# 图像识别系统

## 目录：

课堂研究内容和意义P2

程序介绍P4

关于图片二维码识别P4

关于视频二维码识别P9

关于摄像头识别二维码P14

关于图片识别车牌P19

关于视频识别车牌P24

关于摄像头识别车牌P28

关于图片检测人脸P33

关于视频检测人脸P38

关于摄像头检测人脸P43

关于图片拼接P48

心得体会P57

附录P59

## 设计题目：

图像识别系统设计与实现

## 设计要求：

## 课题研究内容及意义：

本课题基于pycharm与designer进行开发，按照要求设计一个用户界面，其中包含GUI4.py文件作为界面代码实现控件的设计；mainForm4.py作为逻辑代码实现对应控件功能。此程序具备以下功能：

（1）二维码识别：

打开图片，检测并解码QR码，输出结果；

打开摄像头或本地视频文件，检测并解码QR码，输出结果。

支持多个QR码同时检测。

（2）车牌识别：

打开图片，检测并识别车牌字符，输出结果；

打开摄像头或本地视频文件，检测并识别车牌字符，输出结果。

（3）人脸检测：

打开图片，检测人脸，可视化结果；

打开摄像头或本地视频文件，检测人脸，输出结果；

支持多人脸检测。眼睛检测、微笑等表情检测。

（4）图像拼接：

打开2副图像，基于柱面投影拼接成一副大图。

## 程序介绍：

## 二维码识别

图形用户界面, 应用程序

描述已自动生成

## 关于图片二维码识别：

此处界面的输出为label，图片识别二维码按钮为pushButton。实现功能的函数是QRimage。在mainForm4.py中使用self.pushButton.clicked.connect(self.QRimage)将按钮连接槽函数。这是Qrimage的代码：

def QRimage(self):  
 # 弹出文件选择对话框，选择图片  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择图片", "", "Image Files (\*.png \*.jpg \*.jpeg \*.bmp)")  
 if filename:  
 try:  
 # 使用PIL打开图像文件  
 pil\_image = Image.open(filename)  
  
 # 将PIL图像转换为OpenCV格式  
 cv\_image = np.array(pil\_image)  
 cv\_image = cv2.cvtColor(cv\_image, cv2.COLOR\_RGB2BGR) # PIL图像默认是RGB，OpenCV默认是BGR  
  
 # 使用pyzbar库进行二维码识别  
 decoded\_objects = pyzbar.decode(cv\_image)  
  
 # 初始化解码数据列表  
 decoded\_data = []  
  
 # 遍历解码对象，提取数据  
 for obj in decoded\_objects:  
 if obj.type == "QRCODE":  
 # 解码数据并添加到列表中  
 decoded\_data.append(obj.data.decode('utf-8'))  
  
 # 准备显示结果的文本  
 if decoded\_data:  
 result\_text = "<html><head/><body><p align=\"center\">识别结果:</p></body></html>" + "".join(  
 f"<p align=\"center\">{data}</p>" for data in decoded\_data  
 )  
 else:  
 result\_text = "<html><head/><body><p align=\"center\">未识别到二维码</p></body></html>"  
  
 # 更新label显示识别结果  
 self.label.setText(result\_text)  
  
 # 使用cv2.imshow显示图片  
 cv2.imshow('QR Code Image', cv\_image)  
 cv2.waitKey(0) # 等待用户按键  
  
 except IOError as e:  
 # 如果文件打开失败，显示错误消息  
 QMessageBox.warning(self, "错误", f"无法打开图片文件: {e}")  
 except Exception as e:  
 # 显示其他可能的错误  
 QMessageBox.warning(self, "错误", f"二维码识别出错: {e}")  
 finally:  
 # 关闭所有OpenCV窗口  
 cv2.destroyAllWindows()

Pillow库来处理图像，以及pyzbar库来识别二维码。以下是代码中使用到的算法原理的介绍：

1. **文件选择对话框**：通过QFileDialog.getOpenFileName函数弹出一个对话框，让用户选择一个图像文件。这个对话框会过滤出支持的图像格式（如PNG、JPG、JPEG、BMP）。
2. **图像文件读取**：如果用户选择了一个文件，Image.open函数会尝试打开这个文件。Pillow是一个图像处理库，它允许用户读取、操作和保存图像。
3. **二维码识别**：pyzbar.decode函数用于识别图像中的二维码。pyzbar是一个二维码解码库，它使用了图像处理技术来识别和解码图像中的二维码。
4. **解码对象遍历**：遍历pyzbar.decode返回的解码对象列表。对于每个对象，检查其类型是否为"QRCODE"，如果是，则进行下一步处理。
5. **数据解码与提取**：对于每个识别为二维码的对象，使用.decode('utf-8')方法将编码的数据转换为可读的UTF-8格式字符串，并将其添加到decoded\_data列表中。
6. **结果文本准备**：根据解码数据列表是否为空，准备相应的HTML格式文本。如果列表不为空，将每个解码的数据项添加到HTML段落标签<p>中；如果列表为空，则显示未识别到二维码的消息。
7. **更新显示**：通过self.label.setText方法，将准备好的HTML文本设置到GUI中的标签（label）上，以展示识别结果。并且将识别的二维码也创建新窗口进行显示。
8. **异常处理**：代码中包含了两个异常处理部分。第一个是IOError，用于处理文件打开失败的情况；第二个是通用的Exception，用于捕获和处理在二维码识别过程中可能发生的其他错误。
9. **错误消息提示**：使用QMessageBox.warning方法显示错误消息，告知用户为什么二维码识别失败。

在二维码识别过程中，pyzbar库可能使用了多种算法，包括但不限于：

* **特征检测**：识别图像中的二维码模式，如二维码的三个大定位图案。
* **图像二值化**：将图像转换为黑白两色，以简化识别过程。
* **模式匹配**：匹配二维码的特定模式，如定位图案和版本信息。
* **错误校正**：二维码设计中包含了错误校正码，pyzbar会利用这些码来修正可能的识别错误。

运行结果：  
电脑软件的截图

描述已自动生成 QR 代码

描述已自动生成

## 关于视频二维码识别：

视频二维码识别按钮为pushButton\_2。实现功能的函数是QRvideo。在mainForm4.py中使用self.pushButton\_2.clicked.connect(self.QRvideo)将按钮连接槽函数。这是QRvideo的代码：

def QRvideo(self):  
 # 弹出文件选择对话框，选择视频  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择视频文件", "", "Video Files (\*.mp4 \*.avi \*.mov \*.mkv)")  
 if filename:  
 # 使用 OpenCV 读取视频  
 cap = cv2.VideoCapture(filename)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开视频文件，请选择有效的视频文件。")  
 return  
  
 previous\_data = None # 用于存储上一次的识别结果  
 decoded\_data = [] # 用于存储解码结果  
  
 try:  
 # 创建一个窗口来显示视频  
 cv2.namedWindow('Video', cv2.WINDOW\_NORMAL)  
  
 while True:  
 # 读取视频的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # 将BGR帧转换为PIL Image  
 image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
 img = Image.fromarray(image)  
  
 # 使用pyzbar库进行二维码识别  
 decoded\_objects = pyzbar.decode(img)  
  
 # 检查是否识别到二维码  
 if decoded\_objects:  
 for obj in decoded\_objects:  
 if obj.type == "QRCODE":  
 # 获取解码的数据  
 data = obj.data.decode('utf-8')  
 # 如果当前识别结果与上一个不同，则更新显示结果  
 if data != previous\_data:  
 previous\_data = data  
 decoded\_data.append(data)  
  
 # 显示当前帧  
 cv2.imshow('Video', frame)  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
 # 释放视频对象  
 cap.release()  
 # 关闭所有OpenCV窗口  
 cv2.destroyAllWindows()  
  
 # 将识别结果按照换行符输出  
 if decoded\_data:  
 result\_text = "<html><head/><body><p align=\"center\">识别结果:</p></body></html>" + "\n".join(  
 f"<p align=\"center\">{data}</p>" for data in set(decoded\_data) # 使用 set 来去除重复项  
 )  
 self.label.setText(result\_text)  
 else:  
 self.label.setText("<html><head/><body><p align=\"center\">视频中未识别到二维码</p></body></html>")  
  
 except Exception as e:  
 print("Error during QR code decoding from video:", e)  
 self.label.setText("<html><head/><body><p align=\"center\">二维码识别出错</p></body></html>")

使用了PyQt框架来处理图形界面的交互，OpenCV库来处理视频流，Pillow库来处理图像，以及pyzbar库来识别二维码。以下是代码中使用到的算法原理的介绍：

1. **文件选择对话框**：通过QFileDialog.getOpenFileName函数弹出一个对话框，让用户选择一个视频文件。这个对话框会过滤出支持的视频格式（如MP4、AVI、MOV、MKV）。
2. **视频读取**：使用OpenCV的VideoCapture类来读取视频文件。OpenCV是一个强大的计算机视觉库，它允许用户读取、处理和保存视频数据。
3. **视频帧读取**：通过循环调用cap.read()方法来逐帧读取视频内容。ret是一个布尔值，表示是否成功读取了帧；frame是读取到的视频帧图像。并且利用cv2.imshow将每一帧连续的显示出来，如用户知道正在识别二维码。
4. **颜色空间转换**：由于OpenCV默认使用BGR颜色空间，而Pillow和pyzbar使用RGB颜色空间，因此需要使用cv2.cvtColor函数将BGR图像转换为RGB图像。
5. **图像转换**：将OpenCV的图像数据通过Pillow的Image.fromarray方法转换为Pillow图像对象，以便后续处理。
6. **二维码识别**：使用pyzbar.decode函数来识别图像中的二维码。pyzbar会分析图像数据，定位二维码，并尝试解码。
7. **结果更新**：如果识别到新的二维码数据，并且该数据与上一次识别的数据不同，则更新previous\_data变量，并添加到decoded\_data列表中。使用set来去除重复项，确保结果中不包含重复的解码数据。
8. **视频释放**：在视频处理结束后，使用cap.release()方法释放视频对象，避免资源泄露。
9. **结果显示**：将识别到的二维码数据以HTML格式显示在GUI的标签上。如果没有识别到二维码，则显示相应的消息。
10. **异常处理**：使用try...except结构来捕获和处理在视频处理和二维码识别过程中可能发生的异常。

在整个过程中，OpenCV和pyzbar库的算法原理包括：

* **视频流处理**：OpenCV能够逐帧读取视频文件，提供了丰富的视频处理功能。
* **颜色空间转换**：由于不同的图像处理库使用不同的颜色空间，需要进行颜色空间的转换以确保兼容性。
* **图像转换**：将OpenCV的图像数据转换为Pillow可以处理的格式。
* **二维码识别**：pyzbar通过图像处理技术识别二维码，并尝试解码，这涉及到特征检测、模式匹配和错误校正等算法。

运行结果：  
图形用户界面, 应用程序

描述已自动生成 图形用户界面, 应用程序

描述已自动生成

按照视频中先后出现的两个二维码顺序输出。

## 关于摄像头识别二维码：

摄像头识别二维码的按钮是pushButton\_3，对应的槽函数是QRcamera。连接两者self.pushButton\_3.clicked.connect(self.QRcamera)，这是QRcamera的代码：

def QRcamera(self):  
 # 打开摄像头 按下q可以退出摄像头  
 cap = cv2.VideoCapture(0)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开摄像头。")  
 return  
  
 try:  
 while True:  
 # 读取摄像头的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # 将 BGR 帧转换为 PIL Image  
 image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
 img = Image.fromarray(image)  
  
