基于人脸特征点位识别的单目视线估计与眼动绘图

**第二组 郑名哲 陈昊哲 陈文康 王涛**

## 概述

目前对于眼动的研究主要依赖于双目眼动仪，即一个对准瞳孔的红外相机用于计算视线向量，和一个场景相机用于估计视线在场景中的观测点，这对于居家实验环境并不友好。

因此，我们采取了基于facial landmarks和eyes keypoints的单目视线估计策略，借助预训练模型对眼部与瞳孔点位进行跟踪后，将坐标关于屏幕成比例放大，实现单目视线估计。

此外，我们制作了基于PyQT的画图程序，基于单目视线估计实现了眼动操控鼠标移动和单击选择画图工具；同时为了避免眼动控制功能过于复杂，我们引入了对嘴部keypoints的检测与计算，实现嘴部开闭控制开始绘图/停止绘图。

Github链接：<https://github.com/DuNGEOnmassster/HCI_Course_Design_playground/tree/main/Group2>

## 关键算法

**2.1 Landmarks提取工具dlib**

**2.1.1 基本介绍**

 dlib是一个机器学习的开源库，包含了机器学习的很多算法，使用起来很方便，直接包含头文件即可，并且不依赖于其他库（自带图像编解码库源码）。Dlib可以帮助您创建很多复杂的机器学习方面的软件来帮助解决实际问题。目前Dlib已经被广泛的用在行业和学术领域,包括机器人,嵌入式设备,移动电话和大型高性能计算环境。

Dlib是一个使用现代C++技术编写的跨平台的通用库，遵守Boost Software licence. 主要特点如下：

● 完善的文档：每个类每个函数都有详细的文档，并且提供了大量的示例代码，如果你发现文档描述不清晰或者没有文档，告诉作者，作者会立刻添加。

● 可移植代码：代码符合ISO C++标准，不需要第三方库支持，支持win32、Linux、Mac OS X、Solaris、HPUX、BSDs 和 POSIX 系统 。

● 线程支持：提供简单的可移植的线程API 。

● 网络支持：提供简单的可移植的Socket API和一个简单的Http服务器 。

● 图形用户界面：提供线程安全的GUI API 。

● 数值算法：矩阵、大整数、随机数运算等 。

● 机器学习算法

● 图形模型算法

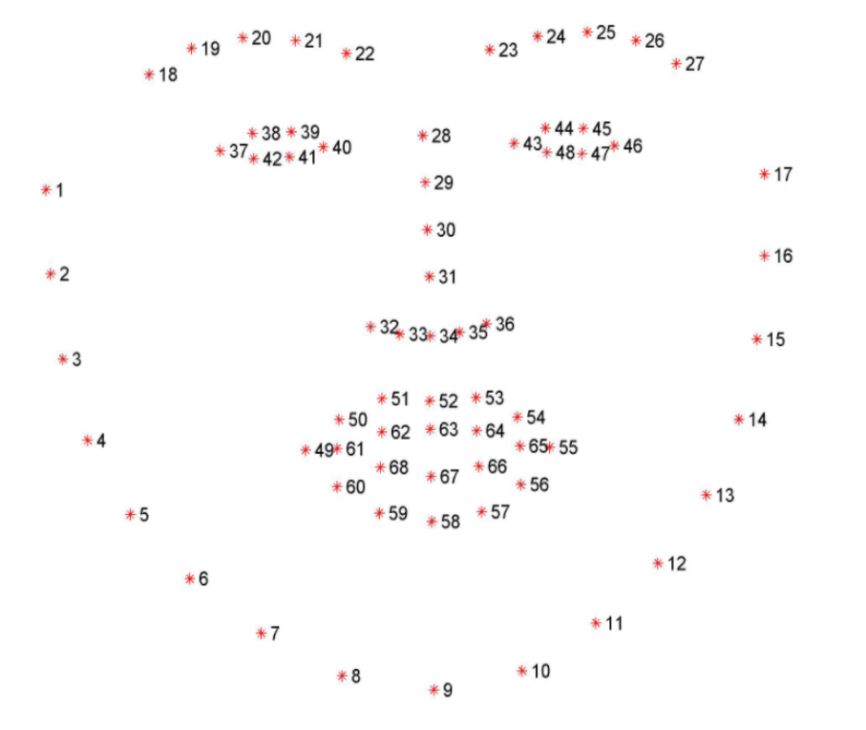
● 图像处理：支持读写Windows BMP文件，不同类型色彩转换

● 数据压缩和完整性算法：CRC32、Md5、不同形式的PPM算法

● 测试：线程安全的日志类和模块化的单元测试框架以及各种测试assert支持

● 一般工具：XML解析、内存管理、类型安全的big/little endian转换、序列化支持和容器类

[shape\_predictor\_68\_face\_landmarks.dat.bz2](http://dlib.net/files/shape_predictor_68_face_landmarks.dat.bz2)模型能够检测人脸的68个特征点，其中在人脸中68个关键点位置示意图如下



检查图像，我们可以看到面部区域可以通过简单的 Python 索引访问（假设使用 Python 进行零索引，因为上图是单索引)：

嘴巴： [48, 68] 。

右眉：[17, 22]。

左眉： [22, 27]。

右眼： [36, 42]。

左眼： [42, 48]。

鼻子： [27, 35]。

下巴： [0, 17]。

这些映射在 imutils 库的 face\_utils 内的 FACIAL\_LANDMARKS\_IDXS 字典中编码：

# 定义一个映射面部索引的字典

# 特定面部区域的标记

FACIAL\_LANDMARKS\_IDXS = OrderedDict([

("mouth", (48, 68)),

("right\_eyebrow", (17, 22)),

("left\_eyebrow", (22, 27)),

("right\_eye", (36, 42)),

("left\_eye", (42, 48)),

("nose", (27, 35)),

("jaw", (0, 17))

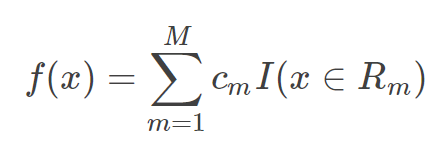
])

使用这个字典，我们可以轻松地将索引提取到面部标志数组中，并只需提供一个字符串作为键即可提取各种面部特征。

**2.1.2人脸关键点检测实现原理**

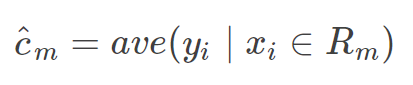
EDT（Ensemble of Regression Trees）

### **回归树模型**

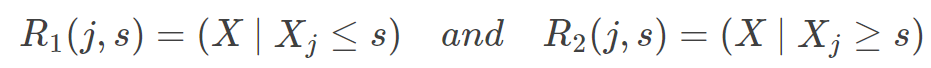
首先简单的介绍回归树。回归树是一种将特征空间进行分割，之后在每一个分割空间进行拟合的简单模型。其数学表达式可以些为：

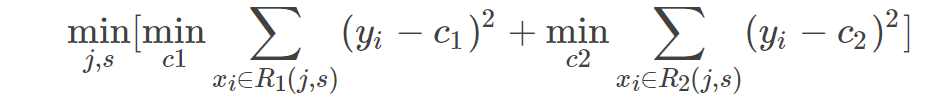
式中 R1,R2,...,RmR1,R2,...,Rm为m个划分的子空间， cmcm为每个子空间对应的权值，加和之后就是这一个回归树模型的输出。建立回归树的过程也就是寻找使误差和模型输出最小的划分和权值的过程。

以平方误差为例，定义误差为首先考虑最优的 cmcm，设定预测模型中表示为 c^mc^m，可以很容易的看到在每个划分的子空间 RR中，最优的cmcm就是当前子空间对应的真实预测值的平均，可以表示为：



