

《程序设计》课程项目报告



FaceTube

团队：NameError

成员：郭沛承、王书扬、杨晨昊

指导老师：鲍扬

目录

Part1：项目简介	P03
Part2：队员分块报告及代码细节	P04
Chapter A.郭沛承's part	P04
Chapter B.王书扬's part	P09
Chapter C.杨晨昊's part	P13

Part1：项目简介

团队名称	NameError	项目名称	FaceTube
------	-----------	------	----------

项目目标

在电脑上实现给用户的面部加动态贴纸并能保存照片、录像的功能。用户打开电脑摄像头、运行程序后，在对话框里选择往脸上加预置的贴纸（各种款式的帽子、眼镜胡子等），选择好后，程序在用户面部相应位置加上选择的动态（会随着用户的面部移动而动）贴纸；此外，用户也可以自行导入贴纸并调整贴纸的位置。用户可以选择拍摄照片或录像进行保存。

项目依赖的标准库及三方库

1.opencv 面部识别+图像合成

```
from cv2 import cv2
```

2.numpy opencv 辅助计算

```
import numpy as np
```

3.PIL 图片格式转换

```
from PIL import Image, ImageTk
```

4.tkinter 图形化界面

```
import tkinter as tk
from tkinter import dialog, filedialog, ttk
```

5.os 保存、读取图片

```
import os
```

GitHub 地址

<https://github.com/gps4869/FaceTube>

Part2: 队员分块报告及代码细节

Chapter A.郭沛承's part

我负责整个程序的核心运行部分：

- 1.摄像头读取函数 videoCapture()
- 2.脸部识别函数 faceRecognition()
- 3.图片格式转化并输出函数 outputAddedImg()
- 4.主循环程序 videoLoop()
- 5.保存图片函数及读取图片函数的对话框部分 save_file() open_file()
- 6.定义 Sticker 类，并创建初始化函数__init__()、创建按钮函数 createButton()、多选框功能 addToImg()、定位函数 getStickerPosition()、添加贴纸进图像函数中除调整大小以外的部分 addTwoImgs() (Sticker 部分方法在被 StickerFamily 调用时有所修改)
- 7.NoteLabel 的提示功能

下面我将按照以上顺序介绍代码细节：

1.摄像头读取函数 videoCapture()

```
def videoCapture():
    #读取摄像头捕捉到的图片
    flag, img1_BGR = camera.read() #第二个参数是一帧一帧读取的图片
    if flag:
        img1_RGBA = cv2.cvtColor(img1_BGR,
                                   cv2.COLOR_BGR2RGBA)
        #把读入的 BGR 格式转换成 RGB 格式
        return flag, img1_RGBA
    else:
        NoteLabel.config(text='Please check your camera!')
        return False, None
```

camera.read()返回两个值，第一个值为读取成功与否，若读取失败表明摄像头存在硬件\权限等问题，不在本程序考虑范围内；若读取成功将读取的 BGR 格式图片转化为 RGBA 格式图片留待后用。

2.脸部识别函数 faceRecognition()

```
faceCascade = cv2.CascadeClassifier(path_of_facerecognition_package)
```

path_of_facerecognition_package 是 opencv 面部识别包的路径。它被作为一个参数传递进这个函数，您可以在主循环函数 videoLoop()中修改此路径（第 56 行）。

```
path = r'D:\Anaconda\Lib\site-packages\cv2\data\haarcascade_frontalface_default.xml'
```

```
faces = faceCascade.detectMultiScale(img1_RGBA,
                                       scaleFactor=1.2,
                                       minNeighbors=5,
                                       minSize=(32, 32))
```

这一步进行了面部识别，如果识别成功则 faces 返回值为 numpy.ndarray 格式，实际上相当于一个嵌套列表，形如[[x,y,height,width]]，其中 (x, y) 为脸部左上角的点的坐标，height

