IntSig 扫描笔SDK

Spen SDK 描述

扫描笔（扫读笔、词典笔） 实时拼图识别SDK
扫描一行文本时，进行文本行图像实时拼接和OCR文字识别提取。

系统要求

- 系统： Linux/Android
  推荐Linux系统（客户普遍反馈同等硬件Android系统要比Linux慢）。
  Android ndk 版本：r16b， 编译API level设定：
    - arm64-v8a :api level 21;
    - armeabi-v7a:api level 19
- Memory: 256MB以上，推荐1GB+
    - SDK运行内存平均在64M（扫描文字越多，内存略有增长）

      > 测试方式 使用procrank 命令观察 测试程序的USS 内存数值（测试程序为读取相机拼图识别的无UI界面Demo，代码参见sdk/example_linuxqt/)
- Camera:
    - 帧率： 大于100fps，推荐120fps。帧率高低影响到扫描速度，高帧率，支持更快的扫描速度。低帧率在快扫时容易丢帧。
    - 分辨率： 确保剪裁后帧图像高度>128px。推荐不小于 8万像素。（剪裁：详见下面输入SDK图像格式及质量要求：）

CPU 要求

CPU : 支持Arm64 or Armv7a, 推荐Arm64 推荐 4核心 1.5+GHz, 推荐RK3326平台 CPU的性能，影响扫描抬笔后的出字速度，影响用户体验。

NPU 要求

目前SDK支持 3款NPU

- 瑞芯微rv1109
    - toolkit版本 1.6+
- 亿智SH506
- 齐感QG2101C

SDK性能

- 性能：在RK3326 Linux平台上达到实时拼图和识别
    - 平均帧拼图速度 < 5ms
- 中英文识别率
    - 字符识别率： > 99%
    - 整行识别率： > 96%

多语言支持

SDK中包含 库，试用授权文件，和语言模型文件，通过更换语言模型文件，可以支持其他语言。

- zh9k.dat
  zh9k.dat 为简体中文模型（支持简体中文，英文字符，常见符号等近9700多个字符）该模型经过扫描笔采集数据训练识别率较高。
- eu144.dat 东欧西里尔语系，包括俄语、白俄罗斯语、乌克兰语、塞尔维亚语、马其顿语、保加利亚语。
- w24k.dat w24k.dat为多语言模型，支持语言总数为52种,包含24133个字符,可用于中文繁体，及日韩语等外语场景。
  多语言模型包含24000多个字符，包含近2万1千个中文简繁体字符，近2300个韩语字符，近200个日语假名符号，以及500多个西欧语言字符和常见符号。 其中**w24k-v.dat**包含竖版识别的多语言模型，可用于竖版扫描场景。
  - 大部分外语未经过扫描笔专门数据训练，并且字形相近字体丰富，识别率可能偏低。
  - 由于多语言模型模型变大，识别时间相比简体中文模型(zh9k)速度要慢。
  - 具体语言列表 *简体中文、繁体中文、日语、韩语、英语、法语、葡萄牙语、德语、意大利语、荷兰语、瑞典语、芬兰语、丹麦语、挪威语、匈牙利语、越南语、南非语、阿尔巴尼亚语、巴斯克语、加泰罗尼亚语、克罗地亚语、捷克语、爱沙尼亚语、冰岛语、爱尔兰语、拉丁语、拉脱维亚语、立陶宛语、马来语、波兰语、罗马尼亚语、斯洛伐克文语、斯洛文尼亚语、斯瓦希里语、土耳其语、威尔士语、马耳他语、克里奥尔语、加利西亚语、世界语、菲律宾语、印度尼西亚语、阿塞拜疆语、西班牙语、俄语、保加利亚语、马其顿语、乌克兰语、塞尔维亚语、白俄罗斯语、希腊语、亚美尼亚语.*

NPU和CPU的语言模型文件 不通用，不同NPU也使用不同的模型文件，请根据使用的NPU型号在SDK相应目录中获得模型文件。

集成编译

集成SDK，需要如下文件：

- 动态库：一般在 系统命名+cpu架构 的文件夹内，libspen.so ，不同平台系统还包括其他so库。
- 头文件：提供C++和C接口头文件，推荐使用C++头文件
- 模型文件：以**dat**结尾的数据文件，如zh9k.dat 或 w24k.dat。
- 授权文件：以**key**结尾的数据文件，SDK自带试用授权文件（在一定日期内可用），联系支持人员可获得测试设备授权文件，也可以申请在线授权。

编译libspen.so 使用的gcc版本：

- Linux-aarch64 gcc6.4
- Linux-armv7-a gcc6.5
- Android ndk16b(armv7-a apilevel 19, arm64 apilevel 21)

SDK 内嵌opencv库(3.4.2),如果应用（或其他算法库)也集成了opencv，需要注意版本是否一致,否则会出现未兼容问题。 SDK 使用了openmp计算库（libgomp.so.1），一般在系统的/usr/lib目录中没有，也可以使用交叉编译工具链中的库。 参考example_linuxqt, example_android 样例代码配置。

