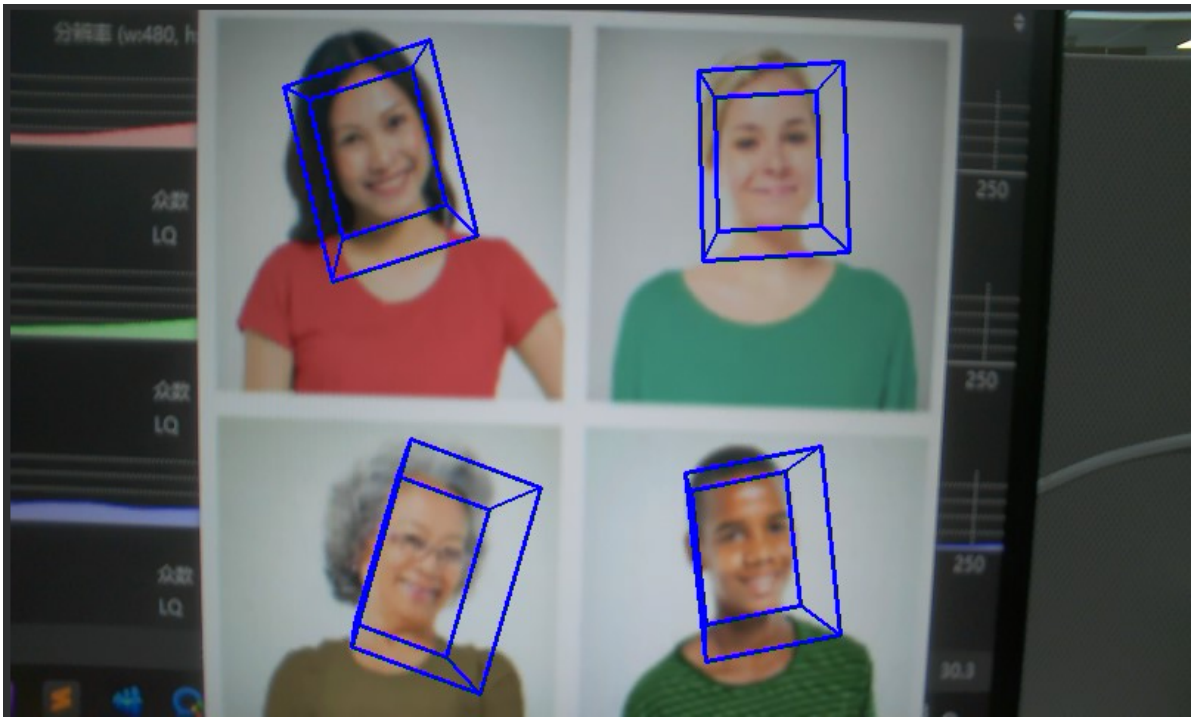
Face orientation detection

Routine Experiment Effect

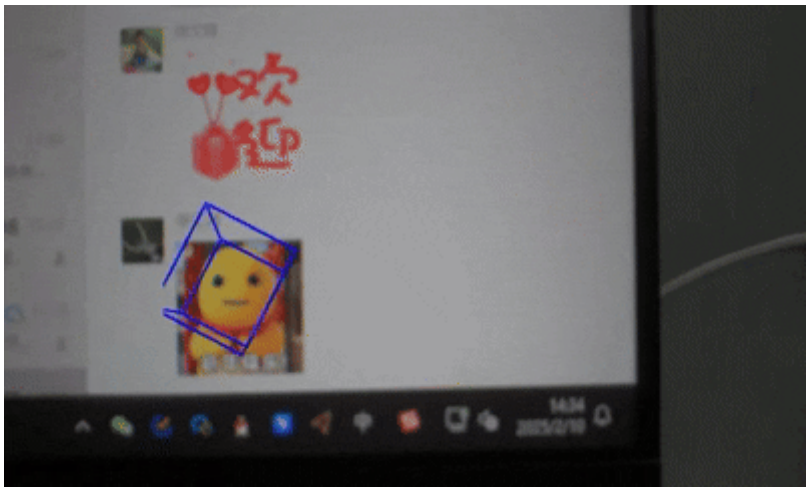
In this section, we will learn how to use K230 to detect and judge the face orientation (posture).

