**libsynexens3 SDK 使用说明 v1.3**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本 | 修改日期 | 适配SDK版本 | 描述 | 作者 |
| v1.0 | 20220704 | v0.1.2 | 初始版本 |  |
| v1.1 | 20220705 | v0.1.3 | 补充接口说明 |  |
| v1.2 | 20220711 | v0.1.3 | 添加ubuntu 环境配置 |  |
| v1.3 | 20230217 | v0.7.3 | 新增滤波接口 |  |

目录

[1. 概述 4](#_Toc9524)

[2. 环境配置 5](#_Toc21794)

[2.1. Windows 环境配置（以vs2017 为例） 5](#_Toc3317)

[2.1.1. 创建vs工程 5](#_Toc19090)

[2.1.2. 选择与SDK库对应的解决方案及解决平台 5](#_Toc28332)

[2.1.3. 在项目属性中配置sdk 的头文件路径、库路径 6](#_Toc1791)

[2.1.4. 配置完成后即可进 7](#_Toc7638)

[2.2. Ubuntu 环境配置（以cmake 为例） 7](#_Toc12494)

[2.2.1. 安装依赖 7](#_Toc19613)

[2.2.2. 编写CmakeLists.txt，此过程需熟悉cmake语法 7](#_Toc10615)

[2.2.3. 创建编译项目文件 8](#_Toc14026)

[2.2.4. make 9](#_Toc17698)

[2.2.5. 执行可执行文件测试效果 9](#_Toc20823)

[2.3. 调用流程图 10](#_Toc14446)

[3. API参考 11](#_Toc2995)

[3.1. 全局接口 11](#_Toc13622)

[3.1.1. get\_device\_info 11](#_Toc22508)

[3.1.2. query\_device 11](#_Toc13950)

[3.1.3. sy3\_create\_pipeline 13](#_Toc6490)

[3.1.4. sy3\_create\_config 13](#_Toc15617)

[3.1.5. sy3\_get\_device\_info 14](#_Toc21895)

[3.2. 基础接口 15](#_Toc2292)

[3.2.1. device::get\_sensor 15](#_Toc8545)

[3.2.2. device::get\_type 15](#_Toc748)

[3.2.3. device::get\_support\_stream 16](#_Toc21759)

[3.2.4. device::get\_support\_format 17](#_Toc21787)

[3.2.5. device::get\_support\_format 18](#_Toc8826)

[3.2.6. config::enable\_stream 18](#_Toc32565)

[3.2.7. config::disable\_stream 19](#_Toc20652)

[3.2.8. config::disable\_all\_streams 20](#_Toc17086)

[3.2.9. pipeline::start 20](#_Toc8407)

[3.2.10. pipeline::get\_process\_engin 21](#_Toc29330)

[3.2.11. pipeline::stop 22](#_Toc24186)

[3.2.12. pipeline::wait\_for\_frames 22](#_Toc18408)

[3.2.13. pipeline::get\_device 23](#_Toc21218)

[3.2.14. frameset::get\_depth\_frame 24](#_Toc12312)

[3.2.15. frameset::get\_ir\_frame 24](#_Toc25147)

[3.2.16. frameset::get\_raw\_frame 25](#_Toc25070)

[3.2.17. frame::get\_width 25](#_Toc1581)

[3.2.18. frame::get\_height 26](#_Toc15962)

[3.2.19. frame::get\_type 26](#_Toc14914)

[3.2.20. depth\_frame::get\_data 27](#_Toc24495)

[3.2.21. depth\_frame::apply\_colormap 28](#_Toc5077)

[3.2.22. frame::get\_profile 29](#_Toc6701)

[3.2.23. frame::dump 29](#_Toc8475)

[3.2.24. points::get\_points 30](#_Toc25122)

[3.2.25. points::get\_length 31](#_Toc29324)

[3.3. SENSOR控制接口 32](#_Toc2050)

[3.3.1. sensor::set\_option 32](#_Toc28658)

[3.3.2. sensor::set\_option 32](#_Toc2521)

[3.3.3. sensor::get\_option 33](#_Toc9967)

[3.3.4. sensor::get\_option 34](#_Toc12158)

[3.3.5. sensor::set\_filter\_value 35](#_Toc21966)

[3.3.6. sensor::get\_filter\_value 35](#_Toc17558)

[3.4. 算法接口 41](#_Toc11777)

[3.4.1. process\_engine::comptute\_points 41](#_Toc32681)

[3.4.2. process\_engine::align\_to\_rgb 42](#_Toc4964)

[3.5. 数据类型 43](#_Toc26316)

[3.5.1. sy3\_error 43](#_Toc1722)

[3.5.2. sy3\_device\_type 44](#_Toc18094)

[3.5.3. sy3\_camera\_info 45](#_Toc7431)

[3.5.4. sy3\_stream 46](#_Toc14096)

[3.5.5. sy3\_format 47](#_Toc12970)

[3.5.6. intrinsics 47](#_Toc16362)

[3.5.7. sy3\_option 48](#_Toc16494)

[4. 示例 51](#_Toc3277)

[4.1. 获取depth 帧 51](#_Toc10720)

[4.2. 获取align数据 52](#_Toc18499)

[4.3. 免责声明 53](#_Toc22454)

