数字图像处理实验报告

班级： 智能专研 25

姓名： 王耀彬

学号： 2025354100103

目录

[实验项目二：人体部件检测系统 1](#_Toc217647464)

[**前言** 1](#_Toc217647465)

[**1 基础人体部件检测** 1](#_Toc217647466)

[**1.1 实验原理** 1](#_Toc217647467)

[**1.1.1 Haar特征与级联分类器** 1](#_Toc217647468)

[**1.2 系统架构设计与实现** 1](#_Toc217647469)

[**1.2.1 全局面部检测** 2](#_Toc217647470)

[**1.2.2 局部器官检测（ROI优化）** 3](#_Toc217647471)

[**1.2.3 瞳孔检测（基于形状分析）** 5](#_Toc217647472)

[**2 其他实现** 6](#_Toc217647473)

[**2.1 AR面具系统** 6](#_Toc217647474)

[**2.2 几何约束过滤器** 8](#_Toc217647475)

[**2.3 隐私马赛克盾** 9](#_Toc217647476)

[**3 附录** 11](#_Toc217647477)

[**3.1 Baseline.py** 11](#_Toc217647478)

[**3.2 ar\_mask.py** 19](#_Toc217647479)

[**3.3 privacy\_mosaic.py** 24](#_Toc217647480)

实验项目二：人体部件检测系统

**前言**

本次实验旨在构建一个实时人体部件检测系统，能够从摄像头视频流中检测人脸及其面部器官（眼睛、鼻子、嘴巴、耳朵、瞳孔），并在画面中进行可视化标注。

系统采用两种核心技术。

·瞳孔：基于形状分析的轮廓检测

·人脸、眼睛、鼻子、嘴巴、耳朵：Haar级联分类器

**1 基础人体部件检测**

**1.1 实验原理**

**1.1.1 Haar特征与级联分类器**

Haar特征是Viola-Jones人脸检测算法的基础，其核心思想是：

1.Haar特征：使用简单的矩形特征（边缘特征、线特征、中心环绕特征）描述图像局部区域。

2.积分图加速：通过积分图实现 时间复杂度的特征计算

3.AdaBoost级联：多个弱分类器级联形成强分类器，逐层过滤非人脸区域

其具体算法原理如下，矩形特征描述：

其中 表示像素 的灰度值。

积分图实现 时间复杂度的特征计算：

任意矩形区域的像素和可通过4次查表计算：

**1.2 系统架构设计与实现**

完整代码见附录‘baseline.py’

整个检测系统封装在FaceDetector类中，采用模块化设计：

|  |
| --- |
| class FaceDetector:  def \_\_init\_\_(self):  # 加载所有级联分类器  self.cascades = {}  for name, path in CASCADE\_PATHS.items():  self.cascades[name] = cv2.CascadeClassifier(path) |

系统共加载6个Haar级联分类器文件：

|  |  |
| --- | --- |
| 分类器文件 | 用途 |
| haarcascade\_frontalface\_alt.xml | 正脸 |
| haarcascade\_eye.xml | 眼睛 |
| haarcascade\_mcs\_nose.xm’ | 鼻子 |
| haarcascade\_mcs\_mouth.xml | 嘴巴 |
| haarcascade\_mcs\_leftear.xml | 左耳 |
| haarcascade\_mcs\_rightear.xml | 右耳 |

**1.2.1 全局面部检测**

关键技术实现：

|  |
| --- |
| def detect\_face(self, frame, gray=None):  if gray is None:  gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)  faces = self.cascades["face"].detectMultiScale(  gray,  scaleFactor=1.1, # 图像金字塔缩放比例  minNeighbors=5, # 候选框合并阈值  minSize=(30, 30), # 最小检测尺寸  flags=cv2.CASCADE\_SCALE\_IMAGE  )  return faces |

