

人脸识别应用编程接口（SDK）

概述

历史分支

开发环境

SDK 集成说明

接口说明

1、SDK接入

1.1 SDK初始化

1.2 激活 License

1.3 获取设备唯一编码

1.4 离线授权

2、SDK 启动

2.1 同步方式启动SDK

2.2 异步方式启动SDK

2.3 SDK重载config.json参数，通过指定config.json文件存放目录

2.4 SDK重载config.json参数，通过指定config.json文件内容

2.5 SDK重载config.json参数，通过指定config.json文件内容，异步方式

3、人脸实时检测与跟踪

3.1 人脸实时数据送检

3.2 人脸实时跟踪回调

3.3 实时人脸属性检测

3.4 人脸实时搜索

3.5 指定图片文件提取人脸特征值

3.6 图片Bitmap 进行特征值提取

3.7 获取人脸的眼睛、嘴巴、鼻子等等landmarks坐标

3.8 人脸特征比对

4、人脸库管理

4.1 人脸入库

4.2 人脸库删除

4.3 人脸库更新

4.4 人脸批量添加特征值

4.5 人脸数据清除

5、参数说明

5.1 图像数据 ArcternImage

5.2 人脸框 ArcternRect

5.3 人脸属性 ArcternAttribute

5.4 人脸属性 ArcternAttrResult

5.5 人脸属性 ArcternFaceAttrTypeEnum

人脸识别应用编程接口（SDK）

概述

人脸识别SDK，包含了人脸检测、人脸识别、人脸属性检测等相关功能,人脸 SDK在 SDK 授权后可以根据自身的相关业务需求结合 SDK 的相关接口灵活的进行上层软件应用的开发。

适用场景：

- 设备无法连接到物联网，处于离线状态。
- 公安内网等专网
- 网络速度缓慢，避免占用过多带宽等环境

双目摄像头:

IR Camerald = Camera.CameraInfo.CAMERA_FACING_FRONT

RGB Camerald = Camera.CameraInfo.CAMERA_FACING_BACK

历史分支

日期	版本	说明	修改人
2021-06-01	1.3	提升核心算法的运行效率；解决客户反馈的问题；	肖大涛,宋建峰
2020-12-21	1.2	新增同步接口：人脸检测、特征值搜索,人脸属性	肖大涛,宋建峰
2020-07-15	1.1	V2版	肖大涛,宋建峰
2020-05-21	1.0	初版完成	宋建峰

开发环境

1. Android Studio >=3.6.2
2. Gradle Version: 3.6.2
3. Gradle Plugin Version: 3.6.2
4. SDK Tool >=25.2.3

SDK 集成说明

- 1.将 Demo工程 faceEngineYtlf/libs/ 目录下的 aar包拷贝到 AS 工程对应的 libs 文件夹下；
- 2.将 Demo工程 faceEngineYtlf/src/main/assets 目录下 ytlf_v2 文件夹，拷贝到 AS 工程对应的 src/main/assets文件夹下；

详细请参照技术案例Demo工程配置和 doc/FaceEngine集成说明截图.png；

接口说明

1、SDK接入

1.1 SDK初始化

实例YTLFFaceManager时，需要指明本地SD卡文件存放目录rootPath，例

如："/sdcard/firefly/"；

首次启动SDK时，会检查本地SD卡是否存在models 和 license公钥等文件，如果没有，那么默认会从App assets 目录下拷贝必需文件到rootPath目录；

```
// 指定 本地SD卡文件存放目录  
YTLFFaceManager.getInstance().initPath(String rootPath);
```

1.2 激活 License

初始化时会检查本地是否存在密钥,若无则进行在线激活,激活结束后会回调激活状态,分为同步和异步两种激活方式;

```
// 异步方式
YTLFFaceManager.getInstance().initLicenseByAsync(String apiKey, new InitLicenseListener(){
    @Override
    public void onLicenseSuccess() {
        toast("激活成功");
    }

    @Override
    public void onLicenseFail(final int code,final String msg) {
        toast("激活失败");
    }
});

// 同步方式 true 表示激活成功
boolean result = YTLFFaceManager.getInstance().initLicense(String apiKey);
```