The code in this section is in [Source code/07.Face/03.face_pose.py]

After connecting to the IDE, run the example code in this section and aim the K230 at the face. You can see that the direction of the current face will be displayed on the screen.



This model is highly compatible, and humanoid creatures can occasionally be recognized.



Code Explanation

Code structure

Import related dependency classes

```
from libs.PipeLine import PipeLine, ScopedTiming
from libs.AIBase import AIBase
from libs.AI2D import Ai2d
import os
import ujson
from media.media import *
from time import *
import nncase_runtime as nn
import ulab.numpy as np
import time
import image
import aidemo
import random
import gc
import sys

global fp
```

Define the face detection class

```
# Custom face detection task class
# 自定义人脸检测任务类
class FaceDetApp(AIBase):
    """
    人脸检测应用类，继承自AIBase基类
    (Face detection application class, inherits from AIBase)
    """
    def __init__(self, kmodel_path, model_input_size, anchors,
confidence_threshold=0.25, nms_threshold=0.3, rgb888p_size=[1280,720],
display_size=[1920,1080], debug_mode=0):
        """
        初始化人脸检测应用类
        (Initialize the face detection application class)

        参数:
        kmodel_path: 模型文件路径
        (kmodel_path: Path to the model file)
        model_input_size: 模型输入尺寸 [width, height]
        (model_input_size: Model input dimensions [width, height])
        anchors: 预定义的锚框
        (anchors: Predefined anchor boxes)
        confidence_threshold: 置信度阈值，低于此值的检测结果将被过滤
        (confidence_threshold: Confidence threshold, detections below this value
will be filtered out)
        nms_threshold: 非极大值抑制阈值
        (nms_threshold: Non-maximum suppression threshold)
        rgb888p_size: 传感器输入图像尺寸
        (rgb888p_size: Sensor input image dimensions)
        display_size: 显示输出尺寸
        (display_size: Display output dimensions)
        debug_mode: 调试模式级别
        (debug_mode: Debug mode level)
        """
        super().__init__(kmodel_path, model_input_size, rgb888p_size,
debug_mode)
        # kmodel路径
        # (Path to the kmodel file)
        self.kmodel_path = kmodel_path
        # 检测模型输入分辨率
        # (Detection model input resolution)
        self.model_input_size = model_input_size
        # 置信度阈值
        # (Confidence threshold)
        self.confidence_threshold = confidence_threshold
        # nms阈值
        # (NMS threshold)
        self.nms_threshold = nms_threshold
        self.anchors = anchors
        # sensor给到AI的图像分辨率，宽16字节对齐
        # (Image resolution from sensor to AI, width aligned to 16 bytes)
        self.rgb888p_size = [ALIGN_UP(rgb888p_size[0], 16), rgb888p_size[1]]
        # 视频输出VO分辨率，宽16字节对齐
        # (Video output resolution, width aligned to 16 bytes)
        self.display_size = [ALIGN_UP(display_size[0], 16), display_size[1]]
        # debug模式
```

```
        # (Debug mode)
        self.debug_mode = debug_mode
        # 实例化Ai2d，用于实现模型预处理
        # (Instantiate Ai2d for model preprocessing)
        self.ai2d = Ai2d(debug_mode)
        # 设置Ai2d的输入输出格式和类型
        # (Set Ai2d input and output formats and types)
        self.ai2d.set_ai2d_dtype(nn.ai2d_format.NCHW_FMT,
nn.ai2d_format.NCHW_FMT, np.uint8, np.uint8)

    # 配置预处理操作，这里使用了pad和resize，Ai2d支持crop/shift/pad/resize/affine，具体
代码请打开/sdcard/app/libs/AI2D.py查看
    # (Configure preprocessing operations, using pad and resize here, Ai2d
supports crop/shift/pad/resize/affine)
    def config_preprocess(self, input_image_size=None):
        """
        配置模型输入数据的预处理操作
        (Configure preprocessing operations for model input data)

        参数:
        input_image_size: 输入图像尺寸，如果为None则使用默认rgb888p_size
        (input_image_size: Input image size, if None uses default rgb888p_size)
        """
        with ScopedTiming("set preprocess config", self.debug_mode > 0):
            # 初始化ai2d预处理配置，默认为sensor给到AI的尺寸，可以通过设置
input_image_size自行修改输入尺寸
            # (Initialize ai2d preprocessing configuration, default is the size
from sensor to AI)
            ai2d_input_size = input_image_size if input_image_size else
self.rgb888p_size
            # 计算padding参数，并设置padding预处理
            # (Calculate padding parameters and set padding preprocessing)
            self.ai2d.pad(self.get_pad_param(), 0, [104, 117, 123])
            # 设置resize预处理
            # (Set resize preprocessing)
            self.ai2d.resize(nn.interp_method.tf_bilinear,
nn.interp_mode.half_pixel)
            # 构建预处理流程,参数为预处理输入tensor的shape和预处理输出的tensor的shape
            # (Build preprocessing pipeline, parameters are the shape of input
tensor and output tensor)
            self.ai2d.build([1, 3, ai2d_input_size[1], ai2d_input_size[0]], [1,
3, self.model_input_size[1], self.model_input_size[0]])

    # 自定义后处理，results是模型输出的array列表，这里使用了aidemo库的
face_det_post_process接口
    # (Custom post-processing, results is the array list output by the model)
    def postprocess(self, results):
        """
        对模型输出进行后处理
        (Post-process the model output)

        参数:
        results: 模型的原始输出
        (results: Raw output from the model)

        返回:
        处理后的检测结果
        (Processed detection results)
```

```
        """
        with ScopedTiming("postprocess", self.debug_mode > 0):
            # 调用aidemo库中的人脸检测后处理函数
            # (Call the face detection post-processing function from aidemo
library)
            res = aidemo.face_det_post_process(self.confidence_threshold,
self.nms_threshold, self.model_input_size[0], self.anchors, self.rgb888p_size,
results)
            if len(res) == 0:
                return res
            else:
                return res[0]

    # 计算padding参数
    # (Calculate padding parameters)
    def get_pad_param(self):
        """
        计算padding参数，确保等比例缩放图像到模型输入尺寸
        (Calculate padding parameters to ensure proportional scaling of the image
to model input size)

        返回:
        padding参数 [top, bottom, left, right]
        (Padding parameters [top, bottom, left, right])
        """
        dst_w = self.model_input_size[0]
        dst_h = self.model_input_size[1]
        # 计算最小的缩放比例，等比例缩放
        # (Calculate the minimum scaling ratio for proportional scaling)
        ratio_w = dst_w / self.rgb888p_size[0]
        ratio_h = dst_h / self.rgb888p_size[1]
        if ratio_w < ratio_h:
            ratio = ratio_w
        else:
            ratio = ratio_h
        new_w = (int)(ratio * self.rgb888p_size[0])
        new_h = (int)(ratio * self.rgb888p_size[1])
        dw = (dst_w - new_w) / 2
        dh = (dst_h - new_h) / 2
        top = (int)(round(0))
        bottom = (int)(round(dh * 2 + 0.1))
        left = (int)(round(0))
        right = (int)(round(dw * 2 - 0.1))
        return [0, 0, 0, 0, top, bottom, left, right]
```