 # 使用pyzbar库进行二维码识别  
 decoded\_objects = pyzbar.decode(img)  
  
 # 检查是否识别到二维码  
 if decoded\_objects:  
 decoded\_data = [obj.data.decode('utf-8') for obj in decoded\_objects if obj.type == 'QRCODE']  
 if decoded\_data:  
 # 如果识别到二维码，将结果按照换行符输出  
 result\_text = "<html><head/><body><p align=\"center\">识别结果:</p></body></html>" + \  
 "\n".join(f"<p align=\"center\">{data}</p>" for data in decoded\_data)  
 self.label.setText(result\_text)  
 else:  
 self.label.setText("<html><head/><body><p align=\"center\">未识别到二维码</p></body></html>")  
 else:  
 self.label.setText("<html><head/><body><p align=\"center\">未识别到二维码</p></body></html>")  
  
 # 显示处理后的帧  
 cv2.imshow('QR Camera', frame)  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
 except Exception as e:  
 print("Error during QR code decoding from camera:", e)  
 self.label.setText("<html><head/><body><p align=\"center\">二维码识别出错</p></body></html>")  
  
 finally:  
 # 释放摄像头资源  
 cap.release()  
 # 关闭所有 OpenCV 窗口  
 cv2.destroyAllWindows()

Pillow库来处理图像，以及pyzbar库来识别二维码。以下是代码中使用到的算法原理的介绍：

1. **摄像头初始化**：使用OpenCV的VideoCapture类来初始化摄像头。参数0通常表示系统的默认摄像头。
2. **视频流读取**：通过循环调用cap.read()方法来逐帧读取摄像头的视频流。ret是一个布尔值，表示是否成功读取了帧；frame是读取到的视频帧图像。
3. **颜色空间转换**：由于OpenCV默认使用BGR颜色空间，而Pillow和pyzbar使用RGB颜色空间，因此需要使用cv2.cvtColor函数将BGR图像转换为RGB图像。
4. **图像转换**：将OpenCV的图像数据通过Pillow的Image.fromarray方法转换为Pillow图像对象，以便后续处理。
5. **二维码识别**：使用pyzbar.decode函数来识别图像中的二维码。pyzbar会分析图像数据，定位二维码，并尝试解码。
6. **结果显示**：如果识别到二维码，将解码的数据以HTML格式显示在GUI的标签上。如果没有识别到二维码，则显示相应的消息。
7. **视频显示**：使用cv2.imshow函数显示处理后的帧，以便用户可以看到摄像头的实时视频。
8. **退出条件**：用户可以按'q'键退出视频流读取循环。
9. **异常处理**：使用try...except结构来捕获和处理在视频处理和二维码识别过程中可能发生的异常。
10. **资源释放**：在结束视频流处理后，使用cap.release()方法释放摄像头资源，并使用cv2.destroyAllWindows()关闭所有OpenCV创建的窗口。

在二维码识别过程中，pyzbar库可能使用了多种算法，包括但不限于：

* **特征检测**：识别图像中的二维码模式，如二维码的三个大定位图案。
* **图像二值化**：将图像转换为黑白两色，以简化识别过程。
* **模式匹配**：匹配二维码的特定模式，如定位图案和版本信息。
* **错误校正**：二维码设计中包含了错误校正码，pyzbar会利用这些码来修正可能的识别错误。

整个过程中，OpenCV和pyzbar库的算法原理包括：

* **视频流处理**：OpenCV能够逐帧读取摄像头的视频流，提供了丰富的视频处理功能。
* **颜色空间转换**：由于不同的图像处理库使用不同的颜色空间，需要进行颜色空间的转换以确保兼容性。
* **图像转换**：将OpenCV的图像数据转换为Pillow可以处理的格式。
* **二维码识别**：pyzbar通过图像处理技术识别二维码，并尝试解码，这涉及到特征检测、模式匹配和错误校正等算法。

运行结果：  
QR 代码

描述已自动生成

## 车牌识别

图形用户界面, 应用程序

描述已自动生成

## 关于图片识别车牌： 图片识别车牌按钮对应pushButton\_4，输出对应label\_2，对应IDimage这个槽函数。这是IDimage的代码：

def carplate\_detect(image):#检测图像中的车牌区域。  
 carplate\_rects = carplate\_haar.detectMultiScale(image, scaleFactor=1.1, minNeighbors=3)  
 return carplate\_rects  
  
def carplate\_extract(image, rects):#从原始图像中裁剪出车牌图像。  
 plates = []  
 for x, y, w, h in rects:  
 carplate\_img = image[y + 15:y + h - 10, x + 15:x + w - 20]  
 plates.append(carplate\_img)  
 return plates  
  
def ocr\_plate(image):#光学字符识别  
 results = ocr.ocr(image, cls=True)  
 if results is None or len(results) == 0:  
 return "No plate detected"  
 else:  
 try:  
 # 返回第一个识别结果  
 return results[0][0][1][0]  
 except IndexError:  
 # 如果索引操作失败，返回未检测到车牌  
 return "No plate detected"  
  
def IDimage(self):  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择图片", "", "Image Files (\*.png \*.jpg \*.jpeg \*.bmp)")  
 detected\_plates = []  
 if filename:  
 # 读取图片  
 img = cv2.imread(filename)  
 cv2.imshow('Original Image', img)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 if img is None:  
 QMessageBox.warning(self, "警告", "无法加载图片，请选择有效的图片文件。")  
 return  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 rects = MainWindow.carplate\_detect(gray)  
 for plate\_img in MainWindow.carplate\_extract(gray, rects):  
 plate\_str = MainWindow.ocr\_plate(plate\_img)  
 if plate\_str != "No plate detected": # 过滤掉未检测到车牌的情况  
 detected\_plates.append(plate\_str)  
  
 # 将列表转换为字符串，用逗号和空格分隔  
 plates\_str = ', '.join(detected\_plates)  
  
 # 设置标签的文本  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)

1. **车牌检测 (carplate\_detect)**：
   * 使用carplate\_haar，这是一个基于Haar特征的级联分类器，用于检测图像中的车牌位置。Haar特征级联是一种广泛应用于人脸检测、行人检测等场景的机器学习模型，它通过组合多个简单的特征（如边缘、线条和纹理）来识别图像中的特定对象。
   * detectMultiScale函数用于在不同的尺度上检测对象，scaleFactor参数控制图像尺寸缩小的比例，minNeighbors参数设置最小邻居数，用于减少误检测。
2. **车牌提取 (carplate\_extract)**：
   * 根据检测到的车牌位置（由x, y, w, h表示的矩形框），从原始图像中裁剪出车牌图像。这里通过在矩形框的基础上添加一些偏移量（例如，上下各加15，左右各减20），以期望获得更准确的车牌图像。
3. **OCR车牌识别 (ocr\_plate)**：
   * ocr.ocr函数用于对车牌图像进行光学字符识别，返回识别结果。cls=True可能表示启用了某种分类或模式识别功能。
   * 如果没有检测到结果或结果列表为空，则返回"No plate detected"。
   * 否则，尝试返回第一个识别结果，这通常表示最有可能的车牌号码。
4. **主函数 (IDimage)**：
   * 使用QFileDialog.getOpenFileName弹出文件选择对话框，让用户选择图片文件。
   * 使用cv2.imread读取图片文件，如果读取失败则显示警告并返回。然后使用cv2.imshow显示图片，方便用户观察原图。
   * 将图片转换为灰度图像，因为车牌检测通常在灰度空间中进行，以减少计算量并保持足够的特征信息。
   * 调用carplate\_detect来检测图像中的所有可能的车牌位置。
   * 对于每个检测到的车牌位置，使用carplate\_extract来提取车牌图像，然后调用ocr\_plate进行OCR识别。
   * 过滤掉未检测到车牌的情况，并将所有检测到的车牌号码存储在列表detected\_plates中。
   * 将车牌号码列表转换为字符串，用逗号和空格分隔，并更新GUI中的标签（label\_2）以显示结果。

运行结果：

注意：需要将车牌照片显示窗口关闭后，才会识别车牌。  
图形用户界面, 应用程序

描述已自动生成

图形用户界面, 网站

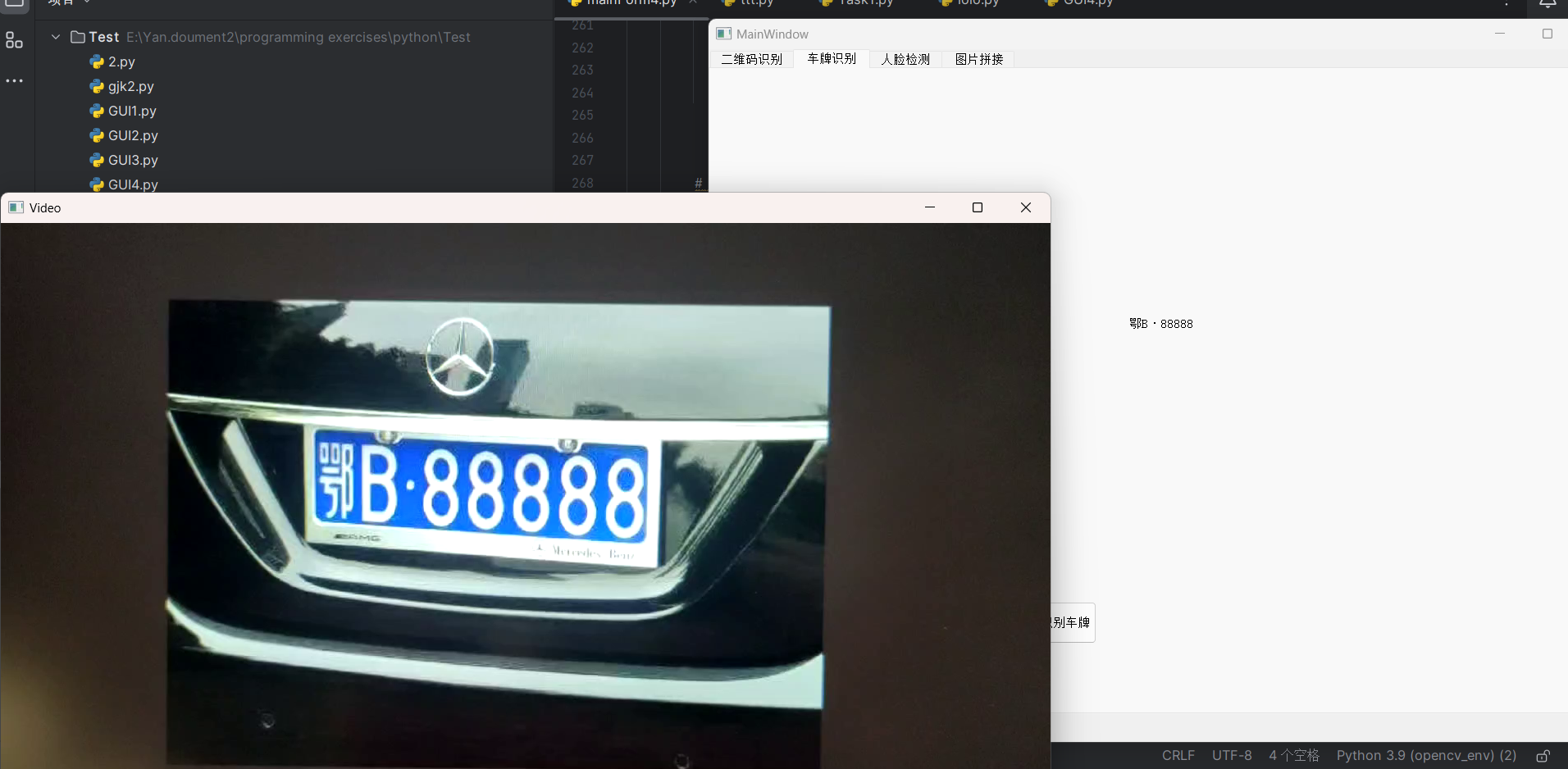
描述已自动生成图形用户界面, 文本, 应用程序

描述已自动生成关于视频识别车牌：  
视频识别车牌按钮对应pushButton\_5，对应槽函数是IDvideo，这是IDvideo的代码：

def IDvideo(self):  
 # 弹出选择文件对话框，选择视频文件  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择视频", "", "Video Files (\*.mp4 \*.avi \*.mov \*.mkv)")  
 if not filename:  
 print("No video selected.")  
 return  
  