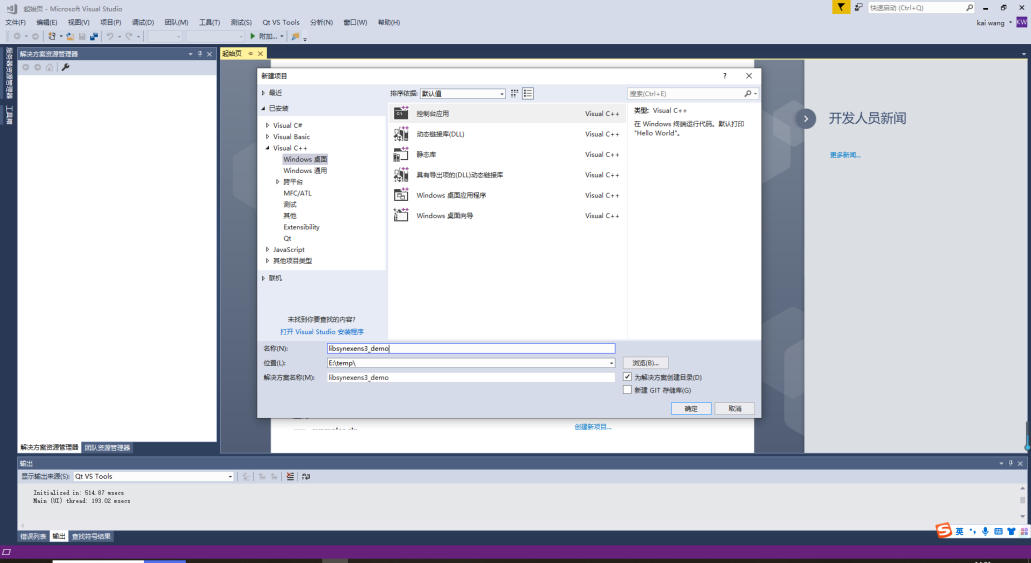
# 概述

适配CS系列模组

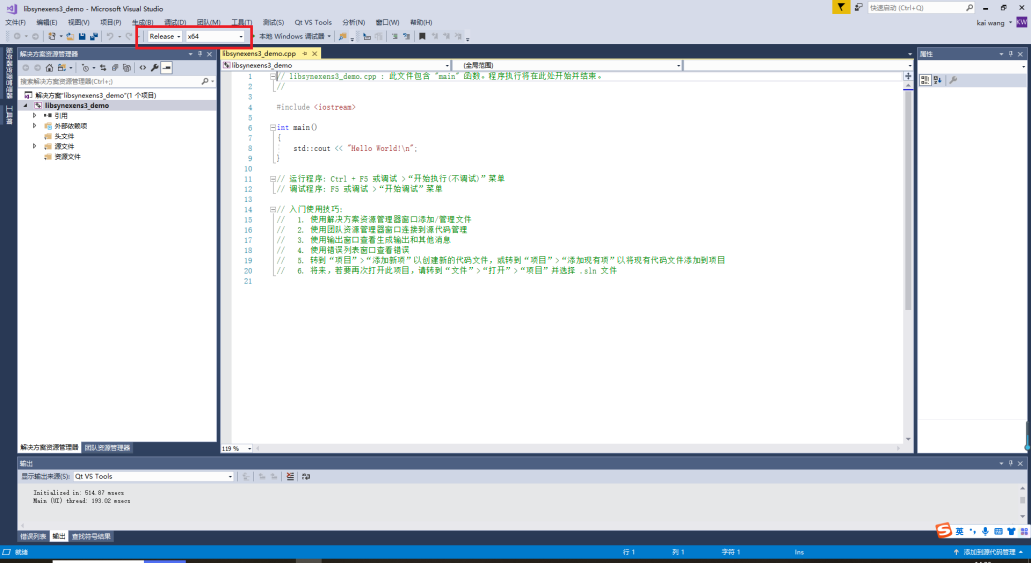
# 环境配置

## Windows 环境配置（以vs2017 为例）

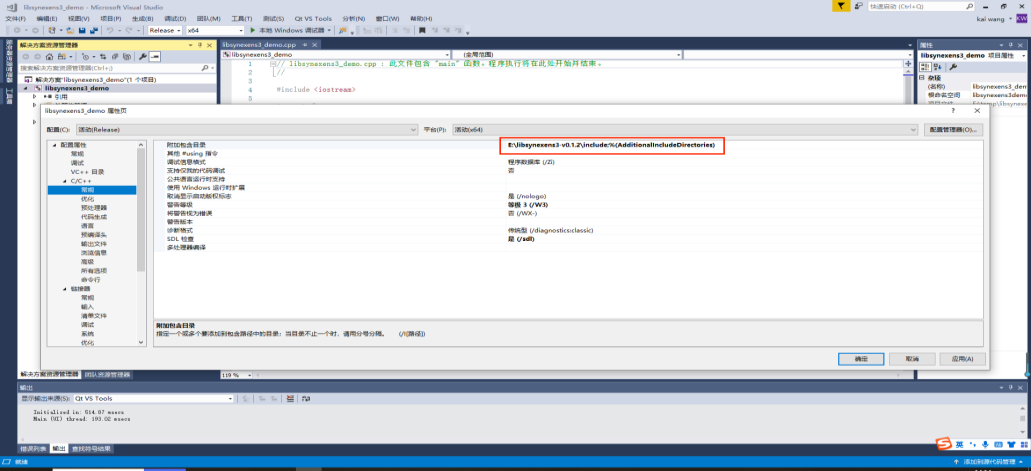
### 创建vs工程

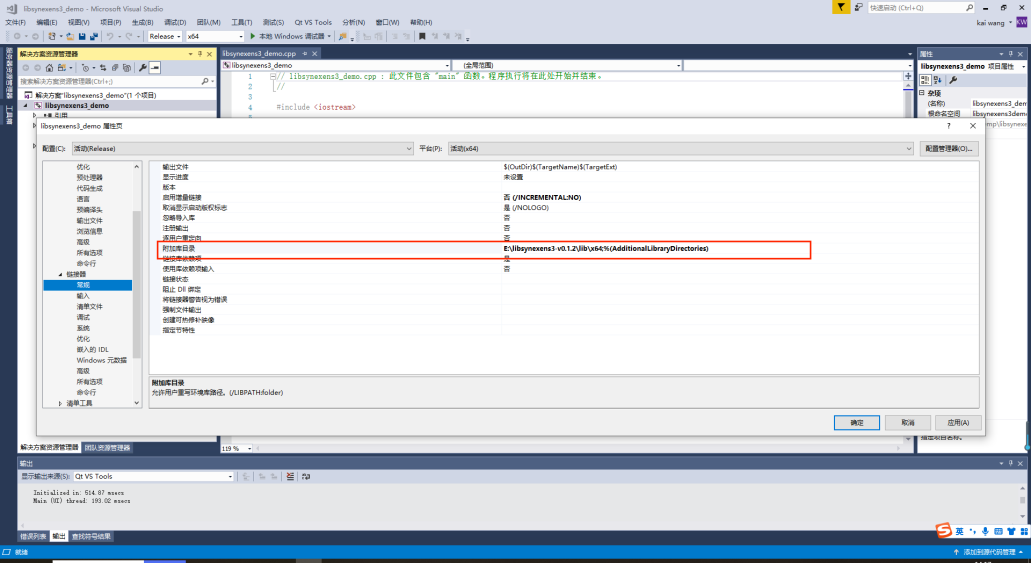


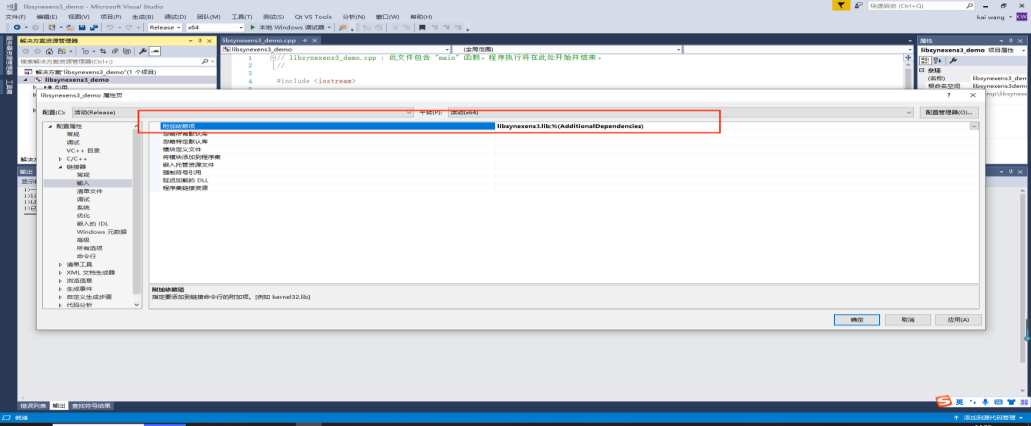
### 选择与SDK库对应的解决方案及解决平台



### 在项目属性中配置sdk 的头文件路径、库路径







### 配置完成后即可进

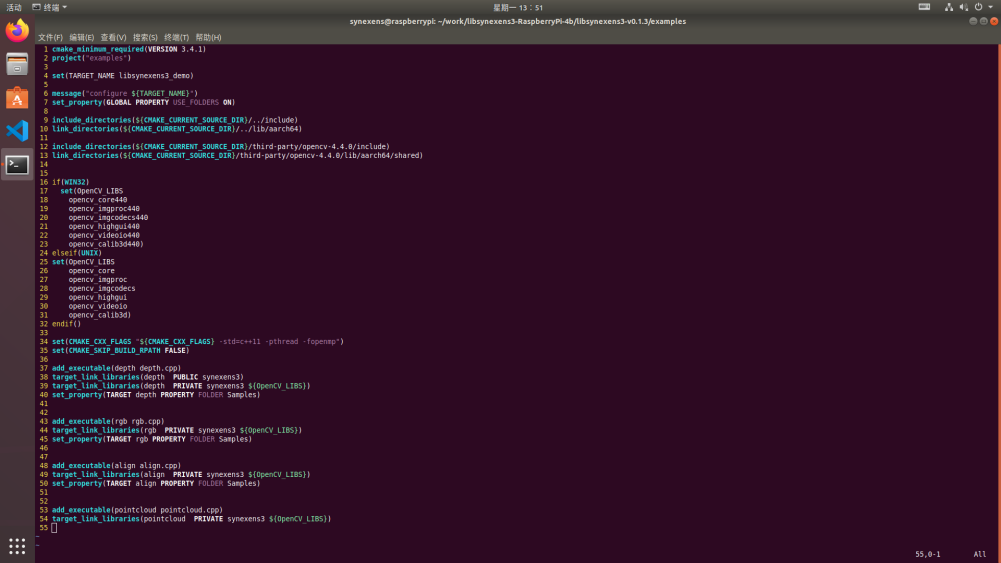
## Ubuntu 环境配置（以cmake 为例）

### 安装依赖

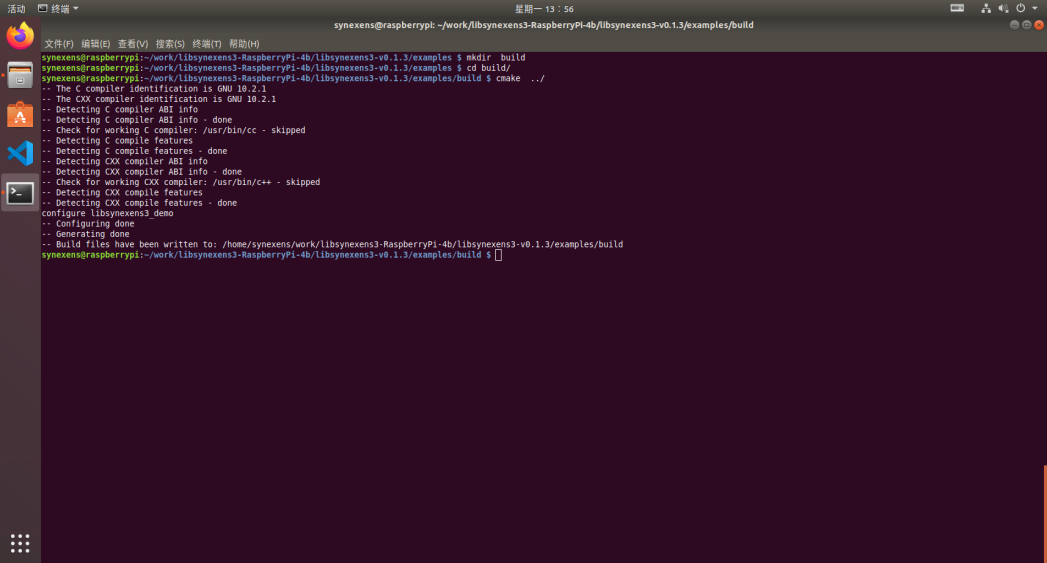
sudo apt-get install libusb-1.0-0-dev

sudo apt-get install libudev-dev

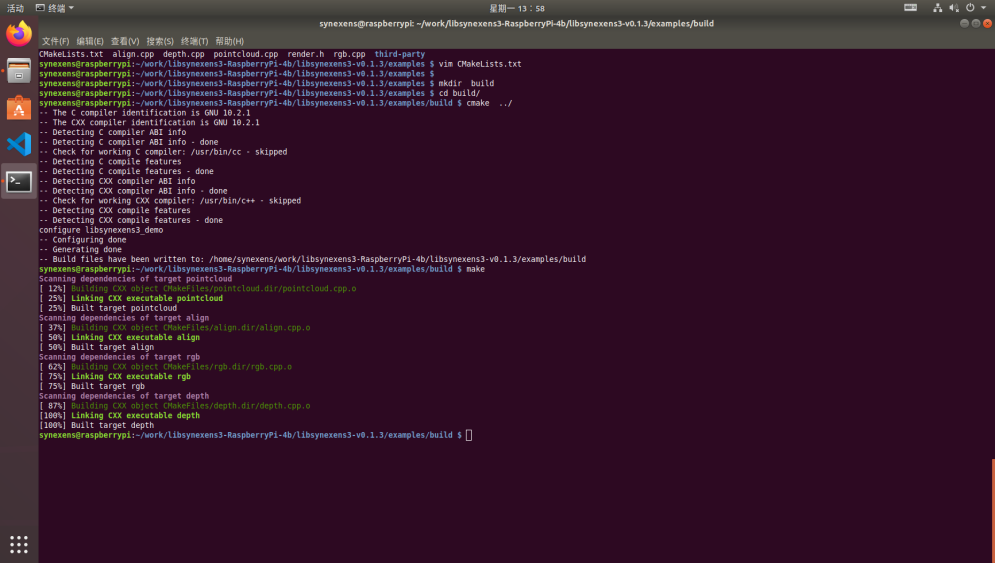
### 编写CmakeLists.txt，此过程需熟悉cmake语法



### 创建编译项目文件



### make

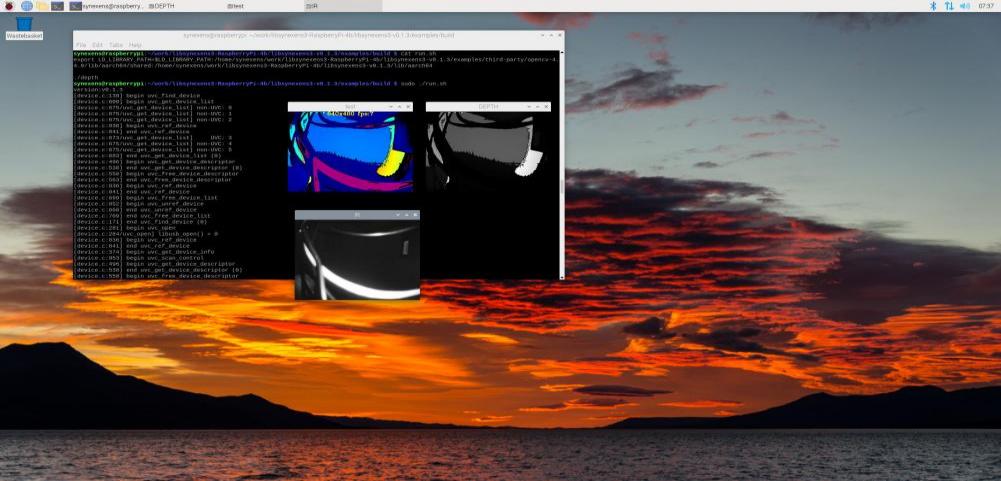


### 执行可执行文件测试效果

执行程序前需配置好 LD\_LIBRARY\_PATH, 以便找到程序依赖的库文件，示例编写了run.sh 脚本以方便执行程序