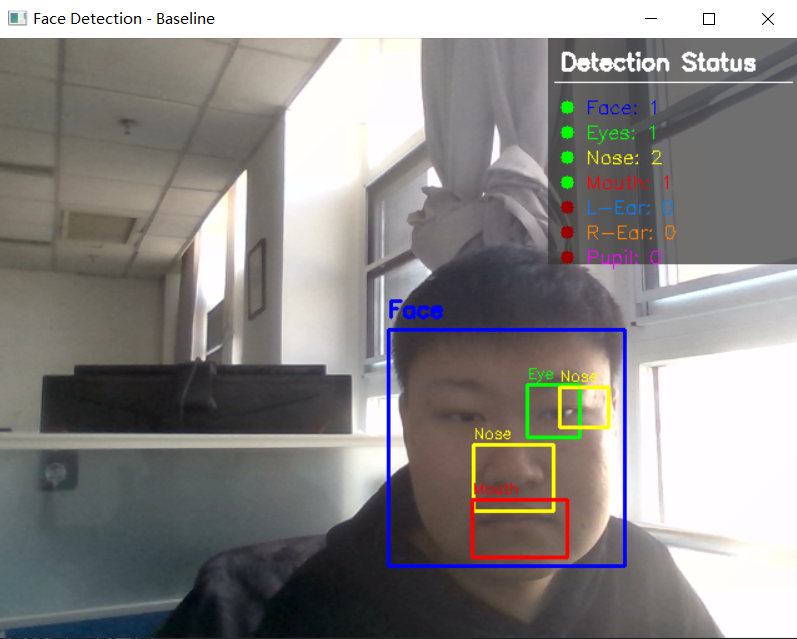
关键参数解析：

·scaleFactor=1.1：每次图像缩小为原来的 ，用于多尺度检测

·minNeighbors=5：只有当某区域被检测到 次时才认定为人脸，有效减少误检

·minSize=(30,30)：过滤小于 像素的候选区域

实现效果如下图：



**1.2.2 局部器官检测（ROI优化）**

直接在全图搜索器官会导致：

·计算开销大：搜索范围过大

·误检率高：背景纹理可能被误检为器官

解决方案：采用ROI（Region of Interest）约束，仅在人脸区域内搜索器官。

特别的，左右耳需要使用不同的分类器分别检测。

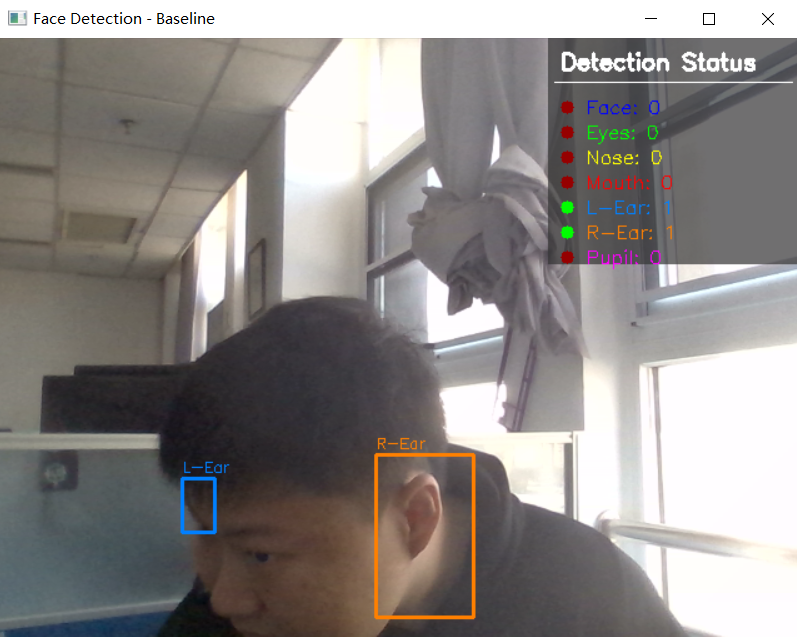
问题发现：在实际测试中发现，当用户呈现侧脸时：

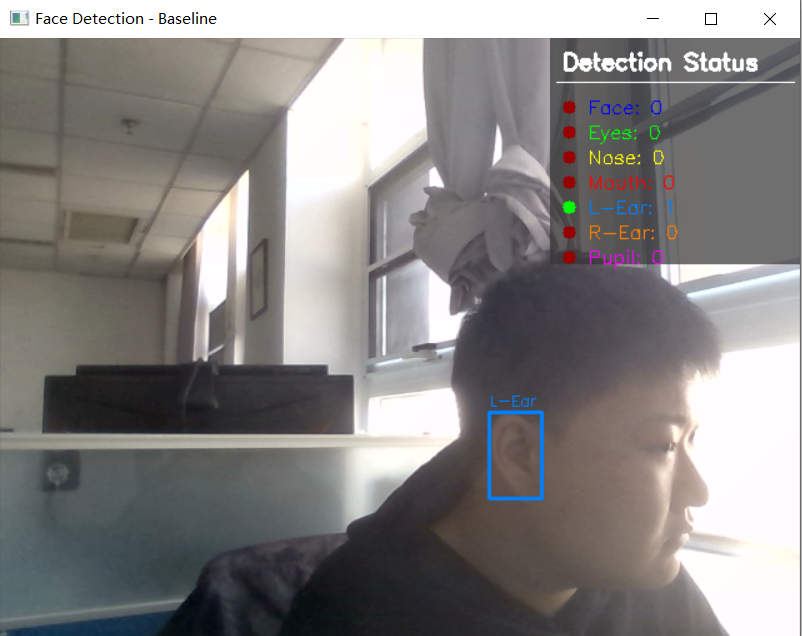
1. 正脸分类器无法检测到人脸（Face: 0）

2. 耳朵检测依赖于人脸ROI，导致耳朵也无法检测

解决方案：实现全局耳朵检测作为备选方案。

实现效果如下图：





**1.2.3 瞳孔检测（基于形状分析）**

与其他器官不同，瞳孔检测不使用级联分类器，而是采用基于形状分析的自定义算法：

**输入眼睛ROI → 灰度化 → 高斯模糊 → 图像反相 → 二值化 → 轮廓查找 → 几何筛选 → 输出瞳孔位置**

瞳孔是眼睛中最暗的区域，直接检测黑色区域较困难。通过\*\*图像反相\*\*操作：

使得瞳孔从暗区变为高亮区域，便于后续阈值分割。

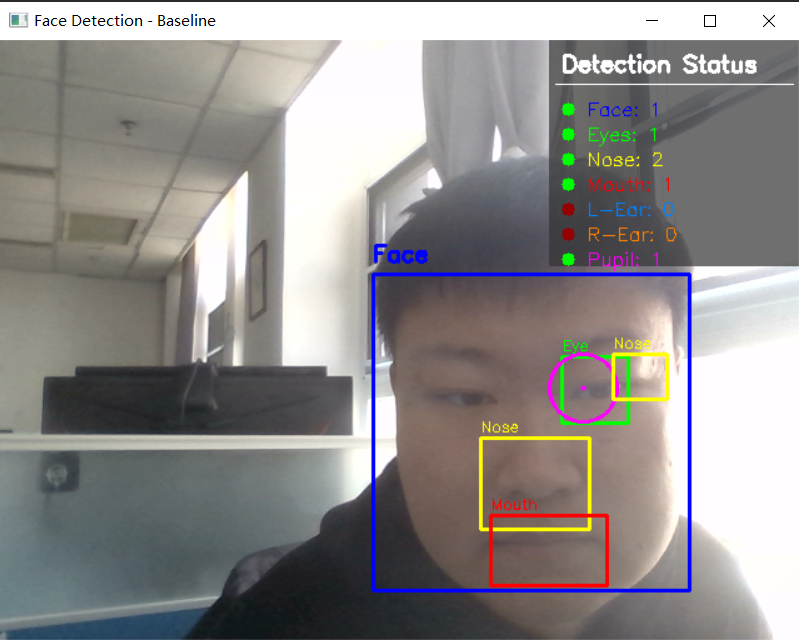
轮廓查找后可能得到多个候选区域，需要通过几何特征筛选出瞳孔：

1. 面积约束：瞳孔面积应占眼睛区域的

2. 圆形度约束：瞳孔接近圆形，其中圆形度计算公式如下，完美圆形的圆形度为1。

统在画面右上角显示实时检测状态面板，包含所有部位的检测计数，红色代表未检测到，绿色的代表检测到。

瞳孔检测见下图。



**2 其他实现**

**2.1 AR面具系统**

实现增强现实（AR）面具佩戴效果，将虚拟面具图像无缝融合到检测到的人脸区域。

核心技术：掩膜（Mask）生成与位运算（Bitwise Operations）

面具融合采用经典的位运算方法，流程如下：

**1. 缩放面具 → 2. 生成二值掩膜 → 3. 逆掩膜抠背景 → 4. 提取面具前景 → 5. 融合**

关键代码实现：

|  |
| --- |
| # 1. 生成二值掩膜（面具非白色区域）  mask\_gray = cv2.cvtColor(mask\_bgr, cv2.COLOR\_BGR2GRAY)  \_, mask\_binary = cv2.threshold(mask\_gray, 240, 255, cv2.THRESH\_BINARY\_INV)  # 2. 生成逆掩膜  mask\_inv = cv2.bitwise\_not(mask\_binary)  # 3. 在人脸ROI中抠出面具形状的空洞  roi\_bg = cv2.bitwise\_and(roi, roi, mask=mask\_inv)  # 4. 提取面具的前景部分  mask\_fg = cv2.bitwise\_and(mask\_bgr, mask\_bgr, mask=mask\_binary)  # 5. 融合背景与前景  result = cv2.add(roi\_bg, mask\_fg) |

实际测试中发现两个问题：

1. 面具太靠下：Haar检测框偏下

2. 覆盖不完整：人脸较大时面具无法覆盖整个脸部

解决方案：添加缩放和偏移参数scale、y\_offset\_ratio

最终效果如下。





**2.2 几何约束过滤器**

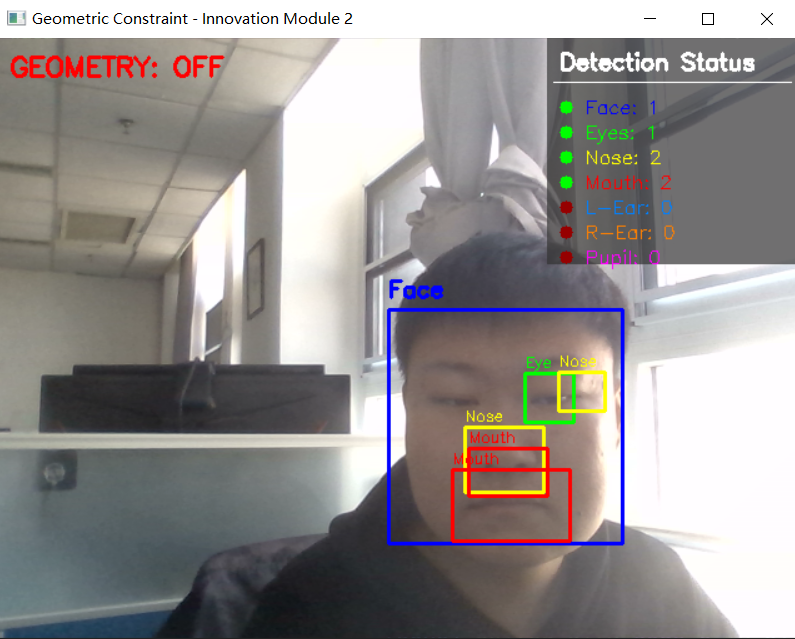
Haar级联分类器存在误检问题，例如将背景纹理误识别为嘴巴。通过引入解剖学常识，对检测结果进行逻辑校验，可有效降低误检率。

