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

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 $O(1)$ 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：

$$f_{Haar} = \sum_{i \in white} I(i) - \sum_{j \in black} I(j) \qquad (1-1)$$

其中 $I(i)$ 表示像素 i 的灰度值。

积分图实现 $O(1)$ 时间复杂度的特征计算：

$$II(x,y) = \sum_{x' \le x, y' \le y} I(x',y') \qquad (1-2)$$

任意矩形区域的像素和可通过 4 次查表计算：

$$\sum_{(x,y) \in R} I(x,y) = II(D) - II(B) - II(C) + II(A) \qquad (1-3)$$

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
```

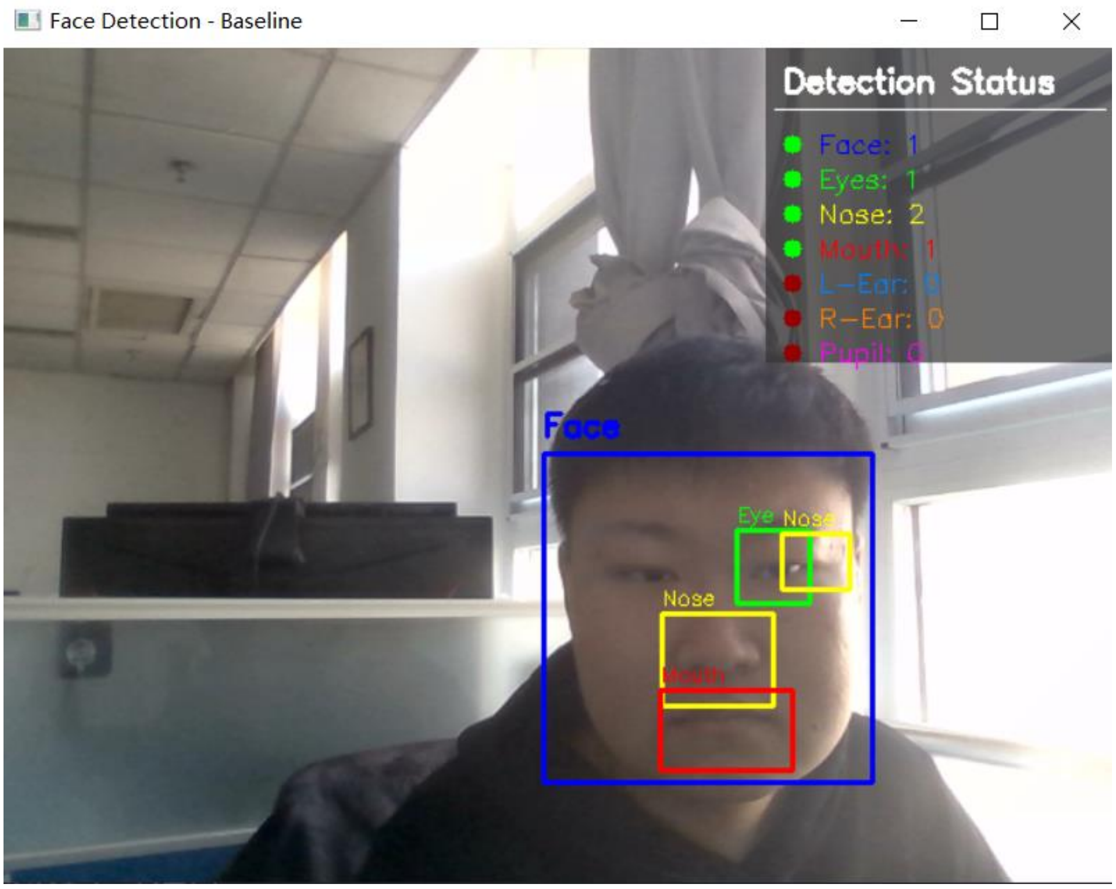
关键参数解析：

