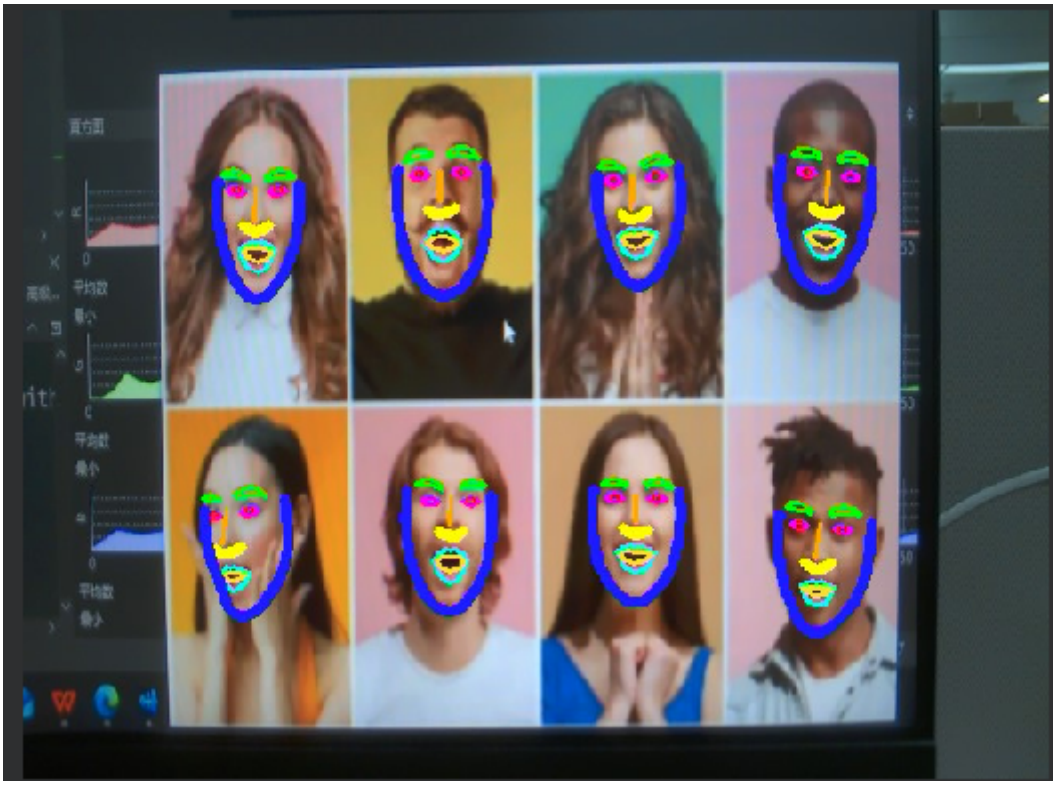
Facial key point recognition

Routine Experiment Effect

In this section, we will learn how to use K230 to realize the function of facial key point recognition.

The code in this section is in [Source code/07.Face/02.face_landmark.py]

After connecting to the IDE, run the sample code in this section and aim the K230 at the face. You can see that all the key points of the face are depicted in the middle of the screen.



Code Explanation

Code structure

Import related dependency classes

```
# 人脸关键点检测
# Complete code for facial landmark detection

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
import _thread

global flm
```

