i]);
            nBlue = qBlue(data[i]);

            red = nRed + (nRed - 127) * param;
            red = (red < 0x00) ? 0x00 : (red > 0xff) ? 0xff : red;
            green = nGreen + (nGreen - 127) * param;
            green = (green < 0x00) ? 0x00 : (green > 0xff) ? 0xff : green;
            blue = nBlue + (nBlue - 127) * param;
            blue = (blue < 0x00) ? 0x00 : (blue > 0xff) ? 0xff : blue;

            data[i] = qRgba(red, green, blue, qAlpha(data[i]));
        }
    }
    else
    {
        for (int i = 0; i < pixels; ++i)
        {
            nRed = qRed(data[i]);
            nGreen = qGreen(data[i]);
            nBlue = qBlue(data[i]);

            red = nRed + (nRed - 127) * value / 100.0;
            red = (red < 0x00) ? 0x00 : (red > 0xff) ? 0xff : red;
            green = nGreen + (nGreen - 127) * value / 100.0;
            green = (green < 0x00) ? 0x00 : (green > 0xff) ? 0xff : green;
            blue = nBlue + (nBlue - 127) * value / 100.0;
            blue = (blue < 0x00) ? 0x00 : (blue > 0xff) ? 0xff : blue;

            data[i] = qRgba(red, green, blue, qAlpha(data[i]));
        }
    }

    return image;
}
```

```
//对比度滑动条
void MainWindow::on_horizontalSlider_duibi_valueChanged(int value)
{
    if(origin_path!=nullptr){
    QImage image(origin_path);
    QImage images=AdjustContrast(image,value);
    QImage Image=ImageCenter(images,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    }
    else
    {
        QMessageBox::warning(nullptr,"提示", "请先选择一张图片！ ",QMessageBox::Yes |
QMessageBox::Yes);
    }
}

//饱和度函数调用
QImage MainWindow::AdjustSaturation(QImage Img, int iSaturateValue)
{
    int red, green, blue, nRed, nGreen, nBlue;
    int pixels = Img.width() * Img.height();
    unsigned int *data = (unsigned int *)Img.bits();

    float Increment = iSaturateValue/100.0;

    float delta = 0;
    float minVal, maxVal;
    float L, S;
    float alpha;

    for (int i = 0; i < pixels; ++i)
    {
        nRed = qRed(data[i]);
        nGreen = qGreen(data[i]);
        nBlue = qBlue(data[i]);

        minVal = std::min(std::min(nRed, nGreen), nBlue);
        maxVal = std::max(std::max(nRed, nGreen), nBlue);
        delta = (maxVal - minVal) / 255.0;
        L = 0.5*(maxVal + minVal) / 255.0;
        S = std::max(0.5*delta / L, 0.5*delta / (1 - L));

        if (Increment > 0)
        {
            alpha = std::max(S, 1 - Increment);
            alpha = 1.0 / alpha - 1;
            red = nRed + (nRed - L*255.0)*alpha;
            red = (red < 0x00) ? 0x00 : (red > 0xff) ? 0xff : red;
            green = nGreen + (nGreen - L*255.0)*alpha;
            green = (green < 0x00) ? 0x00 : (green > 0xff) ? 0xff : green;
```

```
                blue = nBlue + (nBlue - L*255.0)*alpha;
                blue = (blue < 0x00) ? 0x00 : (blue > 0xff) ? 0xff : blue;
            }
            else
            {
                alpha = Increment;
                red = L*255.0 + (nRed - L * 255.0)*(1+alpha);
                red = (red < 0x00) ? 0x00 : (red > 0xff) ? 0xff : red;
                green = L*255.0 + (nGreen - L * 255.0)*(1+alpha);
                green = (green < 0x00) ? 0x00 : (green > 0xff) ? 0xff : green;
                blue = L*255.0 + (nBlue - L * 255.0)*(1+alpha);
                blue = (blue < 0x00) ? 0x00 : (blue > 0xff) ? 0xff : blue;
            }

            data[i] = qRgba(red, green, blue, qAlpha(data[i]));
        }

        return Img;
}


//饱和度
void MainWindow::on_horizontalSlider_baohe_valueChanged(int value)
{
        if(origin_path!=nullptr){
        QImage image(origin_path);
        QImage images=AdjustSaturation(image,value);
        QImage Image=ImageCenter(images,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        }
        else{
            QMessageBox::warning(nullptr,"提示", "请先选择一张图片！ ",QMessageBox::Yes |
QMessageBox::Yes);
        }
}

//工具栏灰度化
void MainWindow::on_action_H_triggered()
{
        QImage image= global_img.copy();
        back_img = image.copy();
        QImage images=gray(image);
        QImage Image=ImageCenter(images,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        global_img = images.copy();
}

//工具栏均值滤波
void MainWindow::on_action_J_triggered()
```

```
{
    QImage image= global_img.copy();
    back_img = image.copy();
    image=junzhi(image);
    QImage Image=ImageCenter(image,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    global_img = image.copy();
}


//工具栏边缘检测
void MainWindow::on_action_B_triggered()
{
    QImage image= global_img.copy();
    back_img = image.copy();
    QImage newImage =bianyuan(image);
    QImage Image=ImageCenter(newImage,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    global_img = newImage.copy();

}

//工具栏伽马变换
void MainWindow::on_action_G_triggered()
{
    QImage image= global_img.copy();
    back_img = image.copy();
    image=gamma(image);
    QImage Image=ImageCenter(image,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    global_img = image.copy();
}

void MainWindow::on_action_About_triggered()
{
        customMsgBox.show();
        customMsgBox.exec();
}

//左转
void MainWindow::on_pushButton_turnleft_clicked()
{
    if(ui->label_show->pixmap()!=nullptr){
        QImage images(ui->label_show->pixmap()->toImage());
        QMatrix matrix;
        matrix.rotate(-90.0);//逆时针旋转 90 度
        images= images.transformed(matrix,Qt::FastTransformation);
        //QImage Image=ImageCenter(images,ui->label_show);
```

```
            ui->label_show->setPixmap(QPixmap::fromImage(images));
            ui->label_show->setAlignment(Qt::AlignCenter);
        }
        else{
            QMessageBox::warning(nullptr,"提示", "请先选择一张图片！",QMessageBox::Yes |
QMessageBox::Yes);
        }
}
//右转
void MainWindow::on_pushButton_turnright_clicked()
{
        if(ui->label_show->pixmap()!=nullptr){
            QImage images(ui->label_show->pixmap()->toImage());
            QMatrix matrix;
            matrix.rotate(90.0);//逆时针旋转 90 度
            images= images.transformed(matrix,Qt::FastTransformation);
            //QImage Image=ImageCenter(images,ui->label_show);
            ui->label_show->setPixmap(QPixmap::fromImage(images));
            ui->label_show->setAlignment(Qt::AlignCenter);
        }
        else{
            QMessageBox::warning(nullptr,"提示", "请先选择一张图片！",QMessageBox::Yes |
QMessageBox::Yes);
        }
}
//垂直镜像
void MainWindow::on_pushButton_turnleft_2_clicked()
{
        if(ui->label_show->pixmap()!=nullptr){
            QImage images(ui->label_show->pixmap()->toImage());
            images = images.mirrored(true, false);
            //QImage Image=ImageCenter(images,ui->label_show);
            ui->label_show->setPixmap(QPixmap::fromImage(images));
            ui->label_show->setAlignment(Qt::AlignCenter);
        }
        else{
            QMessageBox::warning(nullptr,"提示", "请先选择一张图片！",QMessageBox::Yes |
QMessageBox::Yes);
        }
}
//水平镜像
void MainWindow::on_pushButton_turnleft_3_clicked()
{
        if(ui->label_show->pixmap()!=nullptr){
            QImage images(ui->label_show->pixmap()->toImage());
            images = images.mirrored(false, true);
            //QImage Image=ImageCenter(images,ui->label_show);
            ui->label_show->setPixmap(QPixmap::fromImage(images));
            ui->label_show->setAlignment(Qt::AlignCenter);
        }
        else{
```

```
        QMessageBox::warning(nullptr,"提示", "请先选择一张图片！ ",QMessageBox::Yes |
QMessageBox::Yes);
    }
}

//-----------------------高斯滤波-----------------------//
QImage MainWindow::gauss(QImage image,double photometricStandardDeviation, double spatialDecay)
{
    QImage imgCopy = QImage(image);

    double c = -0.5 / (photometricStandardDeviation * photometricStandardDeviation); //-1/2 *光度标准偏差的
平方
    double mu = spatialDecay / (2 - spatialDecay);

    double *exptable = new double[256];
    double *g_table = new double[256];
    for (int i = 0; i <= 255; i++) {
        exptable[i] = (1 - spatialDecay) * exp(c * i * i);
        g_table[i] = mu * i;
    }
    static int width = imgCopy.width();
    static int height = imgCopy.height();
    int length = width * height;
    double *data2Red = new double[length];
    double *data2Green = new double[length];
    double *data2Blue = new double[length];

    int i = 0;

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            QRgb rgb = imgCopy.pixel(x, y);
            data2Red[i] = qRed(rgb);
            data2Green[i] = qGreen(rgb);
            data2Blue[i] = qBlue(rgb);
            i++;
        }
    }
    double *gRed = new double[length];
    double *pRed = new double[length];
    double *rRed = new double[length];

    double *gGreen = new double[length];
    double *pGreen = new double[length];
    double *rGreen = new double[length];

    double *gBlue = new double[length];
    double *pBlue = new double[length];
    double *rBlue = new double[length];

    memcpy(pRed, data2Red, sizeof(double) * length);
```

```
memcpy(rRed, data2Red, sizeof(double) * length);
memcpy(pGreen, data2Green, sizeof(double) * length);
memcpy(rGreen, data2Green, sizeof(double) * length);
memcpy(pBlue, data2Blue, sizeof(double) * length);
memcpy(rBlue, data2Blue, sizeof(double) * length);


double rho0 = 1.0 / (2 - spatialDecay);
for (int k2 = 0; k2 < height; ++k2)
{
    int startIndex = k2 * width;
    double mu = 0.0;
    for (int k = startIndex + 1, K = startIndex + width; k < K; ++k)
    {
        int div0Red = fabs(pRed[k] - pRed[k - 1]);
        mu = exptable[div0Red];
        pRed[k] = pRed[k - 1] * mu + pRed[k] * (1.0 - mu);//公式 1

        int div0Green = fabs(pGreen[k] - pGreen[k - 1]);
        mu = exptable[div0Green];
        pGreen[k] = pGreen[k - 1] * mu + pGreen[k] * (1.0 - mu);//公式 1

        int div0Blue = fabs(pBlue[k] - pBlue[k - 1]);
        mu = exptable[div0Blue];
        pBlue[k] = pBlue[k - 1] * mu + pBlue[k] * (1.0 - mu);//公式 1