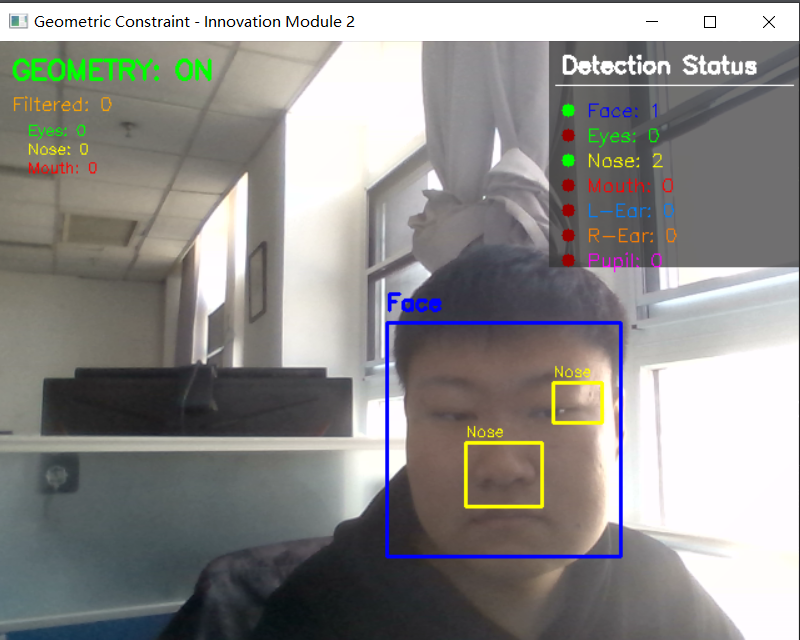
核心思想：器官位置应符合人脸解剖学规律

约束规则：

|  |  |  |
| --- | --- | --- |
| **器官** | **约束条件** | **范围（%）** |
| **眼睛** | 在人脸上部 | 10-55 |
| **鼻子** | 在人脸中部 | 25-75 |
| **嘴巴** | 在人脸下部 | 50-95 |

效果如下：





能看出关闭时，将鼻子识别为嘴的概率很大

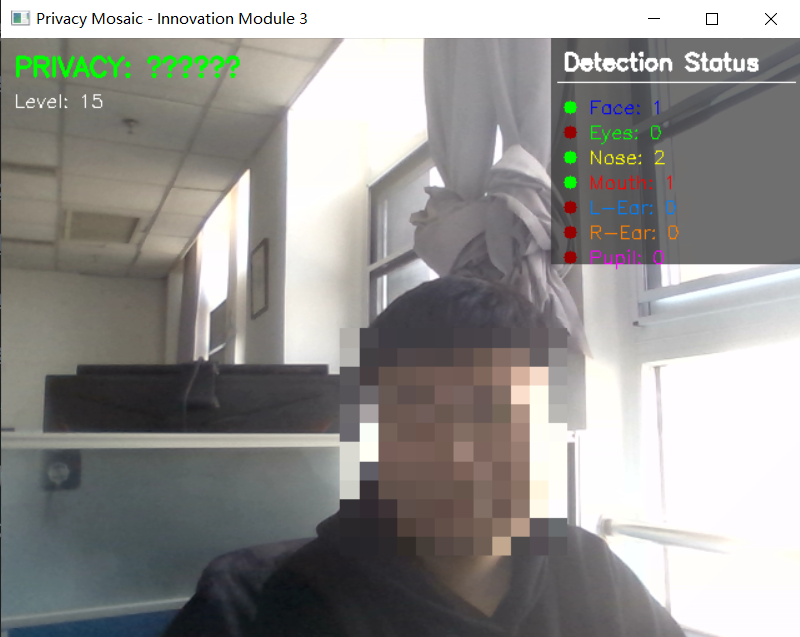
**2.3 隐私马赛克盾**

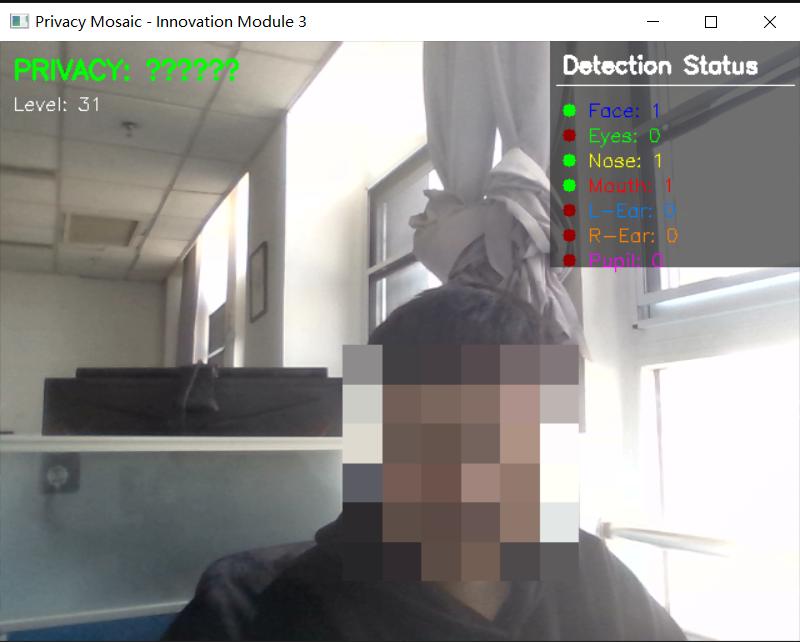
本模块实现实时人脸马赛克效果。在新闻报道、监控回放等场景中，常需要保护个人隐私。