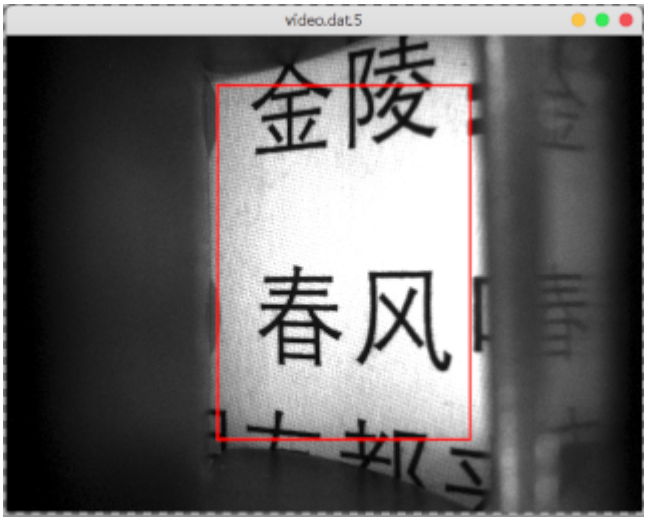
输入SDK图像格式及质量要求：

1. 图像格式要求：

支持YUV(NV21，NV12)，Gray，RGB，BGR。 鉴于大部分设备都是单色摄像头，推荐使用NV12,NV21,Gray8,并且我们引擎内部使用灰度图像进行处理，在使用NV21或NV12格式时，引擎内部只读取了Y分量作为灰度数据。 如果摄像头不是彩色摄像头，不推荐使用BGR或RGB作为输入帧格式。

2. 图像需要做好剪裁

一般设备获得的图像会带有物料边缘,再将帧输入传送到引擎内时，需要做好剪裁，将物料边缘等固定不变的剪裁掉。 如下图所示，摄像头采集的图像只有中心区域包含了有效的文字内容(随着扫描会变动），SDK需要的中心红色框内的图像，在读取的原始图像帧时，需要把红色区域剪裁下来才能送入SDK。



下面以Linux平台 v4l2 Camera 举例说明如果调节图像剪裁：

- 1. 确定 摄像头的设备文件 摄像头文件 一般为，/dev/video**N** ，N为从0开始的数字。如果不确定可使用如下命令获取设备的详细属性信息，来确定设备ID

```
v4l2-ctl -d /dev/video1  --all
```

- 2. 开启补光灯 需要先明确 补光灯 文件地址，和开关参数，示例如下：

```
echo 255 > /sys/class/backlight/camlight/brightness
```

- 3. 取消剪裁,如果设备最大支持640x480分辨率，可以通过如下命令:

```
v4l2-ctl -d /dev/video1 --set-crop=top=0,left=0,width=640height=480
```

- 4. 录制视频 使用如下命令录制视频

```
v4l2-ctl -d /dev/video1 \
--set-fmt-video=width=640,height=480,pixelformat=NV12 \
--stream-mmap=3 \
--stream-skip=3 \
--stream-to=/userdata/spen/video.yuv \
--stream-count=80 \
--stream-poll
```

参数解析

> --set-fmt-video=width=640,height=480,pixelformat=NV12 # 视频大小及格式 --stream-mmap=3 # 帧缓存大小 --stream-skip=3 # 跳过前面 3帧 --stream-to=/userdata/spen/video.yuv #保存视频位置 --stream-count=80 #录制总帧数 --stream-poll

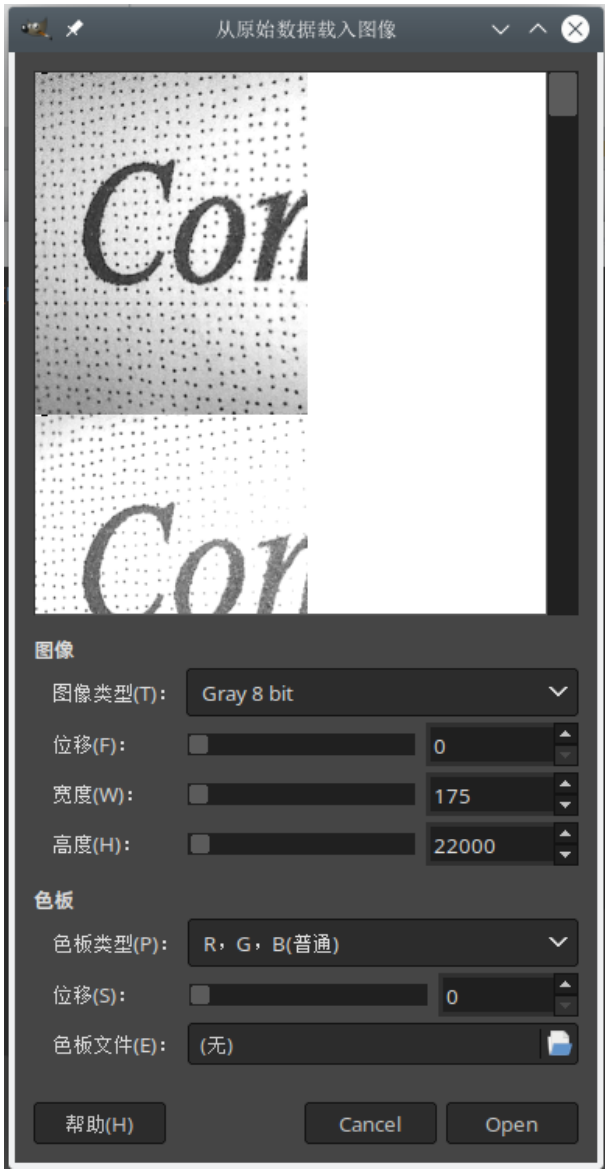
- 5. 在 pc上展示录制好的视频 有三种方法：
  - Ubuntu下使用ffplay工具播放