与 width 表示脸部的宽度且 height 恒等于 width（也就是说它识别的是一个正方形）；如果识别失败会返回一个空的元组。

```
elif abs(face_param[0] - faces[0][0]) > 15 or abs(face_param[1] - faces[0][1]) > 15:
    face_param = list(faces[0][:3])
return face_param
```

opencv 自带的面部识别不是很稳定，不做处理时会不停地在小范围内抖动。此处设计了一个减少震动的机制。face_param 即是上一次识别的 faces 中的三个重要参数，设定若新一次识别得到的 x 值和 y 值与上一次识别的 x, y 值分别相差小于等于 15 个像素点时，仍返回上次的脸部参数，这样可以有效减少震动。但这样的设计会带来另一个问题，如果用户缓慢移动，那么可能出现脸部与贴图相分离的情况，但我认为一般不会出现这样的情况。

在这一步中如果面部识别失败，会在 NoteLabel 中显示相关提示，画面则会在那一瞬间卡住等待 40ms 以后的下次识别。

3. 图片格式转化并输出函数 outputAddedImg()

```
def outputAddedImg(img1_RGBA, label):
    #把图片转化成 tkinter 能输出的格式
    current_image = Image.fromarray(img1_RGBA) #转换为 pillow 图像
    imgtk = ImageTk.PhotoImage(image=current_image)
    #转换为与 tkinter 兼容的照片图像
    label.imgtk = imgtk
    label.config(image=imgtk)
    NoteLabel.config(text='The program runs correctly...')
```

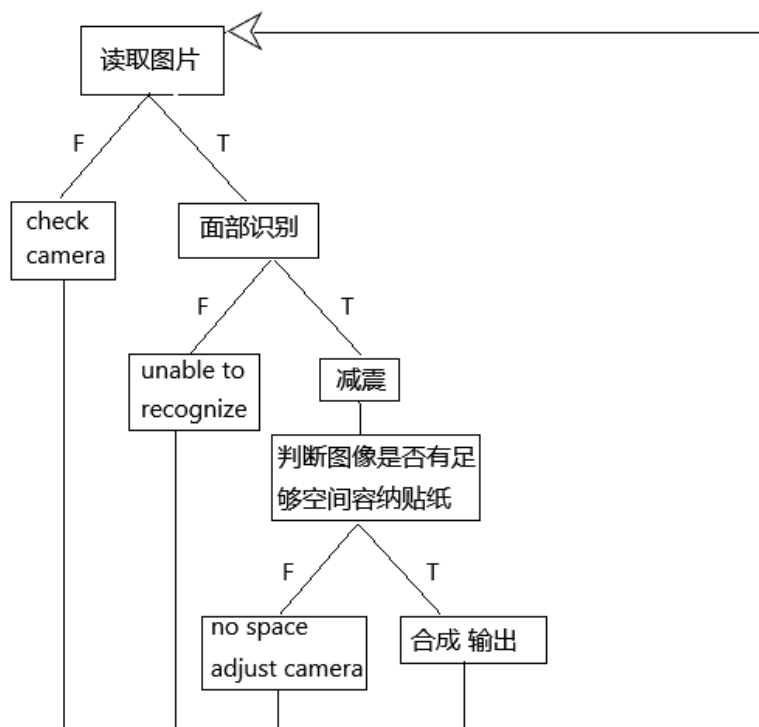
利用 pillow 库将 img1_RGBA 的图像转化为 tkinter 的 Label 控件可以接受的 PIL.ImageTk.PhotoImage 图片格式并输出。同时 NoteLabel 提示程序正常运行。