马赛克效果通过降采样+升采样实现：

**原图ROI → 降采样(1/15) → 最近邻插值升采样 → 方块效果**

实现效果如下：





**3 附录**

**3.1 Baseline.py**

|  |
| --- |
| """  数字图像处理作业2：人体部件检测与增强现实系统  基线系统 - 人脸及器官检测模块  """    **import** cv2  **import** numpy as np  **import** os    # ======================== 配置 ========================  CASCADE\_DIR **=** os.path.join(os.path.dirname(\_\_file\_\_), "cascade\_files")    # Haar级联分类器路径  CASCADE\_PATHS **=** {      "face": os.path.join(CASCADE\_DIR, "haarcascade\_frontalface\_alt.xml"),      "eye": os.path.join(CASCADE\_DIR, "haarcascade\_eye.xml"),      "nose": os.path.join(CASCADE\_DIR, "haarcascade\_mcs\_nose.xml"),      "mouth": os.path.join(CASCADE\_DIR, "haarcascade\_mcs\_mouth.xml"),      "left\_ear": os.path.join(CASCADE\_DIR, "haarcascade\_mcs\_leftear.xml"),      "right\_ear": os.path.join(CASCADE\_DIR, "haarcascade\_mcs\_rightear.xml"),  }      **class** FaceDetector:    **def** \_\_init\_\_(self):          """初始化：加载所有级联分类器"""          self.cascades **=** {}  **for** name, path **in** CASCADE\_PATHS.items():  **if** os.path.exists(path):                  self.cascades[name] **=** cv2.CascadeClassifier(path)                  print(f"[INFO] 已加载分类器: {name}")  **else**:                  print(f"[WARNING] 未找到分类器: {path}")    **def** detect\_face(self, frame, gray**=**None):  **if** gray **is** None:              gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)    **if** "face" **not** **in** self.cascades:  **return** []            # 使用detectMultiScale进行多尺度检测          faces **=** self.cascades["face"].detectMultiScale(              gray,              scaleFactor**=**1.1,      # 每次图像尺寸减小的比例              minNeighbors**=**5,       # 每个目标至少被检测到的次数              minSize**=**(30, 30),     # 目标的最小尺寸              flags**=**cv2.CASCADE\_SCALE\_IMAGE          )    **return** faces    **def** detect\_eyes(self, frame, face\_roi, gray\_roi**=**None):  **if** "eye" **not** **in** self.cascades:  **return** []            x, y, w, h **=** face\_roi            # 仅在人脸上半部分搜索眼睛(眼睛通常在脸部上半区域)          upper\_half\_h **=** int(h **\*** 0.6)    **if** gray\_roi **is** None:              gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)              gray\_roi **=** gray[y:y**+**upper\_half\_h, x:x**+**w]  **else**:              gray\_roi **=** gray\_roi[:upper\_half\_h, :]            eyes\_local **=** self.cascades["eye"].detectMultiScale(              gray\_roi,              scaleFactor**=**1.1,              minNeighbors**=**5,              minSize**=**(20, 20)          )            # 转换为原图坐标          eyes **=** []  **for** (ex, ey, ew, eh) **in** eyes\_local:              eyes.append((x **+** ex, y **+** ey, ew, eh))    **return** eyes    **def** detect\_nose(self, frame, face\_roi, gray\_roi**=**None):  **if** "nose" **not** **in** self.cascades:  **return** []            x, y, w, h **=** face\_roi    **if** gray\_roi **is** None:              gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)              gray\_roi **=** gray[y:y**+**h, x:x**+**w]            noses\_local **=** self.cascades["nose"].detectMultiScale(              gray\_roi,              scaleFactor**=**1.1,              minNeighbors**=**5,              minSize**=**(20, 20)          )            # 转换为原图坐标          noses **=** []  **for** (nx, ny, nw, nh) **in** noses\_local:              noses.append((x **+** nx, y **+** ny, nw, nh))    **return** noses    **def** detect\_mouth(self, frame, face\_roi, gray\_roi**=**None):  **if** "mouth" **not** **in** self.cascades:  **return** []            x, y, w, h **=** face\_roi            # 嘴巴通常在脸部下半部分          lower\_start **=** int(h **\*** 0.5)    **if** gray\_roi **is** None:              gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)              gray\_roi **=** gray[y**+**lower\_start:y**+**h, x:x**+**w]  **else**:              gray\_roi **=** gray\_roi[lower\_start:, :]            mouths\_local **=** self.cascades["mouth"].detectMultiScale(              gray\_roi,              scaleFactor**=**1.1,              minNeighbors**=**10,  # 提高阈值减少误检              minSize**=**(25, 15)          )            # 转换为原图坐标          mouths **=** []  **for** (mx, my, mw, mh) **in** mouths\_local:              mouths.append((x **+** mx, y **+** lower\_start **+** my, mw, mh))    **return** mouths    **def** detect\_ears(self, frame, face\_roi, gray\_roi**=**None):          x, y, w, h **=** face\_roi    **if** gray\_roi **is** None:              gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)              gray\_roi **=** gray[y:y**+**h, x:x**+**w]            ears **=** {"left": [], "right": []}            # 左耳(通常在图像右侧)  **if** "left\_ear" **in** self.cascades:              left\_ears **=** self.cascades["left\_ear"].detectMultiScale(                  gray\_roi,                  scaleFactor**=**1.1,                  minNeighbors**=**3,                  minSize**=**(15, 25)              )  **for** (ex, ey, ew, eh) **in** left\_ears:                  ears["left"].append((x **+** ex, y **+** ey, ew, eh))            # 右耳(通常在图像左侧)  **if** "right\_ear" **in** self.cascades:              right\_ears **=** self.cascades["right\_ear"].detectMultiScale(                  gray\_roi,                  scaleFactor**=**1.1,                  minNeighbors**=**3,                  minSize**=**(15, 25)              )  **for** (ex, ey, ew, eh) **in** right\_ears:                  ears["right"].append((x **+** ex, y **+** ey, ew, eh))    **return** ears    **def** detect\_ears\_global(self, frame, gray**=**None):    **if** gray **is** None:              gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)            # 直方图均衡化增强对比度          gray **=** cv2.equalizeHist(gray)            ears **=** {"left": [], "right": []}            # 左耳检测 - 降低阈值提高检测率  **if** "left\_ear" **in** self.cascades:              left\_ears **=** self.cascades["left\_ear"].detectMultiScale(                  gray,                  scaleFactor**=**1.05,    # 更小的缩放比例，更精细的搜索                  minNeighbors**=**1,      # 降低阈值                  minSize**=**(15, 20)     # 更小的最小尺寸              )  **for** (ex, ey, ew, eh) **in** left\_ears:                  ears["left"].append((ex, ey, ew, eh))            # 右耳检测 - 降低阈值提高检测率  **if** "right\_ear" **in** self.cascades:              right\_ears **=** self.cascades["right\_ear"].detectMultiScale(                  gray,                  scaleFactor**=**1.05,                  minNeighbors**=**1,                  minSize**=**(15, 20)              )  **for** (ex, ey, ew, eh) **in** right\_ears:                  ears["right"].append((ex, ey, ew, eh))    **return** ears    **def** detect\_pupil(self, eye\_roi):  **if** eye\_roi **is** None **or** eye\_roi.size **==** 0:  **return** None            # 检查图像尺寸是否太小  **if** eye\_roi.shape[0] < 10 **or** eye\_roi.shape[1] < 10:  **return** None            # 转为灰度  **if** len(eye\_roi.shape) **==** 3:              gray **=** cv2.cvtColor(eye\_roi, cv2.COLOR\_BGR2GRAY)  **else**:              gray **=** eye\_roi.copy()            # 高斯模糊去噪（减小核大小以保留更多细节）          blurred **=** cv2.GaussianBlur(gray, (5, 5), 0)            # 图像反相 - 使黑色瞳孔变为白色高亮          inverted **=** cv2.bitwise\_not(blurred)            # 使用自适应阈值或降低固定阈值          # 方法1：降低阈值（原来200太高）          \_, thresh **=** cv2.threshold(inverted, 120, 255, cv2.THRESH\_BINARY)            # 形态学操作去除噪点          kernel **=** cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (3, 3))          thresh **=** cv2.morphologyEx(thresh, cv2.MORPH\_OPEN, kernel)          thresh **=** cv2.morphologyEx(thresh, cv2.MORPH\_CLOSE, kernel)            # 查找轮廓          contours, \_ **=** cv2.findContours(thresh, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)    **if** **not** contours:  **return** None            # 几何筛选 - 寻找最接近圆形的轮廓          best\_pupil **=** None          best\_score **=** 0            roi\_h, roi\_w **=** gray.shape[:2]          roi\_area **=** roi\_h **\*** roi\_w          roi\_center\_x **=** roi\_w **//** 2          roi\_center\_y **=** roi\_h **//** 2    **for** contour **in** contours:              area **=** cv2.contourArea(contour)                # 放宽面积约束  **if** area < roi\_area **\*** 0.005 **or** area > roi\_area **\*** 0.6:  **continue**                perimeter **=** cv2.arcLength(contour, True)  **if** perimeter **==** 0:  **continue**                # 圆形度计算              circularity **=** 4 **\*** np.pi **\*** area **/** (perimeter **\*\*** 2)                # 放宽圆形度约束（从0.3降到0.2）  **if** circularity < 0.2:  **continue**                # 计算轮廓中心              M **=** cv2.moments(contour)  **if** M["m00"] **==** 0:  **continue**              cx **=** int(M["m10"] **/** M["m00"])              cy **=** int(M["m01"] **/** M["m00"])                # 优先选择靠近眼睛中心的轮廓              dist\_to\_center **=** np.sqrt((cx **-** roi\_center\_x)**\*\***2 **+** (cy **-** roi\_center\_y)**\*\***2)              max\_dist **=** np.sqrt(roi\_center\_x**\*\***2 **+** roi\_center\_y**\*\***2)              center\_score **=** 1 **-** (dist\_to\_center **/** max\_dist) **if** max\_dist > 0 **else** 0                # 综合评分：圆形度 + 中心距离              score **=** circularity **\*** 0.5 **+** center\_score **\*** 0.5    **if** score > best\_score:                  best\_score **=** score                  (cx, cy), radius **=** cv2.minEnclosingCircle(contour)                  best\_pupil **=** (int(cx), int(cy), max(2, int(radius)))    **return** best\_pupil      **def** draw\_detections(frame, faces, detector, draw\_config**=**None):  **if** draw\_config **is** None:          draw\_config **=** {              "face": True,              "eyes": True,              "nose": True,              "mouth": True,              "ears": True,              "pupil": True          }        # 检测统计      stats **=** {          "face": len(faces),          "eyes": 0,          "nose": 0,          "mouth": 0,          "left\_ear": 0,          "right\_ear": 0,          "pupil": 0      }        gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)    **for** (x, y, w, h) **in** faces:          # 绘制人脸框  **if** draw\_config.get("face", True):              cv2.rectangle(frame, (x, y), (x**+**w, y**+**h), (255, 0, 0), 2)              cv2.putText(frame, "Face", (x, y**-**10),                         cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 0, 0), 2)            face\_roi **=** (x, y, w, h)          gray\_face **=** gray[y:y**+**h, x:x**+**w]            # 检测并绘制眼睛  **if** draw\_config.get("eyes", True):              eyes **=** detector.detect\_eyes(frame, face\_roi, gray\_face.copy())              stats["eyes"] **+=** len(eyes)  **for** (ex, ey, ew, eh) **in** eyes:                  cv2.rectangle(frame, (ex, ey), (ex**+**ew, ey**+**eh), (0, 255, 0), 2)                  cv2.putText(frame, "Eye", (ex, ey**-**5),                             cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0, 255, 0), 1)                    # 瞳孔检测  **if** draw\_config.get("pupil", True):                      eye\_img **=** frame[ey:ey**+**eh, ex:ex**+**ew]                      pupil **=** detector.detect\_pupil(eye\_img)  **if** pupil:                          stats["pupil"] **+=** 1                          px, py, pr **=** pupil                          # 转换到原图坐标                          cv2.circle(frame, (ex**+**px, ey**+**py), pr, (255, 0, 255), 2)                          cv2.circle(frame, (ex**+**px, ey**+**py), 2, (255, 0, 255), **-**1)            # 检测并绘制鼻子  **if** draw\_config.get("nose", True):              noses **=** detector.detect\_nose(frame, face\_roi, gray\_face.copy())              stats["nose"] **+=** len(noses)  **for** (nx, ny, nw, nh) **in** noses:                  cv2.rectangle(frame, (nx, ny), (nx**+**nw, ny**+**nh), (0, 255, 255), 2)                  cv2.putText(frame, "Nose", (nx, ny**-**5),                             cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0, 255, 255), 1)            # 检测并绘制嘴巴  **if** draw\_config.get("mouth", True):              mouths **=** detector.detect\_mouth(frame, face\_roi, gray\_face.copy())              stats["mouth"] **+=** len(mouths)  **for** (mx, my, mw, mh) **in** mouths:                  cv2.rectangle(frame, (mx, my), (mx**+**mw, my**+**mh), (0, 0, 255), 2)                  cv2.putText(frame, "Mouth", (mx, my**-**5),                             cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0, 0, 255), 1)            # 检测并绘制耳朵  **if** draw\_config.get("ears", True):              ears **=** detector.detect\_ears(frame, face\_roi, gray\_face.copy())              stats["left\_ear"] **+=** len(ears["left"])              stats["right\_ear"] **+=** len(ears["right"])  **for** ear **in** ears["left"]:                  ex, ey, ew, eh **=** ear                  cv2.rectangle(frame, (ex, ey), (ex**+**ew, ey**+**eh), (255, 128, 0), 2)                  cv2.putText(frame, "L-Ear", (ex, ey**-**5),                             cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (255, 