 # 初始化视频读取器  
 cap = cv2.VideoCapture(filename)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开视频文件，请选择有效的视频文件。")  
 return  
  
 # 清空标签的文本  
 self.label\_2.clear()  
  
 tick = cv2.getTickCount()  
 while True:  
 # 读取视频的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break # 如果视频结束或无法读取帧，则退出循环  
  
 # 转换为灰度图  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 车牌检测  
 rects = MainWindow.carplate\_detect(gray) # 确保这里调用的是实例方法  
  
 # 存储检测到的车牌号，同时去除重复  
 unique\_plates = set()  
  
 for plate\_img in MainWindow.carplate\_extract(gray, rects): # 同上  
 plate\_str = MainWindow.ocr\_plate(plate\_img) # 同上  
 if plate\_str != "No plate detected":  
 unique\_plates.add(plate\_str)  
  
 # 将集合转换为字符串，用逗号和空格分隔  
 plates\_str = ', '.join(unique\_plates)  
  
 # 更新UI，显示车牌号码  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)  
  
 # 处理所有等待的事件，确保UI响应性  
 QApplication.processEvents()  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
 delay = max(0, 33 - (cv2.getTickCount() - tick), )  
 cv2.waitKey(delay) # 等待一定的毫秒数  
  
 # 更新下一次循环的开始时间  
 tick = cv2.getTickCount()  
 cv2.imshow('Video', frame)  
  
  
  
 # 释放视频读取器并关闭所有OpenCV窗口  
 cap.release()  
 cv2.destroyAllWindows()  
  
 # 最后更新一次标签，以显示最后一次检测到的车牌号码  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)

1. **文件选择对话框**：通过QFileDialog.getOpenFileName函数弹出一个对话框，让用户选择一个视频文件。支持的视频格式包括MP4、AVI、MOV、MKV。
2. **视频读取器初始化**：使用OpenCV的VideoCapture类来读取视频文件。如果无法打开视频文件，会显示警告并退出函数。
3. **视频处理循环与显示**：通过循环调用cap.read()方法和cv2.imshow方法来逐帧读取与显示视频内容。如果读取失败或视频结束，则退出循环。
4. **颜色空间转换**：将读取到的BGR格式的视频帧转换为灰度图像，因为车牌检测通常在灰度空间中进行，以减少计算量并保持足够的特征信息。
5. **车牌检测**：调用carplate\_detect函数，使用Haar特征级联分类器检测图像中的车牌位置。
6. **车牌提取**：对于每个检测到的车牌位置，调用carplate\_extract函数从视频中裁剪出车牌图像。
7. **OCR车牌识别**：对裁剪出的每个车牌图像调用ocr\_plate函数进行光学字符识别，返回识别的车牌号码。
8. **去重复和结果更新**：将识别到的车牌号码存储在unique\_plates集合中，自动去除重复的车牌号码。然后将集合转换为字符串，并更新GUI中的标签label\_2以显示车牌号码。
9. **UI事件处理**：通过调用QApplication.processEvents()确保在视频处理过程中GUI的响应性。
10. **退出条件**：用户可以按'q'键退出视频处理循环。
11. **帧率控制**：通过计算和等待一定的毫秒数来控制视频处理的帧率，以避免过快处理。
12. **资源释放**：在结束视频处理后，使用cap.release()方法释放视频读取器资源，并使用cv2.destroyAllWindows()关闭所有OpenCV窗口。
13. **最终结果更新**：在循环结束后，再次更新标签label\_2，以确保显示最后一次检测到的车牌号码。

运行结果：  
图形用户界面, 应用程序

描述已自动生成 

关于摄像头识别车牌：  
摄像头车牌识别按钮对应pushButton\_6，对应槽函数IDcamera，这是IDcamera的代码：

def IDcamera(self):  
 # 初始化摄像头  
 cap = cv2.VideoCapture(0) # 参数0通常表示系统的默认摄像头  
  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开摄像头，请检查摄像头连接。")  
 return  
  
 # 清空标签的文本  
 self.label\_2.clear()  
  
 try:  
 while True:  
 # 读取摄像头的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break # 如果摄像头结束或无法读取帧，则退出循环  
  
 # 转换为灰度图  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 车牌检测  
 rects = MainWindow.carplate\_detect(gray)  
  
 # 存储检测到的车牌号，同时去除重复  
 unique\_plates = set()  
  
 for plate\_img in MainWindow.carplate\_extract(gray, rects):  
 plate\_str =MainWindow.ocr\_plate(plate\_img)  
 if plate\_str != "No plate detected":  
 unique\_plates.add(plate\_str)  
  
 # 将集合转换为字符串，用逗号和空格分隔  
 plates\_str = ', '.join(unique\_plates)  
  
 # 更新UI，显示车牌号码  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)  
  
 # 处理所有等待的事件，确保UI响应性  
 QApplication.processEvents()  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
 finally:  
 # 释放摄像头并关闭所有OpenCV窗口  
 cap.release()  
 cv2.destroyAllWindows()  
  
 # 最后更新一次标签，以显示最后一次检测到的车牌号码  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)

1. **摄像头初始化**：使用OpenCV的VideoCapture类来初始化并打开系统的默认摄像头。如果摄像头无法打开，会显示警告并退出函数。
2. **视频处理循环**：通过循环调用cap.read()方法来逐帧读取摄像头的视频流。如果读取失败，则退出循环。
3. **颜色空间转换**：将读取到的BGR格式的视频帧转换为灰度图像，因为车牌检测通常在灰度空间中进行，这有助于减少计算量并保持足够的特征信息。
4. **车牌检测**：调用carplate\_detect函数，使用Haar特征级联分类器检测图像中的车牌位置。Haar特征级联是一种机器学习模型，通过学习车牌的特征来识别图像中的车牌。
5. **车牌提取**：对于每个检测到的车牌位置，调用carplate\_extract函数从视频中裁剪出车牌图像。
6. **OCR车牌识别**：对裁剪出的每个车牌图像调用ocr\_plate函数进行光学字符识别，返回识别的车牌号码。
7. **去重复和结果更新**：将识别到的车牌号码存储在unique\_plates集合中，自动去除重复的车牌号码。然后将集合转换为字符串，并更新GUI中的标签label\_2以显示车牌号码。
8. **UI事件处理**：通过调用QApplication.processEvents()确保在视频处理过程中GUI的响应性。
9. **退出条件**：用户可以按'q'键退出视频处理循环。
10. **资源释放**：在结束视频处理后，使用cap.release()方法释放摄像头资源，并使用cv2.destroyAllWindows()关闭所有OpenCV窗口。
11. **最终结果更新**：在循环结束后，再次更新标签label\_2，以确保显示最后一次检测到的车牌号码。

运行结果：  
图形用户界面, 文本, 应用程序

描述已自动生成

## 人脸检测

图形用户界面, 应用程序, Teams

描述已自动生成

关于图片检测人脸：  
图片检测人脸按钮是pushButton\_7，输出对应label\_3，对应槽函数是faceimage，这是faceimage的代码：

def faceimage(self):  
 # 弹出文件选择对话框，选择图片  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择图片", "", "Image Files (\*.png \*.jpg \*.jpeg \*.bmp)")  
 if filename:  
 # 读取图片  
 img = cv2.imread(filename)  
 if img is None:  
 QMessageBox.warning(self, "警告", "无法加载图片，请选择有效的图片文件。")  
 return  
  
 # 将图片转换为灰度图，用于检测  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
 # 加载 Haar 特征级联文件  
 face\_cascade = cv2.CascadeClassifier(  
 "E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_frontalface\_alt.xml")  
 eye\_cascade = cv2.CascadeClassifier(  
 "E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_eye\_tree\_eyeglasses.xml")  
 smile\_cascade = cv2.CascadeClassifier(  
 "E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_smile.xml")  
  
 # 人脸检测  
 faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))  
 if len(faces) > 1: # 如果检测到多个人脸，选择置信度最高的一个  
 faces = sorted(faces, key=lambda f: f[2] \* f[3], reverse=True) # 按面积排序  
 faces = faces[:1] # 只保留面积最大的一个人脸  
  
 # 绘制人脸矩形  
 for (x, y, w, h) in faces:  
 cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)  
  
 # 从检测到的人脸区域截取图片进行眼睛和微笑检测  
 roi\_gray = gray[y:y + h, x:x + w]  
  
 # 眼睛检测，限制为两个  
 eyes = eye\_cascade.detectMultiScale(roi\_gray)  
 eyes = eyes[:2] # 限制结果最多为两个眼睛  
  
 # 绘制眼睛矩形  
 for (ex, ey, ew, eh) in eyes:  
 cv2.rectangle(img, (ex + x, ey + y), (ex + x + ew, ey + y + eh), (0, 255, 0), 2)  
  
 # 微笑检测，限制为一个  
 smiles = smile\_cascade.detectMultiScale(roi\_gray, scaleFactor=1.1, minNeighbors=5)  
 smiles = smiles[:1] # 限制结果最多为一个微笑  
  
 # 绘制微笑矩形  
 for (sx, sy, sw, sh) in smiles:  
 cv2.rectangle(img, (sx + x, sy + y), (sx + x + sw, sy + y + sh), (0, 0, 255), 2)  
  
 # 将带有标记的图片转换为 QImage 对象  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # 转换为 RGB 以符合 QImage 格式  
 h, w, ch = img.shape  
 qt\_image = QImage(img.data, w, h, QImage.Format\_RGB888)  
  
 # 调整图片大小以适应标签，同时保持纵横比  
 scaled\_image = qt\_image.scaled(981, 581, Qt.KeepAspectRatio)  
 scaled\_image = scaled\_image.convertToFormat(QImage.Format\_ARGB32) # 确保图像不是灰白的  
  