• scaleFactor=1.1：每次图像缩小为原来的 $\frac{1}{1.1} \approx 90.9\%$，用于多尺度检测

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

• minSize=(30,30)：过滤小于 30×30 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。

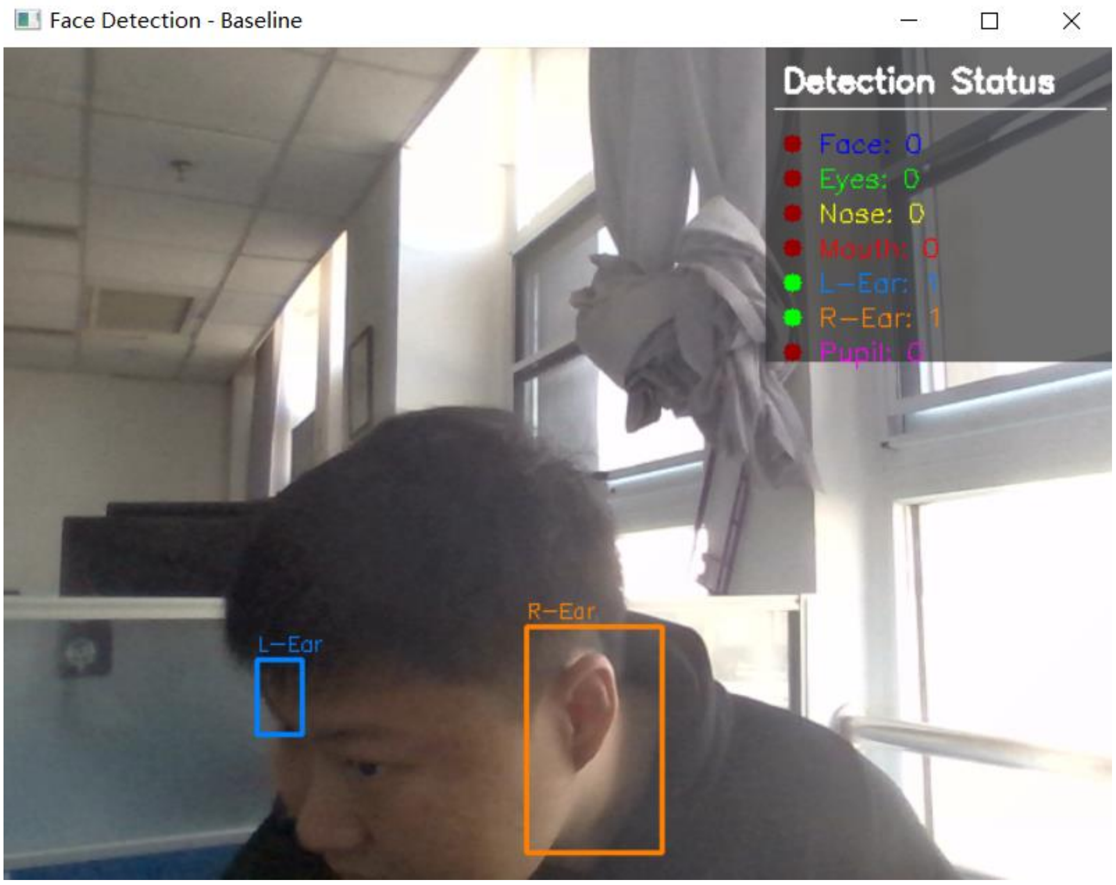
问题发现：在实际测试中发现，当用户呈现侧脸时：

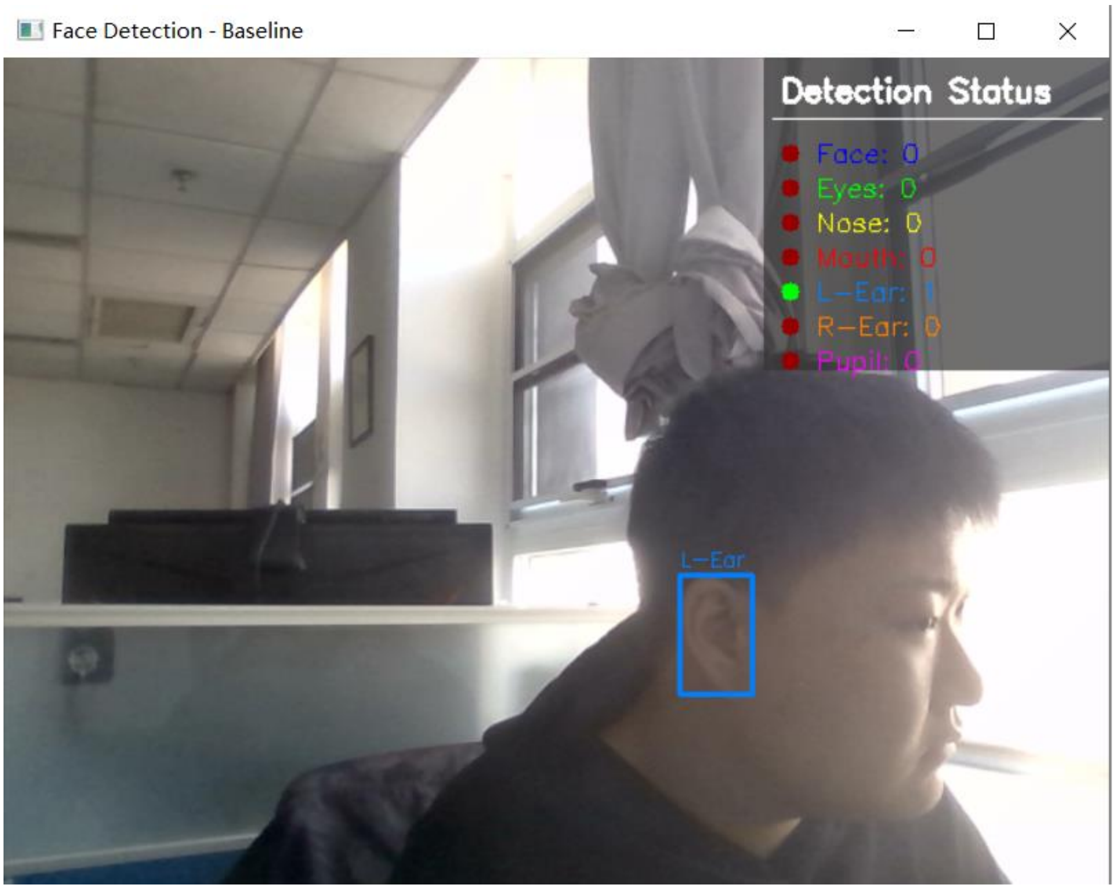
1. 正脸分类器无法检测到人脸（Face: 0）

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：


Face Detection - Baseline
Detection Status
Face: 0
Eyes: 0
Nose: 0
Mouth: 0
L-Ear: 1
R-Ear: 1
Pupil: 0
L-Ear
R-Ear


Face Detection - Baseline
Detection Status
Face: 0
Eyes: 0
Nose: 0
Mouth: 0
L-Ear: 1
