实验名称：基于 PyQt5 的图像处理图形界面开发

实验目的

利用 Python 开发专用的 PyQt5 界面库，制作图像处理的图形用户界面，将各种图像处理的功能整合成一个软件，方便操作演示。

实验原理

PyQt5 有一个独一无二的信号和槽机制来处理事件。信号和槽用于对象之间的通信。当指定事件发生，一个事件信号会被发射。槽可以被任何 Python 脚本调用。当和槽连接的信号被发射时，槽会被调用。

实验内容

1、本图形用户界面主要实现三个功能：图像分割、图像增强、边缘检测；

2、使用 designer 软件设计界面构成，设计三个单选功能按钮（图像分割、图像增强、边缘检测），每个功能按钮下链接四个多选按钮，用于实现功能下不同的方法和参数，更好对比效果。每个功能下的多选按钮与功能按钮单独对应。

图像分割下有：OTSU 法（大津法）、最大熵分割法、迭代阈值分割法、马尔可夫随机场法(其中，由于马尔可夫随机场特殊的分类特性，默认将图像分割成四类)；

图像增强下有：标准直方图均衡(HE)、限制对比度自适应直方图均衡(CLAHE)、单尺度 Retinex 算法(SSR)、多尺度 Retinex 算法(MSR) ；

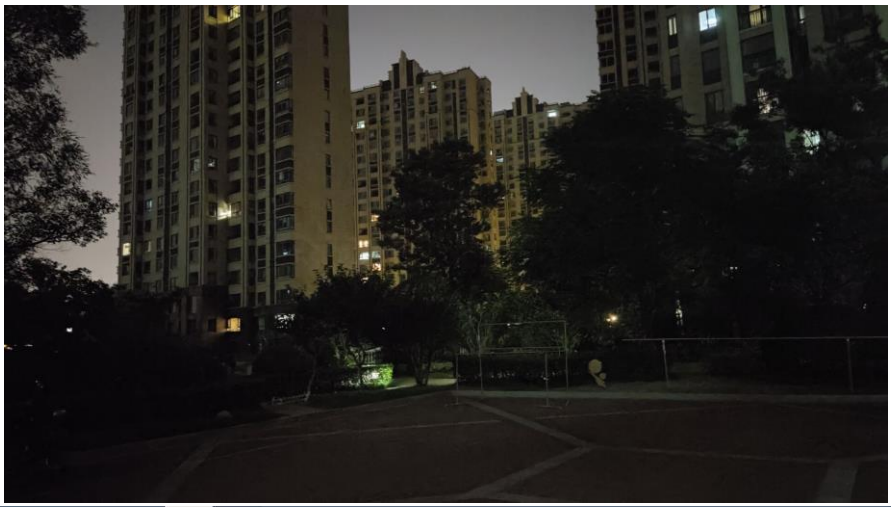
边缘检测下有：Roberts 算子、Sobel 算子、Canny 算子、Laplacian 算子：