    > ffplay -f rawvideo -pixel_format nv12 -video_size 640x480 videoyuv

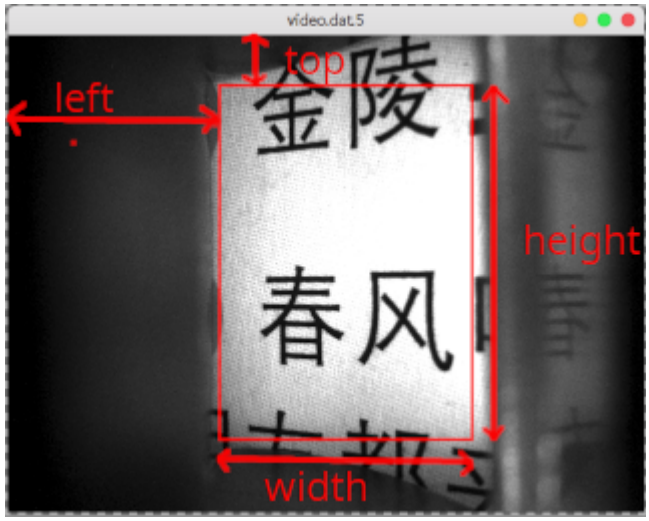
    根据实际情况 更改 pixel_format： nv12(yuv格式），或gray8（灰度格式） video_size 设为实际帧分辨率

  - windows平台可以使用YUVPlayer播放， 在菜单中选择图像格式，图像分辨率等参数。
  - (window/Linux)使用GIMP  图像处理软件
    1. 将视频文件文件后缀改为 .data
    2. 打开GIMP程序，把 .data 视频文件拖拽到GIMP软件中，在弹出的属性窗口中，设置图像格式，图像大小，后点击'Open'： 如果是灰度图像，或YUV（NV12，NV21），图像类型旋

转Gray8bit 宽度，设为图像的 宽 高度，设为图像的 高 x 帧数



并选取清晰的帧图像，在图像上选择合适的矩形剪裁区域参数(left,top,widht,height)。 选取的剪裁区域，**不要包含物料，不要包含阴影**。 为防止不同设备误差过大，**不要紧贴这物料或阴影边缘**，要留出一定距离, 如下图红色框区域。



- 6. 剪裁图像 一般Linux摄像头驱动都提供了剪裁功能，可以通过命令v4l2-ctl进行剪裁设置。

```
v4l2-ctl -d /dev/video1 --set-crop=left=153,top=19,width=318height=424
```

这种剪裁方式，是硬件驱动层面的，速度很快。 剪裁后图像示例：

如果系统不支持驱动层面（如v4l2-ctl）的图像剪裁，可以通过配置ScanConfig的crop_area参数，在SDK内进行剪裁。

```
ScanConfig config;
config.crop_area.left = 153;
config.crop_area.top = 19;
config.crop_area.width = 318;
config.crop_area.height = 424;
sepn.Config(config);
```

3. 图像分辨率要求：

对于图像宽高比不做要求; 分辨率要求， 但建议在图像高度建议128px，如果太大了，SDK会缩放到128px，太小了对于小号字（如6号，7号)识别有影响。举例，上面第4步剪裁的图像区域宽高为328x424，可以设置帧分辨率大小为 100x128，使用v4l2 API来操作摄像头时，驱动会自动进行缩放，以达到较好的效果。

4. 图像角度

传入到SDK的图像需要是“正”的，有些设备摄像头获得的图像是旋转后的。 在传入引擎前，需要将图像转正：

- 方法1： （推荐）物理旋转，即摄像头模组做物理摆放时调整好角度。
- 方法2： 通过Linux 摄像头驱动v4l2 进行旋转

  ```
  v4l2-ctl -d /dev/video1 --set-ctrl=rotate=90
  ```

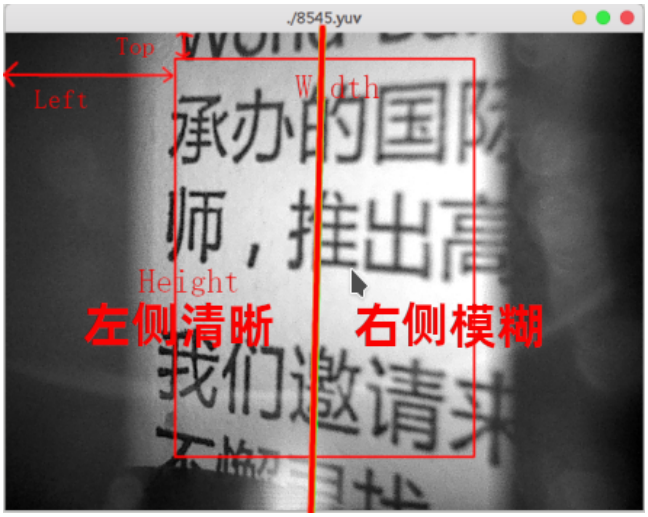
- 方法3： 软件旋转。 自己实现代码旋转, 可参考 example_linuxqt代码。

5. 帧率要求

推荐使用120fps，如果低于120fps，快扫时容易丢帧，造成拼接失败。 如果设备摄像头开口很大，摄像头采集的有效图像宽度较大（一般经剪裁有效区域后图像宽度大概是2被图像高度左右），可以适当降低帧率100fps左右。 在采集拼接过程中，不要做其他耗CPU操作，容易阻塞拼图引擎造成丢帧，如果连续两帧图像偏移超过半帧长度，则容易造成拼图失败。 推荐在数据采集时需要**独立线程+帧缓存**：即在后台1线程中循环采集相机帧，并缓存到队列中。在后台线程2中，循环从帧缓存队列中取图像帧送入SDK。 具体可以参考 example_linuxqt 代码实现。 对于“宽帧”（摄像头后面有开口，采集图像有效区域更广，图像有效文字区域 宽度大于1.4倍高度）设备，可适当降低帧率。