Define the face posture detection application class

```
# 自定义人脸姿态任务类
# (Custom face pose estimation task class)
class FacePoseApp(AIBase):
    """
    人脸姿态估计应用类，继承自AIBase基类
    (Face pose estimation application class, inherits from AIBase)
    """
```

```
    def __init__(self, kmodel_path, model_input_size, rgb888p_size=[1920, 1080],
display_size=[1920, 1080], debug_mode=0):
        """
        初始化人脸姿态估计应用类
        (Initialize the face pose estimation application class)

        参数:
        kmodel_path: 模型文件路径
        (kmodel_path: Path to the model file)
        model_input_size: 模型输入尺寸 [width, height]
        (model_input_size: Model input dimensions [width, height])
        rgb888p_size: 传感器输入图像尺寸
        (rgb888p_size: Sensor input image dimensions)
        display_size: 显示输出尺寸
        (display_size: Display output dimensions)
        debug_mode: 调试模式级别
        (debug_mode: Debug mode level)
        """
        super().__init__(kmodel_path, model_input_size, rgb888p_size,
debug_mode)
        # kmodel路径
        # (Path to the kmodel file)
        self.kmodel_path = kmodel_path
        # 人脸姿态模型输入分辨率
        # (Face pose model input resolution)
        self.model_input_size = model_input_size
        # sensor给到AI的图像分辨率，宽16字节对齐
        # (Image resolution from sensor to AI, width aligned to 16 bytes)
        self.rgb888p_size = [ALIGN_UP(rgb888p_size[0], 16), rgb888p_size[1]]
        # 视频输出VO分辨率，宽16字节对齐
        # (Video output resolution, width aligned to 16 bytes)
        self.display_size = [ALIGN_UP(display_size[0], 16), display_size[1]]
        # debug模式
        # (Debug mode)
        self.debug_mode = debug_mode
        # 实例化Ai2d，用于实现模型预处理
        # (Instantiate Ai2d for model preprocessing)
        self.ai2d = Ai2d(debug_mode)
        # 设置Ai2d的输入输出格式和类型
        # (Set Ai2d input and output formats and types)
        self.ai2d.set_ai2d_dtype(nn.ai2d_format.NCHW_FMT,
nn.ai2d_format.NCHW_FMT, np.uint8, np.uint8)

    # 配置预处理操作，这里使用了affine，Ai2d支持crop/shift/pad/resize/affine，具体代码请
打开/sdcard/app/libs/AI2D.py查看
    # (Configure preprocessing operations, using affine here, Ai2d supports
crop/shift/pad/resize/affine)
    def config_preprocess(self, det, input_image_size=None):
        """
        配置模型输入数据的预处理操作
        (Configure preprocessing operations for model input data)

        参数:
        det: 人脸检测结果，包含边界框信息
        (det: Face detection result containing bounding box information)
        input_image_size: 输入图像尺寸，如果为None则使用默认rgb888p_size
        (input_image_size: Input image size, if None uses default rgb888p_size)
        """
```

```
        with ScopedTiming("set preprocess config", self.debug_mode > 0):
            # 初始化ai2d预处理配置，默认为sensor给到AI的尺寸，可以通过设置
input_image_size自行修改输入尺寸
            # (Initialize ai2d preprocessing configuration, default is the size
from sensor to AI)
            ai2d_input_size = input_image_size if input_image_size else
self.rgb888p_size
            # 计算affine矩阵并设置affine预处理
            # (Calculate affine matrix and set affine preprocessing)
            matrix_dst = self.get_affine_matrix(det)
            self.ai2d.affine(nn.interp_method.cv2_bilinear, 0, 0, 127, 1,
matrix_dst)
            # 构建预处理流程,参数为预处理输入tensor的shape和预处理输出的tensor的shape
            # (Build preprocessing pipeline, parameters are the shape of input
tensor and output tensor)
            self.ai2d.build([1, 3, ai2d_input_size[1], ai2d_input_size[0]], [1,
3, self.model_input_size[1], self.model_input_size[0]])

    # 自定义后处理，results是模型输出的array列表，计算欧拉角
    # (Custom post-processing, results is the array list output by the model,
calculating Euler angles)
    def postprocess(self, results):