另外，还设计有选择文件按钮和开始按钮，便于用户直接操作。

3、用户可直接通过鼠标点击的方法实现上述功能及对应方法，并进行对比。以 Lena 图为例，实现效果如下：

以我自己拍的我家小区夜景为例，演示图像增强效果：

原图：



增强后图像

由于夜景图像较为特殊，在此不做图像分割和边缘检测的演示。

另，各方法效果对比在此不做分析，可见实验报告《图像分割各经典方法的对比研究实验报告》及《图像增强各经典方法的对比研究实验报告》。

实验源代码

配置文件 image_processing.py

```
# -*- coding: utf-8 -*-


# Form implementation generated from reading ui file 'image_processing.ui'
#
# Created by: PyQt5 UI code generator 5.15.4
#
# WARNING: Any manual changes made to this file will be lost when pyuic5 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt5 import QtCore, QtWidgets,QtGui
from PyQt5.QtWidgets import *


import PIL
# from io import BytesIO
import cv2
import numpy as np
import matplotlib as plt


class Ui_MainWindow(QWidget):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1105, 815)
        MainWindow.setInputMethodHints(QtCore.Qt.ImhNone)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayout = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout.setObjectName("verticalLayout")
```

```
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        spacerItem = QtWidgets.QSpacerItem(38, 17, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout.addItem(spacerItem)
        self.radioButton = QtWidgets.QRadioButton(self.centralwidget)
        self.radioButton.setStyleSheet("font: 20pt \"宋体\";")
        self.radioButton.setObjectName("radioButton")
        self.horizontalLayout.addWidget(self.radioButton)
        spacerItem1 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout.addItem(spacerItem1)
        self.radioButton_2 = QtWidgets.QRadioButton(self.centralwidget)
        self.radioButton_2.setStyleSheet("font: 20pt \"宋体\";")
        self.radioButton_2.setObjectName("radioButton_2")
        self.horizontalLayout.addWidget(self.radioButton_2)
        spacerItem2 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout.addItem(spacerItem2)
        self.radioButton_3 = QtWidgets.QRadioButton(self.centralwidget)
        self.radioButton_3.setStyleSheet("font: 20pt \"宋体\";")
        self.radioButton_3.setObjectName("radioButton_3")
        self.horizontalLayout.addWidget(self.radioButton_3)
        spacerItem3 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout.addItem(spacerItem3)
        self.verticalLayout.addLayout(self.horizontalLayout)
        self.gridLayout_3 = QtWidgets.QGridLayout()
        self.gridLayout_3.setObjectName("gridLayout_3")
        spacerItem4 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
```

```
QtWidgets.QSizePolicy.Minimum)
            self.gridLayout_3.addItem(spacerItem4, 0, 4, 1, 1)
            self.checkBox = QtWidgets.QCheckBox(self.centralwidget)
            self.checkBox.setEnabled(True)
            self.checkBox.setTabletTracking(False)
            self.checkBox.setAutoFillBackground(False)
            self.checkBox.setStyleSheet("font: 16pt \"宋体\";")
            self.checkBox.setInputMethodHints(QtCore.Qt.ImhHiddenText)
            self.checkBox.setCheckable(True)
            self.checkBox.setObjectName("checkBox")
            self.gridLayout_3.addWidget(self.checkBox, 0, 1, 1, 1)
            spacerItem5 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
            self.gridLayout_3.addItem(spacerItem5, 0, 6, 1, 1)
            spacerItem6 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
            self.gridLayout_3.addItem(spacerItem6, 0, 2, 1, 1)
            self.checkBox_4 = QtWidgets.QCheckBox(self.centralwidget)
            self.checkBox_4.setStyleSheet("font: 16pt \"宋体\";")
            self.checkBox_4.setObjectName("checkBox_4")
            self.gridLayout_3.addWidget(self.checkBox_4, 0, 7, 1, 1)
            spacerItem7 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
            self.gridLayout_3.addItem(spacerItem7, 0, 0, 1, 1)
            self.checkBox_2 = QtWidgets.QCheckBox(self.centralwidget)
            self.checkBox_2.setEnabled(True)
            self.checkBox_2.setStyleSheet("font: 16pt \"宋体\";")
            self.checkBox_2.setObjectName("checkBox_2")
            self.gridLayout_3.addWidget(self.checkBox_2, 0, 3, 1, 1)
            self.checkBox_3 = QtWidgets.QCheckBox(self.centralwidget)
```

```
        self.checkBox_3.setStyleSheet("font: 16pt \"宋体\";")
        self.checkBox_3.setObjectName("checkBox_3")
        self.gridLayout_3.addWidget(self.checkBox_3, 0, 5, 1, 1)
        spacerItem8 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.gridLayout_3.addItem(spacerItem8, 0, 8, 1, 1)
        self.verticalLayout.addLayout(self.gridLayout_3)
        self.gridLayout = QtWidgets.QGridLayout()
        self.gridLayout.setVerticalSpacing(4)
        self.gridLayout.setObjectName("gridLayout")