6. 图像聚焦清晰

摄像头一般采用定焦方式聚焦，由于扫描笔持笔时，有一定倾斜，采集的帧图像左右两侧清晰度会不同，在调节聚焦清晰度时，要保证帧图像的左侧清晰。



7. 图像曝光适当

一般摄像头有3A算法调节曝光值，可通过调整爆光值，使采集的图像尽量“真实”，不要过曝或曝光不足。 过曝容易图像发白（笔画细的容易丢失笔画），曝光不足图像发暗（浅色文字看不清）。

8. 开头和结尾的“脏”图像帧

“脏”图像帧，是指图像质量差的帧，一般出现在扫描的开头和结尾： 扫描开始时的两三帧容易出现发黑或发白（可能是相机3A程序还未计算出合理的曝光量等参数） 在扫描结束时，由于笔抬到一定程度才会触发抬笔事件，在抬笔过程中，仍然还在采集图像帧，而此时，由于笔抬高（镜头距离纸面距离超过焦距），造成图像变小，变模糊。 这种脏数据容易造成拼图错误影响识别，需要进行剔除，一般可以根据实际设备硬件情况确定开头要去除的帧数，和结尾要去除的帧数（不要过大，以免丢弃正常帧造成文字缺失）。 SDK 提供了接口 用于设置要过滤的帧数。 代码示例：

```
ScanConfig config;
config.first_frames_ignore = 3; // 过滤开头3帧
config.last_frames_ignore = 4; //过滤末尾4帧。
sepn.Config(config);
spen.Begin(...);
// ...
```

左手模式支持

引擎默认为右手模式，即图像中文字是"正"的,且向左运动, 而左手模式，图像正好反转了180度，调用Being()函数开启扫描时，第二个参数设为true 就开启左手模式，Push函数内部会对图像做180度旋转 。

多行扫描支持

多行扫描场景下，需要前一行 扫描完，即前一行所有帧都调用Push(),并且调用GetOcrResult()获得结果 再调用End()后，才能开启下一行的扫描Begin()函数。可参考SDK包中的example_linuxqt 代码。

```
// 第一行
Begin(),
```

```
    Push()
    GetSliceOcrResult()
    Push()
    GetSliceOcrResult()
    ...
    GetOcrResult()
  End()
  // 第二行
  Begin()
    Push()
    GetSliceOcrResult()
    Push()
    GetSliceOcrResult()
    ...
    GetOcrResult()
  End()
```

竖版文字扫描识别支持

竖版文字常见于日语书籍及一些报纸、杂志的文章标题中。SDK从1.4.0版本开始加入竖版文字扫描识别功能，该功能同时需要加载特定**多方向识别模型**才能开启该功能。
多方向识别模型是在常规模型基础上增加了文字方向检测模型和竖版文字识别模型（模型变大，占用内存也会增多）。

扫描方式如图所示， SDK通过检测模型，会自动判断文字方向，对于竖版文字图像使用竖版文字模型来识别。

点读识别功能

点读功能，是指扫描笔笔尖在单词前按一下立即抬起，就能识别出该单词，并进行翻译阅读等。 点读功能需要扫描笔硬件的支持：笔尖的摄像头开口较大，视角较广。 SDK 对点读功能没有专用接口，使用正常扫描接口即可实现点读识别功能。

SDK 授权

SDK 支持离线授权，和在线授权两种方式。 无论在线授权，离线授权，都需要一份授权文件。

试用授权

随sdk 自带一份试用授权文件，在特定日期之内可以使用SDK（SDK初始化函数返回为AUTH_TIME_TRY(1)），但扫描时最多返回前30个字符，并在末尾增加"试用授权" 字样。 客户也可以为测试设备申请测试授权文件。

离线授权

推荐使用离线文件方式授权
客户提供设备的唯一ID**列表**（如MAC，SN/Android/,等),我们提供一份授权文件。
在SDK初始化时加载这个授权文件，完成授权。SDK运行过程中，（只）会进行一次联网，进行设备SDK注册，用于服务器统计实际激活设备数量，没有网络也不影响使用。

```
#include "scan_pen.h"

using namespace spen;

SpenEngine spen_engine;

// 加载授权文件和模型数据，
//  可能返回值：
// INIT_OK(0)                          授权成功，初始化成功
// INIT_MODEL_FAILED (-201)            模型加载失败
// AUTH_FAILED (-100)                  授权失败，授权文件与设备不符
// AUTH_TIME_TRY (1)                   有时间限制的试用授权，在特定日期前，可以试用
// AUTH_ERROR_KEY_NOT_FOUND (-104)   未找到授权文件，仍可以调用SDK里面的功能函数，
但只返回最多30个识别的字符。
// AUTH_READ_DEVICE_ID_ERROR (-106)   无法读取设备ID
int ret = spen_engine.Init("/path/of/auth.key", "/path/of/zh8k.dat");
if(ret==INIT_MODEL_FAILED){
  //  请检查模型文件是否正确
}else if(ret==AUTH_FAILED){
  // 检查授权文件是否正确
}else if(ret==AUTH_TIME_TRY){
  // 试用授权，在特定日期前，可以试用
}else if(ret==AUTH_ERROR_KEY_NOT_FOUND){
  // 未找到授权文件，请检查授权文件是否存在，或进行网络授权。
}else if(ret==AUTH_READ_DEVICE_ID_ERROR){
   // 无法读取设备ID
}
//
```