128, 0), 1)  **for** ear **in** ears["right"]:                  ex, ey, ew, eh **=** ear                  cv2.rectangle(frame, (ex, ey), (ex**+**ew, ey**+**eh), (0, 128, 255), 2)                  cv2.putText(frame, "R-Ear", (ex, ey**-**5),                             cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0, 128, 255), 1)        # 当没有检测到人脸时，进行全局耳朵检测（侧脸场景）  **if** len(faces) **==** 0 **and** draw\_config.get("ears", True):          ears **=** detector.detect\_ears\_global(frame, gray)          stats["left\_ear"] **+=** len(ears["left"])          stats["right\_ear"] **+=** len(ears["right"])  **for** ear **in** ears["left"]:              ex, ey, ew, eh **=** ear              cv2.rectangle(frame, (ex, ey), (ex**+**ew, ey**+**eh), (255, 128, 0), 2)              cv2.putText(frame, "L-Ear", (ex, ey**-**5),                         cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (255, 128, 0), 1)  **for** ear **in** ears["right"]:              ex, ey, ew, eh **=** ear              cv2.rectangle(frame, (ex, ey), (ex**+**ew, ey**+**eh), (0, 128, 255), 2)              cv2.putText(frame, "R-Ear", (ex, ey**-**5),                         cv2.FONT\_HERSHEY\_SIMPLEX, 0.4, (0, 128, 255), 1)    **return** frame, stats      **def** draw\_status\_panel(frame, stats):      """      在图像上绘制检测状态面板        Args:          frame: BGR图像          stats: 检测统计字典        Returns:          frame: 绘制后的图像      """      h, w **=** frame.shape[:2]        # 状态面板背景      panel\_h **=** 180      overlay **=** frame.copy()      cv2.rectangle(overlay, (w**-**200, 0), (w, panel\_h), (50, 50, 50), **-**1)      cv2.addWeighted(overlay, 0.7, frame, 0.3, 0, frame)        # 标题      cv2.putText(frame, "Detection Status", (w**-**190, 25),                 cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 255, 255), 2)      cv2.line(frame, (w**-**195, 35), (w**-**5, 35), (255, 255, 255), 1)        # 各部位状态      status\_items **=** [          ("Face", stats["face"], (255, 0, 0)),          ("Eyes", stats["eyes"], (0, 255, 0)),          ("Nose", stats["nose"], (0, 255, 255)),          ("Mouth", stats["mouth"], (0, 0, 255)),          ("L-Ear", stats["left\_ear"], (255, 128, 0)),          ("R-Ear", stats["right\_ear"], (0, 128, 255)),          ("Pupil", stats["pupil"], (255, 0, 255)),      ]        y\_offset **=** 55  **for** name, count, color **in** status\_items:          # 状态指示器 (绿点=检测到, 红点=未检测到)          indicator\_color **=** (0, 255, 0) **if** count > 0 **else** (0, 0, 150)          cv2.circle(frame, (w**-**185, y\_offset), 5, indicator\_color, **-**1)            # 部位名称和数量          text **=** f"{name}: {count}"          cv2.putText(frame, text, (w**-**170, y\_offset **+** 5),                     cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 1)            y\_offset **+=** 20    **return** frame      **def** test\_with\_image(image\_path):      """使用静态图片测试检测功能"""      print(f"\n[INFO] 使用图片测试模式: {image\_path}")        detector **=** FaceDetector()        frame **=** cv2.imread(image\_path)  **if** frame **is** None:          print(f"[ERROR] 无法读取图片: {image\_path}")  **return**        gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)      faces **=** detector.detect\_face(frame, gray)        print(f"[INFO] 检测到 {len(faces)} 张人脸")        frame, stats **=** draw\_detections(frame, faces, detector)      frame **=** draw\_status\_panel(frame, stats)        # 打印检测统计      print("\n[检测统计]")      print(f"  Face: {stats['face']}")      print(f"  Eyes: {stats['eyes']}")      print(f"  Nose: {stats['nose']}")      print(f"  Mouth: {stats['mouth']}")      print(f"  L-Ear: {stats['left\_ear']}")      print(f"  R-Ear: {stats['right\_ear']}")      print(f"  Pupil: {stats['pupil']}")        # 显示结果      cv2.imshow("Face Detection - Image Test", frame)      print("[INFO] 按任意键保存并退出...")      cv2.waitKey(0)        # 保存结果      output\_path **=** image\_path.replace(".", "\_result.")      cv2.imwrite(output\_path, frame)      print(f"[INFO] 结果已保存: {output\_path}")        cv2.destroyAllWindows()      **def** main():      """主函数 - 实时摄像头检测演示"""      print("=" **\*** 50)      print("数字图像处理作业2 - 人体部件检测系统(基线)")      print("=" **\*** 50)        # 初始化检测器      detector **=** FaceDetector()        # 尝试使用DirectShow后端打开摄像头(Windows专用,更稳定)      print("\n[INFO] 尝试使用DirectShow后端打开摄像头...")      cap **=** cv2.VideoCapture(0, cv2.CAP\_DSHOW)    **if** **not** cap.isOpened():          print("[WARNING] DirectShow失败,尝试默认后端...")          cap **=** cv2.VideoCapture(0)    **if** **not** cap.isOpened():          print("[ERROR] 无法打开摄像头!")          print("[TIP] 请检查摄像头是否被其他程序占用")          print("[TIP] 或使用图片测试模式: python face\_detector.py --image <图片路径>")  **return**        # 设置摄像头参数      cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 640)      cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 480)        print("\n[INFO] 摄像头已开启")      print("[INFO] 按 'q' 退出程序")      print("[INFO] 按 's' 保存当前帧截图")        frame\_count **=** 0      fail\_count **=** 0      max\_fails **=** 10    **while** True:          ret, frame **=** cap.read()  **if** **not** ret:              fail\_count **+=** 1              print(f"[WARNING] 读取帧失败 ({fail\_count}/{max\_fails})")  **if** fail\_count >**=** max\_fails:                  print("[ERROR] 连续读取失败,退出程序!")  **break**  **continue**            fail\_count **=** 0  # 重置失败计数            # 水平翻转(镜像效果更自然)          frame **=** cv2.flip(frame, 1)            # 转换为灰度图          gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)            # 检测人脸          faces **=** detector.detect\_face(frame, gray)            # 绘制所有检测结果          frame, stats **=** draw\_detections(frame, faces, detector)            # 绘制状态面板          frame **=** draw\_status\_panel(frame, stats)            # 显示结果          cv2.imshow("Face Detection - Baseline", frame)            # 键盘事件处理          key **=** cv2.waitKey(1) & 0xFF  **if** key **==** ord('q'):  **break**  **elif** key **==** ord('s'):              filename **=** f"screenshot\_{frame\_count}.png"              cv2.imwrite(filename, frame)              print(f"[INFO] 截图已保存: {filename}")              frame\_count **+=** 1        # 释放资源      cap.release()      cv2.destroyAllWindows()      print("\n[INFO] 程序已退出")      **if** \_\_name\_\_ **==** "\_\_main\_\_":  **import** sys    **if** len(sys.argv) > 2 **and** sys.argv[1] **==** "--image":          # 图片测试模式          test\_with\_image(sys.argv[2])  **else**:          # 摄像头模式          main() |