        self.checkBox_5 = QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox_5.setEnabled(True)
        self.checkBox_5.setTabletTracking(False)
        self.checkBox_5.setAutoFillBackground(False)
        self.checkBox_5.setStyleSheet("font: 16pt \"宋体\";")
        self.checkBox_5.setInputMethodHints(QtCore.Qt.ImhHiddenText)
        self.checkBox_5.setCheckable(True)
        self.checkBox_5.setObjectName("checkBox_5")
        self.gridLayout.addWidget(self.checkBox_5, 0, 1, 1, 1)
        spacerItem9 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.gridLayout.addItem(spacerItem9, 0, 5, 1, 1)
        self.checkBox_9 = QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox_9.setStyleSheet("font: 16pt \"宋体\";")
        self.checkBox_9.setObjectName("checkBox_9")
        self.gridLayout.addWidget(self.checkBox_9, 2, 4, 1, 1)
        spacerItem10 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.gridLayout.addItem(spacerItem10, 0, 0, 1, 1)
        spacerItem11 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
```

```
QtWidgets.QSizePolicy.Minimum)
            self.gridLayout.addItem(spacerItem11, 0, 2, 1, 2)
            spacerItem12 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
            self.gridLayout.addItem(spacerItem12, 2, 2, 1, 2)
            self.checkBox_8 = QtWidgets.QCheckBox(self.centralwidget)
            self.checkBox_8.setStyleSheet("font: 16pt \"宋体\";")
            self.checkBox_8.setObjectName("checkBox_8")
            self.gridLayout.addWidget(self.checkBox_8, 2, 1, 1, 1)
            self.checkBox_7 = QtWidgets.QCheckBox(self.centralwidget)
            self.checkBox_7.setStyleSheet("font: 16pt \"宋体\";")
            self.checkBox_7.setObjectName("checkBox_7")
            self.gridLayout.addWidget(self.checkBox_7, 0, 4, 1, 1)
            self.verticalLayout.addLayout(self.gridLayout)
            self.gridLayout_2 = QtWidgets.QGridLayout()
            self.gridLayout_2.setObjectName("gridLayout_2")
            self.checkBox_10 = QtWidgets.QCheckBox(self.centralwidget)
            self.checkBox_10.setEnabled(True)
            self.checkBox_10.setTabletTracking(False)
            self.checkBox_10.setAutoFillBackground(False)
            self.checkBox_10.setStyleSheet("font: 16pt \"宋体\";")
            self.checkBox_10.setInputMethodHints(QtCore.Qt.ImhHiddenText)
            self.checkBox_10.setCheckable(True)
            self.checkBox_10.setObjectName("checkBox_10")
            self.gridLayout_2.addWidget(self.checkBox_10, 0, 3, 1, 1)
            spacerItem13 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
            self.gridLayout_2.addItem(spacerItem13, 0, 0, 1, 1)
            self.checkBox_14 = QtWidgets.QCheckBox(self.centralwidget)
            self.checkBox_14.setStyleSheet("font: 16pt \"宋体\";")
```

```
        self.checkBox_14.setObjectName("checkBox_14")
        self.gridLayout_2.addWidget(self.checkBox_14, 0, 7, 1, 1)
        spacerItem14 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.gridLayout_2.addItem(spacerItem14, 0, 2, 1, 1)
        self.checkBox_13 = QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox_13.setStyleSheet("font: 16pt \"宋体\";")
        self.checkBox_13.setObjectName("checkBox_13")
        self.gridLayout_2.addWidget(self.checkBox_13, 0, 5, 1, 1)
        spacerItem15 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.gridLayout_2.addItem(spacerItem15, 0, 4, 1, 1)
        spacerItem16 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.gridLayout_2.addItem(spacerItem16, 0, 9, 1, 1)
        self.checkBox_12 = QtWidgets.QCheckBox(self.centralwidget)
        self.checkBox_12.setStyleSheet("font: 16pt \"宋体\";")
        self.checkBox_12.setObjectName("checkBox_12")
        self.gridLayout_2.addWidget(self.checkBox_12, 0, 1, 1, 1)
        spacerItem17 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.gridLayout_2.addItem(spacerItem17, 0, 6, 1, 1)
        self.verticalLayout.addLayout(self.gridLayout_2)
        self.label = QLabel(self.centralwidget)
        self.label.setStyleSheet("QLabel{background:white;}"
                                 "QLabel{color:rgb(300,300,300,120);font-
size:10px;font-weight:bold;font-family:宋体;}"
                                 )
        self.verticalLayout.addWidget(self.label)
        self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
```

```
        self.horizontalLayout_3.setObjectName("horizontalLayout_3")
        spacerItem18 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_3.addItem(spacerItem18)
        self.btn_open = QtWidgets.QPushButton(self.centralwidget)
        self.btn_open.setObjectName("btn_open")