    }

    for (int k = startIndex + width - 2; startIndex <= k; --k)
    {
        int div0Red = fabs(rRed[k] - rRed[k + 1]);
        double mu = exptable[div0Red];
        rRed[k] = rRed[k + 1] * mu + rRed[k] * (1.0 - mu);//公式 3

        int div0Green = fabs(rGreen[k] - rGreen[k + 1]);
        mu = exptable[div0Green];
        rGreen[k] = rGreen[k + 1] * mu + rGreen[k] * (1.0 - mu);//公式 3

        int div0Blue = fabs(rBlue[k] - rBlue[k + 1]);
        mu = exptable[div0Blue];
        rBlue[k] = rBlue[k + 1] * mu + rBlue[k] * (1.0 - mu);//公式 3
    }
    for (int k = startIndex, K = startIndex + width; k < K; k++)
    {
        rRed[k] = (rRed[k] + pRed[k]) * rho0 - g_table[(int)data2Red[k]];
        rGreen[k] = (rGreen[k] + pGreen[k]) * rho0 - g_table[(int)data2Green[k]];
        rBlue[k] = (rBlue[k] + pBlue[k]) * rho0 - g_table[(int)data2Blue[k]];
    }
}

int m = 0;
```

```
for (int k2 = 0; k2 < height; k2++) {
    int n = k2;
    for (int k1 = 0; k1 < width; k1++) {
        gRed[n] = rRed[m];
        gGreen[n] = rGreen[m];
        gBlue[n] = rBlue[m];
        m++;
        n += height;
    }
}

memcpy(pRed, gRed, sizeof(double) * height * width);
memcpy(rRed, gRed, sizeof(double) * height * width);
memcpy(pGreen, gGreen, sizeof(double) * height * width);
memcpy(rGreen, gGreen, sizeof(double) * height * width);
memcpy(pBlue, gBlue, sizeof(double) * height * width);
memcpy(rBlue, gBlue, sizeof(double) * height * width);

for (int k1 = 0; k1 < width; ++k1)
{
    int startIndex = k1 * height;
    double mu = 0.0;
    for (int k = startIndex + 1, K = startIndex + height; k < K; ++k)
    {
        int div0Red = fabs(pRed[k] - pRed[k - 1]);
        mu = exptable[div0Red];
        pRed[k] = pRed[k - 1] * mu + pRed[k] * (1.0 - mu);

        int div0Green = fabs(pGreen[k] - pGreen[k - 1]);
        mu = exptable[div0Green];
        pGreen[k] = pGreen[k - 1] * mu + pGreen[k] * (1.0 - mu);

        int div0Blue = fabs(pBlue[k] - pBlue[k - 1]);
        mu = exptable[div0Blue];
        pBlue[k] = pBlue[k - 1] * mu + pBlue[k] * (1.0 - mu);
    }
    for (int k = startIndex + height - 2; startIndex <= k; --k)
    {
        int div0Red = fabs(rRed[k] - rRed[k + 1]);
        mu = exptable[div0Red];
        rRed[k] = rRed[k + 1] * mu + rRed[k] * (1.0 - mu);

        int div0Green = fabs(rGreen[k] - rGreen[k + 1]);
        mu = exptable[div0Green];
        rGreen[k] = rGreen[k + 1] * mu + rGreen[k] * (1.0 - mu);

        int div0Blue = fabs(rBlue[k] - rBlue[k + 1]);
        mu = exptable[div0Blue];
        rBlue[k] = rBlue[k + 1] * mu + rBlue[k] * (1.0 - mu);
    }
}
```

```
    double init_gain_mu = spatialDecay / (2 - spatialDecay);
    for (int k = 0; k < length; ++k) {
        rRed[k] = (rRed[k] + pRed[k]) * rho0 - gRed[k] * init_gain_mu;
        rGreen[k] = (rGreen[k] + pGreen[k]) * rho0 - gGreen[k] * init_gain_mu;
        rBlue[k] = (rBlue[k] + pBlue[k]) * rho0 - gBlue[k] * init_gain_mu;
    }
    m = 0;
    for (int k1 = 0; k1 < width; ++k1)
    {
        int n = k1;
        for (int k2 = 0; k2 < height; ++k2)
        {

            data2Red[n] = rRed[m];
            data2Green[n] = rGreen[m];
            data2Blue[n] = rBlue[m];
            imgCopy.setPixel(k1, k2, qRgb(data2Red[n], data2Green[n], data2Blue[n]));
            m++;
            n += width;
        }
    }
    delete []data2Red;      data2Red = nullptr;
    delete []data2Green; data2Green = nullptr;
    delete []data2Blue;    data2Blue = nullptr;

    delete []pRed;             pRed = nullptr;
    delete []rRed;             rRed = nullptr;
    delete []gRed;             gRed = nullptr;

    delete []pGreen;          pGreen = nullptr;
    delete []rGreen;          rGreen = nullptr;
    delete []gGreen;          gGreen = nullptr;

    delete []pBlue;           pBlue = nullptr;
    delete []rBlue;           rBlue = nullptr;
    delete []gBlue;           gBlue = nullptr;

    delete []exptable;      exptable = nullptr;
    delete []g_table;        g_table = nullptr;

    return imgCopy;
}

void MainWindow::on_btn_guass_clicked()
{
        QImage image = global_img.copy();
        back_img = image.copy();
        image=gauss(image,20,0.01);
        QImage Image=ImageCenter(image,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
```

```
        ui->label_show->setAlignment(Qt::AlignCenter);
        statusBar()->showMessage(tr("高斯滤波成功！"));
        global_img = image.copy();
}
//-------------------------------------------------//

//-------------Mat 与 QImage 的转化----------------------//
QImage MainWindow::cvMat2QImage(const cv::Mat& mat)
{
    // 8-bits unsigned, NO. OF CHANNELS = 1
    if (mat.type() == CV_8UC1)
    {
        QImage image(mat.cols, mat.rows, QImage::Format_Indexed8);
        // Set the color table (used to translate colour indexes to qRgb values)
        image.setColorCount(256);
        for (int i = 0; i < 256; i++)
        {
            image.setColor(i, qRgb(i, i, i));
        }
        // Copy input Mat
        uchar *pSrc = mat.data;
        for (int row = 0; row < mat.rows; row++)
        {
            uchar *pDest = image.scanLine(row);
            memcpy(pDest, pSrc, mat.cols);
            pSrc += mat.step;
        }
        return image;
    }
    // 8-bits unsigned, NO. OF CHANNELS = 3
    else if (mat.type() == CV_8UC3)
    {
        // Copy input Mat
        const uchar *pSrc = (const uchar*)mat.data;
        // Create QImage with same dimensions as input Mat
        QImage image(pSrc, mat.cols, mat.rows, mat.step, QImage::Format_RGB888);
        return image.rgbSwapped();
    }
    else if (mat.type() == CV_8UC4)
    {
        qDebug() << "CV_8UC4";
        // Copy input Mat
        const uchar *pSrc = (const uchar*)mat.data;
        // Create QImage with same dimensions as input Mat
        QImage image(pSrc, mat.cols, mat.rows, mat.step, QImage::Format_ARGB32);
        return image.copy();
    }
    else
    {
        qDebug() << "ERROR: Mat could not be converted to QImage.";
        return QImage();
```

```
    }
}

Mat MainWindow::QImage2cvMat(QImage image)
{
    cv::Mat mat;
    qDebug() << image.format();
    switch (image.format())
    {
    case QImage::Format_ARGB32:
    case QImage::Format_RGB32:
    case QImage::Format_ARGB32_Premultiplied:
        mat = cv::Mat(image.height(), image.width(), CV_8UC4, (void*)image.constBits(),
image.bytesPerLine());
        break;
    case QImage::Format_RGB888:
        mat = cv::Mat(image.height(), image.width(), CV_8UC3, (void*)image.constBits(),
image.bytesPerLine());
        cv::cvtColor(mat, mat, CV_BGR2RGB);
        break;
    case QImage::Format_Indexed8:
        mat = cv::Mat(image.height(), image.width(), CV_8UC1, (void*)image.constBits(),
image.bytesPerLine());
        break;
    default: break;
    }
    return mat;
}

//--------------------鼠标事件------------------------//
bool showROI = false;
Point prev_pt = Point(-1, -1);
Mat src, background_img, foreground_img;
// 鼠标事件处理函数
static void on_mouse(int event, int x, int y, int flags, void* userdata)
{
    // 松开鼠标左键或不是按住左键拖拽的动作时，把坐标还原
    if (event == EVENT_LBUTTONUP || !(flags & EVENT_FLAG_LBUTTON))
    {
        prev_pt = Point(-1, -1);
    }
    // 按下左键
    else if (event == EVENT_LBUTTONDOWN)
    {
        prev_pt = Point(x, y);
    }
    // 移动鼠标并按住左键拖拽
    else if (event == EVENT_MOUSEMOVE && (flags & EVENT_FLAG_LBUTTON))
    {
        Point pt = Point(x, y);
        // 前景模板上划线
```

```
            line(foreground_img, prev_pt, pt, Scalar(255), 2, 8, 0);
            // 原图上划线
            line(src, prev_pt, pt, Scalar::all(255), 2, 8, 0);
            // 起点等于终点，说明曲线要闭合
            prev_pt = pt;
            imshow("file", src);
        }
        // 点击右键，截取所选区域
        if (event == EVENT_RBUTTONUP)
        {
            Mat dst;
            // 画线闭合区域被白色填充显示在原始图像上
            floodFill(foreground_img, Point(x, y), Scalar(255));
            // img 中被 FG_mask 掩盖后的图像附到 FG 中显示
            src.copyTo(dst, foreground_img);
            namedWindow("ROI", 0);
            imshow("ROI", dst);
            imwrite("ROI.png", dst);  // 保存 ROI 图像
            showROI = true;  // 设置标志以显示 ROI
            waitKey(0);
        }
}

void MainWindow::on_ROI_clicked()
{
    QImage images(origin_path);
    Mat image = QImage2cvMat(images);
    if (image.empty())
    {
        std::cout << "读取文件失败！" << std::endl;
        QMessageBox::warning(nullptr, "提示", "请先打开图片！", QMessageBox::Yes | QMessageBox::Yes);
    }
    // 初始化前景和背景模板
    foreground_img = Mat(image.size(), CV_8UC1, Scalar(0));
    image.copyTo(src);
    namedWindow("file", 0);
    imshow("file", src);