Define the face detection class

```
class FaceDetApp(AIBase):
    '''
    人脸检测应用类
    Face detection application class
    '''
    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
        Initialize face detection application

        参数/Parameters:
            kmodel_path: AI模型路径/AI model path
            model_input_size: 模型输入尺寸/Model input size
            anchors: 锚框配置/Anchor box configuration
            confidence_threshold: 置信度阈值/Confidence threshold
            nms_threshold: NMS阈值/NMS threshold
            rgb888p_size: 原始图像分辨率/Original image resolution
            display_size: 显示分辨率/Display resolution
            debug_mode: 调试模式/Debug mode
        '''
        super().__init__(kmodel_path,model_input_size,rgb888p_size,debug_mode)
        self.kmodel_path=kmodel_path
        self.model_input_size=model_input_size
        self.confidence_threshold=confidence_threshold
```

```
        self.nms_threshold=nms_threshold
        self.anchors=anchors
        self.rgb888p_size=[ALIGN_UP(rgb888p_size[0],16),rgb888p_size[1]]
        self.display_size=[ALIGN_UP(display_size[0],16),display_size[1]]
        self.debug_mode=debug_mode

        # 实例化AI2D对象,用于图像预处理/Initialize AI2D object for image
preprocessing
        self.ai2d=Ai2d(debug_mode)

 self.ai2d.set_ai2d_dtype(nn.ai2d_format.NCHW_FMT,nn.ai2d_format.NCHW_FMT,np.uin
t8, np.uint8)

    def config_preprocess(self,input_image_size=None):
        '''
        配置图像预处理
        Configure image preprocessing

        参数/Parameters:
            input_image_size: 输入图像尺寸/Input image size
        '''
        with ScopedTiming("set preprocess config",self.debug_mode > 0):
            ai2d_input_size=input_image_size if input_image_size else
self.rgb888p_size

            # 配置padding和resize/Configure padding and resize
            self.ai2d.pad(self.get_pad_param(), 0, [104,117,123])
            self.ai2d.resize(nn.interp_method.tf_bilinear,
nn.interp_mode.half_pixel)
            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]])

    def postprocess(self,results):
        '''
        后处理检测结果
        Post-process detection results

        参数/Parameters:
            results: 模型输出结果/Model output results

        返回/Returns:
            处理后的检测框/Processed detection boxes
        '''
        with ScopedTiming("postprocess",self.debug_mode > 0):
            res =
aidemo.face_det_post_process(self.confidence_threshold,self.nms_threshold,self.m
odel_input_size[0],self.anchors,self.rgb888p_size,results)
            if len(res)==0:
                return res
            else:
                return res[0]

    def get_pad_param(self):
        '''
        计算padding参数
        Calculate padding parameters

        返回/Returns:
```

```
            padding参数列表/List of padding parameters
        '''
        dst_w = self.model_input_size[0]
        dst_h = self.model_input_size[1]

        # 计算缩放比例/Calculate scaling ratio
        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

        # 计算新的尺寸和padding值/Calculate new dimensions and padding values
        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 detection application class

```
class FaceDetApp(AIBase):
    '''
    人脸检测应用类
    Face detection application class
    '''
    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
        Initialize face detection application

        参数/Parameters:
            kmodel_path: AI模型路径/AI model path
            model_input_size: 模型输入尺寸/Model input size
            anchors: 锚框配置/Anchor box configuration
            confidence_threshold: 置信度阈值/Confidence threshold
            nms_threshold: NMS阈值/NMS threshold
            rgb888p_size: 原始图像分辨率/Original image resolution
            display_size: 显示分辨率/Display resolution
            debug_mode: 调试模式/Debug mode
        '''
        super().__init__(kmodel_path,model_input_size,rgb888p_size,debug_mode)
        self.kmodel_path=kmodel_path
        self.model_input_size=model_input_size
        self.confidence_threshold=confidence_threshold
        self.nms_threshold=nms_threshold
        self.anchors=anchors
```

```
        self.rgb888p_size=[ALIGN_UP(rgb888p_size[0],16),rgb888p_size[1]]
        self.display_size=[ALIGN_UP(display_size[0],16),display_size[1]]
        self.debug_mode=debug_mode

        # 实例化AI2D对象,用于图像预处理/Initialize AI2D object for image
preprocessing
        self.ai2d=Ai2d(debug_mode)

 self.ai2d.set_ai2d_dtype(nn.ai2d_format.NCHW_FMT,nn.ai2d_format.NCHW_FMT,np.uin
t8, np.uint8)

    def config_preprocess(self,input_image_size=None):
        '''
        配置图像预处理
        Configure image preprocessing

        参数/Parameters:
            input_image_size: 输入图像尺寸/Input image size
        '''
        with ScopedTiming("set preprocess config",self.debug_mode > 0):
            ai2d_input_size=input_image_size if input_image_size else
self.rgb888p_size

            # 配置padding和resize/Configure padding and resize
            self.ai2d.pad(self.get_pad_param(), 0, [104,117,123])
            self.ai2d.resize(nn.interp_method.tf_bilinear,
nn.interp_mode.half_pixel)
            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]])

    def postprocess(self,results):
        '''
        后处理检测结果
        Post-process detection results