        self.btn_open.clicked.connect(self.open_image)

        self.horizontalLayout_3.addWidget(self.btn_open)
        spacerItem19 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_3.addItem(spacerItem19)
        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setStyleSheet("font: 22pt \"黑体\";")
        self.pushButton_2.setObjectName("pushButton_2")
        self.pushButton_2.clicked.connect(self.startImage)

        self.horizontalLayout_3.addWidget(self.pushButton_2)
        spacerItem20 = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding,
QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_3.addItem(spacerItem20)
        self.verticalLayout.addLayout(self.horizontalLayout_3)
        self.label.raise_()
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 1105, 26))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
```

```
MainWindow.setStatusBar(self.statusbar)


self.retranslateUi(MainWindow)
self.radioButton.clicked['bool'].connect(self.checkBox_2.setVisible)
self.radioButton.clicked['bool'].connect(self.checkBox_3.setVisible)
self.radioButton.clicked['bool'].connect(self.checkBox.setVisible)
self.radioButton_2.clicked['bool'].connect(self.checkBox_2.setHidden)
self.radioButton.clicked['bool'].connect(self.checkBox_4.setVisible)
self.radioButton_3.clicked['bool'].connect(self.checkBox_4.setHidden)
self.radioButton_2.clicked['bool'].connect(self.checkBox_3.setHidden)
self.radioButton_3.clicked['bool'].connect(self.checkBox_2.setHidden)
self.radioButton_2.clicked['bool'].connect(self.checkBox.setHidden)
self.radioButton_2.clicked['bool'].connect(self.checkBox_4.setHidden)
self.radioButton_3.clicked['bool'].connect(self.checkBox.setHidden)
self.radioButton_3.clicked['bool'].connect(self.checkBox_3.setHidden)
self.radioButton.clicked['bool'].connect(self.checkBox_5.setHidden)
self.radioButton.clicked['bool'].connect(self.checkBox_7.setHidden)
self.radioButton.clicked['bool'].connect(self.checkBox_8.setHidden)
self.radioButton.clicked['bool'].connect(self.checkBox_9.setHidden)
self.radioButton_2.clicked['bool'].connect(self.checkBox_5.setVisible)
self.radioButton_2.clicked['bool'].connect(self.checkBox_7.setVisible)
self.radioButton_2.clicked['bool'].connect(self.checkBox_8.setVisible)
self.radioButton_2.clicked['bool'].connect(self.checkBox_9.setVisible)
self.radioButton_3.clicked['bool'].connect(self.checkBox_5.setHidden)
self.radioButton_3.clicked['bool'].connect(self.checkBox_7.setHidden)
self.radioButton_3.clicked['bool'].connect(self.checkBox_8.setHidden)
self.radioButton_3.clicked['bool'].connect(self.checkBox_9.setHidden)
self.radioButton.clicked['bool'].connect(self.checkBox_10.setHidden)
self.radioButton.clicked['bool'].connect(self.checkBox_12.setHidden)
self.radioButton.clicked['bool'].connect(self.checkBox_13.setHidden)
```

```
        self.radioButton.clicked['bool'].connect(self.checkBox_14.setHidden)
        self.radioButton_2.clicked['bool'].connect(self.checkBox_10.setHidden)
        self.radioButton_2.clicked['bool'].connect(self.checkBox_12.setHidden)
        self.radioButton_2.clicked['bool'].connect(self.checkBox_13.setHidden)
        self.radioButton_2.clicked['bool'].connect(self.checkBox_14.setHidden)
        self.radioButton_3.clicked['bool'].connect(self.checkBox_10.setVisible)
        self.radioButton_3.clicked['bool'].connect(self.checkBox_12.setVisible)
        self.radioButton_3.clicked['bool'].connect(self.checkBox_13.setVisible)
        self.radioButton_3.clicked['bool'].connect(self.checkBox_14.setVisible)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.radioButton.setText(_translate("MainWindow", "图像分割"))
        self.radioButton_2.setText(_translate("MainWindow", "图像增强"))
        self.radioButton_3.setText(_translate("MainWindow", "边缘检测"))
        self.checkBox.setText(_translate("MainWindow", "OTSU 法"))
        self.checkBox.setVisible(False)
        self.checkBox_4.setText(_translate("MainWindow", "马尔可夫随机场"))
        self.checkBox_4.setVisible(False)
        self.checkBox_2.setText(_translate("MainWindow", "最大熵分割法"))
        self.checkBox_2.setVisible(False)
        self.checkBox_3.setText(_translate("MainWindow", "迭代阈值分割法"))
        self.checkBox_3.setVisible(False)
        self.checkBox_5.setText(_translate("MainWindow", "标准直方图均衡 HE"))
        self.checkBox_5.setVisible(False)
        self.checkBox_9.setText(_translate("MainWindow", "多尺度 Retinex 算法  MSR"))
        self.checkBox_9.setVisible(False)
        self.checkBox_8.setText(_translate("MainWindow", "单尺度 Retinex 算法  SSR"))
```

```
            self.checkBox_8.setVisible(False)
            self.checkBox_7.setText(_translate("MainWindow", "限制对比度自适应直方图均衡