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:../lib/:../../../third-party/ubuntu18.04\_x64/opencv-4.4.0/lib/x64/shared/

./depth



## 调用流程图

目前仅支持轮询方式调用



# API参考

## 全局接口

### get\_device\_info

描述：获取设备信息

语法：

const device\_info \*get\_device\_info(sy3\_error &error) const;

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| device\_info | 设备信息 |

### query\_device

描述：查询设备

语法：

device \*query\_device(sy3\_error &error) const;

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| device | 设备指针 |

### sy3\_create\_pipeline

描述：创建管线

语法：

pipeline \*sy3\_create\_pipeline(const context \*ctx,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| ctx | 当前使用的上下文 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| pipeline | 管线指针 |

### sy3\_create\_config

描述：创建配置参数指针

语法：

config \*sy3\_create\_config(sy3\_error &error)

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| config | 配置参数指针 |

### sy3\_get\_device\_info

描述：获取设备信息

语法：

const char\* sy3\_get\_device\_info(const device\* device,sy3\_camera\_info info, sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| device | 设备指针 | 输入 |
| info | 信息枚举 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| const char\* | 对应的设备信息字符串 |

注意：在成功查找到设备之前调用无效

## 基础接口

### device::get\_sensor

描述：获取sensor指针

语法：

const sensor \*get\_sensor(sy3\_error &error) const;

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| sensor \* | sensor指针 |

### device::get\_type

描述：获取device类型

语法：

const sy3\_device\_type get\_type(sy3\_error &error) const;