 # 显示在 label\_3 中  
 self.label\_3.setPixmap(QPixmap.fromImage(scaled\_image))

用了OpenCV库来进行图像处理和特征检测。以下是代码中使用到的算法原理的介绍：

1. **文件选择对话框**：通过QFileDialog.getOpenFileName函数弹出一个对话框，让用户选择一个图片文件。
2. **图像读取**：使用cv2.imread函数读取用户选择的图片文件。
3. **颜色空间转换**：将读取到的BGR格式的图像转换为灰度图像，因为许多图像处理任务（如人脸检测）在灰度空间中进行更为高效。
4. **加载Haar特征级联**：加载用于人脸、眼睛和微笑检测的Haar特征级联分类器。Haar特征级联是一种机器学习模型，通过学习不同特征的组合来识别图像中的特定对象。
5. **人脸检测**：使用face\_cascade.detectMultiScale函数在图像中检测人脸。detectMultiScale函数可以在不同的尺度上进行检测，scaleFactor参数控制图像尺寸缩小的比例，minNeighbors参数用于减少误检测。
6. **多个人脸处理**：如果检测到多个人脸，通过比较人脸区域的面积，选择面积最大的一个人脸进行后续处理。
7. **绘制人脸矩形**：在检测到的人脸区域绘制矩形框。
8. **眼睛和微笑检测**：在已检测到的人脸区域（ROI）中，使用相应的Haar特征级联分类器检测眼睛和微笑。对于眼睛，限制最多检测两个眼睛；对于微笑，限制最多检测一个微笑。
9. **绘制眼睛和微笑矩形**：在检测到的眼睛和微笑区域绘制矩形框。
10. **图像格式转换**：将处理后的图像从BGR格式转换为RGB格式，以符合QImage所需的格式。
11. **图像大小调整**：调整图像大小以适应GUI组件，同时保持纵横比，确保图像不变形。
12. **图像显示**：将调整大小后的图像显示在GUI的标签（label\_3）中。

在整个过程中，Haar特征级联起到了核心作用。它通过识别图像中的特定模式来定位人脸、眼睛和微笑。

运行结果：  
图形用户界面, 网站

描述已自动生成 图形用户界面, 应用程序, Teams

描述已自动生成

## 关于视频检测人脸识别：

视频检测人脸按钮对应pushButton\_8，对应槽函数是facevideo，这是facevideo的代码：

def facevideo(self):  
 # 弹出文件选择对话框，选择视频  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择视频文件", "", "Video Files (\*.mp4 \*.avi \*.mov \*.mkv)")  
 if filename:  
 # 使用 OpenCV 读取视频  
 cap = cv2.VideoCapture(filename)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开视频文件，请选择有效的视频文件。")  
 return  
  
 # 加载 Haar 特征级联文件  
 face\_cascade = cv2.CascadeClassifier("E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_frontalface\_alt.xml")  
 eye\_cascade = cv2.CascadeClassifier("E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_eye\_tree\_eyeglasses.xml")  
 smile\_cascade = cv2.CascadeClassifier("E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_smile.xml")  
  
 while True:  
 # 读取视频的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # 将图片转换为灰度图，用于检测  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 人脸检测  
 faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))  
 if len(faces) > 1:  
 faces = sorted(faces, key=lambda f: f[2] \* f[3], reverse=True) # 按面积排序  
 faces = faces[:1] # 只保留面积最大的一个人脸  
  
 # 逐个处理检测到的人脸  
 for (x, y, w, h) in faces:  
 # 绘制人脸矩形  
 cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)  
  
 # 从检测到的人脸区域截取图片进行眼睛和微笑检测  
 roi\_gray = gray[y:y + h, x:x + w]  
  
 # 眼睛检测，限制为两个  
 eyes = eye\_cascade.detectMultiScale(roi\_gray)  
 eyes = eyes[:2] # 限制结果最多为两个眼睛  
  
 # 绘制眼睛矩形  
 for (ex, ey, ew, eh) in eyes:  
 cv2.rectangle(frame, (ex + x, ey + y), (ex + x + ew, ey + y + eh), (0, 255, 0), 2)  
  
 # 微笑检测，限制为一个  
 smiles = smile\_cascade.detectMultiScale(roi\_gray, scaleFactor=1.1, minNeighbors=5)  
 smiles = smiles[:1] # 限制结果最多为一个微笑  
  
 # 绘制微笑矩形  
 for (sx, sy, sw, sh) in smiles:  
 cv2.rectangle(frame, (sx + x, sy + y), (sx + x + sw, sy + y + sh), (0, 0, 255), 2)  
  
 # 将带有标记的视频帧转换为 QImage 对象  
 frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB) # 转换为 RGB 以符合 QImage 格式  
 h, w, ch = frame.shape  
 qt\_image = QImage(frame.data, w, h, QImage.Format\_RGB888)  
  
 # 显示在 label\_3 中  
 self.label\_3.setPixmap(QPixmap.fromImage(qt\_image))  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
 # 释放视频对象  
 cap.release()

1. **文件选择对话框**：通过QFileDialog.getOpenFileName函数弹出一个对话框，让用户选择一个视频文件。
2. **视频读取器初始化**：使用OpenCV的VideoCapture类来读取视频文件。如果无法打开视频文件，会显示警告并退出函数。
3. **加载Haar特征级联**：加载用于人脸、眼睛和微笑检测的Haar特征级联分类器。这些分类器是预先训练好的模型，用于在图像中识别特定的模式。
4. **视频处理循环**：通过循环调用cap.read()方法来逐帧读取视频内容。
5. **颜色空间转换**：将读取到的BGR格式的视频帧转换为灰度图像，因为人脸检测通常在灰度空间中进行，以减少计算量并保持足够的特征信息。
6. **人脸检测**：使用face\_cascade.detectMultiScale函数在图像中检测人脸。detectMultiScale函数可以在不同的尺度上进行检测，scaleFactor参数控制图像尺寸缩小的比例，minNeighbors参数用于减少误检测。
7. **多个人脸处理**：如果检测到多个人脸，通过比较人脸区域的面积，选择面积最大的一个人脸进行后续处理。
8. **绘制人脸矩形**：在检测到的人脸区域绘制矩形框。
9. **眼睛和微笑检测**：在已检测到的人脸区域（ROI）中，使用相应的Haar特征级联分类器检测眼睛和微笑。对于眼睛，限制最多检测两个眼睛；对于微笑，限制最多检测一个微笑。
10. **绘制眼睛和微笑矩形**：在检测到的眼睛和微笑区域绘制矩形框。
11. **图像格式转换**：将处理后的图像从BGR格式转换为RGB格式，以符合QImage所需的格式。
12. **图像显示**：将图像显示在GUI的标签（label\_3）中。
13. **退出条件**：用户可以按'q'键退出视频处理循环。
14. **资源释放**：在结束视频处理后，使用cap.release()方法释放视频读取器资源。

运行结果：  
图形用户界面, 应用程序, 网站

描述已自动生成 图形用户界面, 应用程序

描述已自动生成

## 关于摄像头检测人脸： 摄像头检测人脸对应按钮pushButton\_9，对应槽函数facecamera，facecamera代码如下：

def facecamera(self):  
 # 打开摄像头  
 cap = cv2.VideoCapture(0)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开摄像头。")  
 return  
  
 # 加载 Haar 特征级联文件  
 face\_cascade = cv2.CascadeClassifier(  
 "E:\\Tool\\GitHub\\Opencv-master\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade\_frontalface\_alt.xml"  
 )  
 eye\_cascade = cv2.CascadeClassifier(  
 "E:\\Tool\\GitHub\\Opencv-master\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade\_eye\_tree\_eyeglasses.xml"  
 )  
 smile\_cascade = cv2.CascadeClassifier(  
 "E:\\Tool\\GitHub\\Opencv-master\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade\_smile.xml"  
 )  
  
 try:  
 while True:  
 # 读取摄像头的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # 将图片转换为灰度图，用于检测  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 人脸检测  
 faces = face\_cascade.detectMultiScale(  
 gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30)  
 )  
  
 # 限制只处理一个人脸  
 if len(faces) > 0:  
 (x, y, w, h) = faces[0]  
 cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)  
  
 # 从检测到的人脸区域截取图片进行眼睛和微笑检测  
 roi\_gray = gray[y:y + h, x:x + w]  
  
 # 眼睛检测  
 eyes = eye\_cascade.detectMultiScale(roi\_gray)  
 if len(eyes) == 2: # 限制两个眼睛  
 for (ex, ey, ew, eh) in eyes:  
 cv2.rectangle(frame, (ex + x, ey + y), (ex + x + ew, ey + y + eh), (0, 255, 0), 2)  
  
 # 微笑检测  
 smiles = smile\_cascade.detectMultiScale(roi\_gray, scaleFactor=1.1, minNeighbors=5)  
 if len(smiles) > 0: # 限制一个微笑  
 (sx, sy, sw, sh) = smiles[0]  
 cv2.rectangle(frame, (sx + x, sy + y), (sx + x + sw, sy + y + sh), (0, 0, 255), 2)  
  
  
  
  
 # 显示处理后的帧  
 cv2.imshow('Face Camera', frame)  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
 except Exception as e:  
 print("Error during face detection from camera:", e)  
 QMessageBox.warning(self, "错误", "面部检测出错。")  
  
 finally:  
 # 释放摄像头资源  
 cap.release()  
 # 关闭所有 OpenCV 窗口  
 cv2.destroyAllWindows()

1. **摄像头初始化**：使用OpenCV的VideoCapture类来初始化并打开系统的默认摄像头。如果摄像头无法打开，会显示警告并退出函数。
2. **加载Haar特征级联**：加载用于人脸、眼睛和微笑检测的Haar特征级联分类器。这些分类器是预先训练好的模型，能够在图像中识别特定的模式。
3. **视频处理循环**：通过循环调用cap.read()方法来逐帧读取摄像头的视频流。
4. **颜色空间转换**：将读取到的BGR格式的视频帧转换为灰度图像，因为许多图像处理任务（如人脸检测）在灰度空间中进行更为高效。
5. **人脸检测**：使用face\_cascade.detectMultiScale函数在图像中检测人脸。detectMultiScale函数可以在不同的尺度上进行检测，scaleFactor参数控制图像尺寸缩小的比例，minNeighbors参数用于减少误检测。
6. **单个人脸处理**：如果检测到人脸，选择第一个检测到的人脸进行后续处理。在人脸区域绘制矩形框。
7. **眼睛检测**：在已检测到的人脸区域（ROI）中，使用eye\_cascade.detectMultiScale函数检测眼睛。如果检测到两个眼睛，则在每个眼睛区域绘制矩形框。
8. **微笑检测**：在相同的人脸ROI中，使用smile\_cascade.detectMultiScale函数检测微笑。如果检测到微笑，则在微笑区域绘制矩形框。
9. **图像显示**：使用cv2.imshow函数显示带有检测标记的视频帧。
10. **退出条件**：用户可以按'q'键退出视频处理循环。
11. **异常处理**：使用try...except结构来捕获和处理在视频处理和面部检测过程中可能发生的异常。
12. **资源释放**：在结束视频处理后，使用cap.release()方法释放摄像头资源，并使用cv2.destroyAllWindows()关闭所有OpenCV窗口。