CLAHE"))
            self.checkBox_7.setVisible(False)
            self.checkBox_10.setText(_translate("MainWindow", "Sobel 算子"))
            self.checkBox_10.setVisible(False)
            self.checkBox_14.setText(_translate("MainWindow", "Laplacian 算子"))
            self.checkBox_14.setVisible(False)
            self.checkBox_13.setText(_translate("MainWindow", "Canny 算子"))
            self.checkBox_13.setVisible(False)
            self.checkBox_12.setText(_translate("MainWindow", "Roberts 算子"))
            self.checkBox_12.setVisible(False)
            self.btn_open.setText(_translate("MainWindow", "选择文件"))
            self.pushButton_2.setText(_translate("MainWindow", "开始"))

        # 打开文件
        def open_image(self):
            self.filename, _ = QFileDialog.getOpenFileName(self, "打开图片","C:/",
"*.jpg;;*.png")
            if self.filename:
                self.image = cv2.imdecode(np.fromfile(self.filename, dtype=np.uint8), -1)
                self.im = QtGui.QImage(self.filename)
                if self.im.width()< self.label.width() and self.im.height() < self.label.height():

self.label.setPixmap(QtGui.QPixmap.fromImage(self.im))#.scaled(self.label.width(),
self.label.height()))
                else:

self.label.setPixmap(QtGui.QPixmap.fromImage(self.im).scaled(self.label.width(),
self.label.width() * self.im.height() / self.im.width()))
```

```
            # self.label.setAlignment(QtCore.Qt.AlignVCenter|QtCore.Qt.AlignVCenter)


    #  调整图像大小
    def resize_img(self, pro_img, kk):
        row, col = pro_img.shape[0], pro_img.shape[1]
        if row < col:
            if kk == 1:
                q_img = QtGui.QImage(pro_img.tobytes(), col, row, col * 1, QtGui.QImage.Format_Indexed8)
                pix = QtGui.QPixmap(q_img).scaled(self.label.width(), self.label.width() * row / col)
                # self.label.setPixmap(QtGui.QPixmap.fromImage(q_img))
                self.label.setPixmap(pix)
            else:
                q_img = QtGui.QImage(pro_img.tobytes(), col, row, col * 3, QtGui.QImage.Format_BGR888)
                pix = QtGui.QPixmap(q_img).scaled(self.label.width(), self.label.width() * row / col)
                # self.label.setPixmap(QtGui.QPixmap.fromImage(q_img))
                self.label.setPixmap(pix)
        elif row >= col:
            if kk == 1:
                q_img = QtGui.QImage(pro_img.tobytes(), col, row, col * 1, QtGui.QImage.Format_Indexed8)
                pix = QtGui.QPixmap(q_img).scaled(self.label.height() , self.label.height() * col / row)
                # self.label.setPixmap(QtGui.QPixmap.fromImage(q_img))
                self.label.setPixmap(pix)
            else:
                q_img = QtGui.QImage(pro_img.tobytes(), col, row, col * 3,
```

```
QtGui.QImage.Format_BGR888)
                pix = QtGui.QPixmap(q_img).scaled(self.label.height(), self.label.height() *
col / row)
                # self.label.setPixmap(QtGui.QPixmap.fromImage(q_img))
                self.label.setPixmap(pix)

    # 开始
    def startImage(self):
        if self.radioButton.isChecked():
            pro_img = self.image_seg()
            self.resize_img(pro_img, kk=1)
        elif self.radioButton_2.isChecked():
            pro_img = self.image_enhance()
            self.resize_img(pro_img,kk=2)
        elif self.radioButton_3.isChecked():
            pro_img = self.edge_check()
            self.resize_img(pro_img, kk=3)

    # 图像分割分部
    def image_seg(self):
        gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
        blanck = 255 - np.zeros_like(gray)
        thresh1, thresh2, thresh3, thresh4 = [blanck] * 4
        if self.checkBox.isChecked():
            # 调取 opencv 库自带的 OTSU 方法
            _, thresh1 = cv2.threshold(gray, 0, 255, cv2.THRESH_OTSU)
        if self.checkBox_2.isChecked():
            # import getMaxInformationEntropy
            getMaxInformation = getMaxInformationEntropy()
            T2 = getMaxInformation.getmax(gray)
```

```
            _, thresh2 = cv2.threshold(gray, T2, 255, cv2.THRESH_BINARY)
        if self.checkBox_3.isChecked():
            # import Iterative_threshold
            Iterative = Iterative_threshold()
            T3 = Iterative.getOptimalThreshold(gray)
            _, thresh3 = cv2.threshold(gray, T3, 255, cv2.THRESH_BINARY)
        if self.checkBox_4.isChecked():
            # import Markov_random_field
            markov = Markov_random_field()
            thresh4 = markov.markov_random(gray)

        con_img1 = np.concatenate((thresh1, thresh2), axis=1)
        con_img2 = np.concatenate((thresh3, thresh4), axis=1)
        total_img = np.vstack((con_img1, con_img2))