    // 检查窗口是否成功创建
    if (getWindowProperty("file", WND_PROP_AUTOSIZE) != -1) {
        setMouseCallback("file", on_mouse, 0);
        waitKey(0);

        // 如果标志已设置为 true，显示 ROI
        if (showROI)
        {
            Mat dst;
            // img 中被 FG_mask 掩盖后的图像附到 FG 中显示
            src.copyTo(dst, foreground_img);
            namedWindow("ROI", 0);
```

```
                imshow("ROI", dst);
                showROI = false;   // 重置标志
                waitKey(0);
            }
        } else {
            cout << "无法成功创建或显示窗口。" << endl;
        }
}

//----------------------------------------//

void MainWindow::on_BoxFilter_clicked()
{
    Mat dst;
    QImage image = global_img.copy();
    back_img = image.copy();
    Mat src = QImage2cvMat(image);
    dst.create(src.size(), src.type());
    boxFilter(src, dst, -1, Size(5, 5));
    QImage img = cvMat2QImage(dst);
    QImage Image=ImageCenter(img,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("方框滤波成功！"));
    global_img = img.copy();
}

//求九个数的中值
uchar Median(uchar n1, uchar n2, uchar n3, uchar n4, uchar n5,
    uchar n6, uchar n7, uchar n8, uchar n9) {
    uchar arr[9];
    arr[0] = n1;
    arr[1] = n2;
    arr[2] = n3;
    arr[3] = n4;
    arr[4] = n5;
    arr[5] = n6;
    arr[6] = n7;
    arr[7] = n8;
    arr[8] = n9;
    for (int gap = 9 / 2; gap > 0; gap /= 2)//希尔排序
        for (int i = gap; i < 9; ++i)
            for (int j = i - gap; j >= 0 && arr[j] > arr[j + gap]; j -= gap)
                swap(arr[j], arr[j + gap]);
    return arr[4];//返回中值
}

//图像椒盐化
void salt(Mat &image, int num) {
    if (!image.data) return;//防止传入空图
```

```
    int i, j;
    srand(time(NULL));
    for (int x = 0; x < num; ++x) {
        i = rand() % image.rows;
        j = rand() % image.cols;
        image.at<Vec3b>(i, j)[0] = 255;
        image.at<Vec3b>(i, j)[1] = 255;
        image.at<Vec3b>(i, j)[2] = 255;
    }
}

//中值滤波函数
void MedianFlitering(const Mat &src, Mat &dst) {
    if (!src.data)return;
    Mat _dst(src.size(), src.type());
    for(int i=0;i<src.rows;++i)
        for (int j=0; j < src.cols; ++j) {
            if ((i - 1) > 0 && (i + 1) < src.rows && (j - 1) > 0 && (j + 1) < src.cols) {
                _dst.at<Vec3b>(i, j)[0] = Median(src.at<Vec3b>(i, j)[0], src.at<Vec3b>(i + 1, j + 1)[0],
                    src.at<Vec3b>(i + 1, j)[0], src.at<Vec3b>(i, j + 1)[0], src.at<Vec3b>(i + 1, j - 1)[0],
                    src.at<Vec3b>(i - 1, j + 1)[0], src.at<Vec3b>(i - 1, j)[0], src.at<Vec3b>(i, j - 1)[0],
                    src.at<Vec3b>(i - 1, j - 1)[0]);
                _dst.at<Vec3b>(i, j)[1] = Median(src.at<Vec3b>(i, j)[1], src.at<Vec3b>(i + 1, j + 1)[1],
                    src.at<Vec3b>(i + 1, j)[1], src.at<Vec3b>(i, j + 1)[1], src.at<Vec3b>(i + 1, j - 1)[1],
                    src.at<Vec3b>(i - 1, j + 1)[1], src.at<Vec3b>(i - 1, j)[1], src.at<Vec3b>(i, j - 1)[1],
                    src.at<Vec3b>(i - 1, j - 1)[1]);
                _dst.at<Vec3b>(i, j)[2] = Median(src.at<Vec3b>(i, j)[2], src.at<Vec3b>(i + 1, j + 1)[2],
                    src.at<Vec3b>(i + 1, j)[2], src.at<Vec3b>(i, j + 1)[2], src.at<Vec3b>(i + 1, j - 1)[2],
                    src.at<Vec3b>(i - 1, j + 1)[2], src.at<Vec3b>(i - 1, j)[2], src.at<Vec3b>(i, j - 1)[2],
                    src.at<Vec3b>(i - 1, j - 1)[2]);
            }
            else
                _dst.at<Vec3b>(i, j) = src.at<Vec3b>(i, j);
        }
    _dst.copyTo(dst);//拷贝
}
//中值滤波
void MainWindow::on_btn_GAUS_clicked()
{
    Mat Salt_Image,result;
    QImage images=global_img.copy();
    back_img = images.copy();
    Mat image = QImage2cvMat(images);
    image.copyTo(Salt_Image);
    salt(Salt_Image, 3000);

    //MedianFlitering(Salt_Image, result);
    medianBlur(Salt_Image, result, 3);
    QImage img = cvMat2QImage(result);
    QImage Image=ImageCenter(img,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
```

```
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("中值滤波完成"));
    global_img = img.copy();
}

//--------------------------大津算法--------------------------//
//将灰度图像转为三通道的 BGR 图像
cv::Mat gray2BGR(cv::Mat grayImg) {
    if (grayImg.channels() == 3)
        return grayImg;
    cv::Mat bgrImg = cv::Mat::zeros(grayImg.size(), CV_8UC3);
    std::vector<cv::Mat> bgr_channels;
    cv::split(bgrImg, bgr_channels);
    bgr_channels.at(0) = grayImg;
    bgr_channels.at(1) = grayImg;
    bgr_channels.at(2) = grayImg;
    cv::merge(bgr_channels, bgrImg);
    return bgrImg;
}

cv::Mat drawImage(cv::Mat image, vector< vector< Point> > pointV) {
    cv::Mat destImage=image.clone();
    if (destImage.channels()==1)
    {
        destImage = gray2BGR(destImage);

    }
    for (size_t i=0;i<pointV.size();i++)
    {
        for (size_t j = 0; j<pointV.at(i).size(); j++)
        {
            cv::Point point = pointV.at(i).at(j);
            destImage.at<Vec3b>(point) = cv::Vec3b(0, 0, saturate_cast<uchar>(255-i*5));
        }
    }
    return destImage;
}

int OtsuAlgThreshold(const Mat image)
{
    if(image.channels()!=1)
    {
        cout<<"Please input Gray-image!"<<endl;
        return 0;
    }
    int T=0; //Otsu 算法阈值
    double varValue=0; //类间方差中间值保存
    double w0=0; //前景像素点数所占比例
    double w1=0; //背景像素点数所占比例
    double u0=0; //前景平均灰度
```

```
double u1=0; //背景平均灰度
double Histogram[256]={0}; //灰度直方图，下标是灰度值，保存内容是灰度值对应的像素点总数
uchar *data=image.data;
double totalNum=image.rows*image.cols; //像素总数
//计算灰度直方图分布，Histogram 数组下标是灰度值，保存内容是灰度值对应像素点数
for(int i=0;i<image.rows;i++)    //为表述清晰，并没有把 rows 和 cols 单独提出来
{
    for(int j=0;j<image.cols;j++)
    {
        Histogram[data[i*image.step+j]]++;
    }
}
for(int i=0;i<255;i++)
{
    //每次遍历之前初始化各变量
    w1=0;        u1=0;        w0=0;        u0=0;
    //***********背景各分量值计算**************************
    for(int j=0;j<=i;j++) //背景部分各值计算
    {
        w1+=Histogram[j];   //背景部分像素点总数
        u1+=j*Histogram[j]; //背景部分像素总灰度和
    }
    if(w1==0) //背景部分像素点数为 0 时退出
    {
        break;
    }
    u1=u1/w1; //背景像素平均灰度
    w1=w1/totalNum; //  背景部分像素点数所占比例
    //***********背景各分量值计算**************************

    //***********前景各分量值计算**************************
    for(int k=i+1;k<255;k++)
    {
        w0+=Histogram[k];   //前景部分像素点总数
        u0+=k*Histogram[k]; //前景部分像素总灰度和
    }
    if(w0==0) //前景部分像素点数为 0 时退出
    {
        break;
    }
    u0=u0/w0; //前景像素平均灰度
    w0=w0/totalNum; //  前景部分像素点数所占比例
    //***********前景各分量值计算**************************

    //***********类间方差计算******************************
    double varValueI=w0*w1*(u1-u0)*(u1-u0); //当前类间方差计算
    if(varValue<varValueI)
    {
        varValue=varValueI;
```

```
                T=i;
            }
        }
        return T;
}

void MainWindow::on_btn_otsu_clicked()
{
        Mat dst;
        QImage image = global_img.copy();
        back_img = image.copy();
        QImage images = gray(image);
        Mat src = QImage2cvMat(images);
        cvtColor(src,src,CV_RGB2GRAY);
        Mat imageOutput;
        Mat imageOtsu;
        int thresholdValue=OtsuAlgThreshold(src);
        cout<<"类间方差为：  "<<thresholdValue<<endl;
        threshold(src,imageOutput,thresholdValue,255,CV_THRESH_BINARY);
        threshold(src,imageOtsu,0,255,CV_THRESH_OTSU); //Opencv Otsu 算法
        QImage img = cvMat2QImage(imageOtsu);
        QImage Image=ImageCenter(img,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        statusBar()->showMessage(tr("已对图像进行大津算法阈值分割！"));
        global_img = img.copy();
}

//-------------------------自适应阈值分割-------------------------//
enum adaptiveMethod{meanFilter,gaaussianFilter,medianFilter};

void AdaptiveThreshold(cv::Mat& src, cv::Mat& dst, double Maxval, int Subsize, double c, adaptiveMethod
method = meanFilter){

        if (src.channels() > 1)
                cv::cvtColor(src, src, CV_RGB2GRAY);

        cv::Mat smooth;
        switch (method)
        {
        case    meanFilter:
                cv::blur(src, smooth, cv::Size(Subsize, Subsize));    //均值滤波
                break;
        case gaaussianFilter:
                cv::GaussianBlur(src, smooth, cv::Size(Subsize, Subsize),0,0); //高斯滤波
                break;
        case medianFilter:
                cv::medianBlur(src, smooth, Subsize);     //中值滤波
                break;
        default:
                break;
```