        """
        对模型输出进行后处理，计算旋转矩阵和欧拉角
        (Post-process the model output to calculate rotation matrix and Euler
angles)

        参数:
        results: 模型的原始输出
        (results: Raw output from the model)

        返回:
        旋转矩阵R和欧拉角eular
        (Rotation matrix R and Euler angles)
        """
        with ScopedTiming("postprocess", self.debug_mode > 0):
            R, eular = self.get_euler(results[0][0])
            return R, eular

    def get_affine_matrix(self, bbox):
        """
        获取仿射变换矩阵，用于将人脸区域变换到模型输入空间
        (Get the affine transformation matrix to transform the face region to
model input space)

        参数:
        bbox: 边界框坐标 [x, y, width, height]
        (bbox: Bounding box coordinates [x, y, width, height])

        返回:
        仿射变换矩阵
        (Affine transformation matrix)
        """
        with ScopedTiming("get_affine_matrix", self.debug_mode > 1):
            # 设置缩放因子
            # (Set scaling factor)
            factor = 2.7
            # 从边界框提取坐标和尺寸
```

```
            # (Extract coordinates and dimensions from bounding box)
            x1, y1, w, h = map(lambda x: int(round(x, 0)), bbox[:4])
            # 模型输入大小
            # (Model input size)
            edge_size = self.model_input_size[1]
            # 平移距离，使得模型输入空间的中心对准原点
            # (Translation distance to align the center of model input space
with the origin)
            trans_distance = edge_size / 2.0
            # 计算边界框中心点的坐标
            # (Calculate the coordinates of the bounding box center)
            center_x = x1 + w / 2.0
            center_y = y1 + h / 2.0
            # 计算最大边长
            # (Calculate maximum edge length)
            maximum_edge = factor * (h if h > w else w)
            # 计算缩放比例
            # (Calculate scaling ratio)
            scale = edge_size * 2.0 / maximum_edge
            # 计算平移参数
            # (Calculate translation parameters)
            cx = trans_distance - scale * center_x
            cy = trans_distance - scale * center_y
            # 创建仿射矩阵
            # (Create affine matrix)
            affine_matrix = [scale, 0, cx, 0, scale, cy]
            return affine_matrix

    def rotation_matrix_to_euler_angles(self, R):
        """
        将旋转矩阵转换为欧拉角
        (Convert rotation matrix to Euler angles)

        参数:
        R: 3x3旋转矩阵
        (R: 3x3 rotation matrix)

        返回:
        欧拉角 [pitch, yaw, roll]，单位为度
        (Euler angles [pitch, yaw, roll] in degrees)
        """
        # 计算 sin(yaw)
        # (Calculate sin(yaw))
        sy = np.sqrt(R[0, 0] ** 2 + R[1, 0] ** 2)
        if sy < 1e-6:
            # 若 sin(yaw) 过小，说明 pitch 接近 ±90 度
            # (If sin(yaw) is too small, pitch is close to ±90 degrees)
            pitch = np.arctan2(-R[1, 2], R[1, 1]) * 180 / np.pi
            yaw = np.arctan2(-R[2, 0], sy) * 180 / np.pi
            roll = 0
        else:
            # 计算 pitch、yaw、roll 的角度
            # (Calculate pitch, yaw, roll angles)
            pitch = np.arctan2(R[2, 1], R[2, 2]) * 180 / np.pi
            yaw = np.arctan2(-R[2, 0], sy) * 180 / np.pi
            roll = np.arctan2(R[1, 0], R[0, 0]) * 180 / np.pi
        return [pitch, yaw, roll]
```

```
    def get_euler(self, data):
        """
        从模型输出数据中获取旋转矩阵和欧拉角
        (Get rotation matrix and Euler angles from model output data)

        参数:
        data: 模型输出数据
        (data: Model output data)

        返回:
        旋转矩阵R和欧拉角
        (Rotation matrix R and Euler angles)
        """
        # 获取旋转矩阵和欧拉角
        # (Get rotation matrix and Euler angles)
        R = data[:3, :3].copy()
        eular = self.rotation_matrix_to_euler_angles(R)
        return R, eular
```