        参数/Parameters:
            results: 模型输出结果/Model output results

        返回/Returns:
            处理后的检测框/Processed detection boxes
        '''
        with ScopedTiming("postprocess",self.debug_mode > 0):
            res =
aidemo.face_det_post_process(self.confidence_threshold,self.nms_threshold,self.m
odel_input_size[0],self.anchors,self.rgb888p_size,results)
            if len(res)==0:
                return res
            else:
                return res[0]

    def get_pad_param(self):
        '''
        计算padding参数
        Calculate padding parameters

        返回/Returns:
            padding参数列表/List of padding parameters
        '''
```

```
        dst_w = self.model_input_size[0]
        dst_h = self.model_input_size[1]

        # 计算缩放比例/Calculate scaling ratio
        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

        # 计算新的尺寸和padding值/Calculate new dimensions and padding values
        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 key point detection preprocessing class

```
class FaceLandMark:
    '''
    人脸关键点检测主类
    Main class for facial landmark detection
    '''
    def
__init__(self,face_det_kmodel,face_landmark_kmodel,det_input_size,landmark_input
_size,anchors,confidence_threshold=0.25,nms_threshold=0.3,rgb888p_size=
[1920,1080],display_size=[1920,1080],debug_mode=0):
        '''
        初始化人脸关键点检测
        Initialize facial landmark detection

        参数/Parameters:
            face_det_kmodel: 人脸检测模型路径/Face detection model path
            face_landmark_kmodel: 关键点检测模型路径/Landmark detection model path
            det_input_size: 检测模型输入尺寸/Detection model input size
            landmark_input_size: 关键点模型输入尺寸/Landmark model input size
            anchors: 锚框配置/Anchor box configuration
            confidence_threshold: 置信度阈值/Confidence threshold
            nms_threshold: NMS阈值/NMS threshold
            rgb888p_size: 原始图像分辨率/Original image resolution
            display_size: 显示分辨率/Display resolution
            debug_mode: 调试模式/Debug mode
        '''
        self.face_det_kmodel=face_det_kmodel
        self.face_landmark_kmodel=face_landmark_kmodel
        self.det_input_size=det_input_size
        self.landmark_input_size=landmark_input_size
        self.anchors=anchors
```

```
        self.confidence_threshold=confidence_threshold
        self.nms_threshold=nms_threshold
        self.rgb888p_size=[ALIGN_UP(rgb888p_size[0],16),rgb888p_size[1]]
        self.display_size=[ALIGN_UP(display_size[0],16),display_size[1]]
        self.debug_mode=debug_mode

        # 定义人脸不同部位的关键点序列/Define landmark sequences for different facial
parts
        self.dict_kp_seq = [
            [43, 44, 45, 47, 46, 50, 51, 49, 48], # 左眉毛/left_eyebrow
            [97, 98, 99, 100, 101, 105, 104, 103, 102], # 右眉毛/right_eyebrow
            [35, 36, 33, 37, 39, 42, 40, 41], # 左眼/left_eye
            [89, 90, 87, 91, 93, 96, 94, 95], # 右眼/right_eye
            [34, 88], # 瞳孔/pupil
            [72, 73, 74, 86], # 鼻梁/bridge_nose
            [77, 78, 79, 80, 85, 84, 83], # 鼻翼/wing_nose
            [52, 55, 56, 53, 59, 58, 61, 68, 67, 71, 63, 64], # 外唇/out_lip
            [65, 54, 60, 57, 69, 70, 62, 66], # 内唇/in_lip
            [1, 9, 10, 11, 12, 13, 14, 15, 16, 2, 3, 4, 5, 6, 7, 8, 0, 24, 23,
22, 21, 20, 19, 18, 32, 31, 30, 29, 28, 27, 26, 25, 17] # 轮廓/basin
        ]

        # 定义不同部位的显示颜色(ARGB)/Define display colors for different parts
(ARGB)
        self.color_list_for_osd_kp = [
            (255, 0, 255, 0),
            (255, 0, 255, 0),
            (255, 255, 0, 255),
            (255, 255, 0, 255),
            (255, 255, 0, 0),