```
    }

    smooth = smooth - c;

    //阈值处理
    src.copyTo(dst);
    for (int r = 0; r < src.rows;++r){
        const uchar* srcptr = src.ptr<uchar>(r);
        const uchar* smoothptr = smooth.ptr<uchar>(r);
        uchar* dstptr = dst.ptr<uchar>(r);
        for (int c = 0; c < src.cols; ++c){
            if (srcptr[c]>smoothptr[c]){
                dstptr[c] = Maxval;
            }
            else
                dstptr[c] = 0;
        }
    }
}

void MainWindow::on_btn_adapt_clicked()
{
        Mat dst,dst2;
        QImage image = global_img.copy();
        back_img = image.copy();
        QImage images = gray(image);
        Mat src = QImage2cvMat(images);
        dst.create(src.size(), src.type());
        cvtColor(src, src, CV_RGB2BGR);
        if (src.channels() > 1) { cvtColor(src, src, CV_BGR2GRAY); } //转换为灰度图
        double t2 = (double)cv::getTickCount();
        AdaptiveThreshold(src, dst, 255, 21, 10, meanFilter);    //
        t2 = (double)cv::getTickCount() - t2;
        double time2 = (t2 *1000.) / ((double)cv::getTickFrequency());
        std::cout << "my_process=" << time2 << " ms. " << std::endl << std::endl;
        adaptiveThreshold(src, dst2, 255, cv::ADAPTIVE_THRESH_MEAN_C, cv::THRESH_BINARY, 21,
10);
        QImage img = cvMat2QImage(dst);
        QImage Image=ImageCenter(img,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        statusBar()->showMessage(tr("自适应阈值分割！"));
        global_img = img.copy();
}
//-------------------------自适应阈值分割-------------------------//

//-------------------------线条细化-------------------------//
void cvThin(cv::Mat& src, cv::Mat& dst, int intera)
    {
    if(src.type()!=CV_8UC1)
        {
```

```
        cvtColor(src, src, CV_BGR2GRAY);
//          printf("Only binary or grayscale images can be processed\n");
//          return;
        }
    //非原地操作时候，copy src 到 dst
    if(dst.data!=src.data)
        {
        src.copyTo(dst);
        }

    int i, j, n;
    int width, height;
    width = src.cols -1;
    //之所以减 1，是方便处理 8 邻域，防止越界
    height = src.rows -1;
    int step = src.step;
    int   p2,p3,p4,p5,p6,p7,p8,p9;
    uchar* img;
    bool ifEnd;
    int A1;
    cv::Mat tmpimg;
    //n 表示迭代次数
    for(n = 0; n<intera; n++)
        {
        dst.copyTo(tmpimg);
        ifEnd = false;
        img = tmpimg.data;
        for(i = 1; i < height; i++)
            {
            img += step;
            for(j =1; j<width; j++)
                {
                uchar* p = img + j;
                A1 = 0;
                if( p[0] > 0)
                    {
                    if(p[-step]==0&&p[-step+1]>0) //p2,p3 01 模式
                        {
                        A1++;
                        }
                    if(p[-step+1]==0&&p[1]>0) //p3,p4 01 模式
                        {
                        A1++;
                        }
                    if(p[1]==0&&p[step+1]>0) //p4,p5 01 模式
                        {
                        A1++;
                        }
                    if(p[step+1]==0&&p[step]>0) //p5,p6 01 模式
                        {
                        A1++;
```

```
                    }
                if(p[step]==0&&p[step-1]>0) //p6,p7 01 模式
                    {
                    A1++;
                    }
                if(p[step-1]==0&&p[-1]>0) //p7,p8 01 模式
                    {
                    A1++;
                    }
                if(p[-1]==0&&p[-step-1]>0) //p8,p9 01 模式
                    {
                    A1++;
                    }
                if(p[-step-1]==0&&p[-step]>0) //p9,p2 01 模式
                    {
                    A1++;
                    }
                p2 = p[-step]>0?1:0;
                p3 = p[-step+1]>0?1:0;
                p4 = p[1]>0?1:0;
                p5 = p[step+1]>0?1:0;
                p6 = p[step]>0?1:0;
                p7 = p[step-1]>0?1:0;
                p8 = p[-1]>0?1:0;
                p9 = p[-step-1]>0?1:0;
                if((p2+p3+p4+p5+p6+p7+p8+p9)>1 && (p2+p3+p4+p5+p6+p7+p8+p9)<7   &&
A1==1)
                    {
                    if((p2==0||p4==0||p6==0)&&(p4==0||p6==0||p8==0)) //p2*p4*p6=0 &&
p4*p6*p8==0
                        {
                        dst.at<uchar>(i,j) = 0; //满足删除条件，设置当前像素为 0
                        ifEnd = true;
                        }
                    }
                }
            }
        }

    dst.copyTo(tmpimg);
    img = tmpimg.data;
    for(i = 1; i < height; i++)
        {
        img += step;
        for(j =1; j<width; j++)
            {
            A1 = 0;
            uchar* p = img + j;
            if( p[0] > 0)
                {
                if(p[-step]==0&&p[-step+1]>0) //p2,p3 01 模式
```

```
                {
                A1++;
                }
            if(p[-step+1]==0&&p[1]>0) //p3,p4 01 模式
                {
                A1++;
                }
            if(p[1]==0&&p[step+1]>0) //p4,p5 01 模式
                {
                A1++;
                }
            if(p[step+1]==0&&p[step]>0) //p5,p6 01 模式
                {
                A1++;
                }
            if(p[step]==0&&p[step-1]>0) //p6,p7 01 模式
                {
                A1++;
                }
            if(p[step-1]==0&&p[-1]>0) //p7,p8 01 模式
                {
                A1++;
                }
            if(p[-1]==0&&p[-step-1]>0) //p8,p9 01 模式
                {
                A1++;
                }
            if(p[-step-1]==0&&p[-step]>0) //p9,p2 01 模式
                {
                A1++;
                }
            p2 = p[-step]>0?1:0;
            p3 = p[-step+1]>0?1:0;
            p4 = p[1]>0?1:0;
            p5 = p[step+1]>0?1:0;
            p6 = p[step]>0?1:0;
            p7 = p[step-1]>0?1:0;
            p8 = p[-1]>0?1:0;
            p9 = p[-step-1]>0?1:0;
            if((p2+p3+p4+p5+p6+p7+p8+p9)>1 && (p2+p3+p4+p5+p6+p7+p8+p9)<7    &&
A1==1)
                {
                if((p2==0||p4==0||p8==0)&&(p2==0||p6==0||p8==0)) //p2*p4*p8=0 &&
p2*p6*p8==0
                    {
                    dst.at<uchar>(i,j) = 0; //满足删除条件，设置当前像素为 0
                    ifEnd = true;
                    }
                }
            }
```

```
                    }
                }

            //如果两个子迭代已经没有可以细化的像素了，则退出迭代
            if(!ifEnd) break;
            }
        }

void MainWindow::on_btn_canny_clicked()
{
    Mat dst,g_cannyDetectedEdges;
    QImage image = global_img.copy();
    back_img = image.copy();
    QImage images = gray(image);
    Mat src = QImage2cvMat(images);
    cvThin(src,src,15);
    QImage img = cvMat2QImage(src);
    QImage Image=ImageCenter(img,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("已对图像进行线条细化"));
    global_img = img.copy();
}

//------------------------裂纹标记------------------------//
/*利用查找表(Look-up table)增加图像对比度*/
void addContrast(Mat & srcImg) {
    Mat lookUpTable(1, 256, CV_8U);
    double temp = pow(1.1, 5);
    uchar* p = lookUpTable.data;
    for (int i = 0; i < 256; ++i)
        p[i] = saturate_cast<uchar>(i * temp);
    LUT(srcImg, lookUpTable, srcImg);
}

/*  二值化图像。0->0,非 0->255 */
void binaryzation(Mat & srcImg) {
    Mat lookUpTable(1, 256, CV_8U, Scalar(255));
    lookUpTable.data[0] = 0;
    LUT(srcImg, lookUpTable, srcImg);
}

/*  提取连通域的骨架  */
void thinImage(Mat & srcImg) {
    vector<Point> deleteList;
    int neighbourhood[9];
    int nl = srcImg.rows;
    int nc = srcImg.cols;
    bool inOddIterations = true;
    while (true) {
        for (int j = 1; j < (nl - 1); j++) {
```

```
            uchar* data_last = srcImg.ptr<uchar>(j - 1);
            uchar* data = srcImg.ptr<uchar>(j);
            uchar* data_next = srcImg.ptr<uchar>(j + 1);
            for (int i = 1; i < (nc - 1); i++) {
                if (data[i] == 255) {
                    int whitePointCount = 0;
                    neighbourhood[0] = 1;
                    if (data_last[i] == 255) neighbourhood[1] = 1;
                    else    neighbourhood[1] = 0;
                    if (data_last[i + 1] == 255) neighbourhood[2] = 1;
                    else    neighbourhood[2] = 0;
                    if (data[i + 1] == 255) neighbourhood[3] = 1;
                    else    neighbourhood[3] = 0;
                    if (data_next[i + 1] == 255) neighbourhood[4] = 1;
                    else    neighbourhood[4] = 0;
                    if (data_next[i] == 255) neighbourhood[5] = 1;
                    else    neighbourhood[5] = 0;
                    if (data_next[i - 1] == 255) neighbourhood[6] = 1;
                    else    neighbourhood[6] = 0;
                    if (data[i - 1] == 255) neighbourhood[7] = 1;
                    else    neighbourhood[7] = 0;
                    if (data_last[i - 1] == 255) neighbourhood[8] = 1;
                    else    neighbourhood[8] = 0;
                    for (int k = 1; k < 9; k++) {
                        whitePointCount = whitePointCount + neighbourhood[k];
                    }
                    if ((whitePointCount >= 2) && (whitePointCount <= 6)) {
                        int ap = 0;
                        if ((neighbourhood[1] == 0) && (neighbourhood[2] == 1)) ap++;
                        if ((neighbourhood[2] == 0) && (neighbourhood[3] == 1)) ap++;
                        if ((neighbourhood[3] == 0) && (neighbourhood[4] == 1)) ap++;
                        if ((neighbourhood[4] == 0) && (neighbourhood[5] == 1)) ap++;
                        if ((neighbourhood[5] == 0) && (neighbourhood[6] == 1)) ap++;
                        if ((neighbourhood[6] == 0) && (neighbourhood[7] == 1)) ap++;
                        if ((neighbourhood[7] == 0) && (neighbourhood[8] == 1)) ap++;
                        if ((neighbourhood[8] == 0) && (neighbourhood[1] == 1)) ap++;
                        if (ap == 1) {
                            if (inOddIterations && (neighbourhood[3] * neighbourhood[5] *
neighbourhood[7] == 0)
                                && (neighbourhood[1] * neighbourhood[3] * neighbourhood[5] == 0)) {
                                deleteList.push_back(Point(i, j));
                            }
                            else if (!inOddIterations && (neighbourhood[1] * neighbourhood[5] *
neighbourhood[7] == 0)
                                && (neighbourhood[1] * neighbourhood[3] * neighbourhood[7] == 0)) {
                                deleteList.push_back(Point(i, j));
                            }
                        }
                    }
                }
            }
```