运行结果：  
图形用户界面, 网站

描述已自动生成

## 图像拼接

图形用户界面, 应用程序

描述已自动生成

关于图像拼接：  
图片1对应按钮pushButton\_10，对应的槽函数是choiceA，图片2对应按钮pushButton\_11，对应的槽函数是choiceB，拼接对应的按钮是pushButton\_12，对应的槽函数是AandB，然后还需要用到select\_image;convert\_cv\_qt;detect;match\_keypoints;drawMatches这几个函数，下面是代码：

def choiceB(self):  
 # 用户选择图片B  
 self.select\_image('B')  
  
def choiceA(self):  
 # 用户选择图片A  
 self.select\_image('A')  
  
def select\_image(self, image\_type):  
 # 弹出文件选择对话框，选择图片  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择图片", "", "Image Files (\*.png \*.jpg \*.jpeg \*.bmp)")  
 if filename:  
 # 读取图片  
 img = cv2.imread(filename)  
 if img is None:  
 QMessageBox.warning(self, "警告", "无法加载图片，请选择有效的图片文件。")  
 else: # 显示图片

cv2.imshow(f"Selected {image\_type} Image", img)

cv2.waitKey(0) # 等待用户按下任意键以关闭窗口

# 根据选择的图片类型存储图片  
 # 根据选择的图片类型存储图片  
 if image\_type == 'A':  
 self.imageA = img  
 elif image\_type == 'B':  
 self.imageB = img  
 else:  
 QMessageBox.warning(self, "警告", "无效的图片类型。")  
  
  
def AandB(self):  
 img\_left = self.imageA  
 img\_right = self.imageB  
 # 模块一：提取特征  
 kps\_left, features\_left = MainWindow.detect(img\_left)  
 kps\_right, features\_right = MainWindow.detect(img\_right)  
 # 模块二：特征匹配  
 matches, H, good = MainWindow.match\_keypoints(kps\_left, kps\_right, features\_left, features\_right, 0.5, 0.99)  
 # 模块三：透视变换-拼接  
 vis = MainWindow.drawMatches(img\_left, img\_right, kps\_left, kps\_right, matches, H)  
 # 显示拼接图形  
 # plt.figure(),plt.axis('off')  
 # plt.imshow(cv2.cvtColor(vis, cv2.COLOR\_BGR2RGB))  
 # plt.show()  
 qt\_image = self.convert\_cv\_qt(vis)  
 # 显示在 label\_4 中  
 self.label\_4.setPixmap(QPixmap.fromImage(qt\_image))  
  
def convert\_cv\_qt(self, cv\_img):  
 *"""Convert from an opencv image to QPixmap"""* rgb\_image = cv2.cvtColor(cv\_img, cv2.COLOR\_BGR2RGB)  
 h, w, ch = rgb\_image.shape  
 bytes\_per\_line = ch \* w  
 convert\_to\_Qt\_format = QImage(rgb\_image.data, w, h, bytes\_per\_line, QImage.Format\_RGB888)  
 p = convert\_to\_Qt\_format.scaled(981, 581, Qt.KeepAspectRatio)  
 return p  
  
  
  
def detect(image):  
 # 创建SIFT生成器  
 # descriptor是一个对象，这里使用的是SIFT算法  
 descriptor = cv2.SIFT\_create()  
 # 检测特征点及其描述子（128维向量）  
 kps, features = descriptor.detectAndCompute(image, None)  
 return (kps, features)  
  
  
def match\_keypoints(kps\_left, kps\_right, features\_left, features\_right, ratio, threshold):  
 # kps\_left,kps\_right,features\_left,features\_right: 两幅图形的特征点坐标及特征向量  
 # 创建暴力匹配器  
 matcher = cv2.DescriptorMatcher\_create("BruteForce")  
 # 使用knn检测，匹配left,right图的特征点  
 raw\_matches = matcher.knnMatch(features\_left, features\_right, 2)  
 print('左右图的匹配特征点数：', len(raw\_matches))  
 matches = [] # 存坐标，为了后面  
 good = [] # 存对象，为了后面的演示  
 # 筛选匹配点  
 for m in raw\_matches:  
 # 筛选条件  
 # print(m[0].distance,m[1].distance)  
 if len(m) == 2 and m[0].distance < m[1].distance \* ratio:  
 good.append([m[0]])  
 matches.append((m[0].queryIdx, m[0].trainIdx))  
 """  
 queryIdx：测试图像的特征点描述符的下标==>img\_left  
 trainIdx：样本图像的特征点描述符下标==>img\_right  
 distance：代表这怡翠匹配的特征点描述符的欧式距离，数值越小也就说明俩个特征点越相近。  
 """  
 # 特征点对数大于4就够用来构建变换矩阵了  
 kps\_left = np.float32([kp.pt for kp in kps\_left])  
 kps\_right = np.float32([kp.pt for kp in kps\_right])  
 print('筛选后匹配点数:', len(matches))  
 if len(matches) > 4:  
 # 获取匹配点坐标  
 pts\_left = np.float32([kps\_left[i] for (i, \_) in matches])  
 pts\_right = np.float32([kps\_right[i] for (\_, i) in matches])  
 # 计算变换矩阵(采用ransac算法从pts中选择一部分点)  
 H, status = cv2.findHomography(pts\_right, pts\_left, cv2.RANSAC, threshold)  
 return (matches, H, good)  
 return None  
  
def drawMatches(img\_left, img\_right, kps\_left, kps\_right, matches, H):  
 # 获取图片宽度和高度  
 h\_left, w\_left = img\_left.shape[:2]  
 h\_right, w\_right = img\_right.shape[:2]  
 """对imgB进行透视变换  
 由于透视变换会改变图片场景的大小，导致部分图片内容看不到  
 所以对图片进行扩展:高度取最高的，宽度为两者相加"""  
 image = np.zeros((max(h\_left, h\_right), w\_left + w\_right, 3), dtype='uint8')  
 # 初始化  
 image[0:h\_left, 0:w\_left] = img\_right  
 # 利用以获得的单应性矩阵进行变透视换"""  
 image = cv2.warpPerspective(image, H, (image.shape[1], image.shape[0])) # (w,h  
 # 将透视变换后的图片与另一张图片进行拼接"""  
 image[0:h\_left, 0:w\_left] = img\_left  
 return image

1. **图像选择 (choiceA 和 choiceB)**：
   * 这两个函数允许用户通过弹出的文件选择对话框选择图片，会先后出现两个窗口，显示用户选择了哪两个图片，并根据用户选择（图片A或图片B）将图片存储在相应的类属性中。
2. **通用图像选择 (select\_image)**：
   * 弹出文件选择对话框，让用户选择图片文件，然后使用cv2.imread读取图片。如果图片读取成功，根据传入的image\_type参数，将图片存储在类的不同属性中。
3. **特征点检测 (detect)**：
   * 使用SIFT（尺度不变特征变换）算法来检测图像中的关键点并计算它们的特征描述符。SIFT是一种流行的局部特征提取算法，能够提取对旋转、缩放和亮度变化保持不变的特征点。
4. **特征点匹配 (match\_keypoints)**：
   * 使用暴力匹配器（BruteForce）和knnMatch函数来找到两个图像特征描述符之间的匹配点。通过设置匹配比率ratio和距离阈值，筛选出最佳的匹配点对。
   * 使用RANSAC算法计算单应性矩阵H，该算法能够从匹配点对中选择最佳的点对来计算变换矩阵，同时忽略离群点。
5. **透视变换和拼接 (drawMatches)**：
   * 使用cv2.warpPerspective函数对一个图像进行透视变换，使其与另一个图像对齐。透视变换是一种几何变换，可以校正视角差异或将图像的一部分与另一部分对齐。
   * 将变换后的图像与原图像拼接在一起，创建一个无缝的大图像。
6. **图像显示**：
   * 将处理后的图像通过OpenCV转换为QPixmap对象，并显示在GUI的标签（label\_4）中。
7. **图像格式转换 (convert\_cv\_qt)**：
   * 将OpenCV图像格式转换为PyQt的QImage格式，以便于在图形用户界面中显示。
8. **异常处理**：
   * 使用try...except结构来捕获和处理在图像处理过程中可能发生的异常。
9. **资源释放**：
   * 在结束图像处理后，释放相关的资源，如关闭摄像头和销毁所有OpenCV窗口。

整个过程中，SIFT特征提取和RANSAC算法是关键技术。SIFT用于提取图像的局部特征并生成特征描述符，而RANSAC用于从特征点匹配中选择最佳点对，计算单应性矩阵，从而实现图像的透视变换和拼接。

运行结果：

注意：图片1和图片2显示后，需要关掉两个图片的窗口，才能进行拼接。  
图形用户界面, 应用程序, 网站

描述已自动生成图形用户界面, 应用程序

描述已自动生成图形用户界面, 应用程序, 网站

描述已自动生成图形用户界面, 应用程序

描述已自动生成火车开在铁轨上

描述已自动生成

## 心得体会

**遇到的困难：**

1. **环境配置问题**：在开始编写代码之前，我需要确保开发环境中安装了所有必需的库，如PyQt5、OpenCV、PIL、pyzbar等。由于不同库的版本兼容性问题，我花费了大量时间来解决依赖冲突。
2. **代码逻辑错误**：在实现图像处理和视频流处理功能时，我遇到了许多逻辑上的错误。例如，最初我未能正确处理图像和视频流中的异常情况，导致程序崩溃。
3. **UI设计挑战**：使用PyQt5设计用户界面时，我需要理解信号与槽的机制，以及如何将按钮点击事件与相应的处理函数相连接。起初，我对如何布局组件以及如何响应用户操作感到困惑。
4. **多线程处理**：在处理实时视频流和摄像头输入时，我需要了解多线程的概念，以确保用户界面的流畅性。这对我来说是一个全新的领域。
5. **错误处理**：我需要确保程序能够优雅地处理各种错误情况，如文件读取失败、摄像头无法打开等。编写健壮的错误处理代码是一个挑战。

**如何解决：**

1. **系统学习**：我通过在线教程和文档系统地学习了所需的库和框架，这帮助我理解了它们的工作原理和最佳实践。
2. **小步快跑**：我采用了分阶段的方法来构建程序，每次只添加一小部分功能，然后进行测试。这使得问题更容易被识别和解决。
3. **社区支持**：我利用了CSDN，GitHub等在线社区，从其他开发者那里学习如何解决类似问题。
4. **代码审查**：我向同学和老师展示了我的代码，他们的反馈帮助我改进了设计并发现了潜在的问题。
5. **实践和调试**：我通过大量的实践和调试来提高代码质量。我学会了使用调试器来逐步跟踪程序的执行，并识别问题所在。

**心得体会：**

1. **持续学习**：技术不断进步，作为一名软件工程师，我必须持续学习新工具和技术。
2. **实践是关键**：理论知识很重要，但没有什么比亲自动手写代码更能加深理解。
3. **耐心和毅力**：编程是一个复杂的过程，需要大量的耐心和毅力。遇到问题时不要轻易放弃。
4. **错误是进步的阶梯**：遇到错误和失败是编程过程中不可避免的，但它们是学习和成长的机会。

通过这次经历，我不仅提升了自己的编程技能，还学会了如何更有效地解决问题。

附录

GUI4.py:

# -\*- coding: utf-8 -\*-  
  
# Form implementation generated from reading ui file 'UI3.ui'  
#  
# Created by: PyQt5 UI code generator 5.15.7  
#  
# 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, QtGui, QtWidgets  
  
  
class Ui\_MainWindow(object):  
 def setupUi(self, MainWindow):  
 MainWindow.setObjectName("MainWindow")  
 MainWindow.resize(1112, 845)  
 self.centralwidget = QtWidgets.QWidget(MainWindow)  
 self.centralwidget.setObjectName("centralwidget")  
 self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)  
 self.tabWidget.setGeometry(QtCore.QRect(0, 0, 1111, 811))  
 self.tabWidget.setObjectName("tabWidget")  
 self.tab = QtWidgets.QWidget()  
 self.tab.setObjectName("tab")  
 self.pushButton = QtWidgets.QPushButton(self.tab)  
 self.pushButton.setGeometry(QtCore.QRect(90, 660, 131, 51))  
 self.pushButton.setObjectName("pushButton")  
 self.pushButton\_2 = QtWidgets.QPushButton(self.tab)  
 self.pushButton\_2.setGeometry(QtCore.QRect(220, 660, 131, 51))  
 self.pushButton\_2.setObjectName("pushButton\_2")  
 self.pushButton\_3 = QtWidgets.QPushButton(self.tab)  
 self.pushButton\_3.setGeometry(QtCore.QRect(350, 660, 141, 51))  
 self.pushButton\_3.setObjectName("pushButton\_3")  
 self.label = QtWidgets.QLabel(self.tab)  
 self.label.setGeometry(QtCore.QRect(50, 30, 981, 581))  
 self.label.setObjectName("label")  
 self.tabWidget.addTab(self.tab, "")  
 self.tab\_2 = QtWidgets.QWidget()  
 self.tab\_2.setObjectName("tab\_2")  
 self.label\_2 = QtWidgets.QLabel(self.tab\_2)  
 self.label\_2.setGeometry(QtCore.QRect(60, 20, 981, 581))  
 self.label\_2.setObjectName("label\_2")  
 self.pushButton\_4 = QtWidgets.QPushButton(self.tab\_2)  
 self.pushButton\_4.setGeometry(QtCore.QRect(110, 650, 121, 51))  
 self.pushButton\_4.setObjectName("pushButton\_4")  
 self.pushButton\_5 = QtWidgets.QPushButton(self.tab\_2)  
 self.pushButton\_5.setGeometry(QtCore.QRect(230, 650, 121, 51))  
 self.pushButton\_5.setObjectName("pushButton\_5")  
 self.pushButton\_6 = QtWidgets.QPushButton(self.tab\_2)  
 self.pushButton\_6.setGeometry(QtCore.QRect(350, 650, 121, 51))  
 self.pushButton\_6.setObjectName("pushButton\_6")  
 self.tabWidget.addTab(self.tab\_2, "")  
 self.tab\_3 = QtWidgets.QWidget()  
 self.tab\_3.setObjectName("tab\_3")  
 self.label\_3 = QtWidgets.QLabel(self.tab\_3)  
 self.label\_3.setGeometry(QtCore.QRect(60, 20, 981, 581))  
 self.label\_3.setObjectName("label\_3")  
 self.pushButton\_7 = QtWidgets.QPushButton(self.tab\_3)  
 self.pushButton\_7.setGeometry(QtCore.QRect(130, 640, 121, 51))  
 self.pushButton\_7.setObjectName("pushButton\_7")  
 self.pushButton\_8 = QtWidgets.QPushButton(self.tab\_3)  
 self.pushButton\_8.setGeometry(QtCore.QRect(250, 640, 121, 51))  
 self.pushButton\_8.setObjectName("pushButton\_8")  
 self.pushButton\_9 = QtWidgets.QPushButton(self.tab\_3)  
 self.pushButton\_9.setGeometry(QtCore.QRect(370, 640, 121, 51))  
 self.pushButton\_9.setObjectName("pushButton\_9")  
 self.tabWidget.addTab(self.tab\_3, "")  
 self.tab\_4 = QtWidgets.QWidget()  
 self.tab\_4.setObjectName("tab\_4")  
 self.label\_4 = QtWidgets.QLabel(self.tab\_4)  
 self.label\_4.setGeometry(QtCore.QRect(60, 20, 981, 581))  
 self.label\_4.setObjectName("label\_4")  
 self.pushButton\_10 = QtWidgets.QPushButton(self.tab\_4)  
 self.pushButton\_10.setGeometry(QtCore.QRect(130, 670, 121, 51))  
 self.pushButton\_10.setObjectName("pushButton\_10")  
 self.pushButton\_11 = QtWidgets.QPushButton(self.tab\_4)  
 self.pushButton\_11.setGeometry(QtCore.QRect(250, 670, 121, 51))  
 self.pushButton\_11.setObjectName("pushButton\_11")  
 self.pushButton\_12 = QtWidgets.QPushButton(self.tab\_4)  
 self.pushButton\_12.setGeometry(QtCore.QRect(370, 670, 121, 51))  
 self.pushButton\_12.setObjectName("pushButton\_12")  
 self.tabWidget.addTab(self.tab\_4, "")  
 MainWindow.setCentralWidget(self.centralwidget)  
 self.statusbar = QtWidgets.QStatusBar(MainWindow)  
 self.statusbar.setObjectName("statusbar")  
 MainWindow.setStatusBar(self.statusbar)  
  
 self.retranslateUi(MainWindow)  
 self.tabWidget.setCurrentIndex(0)  
 QtCore.QMetaObject.connectSlotsByName(MainWindow)  
  
 def retranslateUi(self, MainWindow):  
 \_translate = QtCore.QCoreApplication.translate  
 MainWindow.setWindowTitle(\_translate("MainWindow", "MainWindow"))  
 self.pushButton.setText(\_translate("MainWindow", "图片识别二维码"))  
 self.pushButton\_2.setText(\_translate("MainWindow", "视频识别二维码"))  
 self.pushButton\_3.setText(\_translate("MainWindow", "摄像头识别二维码"))  
 self.label.setText(\_translate("MainWindow", "<html><head/><body><p align=\"center\">输出</p></body></html>"))  
 self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab), \_translate("MainWindow", "二维码识别"))  
 self.label\_2.setText(\_translate("MainWindow", "<html><head/><body><p align=\"center\">输出</p></body></html>"))  
 self.pushButton\_4.setText(\_translate("MainWindow", "图片识别车牌"))  
 self.pushButton\_5.setText(\_translate("MainWindow", "视频识别车牌"))  
 self.pushButton\_6.setText(\_translate("MainWindow", "摄像头识别车牌"))  
 self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab\_2), \_translate("MainWindow", "车牌识别"))  
 self.label\_3.setText(\_translate("MainWindow", "<html><head/><body><p align=\"center\">输出</p></body></html>"))  
 self.pushButton\_7.setText(\_translate("MainWindow", "图片检测人脸"))  
 self.pushButton\_8.setText(\_translate("MainWindow", "视频检测人脸"))  
 self.pushButton\_9.setText(\_translate("MainWindow", "摄像头检测人脸"))  
 self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab\_3), \_translate("MainWindow", "人脸检测"))  
 self.label\_4.setText(\_translate("MainWindow", "<html><head/><body><p align=\"center\">输出</p></body></html>"))  
 self.pushButton\_10.setText(\_translate("MainWindow", "图片1"))  
 self.pushButton\_11.setText(\_translate("MainWindow", "图片2"))  
 self.pushButton\_12.setText(\_translate("MainWindow", "拼接"))  
 self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab\_4), \_translate("MainWindow", "图片拼接"))

mainForm4.py:

from PyQt5.QtGui import QImage, QPixmap  
from PyQt5.QtCore import Qt  
from PyQt5.QtWidgets import QApplication, QMainWindow, QFileDialog, QMessageBox, QLabel  
from PIL import Image  
import pyzbar.pyzbar as pyzbar  
from GUI4 import \*  
import cv2  
import matplotlib.pyplot as plt  
import imutils  
import os  
import numpy as np  
import pytesseract  
pytesseract.pytesseract.tesseract\_cmd = r"E:/Yan.All2/Tesseract-OCR/tesseract.exe"  
import logging  
from paddleocr import PaddleOCR  
logging.getLogger('paddleocr').setLevel(logging.WARNING)  
# 初始化车牌检测器  
carplate\_haar = cv2.CascadeClassifier(  
 r"E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_russian\_plate\_number.xml")  
ocr = PaddleOCR(use\_angle\_cls=True)  
  
  
class MainWindow(QMainWindow, Ui\_MainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 self.setupUi(self)  
 cv2.ocl.setUseOpenCL(False)  
 self.pushButton.clicked.connect(self.QRimage) # 连接按钮的 clicked 信号到 QRimage 方法  
 self.pushButton\_2.clicked.connect(self.QRvideo) # 连接按钮的 clicked 信号到 QRvideo 方法  
 self.pushButton\_3.clicked.connect(self.QRcamera) # 连接摄像头识别二维码按钮的信号到 QRcamera 方法  
 self.pushButton\_4.clicked.connect(self.IDimage)  
 self.pushButton\_5.clicked.connect(self.IDvideo)  
 self.pushButton\_6.clicked.connect(self.IDcamera)  
 self.pushButton\_7.clicked.connect(self.faceimage)  
 self.pushButton\_8.clicked.connect(self.facevideo)  
 self.pushButton\_9.clicked.connect(self.facecamera)  
 self.pushButton\_10.clicked.connect(self.choiceA)  
 self.pushButton\_11.clicked.connect(self.choiceB)  
 self.pushButton\_12.clicked.connect(self.AandB)  
 self.imageA = None # 初始化图片A  
 self.imageB = None # 初始化图片B  
  
 def choiceB(self):  
 # 用户选择图片B  
 self.select\_image('B')  
  
 def choiceA(self):  
 # 用户选择图片A  
 self.select\_image('A')  
  
 def select\_image(self, image\_type):  
 # 弹出文件选择对话框，选择图片  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择图片", "", "Image Files (\*.png \*.jpg \*.jpeg \*.bmp)")  
 if filename:  
 # 读取图片  
 img = cv2.imread(filename)  
 if img is None:  
 QMessageBox.warning(self, "警告", "无法加载图片，请选择有效的图片文件。")  
 else:  
 # 显示图片  
 cv2.imshow(f"Selected {image\_type} Image", img)  
 cv2.waitKey(0) # 等待用户按下任意键以关闭窗口  
 # 根据选择的图片类型存储图片  
 if image\_type == 'A':  
 self.imageA = img  
 elif image\_type == 'B':  
 self.imageB = img  
 else:  
 QMessageBox.warning(self, "警告", "无效的图片类型。")  
  
  
 def AandB(self):  
 img\_left = self.imageA  
 img\_right = self.imageB  
 # 模块一：提取特征  
 kps\_left, features\_left = MainWindow.detect(img\_left)  
 kps\_right, features\_right = MainWindow.detect(img\_right)  
 # 模块二：特征匹配  
 matches, H, good = MainWindow.match\_keypoints(kps\_left, kps\_right, features\_left, features\_right, 0.5, 0.99)  
 # 模块三：透视变换-拼接  
 vis = MainWindow.drawMatches(img\_left, img\_right, kps\_left, kps\_right, matches, H)  
 # 显示拼接图形  
 # plt.figure(),plt.axis('off')  
 # plt.imshow(cv2.cvtColor(vis, cv2.COLOR\_BGR2RGB))  
 # plt.show()  
 qt\_image = self.convert\_cv\_qt(vis)  
 # 显示在 label\_4 中  
 self.label\_4.setPixmap(QPixmap.fromImage(qt\_image))  
  
 def convert\_cv\_qt(self, cv\_img):  
  
 rgb\_image = cv2.cvtColor(cv\_img, cv2.COLOR\_BGR2RGB)  
 h, w, ch = rgb\_image.shape  
 bytes\_per\_line = ch \* w  
 convert\_to\_Qt\_format = QImage(rgb\_image.data, w, h, bytes\_per\_line, QImage.Format\_RGB888)  
 p = convert\_to\_Qt\_format.scaled(981, 581, Qt.KeepAspectRatio)  
 return p  
  