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| sy3\_device\_type \* | 设备类型 |

### device::get\_support\_stream

描述：获取设备支持的数据流列表

语法：

const std::vector<sy3\_stream> get\_support\_stream(sy3\_error &error) const

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| const std::vector<sy3\_stream> | 数据流列表 |

### device::get\_support\_format

描述：获取支持的数据格式

语法：

const std::vector<sy3\_format> get\_support\_format(sy3\_error &error) const;

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| const std::vector<sy3\_format> | 数据格式列表 |

### device::get\_support\_format

描述：获取设备指定数据流所支持的格式

语法：

const std::vector<sy3\_format> get\_support\_format(sy3\_stream stream,sy3\_error &error) const;

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| stream | 数据流类型 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| const std::vector<sy3\_format> | 数据格式列表 |

### config::enable\_stream

描述：使能指定的数据流

语法：

void enable\_stream(sy3\_stream stream, uint16\_t width, uint16\_t height,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| stream | 要启用的数据流类型 | 输入 |
| width | 数据流图像的宽 | 输入 |
| height | 数据流图像的高 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：无

注意：此函数仅在pipeline 启动前调用有效，在pipeline 运行过程中调用无效。

### config::disable\_stream

描述：禁用指定的数据流

语法：

void disable\_stream(sy3\_stream stream,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| stream | 要禁用的数据流类型 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：无

### config::disable\_all\_streams

描述：禁用全部数据流

语法：

void disable\_all\_streams(sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：无

### pipeline::start

描述：启动pipeline

语法：

void start(const config \*cfg,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| cfg | pipeline配置 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：无

### pipeline::get\_process\_engin

描述：获取算法实例指针

语法：

process\_engine\* get\_process\_engin(sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| process\_engine\* | 算法实例指针 |

### pipeline::stop

描述：停止pipeline

语法：

void stop(sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：无

### pipeline::wait\_for\_frames

描述：获取pipelinen帧集合

语法：

frameset \*wait\_for\_frames(unsigned int timeout\_ms,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| timeout\_ms | 超时 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| frameset \* | 帧集合 |

### pipeline::get\_device

描述：获取当前设备

语法：

const device \*get\_device(sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| const device \* | 设备指针 |

### frameset::get\_depth\_frame

描述：获取depth帧

语法：

depth\_frame \*get\_depth\_frame();

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| depth\_frame \* | depth 帧数据指针 |

### frameset::get\_ir\_frame

描述：获取ir帧

语法：

ir\_frame \*get\_ir\_frame();

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| ir\_frame \* | ir帧数据指针 |

### frameset::get\_raw\_frame

描述：获取设备信息

语法：

raw\_frame \*get\_raw\_frame();

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| raw\_frame \* | raw帧数据指针 |

注意：目前raw\_frame 数据只对内部标定开放,外部用户无法获取到raw数据

### frame::get\_width

描述：获取帧图像的宽

语法：

const int get\_width();

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| const int | 宽 |

### frame::get\_height

描述：获取帧图像的高

语法：

const int get\_height() ;

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| const int | 高 |

### frame::get\_type

描述：获取帧类型

语法：

const sy3\_stream get\_type() ;

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| const sy3\_stream | 帧类型 |

### depth\_frame::get\_data

描述：获取帧图像数组,数据以uint16类型二维数组(uint16[h][w])的形式存储，并将数组指针作为返回值返回给调用者。具体数据格式如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| depth[0][0] | depth[0][1] | ... | depth[0][width-2] | depth[0][width-1] |
| depth[1][0] | depth[1][1] | ... | depth[1][width-2] | depth[1][width-1] |
| ...  ... | | | | |
| depth[height-1][0] | depth[height-1][1] | ... | depth[height-1][width-2] | depth[height-1][width-1] |

语法：

void \*get\_data();

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| void \* | 帧数据指针 |

注意：返回值类型为void\*,用户需要手动转化成uint16\*

### depth\_frame::apply\_colormap

描述：获取映射rgb(BGR 格式)后的数据,数据以uint8类型三维数组(uint8[h][w][3])的形式存储，并将数组指针作为返回值返回给调用者。具体数据格式如下：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| rgb[0][0][(b)0] | rgb[0][0][(g)1] | rgb[0][0][(r)2] |  |  |  | ... | rgb[0][w-1][0] | rgb[0][w-1][1] | rgb[0][w-1][2] |
| rgb[1][0][0] | rgb[1][0][1] | rgb[1][0][2] |  |  |  | ... | rgb[1][w-1][0] | rgb[1][w-1][1] | rgb[1][w-1][2] |
| ... | | | | | | | | | |
| rgb[h-1][0][0] | rgb[h-1][0][1] | rgb[h-1][0][2] |  |  |  | ... | rgb[h-1][w-1][0] | rgb[h-1][[w-1][1] | rgb[h-1][[w-1][2] |

语法：

uint8\_t \*apply\_colormap();

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| uint8\_t \* | rgb 图像数组指针 |

### frame::get\_profile

描述：获取帧配置

语法：

const stream\_profile \*get\_profile() const;

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| const stream\_profile \* | 帧配置指针 |

### frame::dump

描述：保存帧数到本地，供调试使用

语法：

int dump(const char \*filenam) ;

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| filenam | 要保存的文件名 | 输入 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| int | 执行结果 |

注意：该函数不会自动创建文件夹，若路径中存在尚未创建的文件夹请手动创建。

### points::get\_points

描述：获取三维点云数据，数据以float类型三维数组float[h][w][3]的形式存储。并将数组指针作为返回值返回给调用者。具体数据格式如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| points[0][0][(x)0] | points[0][0][(y)1] | points[0][0][(z)2] | ... | points[0][w-1][0] | points[0][w-1][1] | points[0][w-1][2] |
| points [1][0][0] | points[1][0][1] | points[1][0][2] | ... | points[1][w-1][0] | points[1][w-1][1] | points[1][w-1][2] |
| points [h-1][0][0] | points[h-1][0][1] | points[h-1][0][2] | ... | points[h-1][w-1][0] | points[h-1][[w-1][1] | points[h-1][[w-1][2] |