        return total_img


# 图像增强分部
def image_enhance(self):
        blanck = 255 - np.zeros_like(self.image)
        result1, result2, result3, result4 = [blanck] * 4
        if self.checkBox_5.isChecked():
            he = HE()
            result1 = he.imgHE(self.image)
        if self.checkBox_7.isChecked():
            clahe = CLAHE()
            result2 = clahe.imgCLAHE(self.image)
        if self.checkBox_8.isChecked():
            ssr = SSRetinex()
            result3 = ssr.imgSSR(self.image)
```

```
        if self.checkBox_9.isChecked():
            msr = MSRetinex()
            result4 = msr.imgMSR(self.image)

        con_img1 = np.concatenate((result1, result2), axis=1)
        con_img2 = np.concatenate((result3, result4), axis=1)
        total_img = np.vstack((con_img1, con_img2))
        return total_img

    # 边缘检测分部
    def edge_check(self):
        blanck = 255 - np.zeros_like(self.image)
        roberts, sobel, canny, laplacian = [blanck] * 4
        gray = cv2.cvtColor(self.image, cv2.COLOR_BGR2GRAY)
        img_gray = cv2.GaussianBlur(gray, (3, 3), 0)    # 用高斯平滑处理原图像降噪
        img = cv2.GaussianBlur(self.image, (3, 3), 0)

        # Roberts 算子
        if self.checkBox_12.isChecked():
            kernelx = np.array([[-1, 0], [0, 1]], dtype=int)
            kernely = np.array([[0, -1], [1, 0]], dtype=int)
            x = cv2.filter2D(img, cv2.CV_16S, kernelx)
            y = cv2.filter2D(img, cv2.CV_16S, kernely)
            absX = cv2.convertScaleAbs(x)
            absY = cv2.convertScaleAbs(y)
            roberts = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

        # Sobel 算子
        if self.checkBox_10.isChecked():
            x = cv2.Sobel(img, cv2.CV_16S, 1, 0)
```

```
            y = cv2.Sobel(img, cv2.CV_16S, 0, 1)
            absX = cv2.convertScaleAbs(x)   # 转回 uint8
            absY = cv2.convertScaleAbs(y)
            sobel = cv2.addWeighted(absX, 0.5, absY, 0.5, 0)

        # Canny 算子
        if self.checkBox_13.isChecked():
            canny = self.image.copy()
            canny_edge = cv2.Canny(img, 25, 100)
            canny[canny_edge == 255] = 255

        # Laplacian 算子
        if self.checkBox_14.isChecked():
            dst = cv2.Laplacian(img, cv2.CV_16S, ksize=3)
            laplacian = cv2.convertScaleAbs(dst)

        con_img1 = np.concatenate((roberts, sobel), axis=1)
        con_img2 = np.concatenate((canny, laplacian), axis=1)
        total_img = np.vstack((con_img1, con_img2))
        return total_img

    def img_scaled(self, pro_img):
        height = pro_img.shape[0]
        width = pro_img.shape[1]
        if height < width:
            return self.label.width(), self.label.width() * height / width
        else:
            return self.label.height() * width / height, self.label.height()

# 以下为各个分区所使用的方法
```

```
# 最大熵阈值分割
class getMaxInformationEntropy():
    # 获取各个像素的概率
    def getPixelProbability(self, gray):
        p_arr = np.zeros([256, ])
        gray = gray.ravel()
        for p in gray:
            p_arr[p] += 1
        return p_arr / len(gray)

    # 分离前景和背景
    def separateForegroundAndBackground(self, arr, t):
        f_arr, b_arr = arr[:t], arr[t:]
        return f_arr, b_arr

    # 获取某阈值下的信息熵
    def informationEntropy(self, arr):
        Pn = np.sum(arr)
        if Pn == 0:
            return 0
        iE = 0
        for p_i in arr:
            if p_i == 0:
                continue
            iE += (p_i / Pn) * np.log(p_i / Pn)
        return -iE

    # 获取最优阈值
    def getmax(self, gray):
        p_arr = self.getPixelProbability(gray)
```

```
        IES = np.zeros([256, ])
        for t in range(1, len(IES)):
            f_arr, b_arr = self.separateForegroundAndBackground(p_arr, t)
            IES[t] = self.informationEntropy(f_arr) + self.informationEntropy(b_arr)
        return np.argmax(IES)


# 迭代阈值分割
class Iterative_threshold():
    # 获取各个像素的个数
    def getPixelNumber(self, gray):
        p_arr = np.zeros([256, ])
        gray = gray.ravel()
        for p in gray:
            p_arr[p] += 1
        return p_arr


    # 计算前景背景的灰度均值
    def getFBMeanGrayLevel(self, arr, t):
        FM, BM = 0, 0
        fl, bl = 1, 1
        for i, p in enumerate(arr):
            if i < t:
                FM += i * p
                fl += p
            else:
                BM += i * p
                bl += p
        FM /= fl
        BM /= bl
        return FM, BM
```

```
    # 迭代获取最优阈值
    def getOptimalThreshold(self, gray):
        p_arr = self.getPixelNumber(gray)
        best_threshold = 127
        while True:
            FM, BM = self.getFBMeanGrayLevel(p_arr, best_threshold)