Define the main class for face pose detection

```
# 人脸姿态任务类
# (Face pose task class)
class FacePose:
    """
    集成人脸检测和姿态估计的综合任务类
    (Integrated task class for face detection and pose estimation)
    """
    def __init__(self, face_det_kmodel, face_pose_kmodel, det_input_size,
pose_input_size, anchors, confidence_threshold=0.25, nms_threshold=0.3,
rgb888p_size=[1280, 720], display_size=[1920, 1080], debug_mode=0):
        """
        初始化人脸姿态任务类
        (Initialize the face pose task class)

        参数:
        face_det_kmodel: 人脸检测模型路径
        (face_det_kmodel: Path to face detection model)
        face_pose_kmodel: 人脸姿态模型路径
        (face_pose_kmodel: Path to face pose model)
        det_input_size: 检测模型输入尺寸
        (det_input_size: Detection model input size)
        pose_input_size: 姿态模型输入尺寸
        (pose_input_size: Pose model input size)
        anchors: 预定义的锚框
        (anchors: Predefined anchor boxes)
        confidence_threshold: 置信度阈值
        (confidence_threshold: Confidence threshold)
        nms_threshold: 非极大值抑制阈值
        (nms_threshold: Non-maximum suppression threshold)
        rgb888p_size: 传感器输入图像尺寸
        (rgb888p_size: Sensor input image dimensions)
        display_size: 显示输出尺寸
        (display_size: Display output dimensions)
```

```
        debug_mode: 调试模式级别
        (debug_mode: Debug mode level)
        """
        # 人脸检测模型路径
        # (Path to face detection model)
        self.face_det_kmodel = face_det_kmodel
        # 人脸姿态模型路径
        # (Path to face pose model)
        self.face_pose_kmodel = face_pose_kmodel
        # 人脸检测模型输入分辨率
        # (Face detection model input resolution)
        self.det_input_size = det_input_size
        # 人脸姿态模型输入分辨率
        # (Face pose model input resolution)
        self.pose_input_size = pose_input_size
        # anchors
        # (Anchor boxes)
        self.anchors = anchors
        # 置信度阈值
        # (Confidence threshold)
        self.confidence_threshold = confidence_threshold
        # nms阈值
        # (NMS threshold)
        self.nms_threshold = nms_threshold
        # sensor给到AI的图像分辨率，宽16字节对齐
        # (Image resolution from sensor to AI, width aligned to 16 bytes)
        self.rgb888p_size = [ALIGN_UP(rgb888p_size[0], 16), rgb888p_size[1]]
        # 视频输出VO分辨率，宽16字节对齐
        # (Video output resolution, width aligned to 16 bytes)
        self.display_size = [ALIGN_UP(display_size[0], 16), display_size[1]]
        # debug_mode模式
        # (Debug mode)
        self.debug_mode = debug_mode
        # 初始化人脸检测和姿态估计模型
        # (Initialize face detection and pose estimation models)
        self.face_det = FaceDetApp(self.face_det_kmodel,
model_input_size=self.det_input_size, anchors=self.anchors,
confidence_threshold=self.confidence_threshold,
nms_threshold=self.nms_threshold, rgb888p_size=self.rgb888p_size,
display_size=self.display_size, debug_mode=0)
        self.face_pose = FacePoseApp(self.face_pose_kmodel,
model_input_size=self.pose_input_size, rgb888p_size=self.rgb888p_size,
display_size=self.display_size)
        # 配置人脸检测预处理
        # (Configure face detection preprocessing)
        self.face_det.config_preprocess()

    # run函数
    # (Run function)
    def run(self, input_np):
        """
        处理输入图像，执行人脸检测和姿态估计
        (Process input image, perform face detection and pose estimation)

        参数:
        input_np: 输入图像numpy数组
        (input_np: Input image numpy array)
```

```
        返回:
        人脸检测结果和对应的姿态估计结果
        (Face detection results and corresponding pose estimation results)
        """
        # 人脸检测
        # (Face detection)
        det_boxes = self.face_det.run(input_np)
        pose_res = []
        for det_box in det_boxes:
            # 对检测到的每一个人脸做人脸姿态估计
            # (Perform pose estimation on each detected face)
            self.face_pose.config_preprocess(det_box)
            R, eular = self.face_pose.run(input_np)
            pose_res.append((R, eular))
        return det_boxes, pose_res


    # 绘制人脸姿态角效果
    # (Draw face pose angle effect)
    def draw_result(self, pl, dets, pose_res):
        """
        绘制人脸姿态角效果
        (Draw face pose angle visualization)

        参数:
        dets: 人脸检测结果
        (dets: Face detection results)
        pose_res: 姿态估计结果
        (pose_res: Pose estimation results)
        """
        pl.osd_img.clear()
        if dets:
            # 创建一个空白图像用于绘制
            # (Create a blank image for drawing)
            draw_img_np = np.zeros((self.display_size[1], self.display_size[0],