R-Ear: 0
Pupil: 0
L-Ear

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：

$$Iinv(x,y) = 255 - I(x,y) \qquad (1-4)$$

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：

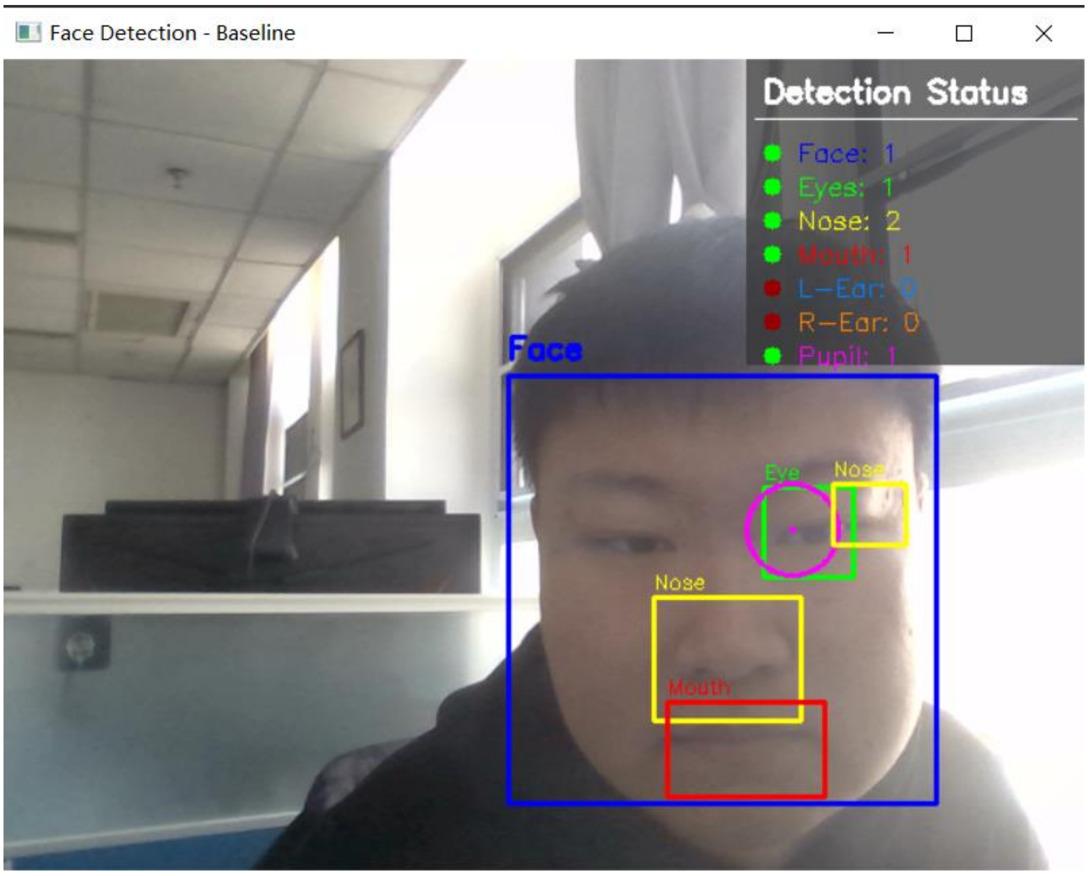
1. 面积约束：瞳孔面积应占眼睛区域的 1%~50%

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。

$$Circularity = \frac{4\pi \cdot Area}{Perimeter^2} \qquad (1-5)$$

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。



Detection Status
● Face: 1
● Eyes: 1
● Nose: 2
● Mouth: 1
● L-Ear: 0
● R-Ear: 0
● Pupil: 1

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)
```