  
  
 def detect(image):  
 # 创建SIFT生成器  
 # descriptor是一个对象，这里使用的是SIFT算法  
 descriptor = cv2.SIFT\_create()  
 # 检测特征点及其描述子（128维向量）  
 kps, features = descriptor.detectAndCompute(image, None)  
 return (kps, features)  
  
  
 def match\_keypoints(kps\_left, kps\_right, features\_left, features\_right, ratio, threshold):  
 # kps\_left,kps\_right,features\_left,features\_right: 两幅图形的特征点坐标及特征向量  
 # 创建暴力匹配器  
 matcher = cv2.DescriptorMatcher\_create("BruteForce")  
 # 使用knn检测，匹配left,right图的特征点  
 raw\_matches = matcher.knnMatch(features\_left, features\_right, 2)  
 print('左右图的匹配特征点数：', len(raw\_matches))  
 matches = [] # 存坐标，为了后面  
 good = [] # 存对象，为了后面的演示  
 # 筛选匹配点  
 for m in raw\_matches:  
 # 筛选条件  
 # print(m[0].distance,m[1].distance)  
 if len(m) == 2 and m[0].distance < m[1].distance \* ratio:  
 good.append([m[0]])  
 matches.append((m[0].queryIdx, m[0].trainIdx))  
 """  
 queryIdx：测试图像的特征点描述符的下标==>img\_left  
 trainIdx：样本图像的特征点描述符下标==>img\_right  
 distance：代表这怡翠匹配的特征点描述符的欧式距离，数值越小也就说明俩个特征点越相近。  
 """  
 # 特征点对数大于4就够用来构建变换矩阵了  
 kps\_left = np.float32([kp.pt for kp in kps\_left])  
 kps\_right = np.float32([kp.pt for kp in kps\_right])  
 print('筛选后匹配点数:', len(matches))  
 if len(matches) > 4:  
 # 获取匹配点坐标  
 pts\_left = np.float32([kps\_left[i] for (i, \_) in matches])  
 pts\_right = np.float32([kps\_right[i] for (\_, i) in matches])  
 # 计算变换矩阵(采用ransac算法从pts中选择一部分点)  
 H, status = cv2.findHomography(pts\_right, pts\_left, cv2.RANSAC, threshold)  
 return (matches, H, good)  
 return None  
  
 def drawMatches(img\_left, img\_right, kps\_left, kps\_right, matches, H):  
 # 获取图片宽度和高度  
 h\_left, w\_left = img\_left.shape[:2]  
 h\_right, w\_right = img\_right.shape[:2]  
 """对imgB进行透视变换  
 由于透视变换会改变图片场景的大小，导致部分图片内容看不到  
 所以对图片进行扩展:高度取最高的，宽度为两者相加"""  
 image = np.zeros((max(h\_left, h\_right), w\_left + w\_right, 3), dtype='uint8')  
 # 初始化  
 image[0:h\_left, 0:w\_left] = img\_right  
 # 利用以获得的单应性矩阵进行变透视换"""  
 image = cv2.warpPerspective(image, H, (image.shape[1], image.shape[0])) # (w,h  
 # 将透视变换后的图片与另一张图片进行拼接"""  
 image[0:h\_left, 0:w\_left] = img\_left  
 return image  
  
 def carplate\_detect(image):#检测图像中的车牌区域。  
 carplate\_rects = carplate\_haar.detectMultiScale(image, scaleFactor=1.1, minNeighbors=3)  
 return carplate\_rects  
  
 def carplate\_extract(image, rects):#从原始图像中裁剪出车牌图像。  
 plates = []  
 for x, y, w, h in rects:  
 carplate\_img = image[y + 15:y + h - 10, x + 15:x + w - 20]  
 plates.append(carplate\_img)  
 return plates  
  
 def ocr\_plate(image):#光学字符识别  
 results = ocr.ocr(image, cls=True)  
 if results is None or len(results) == 0:  
 return "No plate detected"  
 else:  
 try:  
 # 返回第一个识别结果  
 return results[0][0][1][0]  
 except IndexError:  
 # 如果索引操作失败，返回未检测到车牌  
 return "No plate detected"  
  
 def IDimage(self):  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择图片", "", "Image Files (\*.png \*.jpg \*.jpeg \*.bmp)")  
 detected\_plates = []  
 if filename:  
 # 读取图片  
 img = cv2.imread(filename)  
 cv2.imshow('Original Image', img)  
 cv2.waitKey(0)  
 cv2.destroyAllWindows()  
 if img is None:  
 QMessageBox.warning(self, "警告", "无法加载图片，请选择有效的图片文件。")  
 return  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 rects = MainWindow.carplate\_detect(gray)  
 for plate\_img in MainWindow.carplate\_extract(gray, rects):  
 plate\_str = MainWindow.ocr\_plate(plate\_img)  
 if plate\_str != "No plate detected": # 过滤掉未检测到车牌的情况  
 detected\_plates.append(plate\_str)  
  
 # 将列表转换为字符串，用逗号和空格分隔  
 plates\_str = ', '.join(detected\_plates)  
  
 # 设置标签的文本  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)  
  
 def IDvideo(self):  
 # 弹出选择文件对话框，选择视频文件  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择视频", "", "Video Files (\*.mp4 \*.avi \*.mov \*.mkv)")  
 if not filename:  
 print("No video selected.")  
 return  
  
 # 初始化视频读取器  
 cap = cv2.VideoCapture(filename)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开视频文件，请选择有效的视频文件。")  
 return  
  
 # 清空标签的文本  
 self.label\_2.clear()  
  
 tick = cv2.getTickCount()  
 while True:  
 # 读取视频的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break # 如果视频结束或无法读取帧，则退出循环  
  
 # 转换为灰度图  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 车牌检测  
 rects = MainWindow.carplate\_detect(gray) # 确保这里调用的是实例方法  
  
 # 存储检测到的车牌号，同时去除重复  
 unique\_plates = set()  
  
 for plate\_img in MainWindow.carplate\_extract(gray, rects): # 同上  
 plate\_str = MainWindow.ocr\_plate(plate\_img) # 同上  
 if plate\_str != "No plate detected":  
 unique\_plates.add(plate\_str)  
  
 # 将集合转换为字符串，用逗号和空格分隔  
 plates\_str = ', '.join(unique\_plates)  
  
 # 更新UI，显示车牌号码  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)  
  
 # 处理所有等待的事件，确保UI响应性  
 QApplication.processEvents()  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
 delay = max(0, 33 - (cv2.getTickCount() - tick), )  
 cv2.waitKey(delay) # 等待一定的毫秒数  
  
 # 更新下一次循环的开始时间  
 tick = cv2.getTickCount()  
 cv2.imshow('Video', frame)  
  
  
  
 # 释放视频读取器并关闭所有OpenCV窗口  
 cap.release()  
 cv2.destroyAllWindows()  
  
 # 最后更新一次标签，以显示最后一次检测到的车牌号码  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)  
  
 def IDcamera(self):  
 # 初始化摄像头  
 cap = cv2.VideoCapture(0) # 参数0通常表示系统的默认摄像头  
  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开摄像头，请检查摄像头连接。")  
 return  
  
 # 清空标签的文本  
 self.label\_2.clear()  
  
 try:  
 while True:  
 # 读取摄像头的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break # 如果摄像头结束或无法读取帧，则退出循环  
  
 # 转换为灰度图  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 车牌检测  
 rects = MainWindow.carplate\_detect(gray)  
  
 # 存储检测到的车牌号，同时去除重复  
 unique\_plates = set()  
  
 for plate\_img in MainWindow.carplate\_extract(gray, rects):  
 plate\_str = MainWindow.ocr\_plate(plate\_img)  
 if plate\_str != "No plate detected":  
 unique\_plates.add(plate\_str)  
  
 # 将集合转换为字符串，用逗号和空格分隔  
 plates\_str = ', '.join(unique\_plates)  
  
 # 更新UI，显示车牌号码  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)  
  
 # 处理所有等待的事件，确保UI响应性  
 QApplication.processEvents()  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
 finally:  
 # 释放摄像头并关闭所有OpenCV窗口  
 cap.release()  
 cv2.destroyAllWindows()  
  
 # 最后更新一次标签，以显示最后一次检测到的车牌号码  
 self.label\_2.setText(plates\_str)  
 self.label\_2.setAlignment(Qt.AlignCenter)  
  
 def faceimage(self):  
 # 弹出文件选择对话框，选择图片  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择图片", "", "Image Files (\*.png \*.jpg \*.jpeg \*.bmp)")  
 if filename:  
 # 读取图片  
 img = cv2.imread(filename)  
 if img is None:  
 QMessageBox.warning(self, "警告", "无法加载图片，请选择有效的图片文件。")  
 return  
  
 # 将图片转换为灰度图，用于检测  
 gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
  
 # 加载 Haar 特征级联文件  
 face\_cascade = cv2.CascadeClassifier(  
 "E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_frontalface\_alt.xml")  
 eye\_cascade = cv2.CascadeClassifier(  
 "E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_eye\_tree\_eyeglasses.xml")  
 smile\_cascade = cv2.CascadeClassifier(  
 "E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_smile.xml")  
  
 # 人脸检测  
 faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))  
 if len(faces) > 1: # 如果检测到多个人脸，选择置信度最高的一个  
 faces = sorted(faces, key=lambda f: f[2] \* f[3], reverse=True) # 按面积排序  
 faces = faces[:1] # 只保留面积最大的一个人脸  
  
 # 绘制人脸矩形  
 for (x, y, w, h) in faces:  
 cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 0), 2)  
  
 # 从检测到的人脸区域截取图片进行眼睛和微笑检测  
 roi\_gray = gray[y:y + h, x:x + w]  
  
 # 眼睛检测，限制为两个  
 eyes = eye\_cascade.detectMultiScale(roi\_gray)  
 eyes = eyes[:2] # 限制结果最多为两个眼睛  
  
 # 绘制眼睛矩形  
 for (ex, ey, ew, eh) in eyes:  
 cv2.rectangle(img, (ex + x, ey + y), (ex + x + ew, ey + y + eh), (0, 255, 0), 2)  
  
 # 微笑检测，限制为一个  
 smiles = smile\_cascade.detectMultiScale(roi\_gray, scaleFactor=1.1, minNeighbors=5)  
 smiles = smiles[:1] # 限制结果最多为一个微笑  
  
 # 绘制微笑矩形  
 for (sx, sy, sw, sh) in smiles:  
 cv2.rectangle(img, (sx + x, sy + y), (sx + x + sw, sy + y + sh), (0, 0, 255), 2)  
  
 # 将带有标记的图片转换为 QImage 对象  
 img = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB) # 转换为 RGB 以符合 QImage 格式  
 h, w, ch = img.shape  
 qt\_image = QImage(img.data, w, h, QImage.Format\_RGB888)  
  
 # 调整图片大小以适应标签，同时保持纵横比  
 scaled\_image = qt\_image.scaled(981, 581, Qt.KeepAspectRatio)  
 scaled\_image = scaled\_image.convertToFormat(QImage.Format\_ARGB32) # 确保图像不是灰白的  
  