**3.2 ar\_mask.py**

|  |
| --- |
| **import** cv2  **import** numpy as np  **import** os    # 导入基线检测器  **from** baseline **import** FaceDetector, draw\_status\_panel    # ======================== 配置 ========================  MASK\_DIR **=** os.path.join(os.path.dirname(\_\_file\_\_), "Face")      **class** ARMaskApplier:    **def** \_\_init\_\_(self):          """初始化：加载所有可用面具"""          self.masks **=** []          self.mask\_names **=** []          self.current\_mask\_idx **=** 0            # 加载Face目录下所有图片作为面具  **if** os.path.exists(MASK\_DIR):  **for** filename **in** sorted(os.listdir(MASK\_DIR)):  **if** filename.lower().endswith(('.jpg', '.jpeg', '.png', '.bmp')):                      mask\_path **=** os.path.join(MASK\_DIR, filename)                      mask\_img **=** cv2.imread(mask\_path, cv2.IMREAD\_UNCHANGED)  **if** mask\_img **is** **not** None:                          self.masks.append(mask\_img)                          self.mask\_names.append(filename)                          print(f"[INFO] 已加载面具: {filename}")    **if** len(self.masks) **==** 0:              print("[WARNING] 未找到面具图片，请在 Face/ 目录下放置面具图片")    **def** switch\_mask(self):          """切换到下一个面具"""  **if** len(self.masks) > 0:              self.current\_mask\_idx **=** (self.current\_mask\_idx **+** 1) **%** len(self.masks)              print(f"[INFO] 切换面具: {self.mask\_names[self.current\_mask\_idx]}")    **def** get\_current\_mask(self):          """获取当前面具"""  **if** len(self.masks) > 0:  **return** self.masks[self.current\_mask\_idx]  **return** None    **def** apply\_mask(self, frame, face\_roi):          mask\_img **=** self.get\_current\_mask()  **if** mask\_img **is** None:  **return** frame            x, y, w, h **=** face\_roi            # 确保坐标有效  **if** x < 0 **or** y < 0 **or** x**+**w > frame.shape[1] **or** y**+**h > frame.shape[0]:  **return** frame    **try**:              # 1. 将面具缩放到人脸大小              mask\_resized **=** cv2.resize(mask\_img, (w, h), interpolation**=**cv2.INTER\_AREA)                # 2. 检查面具是否有Alpha通道  **if** mask\_resized.shape[2] **==** 4:                  # 有Alpha通道的情况（PNG透明图）                  alpha **=** mask\_resized[:, :, 3] **/** 255.0                  mask\_bgr **=** mask\_resized[:, :, :3]  **else**:                  # 无Alpha通道，使用颜色阈值生成掩膜                  # 假设白色或接近白色为背景                  mask\_gray **=** cv2.cvtColor(mask\_resized, cv2.COLOR\_BGR2GRAY)                  \_, binary\_mask **=** cv2.threshold(mask\_gray, 240, 255, cv2.THRESH\_BINARY\_INV)                  alpha **=** binary\_mask **/** 255.0                  mask\_bgr **=** mask\_resized                # 3. 获取人脸ROI              roi **=** frame[y:y**+**h, x:x**+**w]                # 4. Alpha混合              # 公式: output = alpha \* mask + (1-alpha) \* background  **for** c **in** range(3):                  roi[:, :, c] **=** (alpha **\*** mask\_bgr[:, :, c] **+**                                 (1 **-** alpha) **\*** roi[:, :, c]).astype(np.uint8)                # 5. 将处理后的ROI放回原图              frame[y:y**+**h, x:x**+**w] **=** roi    **except** Exception as e:              print(f"[ERROR] 面具应用失败: {e}")    **return** frame    **def** apply\_mask\_bitwise(self, frame, face\_roi, scale**=**1.15, y\_offset\_ratio**=-**0.1):          mask\_img **=** self.get\_current\_mask()  **if** mask\_img **is** None:  **return** frame            x, y, w, h **=** face\_roi            # 计算放大后的面具尺寸          new\_w **=** int(w **\*** scale)          new\_h **=** int(h **\*** scale)            # 计算新的位置（居中并向上偏移）          new\_x **=** x **-** (new\_w **-** w) **//** 2          new\_y **=** y **-** (new\_h **-** h) **//** 2 **+** int(h **\*** y\_offset\_ratio)            # 边界检查和裁剪          frame\_h, frame\_w **=** frame.shape[:2]            # 计算有效区域          src\_x1 **=** max(0, **-**new\_x)          src\_y1 **=** max(0, **-**new\_y)          src\_x2 **=** min(new\_w, frame\_w **-** new\_x)          src\_y2 **=** min(new\_h, frame\_h **-** new\_y)            dst\_x1 **=** max(0, new\_x)          dst\_y1 **=** max(0, new\_y)          dst\_x2 **=** min(frame\_w, new\_x **+** new\_w)          dst\_y2 **=** min(frame\_h, new\_y **+** new\_h)            # 检查有效性  **if** src\_x2 <**=** src\_x1 **or** src\_y2 <**=** src\_y1:  **return** frame    **try**:              # 1. 将面具缩放到放大后的尺寸              mask\_resized **=** cv2.resize(mask\_img, (new\_w, new\_h), interpolation**=**cv2.INTER\_AREA)                # 2. 裁剪面具到有效区域              mask\_cropped **=** mask\_resized[src\_y1:src\_y2, src\_x1:src\_x2]                # 如果有Alpha通道，转为3通道  **if** mask\_cropped.shape[2] **==** 4:                  mask\_bgr **=** mask\_cropped[:, :, :3]                  alpha\_channel **=** mask\_cropped[:, :, 3]  **else**:                  mask\_bgr **=** mask\_cropped                  # 创建掩膜：非白色区域为前景                  mask\_gray **=** cv2.cvtColor(mask\_bgr, cv2.COLOR\_BGR2GRAY)                  \_, alpha\_channel **=** cv2.threshold(mask\_gray, 240, 255, cv2.THRESH\_BINARY\_INV)                # 3. 生成二值掩膜              mask\_binary **=** alpha\_channel              mask\_inv **=** cv2.bitwise\_not(mask\_binary)                # 4. 获取目标ROI              roi **=** frame[dst\_y1:dst\_y2, dst\_x1:dst\_x2]                # 确保尺寸匹配  **if** roi.shape[:2] !**=** mask\_bgr.shape[:2]:  **return** frame                # 5. 使用bitwise\_and在ROI中抠出面具形状的黑色区域              roi\_bg **=** cv2.bitwise\_and(roi, roi, mask**=**mask\_inv)                # 6. 使用bitwise\_and提取面具的前景部分              mask\_fg **=** cv2.bitwise\_and(mask\_bgr, mask\_bgr, mask**=**mask\_binary)                # 7. 使用add融合背景和前景              result **=** cv2.add(roi\_bg, mask\_fg)                # 8. 放回原图              frame[dst\_y1:dst\_y2, dst\_x1:dst\_x2] **=** result    **except** Exception as e:              print(f"[ERROR] 面具应用失败: {e}")    **return** frame      **def** draw\_detections\_with\_mask(frame, faces, detector, mask\_applier, mask\_enabled**=**True):      stats **=** {          "face": len(faces),          "eyes": 0,          "nose": 0,          "mouth": 0,          "left\_ear": 0,          "right\_ear": 0,          "pupil": 0      }        gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)    **for** (x, y, w, h) **in** faces:          face\_roi **=** (x, y, w, h)            # 应用面具  **if** mask\_enabled **and** mask\_applier **is** **not** None:              frame **=** mask\_applier.apply\_mask\_bitwise(frame, face\_roi)  **else**:              # 不应用面具时绘制人脸框              cv2.rectangle(frame, (x, y), (x**+**w, y**+**h), (255, 0, 0), 2)              cv2.putText(frame, "Face", (x, y**-**10),                         cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 0, 0), 2)            # 检测其他器官（即使戴了面具也可以检测，用于统计）          gray\_face **=** gray[y:y**+**h, x:x**+**w]            eyes **=** detector.detect\_eyes(frame, face\_roi, gray\_face.copy())          stats["eyes"] **+=** len(eyes)            noses **=** detector.detect\_nose(frame, face\_roi, gray\_face.copy())          stats["nose"] **+=** len(noses)            mouths **=** detector.detect\_mouth(frame, face\_roi, gray\_face.copy())          stats["mouth"] **+=** len(mouths)            ears **=** detector.detect\_ears(frame, face\_roi, gray\_face.copy())          stats["left\_ear"] **+=** len(ears["left"])          stats["right\_ear"] **+=** len(ears["right"])    **return** frame, stats      **def** main():      """主函数 - AR面具演示"""      print("=" **\*** 50)      print("数字图像处理作业2 - 创新模块1：AR面具系统")      print("=" **\*** 50)        # 初始化检测器和面具应用器      detector **=** FaceDetector()      mask\_applier **=** ARMaskApplier()        # 打开摄像头      print("\n[INFO] 尝试打开摄像头...")      cap **=** cv2.VideoCapture(0, cv2.CAP\_DSHOW)    **if** **not** cap.isOpened():          cap **=** cv2.VideoCapture(0)    **if** **not** cap.isOpened():          print("[ERROR] 无法打开摄像头!")  **return**        cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 640)      cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 480)        print("\n[INFO] 摄像头已开启")      print("[INFO] 按键说明：")      print("  [m] - 开关面具模式")      print("  [n] - 切换下一个面具")      print("  [s] - 保存截图")      print("  [q] - 退出程序")        mask\_enabled **=** True      frame\_count **=** 0    **while** True:          ret, frame **=** cap.read()  **if** **not** ret:  **continue**            # 水平翻转          frame **=** cv2.flip(frame, 1)            # 检测人脸          gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)          faces **=** detector.detect\_face(frame, gray)            # 绘制检测结果（可选面具）          frame, stats **=** draw\_detections\_with\_mask(              frame, faces, detector, mask\_applier, mask\_enabled          )            # 绘制状态面板          frame **=** draw\_status\_panel(frame, stats)            # 绘制模式提示          mode\_text **=** "MASK: ON" **if** mask\_enabled **else** "MASK: OFF"          mode\_color **=** (0, 255, 0) **if** mask\_enabled **else** (0, 0, 255)          cv2.putText(frame, mode\_text, (10, 30),                     cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, mode\_color, 2)    **if** mask\_enabled **and** len(mask\_applier.mask\_names) > 0:              mask\_text **=** f"Current: {mask\_applier.mask\_names[mask\_applier.current\_mask\_idx]}"              cv2.putText(frame, mask\_text, (10, 55),                         cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)            # 显示结果          cv2.imshow("AR Mask - Innovation Module 1", frame)            # 键盘事件          key **=** cv2.waitKey(1) & 0xFF  **if** key **==** ord('q'):  **break**  **elif** key **==** ord('m'):              mask\_enabled **=** **not** mask\_enabled              print(f"[INFO] 面具模式: {'开启' if mask\_enabled else '关闭'}")  **elif** key **==** ord('n'):              mask\_applier.switch\_mask()  **elif** key **==** ord('s'):              filename **=** f"ar\_mask\_screenshot\_{frame\_count}.png"              cv2.imwrite(filename, frame)              print(f"[INFO] 截图已保存: {filename}")              frame\_count **+=** 1        cap.release()      cv2.destroyAllWindows()      print("\n[INFO] 程序已退出")      **if** \_\_name\_\_ **==** "\_\_main\_\_":      main() |