在客户开发集成过程中，可以使用带有时间限制的试用授权文件（在SDK包中包含试用授权文件）

在线授权

客户如果不方便再出厂前获得所有设备的设备ID(MAC地址)列表,可以使用此种方式。 通过联网的方式，获得一个设备的授权文件。
客户付款购买后，申请获得 vendor_id, 技术支持人员根据 客户名称(英文字母拼写），及客户购买SDK产品名(SpenSDK,SpenOcrSDK，SpenStitchSDK)，在后台页面配置，并生成客户最终的vendor_id（vendor_id 一般为 客户名称的英文字母拼写加上一串数字），举例，客户名称为**IntSig**，则最终生成的vendor_id为 **IntSig-**

723133
客户应用程序再首次启动时，通过调用SDK的 **Spen::OnLineAuth(vendor_id,SDK_name, storeage_dir)** 函数联网获取授权文件，该函数成功返回后会将授权文件保存目录中，然后正常初始化加载授权文件即可完成授权。
SDK产品名称（SDK_name)
SpenSDK: 完整扫描笔SDK
SpenOcrSDK:只包含ocr的SDK
SpenStitchSDK： 只包含拼图的SDK（未发布）

```
#include "scan_pen.h"

using namespace spen;

SpenEngine spen_engine;

// 联网授权，获得 授权文件,
int ret = Spen::OnLineAuth("ClientID","SpenSDK",
"/internal_storage/spen/");
if(ret!=AUTH_OK){
  // 授权错误
  return;
}
// 联网授权成功的话，授权文件会保存到第三个参数所在的目录中，文件名称为
spen.key("/internal_storage/spen/spen.key")


// 初始化，
// 联网授权模式下， 第一个参数可以为空，
int ret = spen_engine.Init("/internal_storage/spen/spen.key",
"/path/of/zh8k.dat");
if(ret==AUTH_FAILED){
  // 授权失败
  // return
}
//
```

Demo示例

SDK目录中，我们提供了Linux/Qt平台的Demo代码，和Android平台JNI 代码示例。
请用户仔细阅读（example_linuxqt)代码，代码中，包含调用的完整流程，及一个简单的相机帧缓存实现。
如果用户Linux设备支持Qt，也可以编译上机运行。主要调节硬件相关参数。

C++ API 调用示例

授权

离线授权

```
SpenEngine spen_engine;
ret =spen_engine.Init("/path/of/auty.key","/path/of/model");
if(ret==AUTH_OK){
  // 授权成功
}else if(ret==AUTH_TIME_TRY){
  // 带有效期的试用授权
}else if(ret==AUTH_ERROR_KEY_NOT_FOUND){
  //授权文件未找到，功能限制的试用授权（最多返回前30个字符）
}else{
  //INIT_MODEL_FAILED             模型加载失败
  //AUTH_FAILED                   授权失败，授权文件与设备不符
  // AUTH_READ_DEVICE_ID_ERROR    无法读取设备ID
}
```

在线授权

使用MAC地址作为设备ID：

```
// 第一次调用时会联网获取授权文件，之后调用检测到本地授权文件，就不会再联网。
int ret = Spen::OnLineAuth("ClientID","SpenSDK",
"/internal_storage/spen/","");
if(ret<0){
  // 在线授权失败
   // AUTH_EXCEED_MAX_COUNT -101     // 在线授权失败，达到设备数量限制
   // AUTH_NETWORK_ERROR -102        //在线授权失败，网络错误
   // AUTH_SERVER_ERROR -103         // 在线授权失败，服务器异常
   // AUTH_FILE_SAVE_ERROR -105         // 授权文件本地保存错误
}

SpenEngine spen_engine;
ret =spen_engine.Init("","/path/of/model");
if(ret!=AUTH_OK){
}
```

使用Android SN码地址作为设备ID：

```
// 第一次调用时会联网获取授权文件，之后调用检测到本地授权文件，就不会再联网。
int ret = Spen::OnLineAuth("ClientID","SpenSDK",
"/internal_storage/spen/","ro.serialno");
if(ret<0){
  // 在线授权失败
   // AUTH_EXCEED_MAX_COUNT -101     // 在线授权失败，达到设备数量限制
   // AUTH_NETWORK_ERROR -102        //在线授权失败，网络错误
   // AUTH_SERVER_ERROR -103         // 在线授权失败，服务器异常
   // AUTH_FILE_SAVE_ERROR -105         // 授权文件本地保存错误
}
```

```
SpenEngine spen_engine;
ret =spen_engine.Init("","/path/of/model");
if(ret!=AUTH_OK){
}
```

基本扫描流程

```
SpenEngine spen_engine;

// 授权及加载模型数据
//{... 参考上面授权流程 }


//开启一行扫描
spen_engine.Begin(true/*精准模式*/, false/*右手模式*/);

for(;;){
    Frame frame;
    // 读取当前图像帧
    //frame = queue.front()

    spen_engine.Push(frame,false);

    std::string slice_txt;
    // 获取实时拼图的识别结果
    ret = spen_engine.GetSliceOcrResult(slice_txt, stop_scan?true;false);
    if(ret>0){
        // TODO 显示结果 slice_txt
    }

    if(stop_scan){
        break;
    }
}
Image image;
// 获取拼接图像，用于开发中查看拼图效果
spen_engine.GetRawImage(image);

std::string text;
// 最后获取完整拼图做识别结果
spen_engine.GetOcrResult(text);
// TODO 显示结果 text

//开启一行扫描，清理内存
spen_engine.End();
```