这样就很容易找到了适合的 cmcm，于是如何建立回归树这一问题的关键在于如何确定特征子空间的划分。这一寻找过程可以描述如下：

1. 在特征集合中找到用于分割的点ss，和分割的变量jj，这样可以将空间划分为两部分
2. 之后在最小平方准则下找到最优的j,s

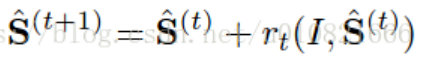


其中的c1,c2可以通过上面提到的公式找到，即：



于是，通过这样的方式就可以建立一个回归树模型。

**回归器的级联**



其中S(t)S(t)表示第t级回归器的形状，是一个由坐标组成的向量，t表示级联的级数，I为图像，rtrt表示当前级的回归器，是需要训练的量。

基于树的回归器 tree based regressor

* 形状不变的分裂测试 shape invariant split tests
* 选择节点进行分裂 choosing the node splits
* 特征选择 feature selection

特征选择和每棵树中的节点如何分裂，合并起来一起说。

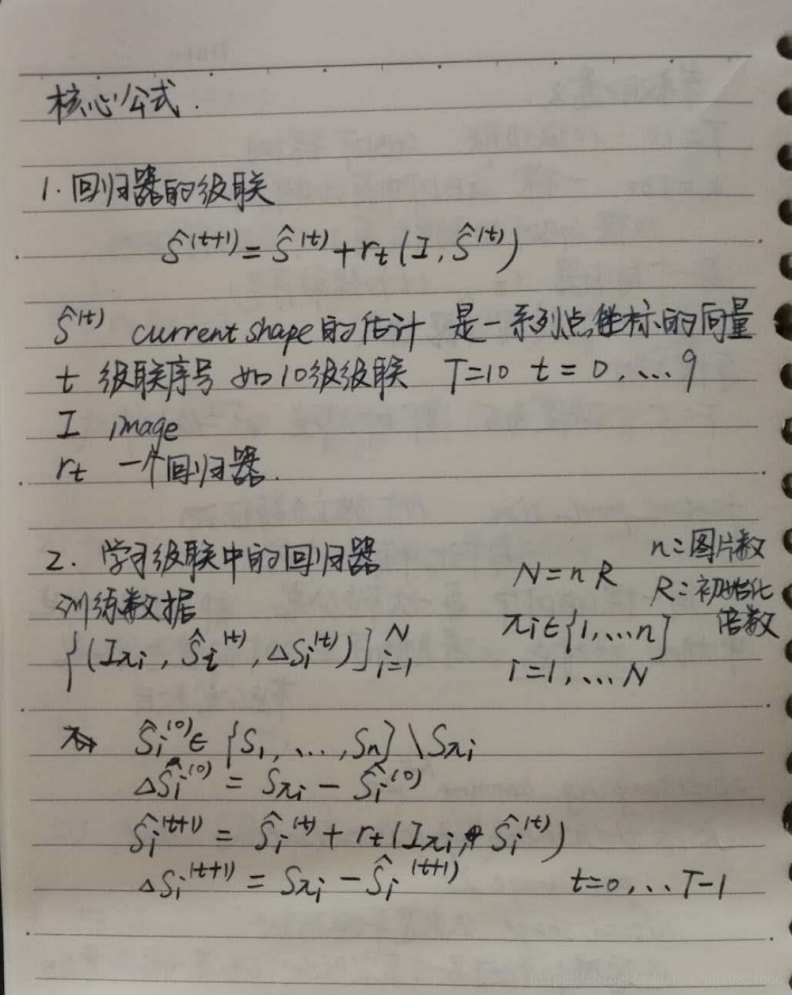
特征是像素差 pixel difference。

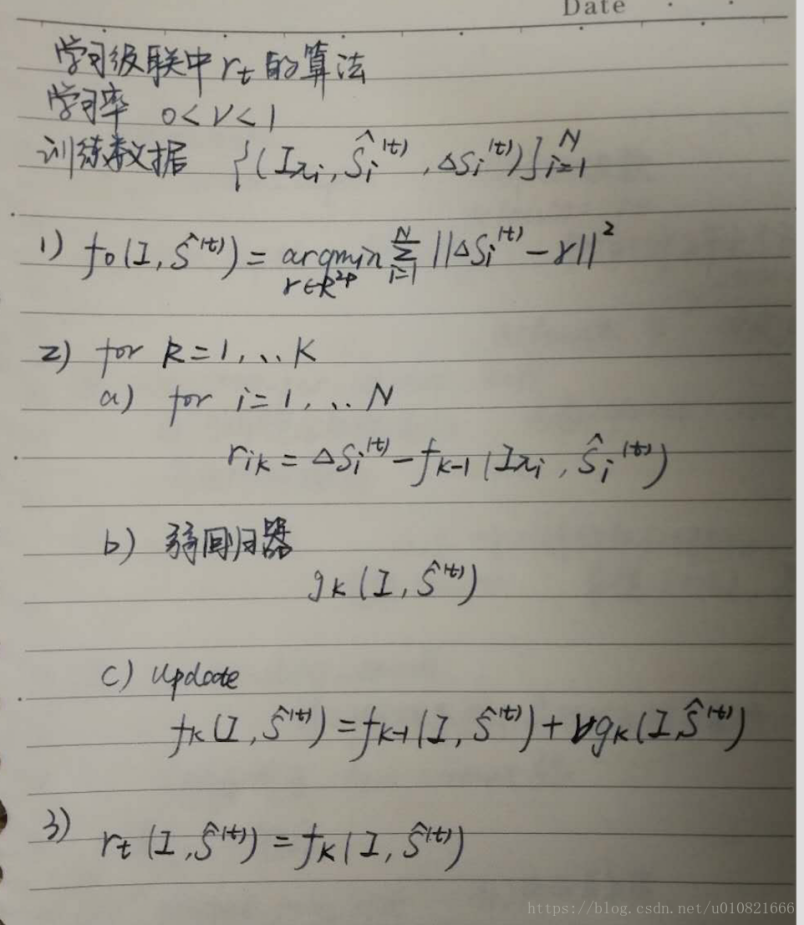
建立一个特征池feature pool（所有的GBDT树，比如说10棵，相互独立，那就有10个特征池）。特征池中有几百上千个随机挑选的点的坐标（对应于每一幅图像中的不同像素值）。

节点分裂的时候，从特征池中挑选两个点，计算每一幅图像在这两个点的像素值和像素差pixel difference，然后随机产生一个分裂阈值，使用分裂阈值作为分裂依据，如果像素差小于这个阈值向左分裂，反之向右分裂。