1.3 获取设备唯一编码

获取当前设备的 signature , 设备的唯一编码;

```
/*
String :返回当前设备的 signature 即设备当前的唯一编码
*/
String signature = YTLFFaceManager.getInstance().getSignature();
```

1.4 离线授权

如果设备无法连接互联网,那么可以通过SDK中的接口getSignature()获取设备的signature,每台硬件设备对应唯一signature,例如: 3418eb8df575c92527b7ffc8cdaf34k9;

然后将signature的32位字符串发给平台进行授权,生成license文件,再放到ytlf_v2/license目录,例如: sdcard/firefly/ytlf_v2/license/license;

2、 SDK 启动

启动SDK时,会检测运行环境和初始化SDK,有同步和异步两种启动方式;

2.1 同步方式启动SDK

```
/*
通过FACE_PATH 判断出 config_path, 即Config.json 文件存放目录
eg: sdcard/firefly/ytlf_v2/config.json"

Int:0 表示成功、1 表示失败
```

```

    */
    int result = YTLFFaceManager.getInstance().startFaceSDK();

    /*
        config_json 指明config.json 内容

        Int:0 表示成功、1 表示失败
    */
    int result = YTLFFaceManager.getInstance().startFaceSDKByConfigJson(String config_json);

```

2.2 异步方式启动SDK

为防止初始化并启动SDK时占用过多时间，造成阻塞主线程，可以使用异步方式启动SDK；

```

    /*
        config_json 指明config.json 内容
        runSDKCallback 异步启动后的回调方法
    */
    YTLFFaceManager.getInstance().startFaceSDKByAsynchronous(String config_json, new
    RunSDKCallback(){
        @Override
        public void onRunSDKListener(int i) { //Int:0 表示成功、1 表示失败
        }
    });

```

2.3 SDK重载config.json参数，通过指定config.json文件存放目录

SDK重新加载config.json文件，从而实现人脸参数重载；

```

    /*
        configPath config.json文件存放目录； eg:/sdcard/firefly/ytlf_v2/config.json
        Int:0 表示成功、1 表示失败
    */
    int result = YTLFFaceManager.getInstance().reloadConfig(String configPath);

```

2.4 SDK重载config.json参数，通过指定config.json文件内容

SDK重新加载config.json文件内容，从而实现人脸参数重载；

```

    /*
        configJson config.json 文件内容
        Int:0 表示成功、1 表示失败
    */
    int result = YTLFFaceManager.getInstance().reloadConfigJson(String configJson);

```

2.5 SDK重载config.json参数，通过指定config.json文件内容，异步方式

SDK重新加载config.json文件内容，从而实现人脸参数重载；
为防止重新加载时占用过多时间，造成阻塞主线程，可以使用异步方式；

```
/*
    configJson config.json 文件内容
    reloadSDKCallback 异步回调接口
*/
YTLFFaceManager.getInstance().reloadConfigJsonByAsync(String configJson, ReloadSDKCallback
reloadSDKCallback);
```

3、人脸实时检测与跟踪

3.1 人脸实时数据送检

根据传入检测器的数据进行人脸检测与跟踪,并实时返回人脸检测的结果以及人脸跟踪的结果,注意:输入人脸检测器的数据人脸方向应为正向,即人脸角度应为 0 度而非其他的角度;

目前通过输送 RGB 以及 IR 视频流来进行人脸检测跟踪以及活体的相关检测
当不需要打开 IR 摄像头或不需要活体检测时 IR 数据参数可传 RGB 参数,不可传 NULL;

```
//从摄像头回调函数中获取视频流，实时输 RGB 视频流以及 IR 视频流
YTLFFaceManager.getInstance().doDelivery(ArcTernImage img_rgb, ArcTernImage img_ir)

//设置RGB和IR监听回调函数
YTLFFaceManager.getInstance().setOnDetectCallBack(new DetectCallBack() {
    // RGB 回调:
    @Override
    public void onDetectListener(ArcTernImage arcTernImage, ArcTernRect[] arcTernRects, float[]
confidences) {

    }

    // IR 回调:
    @Override
    public void onLivingDetectListener(ArcTernImage arcTernImage, ArcTernRect[] arcTernRects, float[]
confidences) {

    }
});

/*参数说明:
    arcTernImage RGB/IR 检测的人脸图数据
    arcTernRects RGB/IR 检测到的人脸框集合
    confidences RGB/IR 检测到每个人脸的置信度
*/
```