            (255, 255, 170, 0),
            (255, 255, 255, 0),
            (255, 0, 255, 255),
            (255, 255, 220, 50),
            (255, 30, 30, 255)
        ]

        # 实例化检测和关键点模型/Initialize detection and landmark models

 self.face_det=FaceDetApp(self.face_det_kmodel,model_input_size=self.det_input_s
ize,anchors=self.anchors,confidence_threshold=self.confidence_threshold,nms_thre
shold=self.nms_threshold,rgb888p_size=self.rgb888p_size,display_size=self.displa
y_size,debug_mode=0)

 self.face_landmark=FaceLandMarkApp(self.face_landmark_kmodel,model_input_size=s
elf.landmark_input_size,rgb888p_size=self.rgb888p_size,display_size=self.display
_size)
        self.face_det.config_preprocess()

    def run(self,input_np):
        '''
        运行人脸关键点检测
        Run facial landmark detection

        参数/Parameters:
            input_np: 输入图像/Input image

        返回/Returns:
```

```
            det_boxes: 检测框/Detection boxes
            landmark_res: 关键点结果/Landmark results
        '''
        det_boxes=self.face_det.run(input_np)
        landmark_res=[]
        for det_box in det_boxes:
            self.face_landmark.config_preprocess(det_box)
            res=self.face_landmark.run(input_np)
            landmark_res.append(res)
        return det_boxes,landmark_res

    def draw_result(self,pl,dets,landmark_res):
        '''
        绘制检测结果
        Draw detection results

        参数/Parameters:
            pl: 图像处理Pipeline/Image processing pipeline
            dets: 检测框/Detection boxes
            landmark_res: 关键点结果/Landmark results
        '''
        pl.osd_img.clear()
        if dets:
            # 创建绘图缓冲/Create drawing buffer
            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)

            for pred in landmark_res:
                # 绘制每个检测到的人脸的关键点/Draw landmarks for each detected face
                for sub_part_index in range(len(self.dict_kp_seq)):
                    sub_part = self.dict_kp_seq[sub_part_index]
                    face_sub_part_point_set = []

                    # 收集关键点坐标/Collect landmark coordinates
                    for kp_index in range(len(sub_part)):
                        real_kp_index = sub_part[kp_index]
                        x, y = pred[real_kp_index * 2], pred[real_kp_index * 2 +
1]
                        x = int(x * self.display_size[0] //
self.rgb888p_size[0])
                        y = int(y * self.display_size[1] //
self.rgb888p_size[1])
                        face_sub_part_point_set.append((x, y))

                    # 根据不同部位选择不同的绘制方式/Choose different drawing methods
for different parts
                    if sub_part_index in (9, 6):
                        color =
np.array(self.color_list_for_osd_kp[sub_part_index],dtype = np.uint8)
                        face_sub_part_point_set =
np.array(face_sub_part_point_set)
                        aidemo.polylines(draw_img_np,
face_sub_part_point_set,False,color,5,8,0)
                    elif sub_part_index == 4:
                        color = self.color_list_for_osd_kp[sub_part_index]
                        for kp in face_sub_part_point_set:
```

```
                            x,y = kp[0],kp[1]
                            draw_img.draw_circle(x,y ,2, color, 1)
                    else:
                            color =
np.array(self.color_list_for_osd_kp[sub_part_index],dtype = np.uint8)
                            face_sub_part_point_set =
np.array(face_sub_part_point_set)
                            aidemo.contours(draw_img_np,
face_sub_part_point_set,-1,color,2,8)
            pl.osd_img.copy_from(draw_img)
```