语法：

float \*get\_points();

参数：无

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| float \* | 点云数组指针 |

### points::get\_length

描述：获取点云长度

语法：

int get\_length();

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| int | 点云长度 |

## SENSOR控制接口

### sensor::set\_option

描述：配置sensor功能属性值

语法：

int set\_option(sy3\_option option, uint16\_t value,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| option | 功能项枚举 | 输入 |
| value | 设置的值 | 输入 |
| error | 执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| int | 是否成功 |

### sensor::set\_option

描述：配置sensor功能项的极值

语法：

int set\_option(sy3\_option option, uint16\_t max, uint16\_t min,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| option | 功能项枚举 | 输入 |
| max | 最大值 | 输入 |
| min | 最小值 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| int | 是否成功 |

### sensor::get\_option

描述：获取sensor功能属性极值范围

语法：

int get\_option(sy3\_option option, uint16\_t &max, uint16\_t &min,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| option | 功能项枚举 | 输入 |
| max | 最大值 | 输入 |
| min | 最小值 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| int | 是否成功 |

### sensor::get\_option

描述：获取sensor功能属性极值

语法：

int get\_option(sy3\_option option, uint16\_t &value,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| option | 功能项枚举 | 输入 |
| value | 属性值 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| int | 是否成功 |

### sensor::set\_filter\_value

描述：滤波函数

语法：

set\_filter\_value(FilterType filter\_type,FILTER\_THRESHOLD threshold\_value,int num)

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| filter\_type | 滤波类型 | 输入 |
| threshold\_value | 滤波类型 | 输入 |
| num | 滤波参数长度 | 输入 |

### sensor::get\_filter\_value

描述：滤波函数

语法：

get\_filter\_value(FilterType filter\_type,FILTER\_THRESHOLD threshold\_value, int& num)

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| filter\_type | 滤波类型 |  |
| threshold\_value | 滤波类型 |  |
| num | 滤波参数长度 |  |

滤波函数参数设置说明：

例: 幅值滤波 AMPLITITUD

FILTER\_THRESHOLD threshold\_value{ 0 };

threshold\_value[0] = 10;// amplititud\_threshold

set\_filter\_value(FilterType::AMPLITITUD, threshold\_value, 1);

int num = 0;

get\_filter\_value(FilterType::AMPLITITUD, threshold\_value, num);

std::cout << "set amplititud\_threshold = " << threshold\_value[0] << std::endl;

例:中值滤波 MEDIAN

FILTER\_THRESHOLD threshold\_value{ 0 };

threshold\_value[0] = 3;// median\_ksize

threshold\_value[1] = 1;// median\_iterations

set\_filter\_value(FilterType::MEDIAN, threshold\_value, 2);

int num = 0;

get\_filter\_value(FilterType::MEDIAN, threshold\_value, num);

std::cout << "set median\_ksize = " << threshold\_value[0] << " median\_iterations = " << threshold\_value[1] << std::endl;

例:高斯滤波 GAUSS

FILTER\_THRESHOLD threshold\_value{ 0 };

threshold\_value[0] = 3;// median\_ksize

threshold\_value[1] = 1;// median\_iterations

set\_filter\_value(FilterType::GAUSS, threshold\_value, 2);

int num = 0;

get\_filter\_value(FilterType::GAUSS, threshold\_value, num);

std::cout << "set gauss\_ksize = " << threshold\_value[0] << " gauss\_iterations = " << threshold\_value[1] << std::endl;

例:边界滤波 EDGE

sy3::FILTER\_THRESHOLD threshold\_value{ 0 };

threshold\_value[0] = 50;//edge\_threshold

set\_filter\_value(sy3::FilterType::EDGE, threshold\_value, 1);

int num = 0;

get\_filter\_value(sy3::FilterType::EDGE, threshold\_value, num);

std::cout << "set edge\_threshold = " << threshold\_value[0] << std::endl;

例:斑点滤波 SPECKLE

// speckle\_max\_diff threshold\_value[0] = 40; speckle\_size threshold\_value[1] = 100

sy3::FILTER\_THRESHOLD threshold\_value{ 0 };

threshold\_value[0] = 40;// speckle\_size

threshold\_value[1] = 100;// speckle\_max\_diff

set\_filter\_value(sy3::FilterType::SPECKLE, threshold\_value, 2);

int num = 0;

get\_filter\_value(sy3::FilterType::SPECKLE, threshold\_value, num);

std::cout << "Set speckle\_size = " << threshold\_value[0] << " speckle\_max\_diff = " << threshold\_value[1] << std::endl;

例:Sobel滤波 SOBEL

sy3::FILTER\_THRESHOLD threshold\_value{ 0 };

threshold\_value[0] = 150;// sobel\_threshold

set\_filter\_value(sy3::FilterType::SOBEL, threshold\_value, 1);

int num = 0;

get\_filter\_value(sy3::FilterType:: SOBEL, threshold\_value, num);

std::cout << "Set sobel\_threshold = " << threshold\_value[0] << std::endl;

例:边界滤波2 EDGE\_MAD

sy3::FILTER\_THRESHOLD threshold\_value{ 0 };

threshold\_value[0] = 15;// EDGE\_MAD\_threshold

set\_filter\_value(sy3::FilterType::EDGE\_MAD, threshold\_value, 1);

int num = 0;

get\_filter\_value(sy3::FilterType::EDGE\_MAD, threshold\_value, num);

std::cout << "set edge\_mad\_threshold = " << threshold\_value[0] << std::endl;