4), dtype=np.uint8)
            draw_img = image.Image(self.display_size[0], self.display_size[1],
image.ARGB8888, alloc=image.ALLOC_REF, data=draw_img_np)
            # 设置线条颜色为红色 (BGRA格式)
            # (Set line color to red (BGRA format))
            line_color = np.array([255, 0, 0, 255], dtype=np.uint8)    #bgra
            for i, det in enumerate(dets):
                # （1）获取人脸姿态矩阵和欧拉角
                # (Get face pose matrix and Euler angles)
                projections, center_point = self.build_projection_matrix(det)
                R, euler = pose_res[i]
                # （2）遍历人脸投影矩阵的关键点，进行投影，并将结果画在图像上
                # (Traverse the key points of the face projection matrix, project
them, and draw the results on the image)
                first_points = []
                second_points = []
                for pp in range(8):
                    # 计算投影点坐标
                    # (Calculate projection point coordinates)
                    sum_x, sum_y = 0.0, 0.0
                    for cc in range(3):
                        sum_x += projections[pp][cc] * R[cc][0]
                        sum_y += projections[pp][cc] * (-R[cc][1])
```

```
                    center_x, center_y = center_point[0], center_point[1]
                    # 转换到显示坐标系
                    # (Convert to display coordinate system)
                    x = (sum_x + center_x) / self.rgb888p_size[0] *
self.display_size[0]
                    y = (sum_y + center_y) / self.rgb888p_size[1] *
self.display_size[1]
                    # 确保坐标在有效范围内
                    # (Ensure coordinates are within valid range)
                    x = max(0, min(x, self.display_size[0]))
                    y = max(0, min(y, self.display_size[1]))
                    if pp < 4:
                        first_points.append((x, y))
                    else:
                        second_points.append((x, y))
                # 绘制立方体线框
                # (Draw cube wireframe)
                first_points = np.array(first_points, dtype=np.float)
                aidemo.polylines(draw_img_np, first_points, True, line_color, 2,
8, 0)
                second_points = np.array(second_points, dtype=np.float)
                aidemo.polylines(draw_img_np, second_points, True, line_color,
2, 8, 0)
                # 连接前后面的对应顶点
                # (Connect corresponding vertices of front and back faces)
                for ll in range(4):
                    x0, y0 = int(first_points[ll][0]), int(first_points[ll][1])
                    x1, y1 = int(second_points[ll][0]), int(second_points[ll]
[1])
                    draw_img.draw_line(x0, y0, x1, y1, color=(255, 0, 0, 255),
thickness=2)
            # 将绘制结果复制到画布
            # (Copy drawing results to canvas)
            pl.osd_img.copy_from(draw_img)

    def build_projection_matrix(self, det):
        """
        构建人脸3D投影矩阵
        (Build 3D projection matrix for face)

        参数:
        det: 人脸检测结果，包含边界框信息
        (det: Face detection result containing bounding box information)

        返回:
        投影矩阵和中心点坐标
        (Projection matrix and center point coordinates)
        """
        x1, y1, w, h = map(lambda x: int(round(x, 0)), det[:4])
        # 计算边界框中心坐标
        # (Calculate bounding box center coordinates)
        center_x = x1 + w / 2.0
        center_y = y1 + h / 2.0
        # 定义后部（rear）和前部（front）的尺寸和深度
        # (Define dimensions and depth of rear and front)
        rear_width = 0.5 * w
        rear_height = 0.5 * h
        rear_depth = 0
```

```
        factor = np.sqrt(2.0)
        front_width = factor * rear_width
        front_height = factor * rear_height
        front_depth = factor * rear_width  # 使用宽度来计算深度，也可以使用高度，取决于需求
        # (Use width to calculate depth, can also use height depending on requirements)
        # 定义立方体的顶点坐标
        # (Define cube vertex coordinates)
        temp = [
            [-rear_width, -rear_height, rear_depth],
            [-rear_width, rear_height, rear_depth],
            [rear_width, rear_height, rear_depth],
            [rear_width, -rear_height, rear_depth],
            [-front_width, -front_height, front_depth],
            [-front_width, front_height, front_depth],
            [front_width, front_height, front_depth],
            [front_width, -front_height, front_depth]
        ]
        projections = np.array(temp)
        # 返回投影矩阵和中心坐标
        # (Return projection matrix and center coordinates)
        return projections, (center_x, center_y)
```