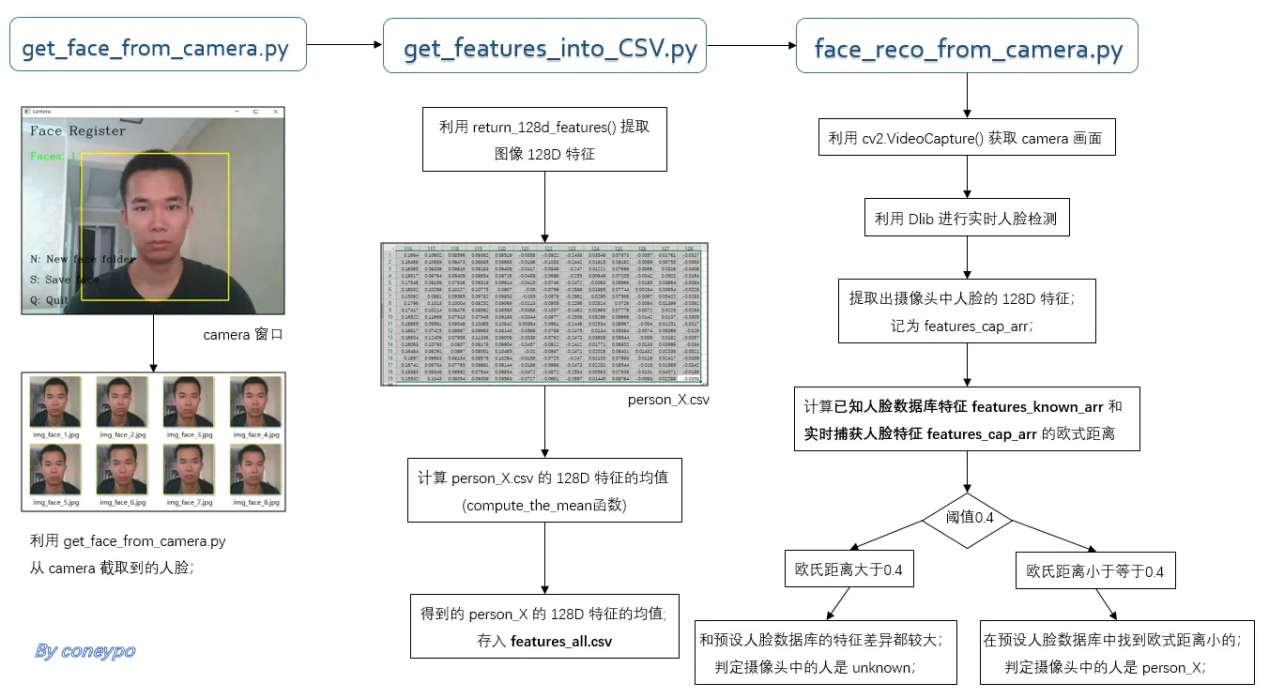
重复这样的过程，可以将所有的图片分为两部分（一个节点分裂，将图片分为左右两部分），然后根据方差判断根据哪两个点（特征池中挑选的）和哪个阈值分裂地比较好，就选择这个点和阈值。

依次分裂直至叶子节点（都是二叉树，且深度指定，叶子节点的数量是固定的）。





1. 处理缺失标签 handling missing labels

**2.1.3 Dlib流程**

**2.2 Keypoints提取工具mediapipe**

**2.2.1 基本介绍**

MediaPipe 是一款由 Google Research 开发并开源的多媒体机器学习模型应用框架。

MediaPipe 的核心框架由 C++ 实现，并提供 Java 以及 Objective C 等语言的支持。MediaPipe 的主要概念包括数据包（Packet）、数据流（Stream）、计算单元（Calculator）、图（Graph）以及子图（Subgraph）。数据包是最基础的数据单位，一个数据包代表了在某一特定时间节点的数据，例如一帧图像或一小段音频信号；数据流是由按时间顺序升序排列的多个数据包组成，一个数据流的某一特定时间戳（Timestamp）只允许至多一个数据包的存在；而数据流则是在多个计算单元构成的图中流动。MediaPipe 的图是有向的——数据包从数据源（Source Calculator或者 Graph Input Stream）流入图直至在汇聚结点（Sink Calculator 或者Graph Output Stream）离开。

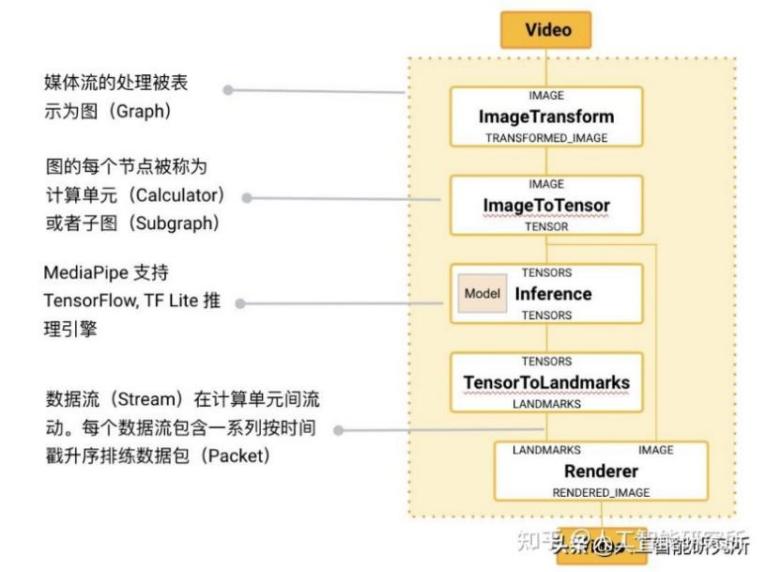


图2 mediapipe基本流程

OpenCV+mediapipe实现虹膜检测

MediaPipe的人脸landmark提供了468个点位的人脸点云数据，数据的编号如下：

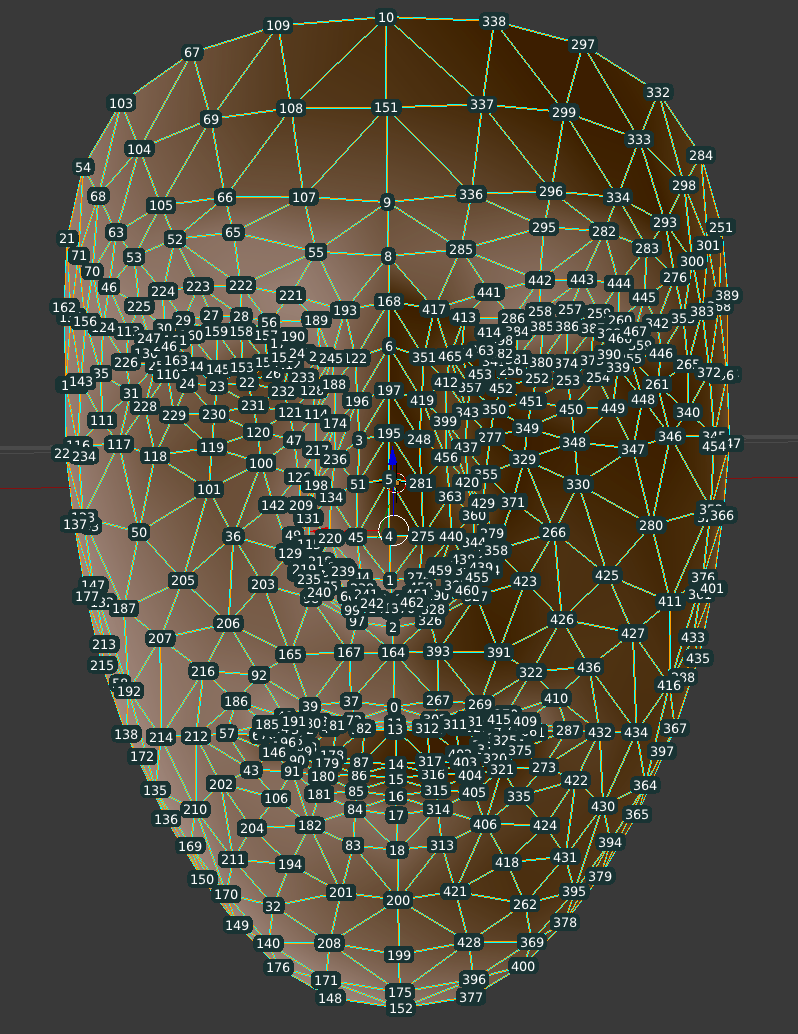


图3 mediapipe面部特征点

根据编号，容易筛选出左眼与右眼所在的区域所有点，然后选取需要的特征点，作为后续处理程序的输入。

关键代码：

\_, frame = cam.read()  
 frame = cv2.flip(frame, 1)

frame\_h, frame\_w, \_ = frame.shape # 获取图像尺寸：高，宽  
 rgb\_frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)  
 output = face\_mesh.process(rgb\_frame)  
 landmark\_points = output.multi\_face\_landmarks # 获取关键点位