Define the face key point detection application class

```
class FaceLandMarkApp(AIBase):
    '''
    人脸关键点检测应用类
    Facial landmark detection application class
    '''
    def __init__(self,kmodel_path,model_input_size,rgb888p_size=
[1920,1080],display_size=[1920,1080],debug_mode=0):
        '''
        初始化人脸关键点检测应用
        Initialize facial landmark detection application

        参数/Parameters:
            kmodel_path: AI模型路径/AI model path
            model_input_size: 模型输入尺寸/Model input size
            rgb888p_size: 原始图像分辨率/Original image resolution
            display_size: 显示分辨率/Display resolution
            debug_mode: 调试模式/Debug mode
        '''
        super().__init__(kmodel_path,model_input_size,rgb888p_size,debug_mode)
        self.kmodel_path=kmodel_path
        self.model_input_size=model_input_size
        self.rgb888p_size=[ALIGN_UP(rgb888p_size[0],16),rgb888p_size[1]]
        self.display_size=[ALIGN_UP(display_size[0],16),display_size[1]]
        self.debug_mode=debug_mode
        self.matrix_dst=None

        # 实例化AI2D对象/Initialize AI2D object
        self.ai2d=Ai2d(debug_mode)

 self.ai2d.set_ai2d_dtype(nn.ai2d_format.NCHW_FMT,nn.ai2d_format.NCHW_FMT,np.uin
t8, np.uint8)
•
    def config_preprocess(self,det,input_image_size=None):
        '''
        配置图像预处理
        Configure image preprocessing

        参数/Parameters:
            det: 人脸检测框/Face detection box
            input_image_size: 输入图像尺寸/Input image size
        '''
        with ScopedTiming("set preprocess config",self.debug_mode > 0):
```

```
            ai2d_input_size=input_image_size if input_image_size else
self.rgb888p_size

            # 获取仿射变换矩阵/Get affine transformation matrix
            self.matrix_dst = self.get_affine_matrix(det)
            affine_matrix = [self.matrix_dst[0][0],self.matrix_dst[0]
[1],self.matrix_dst[0][2],
                             self.matrix_dst[1][0],self.matrix_dst[1]
[1],self.matrix_dst[1][2]]

            # 配置仿射变换/Configure affine transformation
            self.ai2d.affine(nn.interp_method.cv2_bilinear,0, 0, 127,
1,affine_matrix)
            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]])

    def postprocess(self,results):
        '''
        后处理关键点检测结果
        Post-process landmark detection results

        参数/Parameters:
            results: 模型输出结果/Model output results

        返回/Returns:
            处理后的关键点坐标/Processed landmark coordinates
        '''
        with ScopedTiming("postprocess",self.debug_mode > 0):
            pred=results[0]
            half_input_len = self.model_input_size[0] // 2

            # 转换关键点坐标/Transform landmark coordinates
            pred = pred.flatten()
            for i in range(len(pred)):
                pred[i] += (pred[i] + 1) * half_input_len

            # 获取逆变换矩阵/Get inverse transformation matrix
            matrix_dst_inv = aidemo.invert_affine_transform(self.matrix_dst)
            matrix_dst_inv = matrix_dst_inv.flatten()

            # 对每个关键点进行逆变换/Apply inverse transform to each landmark
            half_out_len = len(pred) // 2
            for kp_id in range(half_out_len):
                old_x = pred[kp_id * 2]
                old_y = pred[kp_id * 2 + 1]
                new_x = old_x * matrix_dst_inv[0] + old_y * matrix_dst_inv[1] +
matrix_dst_inv[2]
                new_y = old_x * matrix_dst_inv[3] + old_y * matrix_dst_inv[4] +
matrix_dst_inv[5]
                pred[kp_id * 2] = new_x
                pred[kp_id * 2 + 1] = new_y
            return pred

    def get_affine_matrix(self,bbox):
        '''
        获取仿射变换矩阵
        Get affine transformation matrix
```

```
        参数/Parameters:
            bbox: 人脸检测框/Face detection box

        返回/Returns:
            仿射变换矩阵/Affine transformation matrix
        '''
        with ScopedTiming("get_affine_matrix", self.debug_mode > 1):
            x1, y1, w, h = map(lambda x: int(round(x, 0)), bbox[:4])

            # 计算缩放比例和中心点/Calculate scale ratio and center point
            scale_ratio = (self.model_input_size[0]) / (max(w, h) * 1.5)
            cx = (x1 + w / 2) * scale_ratio
            cy = (y1 + h / 2) * scale_ratio
            half_input_len = self.model_input_size[0] / 2

            # 构建仿射矩阵/Build affine matrix
            matrix_dst = np.zeros((2, 3), dtype=np.float)
            matrix_dst[0, 0] = scale_ratio
            matrix_dst[0, 1] = 0
            matrix_dst[0, 2] = half_input_len - cx
            matrix_dst[1, 0] = 0
            matrix_dst[1, 1] = scale_ratio
            matrix_dst[1, 2] = half_input_len - cy
            return matrix_dst
```