Linux/Qt Demo 使用说明

example_linuxqt目录的scan_demo 是Linux平台的一个体验Demo。
编译example_linuxqt时，可以通过宏控制关闭QT GUI，可以生成一个纯命令行的Demo。 拷贝 scan_demo libgomp.so.1 zh8k.dat demo.sh 到 设备的 /userdata/spen/目录
拷贝试用授权文件 到/userdata/spen/auth.key
修改demo.sh里面的相关设备参数和相机参数，

启动demo 执行 demo.sh 启动demo，

```
#!/bin/bash

## define hardware arguments
# camera  设备文件
export DEV_CAMERA="/dev/video1"
# 笔尖按键 事件文件
export EVENT_BTN="/dev/input/event3"
# LED光照 设备文件
export DEV_LED="/sys/class/backlight/camlight/brightness"
# LED光照开关 数据 ，如`echo 255 > $DEV_LED`表示把LED等调到最亮
export LED_ON="255"
export LED_OFF="0"

## set camera crop region
# 设置Camera 图像帧剪裁区域，具体参数获取，阅读文档后面部分内容。
v4l2-ctl -d $DEV_CAMERA --set-crop=left=190,top=38,width=300,height=400

export XDG_RUNTIME_DIR=/tmp/.xdg

## wayland or linuxfb
export QT_QPA_PLATFORM=wayland

export LD_LIBRARY_PATH=/userdata/spen/
# -video 使用该参数可以保存扫描的视频数据
# -s nnxnn 指定读取的camera 帧大小
# -r n 表示对读取 的每帧图像做旋转的角度数。
/userdata/spen/scan_demo -video  -s 150x200 -r 0   &
```

扫描一行后，视频数据会保存在 /userdata/spen/video.dat文件中，用于调试用
扫描的图像将会保存到/userdata/spen/image/目录中，图像文件格式为pgm裸数据格式（Ubuntu下可以直接查看，Window下使用GIMP软件查看)

Android平台Java接口 Android JNI Demo

Android平台上， 我们只SDK提供了 C++接口的动态库和可编译的JNI Example代码。SDK目录中我们提供了Android平台的JNI示例代码，可以直接使用ndk编译获得可用的Java接口，用户可以根据需求自行修改Jni代码达到更高的性能。
如果客户需要使用Java接口的库，可以使用Android 的NDK工具（推荐使用NDK-r16b）,编译 example_android 即可获得Java接口库。

```
$ cd example_android
$ ndk-build
$ ls libs
```

客户可以根据自身需要修改example_android/jni/下的代码，以获得最佳性能。

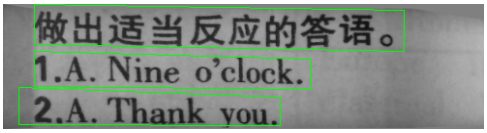
如何使用SDK的OCR文本行识别功能

如果客户有自己的拼图算法，能够得到扫描的单行文本图像，则可以只使用SDK的OCR函数RecogLine()，示例如下。

```
#include "scan_pen.h"

using namespace spen;

SpenEngine spen_engine;

// 加载模型数据
int ret = spen_engine.Init("/path/of/key", "/path/of/zh8k.dat");
if(ret<0){
    // fail to load model.
}


Frame line_image;// 拼接剪裁好的单行文本图像。
line_image.buf =  pixel ;//文本行图像的像素
line_image.width = 120; // 文本行图像的宽度
line_image.height = 30;// 文本行图像的高度
line_image.format = GRAY;// 支持 GRAY， RGB，BGR格式。
std::vector<CharItem> char_array
spen_engine.RecogLine( line_image, char_array);
for(CharItem item: char_array){
  printf("%s",item.ch.c_str());
}

// 对于图像中有多行文本的情况
Frame lines_image; // 拼接好的可能有多行的文本图像。
std::vector<LineItem> line_items;
spen_engine.DetectLines( line_image, line_items);
for(LineItem& line:line_items){
  printf(" corner points left,top: %f,%f, right,top:%f,%f,
right,bottom:%f,%f, left,bottom:%f,%f\n",

line.vertexes[0].x,line.vertexes[0].y,line.vertexes[1].x,line.vertexes[1].y
,

line.vertexes[2].x,line.vertexes[2].y,line.vertexes[3].x,line.vertexes[3].y
);
  // 将文本行四边形， 透视变化成矩形，得到 单行文本行图像
  // 对单行图像进行 识别  调用RecogLine
```

```
}
```

多行文本情况,DetectLines 检测出来的文本行块如下图所示绿色四边形框。



对于比较稀疏的一行文字，可能会检测为多个四边形框。

C++ API Reference

Spen SDK

- **构造拼图识别引擎实例**

```
SpenEngine()
```

- **从文件初始化SDK**

```
int Init(const std::string& key_file_path, const std::string& model)
```

从文件加载授权文件，和ocr模型， 参数：

  - key_file_path : 授权文件路径
  - model :模型路径
    返回值： >=0 成功， <0 失败，具体返回值意义如下：
    - INIT_OK(0) 授权成功，初始化成功
    - INIT_MODEL_FAILED (-201) 模型加载失败
    - AUTH_FAILED (-100) 授权失败，授权文件与设备不符
    - AUTH_TIME_TRY (1) 有时间限制的试用授权，在特定日期前，可以试用
    - AUTH_ERROR_KEY_NOT_FOUND (-104) 未找到授权文件，仍可以调用SDK里面的功能函数，但只返回最多30个
    - AUTH_READ_DEVICE_ID_ERROR (-106) 授权失败，无法读取设备ID(MAC地址，或SN码等ID)