3.2 人脸实时跟踪回调

如果回调接口有数据返回，那么说明实时数据正在检测跟踪；

```
//设置人脸实时检测跟踪回调
YTLFFaceManager.getInstance().setOnTrackCallBack(new TrackCallBack() {

/*人脸实时检测跟踪
    arcternImage 检测的人脸图数据
    trackIds 图像中人脸的跟踪 ID
    arcternRects 检测到的人脸位置
*/
    @Override
    public void onTrackListener(ArcternImage arcternImage, long[] trackIds, ArcternRect[] arcternRects) {

    }
});

//人脸图片检测 同步方式
Rect rect = YTLFFaceManager.getInstance().doDetect(Bitmap bitmap)
```

3.3 实时人脸属性检测

进行实时人脸检测跟踪时，添加人脸属性检测回调方法，可以实时接收人脸属性，包括活体检测值、质量、人脸角度、口罩、图像颜色。

```
//设置实时人脸属性检测回调
YTLFFaceManager.getInstance().setOnAttributeCallBack(new AttributeCallBack() {

/*人脸属性监听回调：
    arcternImage 检测的人脸图数据
    arcternRects 检测到的人脸位置
    trackIds 图像中人脸的跟踪 ID
    arcternAttribute 图像中所有人脸的所有属性
*/
    @Override
    public void onAttributeListener(ArcternImage arcternImage, long[] trackIds, ArcternRect[] arcternRects, ArcternAttribute[][] arcternAttributes, int[] landmarks) {
        StringBuilder s = new StringBuilder();
        for (int i = 0; i < arcternRects.length; i++) {
            for (int j = 0; j < arcternAttributes[i].length; j++) {
                ArcternAttribute attr = arcternAttributes[i][j];
                switch (j) {
                    case ArcternAttribute.ArcternFaceAttrTypeEnum.POSE_PITCH:
                        s.append("人脸角度:\n以x轴为中心,脸部上下俯仰角度:").append(attr.confidence);
                        break;
                    case ArcternAttribute.ArcternFaceAttrTypeEnum.POSE_YAW:
                        s.append("\n以y轴为中心,脸部左右旋转角度:").append(attr.confidence);
                        break;
                    case ArcternAttribute.ArcternFaceAttrTypeEnum.POSE_ROLL:
                        s.append("\n以中心点为中心, x-y平面旋转角度:").append(attr.confidence);
                        break;
                    case ArcternAttribute.ArcternFaceAttrTypeEnum.QUALITY:
                        s.append("\n人脸质量:").append(attr.confidence);

```

```

        break;

    case ArcternAttribute.ArcternFaceAttrTypeEnum.LIVENESS_IR:
        if (attr.label != -1) {
            if (attr.confidence >= 0.5) {
                s.append("\n活体检测:活体 ").append(attr.confidence);
            } else {
                s.append("\n活体检测:非活体 ").append(attr.confidence);
            }
        }
        break;

    case ArcternAttribute.ArcternFaceAttrTypeEnum.IMAGE_COLOR:
        if (attr.label == ArcternAttribute.LabelFaceImgColor.COLOR) {
            s.append("\n彩图 ").append(attr.confidence);
        } else {
            s.append("\n黑白图 ").append(attr.confidence);
        }
        break;

    case ArcternAttribute.ArcternFaceAttrTypeEnum.FACE_MASK:
        if (attr.label == ArcternAttribute.LabelFaceMask.MASK) {
            s.append("\n口罩 ").append(attr.confidence);
        } else {
            s.append("\n未戴口罩 ").append(attr.confidence);
        }
        break;
    }
}
}
});

/**
 * 通过人脸框检测当前人脸属性 同步方式
 * @param bitmap
 * @param mask 需要检测的人脸属性 eg:ArcternAttrResult.ARCTERN_FACE_ATTR_MASK_ALL
 * @return
 */
ArcternAttrResult arcternAttrResult = YTLFFaceManager.getInstance().getFaceAttrs(Bitmap bitmap, int
mask)