**3.3 privacy\_mosaic.py**

|  |
| --- |
| **import** cv2  **import** numpy as np  **import** os    # 导入基线检测器  **from** baseline **import** FaceDetector, draw\_status\_panel      **class** PrivacyMosaic:    **def** \_\_init\_\_(self, default\_level**=**15):          """          初始化马赛克处理器            Args:              default\_level: 默认马赛克等级（像素块大小）          """          self.level **=** default\_level          self.min\_level **=** 5          self.max\_level **=** 30    **def** increase\_level(self):          """增加马赛克强度（更模糊）"""  **if** self.level < self.max\_level:              self.level **+=** 2              print(f"[INFO] 马赛克等级: {self.level}")    **def** decrease\_level(self):          """降低马赛克强度（更清晰）"""  **if** self.level > self.min\_level:              self.level **-=** 2              print(f"[INFO] 马赛克等级: {self.level}")    **def** apply\_mosaic(self, frame, roi\_rect):          x, y, w, h **=** roi\_rect            # 边界检查          frame\_h, frame\_w **=** frame.shape[:2]          x **=** max(0, x)          y **=** max(0, y)          w **=** min(w, frame\_w **-** x)          h **=** min(h, frame\_h **-** y)    **if** w <**=** 0 **or** h <**=** 0:  **return** frame    **try**:              # 1. 提取ROI              roi **=** frame[y:y**+**h, x:x**+**w]                # 2. 计算缩小后的尺寸              small\_w **=** max(1, w **//** self.level)              small\_h **=** max(1, h **//** self.level)                # 3. 降采样（使用INTER\_LINEAR平滑采样）              small **=** cv2.resize(roi, (small\_w, small\_h), interpolation**=**cv2.INTER\_LINEAR)                # 4. 升采样（使用INTER\_NEAREST最近邻插值，产生方块效果）              mosaic **=** cv2.resize(small, (w, h), interpolation**=**cv2.INTER\_NEAREST)                # 5. 放回原图              frame[y:y**+**h, x:x**+**w] **=** mosaic    **except** Exception as e:              print(f"[ERROR] 马赛克应用失败: {e}")    **return** frame    **def** apply\_mosaic\_face(self, frame, face\_rect, expand\_ratio**=**0.1):          x, y, w, h **=** face\_rect            # 扩展区域          expand\_w **=** int(w **\*** expand\_ratio)          expand\_h **=** int(h **\*** expand\_ratio)            new\_x **=** x **-** expand\_w          new\_y **=** y **-** expand\_h          new\_w **=** w **+** 2 **\*** expand\_w          new\_h **=** h **+** 2 **\*** expand\_h    **return** self.apply\_mosaic(frame, (new\_x, new\_y, new\_w, new\_h))    **def** apply\_mosaic\_eyes(self, frame, eye\_rects, expand\_ratio**=**0.3):  **for** (ex, ey, ew, eh) **in** eye\_rects:              # 扩展眼睛区域              expand\_w **=** int(ew **\*** expand\_ratio)              expand\_h **=** int(eh **\*** expand\_ratio)                new\_x **=** ex **-** expand\_w              new\_y **=** ey **-** expand\_h              new\_w **=** ew **+** 2 **\*** expand\_w              new\_h **=** eh **+** 2 **\*** expand\_h                frame **=** self.apply\_mosaic(frame, (new\_x, new\_y, new\_w, new\_h))    **return** frame      **def** draw\_detections\_with\_mosaic(frame, faces, detector, mosaic\_processor,                                   mosaic\_mode**=**"off"):      stats **=** {          "face": len(faces),          "eyes": 0,          "nose": 0,          "mouth": 0,          "left\_ear": 0,          "right\_ear": 0,          "pupil": 0      }        gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)    **for** (x, y, w, h) **in** faces:          face\_roi **=** (x, y, w, h)          gray\_face **=** gray[y:y**+**h, x:x**+**w]            # 检测眼睛（用于eyes模式和统计）          eyes **=** detector.detect\_eyes(frame, face\_roi, gray\_face.copy())          stats["eyes"] **+=** len(eyes)            # 应用马赛克  **if** mosaic\_mode **==** "face":              # 整脸马赛克              frame **=** mosaic\_processor.apply\_mosaic\_face(frame, face\_roi)  **elif** mosaic\_mode **==** "eyes":              # 仅眼睛马赛克              frame **=** mosaic\_processor.apply\_mosaic\_eyes(frame, eyes)              # 绘制人脸框（眼睛模式下显示人脸框）              cv2.rectangle(frame, (x, y), (x**+**w, y**+**h), (255, 0, 0), 2)  **else**:              # 正常模式：绘制所有检测框              cv2.rectangle(frame, (x, y), (x**+**w, y**+**h), (255, 0, 0), 2)              cv2.putText(frame, "Face", (x, y**-**10),                         cv2.FONT\_HERSHEY\_SIMPLEX, 0.6, (255, 0, 0), 2)                # 绘制眼睛  **for** (ex, ey, ew, eh) **in** eyes:                  cv2.rectangle(frame, (ex, ey), (ex**+**ew, ey**+**eh), (0, 255, 0), 2)            # 其他器官检测（用于统计）          noses **=** detector.detect\_nose(frame, face\_roi, gray\_face.copy())          stats["nose"] **+=** len(noses)            mouths **=** detector.detect\_mouth(frame, face\_roi, gray\_face.copy())          stats["mouth"] **+=** len(mouths)            ears **=** detector.detect\_ears(frame, face\_roi, gray\_face.copy())          stats["left\_ear"] **+=** len(ears["left"])          stats["right\_ear"] **+=** len(ears["right"])    **return** frame, stats      **def** main():      """主函数 - 隐私马赛克演示"""      print("=" **\*** 50)      print("数字图像处理作业2 - 创新模块3：隐私马赛克盾")      print("=" **\*** 50)        # 初始化      detector **=** FaceDetector()      mosaic\_processor **=** PrivacyMosaic(default\_level**=**15)        # 打开摄像头      print("\n[INFO] 尝试打开摄像头...")      cap **=** cv2.VideoCapture(0, cv2.CAP\_DSHOW)    **if** **not** cap.isOpened():          cap **=** cv2.VideoCapture(0)    **if** **not** cap.isOpened():          print("[ERROR] 无法打开摄像头!")  **return**        cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, 640)      cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, 480)        print("\n[INFO] 摄像头已开启")      print("[INFO] 按键说明：")      print("  [p] - 切换马赛克模式 (关闭 → 整脸 → 仅眼睛)")      print("  [+] - 增加马赛克强度")      print("  [-] - 降低马赛克强度")      print("  [s] - 保存截图")      print("  [q] - 退出程序")        mosaic\_modes **=** ["off", "face", "eyes"]      mosaic\_mode\_names **=** {"off": "关闭", "face": "整脸", "eyes": "仅眼睛"}      current\_mode\_idx **=** 0      frame\_count **=** 0    **while** True:          ret, frame **=** cap.read()  **if** **not** ret:  **continue**            # 水平翻转          frame **=** cv2.flip(frame, 1)            # 检测人脸          gray **=** cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)          faces **=** detector.detect\_face(frame, gray)            # 绘制检测结果（可选马赛克）          current\_mode **=** mosaic\_modes[current\_mode\_idx]          frame, stats **=** draw\_detections\_with\_mosaic(              frame, faces, detector, mosaic\_processor, current\_mode          )            # 绘制状态面板          frame **=** draw\_status\_panel(frame, stats)            # 绘制模式提示          mode\_text **=** f"PRIVACY: {mosaic\_mode\_names[current\_mode]}"          mode\_color **=** (0, 255, 0) **if** current\_mode !**=** "off" **else** (0, 0, 255)          cv2.putText(frame, mode\_text, (10, 30),                     cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, mode\_color, 2)    **if** current\_mode !**=** "off":              level\_text **=** f"Level: {mosaic\_processor.level}"              cv2.putText(frame, level\_text, (10, 55),                         cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)            # 显示结果          cv2.imshow("Privacy Mosaic - Innovation Module 3", frame)            # 键盘事件          key **=** cv2.waitKey(1) & 0xFF  **if** key **==** ord('q'):  **break**  **elif** key **==** ord('p'):              current\_mode\_idx **=** (current\_mode\_idx **+** 1) **%** len(mosaic\_modes)              print(f"[INFO] 马赛克模式: {mosaic\_mode\_names[mosaic\_modes[current\_mode\_idx]]}")  **elif** key **==** ord('+') **or** key **==** ord('='):              mosaic\_processor.increase\_level()  **elif** key **==** ord('-') **or** key **==** ord('\_'):              mosaic\_processor.decrease\_level()  **elif** key **==** ord('s'):              filename **=** f"privacy\_screenshot\_{frame\_count}.png"              cv2.imwrite(filename, frame)              print(f"[INFO] 截图已保存: {filename}")              frame\_count **+=** 1        cap.release()      cv2.destroyAllWindows()      print("\n[INFO] 程序已退出")      **if** \_\_name\_\_ **==** "\_\_main\_\_":      main() |