- **从内存初始化SDK**

```
int Init(const char* key_data, size_t key_data_size,const char* model,
size_t size)
```

从内存加载授权文件，和ocr模型， 参数：

  - key_data： 授权文件内存地址

- key_data_size: 授权文件大小
- model :模型内存地址
- size: 模型字节数

  返回值： >=0 成功， <0 失败，具体返回值意义如下：
  - INIT_OK(0) 授权成功，初始化成功
  - INIT_MODEL_FAILED (-201) 模型加载失败
  - AUTH_FAILED (-100) 授权失败，授权文件与设备不符
  - AUTH_TIME_TRY (1) 有时间限制的试用授权，在特定日期前，可以试用
  - AUTH_ERROR_KEY_NOT_FOUND (-104) 未找到授权文件，仍可以调用SDK里面的功能函数，但只返回最多30个
  - AUTH_READ_DEVICE_ID_ERROR (-106) 授权失败，无法读取设备ID(MAC地址，或SN码等ID)

- **配置扫描相关选项**

```
void Config(ScanConfig config)
```

用于配置扫描性能相关参数：

```
struct ScanConfig{
  struct {
    int left=0;           // 剪裁区域 左上角顶点 x坐标（图像左上角为原点）
    int top=0;            // 剪裁区域 左上角顶点 y坐标（图像左上角为原点）
    int width=0;          // 剪裁区域宽度
    int height=0;         // 剪裁区域高度
  } crop_area;                  // 帧图像有效区域剪裁参数. 推荐在相机驱动这进行剪裁,默认不做剪裁。
  int rotate_angle=0;           // 帧图像旋转角度，可选 0,90,180,270。 默认0。
  int first_frames_ignore = 3; // 扫描时，要丢弃的开始的帧数（相机刚打开前面几帧曝光不正常，过黑或过白）。 默认丢弃开始的3帧图像。
  int last_frames_ignore = 4;  // 扫描结束时 要丢弃掉最后面（抬笔过程中图像异常）的帧数。默认丢弃最后4帧图像。
  bool fast_mode = false;       //  快速识别模式,抬笔后快速出结果，默认为false。
};
```

crop_area的坐标相对于输入的图像的，即同时设置了剪裁和旋转，crop_are坐标参数是旋转之前的。

- **开始一行扫描**

```
int Begin(bool precised = true, bool left_hand=false)
```

开启 新一行图像扫描识别
参数：

  - precised： true:精准扫描 只识别滑过的文字， false：识别所有扫描的文字
  - left_hand true为左手模式（会对输入图像做180度旋转），false为右手模式
    返回值： 0

- **实时处理帧图像**

```
int Push(const Frame& frame, bool last)
```

追加一帧图像，进行拼图
参数：

  - frame：相机获得的帧图像数据，推荐使用GRAY模式，可以将NV12/NV21直接作为Gray模式（把Y分量作为灰度数据）
    - frame.buf : 图像像素数据
    - frame.width: 图像宽度
    - frame.height: 图像高度
    - frame.foramt :数据格式，支持NV12，NV21，GRAY格式
    - frame.sequence: 未使用
  - last: : false 为中间帧，true为最后一帧 ，可以都传false

返回值：

  - 返回已经拼接的图像的宽度,
  - 如果 返回 -1，表示Push前没有调用Begin()
  - 如果 返回 -2，表示扫描长度过长（超过5000像素宽度）,不再接收新图像帧进行拼接(正常用户使用中不会超过这个限制)

- **获取扫描的文本行图像 （Debug使用）**

```
int GetImage(Image& output)
```

获得当前拼接剪裁的文本行图像，一定要在End()之前调用，用于 开发 debug（会消耗额外cpu资源）

- **获取扫描的完整图像（Debug使用）**

```
int GetRawImage(Image& output)
```

获得当前拼接的原始图像，一定要在End()之前调用，用于 开发 debug

- **获取实时识别结果**

```
 int GetSliceOcrResult(std::string& output, bool last)
```

边扫过程中， 调用此函数，获得已识别的文字结果，并根据扫描长度触发新扫图像的识别。 参数：
- output: 用于存放识别文字结果
- last: 已弃用，可以都传false
返回值： <0 失败 , =0 结果未更新， >0 结果有更新

- **获取完整识别结果**

```
int GetOcrResult(std::string& output)
```

在扫完一行后，获取当前拼好的整张图像做OCR， 返回OCR结果， 一定要在End()之前调用 参数：
- output: 用于存放识别文字结果
返回值： <0 失败

- **结束一行扫描**

```
void End()
```

结束 一行的扫描识别，清理内存数据。

- **保存视频帧数据（Debug使用）**

```
int SaveFrameData(const char* data_file_path)
```

保存视频帧裸数据，用于Debug目的，**正式产品中请勿调用**。 在Begin()之前调用该函数。 会将Begin()和End()之间所有Push()的帧数据到保存到文件**data_file_path**中。

参数:

  - data_file_path 视频帧裸数据文件路径， 注意**不是目录地址，是文件地址.** 返回值: <0 文件打开失败。

- **识别一行文本行图像**

```
int RecogLine(const Frame& line_image, std::vector<CharItem>&
char_items);
```