            if best_threshold == (FM + BM) // 2:
                break
            best_threshold = (FM + BM) // 2
        return int(best_threshold)


# 马尔可夫随机场
class Markov_random_field():
    def markov_random(self, gray, cluster_num = 4):
        maxiter = 50
        m = np.size(gray, 0)  # m 为行数
        n = np.size(gray, 1)  # n 为列数
        # G_double = np.array(gray, dtype=np.float64)
        label = np.random.randint(1, cluster_num + 1, [m, n])
        iter = 0


        f_u = np.array([0, 1, 0, 0, 0, 0, 0, 0, 0]).reshape(3, 3)
        f_d = np.array([0, 0, 0, 0, 0, 0, 0, 1, 0]).reshape(3, 3)
        f_l = np.array([0, 0, 0, 1, 0, 0, 0, 0, 0]).reshape(3, 3)
        f_r = np.array([0, 0, 0, 0, 0, 1, 0, 0, 0]).reshape(3, 3)
        f_ul = np.array([1, 0, 0, 0, 0, 0, 0, 0, 0]).reshape(3, 3)
        f_ur = np.array([0, 0, 1, 0, 0, 0, 0, 0, 0]).reshape(3, 3)
        f_dl = np.array([0, 0, 0, 0, 0, 0, 1, 0, 0]).reshape(3, 3)
        f_dr = np.array([0, 0, 0, 0, 0, 0, 0, 0, 1]).reshape(3, 3)
```

```
while iter < maxiter:
    iter = iter + 1
    # print(iter)

    label_u = cv2.filter2D(np.array(label, dtype=np.uint8), -1, f_u)
    label_d = cv2.filter2D(np.array(label, dtype=np.uint8), -1, f_d)
    label_l = cv2.filter2D(np.array(label, dtype=np.uint8), -1, f_l)
    label_r = cv2.filter2D(np.array(label, dtype=np.uint8), -1, f_r)
    label_ul = cv2.filter2D(np.array(label, dtype=np.uint8), -1, f_ul)
    label_ur = cv2.filter2D(np.array(label, dtype=np.uint8), -1, f_ur)
    label_dl = cv2.filter2D(np.array(label, dtype=np.uint8), -1, f_dl)
    label_dr = cv2.filter2D(np.array(label, dtype=np.uint8), -1, f_dr)
    p_c = np.zeros((cluster_num, m, n))

    for i in range(cluster_num):
        label_i = (i + 1) * np.ones((m, n))
        u_T = 1 * np.logical_not(label_i - label_u)
        d_T = 1 * np.logical_not(label_i - label_d)
        l_T = 1 * np.logical_not(label_i - label_l)
        r_T = 1 * np.logical_not(label_i - label_r)
        ul_T = 1 * np.logical_not(label_i - label_ul)
        ur_T = 1 * np.logical_not(label_i - label_ur)
        dl_T = 1 * np.logical_not(label_i - label_dl)
        dr_T = 1 * np.logical_not(label_i - label_dr)
        temp = u_T + d_T + l_T + r_T + ul_T + ur_T + dl_T + dr_T

        p_c[i, :] = (1.0 / 8) * temp

    p_c[p_c == 0] = 0.001
```

```
    mu = np.zeros((1, cluster_num))
    sigma = np.zeros((1, cluster_num))
    for i in range(cluster_num):
        index = np.where(label == (i + 1))
        data_c = gray[index]
        mu[0, i] = np.mean(data_c)
        sigma[0, i] = np.var(data_c)
    # p_sc 为 psw，是 w 成立条件下 s 的条件概率
    p_sc = np.zeros((cluster_num, m, n))
    one_a = np.ones((m, n))

    for j in range(cluster_num):
        MU = mu[0, j] * one_a
        p_sc[j, :] = (1.0 / np.sqrt(2 * np.pi * sigma[0, j])) * np.exp(
            -1. * ((gray - MU) ** 2) / (2 * sigma[0, j]))

    X_out = np.log(p_c) + np.log(p_sc)
    label_c = X_out.reshape(cluster_num, m * n)
    label_c_t = label_c.T
    label_m = np.argmax(label_c_t, axis=1)
    label_m = label_m + np.ones(label_m.shape)
    label = label_m.reshape(m, n)

label = label - np.ones(label.shape)  # 为了出现 0
label = label.astype(np.uint8)
lable_w = np.zeros_like(label)
for i in range(cluster_num - 1):
    lable_w[label == i] = np.round(255 / (cluster_num - 1) * i)
lable_w[label == cluster_num - 1] = 255
```

```
        return    lable_w


# 直方图均衡法
class HE:
    def imgHE(self, image):   # 输入为彩色三通道图像
        ycrcb = cv2.cvtColor(image, cv2.COLOR_BGR2YCR_CB)   # 将 RGB 图像转入 YCbCr 空间中
        # channel = cv2.split(ycrcb)   # 将 YCbCr 通道分离
        Y = ycrcb[:, :, 0]
        Cb = ycrcb[:, :, 1].astype(np.uint8)
        Cr = ycrcb[:, :, 2].astype(np.uint8)
        m = np.size(Y, 0)   # m 为行数
        n = np.size(Y, 1)   # n 为列数
        # 将灰度图转为一列一维数组
        Y2 = Y.reshape(m * n, 1)
        # c 为各灰度出现次数，总和 C 为全部像素点数
        c = np.zeros((256, 1))
        for i in range(0, 256):
            c[i] = np.sum(Y2 == i)
        # c = c[1:]
        C = np.sum(c)
        # p 为各灰度出现概率，总和 P 为 1
        p = c / C
        ck = np.cumsum(p)
        fk = 255 * ck
        fk = np.round(fk)
        Y = Y.astype(np.uint8)
        Y3 = np.zeros((m * n, 1))
        for j in range(256):
            Y3[Y2 == j] = fk[j]
```

```
            YN = Y3.reshape(m, n)
            YN = YN.astype(np.uint8)
            ycrcb_out = cv2.merge([YN, Cb, Cr])
            # YM = cv2.cvtColor(YN,cv2.COLOR_GRAY2RGB)  # 三通道都是一样的值
            # YM = cv2.applyColorMap(YN, cv2.COLORMAP_HSV)  # 伪彩色
            G = cv2.cvtColor(ycrcb_out, cv2.COLOR_YCR_CB2BGR)
            return G

# 限制对比度的自适应直方图均衡
class CLAHE:
    def imgCLAHE(self, image):