```
            }
            if (deleteList.size() == 0)
                break;
            for (size_t i = 0; i < deleteList.size(); i++) {
                Point tem;
                tem = deleteList[i];
                uchar* data = srcImg.ptr<uchar>(tem.y);
                data[tem.x] = 0;
            }
            deleteList.clear();

            inOddIterations = !inOddIterations;
        }
    }

    /* 计算宽高信息的放置位置 */
    Point calInfoPosition(int imgRows, int imgCols, int padding, const std::vector<cv::Point>& domain) {
        long xSum = 0;
        long ySum = 0;
        for (auto it = domain.cbegin(); it != domain.cend(); ++it) {
            xSum += it->x;
            ySum += it->y;
        }
        int x = 0;
        int y = 0;
        x = (int)(xSum / domain.size());
        y = (int)(ySum / domain.size());
        if (x < padding)
            x = padding;
        if (x > imgCols - padding)
            x = imgCols - padding;
        if (y < padding)
            y = padding;
        if (y > imgRows - padding)
            y = imgRows - padding;

        return cv::Point(x, y);
    }

    Scalar random_color(RNG& _rng) {
        int icolor = (unsigned)_rng;
        return Scalar(icolor & 0xFF, (icolor >> 8) & 0xFF, (icolor >> 16) & 0xFF);
    }

    void MainWindow::on_btn_biaoji_clicked()
    {
        Mat srcImg, dstImg, tempImg, temp;
        QImage image = global_img.copy();
        back_img = image.copy();
        image = gauss(image,20,0.02);
        srcImg = QImage2cvMat(image);
```

```
    cvtColor(srcImg, dstImg, CV_BGR2GRAY);
    //增加对比度
    addContrast(dstImg);
    //边缘检测
    Canny(dstImg, dstImg, 50, 150);

    Mat kernel = getStructuringElement(MorphShapes::MORPH_ELLIPSE, Size(3, 3));
    dilate(dstImg, dstImg, kernel);
    morphologyEx(dstImg, dstImg, CV_MOP_CLOSE, kernel, Point(-1, -1), 3);
    morphologyEx(dstImg, dstImg, CV_MOP_CLOSE, kernel);

    QImage img = cvMat2QImage(dstImg);
    QImage Image=ImageCenter(img,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("裂纹标记完成"));
    global_img = img.copy();
}


//------------------------图片拼接------------------------//
//打开图片
void MainWindow::on_pushButton_2_clicked()
{
    srcDirPath = QFileDialog::getExistingDirectory(this, tr("Choose folder"), "/", QFileDialog::ShowDirsOnly |
QFileDialog::DontResolveSymlinks);
    qDebug() << srcDirPath;
    //FileInfo
    QFileInfo OpenFileInfo;
    OpenFileInfo = QFileInfo(srcDirPath);
    OpenFilePath = OpenFileInfo.filePath();
    ui->lineEdit->setText(OpenFilePath);
}


void MainWindow::on_btn_imgstit_clicked()
{
    // read images from folder
    vector<Mat> images;
    QDir dir(srcDirPath);
    QStringList filters;
    filters << "*.jpg" << "*.png" << "*.bmp";
    QFileInfoList fileList = dir.entryInfoList(filters, QDir::Files|QDir::NoDotAndDotDot);
    foreach(QFileInfo fileInfo, fileList) {
            Mat img = imread(fileInfo.absoluteFilePath().toStdString());
            images.push_back(img);
       }
    //  使用 stitch 函数进行拼接
    Ptr<Stitcher> stitcher = Stitcher::create();
    if (stitcher.empty()) {
        cout << "Failed to create stitcher object!" << endl;
```

```
    } else {
        cout << "Stitcher object created successfully!" << endl;
    }

    Mat result;
    Stitcher::Status status = stitcher->stitch(images,result);

    if (status != Stitcher::OK)
    {
        QMessageBox::warning(nullptr,"提示", "Can't stitch images！",QMessageBox::Yes |
QMessageBox::Yes);
        cout << "Can't stitch images, error code = " << int(status) << endl;
    }
    waitKey();

    QString resultPath = srcDirPath + "/result.jpg";
    imwrite(resultPath.toStdString(), result);
    QImage img = cvMat2QImage(result);
    QImage Image=ImageCenter(img,ui->label_stitcher);
    ui->label_stitcher->setPixmap(QPixmap::fromImage(Image));
    ui->label_stitcher->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("图像全景拼接完成！"));

    for(unsigned int i = 0; i < images.size(); ++i)
    {
        images[i].release();
    }
    cout << "images[i] release successfully! " << endl;

    // clear images
    images.clear();
    cout << "images clear successfully! " << endl;

    // release stitcher
    stitcher.release();
    cout << "stitcher release successfully! " << endl;
}



//--------------------------图像网格化裁剪--------------------------//
void MainWindow::on_btn_cut_clicked()
{
    Mat img;
    QImage image = global_img.copy();
    back_img = image.copy();
    img = QImage2cvMat(image);
    Mat image_copy = img.clone();
    int imgheight = img.rows;
    int imgwidth = img.cols;
    int M = imgheight/4;
```

```
int N = imgwidth/5;

int x1 = 0;
int y1 = 0;
cout << imgheight << endl;
cout << imgwidth << endl;
for (int y = 0; y<imgheight; y=y+M)
{
    for (int x = 0; x<imgwidth; x=x+N)
    {
        if ((imgheight - y) < M || (imgwidth - x) < N)
        {
            break;
        }
        y1 = y + M;
        x1 = x + N;
        string a = to_string(x);
        string b = to_string(y);

        if (x1 >= imgwidth && y1 >= imgheight)
        {
            x = imgwidth - 1;
            y = imgheight - 1;
            x1 = imgwidth - 1;
            y1 = imgheight - 1;

            // crop the patches of size MxN
            Mat tiles = image_copy(Range(y, imgheight), Range(x, imgwidth));
            //save each patches into file directory
            //imwrite("saved_patches/tile" + a + '_' + b + ".jpg", tiles);
            imwrite("C:/Users/D/Pictures/new/" + a + '_' + b + ".jpg", tiles);
            rectangle(img, Point(x,y), Point(x1,y1), Scalar(0,255,0), 1);
        }
        else if (y1 >= imgheight)
        {
            y = imgheight - 1;
            y1 = imgheight - 1;
            // crop the patches of size MxN
            Mat tiles = image_copy(Range(y, imgheight), Range(x, x+N));
            //save each patches into file directory
            //imwrite("saved_patches/tile" + a + '_' + b + ".jpg", tiles);
            imwrite("C:/Users/D/Pictures/new/" + a + '_' + b + ".jpg", tiles);
            rectangle(img, Point(x,y), Point(x1,y1), Scalar(0,255,0), 1);
        }
        else if (x1 >= imgwidth)
        {
            x = imgwidth - 1;
            x1 = imgwidth - 1;
            // crop the patches of size MxN
            Mat tiles = image_copy(Range(y, y+M), Range(x, imgwidth));
            //save each patches into file directory
```

```
                    //imwrite("saved_patches/tile" + a + '_' + b + ".jpg", tiles);
                    imwrite("C:/Users/D/Pictures/new/" + a + '_' + b + ".jpg", tiles);
                    rectangle(img, Point(x,y), Point(x1,y1), Scalar(0,255,0), 1);
                }
                else
                {
                    // crop the patches of size MxN
                    Mat tiles = image_copy(Range(y, y+M), Range(x, x+N));
                    //save each patches into file directory
                    imwrite("C:/Users/D/Pictures/new/" + a + '_' + b + ".jpg", tiles);
                    //imwrite("saved_patches/tile" + a + '_' + b + ".jpg", tiles);
                    rectangle(img, Point(x,y), Point(x1,y1), Scalar(0,255,0), 1);
                }
            }
        }
        QImage IMG = cvMat2QImage(img);
        QImage Image=ImageCenter(IMG,ui->label_show);
        ui->label_show->setPixmap(QPixmap::fromImage(Image));
        ui->label_show->setAlignment(Qt::AlignCenter);
        statusBar()->showMessage(tr("图像网格化裁剪完成！"));
        global_img = IMG.copy();
}


//-------------------------图像分割-------------------------//
Vec3b RandomColor(int value){
        value=value%255;  //生成 0~255 的随机数
        RNG rng;
        int aa=rng.uniform(0,value);
        int bb=rng.uniform(0,value);
        int cc=rng.uniform(0,value);
        return Vec3b(aa,bb,cc);
}

void MainWindow::on_btn_watershed_clicked(){
        QImage img = global_img.copy();
        back_img = img.copy();
        Mat image = QImage2cvMat(img);
        cvtColor(image,image,CV_RGB2BGR);//灰度转换

        //灰度化，滤波，Canny 边缘检测
        Mat imageGray;
        cvtColor(image,imageGray,CV_BGR2GRAY);//灰度转换
        GaussianBlur(imageGray,imageGray,Size(5,5),2);    //高斯滤波
        medianBlur(imageGray, imageGray, 3);
        Canny(imageGray,imageGray,80,150);

        //查找轮廓
        vector<vector<Point>> contours;
        vector<Vec4i> hierarchy;
```

```
    findContours(imageGray,contours,hierarchy,RETR_TREE,CHAIN_APPROX_SIMPLE,Point());