4. 主循环程序 videoLoop()

```
def videoLoop():
    global img1_RGBA, face_param
    flag, img1_RGBA = videoCapture()
    path = r'D:\Anaconda\Lib\site-packages\cv2\data\haarcascade_frontalface_default.xml'
    if flag:
        face_param = faceRecognition(img1_RGBA, path, face_param)
        if face_param:
            spaceFlags = []
            for sticker in stickers:
                spaceFlag, img1_RGBA = sticker.addTwoImgs(
                    img1_RGBA, face_param)
                spaceFlags.append(spaceFlag)
            if not False in spaceFlags:
                outputAddedImg(img1_RGBA, ImgOutput)
    HatFamily.check()
```

```
root.after(40, videoLoop) #每 40 毫秒循环一次这个主程序
```

主循环程序的逻辑分支如下图所示：



```
global img1_RGBA, face_param
```

为了保证程序的连续性以及减震功能的实现，需要向 videoLoop 函数传入上一次的图像 img1_RGBA 和 face_param，但由于 root.after() 函数不支持传递参数，所以在开始对 img1_RGBA 和 face_param 作全局定义。

```
spaceFlags = []
```

spaceFlags 是一个在函数外部全局定义的列表，列表中的值即为要加入图像的 Sticker。

```
root.after(40, videoLoop)
```

主循环程序每 40ms 刷新一次。

5.保存图片函数及读取图片函数的对话框部分 save_file() open_file()

```
choice = dialog.Dialog(
    None, {
        'title': 'File Modified',
        'text': '注意路径中不能含有中文字符! ',
        'bitmap': 'warning',
```

```

        'default': 0,
        'strings': ('OK', 'Cancel')
    })

```

利用tkinter的dialog.Dialog函数弹出对话框提示不能有中文路径。strings形参为两个选项，若选择第一项，则choice.num值等于0，选第二项则choice.num值等于1。

```

try:
    print('保存文件: ', file_path)
    cv2.imwrite(filename=file_path, img=img1_BGR)
    dialog.Dialog(
        None, {
            'title': 'File Modified',
            'text': '保存完成',
            'bitmap': 'warning',
            'default': 0,
            'strings': ('OK', 'Cancel')
        })
    print('保存完成')
except:
    print('Close by user or use the wrong path.')

```

这是一处异常处理。如果用户输入的路径有错误或者用户手动关闭会避免报错。

6.Sticker 类

```

def __init__(self, name, path, faceSpot, stickerSpot):
    self.name = name
    self.path = path
    self.faceSpot = faceSpot
    self.stickerSpot = stickerSpot

```

初始化函数传递的参数如图。faceSpot和stickerSpot是两个用于定位的列表。定位方法：faceSpot是脸部用于定位的点，输入一个list，含有两个值。

定义脸部的左上角为脸部原点。

第一个值是脸部原点到该点横向距离占脸部总宽度的比例。

第二个值是脸部原点到该点纵向距离占脸部总高度的比例。

同理，stickerSpot是用于在贴图上定位的列表。

两个点重合即可将贴图定位到图片上。

```

def createButton(self):
    self.v = tk.IntVar()
    self.button = tk.Checkbutton(root,
                                  text=self.name,
                                  variable=self.v,
                                  command=self.addToImg)
    self.button.grid(sticky='W' + 'E' + 'N' + 'S')

```

创建多选框，v为多选框的值，勾选后会自动变成1，取消勾选会变成0。

```
def addToImg(self):
    if self.v.get() == 1: #如果相应的多选框被选中
        stickers.append(self)
    else:
        stickers.remove(self)
```

被选中后把对应的贴纸加到列表 stickers 中，反之从中移除。

```
def addTwoImgs(self, img1_RGBA, face_param):
    .....
    try:
        self.rows, self.cols = self.img.shape[:2]
    except:
        NoteLabel.config(text='Fail in loading sticker!')
```