landmarks = landmark\_points[0].landmark # 获取面部关键点

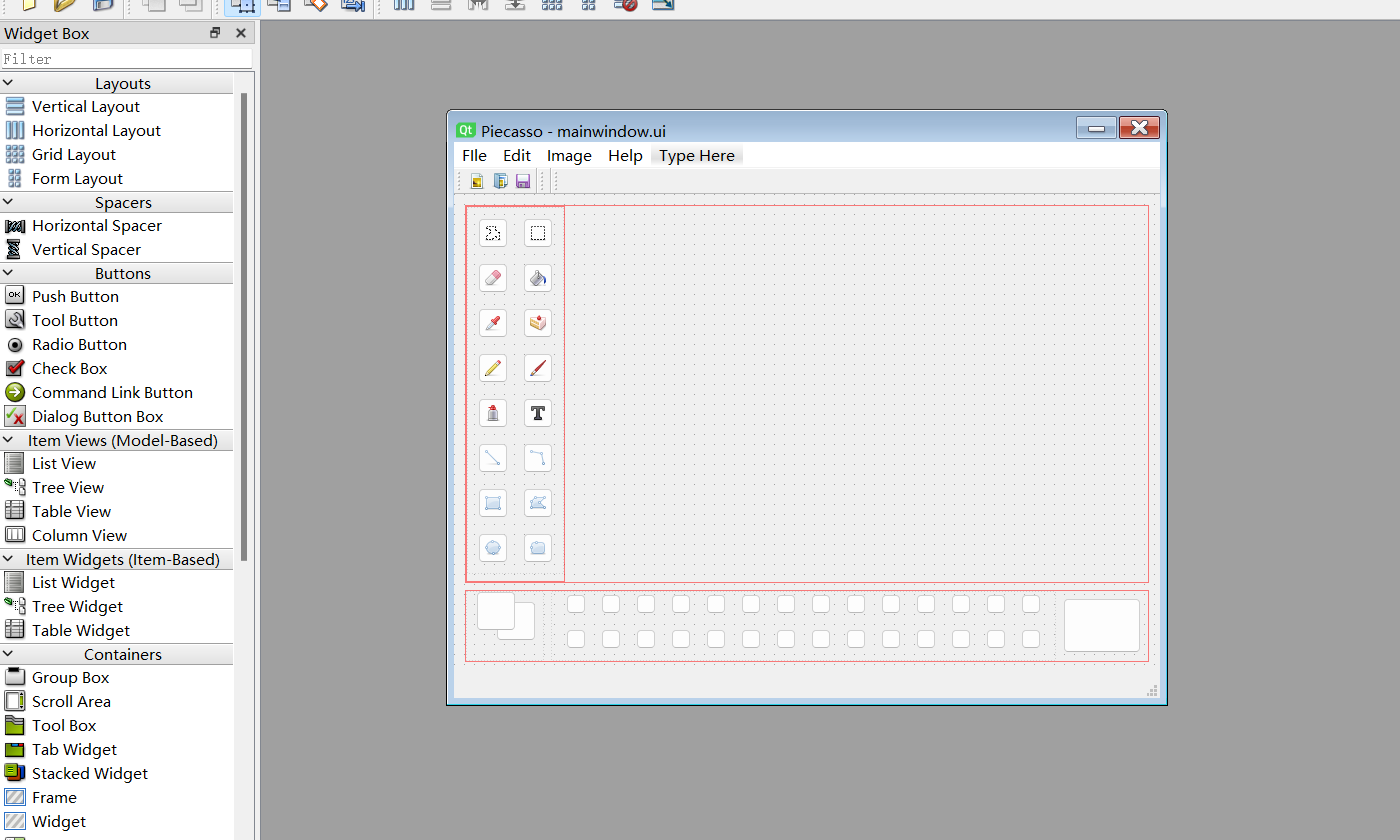
# 得到关键点位于图像中的的坐标  
 for landmark landmarks：   
 x = int(landmark.x \* frame\_w)  
 y = int(landmark.y \* frame\_h)

实际使用过程中，通过opencv将视频流转化为图像，使用mediapipe对每张图像进行虹膜相关特征点位提取，即可获得实时的虹膜位置与特征。

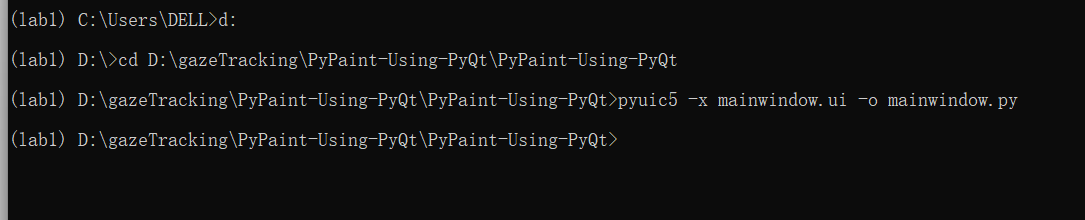
**2.3 PyQT画图关键算法**

**2.3.1 UI架构**

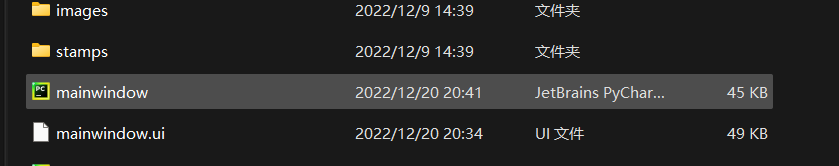
首先在QtDesigner上设计好UI，保存为.ui文件

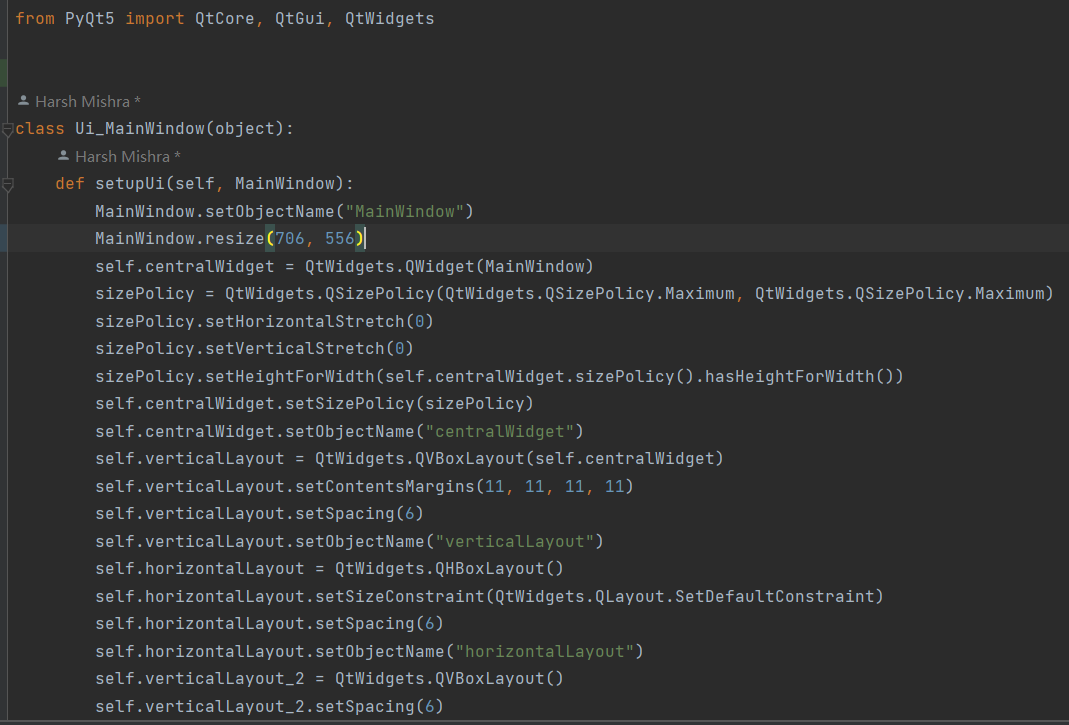


然后使用pyuic5命令转换为.py文件即可



成功





**2.3.2 Button控件原理**

在MainWindow.py中布局窗口，以一段代码了解一下：

MainWindow.setObjectName("MainWindow")  
MainWindow.resize(549, 452)  
self.centralWidget = QtWidgets.QWidget(MainWindow)