            image = cv2.cvtColor(image, cv2.COLOR_BGRA2BGR)
            b, g, r = cv2.split(image)
            clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
            b = clahe.apply(b)
            g = clahe.apply(g)
            r = clahe.apply(r)
            image = cv2.merge([b, g, r])
            return image

# 单尺度 Retinex
class SSRetinex:
    # 图像预处理（除去 0）
    def replaceZeros(self, img):
            min_img = np.min(img[np.nonzero(img)])
            img[img == 0] = min_img
            return img

    '''
    def SSR(self, img, size):  # 单尺度 Retinex(Single Scale Retinex)  cv2 库函数版
```

```
            L_blur = cv2.GaussianBlur(img, (size, size), 0)
            h, w = img.shape[:2]
            dst_img = np.zeros((h, w), dtype=np.float32)
            dst_Lblur = np.zeros((h, w), dtype=np.float32)
            dst_R = np.zeros((h, w), dtype=np.float32)

            img = replaceZeros(img).astype(np.float32)
            L_blur = replaceZeros(L_blur).astype(np.float32)
            dst_img = cv2.log(img/255.0)
            dst_Lblur = cv2.log(L_blur/255.0)
            dst_IxL = cv2.multiply(dst_img, dst_Lblur)
            log_R = cv2.subtract(dst_img, dst_IxL)
            dst_R = cv2.normalize(log_R, None, 0, 255, cv2.NORM_MINMAX)
            log_uint8 = cv2.convertScaleAbs(dst_R)
            return log_uint8
        '''

        def SSR(self, img, GaussLB_size):  # 单尺度 Retinex(Single Scale Retinex)  numpy 库函数版
            L_blur = cv2.GaussianBlur(img, (GaussLB_size, GaussLB_size), 0)
            img = self.replaceZeros(img).astype(np.float32)
            L_blur = self.replaceZeros(L_blur).astype(np.float32)
            log_R = (np.log(img / 255.0) - np.log(img / 255.0) * np.log(L_blur / 255.0)).astype(np.float32)
            R = (log_R - np.min(log_R)) / (np.max(log_R) - np.min(log_R)) * 255.0
            R = R.astype(np.uint8)
            return R

        def imgSSR(self, image):
            image = cv2.cvtColor(image, cv2.COLOR_BGRA2BGR)
```

```
        b, g, r = cv2.split(image)
        b_out = self.SSR(b, 3)
        g_out = self.SSR(g, 3)
        r_out = self.SSR(r, 3)
        result = cv2.merge([b_out, g_out, r_out])
        return result

# 多尺度 Retinex
class MSRetinex:
# 图像预处理（除去 0）
    def replaceZeros(self, img):
        min_img = np.min(img[np.nonzero(img)])
        img[img == 0] = min_img
        return img

    '''
    # 多尺度 Retinex(Multi Scale Retinex)  cv2 库函数版
    def MSR(self, img, GaussLB_scales):
        num_scales = len(GaussLB_scales)
        weight = 1/num_scales
        h, w = img.shape[:2]
        img = replaceZeros(img).astype(np.float32)
        log_R = np.zeros((h,w),dtype=np.float32)

        for i in range(num_scales):
            L_blur = cv2.GaussianBlur(img, (GaussLB_scales[i], GaussLB_scales[i]), 0)
            L_blur = replaceZeros(L_blur).astype(np.float32)
            dst_img = cv2.log(img/255.0)
            dst_Lblur = cv2.log(L_blur/255.0)
            dst_IxL = cv2.multiply(dst_img, dst_Lblur)
```

```
            log_R += weight*cv2.subtract(dst_img, dst_IxL)
        dst_R = cv2.normalize(log_R, None, 0, 255, cv2.NORM_MINMAX)
        log_uint8 = cv2.convertScaleAbs(dst_R)
        return log_uint8
    '''

    # 多尺度 Retinex(Multi Scale Retinex)  numpy 库函数版
    # img 为一层二维数组，L_blur 为高斯滤波尺度数层个二维数组
    def MSR(self, img, GaussLB_scales):
        num_scales = len(GaussLB_scales)
        weight = 1 / num_scales
        img = self.replaceZeros(img).astype(np.float32)
        height, width = img.shape
        L_blur = np.zeros((height, width), dtype=np.float32)
        log_R = np.zeros((height, width), dtype=np.float32)
        for i in range(num_scales):
            L_blur = cv2.GaussianBlur(img, (GaussLB_scales[i], GaussLB_scales[i]), 0)
            L_blur = self.replaceZeros(L_blur).astype(np.float32)
            log_R += weight * (np.log(img / 255.0) - np.log(img / 255.0) * np.log(L_blur /
255.0))
        R = (log_R - np.min(log_R)) / (np.max(log_R) - np.min(log_R)) * 255.0
        R = R.astype(np.uint8)
        return R

    def imgMSR(self, image):
        image = cv2.cvtColor(image, cv2.COLOR_BGRA2BGR)
        b, g, r = cv2.split(image)
        GaussLB_scales = [3, 5, 15]
        b_out = self.MSR(b, GaussLB_scales)
        g_out = self.MSR(g, GaussLB_scales)
```

```
        r_out = self.MSR(r, GaussLB_scales)
        result = cv2.merge([b_out, g_out, r_out])
        return result
```

main 函数 main.py

```
import sys
from PyQt5.QtWidgets import *
from PyQt5 import QtGui, QtWidgets, QtCore
from image_processing import Ui_MainWindow

if __name__ == '__main__':
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```