若读取贴纸失败，在读取时不会报错，但在这一步读取相关参数时会报错。对此问题进行异常处理。

```
roi = img1_RGBA[self.y1:self.y2, self.x1:self.x2]
sticker_gray = cv2.cvtColor(self.img, cv2.COLOR_BGR2GRAY)
ret, mask = cv2.threshold(sticker_gray, 10, 255, cv2.THRESH_BINARY)
mask_inv = cv2.bitwise_not(mask)
img1_bg = cv2.bitwise_and(roi, roi, mask=mask_inv)
self.img = cv2.cvtColor(self.img, cv2.COLOR_BGR2RGBA)
dst = cv2.add(img1_bg, self.img)
img1_RGBA[self.y1:self.y2, self.x1:self.x2] = dst
```

这一段通过制作掩膜先把原图中要被贴图覆盖的部分剪掉，然后再把贴图和原图合并，这样就不会出现颜色重叠的问题。

7. NoteLabel 的提示功能

主程序内创建：

```
NoteLabel = tk.Label(root)
NoteLabel.grid(row=0, column=0)
```

几处提示错误：

摄像头错误：

```
NoteLabel.config(text='Please check your camera!')
```

人脸识别错误：

```
NoteLabel.config(
    text=
        "Unable to recognize face! Please adjust camera angle or fa
ce to camera!"
)
```

贴纸加载错误：

```
NoteLabel.config(text='Fail in loading sticker!')
```

贴纸空间错误：


```
NoteLabel.config(text="No enough space for stickers!")
```

正确运行时提示:

```
NoteLabel.config(text='The program runs correctly...')
```

Chapter B.王书扬's part

本人王书扬负责本项目的两部分内容:一是让用户可以用鼠标自行拖动改变导入的自定义贴纸与脸部的相对位置;二是美化交互界面,在原窗口中显示内置贴纸的大类,点击后弹出子窗口显示具体款式的名称和样式,让用户选择。

一、自定义贴纸的移动

由于要将可以手动移动自定义贴纸的信息传递给用户,因此考虑用 Tkinter 的 Label 控件传递文字信息,并用 bind 函数将鼠标事件与此文本框绑定。bind 函数的调用规则为:窗体对象.bind(事件类型,回调函数)。此处的事件类型为鼠标拖动事件,回调函数需要实现将鼠标位置的变化传递为自定义贴纸位置的变化。

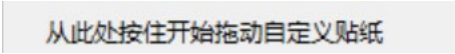
代码如下(加了下划线的为本人添加的代码):

```
def open_file():
    #读取读者自定义的贴纸
    choice = dialog.Dialog(
        None, {
            'title': 'File Modified',
            'text': '图片路径中不能含有中文字符! ',
            'bitmap': 'warning',
            'default': 0,
            'strings': ('OK', 'Cancle')
        })
    if choice == 0:
        global selfcustomizeSticker
        selfcustomizesticker_path = tk.filedialog.askopenfilename(
            title=u'打开贴图')
        selfcustomizeSticker = Sticker(name='selfcustomizeSticker',
                                         path=selfcustomizesticker_path,
                                         faceSpot=[0, 0],
                                         stickerSpot=[0, 0])
        selfcustomizeSticker.createButton()
        label = tk.Label(root, text="从此处按住开始拖动自定义贴纸")
        label.grid()
        label.bind("<B1-Motion>", moveimg)

def moveimg(event):
```

```
height, width = selfcustomizeSticker.img.shape[:2]
selfcustomizeSticker.stickerSpot[0] = -event.x / width
selfcustomizeSticker.stickerSpot[1] = -event.y / height
```

由于 selfcustomizeSticker 在 open_file() 内创建，要在此函数外的 moveimg(event) 中修改其 stickerSpot 属性，需要将其设置为全局变量。

在导入自定义贴纸后创建 label 文本框，向用户传递信息“从此处按住开始拖动自定义贴纸”。(效果图：)