在paint.py中定义了Canvas，MainWindow两个类

其中，Canvas类定义了几种常见的事件：

def mousePressEvent(self, e):  
 fn = getattr(self, "%s\_mousePressEvent" % self.mode, None)  
 if fn:  
 return fn(e)  
  
def mouseMoveEvent(self, e):  
 fn = getattr(self, "%s\_mouseMoveEvent" % self.mode, None)  
 if fn:  
 return fn(e)  
  
def mouseReleaseEvent(self, e):  
 fn = getattr(self, "%s\_mouseReleaseEvent" % self.mode, None)  
 if fn:  
 return fn(e)  
  
def mouseDoubleClickEvent(self, e):  
 fn = getattr(self, "%s\_mouseDoubleClickEvent" % self.mode, None)  
 if fn:  
 return fn(e)

并实现了按键的功能，如：

def stamp\_mousePressEvent(self, e):  
 p = QPainter(self.pixmap())  
 stamp = self.current\_stamp  
 p.drawPixmap(e.x() - stamp.width() // 2, e.y() - stamp.height() // 2, stamp)  
 self.update()

在MainWindow类中实现了界面交互的部分，如：

鼠标点击切换stamp

def next\_stamp(self):  
 self.current\_stamp\_n += 1  
 if self.current\_stamp\_n >= len(STAMPS):  
 self.current\_stamp\_n = 0  
  
 pixmap = QPixmap(STAMPS[self.current\_stamp\_n])  
 self.stampnextButton.setIcon(QIcon(pixmap))  
  
 self.canvas.current\_stamp = pixmap

翻转功能：

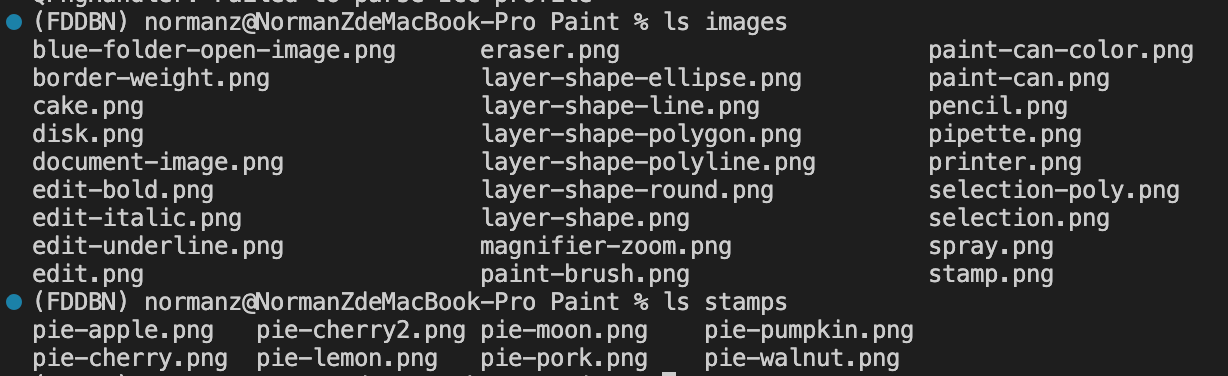
def flip\_horizontal(self):  
 pixmap = self.canvas.pixmap()  
 self.canvas.setPixmap(pixmap.transformed(QTransform().scale(-1, 1)))  
  
def flip\_vertical(self):  
 pixmap = self.canvas.pixmap()  
 self.canvas.setPixmap(pixmap.transformed(QTransform().scale(1, -1)))

**2.2.3 图标添加原理**

以添加蛋糕按键图标与蛋糕种类选择图标为例

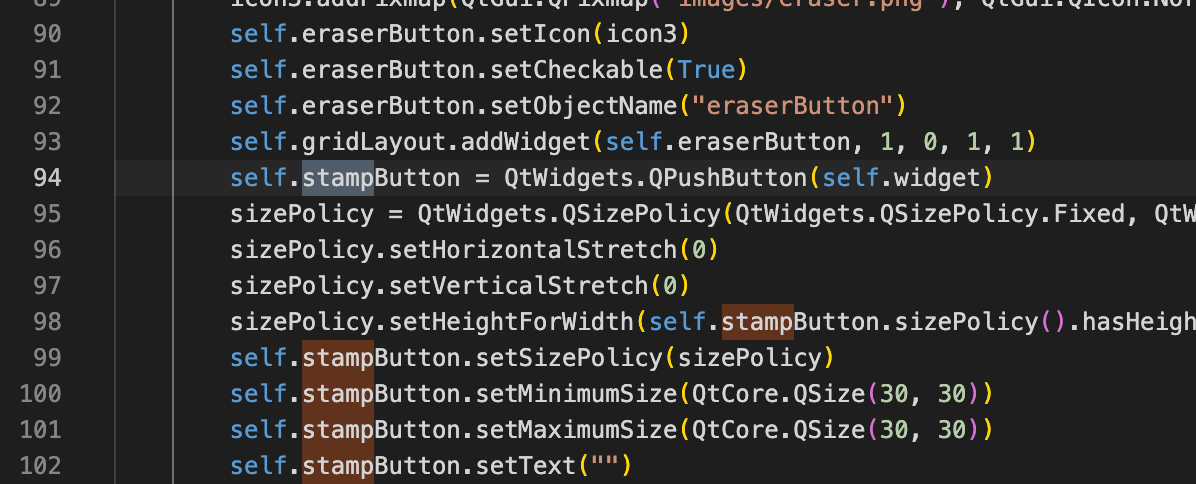


所有按键图标存放在./image目录下，所有蛋糕图标存放在./stamp目录下

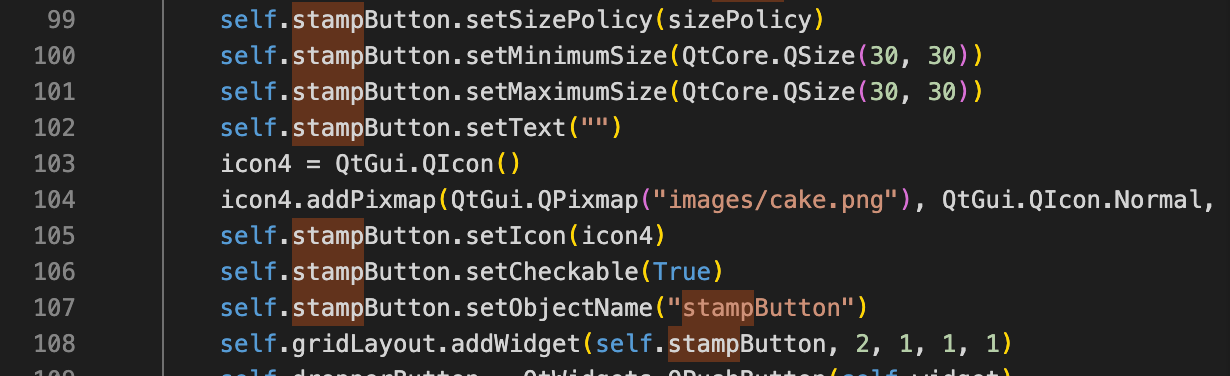


在main\_windows.py中

使用QtWidgets定义蛋糕的按钮stampButton



使用cake.png作为stampButton的图标



在paint.py中，构造切换蛋糕函数，实际上是更换展示图片

def next\_stamp(self):

self.current\_stamp\_n += 1

if self.current\_stamp\_n >= len(STAMPS):

self.current\_stamp\_n = 0

pixmap = QPixmap(STAMPS[self.current\_stamp\_n])

self.stampnextButton.setIcon(QIcon(pixmap))

self.canvas.current\_stamp = pixmap

其中STAMPS列表定义如下，内容为所有蛋糕名

STAMP\_DIR = './stamps'