例:大津滤波 OKADA

sy3::FILTER\_THRESHOLD threshold\_value{ 0 };

threshold\_value[0] = 15;// EDGE\_MAD\_threshold

set\_filter\_value(sy3::FilterType::OKADA, threshold\_value, 1);

int num = 0;

get\_filter\_value(sy3::FilterType::OKADA, threshold\_value, num);

std::cout << "set okada\_diff = " << threshold\_value[0] << std::endl;

参数范围说明表：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 滤波接口 | 参数1-min | 参数1-max | 参数1推荐值 | 参数2-min | 参数2-max | 参数2推荐值 |
| AMPLITITUD | 0 | 100 | 6 |  |  |  |
| MEDIAN | 3 | 5 | 3 | 0 | 5 | 1 |
| EDGE | 20 | 200 | 50 |  |  |  |
| SPECKLE | 24 | 200 | 40 | 40 | 200 |  |
| GAUSS | 3 | 5 | 3 | 0 | 5 | 1 |
| EDGE\_MAD | 5 | 100 | 15 |  |  |  |
| SOBEL | 20 | 300 | 150 |  |  |  |
| OKADA | 10 | 100 | 10 |  |  |  |

滤波调用顺序：

CS20:中值、边界、斑点、中值

DepthFilter(depth, FilterType::MEDIAN);

DepthFilter(depth, FilterType::EDGE);

DepthFilter(depth, FilterType::MEDIAN);

CS30:前段内置了中值、边界、中值

DepthFilter(depth, FilterType::MEDIAN);

DepthFilter(depth, FilterType::EDGE);

DepthFilter(depth, FilterType::MEDIAN);

后端可再加入 斑点、中值滤波。

DepthFilter(depth, FilterType::SPECKLE);

DepthFilter(depth, FilterType::MEDIAN);

## 算法接口

### process\_engine::comptute\_points

描述：计算点云

语法：

points \*comptute\_points(depth\_frame \*depth,sy3\_error &error) const;

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| depth | 需要转换成点云的depth帧 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| points | 点云指针 |

注意：函数内部为点云申请了内存，如不再需要此点云数据需手动释放掉，即调用delete points

### process\_engine::align\_to\_rgb

描述：rgbd对齐

语法：

sy3::frameset \*align\_to\_rgb(depth\_frame \*depth,rgb\_frame \*rgb,sy3\_error &error);

参数：

|  |  |  |
| --- | --- | --- |
| 参数名称 | 描述 | 输入/输出 |
| depth | depth指针 | 输入 |
| rgb | rgb 指针 | 输入 |
| error | 函数执行状态 | 输出 |

返回值：

|  |  |
| --- | --- |
| 返回值 | 描述 |
| frameset \* | 对齐好的 rgb 与 depth集合 |

注意：目前仅支持rgb分辨率为1920x1080向depth分辨率为640x480进行映射。映射成功输出的align-rgb 分辨率为1920x1080，align-depth分辨率为1920x1080

## 数据类型

### sy3\_error

enum sy3\_error

{

SUCCESS = 0,

INVALID\_PID,

INVALID\_VID,

DEVICE\_NOT\_FOUND,

INVALID\_FORMAT,

INCONSISTENCY\_RES,

OPEN\_FAILED,

NOT\_IMPLEMENTED,

INVALID\_INSTANCE,

}sy3\_error;

参数：

|  |  |
| --- | --- |
| 参数名称 | 描述 |
| SUCCESS | 执行成功 |
| INVALID\_PID | 无效的PID |
| INVALID\_VID | 无效的VID |
| DEVICE\_NOT\_FOUND | 未发现设备 |
| INVALID\_FORMAT | 格式错误 |
| INCONSISTENCY\_RES | 分辨率不匹配 |
| OPEN\_FAILED | 打开失败 |
| NOT\_IMPLEMENTED | 接口未实现 |
| INVALID\_INSTANCE | 无效实例 |

### sy3\_device\_type

enum sy3\_device\_type

{

DEVICE\_CS30,

DEVICE\_CS20,

} sy3\_device\_type;

参数：

|  |  |
| --- | --- |
| 参数说明 | 描述 |
| DEVICE\_CS30 | CS30模组 |
| DEVICE\_CS20 | CS20 模组 |

### sy3\_camera\_info

enum sy3\_camera\_info

{

SY3\_CAMERA\_INFO\_NAME,

SY3\_CAMERA\_INFO\_SERIAL\_NUMBER,

SY3\_CAMERA\_INFO\_FIRMWARE\_VERSION,

SY3\_CAMERA\_INFO\_RECOMMENDED\_FIRMWARE\_VERSION,

SY3\_CAMERA\_INFO\_PRODUCT\_ID,

SY3\_CAMERA\_INFO\_COUNT

} sy3\_camera\_info;

参数：

|  |  |
| --- | --- |
| 参数说明 | 描述 |
| SY3\_CAMERA\_INFO\_NAME | Camera名称 |
| SY3\_CAMERA\_INFO\_SERIAL\_NUMBER | 串号 |
| SY3\_CAMERA\_INFO\_FIRMWARE\_VERSION | 固件版本号 |
| SY3\_CAMERA\_INFO\_RECOMMENDED\_FIRMWARE\_VERSION | 当前SDK建议使用的固件版本号 |
| SY3\_CAMERA\_INFO\_PRODUCT\_ID | 设备PID |

### sy3\_stream

enum sy3\_stream

{

SY3\_STREAM\_NONE,

SY3\_STREAM\_DEPTH=2,

SY3\_STREAM\_RGB,

SY3\_STREAM\_IR,

SY3\_STREAM\_COUNT,

} sy3\_stream;

描述：

|  |  |
| --- | --- |
| 参数说明 | 描述 |
| SY3\_STREAM\_NONE |  |
| SY3\_STREAM\_DEPTH | 深度数据流 |
| SY3\_STREAM\_RGB | RGB数据流 |
| SY3\_STREAM\_IR | IR数据流 |

### sy3\_format

struct sy3\_format {

sy3\_stream stream;

int width;

int height;

}sy3\_format;