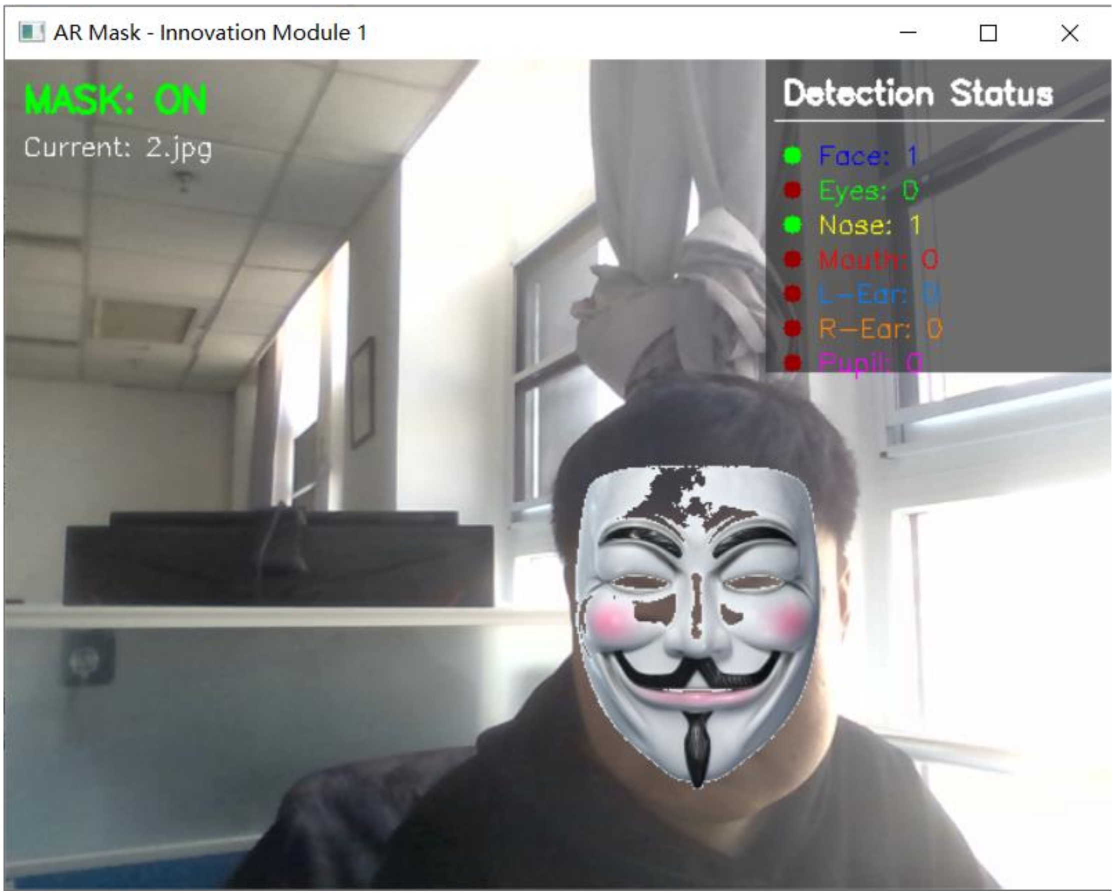
实际测试中发现两个问题：

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

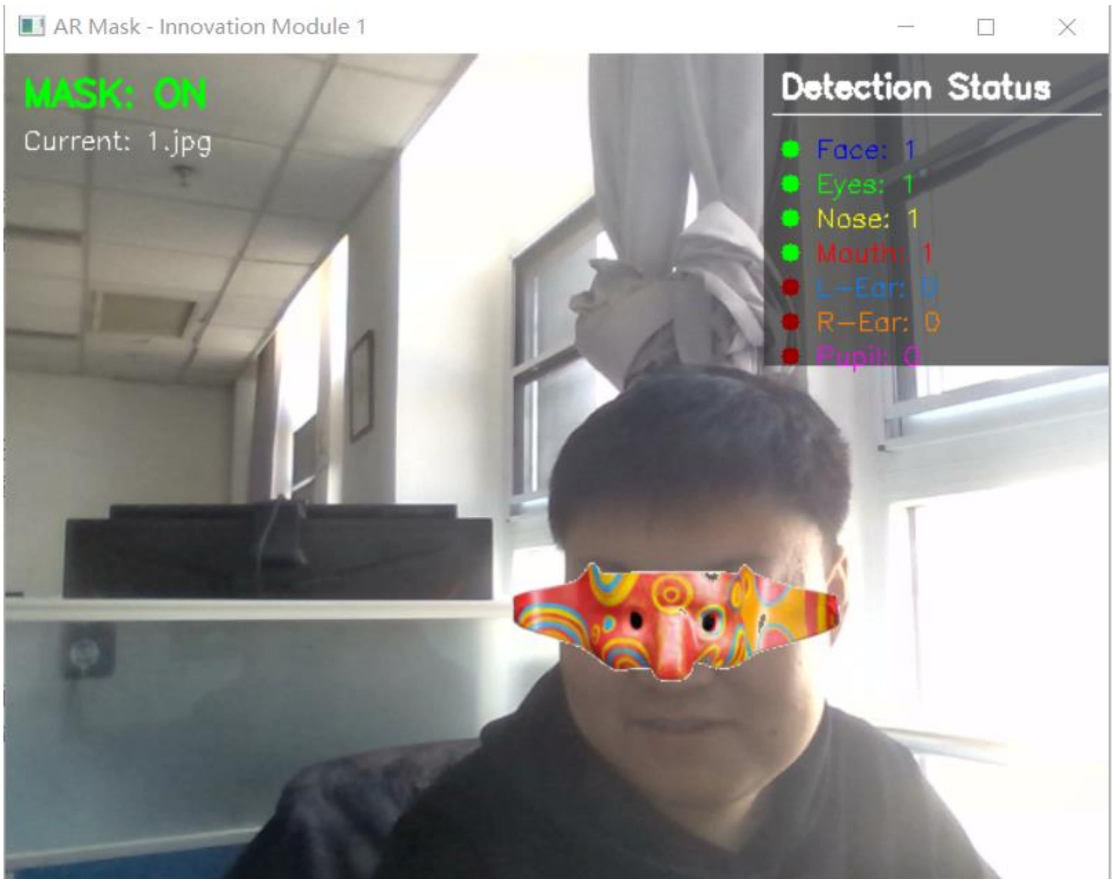
解决方案：添加缩放和偏移参数 scale、y_offset_ratio

最终效果如下。


```
AR Mask - Innovation Module 1

MASK: ON
Current: 2.jpg

Detection Status
● Face: 1
● Eyes: 0
● Nose: 1
● Mouth: 0
● L-Ear: 0
● R-Ear: 0
● Pupil: 0
```


```
AR Mask - Innovation Module 1

MASK: ON
Current: 1.jpg

Detection Status
● Face: 1
● Eyes: 1
● Nose: 1
● Mouth: 1
● L-Ear: 0
● R-Ear: 0
● Pupil: 0
```