STAMPS = [os.path.join(STAMP\_DIR, f) for f in os.listdir(STAMP\_DIR)]

## 算法实现流程

**landmark部分：使用dlib**

def mouth\_aspect\_ratio(mouth):

    # compute the euclidean distances between the two sets of

    # vertical mouth landmarks (x, y)-coordinates

    A = dist.euclidean(mouth[2], mouth[10])  # 51, 59

    B = dist.euclidean(mouth[4], mouth[8])  # 53, 57

    # compute the euclidean distance between the horizontal

    # mouth landmark (x, y)-coordinates

    C = dist.euclidean(mouth[0], mouth[6])  # 49, 55

    # compute the mouth aspect ratio

    mar = (A + B) / (2.0 \* C)

    # return the mouth aspect ratio

    return mar

提取嘴部特征

detector = dlib.get\_frontal\_face\_detector()

predictor = dlib.shape\_predictor("./shape\_predictor\_68\_face\_landmarks.dat")

获取脸部位置检测器并载入模型

shape = predictor(gray, rects[0])

            shape = face\_utils.shape\_to\_np(shape)

输入摄像头图像与内部人脸检测的边界框的位置信息

并化为np数组

张口检测

检测原理：类似眨眼检测，计算Mouth [Aspect](https://so.csdn.net/so/search?q=Aspect&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/u012693479/article/details/_blank) Ratio,MAR.当MAR大于设定的阈值时，认为张开了嘴巴。

mouthMAR = mouth\_aspect\_ratio(mouth)

            mar = mouthMAR

            # compute the convex hull for the mouth, then visualize the mouth

            mouthHull = cv2.convexHull(mouth)

计算嘴巴的凸包，然后可视化嘴巴特征点，计算嘴长宽比例

 if mar > MOUTH\_AR\_THRESH:

                cv2.putText(frame, "Mouse state changed!", (800, 20),

                            cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, (0, 0, 255), 2)

                if Mouse\_flag:

                    Mouse\_flag = False

                    pyautogui.sleep(0.5)

                    pyautogui.mouseUp()

                else:

                    Mouse\_flag = True

                    pyautogui.sleep(0.5)

                    pyautogui.mouseDown()

判断嘴的长宽比例是否达到阈值以判定是否长按鼠标左键

**eyes keypoint部分：使用[MediaPipe Face Mesh](https://google.github.io/mediapipe/solutions/face_mesh.html)**

Mediapipe获取人脸标记点位：

face\_mesh = mp.solutions.face\_mesh.FaceMesh(refine\_landmarks=True)

face\_mesh为一种对象的实例，后续用以处理图线与人脸点位

使用pyautogui获取屏幕尺寸：

screen\_w, screen\_h = pyautogui.size()

获取人脸标记点位与视频帧图像尺寸

while True:  
 \_, frame = cam.read()  
 frame = cv2.flip(frame, 1)  
 output = face\_mesh.process(frame)  
 landmark\_points = output.multi\_face\_landmarks  
 frame\_h, frame\_w, \_ = frame.shape

face\_mesh对上述图像进行处理后得到图像中人体的各个关键标记点landmark\_points

frame\_h, frame\_w则是图像的高与宽，方便之后图像中坐标与屏幕坐标的变换

1. 根据视频中人脸的移动控制鼠标移动

为了算法简洁与便于实行，处理数据仅为右眼虹膜的上、下、左、右四点

if landmark\_points:  
 landmarks = landmark\_points[0].landmark  
 for id, landmark in enumerate(landmarks[474:478]):  
 x = int(landmark.x \* frame\_w)  
 y = int(landmark.y \* frame\_h)  
 cv2.circle(frame, (x, y), 3, (0, 255, 0))  
 if id == 1:  
 screen\_x = screen\_w \* landmark.x  
 screen\_y = screen\_h \* landmark.y  
 pyautogui.moveTo(screen\_x, screen\_y)

捕捉到landmark\_points，则：

获取对应人脸各个关键点的标记点位landmarks，选择其中[474:478]即为右眼虹膜的四个关键点

使用cv2.circle为各个关键点画圈以在视频中显现

使用第474号标记点对应屏幕中的坐标作为下一时刻鼠标移动至的坐标

1. 眨眼控制鼠标点击的实现

首先判断是否检测到瞳孔

if left\_pupil != None and right\_pupil != None:

close\_count = 0

cv2.putText(frame, "Detect pupils, you can click", (15, 50), cv2.FONT\_HERSHEY\_DUPLEX, 1.6, (147, 58, 31), 1)

再调用gaze实例中的blinking方法检测是否眨眼，是则call鼠标单击，0.5s作为判定时间

if gaze.is\_blinking():

text = "Blinking"

click\_flag = click\_flag ^ 1

pyautogui.click()

pyautogui.sleep(0.5)

其中is\_blinking定义如下

def is\_blinking(self):

"""Returns true if the user closes his eyes"""

if self.pupils\_located:

blinking\_ratio = (self.eye\_left.blinking + self.eye\_right.blinking) / 2

return blinking\_ratio > 3.8

其中blinking的基本思想是：通过计算上下眼睑的距离判断是否眨眼，低于某个阈值则认定为眨眼

self.blinking = self.\_blinking\_ratio(landmarks, points)

需要调用函数blinking\_ratio完成计算

def \_blinking\_ratio(self, landmarks, points):

"""Calculates a ratio that can indicate whether an eye is closed or not.

It's the division of the width of the eye, by its height.

Arguments:

landmarks (dlib.full\_object\_detection): Facial landmarks for the face region

points (list): Points of an eye (from the 68 Multi-PIE landmarks)

Returns:

The computed ratio

"""

left = (landmarks.part(points[0]).x, landmarks.part(points[0]).y)

right = (landmarks.part(points[3]).x, landmarks.part(points[3]).y)

top = self.\_middle\_point(landmarks.part(points[1]), landmarks.part(points[2]))

bottom = self.\_middle\_point(landmarks.part(points[5]), landmarks.part(points[4]))

eye\_width = math.hypot((left[0] - right[0]), (left[1] - right[1]))

eye\_height = math.hypot((top[0] - bottom[0]), (top[1] - bottom[1]))

try:

ratio = eye\_width / eye\_height

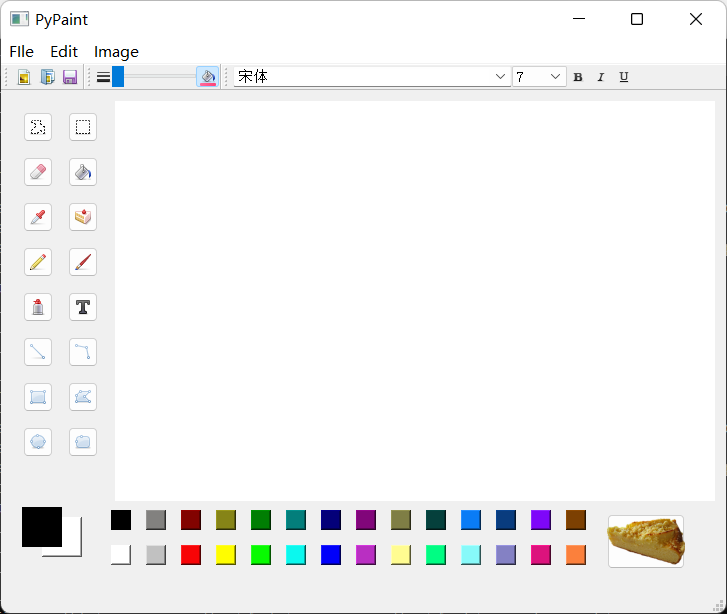
except ZeroDivisionError:

ratio = None

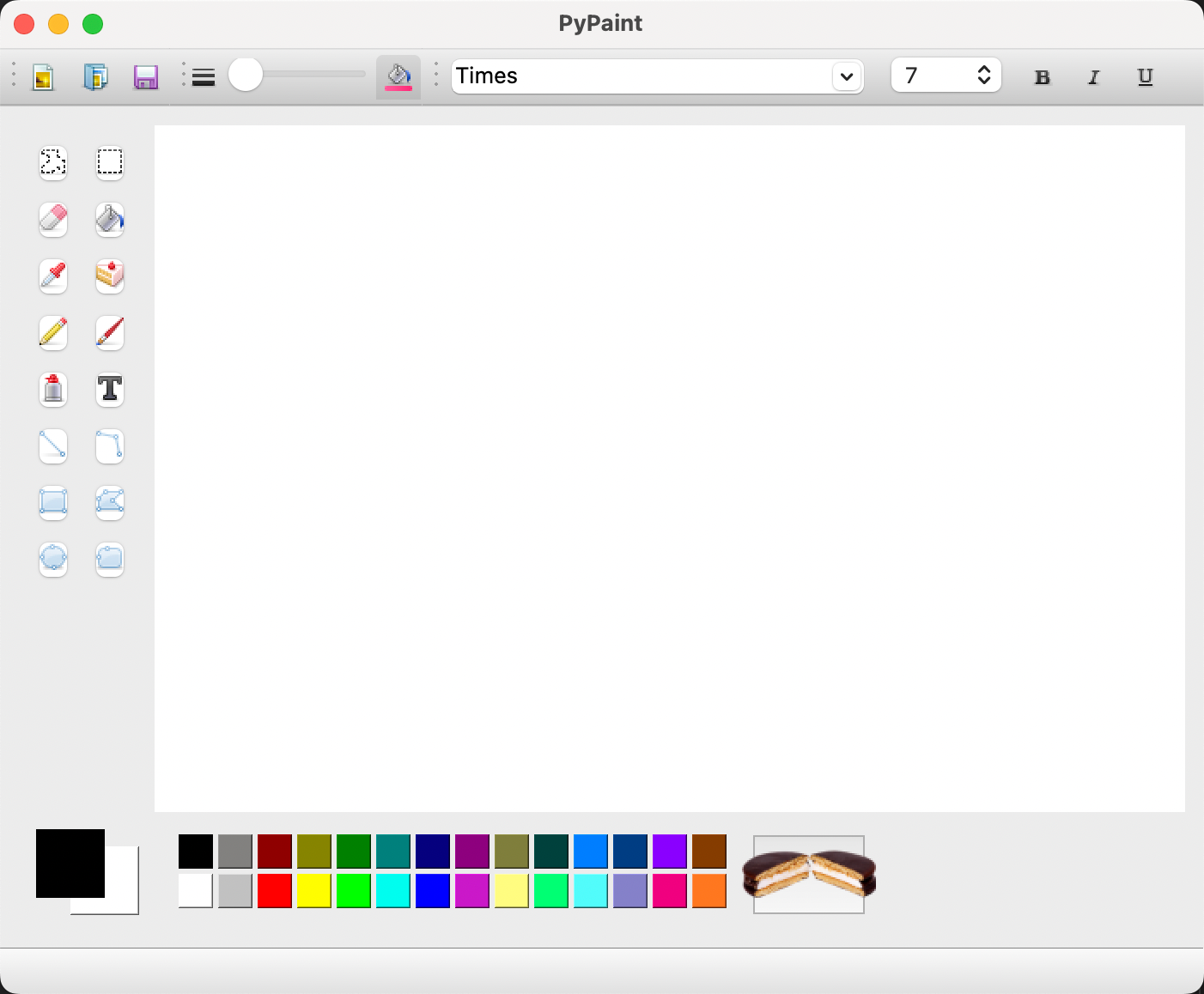
return ratio

## 算法结果分析&界面操作方法

**4.1 画图界面操作方法&分析**



Windows10下PyPaint界面



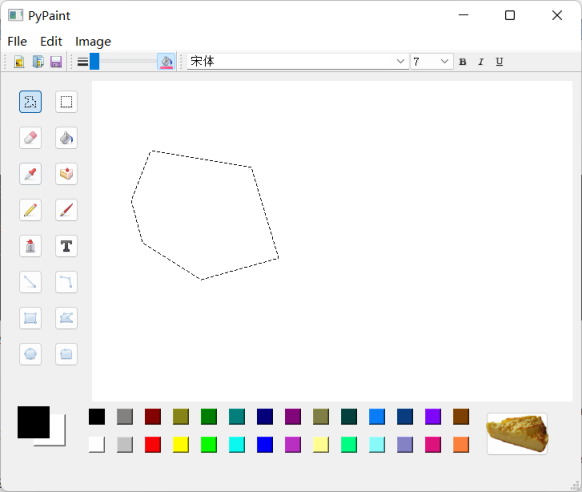
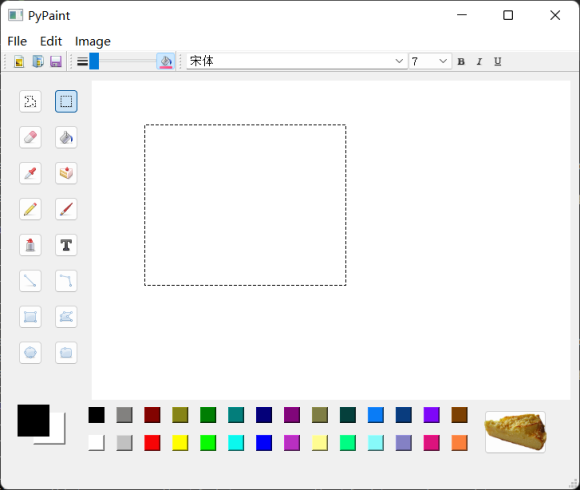
MacOS下PyPaint界面

界面主要实现的功能如下：

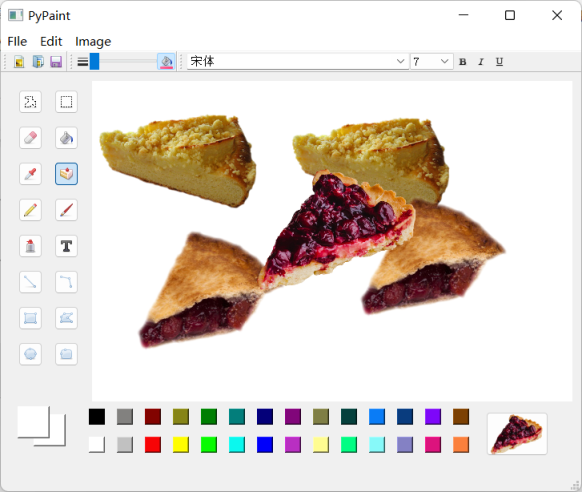
MODES = [  
 'selectpoly', 'selectrect',  
 'eraser', 'fill',  
 'dropper', 'stamp',  
 'pen', 'brush',  
 'spray', 'text',  
 'line', 'polyline',  
 'rect', 'polygon',  
 'ellipse', 'roundrect'  
]

即：矩形&多边形选择框、橡皮擦、填充器、滴管（用于测定某块像素的颜色）、贴纸（各种类型的蛋糕）、钢笔&铅笔、扩散器（该功能尚未完善）、文本编辑、图形绘制（包括各种线段、形状）