描述：

|  |  |
| --- | --- |
| 参数说明 | 描述 |
| stream | 数据流类型 |
| width | 宽 |
| height | 高 |

### intrinsics

struct sy3\_intrinsics{

int width;

int height;

float ppx;

float ppy;

float fx;

float fy;

float coeffs[5];

} sy3\_intrinsics;

描述：

|  |  |
| --- | --- |
| 参数说明 | 描述 |
| width | width |
| height | height |
| ppx | ppx |
| ppy | ppy |
| fx | fx |
| fy | fy |
| coeffs[5] | coeffs |

### sy3\_option

typedef enum sy3\_option{

SY3\_OPTION\_EXPOSURE,

SY3\_OPTION\_EXPOSURE\_RANGE,

SY3\_OPTION\_DISTANCE\_RANGE,

SY3\_OPTION\_DEFAULT\_DISTANCE\_RANGE,

SY3\_OPTION\_RGB\_IMAGE\_FLIP,

SY3\_OPTION\_RGB\_IMAGE\_MIRROR,

SY3\_OPTION\_TOF\_IMAGE\_FLIP,

SY3\_OPTION\_TOF\_IMAGE\_MIRROR,

SY3\_OPTION\_DEPTH\_IMAGE\_MIRROR,

SY3\_OPTION\_DEPTH\_IMAGE\_FILTER,

SY3\_OPTION\_COUNT,

} sy3\_option;

描述：

|  |  |
| --- | --- |
| 参数说明 | 描述 |
| SY3\_OPTION\_EXPOSURE | 设置曝光时间,单位us |
| SY3\_OPTION\_EXPOSURE\_RANGE | 设置曝光时间范围 |
| SY3\_OPTION\_DISTANCE\_RANGE | 设置显示的距离范围 |
| SY3\_OPTION\_DEFAULT\_DISTANCE\_RANGE | 设置默认的距离范围 |
| SY3\_OPTION\_RGB\_IMAGE\_FLIP | 设置Rgb flip |
| SY3\_OPTION\_RGB\_IMAGE\_MIRROR | 设置Rgb mirror |
| SY3\_OPTION\_TOF\_IMAGE\_FLIP | 设置Tof flip |
| SY3\_OPTION\_TOF\_IMAGE\_MIRROR | 设置Tof mirror |
| SY3\_OPTION\_DEPTH\_IMAGE\_MIRROR | 设置depth mirror |
| SY3\_OPTION\_DEPTH\_IMAGE\_FILTER | 设置depth filter |

# 示例

## 获取depth 帧

//仅截取关键代码,详细代码请参阅samples 源码

#include "libsynexens3/libsynexens3.h"

int main(int argc, char \*\*argv)

{

sy3::sy3\_error e;

printf("version:%s \n", sy3::sy3\_get\_version(e));

sy3::context \*ctx = sy3::sy3\_create\_context(e);

sy3::device \*dev = ctx->query\_device(e);

if (e != sy3::sy3\_error::SUCCESS) {

printf("error:%s \n", sy3::sy3\_error\_to\_string(e));

return 0;

}

sy3::pipeline \*pline = sy3::sy3\_create\_pipeline(ctx, e);

sy3::config \*cfg = sy3\_create\_config(e);

cfg->enable\_stream(sy3::sy3\_stream::SY3\_STREAM\_DEPTH,640, 480, e);

pline->start(cfg, e);

bool quit = false;

while (!quit)

{

sy3::frameset \*frameset = pline->wait\_for\_frames(SY3\_DEFAULT\_TIMEOUT, e);

sy3::depth\_frame \*depth\_frame = frameset->get\_depth\_frame();

show\_depth\_frame(depth\_frame);

delete frameset;

if (cv::waitKey(1) == 'q') {

pline->stop(e);

quit = true;

}

}

system("pause");

return 0;

}

## 获取align数据

//仅截取关键代码,详细代码请参阅samples 源码

#include "libsynexens3/libsynexens3.h"

int main(int argc, char \*\*argv)

{

sy3::sy3\_error e;

sy3::context \*ctx = sy3::sy3\_create\_context(e);

sy3::device \*dev = ctx->query\_device(e);

if (e != sy3::sy3\_error::SUCCESS) {

return 0;

}

sy3::pipeline \*pline = sy3::sy3\_create\_pipeline(ctx, e);

sy3::config \*cfg = sy3\_create\_config(e);

cfg->enable\_stream(sy3::sy3\_stream::SY3\_STREAM\_DEPTH, 640, 480, e);

cfg->enable\_stream(sy3::sy3\_stream::SY3\_STREAM\_RGB, 1920, 1080, e);

pline->start(cfg, e);

bool quit = false;

while (!quit)

{

switch (cv::waitKey(1)) {

case 'q': {

pline->stop(e);quit = true;

}break;

case 'e': {

//设置曝光时间，单位us

dev->get\_sensor(e)->set\_option(sy3::sy3\_option::SY3\_OPTION\_EXPOSURE, 10, e);

}break;

default: break;

}

sy3::frameset \*frameset = pline->wait\_for\_frames(SY3\_DEFAULT\_TIMEOUT, e);

sy3::depth\_frame \*depth\_frame = frameset->get\_depth\_frame();

sy3::rgb\_frame \*rgb\_frame = frameset->get\_rgb\_frame();

sy3::process\_engine \*engine = pline->get\_process\_engin(e)

sy3::frameset \*align\_set=engine->align\_to\_rgb(depth\_frame,rgb\_frame,e);

show\_depth\_frame(align\_set->get\_depth\_frame(), "algin\_depth");

show\_rgb\_rgb\_frame(align\_set->get\_rgb\_frame(), "algin\_rgb");

delete frameset;

delete align\_set;

}

return 0;}

## 免责声明

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。本公司对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销或特定用途的适用性的声明或担保。本公司对因这些信息及使用这些信息而引起的后果不承担任何责任。未经本公司书面批准，不得将该产品用作生命维持系统中的关键组件。