2.2 几何约束过滤器

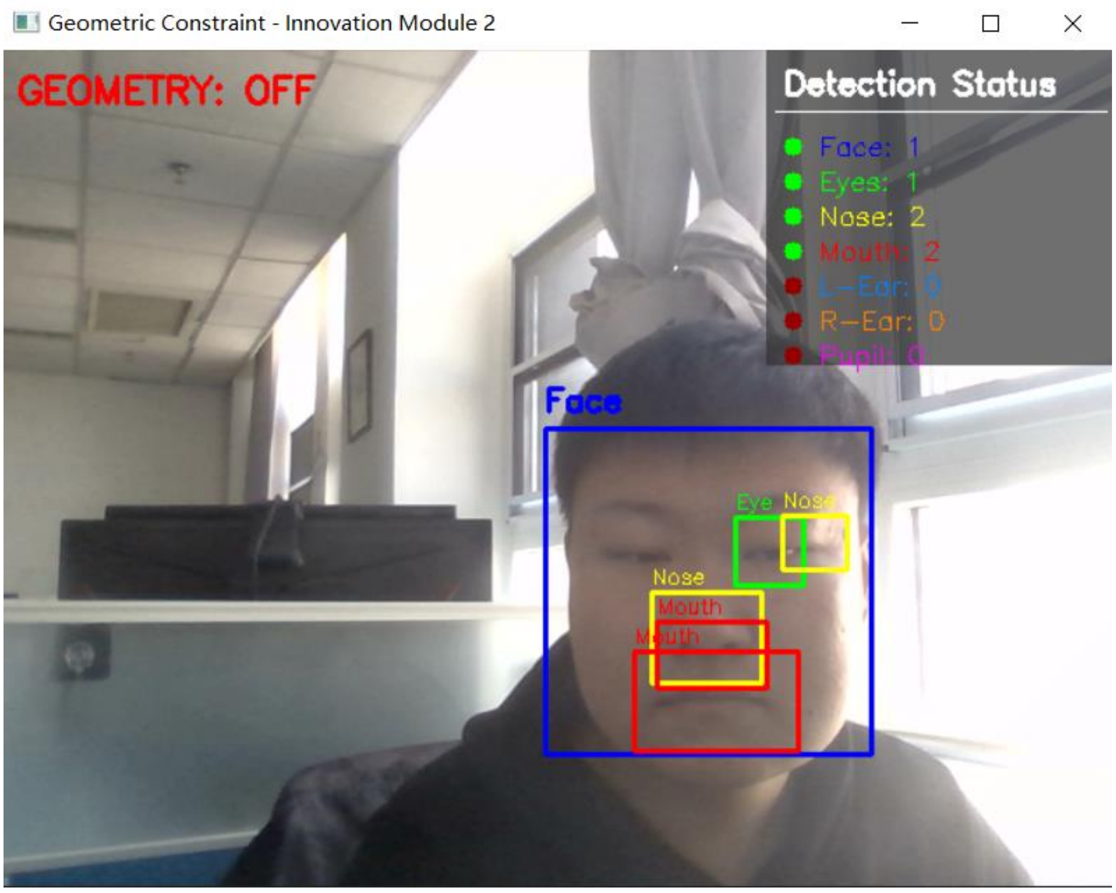
Haar 级联分类器存在误检问题，例如将背景纹理误识别为嘴巴。通过引入解剖学常识，对检测结果进行逻辑校验，可有效降低误检率。