用 bind 函数绑定鼠标拖动事件“<B1-Motion>”——拖动左键出发事件，和回调函数 moveimg(event)。在 moveimg(event) 中将自定义贴纸 selfcustomizeSticker 用于定位的 stickerSpot 数值更改为正在拖动的鼠标的位置的横纵坐标对应的图片中的比例。因为 class Sticker 中关于 self 横纵坐标的定义如下（在 getStickerPosition(self, face_param) 中）：

```
self.x1 = int(self.img_x - self.stickerSpot_x)
self.y1 = int(self.img_y - self.stickerSpot_y)
```

因此为实现鼠标拖动方向与贴图移动方向的一致，在 event.x, event.y 前均加了负号。

综上，成功实现了用鼠标拖动自定义贴纸的功能。

二、美化交互界面

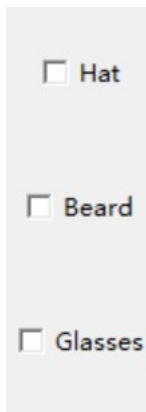
初框架中只是简单地将每一个内置贴纸都以 Checkbutton 控件形式显示贴纸名字，放在原窗口中供用户选择，我们拟实现在原窗口中只显示各组贴纸的 familyname (“Hat” “Glasses” “Beard”)，用户点击相应的大类后再在跳出的新窗口中选择具体款式，并显示款式图样。因此使用 Tkinter 中的 Toplevel 组件来创建顶级窗口，并需要新增 StickerFamily 类来创建原窗口中的大类控件和顶级窗口；利用初框架中的 Sticker.createbutton() 在新窗口中创建各款式复选框，还需要将贴纸图片显示在旁边。

新增 StickerFamily 类，并创建原窗口中的控件的代码如下：

```
class StickerFamily:
    def __init__(self, familyname, contents):
        self.familyname = familyname
        self.contents = contents

    def createfamilyButton(self, row, column):
        self.v = tk.IntVar()
        self.button = tk.Checkbutton(root,
                                     text=self.familyname,
                                     variable=self.v,
                                     command=self.openToplevel) #多选框
        self.button.grid(row=row, column=column, sticky='W' + 'E' + 'N' + 'S')
```

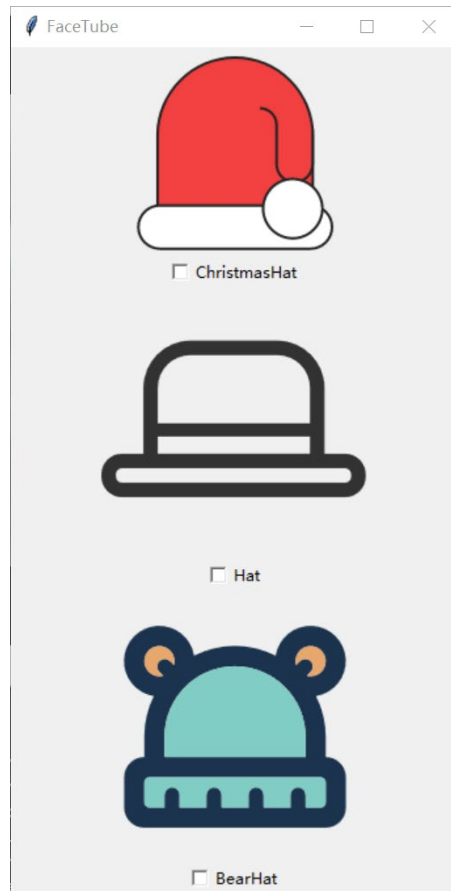
contents 为装有同一类的内置贴纸 (Sticker 类) 的一个列表。点击原窗口中 familyname 的复选框后执行 command：打开 Toplevel。效果图：