Perform face orientation detection

```
def exce_demo(pl):
    '''
    执行演示
    Execute demonstration

    参数/Parameters:
        pl: 图像处理/Image processing pipeline
    '''
    global fp

    # 获取显示参数/Get display parameters
    display_mode = pl.display_mode
    rgb888p_size = pl.rgb888p_size
    display_size = pl.display_size

    face_det_kmodel_path="/sdcard/kmodel/face_detection_320.kmodel"
    face_pose_kmodel_path="/sdcard/kmodel/face_pose.kmodel"
    anchors_path="/sdcard/utils/prior_data_320.bin"
    face_det_input_size=[320,320]
    face_pose_input_size=[120,120]
    confidence_threshold=0.5
    nms_threshold=0.2
    anchor_len=4200
    det_dim=4
    anchors = np.fromfile(anchors_path, dtype=np.float)
    anchors = anchors.reshape((anchor_len,det_dim))
```

```
 fp=FacePose(face_det_kmodel_path,face_pose_kmodel_path,det_input_size=face_det_
input_size,pose_input_size=face_pose_input_size,anchors=anchors,confidence_thres
hold=confidence_threshold,nms_threshold=nms_threshold,rgb888p_size=rgb888p_size,
display_size=display_size)


    try:
        while True:
            with ScopedTiming("total",1):
                img=pl.get_frame()
                det_boxes,pose_res=fp.run(img)
                fp.draw_result(pl,det_boxes,pose_res)
                pl.show_image()
                gc.collect()
    except Exception as e:
        print("人脸朝向识别功能退出/Face landmark detection exit")
    finally:
        fp.face_det.deinit()
        fp.face_pose.deinit()
```