```

3.4 人脸实时搜索

根据传入检测器的数据进行实时搜索，从数据库中搜索相似度大于设置相似度最高的一个 ID,通过 ID 可获取到识别的人脸以及其相关信息；

```

//设置人脸实时搜索回调
YTLFFaceManager.getInstance().setOnSearchCallBack(new SearchCallBack() {

    /*搜索回调：
    arcternImage 检测的人脸图数据
    trackIds 图像中人脸的跟踪 ID
    arcternRects 检测到的人脸位置
    */

```

```

searchId_list 图像中识别人脸的 id 集合
*/
@Override
public void onSearchListener(ArcternImage arcternImage, long[] trackId_list, ArcternRect[] arcternRects,
long[] searchId_list, int[] landmarks, float[] socre) {
    if (searchId_list.length > 0 && searchId_list[0] != -1) {
        Person person = DBHelper.get(searchId_list[0]);
        if (person != null) {
            //通过 ID 可获取到识别的人脸以及其相关信息;
        }
    } else {
        // 人脸不存在;
    }
}
}

/**
 * 搜索人脸信息 同步方式
 * 过特征值搜索数据库中比对分数最高的人脸
 * @param feature
 * @return
 */
long id = doSearch(byte[] feature);

```

3.5 指定图片文件提取人脸特征值

根据检测的人脸提取人脸特征值，可以用于人脸比对；

```

/*指定图片文件提取人脸特征值：
    imagePath 图片地址
    etractCallBack 人脸特征提取回调
*/
YTLFFaceManager.getInstance().doFeature(imagePath, new ExtractCallBack())

/*
    arcternImage 提取特征值的图片
    bytes 多个人脸的人脸特征值集合
    arcternRects 人脸检测结果集合
*/
@Override
public void onExtractFeatureListener(ArcternImage arcternImage, byte[][] bytes, ArcternRect[]
arcternRects) {

    if (features.length > 0) {
        debugLog("bitmapFeature.length: " + features[0].length);
    } else {
        Tools.debugLog("特征值为空!!!");
    }
}
});

/*提取人脸特征值 同步方式
    bitmap 图片bitmap
*/
ArcternFeatureResult featureResult = YTLFFaceManager.getInstance().doFeature(bitmap)

```


3.6 图片Bitmap 进行特征值提取

根据人脸图片Bitmap 提取人脸特征值，可以用于人脸比对；

```
/*指定人脸图片Bitmap提取人脸特征值：
   bitmap 图片Bitmap数据
   extractCallBack 人脸特征提取回调
*/
YTLFFaceManager.getInstance().doFeature(Bitmap bitmap, new ExtractCallBack())

/*
   arcternImage 提取特征值的图片
   bytes 多个人脸的人脸特征值集合
   arcternRects 人脸检测结果集合
*/
@Override
public void onExtractFeatureListener(ArcternImage arcternImage, byte[][] bytes, ArcternRect[] arcternRects) {

    if (features.length > 0) {
        debugLog("bitmapFeature.length: " + features[0].length);
    } else {
        Tools.debugLog("特征值为空!!!");
    }
}
});
```

3.7 获取人脸的眼睛、嘴巴、鼻子等等landmarks坐标

根据人脸ArcternImage提取人脸特征值，可以用于获取人脸的眼睛、嘴巴、鼻子等等landmarks坐标；

```
/*
   指定ArcternImage提取人脸特征值，包含landmarks
*/
ArcternAttrResult ArcternAttrResult = YTLFFaceManager.getInstance().doFeature(ArcternImage arcternImage)
```

3.8 人脸特征比对

通过两个特征值进行比对,得出两个人脸特征值的相似度；

```
/*
    feature1 第一个人脸特征值
    Feature2 第二个人脸特征值

    return float :返回人脸比对相似度
*/
float result = YTLFFaceManager.getInstance().doFeature(byte[] feature1, byte[] feature2)
```

4、人脸库管理

4.1 人脸入库

将提取出来的人脸特征值和相关 Id，添加到SDK人脸数据库；

```
/*
    id 人脸 ID
    feature 人脸的特征值

    Int:0 表示执行成功、1 表示执行失败
*/
int result = YTLFFaceManager.getInstance().dataBaseAdd(long id, byte[] feature)
```

4.2 人脸库删除

根据指定的 Id，删除SDK人脸数据库的人脸信息；

```
/*
    id 人脸 ID

    Int:0 表示执行成功、1 表示执行失败
*/
int result = YTLFFaceManager.getInstance().dataBaseDelete(long id)
```