    Mat imageContours=Mat::zeros(image.size(),CV_8UC1);   //轮廓
    Mat marks(image.size(),CV_32S);     //Opencv 分水岭第二个矩阵参数
    marks=Scalar::all(0);
    int index = 0;
    int compCount = 0;
    for( ; index >= 0; index = hierarchy[index][0], compCount++ )
    {
        //对 marks 进行标记，对不同区域的轮廓进行编号，相当于设置注水点，有多少轮廓，就有多少
注水点
        drawContours(marks, contours, index, Scalar::all(compCount+1), 1, 8, hierarchy);
        drawContours(imageContours,contours,index,Scalar(255),1,8,hierarchy);
    }
    //我们来看一下传入的矩阵 marks 里是什么东西
    Mat marksShows;
    convertScaleAbs(marks,marksShows);
    watershed(image,marks);

    //我们再来看一下分水岭算法之后的矩阵 marks 里是什么东西
    Mat afterWatershed;
    convertScaleAbs(marks,afterWatershed);

    //对每一个区域进行颜色填充
    Mat PerspectiveImage=Mat::zeros(image.size(),CV_8UC3);
    for(int i=0;i<marks.rows;i++)
    {
        for(int j=0;j<marks.cols;j++)
        {
            int index=marks.at<int>(i,j);
            if(marks.at<int>(i,j)==-1)
            {
                PerspectiveImage.at<Vec3b>(i,j)=Vec3b(255,255,255);
            }
            else
            {
                PerspectiveImage.at<Vec3b>(i,j) =RandomColor(index);
            }
        }
    }
    //分割并填充颜色的结果跟原始图像融合
    Mat wshed;
    addWeighted(image,0.4,PerspectiveImage,0.6,0,wshed);
    cvtColor(wshed, wshed, CV_BGR2RGB);
    QImage IMG = cvMat2QImage(wshed);
    QImage Image=ImageCenter(IMG,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("图像分割完成！"));
    global_img = IMG.copy();
```

```
    waitKey();
}

void MainWindow::on_btn_back_clicked()
{
    QImage IMG = back_img.copy();
    QImage Image=ImageCenter(IMG,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("撤销完成！"));
}


void MainWindow::on_btn_open_clicked()
{
    QImage img=global_img.copy();
    back_img = img.copy();
    Mat srcImage = QImage2cvMat(img);

    Mat element;
    element = getStructuringElement(MORPH_RECT, Size(15, 15));

    Mat dstImage;
    morphologyEx(srcImage, dstImage, MORPH_OPEN, element);

    QImage IMG = cvMat2QImage(dstImage);
    QImage Image=ImageCenter(IMG,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("图像开运算完成！"));
    waitKey(0);
    global_img = IMG.copy();
}

void MainWindow::on_btn_close_clicked()
{
    QImage img=global_img.copy();
    back_img = img.copy();
    Mat srcImage = QImage2cvMat(img);

    Mat element;
    element = getStructuringElement(MORPH_RECT, Size(15, 15));

    Mat dstImage;
    morphologyEx(srcImage, dstImage, MORPH_CLOSE, element);
    QImage IMG = cvMat2QImage(dstImage);
    QImage Image=ImageCenter(IMG,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("图像闭运算完成！"));
    waitKey(0);
```

```
    global_img = IMG.copy();
}

//连接图像中断裂的边缘
vector<Point> breakImage(Mat &src, Mat &dst, int DisThre)
{
    if (dst.data != src.data) src.copyTo(dst);

    vector<Point> pointxy;
    Point ptPoint;
    Size size = src.size();
    int nSize, dx, dy;
    float distance;

    for (int i = 1; i < size.height - 1; i++)
    {
        uchar *dataPre = dst.ptr<uchar>(i - 1);
        uchar *dataCurr = dst.ptr<uchar>(i);
        uchar *dataNext = dst.ptr<uchar>(i + 1);
        for (int j = 1; j < size.width - 1; j++)
        {
            //  p9 p2 p3
            //  p8 p1 p4
            //  p7 p6 p5
            int p1 = dataCurr[j];
            if (p1 != 255) continue;
            int p2 = dataPre[j];
            int p3 = dataPre[j + 1];
            int p4 = dataCurr[j + 1];
            int p5 = dataNext[j + 1];
            int p6 = dataNext[j];
            int p7 = dataNext[j - 1];
            int p8 = dataCurr[j - 1];
            int p9 = dataPre[j - 1];

            if (p1 == 255)
            {
                if ((p2 + p3 + p4 + p5 + p6 + p7 + p8 + p9) == 255)
                {
                    printf("p1 = 1");
                    ptPoint.x = j;
                    ptPoint.y = i;
                    pointxy.push_back( ptPoint );
                    printf("x:%d y:%d\n", j, i);
                }
            }

        }
    }
    nSize = (int)pointxy.size();
    printf("size:%d\n", nSize);
```

```
    for (int i = 0; i < nSize - 1; i++)
    {
        for (int j = i + 1; j < nSize; j++)
        {
            dx = pointxy[i].x - pointxy[j].x;
            dy = pointxy[i].y - pointxy[j].y;
            distance = (float)(dx * dx + dy * dy);
            if (distance <= DisThre * DisThre)
            {
                line(dst, pointxy[i], pointxy[j], Scalar(255, 255, 255));
            }
        }
    }
    return pointxy;
}

void MainWindow::on_btn_edge_clicked()
{
    QImage img=global_img.copy();
    back_img = img.copy();
    Mat imageSource = QImage2cvMat(img);
    cvtColor(imageSource, imageSource, CV_RGB2BGR);
    cvtColor(imageSource, imageSource, CV_BGR2GRAY);
    Mat image;
    GaussianBlur(imageSource, image, Size(3, 3), 0);//高斯滤波
    Canny(image, image, 100, 250);//canny 算子边缘检测
    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;
    findContours(image, contours, hierarchy,RETR_EXTERNAL, CV_CHAIN_APPROX_NONE, Point());
    Mat imageContours = Mat::zeros(image.size(), CV_8UC1);
    Mat Contours = Mat::zeros(image.size(), CV_8UC1);  //绘制
    int tep=0;
    for (size_t i = 0; i < contours.size(); i++)
    {
        //contours[i]代表的是第 i 个轮廓，contours[i].size()代表的是第 i 个轮廓上所有的像素点数
        for (size_t j = 0; j < contours[i].size(); j++)
        {
            //绘制出 contours 向量内所有的像素点
            Point P = Point(contours[i][j].x, contours[i][j].y);
            Contours.at<uchar>(P) = 255;
        }
        tep++;
        cout << "向量 hierarchy 的第" << i << "个元素内容为：" << endl << hierarchy[i] << endl << endl;
        //绘制轮廓
        drawContours(imageContours, contours, i, Scalar(255), 1, 8, hierarchy);
    }
    QImage IMG = cvMat2QImage(imageContours);
    QImage Image=ImageCenter(IMG,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
```

```
    statusBar()->showMessage(tr("图像节理数量统计完成！"));
    global_img = IMG.copy();
    waitKey(0);
}

/// 全局变量
Mat SRC, erosion_dst, dilation_dst;
int erosion_elem = 0;
int erosion_size = 0;
int dilation_elem = 0;
int dilation_size = 0;
int const max_elem = 2;
int const max_kernel_size = 10;

/** Function Headers */
void Erosion(int, void*);//腐蚀操作
void Dilation(int, void*);//膨胀操作

void MainWindow::on_btn_dilate_clicked()
{
    QImage img=global_img.copy();
    back_img = img.copy();
    SRC = QImage2cvMat(img);
    cvtColor(SRC, SRC, CV_RGB2BGR);
    /// Load an image
    if (!SRC.data)
    {
        QMessageBox::warning(nullptr,"提示", "请先打开图片！",QMessageBox::Yes |
QMessageBox::Yes);
    }
    /// Create windows
    namedWindow("Dilation Demo", WINDOW_AUTOSIZE);
    /// 膨胀操作滚动条
    createTrackbar("Element:", "Dilation Demo", &dilation_elem, max_elem, Dilation);
    createTrackbar("Kernel", "Dilation Demo", &dilation_size, max_kernel_size, Dilation);
    // 开始
    Dilation(0, 0);
    waitKey(0);
}

void MainWindow::on_btn_erode_clicked()
{
    QImage img=global_img.copy();
    back_img = img.copy();
    SRC = QImage2cvMat(img);
    cvtColor(SRC, SRC, CV_RGB2BGR);
    /// Load an image
    if (!SRC.data)
    {
        QMessageBox::warning(nullptr,"提示", "请先打开图片！",QMessageBox::Yes |
```

```
QMessageBox::Yes);
    }
    /// Create windows
    namedWindow("Erosion Demo", WINDOW_AUTOSIZE);
    /// 腐蚀操作滚动条
    createTrackbar("Element:", "Erosion Demo", &erosion_elem, max_elem, Erosion);
    createTrackbar("Kernel", "Erosion Demo", &erosion_size, max_kernel_size, Erosion);
    // 开始
    Erosion(0, 0);
    waitKey(0);
}

/**  @function Erosion  */
void Erosion(int, void*)
{
    int erosion_type;

    if (erosion_elem == 0)
    {
        erosion_type = MORPH_RECT;  //矩形结构元素
    }
    else if (erosion_elem == 1)
    {
        erosion_type = MORPH_CROSS;  //十字结构元素
    }
    else
    {
        erosion_type = MORPH_ELLIPSE;//椭圆结构元素
    }
    //生成核（结构元素）
    Mat element = getStructuringElement(erosion_type, Size(2 * erosion_size + 1, 2 * erosion_size + 1),
    Point(erosion_size, erosion_size));

    //腐蚀操作
    //erode(SRC, erosion_dst, element);
    erode(SRC, erosion_dst, element, Point(-1, -1), -1);
    imshow("Erosion Demo", erosion_dst);
    imwrite("/myImage/images/erode.jpg",erosion_dst);
}

/** @function Dilation */
void Dilation(int, void*)
{
    int dilation_type;
    if (dilation_elem == 0){
        //矩形结构元素
        dilation_type = MORPH_RECT;
    }
    else if (dilation_elem == 1){
        //十字结构元素
```

```
        dilation_type = MORPH_CROSS;
    }
    else
        //椭圆结构元素
        dilation_type = MORPH_ELLIPSE;
    //生成核（结构元素）
    Mat element = getStructuringElement(dilation_type, Size(2 * dilation_size + 1, 2 * dilation_size + 1),
    Point(dilation_size, dilation_size));
    //腐蚀操作
    //dilate(SRC, dilation_dst, element);
    dilate(SRC, dilation_dst, element, Point(-1, -1), -1);
    imshow("Dilation Demo", dilation_dst);
    imwrite("/myImage/images/Dilation.jpg",dilation_dst);
}

