**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