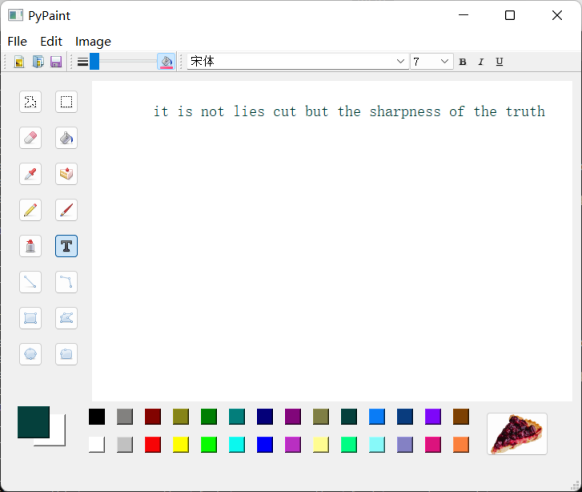
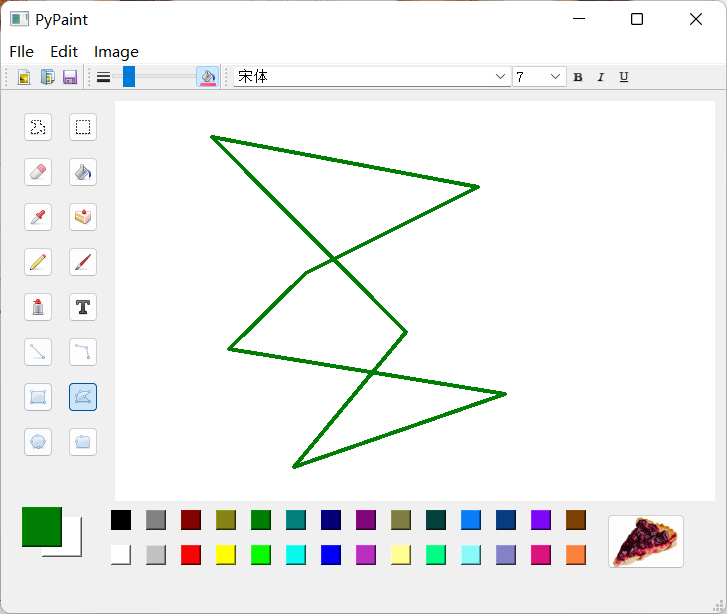
下面是部分功能展示：

Selectrect & Selectpoly

Stamp & Dropper

Text & Polyline

**4.2 眼动画图操作方法&分析**

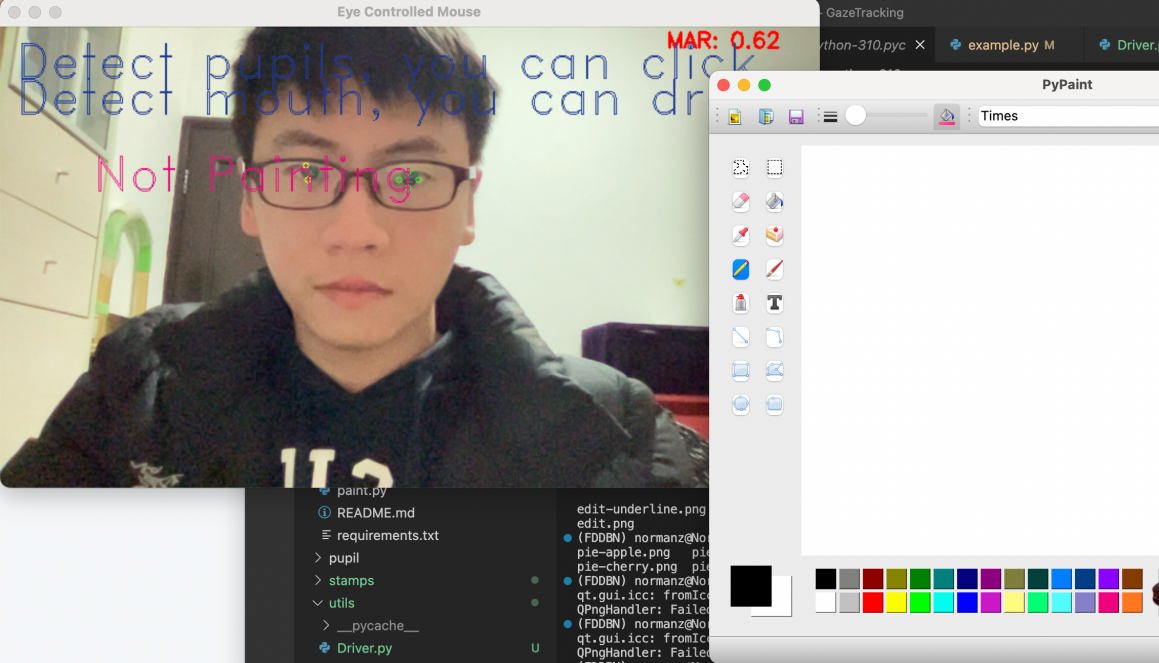
run `python demo.py`



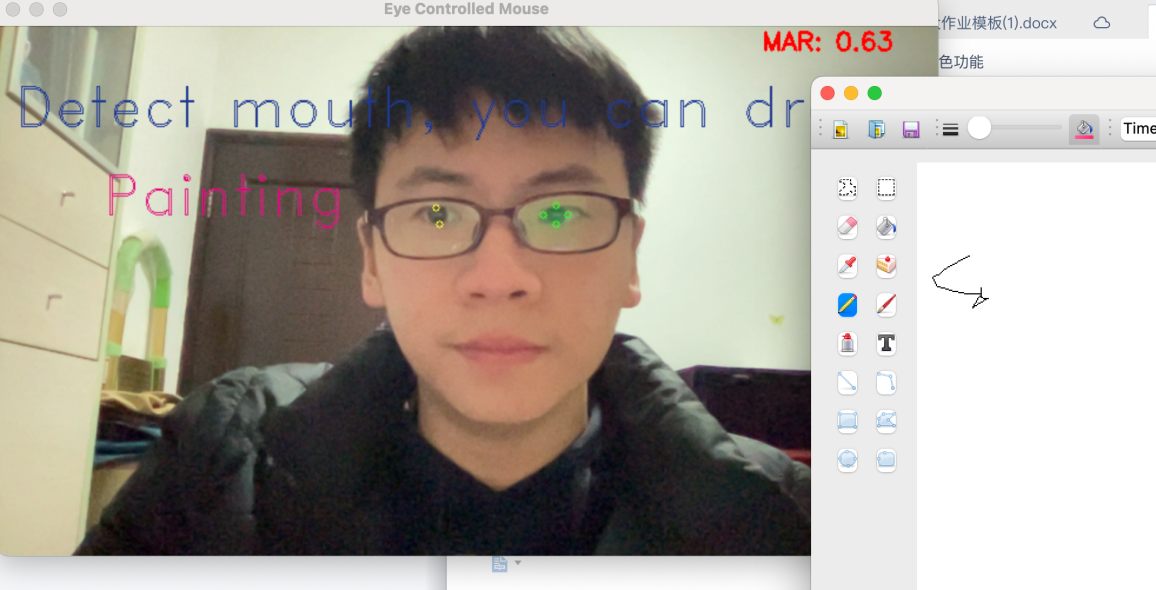
当显示“Detect pupils, you can click”时，可以通过眨眼控制鼠标单击

当显示“Detect mouth, you can draw”时，可以通过张嘴控制绘画落笔/抬笔

第三行粉色字体提示绘图状态“Painting”or “Not Painting”



当前两行字体都出现时，标志着可以正常眼动绘图，通过眼睛移动控制鼠标移动，眨眼控制鼠标单击，张嘴控制画笔落笔和抬笔



如图所示，未检测到瞳孔时，也可以通过眼球移动来控制鼠标移动完成绘画，这是考虑到基于mediapipe的瞳孔检测对光线要求较高，所以使用基于dlib获取的眼部landmarks来移动鼠标，克服弱光条件下无法绘画的问题

References:

1. <https://pyimagesearch.com/2019/12/16/training-a-custom-dlib-shape-predictor/>
2. <https://medium.datadriveninvestor.com/training-alternative-dlib-shape-predictor-models-using-python-d1d8f8bd9f5c>
3. <https://www.youtube.com/watch?v=U3jx43uuAGg>
4. <https://realpython.com/python-pyqt-gui-calculator/#creating-a-calculator-app-with-python-and-pyqt>
5. <https://google.github.io/mediapipe/solutions/face_detection.html>