 # 显示在 label\_3 中  
 self.label\_3.setPixmap(QPixmap.fromImage(scaled\_image))  
  
 def facevideo(self):  
 # 弹出文件选择对话框，选择视频  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择视频文件", "", "Video Files (\*.mp4 \*.avi \*.mov \*.mkv)")  
 if filename:  
 # 使用 OpenCV 读取视频  
 cap = cv2.VideoCapture(filename)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开视频文件，请选择有效的视频文件。")  
 return  
  
 # 加载 Haar 特征级联文件  
 face\_cascade = cv2.CascadeClassifier("E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_frontalface\_alt.xml")  
 eye\_cascade = cv2.CascadeClassifier("E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_eye\_tree\_eyeglasses.xml")  
 smile\_cascade = cv2.CascadeClassifier("E:\Tool\GitHub\Opencv-master\opencv-master\opencv-master\data\haarcascades\haarcascade\_smile.xml")  
  
 while True:  
 # 读取视频的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # 将图片转换为灰度图，用于检测  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 人脸检测  
 faces = face\_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))  
 if len(faces) > 1:  
 faces = sorted(faces, key=lambda f: f[2] \* f[3], reverse=True) # 按面积排序  
 faces = faces[:1] # 只保留面积最大的一个人脸  
  
 # 逐个处理检测到的人脸  
 for (x, y, w, h) in faces:  
 # 绘制人脸矩形  
 cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)  
  
 # 从检测到的人脸区域截取图片进行眼睛和微笑检测  
 roi\_gray = gray[y:y + h, x:x + w]  
  
 # 眼睛检测，限制为两个  
 eyes = eye\_cascade.detectMultiScale(roi\_gray)  
 eyes = eyes[:2] # 限制结果最多为两个眼睛  
  
 # 绘制眼睛矩形  
 for (ex, ey, ew, eh) in eyes:  
 cv2.rectangle(frame, (ex + x, ey + y), (ex + x + ew, ey + y + eh), (0, 255, 0), 2)  
  
 # 微笑检测，限制为一个  
 smiles = smile\_cascade.detectMultiScale(roi\_gray, scaleFactor=1.1, minNeighbors=5)  
 smiles = smiles[:1] # 限制结果最多为一个微笑  
  
 # 绘制微笑矩形  
 for (sx, sy, sw, sh) in smiles:  
 cv2.rectangle(frame, (sx + x, sy + y), (sx + x + sw, sy + y + sh), (0, 0, 255), 2)  
  
 # 将带有标记的视频帧转换为 QImage 对象  
 frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB) # 转换为 RGB 以符合 QImage 格式  
 h, w, ch = frame.shape  
 qt\_image = QImage(frame.data, w, h, QImage.Format\_RGB888)  
  
 # 显示在 label\_3 中  
 self.label\_3.setPixmap(QPixmap.fromImage(qt\_image))  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
 # 释放视频对象  
 cap.release()  
  
 def facecamera(self):  
 # 打开摄像头  
 cap = cv2.VideoCapture(0)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开摄像头。")  
 return  
  
 # 加载 Haar 特征级联文件  
 face\_cascade = cv2.CascadeClassifier(  
 "E:\\Tool\\GitHub\\Opencv-master\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade\_frontalface\_alt.xml"  
 )  
 eye\_cascade = cv2.CascadeClassifier(  
 "E:\\Tool\\GitHub\\Opencv-master\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade\_eye\_tree\_eyeglasses.xml"  
 )  
 smile\_cascade = cv2.CascadeClassifier(  
 "E:\\Tool\\GitHub\\Opencv-master\\opencv-master\\opencv-master\\data\\haarcascades\\haarcascade\_smile.xml"  
 )  
  
 try:  
 while True:  
 # 读取摄像头的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # 将图片转换为灰度图，用于检测  
 gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  
  
 # 人脸检测  
 faces = face\_cascade.detectMultiScale(  
 gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30)  
 )  
  
 # 限制只处理一个人脸  
 if len(faces) > 0:  
 (x, y, w, h) = faces[0]  
 cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)  
  
 # 从检测到的人脸区域截取图片进行眼睛和微笑检测  
 roi\_gray = gray[y:y + h, x:x + w]  
  
 # 眼睛检测  
 eyes = eye\_cascade.detectMultiScale(roi\_gray)  
 if len(eyes) == 2: # 限制两个眼睛  
 for (ex, ey, ew, eh) in eyes:  
 cv2.rectangle(frame, (ex + x, ey + y), (ex + x + ew, ey + y + eh), (0, 255, 0), 2)  
  
 # 微笑检测  
 smiles = smile\_cascade.detectMultiScale(roi\_gray, scaleFactor=1.1, minNeighbors=5)  
 if len(smiles) > 0: # 限制一个微笑  
 (sx, sy, sw, sh) = smiles[0]  
 cv2.rectangle(frame, (sx + x, sy + y), (sx + x + sw, sy + y + sh), (0, 0, 255), 2)  
  
  
  
  
 # 显示处理后的帧  
 cv2.imshow('Face Camera', frame)  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
 except Exception as e:  
 print("Error during face detection from camera:", e)  
 QMessageBox.warning(self, "错误", "面部检测出错。")  
  
 finally:  
 # 释放摄像头资源  
 cap.release()  
 # 关闭所有 OpenCV 窗口  
 cv2.destroyAllWindows()  
  
 def QRimage(self):  
 # 弹出文件选择对话框，选择图片  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择图片", "", "Image Files (\*.png \*.jpg \*.jpeg \*.bmp)")  
 if filename:  
 try:  
 # 使用PIL打开图像文件  
 pil\_image = Image.open(filename)  
  
 # 将PIL图像转换为OpenCV格式  
 cv\_image = np.array(pil\_image)  
 cv\_image = cv2.cvtColor(cv\_image, cv2.COLOR\_RGB2BGR) # PIL图像默认是RGB，OpenCV默认是BGR  
  
 # 使用pyzbar库进行二维码识别  
 decoded\_objects = pyzbar.decode(cv\_image)  
  
 # 初始化解码数据列表  
 decoded\_data = []  
  
 # 遍历解码对象，提取数据  
 for obj in decoded\_objects:  
 if obj.type == "QRCODE":  
 # 解码数据并添加到列表中  
 decoded\_data.append(obj.data.decode('utf-8'))  
  
 # 准备显示结果的文本  
 if decoded\_data:  
 result\_text = "<html><head/><body><p align=\"center\">识别结果:</p></body></html>" + "".join(  
 f"<p align=\"center\">{data}</p>" for data in decoded\_data  
 )  
 else:  
 result\_text = "<html><head/><body><p align=\"center\">未识别到二维码</p></body></html>"  
  
 # 更新label显示识别结果  
 self.label.setText(result\_text)  
  
 # 使用cv2.imshow显示图片  
 cv2.imshow('QR Code Image', cv\_image)  
 cv2.waitKey(0) # 等待用户按键  
  
 except IOError as e:  
 # 如果文件打开失败，显示错误消息  
 QMessageBox.warning(self, "错误", f"无法打开图片文件: {e}")  
 except Exception as e:  
 # 显示其他可能的错误  
 QMessageBox.warning(self, "错误", f"二维码识别出错: {e}")  
 finally:  
 # 关闭所有OpenCV窗口  
 cv2.destroyAllWindows()  
  
 def QRvideo(self):  
 # 弹出文件选择对话框，选择视频  
 filename, \_ = QFileDialog.getOpenFileName(self, "选择视频文件", "", "Video Files (\*.mp4 \*.avi \*.mov \*.mkv)")  
 if filename:  
 # 使用 OpenCV 读取视频  
 cap = cv2.VideoCapture(filename)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开视频文件，请选择有效的视频文件。")  
 return  
  
 previous\_data = None # 用于存储上一次的识别结果  
 decoded\_data = [] # 用于存储解码结果  
  
 try:  
 # 创建一个窗口来显示视频  
 cv2.namedWindow('Video', cv2.WINDOW\_NORMAL)  
  
 while True:  
 # 读取视频的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # 将BGR帧转换为PIL Image  
 image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
 img = Image.fromarray(image)  
  
 # 使用pyzbar库进行二维码识别  
 decoded\_objects = pyzbar.decode(img)  
  
 # 检查是否识别到二维码  
 if decoded\_objects:  
 for obj in decoded\_objects:  
 if obj.type == "QRCODE":  
 # 获取解码的数据  
 data = obj.data.decode('utf-8')  
 # 如果当前识别结果与上一个不同，则更新显示结果  
 if data != previous\_data:  
 previous\_data = data  
 decoded\_data.append(data)  
  
 # 显示当前帧  
 cv2.imshow('Video', frame)  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
 # 释放视频对象  
 cap.release()  
 # 关闭所有OpenCV窗口  
 cv2.destroyAllWindows()  
  
 # 将识别结果按照换行符输出  
 if decoded\_data:  
 result\_text = "<html><head/><body><p align=\"center\">识别结果:</p></body></html>" + "\n".join(  
 f"<p align=\"center\">{data}</p>" for data in set(decoded\_data) # 使用 set 来去除重复项  
 )  
 self.label.setText(result\_text)  
 else:  
 self.label.setText("<html><head/><body><p align=\"center\">视频中未识别到二维码</p></body></html>")  
  
 except Exception as e:  
 print("Error during QR code decoding from video:", e)  
 self.label.setText("<html><head/><body><p align=\"center\">二维码识别出错</p></body></html>")  
  
  
  
 def QRcamera(self):  
 # 打开摄像头 按下q可以退出摄像头  
 cap = cv2.VideoCapture(0)  
 if not cap.isOpened():  
 QMessageBox.warning(self, "警告", "无法打开摄像头。")  
 return  
  
 try:  
 while True:  
 # 读取摄像头的下一帧  
 ret, frame = cap.read()  
 if not ret:  
 break  
  
 # 将 BGR 帧转换为 PIL Image  
 image = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
 img = Image.fromarray(image)  
  
 # 使用pyzbar库进行二维码识别  
 decoded\_objects = pyzbar.decode(img)  
  
 # 检查是否识别到二维码  
 if decoded\_objects:  
 decoded\_data = [obj.data.decode('utf-8') for obj in decoded\_objects if obj.type == 'QRCODE']  
 if decoded\_data:  
 # 如果识别到二维码，将结果按照换行符输出  
 result\_text = "<html><head/><body><p align=\"center\">识别结果:</p></body></html>" + \  
 "\n".join(f"<p align=\"center\">{data}</p>" for data in decoded\_data)  
 self.label.setText(result\_text)  
 else:  
 self.label.setText("<html><head/><body><p align=\"center\">未识别到二维码</p></body></html>")  
 else:  
 self.label.setText("<html><head/><body><p align=\"center\">未识别到二维码</p></body></html>")  
  
 # 显示处理后的帧  
 cv2.imshow('QR Camera', frame)  
  
 # 按 'q' 退出循环  
 if cv2.waitKey(1) & 0xFF == ord('q'):  
 break  
  
 except Exception as e:  
 print("Error during QR code decoding from camera:", e)  
 self.label.setText("<html><head/><body><p align=\"center\">二维码识别出错</p></body></html>")  
  
 finally:  
 # 释放摄像头资源  
 cap.release()  
 # 关闭所有 OpenCV 窗口  
 cv2.destroyAllWindows()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 import sys  
 app = QApplication(sys.argv)  
 mainWindow = MainWindow()  
 mainWindow.show()  
 sys.exit(app.exec\_())