A brief description of face orientation (posture) detection algorithm

Common application scenarios

Driving safety monitoring

- Detecting whether the driver is looking ahead
- Identify head shaking during fatigue driving
- Analyzing driver distraction

Human-computer interaction

- VR/AR headset tracking
- Eye tracking control
- Smart device viewing angle adaptation

Security monitoring

- Abnormal behavior identification
- Attention Analysis
- Crowd flow statistics

Smart Retail

- Customer attention analysis
- Shelf hot spot analysis
- Shopping behavior research

Photography Assist

- Automatic portrait composition
- Tips for best angles
- Optimization of group photo

Algorithm Overview

The basic principles of the face orientation detection algorithm can be summarized as follows:

Basic method types:

- Keypoint-based method: using facial keypoints to calculate Euler angles (pitch, yaw, roll)
- Direct regression method: Directly predicting posture angles using deep learning
- 3D model fitting: Align the 2D face with the 3D model to estimate the pose

Common technical implementations:

- PnP algorithm: solving posture using 3D key points and 2D projection relationships
- CNN regression: directly output three Euler angle values
- Multi-task learning: predicting key points and pose angles simultaneously

Key steps:

- Face detection and positioning
- Feature extraction (keypoints or deep features)
- Pose Angle Estimation
- Post-processing optimization

This example uses a hybrid method of deep learning + geometric transformation

In the deep learning part, a two-stage detection method is used: first use FaceDetApp to detect the face, then use FacePoseApp to estimate the pose, and then directly output the rotation matrix (3x3 matrix) through kmodel (deep learning model)

In the set transformation part, affine transformation is used in preprocessing to align and normalize the face area, and the rotation matrix is converted to Euler angles (pitch, yaw, roll) in postprocessing, and finally the projection matrix is used for 3D visualization.

The specific process is as follows:

Input image → Face detection → Affine transformation alignment → Deep learning model predicts rotation matrix → Matrix to Euler angle → 3D projection visualization