//-------------------------------漫水滤波---------------------------//
void MainWindow::on_btn_qumaoci_clicked()
{
    Mat dst;
    QImage image = global_img.copy();
    back_img = image.copy();
    Mat im_in = QImage2cvMat(image);
    cvtColor(im_in,im_in,CV_BGR2GRAY);
    cv::Mat im_th;
    cv::threshold(im_in, im_th, 220, 255, cv::THRESH_BINARY_INV);
    // Floodfill from point (0, 0)
    cv::Mat im_floodfill = im_th.clone();
    cv::floodFill(im_floodfill, cv::Point(0,0), cv::Scalar(0));
    // Invert floodfilled image
    cv::Mat im_floodfill_inv;
    cv::bitwise_not(im_floodfill, im_floodfill_inv);
    // Combine the two images to get the foreground.
    cv::Mat im_out = (im_th | im_floodfill_inv);
    // Display images
    imwrite("Thresholded_Image.jpg", im_th);
    QImage img = cvMat2QImage(im_th);
    QImage Image=ImageCenter(img,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("去毛刺成功！"));
    global_img = img.copy();
}

void MainWindow::on_btn_hough_clicked()
{
    Mat dst;
    QImage image = global_img.copy();
    back_img = image.copy();
    Mat srcImage = QImage2cvMat(image);
    Mat midImage, dstImage; // 临时变量和目标图的定义
```

```
    // 2. 进行边缘检测和转化为灰度图
    Canny(srcImage, midImage, 50, 200, 3);
    cvtColor(midImage, dstImage, COLOR_GRAY2BGR);
    // 3. 进行霍夫线转换
    vector<Vec4i> lines; // 定义一个矢量结构 lines 用于存放得到的线段矢量集合
    HoughLinesP(midImage, lines, 1, CV_PI / 360, 80, 50, 10);
    // 过滤掉长度小于阈值的线段
    int minLineLengthThreshold = 45; // 根据需要调整阈值
    vector<Vec4i> filteredLines;
    for (size_t i = 0; i < lines.size(); i++) {
        Vec4i l = lines[i];
        double lineLength = sqrt(pow(l[2] - l[0], 2) + pow(l[3] - l[1], 2));
        if (lineLength >= minLineLengthThreshold) {
            filteredLines.push_back(l);
        }
    }

    // 合并距离相近的线段
    int mergeDistanceThreshold = 150; // 调整距离阈值
    vector<Vec4i> mergedLines;
    for (size_t i = 0; i < filteredLines.size(); i++) {
        Vec4i currentLine = filteredLines[i];
        bool merged = false;
        for (size_t j = 0; j < mergedLines.size(); j++) {
            Vec4i mergedLine = mergedLines[j];
            double distance = sqrt(pow(currentLine[0] - mergedLine[2], 2) + pow(currentLine[1] -
mergedLine[3], 2));
            // 如果距离小于阈值，认为它们是相近的线段，合并
            if (distance < mergeDistanceThreshold) {
                mergedLine[2] = currentLine[2];
                mergedLine[3] = currentLine[3];
                merged = true;
                break;
            }
        }

        // 如果不与任何已合并线段相邻，添加到合并线段集合
        if (!merged) {
            mergedLines.push_back(currentLine);
        }
    }

    // 标记后的线段延长
    int extensionLength = 100; // 调整加长的长度
    vector<Vec4i> extendedLines;
    for (size_t i = 0; i < mergedLines.size(); i++) {
        Vec4i l = mergedLines[i];

        // 计算线段的角度
        double angle = atan2(l[3] - l[1], l[2] - l[0]);
```

```
        // 延长线段
        int x1 = l[0] - extensionLength * cos(angle);
        int y1 = l[1] - extensionLength * sin(angle);
        int x2 = l[2] + extensionLength * cos(angle);
        int y2 = l[3] + extensionLength * sin(angle);
        extendedLines.push_back(Vec4i(x1, y1, x2, y2));
    }

    // 过滤掉长度小于阈值的线段
    minLineLengthThreshold = 80; // 根据需要调整阈值
    for (size_t i = 0; i < lines.size(); i++) {
        Vec4i l = lines[i];
        double lineLength = sqrt(pow(l[2] - l[0], 2) + pow(l[3] - l[1], 2));
        if (lineLength >= minLineLengthThreshold) {
            filteredLines.push_back(l);
        }
    }

    // 再次合并距离相近的线段
    vector<Vec4i> finalMergedLines;
    for (size_t i = 0; i < extendedLines.size(); i++) {
        Vec4i currentLine = extendedLines[i];
        bool merged = false;
        for (size_t j = 0; j < finalMergedLines.size(); j++) {
            Vec4i mergedLine = finalMergedLines[j];
            double distance = sqrt(pow(currentLine[0] - mergedLine[2], 2) + pow(currentLine[1] -
mergedLine[3], 2));

            // 如果距离小于阈值，认为它们是相近的线段，合并
            if (distance < mergeDistanceThreshold) {
                mergedLine[2] = currentLine[2];
                mergedLine[3] = currentLine[3];
                merged = true;
                break;
            }
        }

        // 如果不与任何已合并线段相邻，添加到最终合并线段集合
        if (!merged) {
            finalMergedLines.push_back(currentLine);
        }
    }

    // 绘制最终合并后的线段
    for (size_t i = 0; i < finalMergedLines.size(); i++) {
        Vec4i l = finalMergedLines[i];
        line(dstImage, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0, 255, 0), 20, LINE_AA);
    }
```

```
    int tep = finalMergedLines.size(); //  使用合并后的线段数量作为计数
    cout << tep << endl;
    crack_num = tep;

    ui->crack_num_all->setText(QString::number(tep));

    QImage img = cvMat2QImage(dstImage);
    QImage Image = ImageCenter(img, ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("霍夫变换成功！"));
    global_img = img.copy();
}

void MainWindow::on_btn_tongtailvbo_clicked()
{
    QImage img=global_img.copy();
    back_img = img.copy();
    Mat src = QImage2cvMat(img);
    Mat gray_src, edge, LOGdst;

    if (!src.data) { printf("could not load image..."); }
    cvtColor(src, gray_src, CV_BGR2GRAY);

    Mat gauss_output, gauss_output_2;
    //定义 x 方向的模糊因子
    float sigma_x=20.0;       //该参数决定了邻接像素的权重
    float sigma_y= sigma_x;
    //不同的高斯核卷积，实现了不同尺度特征，可以近似 LoG
    GaussianBlur(gray_src, gauss_output, Size(3, 3), sigma_x, sigma_y);
    GaussianBlur(gray_src, gauss_output_2, Size(11, 11), sigma_x, sigma_y);

    //imshow("gauss_output", gauss_output);
    //基于 LoG 方法
    Laplacian(gauss_output, LOGdst, -1, 3, 1.0, 0.0);
    //imshow("LoGdst", LOGdst);

    //基于 DoG  近似
    Mat DOGdst(src.size(), CV_32S);
    subtract(gauss_output_2, gauss_output, DOGdst);

    convertScaleAbs(DOGdst, DOGdst);
    normalize(DOGdst, DOGdst, 0, 255, NORM_MINMAX, CV_8UC1);

    imshow("DoGdst", DOGdst);
    //基于指针的操作比采用 at 会快一个数量级

    //基于自定义模板卷积核的实现,在经过 NMS 后效果或许会更好

    Mat LoG_kernel = (Mat_<signed>(5, 5) << 0,    0,    -1,    0, 0,
```

```
                                                    0,   -1, -2, -1, 0,
                                                    -1, -2, 16, -2, -1,
                                                    0,   -1,  -2, -1, 0,
                                                    0,   0,   -1,   0, 0);
    Mat self_define, gauss_output2;
    GaussianBlur(gray_src, gauss_output2, Size(5, 5),0,0);
    filter2D(gauss_output2, self_define, CV_32FC1, LoG_kernel);
    convertScaleAbs(self_define, self_define);
    normalize(self_define, self_define, 0, 255, NORM_MINMAX, CV_8UC1);
    imshow("self_define", self_define);

    QImage IMG = cvMat2QImage(DOGdst);
    QImage Image=ImageCenter(IMG,ui->label_show);
    ui->label_show->setPixmap(QPixmap::fromImage(Image));
    ui->label_show->setAlignment(Qt::AlignCenter);
    statusBar()->showMessage(tr("滤波完成！"));
    waitKey(0);
    global_img = IMG.copy();

}

int Thinning(unsigned char * ucBinedImg, unsigned char * ucThinnedImage, long lWidth, long lHeight, long
lIterativeLimit)
{
    if(ucBinedImg == NULL)
        return -1;

    if(ucThinnedImage == NULL)
        return -2;

    if(lIterativeLimit == -1)
        lIterativeLimit = 60000;

    unsigned char x1, x2, x3, x4, x5, x6, x7, x8; //xp;
    unsigned char g1, g2, g3, g4;
    unsigned char b1, b2, b3, b4;
    unsigned char np1, np2, npm;
    unsigned char *pUp, *pDown, *pImg;
    long       lDeletedPoints = 0;

    // set border
    memcpy(ucThinnedImage, ucBinedImg, lWidth*lHeight);

    for(long it=0; it<lIterativeLimit; it++)
    {
        lDeletedPoints = 0;
        for(long i=1; i<lHeight-1; i++)
        {
            // init neighborhood
            pUp = ucBinedImg + (i-1)*lWidth;
            pImg = ucBinedImg + i*lWidth ;
```

```
pDown = ucBinedImg + (i+1)*lWidth ;
for( long j=1; j<lWidth-1; j++)
{
    pUp++;
    pImg++;
    pDown++;

    if(!*pImg) continue;

    x6 = *(pUp-1);
    x5 = *(pImg-1);
    x4 = *(pDown-1);

    x7 = *pUp;
    //xp = *pImg;
    x3 = *pDown;

    x8 = *(pUp+1);
    x1 = *(pImg + 1);
    x2 = *(pDown + 1);

    b1 = !x1 && (x2 == 1 || x3 == 1);
    b2 = !x3 && (x4 == 1 || x5 == 1);
    b3 = !x5 && (x6 == 1 || x7 == 1);
    b4 = !x7 && (x8 == 1 || x1 == 1);

    g1 = (b1 + b2 + b3 + b4) == 1;
    np1 = x1|| x2;
    np1 += x3 || x4;
    np1 += x5 || x6;
    np1 += x7 || x8;
    np2 = x2|| x3;
    np2 += x4 || x5;
    np2 += x6 || x7;
    np2 += x8 || x1;

    npm = np1>np2?np2:np1;
    g2 = npm>=2 && npm<=3;

    g3 = (x1 && (x2 || x3 || !x8)) == 0;
    g4 = (x5 && (x6 || x7 || !x4)) == 0;