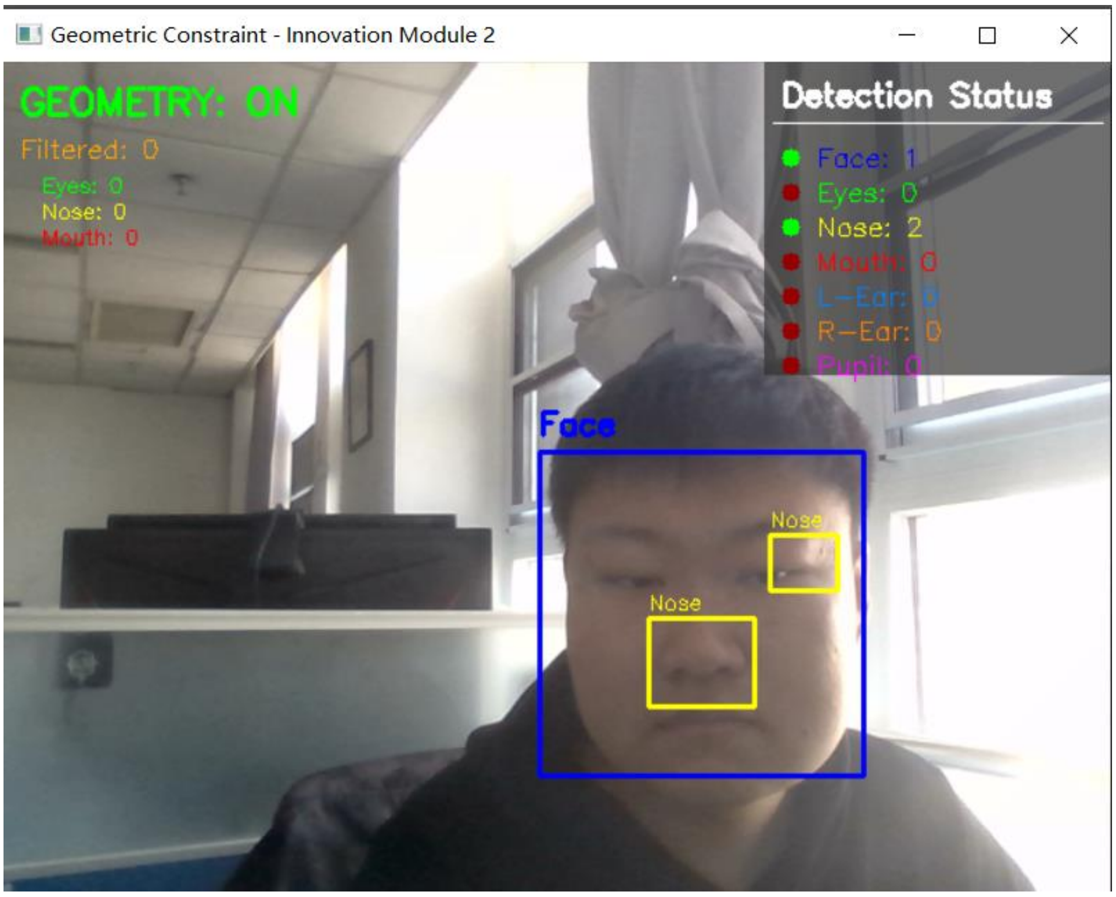
核心思想：器官位置应符合人脸解剖学规律

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

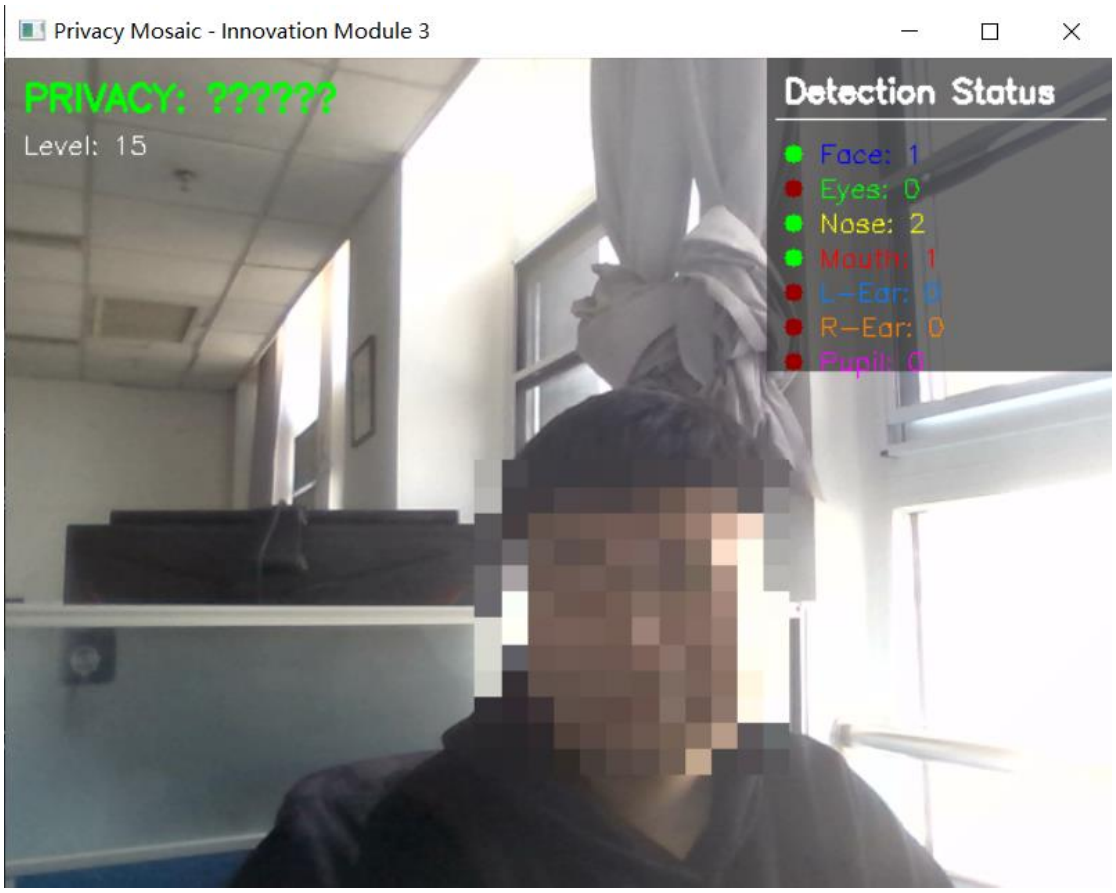
2.3 隐私马赛克盾

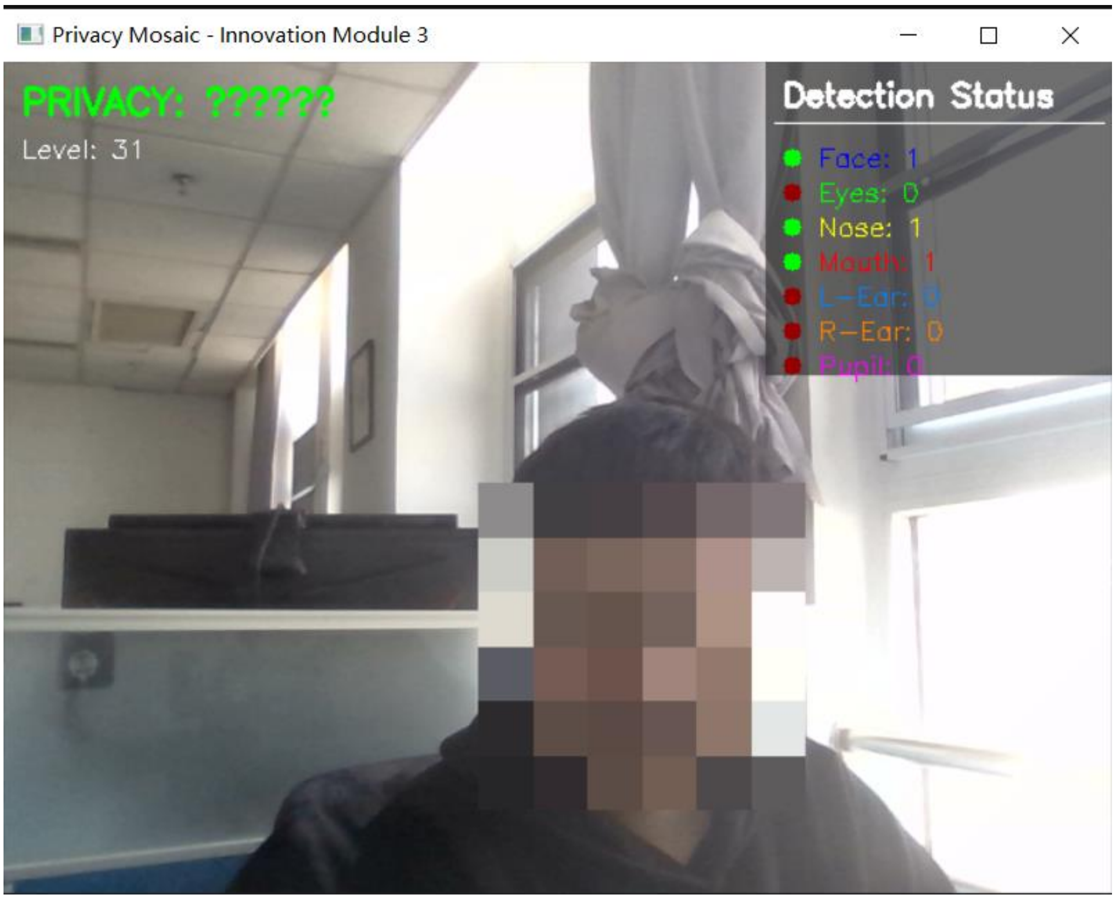
本模块实现实时人脸马赛克效果。在新闻报道、监控回放等场景中，常需要保护个人隐私。

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：


Privacy Mosaic - Innovation Module 3
PRIVACY: ??????
Level: 15
Detection Status
Face: 1
Eyes: 0
Nose: 2
Mouth: 1
L-Ear: 0
R-Ear: 0
Pupil: 0


Privacy Mosaic - Innovation Module 3
PRIVACY: ??????
Level: 31
Detection Status
Face: 1
Eyes: 0
Nose: 1
Mouth: 1
L-Ear: 0
R-Ear: 0
Pupil: 0

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)
```

```python
        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'):
```

```python
            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

```python
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()
```