Perform facial landmark detection

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

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

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

    # 加载模型和配置/Load models and configurations
    face_det_kmodel_path="/sdcard/examples/kmodel/face_detection_320.kmodel"
    face_landmark_kmodel_path="/sdcard/examples/kmodel/face_landmark.kmodel"
    anchors_path="/sdcard/examples/utils/prior_data_320.bin"
    face_det_input_size=[320,320]
    face_landmark_input_size=[192,192]
    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))
```

```
    # 创建人脸关键点检测实例/Create facial landmark detection instance

 flm=FaceLandMark(face_det_kmodel_path,face_landmark_kmodel_path,det_input_size=
face_det_input_size,landmark_input_size=face_landmark_input_size,anchors=anchors
,confidence_threshold=confidence_threshold,nms_threshold=nms_threshold,rgb888p_s
ize=rgb888p_size,display_size=display_size)

    try:
        while True:
            # 主循环:获取图像-推理-显示/Main loop: get image - inference - display
            img=pl.get_frame()
            det_boxes,landmark_res=flm.run(img)
            flm.draw_result(pl,det_boxes,landmark_res)
            pl.show_image()
            gc.collect()
            time.sleep_us(10)
    except Exception as e:
        print("人脸关键点检测功能退出/Face landmark detection exit")
    finally:
        # 清理资源 / Clean up resources
        flm.face_det.deinit()
        flm.face_landmark.deinit()

def exit_demo():
    '''
    退出演示
    Exit demonstration
    '''
    global flm
    flm.face_det.deinit()
    flm.face_landmark.deinit()


if __name__=="__main__":
    # 主程序入口 / Main program entry
    rgb888p_size=[1920,1080]
    display_size=[640,480]
    display_mode="lcd"

    # 创建和初始化图像处理Pipeline/Create and initialize image processing pipeline
    pl = PipeLine(rgb888p_size=rgb888p_size, display_size=display_size,
display_mode=display_mode)
    pl.create()
    exce_demo(pl)
    pl.destroy()
```

A brief introduction to facial key point detection algorithm

Common application scenarios

Face attribute analysis

- Feature point extraction for expression recognition
- Facial feature localization for age estimation
- Gender judgment feature reference

Image processing and beautification

- Precise microdermabrasion positioning
- Facial features adjustment (bigger eyes, thinner face, etc.)
- Beauty effects overlay
- Smart filter effects

Animation and special effects

- Expression animation driver
- Face replacement positioning
- 3D Avatar Mapping
- Dynamic sticker alignment

Face pose analysis

- Head rotation angle estimation
- Determine the direction of sight
- Gaze Prediction

Image quality assessment

- Face clarity detection
- Lighting condition assessment
- Determination of occlusion degree

Algorithm Overview

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

Basic process

- First perform face detection to locate the face area
- Detect key point locations within the face area
- Usually 68 or 106 key points are detected, including eyes, eyebrows, nose, mouth, facial contour, etc.

Main algorithm method

- Traditional methods: ASM (Active Shape Model), AAM (Active Appearance Model)
- Deep learning methods: CNN regression, heat map prediction, etc.
- Cascade regression method: step-by-step iterative optimization from initial shape

Key technical points

- Using multi-scale feature extraction
- Introducing shape constraints to ensure reasonable distribution of key points
- Using data augmentation to improve robustness
- Design loss function to balance accuracy and speed

The face detection in this routine is based on the face key point detection of the deep learning CNN regression method, which is also one of the most mainstream technical routes for face key point detection. First use FaceDetApp to detect the face, obtain the face frame, and then use FaceLandMarkApp to locate the key points.

The main steps in the key point detection stage are:

- Align and normalize face regions using affine transformation
- Keypoint prediction via deep neural network (loaded via kmodel)
- Post-process the prediction results, including coordinate transformation and inverse affine transformation