     // first part
     if(g1 && g2 && g3)
     {
          // delete this point
          ucThinnedImage[lWidth*i + j] = 0;
          ++lDeletedPoints;
     }
}
```

```
  }
//syn
memcpy(ucBinedImg, ucThinnedImage, lWidth*lHeight);

for(long i=1; i<lHeight-1; i++)
{
    // init neighborhood
    pUp = ucBinedImg + (i-1)*lWidth;
    pImg = ucBinedImg + i*lWidth ;
    pDown = ucBinedImg + (i+1)*lWidth ;

    for( long j=1; j<lWidth-1; j++)
    {
        pUp++;
        pImg++;
        pDown++;
        if(!*pImg)
            continue;
        x6 = *(pUp-1);
        x5 = *(pImg-1);
        x4 = *(pDown-1);
        x7 = *pUp;

        //xp = *pImg;
        x3 = *pDown;
        x8 = *(pUp+1);
        x1 = *(pImg + 1);
        x2 = *(pDown + 1);
        b1 = !x1 && (x2 == 1 || x3 == 1);
         b2 = !x3 && (x4 == 1 || x5 == 1);
         b3 = !x5 && (x6 == 1 || x7 == 1);
         b4 = !x7 && (x8 == 1 || x1 == 1);
         g1 = (b1 + b2 + b3 + b4) == 1;
         np1 = x1|| x2;
         np1 += x3 || x4;
         np1 += x5 || x6;
         np1 += x7 || x8;
         np2 = x2|| x3;
         np2 += x4 || x5;
         np2 += x6 || x7;
         np2 += x8 || x1;
         npm = np1>np2?np2:np1;
         g2 = npm>=2 && npm<=3;
         g3 = (x1 && (x2 || x3 || !x8)) == 0;
         g4 = (x5 && (x6 || x7 || !x4)) == 0;

         // second part
         if(g1 && g2 && g4)
         {
             // delete this point
             ucThinnedImage[lWidth*i + j] = 0;
```

```
                        ++lDeletedPoints;
                    }
                }
            }
            //syn
            memcpy(ucBinedImg, ucThinnedImage, lWidth*lHeight);
            // if no points to be deleted
            if(lDeletedPoints == 0)
                break;
        }

        // clear edge bar
        for(long i=0; i<lHeight; i++)
        {
            for(long j=0; j<lWidth; j++)
            {
                if(i<16)
                    ucThinnedImage[i*lWidth+j] = 0;
                else if(i>=lHeight-16)
                    ucThinnedImage[i*lWidth+j] = 0;
                else if(j<16)
                    ucThinnedImage[i*lWidth+j] = 0;
                else if(j>=lWidth-16)
                    ucThinnedImage[i*lWidth+j] = 0;
            }
        }
        return 0;
}

void Thinning(Mat& src,Mat& dst,long IterativeLimit=-1)
{
    Mat bin_img=src&1;
    if(!dst.empty()){dst.release();}
    dst=Mat::zeros(src.size(),CV_8UC1);
    Thinning(bin_img.data,dst.data,bin_img.cols,bin_img.rows,IterativeLimit);
    dst*=255;
}

void MainWindow::on_doubleSpinBox_valueChanged()
{
    QString::number(ui->doubleSpinBox->value());
    RC=ui->doubleSpinBox->value();
}

void MainWindow::on_doubleSpinBox_K1_valueChanged()
{
    QString::number(ui->doubleSpinBox_K1->value());
    K1=ui->doubleSpinBox_K1->value();
}

void MainWindow::on_doubleSpinBox_K2_valueChanged()
```

```
{
    QString::number(ui->doubleSpinBox_K2->value());
    K2=ui->doubleSpinBox_K2->value();
}

void MainWindow::on_doubleSpinBox_K3_valueChanged()
{
    QString::number(ui->doubleSpinBox_K3->value());
    K3=ui->doubleSpinBox_K3->value();
}

void MainWindow::on_doubleSpinBox_area_valueChanged()
{
    QString::number(ui->doubleSpinBox_area->value());
    area=ui->doubleSpinBox_area->value();
}

void MainWindow::on_pushButton_5_clicked()
{
    int Jv=crack_num/area;
    ui->crack_num->setText(QString::number(Jv));
    double Kv=0, BQ=0, BQ_=0;
    if(Jv<3) Kv=0.85;
    else if(3<Jv && Jv<10) Kv=0.65;
    else if(10<Jv && Jv<20) Kv=0.45;
    else if(20<Jv && Jv<35) Kv=0.25;
    else Kv=0.15;

    if(RC>90*Kv+30)
    {
        BQ=100+3*(90*Kv+30)+250*Kv;
        cout << RC << endl;
        cout << Kv << endl;
        cout << BQ << endl;
    }
    else if(Kv>0.04*RC+0.4)
    {
        BQ=100+3*RC+250*(0.04*RC+0.4);
        cout << RC << endl;
        cout << Kv << endl;
        cout << BQ << endl;
    }
    else printf("wrong");

    BQ_=BQ-100*(K1+K2+K3);

    if(BQ>550 || BQ_>550) ui->crack_num_2->setText(tr("I级"));
    else if((451<=BQ_ && BQ_<=550) || (451<=BQ && BQ<=550)) ui->crack_num_2->setText(tr("II级"));
    else if((351<=BQ_ && BQ_<=450) || (351<=BQ && BQ<=450)) ui->crack_num_2->setText(tr("III级"));
    else if((251<=BQ_ && BQ_<=350) || (251<=BQ && BQ<=350)) ui->crack_num_2->setText(tr("IV级"));
    else ui->crack_num_2->setText(tr("V级"));
```

```
}

void MainWindow::on_doubleSpinBox_d_valueChanged()
{
    QString::number(ui->doubleSpinBox_d->value());
    d=ui->doubleSpinBox_d->value();
}

void MainWindow::on_doubleSpinBox_d1_valueChanged()
{
    QString::number(ui->doubleSpinBox_d1->value());
    d1=ui->doubleSpinBox_d1->value();
}

void MainWindow::on_horizontalSlider_E_valueChanged()
{
    value_e=ui->horizontalSlider_E->value();
    ui->label_E_NUM->setText(QString::number(value_e));
}

void MainWindow::on_horizontalSlider_V_valueChanged()
{
    value_v=ui->horizontalSlider_V->value();
    ui->label_V_NUM->setText(QString::number(value_v));
}

void MainWindow::on_doubleSpinBox_delta_valueChanged()
{
    QString::number(ui->doubleSpinBox_delta->value());
    delta=ui->doubleSpinBox_delta->value();
}

void MainWindow::on_doubleSpinBox_r_valueChanged()
{
    QString::number(ui->doubleSpinBox_r->value());
    r=ui->doubleSpinBox_r->value();
}

void MainWindow::on_doubleSpinBox_L_valueChanged()
{
    QString::number(ui->doubleSpinBox_L->value());
    L=ui->doubleSpinBox_L->value();
}

void MainWindow::on_doubleSpinBox_S_valueChanged()
{
    QString::number(ui->doubleSpinBox_S->value());
    S=ui->doubleSpinBox_S->value();
}

void MainWindow::on_doubleSpinBox_m_valueChanged()
```

```
{
    QString::number(ui->doubleSpinBox_m->value());
    m=ui->doubleSpinBox_m->value();
}

void MainWindow::on_doubleSpinBox_x_valueChanged()
{
    QString::number(ui->doubleSpinBox_x->value());
    X=ui->doubleSpinBox_x->value();
}

void MainWindow::on_doubleSpinBox_miu_valueChanged()
{
    QString::number(ui->doubleSpinBox_miu->value());
    miu=ui->doubleSpinBox_miu->value();
}

void MainWindow::on_doubleSpinBox_p_valueChanged()
{
    QString::number(ui->doubleSpinBox_p->value());
    p=ui->doubleSpinBox_p->value();
}

void MainWindow::on_pushButton_6_clicked()
{
    double D=0;
    D=d/d1;
    QString str_D = QString::number(D);
    ui->show_D->setText(str_D);

    double V=0;
    V=value_v*(d/10);
    ui->show_V->setText(QString::number(V));

    double E=0;
    E=value_e*(d/10);
    ui->show_E->setText(QString::number(E));

    double K=0;
    K=E/V;
    ui->show_K->setText(QString::number(K));

    double q=0;
    q=(3.1415926535/4)*(d1/10)*(d1/10)*delta;
    ui->show_q->setText(QString::number(q));

    double Q=0;
    Q=r*L*S;
    ui->show_Q->setText(QString::number(Q));

    double N=0;
```

```
    N=(r*S*m*miu)/(X*p);
    ui->show_N->setText(QString::number(ceil(N)));
}

// mainwindow 文件
namespace Ui { class MainWindow; }

class MainWindow : public QMainWindow
{
    Q_OBJECT // 宏，使用 Qt 信号与槽机制必须添加
public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    void showascii();
    int index =0;//图片 index
    int crack_num=0;
    double RC=0;
    double K1=0, K2=0, K3=0;
    double area=0;
    double d=0, d1=0;
    int value_v=0, value_e=0;
    double delta=0, r=0, L=0, S=0, m=0, X=0, miu=0, p=0;
    QString stom(int s);
    QImage junzhi(QImage image);
    QImage gamma(QImage image);
    QImage gauss(QImage image,double photometricStandardDeviation, double spatialDecay);
    QImage cvMat2QImage(const cv::Mat& mat);
    Mat QImage2cvMat(QImage image);

private:
    Ui::MainWindow *ui; // Ui::MainWindow 类型的一个指针，指向可视化的界面
    bool language=true;
    bool isstart=false;
    QString origin_path;//目前处理的图片的原图
    QString srcDirPathListS;;//目前处理的图片的原图
    QString OpenFilePath;
    QMessageBox customMsgBox;
    QString srcDirPath;
    QImage global_img;
    QImage back_img;

    float thresh=1;//滑块值
    int x,y,x1,y1;//鼠标点
    int flag;//用于判断哪个图像处理操作使用到了滑块
    int MORPH=0;//形态学内核：MORPH_RECT (0) 矩形、 MORPH_CROSS (1)十字交叉型 、
MORPH_ELLIPSE (2)椭圆型
};

#endif // MAINWINDOW_H
```