4.3 人脸库更新

根据指定的 Id，更新SDK人脸数据库的人脸信息；

```
/*
    id 人脸 ID
    feature 人脸的特征值

    Int:0 表示执行成功、1 表示执行失败
*/
int result = YTLFFaceManager.getInstance().dataBaseUpdate(long id, byte[] feature)
```

4.4 人脸批量添加特征值

根据指定的多个 Id，批量更新SDK人脸数据库对应的多个人脸信息；

```
/*
    id 人脸 ID 数组
    feature 多个人脸特征值数组

    Int:0 表示执行成功、1 表示执行失败
*/
int result = YTLFFaceManager.getInstance().dataBaseAdd(long[] id, byte[][] feature)
```

4.5 人脸数据清除

删除SDK人脸数据库的所有人脸信息；

```
/*
    Int:0 表示执行成功、1 表示执行失败
*/
int result = YTLFFaceManager.getInstance().dataBaseClear()
```

5、参数说明

5.1 图像数据 ArcTernImage

```
public static final int ARCTERN_IMAGE_FORMAT_GRAY = 0; //灰度图
public static final int ARCTERN_IMAGE_FORMAT_RGB = 1; //RGB 格式图
public static final int ARCTERN_IMAGE_FORMAT_NV21 = 6; //NV21 格式

public static final int ARCTERN_IMAGE_FORMAT_YUV420sp = ARCTERN_IMAGE_FORMAT_NV21; //YUV流
NV21 格式Camera

public static final int ARCTERN_IMAGE_FORMAT_YUV420sp_HS =
ARCTERN_IMAGE_FORMAT_YUV420sp; //YUV420 海思用
```

参数	类型	描述
FrameID	long	Config.json 文件内容
Image_format	int	图像数据格式
width	int	图像宽度
height	int	图像高度
gdata	Byte[]	图像数据

5.2 人脸框 ArcternRect

参数	类型	描述
x	int	人脸左上角X坐标
y	int	人脸左上角Y坐标
width	int	人脸框宽度
height	int	人脸框高度

5.3 人脸属性 ArcternAttribute

参数	类型	描述
Label	String	属性结果，类型如下ArcternFaceAttrTypeEnum
confidence	Float	属性结果分数

5.4 人脸属性 ArcternAttrResult

参数	类型	描述
arcternAttributes	ArcternAttribute[] []	属性结果，二维数组，类型如上5.3 人脸属性 ArcternAttribute
landmark	Int[]	landmark坐标

5.5 人脸属性 ArcternFaceAttrTypeEnum

```
ArcternAttribute.ArcternFaceAttrTypeEnum.GENDER = 0;  
ArcternAttribute.ArcternFaceAttrTypeEnum.AGE = 1;  
ArcternAttribute.ArcternFaceAttrTypeEnum.POSE_PITCH = 2;  
ArcternAttribute.ArcternFaceAttrTypeEnum.POSE_YAW = 3;  
ArcternAttribute.ArcternFaceAttrTypeEnum.POSE_ROLL = 4;  
ArcternAttribute.ArcternFaceAttrTypeEnum.QUALITY = 5;  
ArcternAttribute.ArcternFaceAttrTypeEnum.LIVENESS_IR = 6;  
ArcternAttribute.ArcternFaceAttrTypeEnum.IMAGE_COLOR = 7;  
ArcternAttribute.ArcternFaceAttrTypeEnum.FACE_MASK = 8;
```

值	描述	label值 Int类型
GENDER	性别	0:不确定, 1 :男, 2 :女
AGE	年龄	0:灰度图, 1 :彩色图
POSE_PITCH	人脸角度 X轴	以 x 轴为中心,脸部上下俯仰类型
POSE_YAW	人脸角度 Y轴	以 y 轴为中心,脸部左右旋转类型
POSE_ROLL	人脸角度 中心点	以中心点为中心,x-y 平面旋转类型
QUALITY	人脸质量	confidence 人脸值
LIVENESS_IR	红外活体	confidence < 0.5:非活体, confidence >= 0.5 :活体
IMAGE_COLOR	图片类型	0:灰度图, 1 :彩色图
FACE_MASK	口罩检测	0:无口罩, 1 :带口罩