在新窗口中建立各款式复选框并显示贴纸图片的代码如下：

```
def openToplevel(self):  
    if self.v.get() == 1: #如果相应的多选框被选中  
        self.toplevel = tk.Toplevel()  
        for content in self.contents:  
            im = Image.open(content.path)  
            content.img1 = ImageTk.PhotoImage(im)  
            content.label = tk.Label(self.toplevel)  
            content.label.grid()  
            content.label.config(image=content.img1)  
            content.v = tk.IntVar()  
            content.button = tk.Checkbutton(self.toplevel,  
                                           text=content.name,  
                                           variable=content.v,  
                                           command=content.addToImg) #单选框  
            content.button.grid(sticky='W' + 'E' + 'N' + 'S')  
        self.toplevel.mainloop()
```

如果相应的多选框被选中，就创建顶级窗口，对同一类中的贴图进行相同操作，即用 `ImageTk.PhotoImage()` 将贴图转换为与 `tkinter` 兼容的照片图像，并在文本框中展示，再利用 `Checkbutton` 控件创建相应的复选框供用户选择是否添加此贴纸。效果图：



为保持原窗口中 `familybutton` 的选中情况与其顶级窗口中的选中情况的一致性，防止顶级窗口中并没有选中任何贴纸但却由于用户已点击原窗口中的该 `familybutton` 而 `familybutton` 复选框一直处于选中状态，添加如下代码：

```
def check(self):
    flag = False
    for Sticker in stickers:
        if Sticker in self.contents:
            flag = True
    if flag == False:
        self.v.set(0)
    else:
        self.v.set(1)
```

并在主循环 `videoloop()` 的最后调用各个 `StickerFamily` 的 `check` 属性，代码如下：

```
HatFamily.check()
BeardFamily.check()
GlassesFamily.check()
root.after(40, videoLoop)
```

同时做了一些因新增 `StickerFamily` 类而带来的小改动。
综上，成功实现了美化交互界面的任务。

Chapter C.杨晨昊's part

1、搜集 Hat、Glasses、Beard 分类的正方形矢量图片，将其做成程序的默认内置贴纸，写出相应函数

具体程序如：

```
Hat = Sticker(name='Hat',  
              path='Hat.png',  
              faceSpot=[0.5, 0],  
              stickerSpot=[0.5, 1])
```

faceSpot 确定面部的连接点，以检测到面部正方形的左上角为原点构建坐标系，数值表示连接点在面部构成的正方形的比例。如[0.5,0]表示了面部中线，头顶直线这两条线的交点。程序中将该点作为贴图与面部的连接点，基于 Hat、Glasses、Beard 在面部的不同位置，faceSpot 的具体数值也不同。Glasses 分类的 faceSpot=[0.5, 0.4]，Beard 分类的 faceSpot=[0.5, 0.8]

stickerSpot 确定图片的连接点，具体原理和 faceSpot 类似，其中 Hat 分类的连接点 stickerSpot= [0.5,1]，其余 stickerSpot=[0.5,0.5]，将图片中心作为连接点。

以比例来确定连接点的方式确保了贴图时，连接点可以根据图片、面部大小、位置的变化而实时变化，具体变现在贴图可以紧贴面部位置。

2、修改了 addTwoImgs、getStickerPosition 函数，让贴图能够随检测到人脸大小的变化而实时变化

```
self.img = cv2.imread(self.path)  
self.img = cv2.resize(self.img, (int(face_param[2] / 1), int(face_param[2] / 1)), interpolation=cv2.INTER_CUBIC)
```

face_param[2]表示了检测到的面部正方形的边长，使用 cv2 的 resize 函数，让贴图能够随检测到人脸大小的变化而实时变化，优化了这一部分功能。

```
self.img_x = self.faceSpot[0] * face_param[2] + face_param[0]  
self.img_y = self.faceSpot[1] * face_param[2] + face_param[1]  
self.stickerSpot_x = self.stickerSpot[0] * face_param[2]  
self.stickerSpot_y = self.stickerSpot[1] * face_param[2]
```

self.faceSpot 和 self.stickerSpot 都以比例来确定面部、图片的连接点，使贴图可以根据面部边长 face_param[2]的变化而实时变化。

3、录制 demo