识别文本行图像，支持NV12，NV21,BGR,RGB,GRAY图像格式 参数：

  - line_image , 单行文本图像
  - char_items, 识别的文字内容。 返回值 : <0 失败

- **检测文本块位置**

```

```

```
int DetectLines(const Frame& lines_image, std::vector<LineItem>&
line_items);
```

检测文本图像的文本行块，支持NV12，NV21,BGR,RGB,GRAY图像格式 参数： - lines_image , 包行多行的文本图像 - line_items, 检测到的多个文本行块，每一个文本行块为一个四边形，需要做透视变换后获得可识别的文本行图像。 返回值 : <0 失败

- **在线授权**

  ```
  int OnLineAuth(const std::string& vendor_id, const std::string&
  SDK_name, const   std::string& data_storage_dir, const std::string&
  sn_key="");
  ```

  参数：

  - vendor_id , 客户ID，付费购买后获得
    - SDK_name: 参数值可选如下：如果为完整扫描笔SDK，"SpenSDK"。 如果只购买文本行识别，为"SpenOcrSDK"。
    - data_storage_dir, 授权文件存储目录，建议试用永久存储
    - sn_key ,当使用mac地址时，传空字符串，如果时使用Android 的SN时，则为sn的KEY名称,默认为"ro.serialno" 返回值 : <0 失败 ，具体返回值意义如下：
    - AUTH_FAILED -100 在线授权失败，授权文件与设备不符
    - AUTH_EXCEED_MAX_COUNT -101 在线授权失败，达到设备数量限制
    - AUTH_NETWORK_ERROR -102 在线授权失败，网络错误
    - AUTH_SERVER_ERROR -103 在线授权失败，服务器异常
    - AUTH_FILE_SAVE_ERROR -105 在线授权失败，授权文件本地保存错误
    - AUTH_READ_DEVICE_ID_ERROR -106 在线授权失败，无法读取设备ID

  联网授权, 发送设备ID(MAC地址)到授权服务器，获取授权文件。 如果检测到已获得授权文件（判断第三个参数目录中是否有合法授权文件），则立即返回，不会重复联网授权， 即可以多次调用。 因为有联网请求，推荐在后台线程中调用该函数。 授权成功，会把授权文件保存再 {data_storeage_dir}/spen.key 文件中。 联网授权成功后。 仍需要调用函数spen.Init(key_file_path,model) 初始化引擎，但第一个参数可以为空，也可以为{data_storeage_dir}/spen.key

- **判断字符串语言类型** int CheckTextLanguage(const std::string text,LangType& outtype); 参数:

  - text ,待判断字符串
  - outtype，输出结果

  判断字符串语言类型，仅支持 英语(LangCode::English)，中文LangCode::Chinese)，日语(LangCode::Japanese)，韩语(LangCode::Korean),和无法判断(LangCode::Default) 返回结果保存在LangType.lang(LangCode中), LangType.score 存放判断结果的可信度仅作参考。

C API 说明

某些平台无法使用C++ API，可以使用SDK提供的C API，C 函数和C++函数基本是对应的，具体函数说明请参考头文件和相应的C++ 函数。

```
#include "scan_pen_c.h"

void *engine;
ret =spen_Init(&engine,"/path/of/key", "/path/of/model/data");
// check ret

struct ScanCfg config;
spen_config_init(&config);
config.first_frames_ignore = 6;
config.last_frames_ignore = 4;
spen_Config(engine, config);

spen_Begin(engine, 1,0);

spen_Push(engine, pixel_buf, width,height,0);

char text_buf[MAX_BYTE_IN_LINE];
spen_GetOcrResult(engine,text_buf);
printf("%s\n",text_buf);

spen_Release(engine);
```

常见用户问题与建议

1. 帧图像模糊，发白，或发暗，不清晰
   需要检查摄像头软硬件相关参数，聚焦，曝光等。

2. 帧图像剪裁不正确
   参考输入SDK图像格式及质量要求：进行调节

3. 丢帧，帧不连贯

   - 情况1，设备cpu性能差(如单核cpu)，没法及时读取所有帧数据，建议更换更好cpu。
   - 情况2，代码不当，在帧读取线程做了其他耗时操作，阻塞了帧读取操作造成丢帧。
   - 情况3，系统中其他进程在占用CPU。
     - 有些系统3A进程在动态计算camera参数时会占用较多CPU，可尝试禁用3A进程，使用固定曝光白平衡等参数。

4. 帧顺序不正确
   后面的帧跑到前面，一般是代码有bug。

5. 识别效果差
   一般是图像质量较差，参考上面问题1，2。

6. 识别结果文字中间容易丢字
   一般是丢帧造成，参考上面问题3。

7. 扫描识别速度慢

- 情况1，cpu性能差，可尝试开启CPU性能模式（一般电量低时cpu会处于省电模式）。
- 情况2，对于NPU，请确保NPU更新到最新驱动。
- 情况3，可优化代码，减少不必要的内存拷贝，图像处理等。 可尝试操作：
- 将camera输出（剪裁后）帧高度调节到128像素，减少sdk内部图像缩放操作。

8. 补光灯 光照存在光斑 由于笔尖透明窗口内存会存在反光，如果透明窗存在弯曲，则反射光线不均匀容易形成光斑，在扫描浅色文字时，光斑容易和笔画无法区分造成错误。

9. 摄像头设备建议 建议设备后侧开口，以扫描更广的文字图像，扫描到的文字区域越宽，快扫时拼图出错率越低。 摄像头朝向与笔身有一定夹角（正常手持笔笔身与桌面是有倾斜角的）

10.