|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Revised Date | Adapt SDK version | Description | Editor |
| v1.0 | July4,2022 | v0.1.2 | Initial Version | Kai |
| v1.1 | July5,2022 | v0.1.3 | Add interface description | Kai |
| v1.2 | July11,2022 | v0.1.3 | Add ubuntu environment configuration | Kai |
| v1.3 | February 17,2023 | v0.7.3 | Add some new interface | Kai |

**Libsynexens3 SDK instructions**

**Contents**

[1. Descriptions 1](#_Toc129375289)

[2. Windows environment configuration 2](#_Toc129375290)

[2.1. Windows environment configuration（vs2017 as example） 2](#_Toc129375291)

[2.2. Ubuntu environment configuration（cmake as example） 4](#_Toc129375292)

[2.3. Calling process diagram 7](#_Toc129375293)

[3. API Reference 7](#_Toc129375294)

[3.1. Global interface 7](#_Toc129375295)

[3.2. Basic interface 10](#_Toc129375296)

[3.3. SENSOR control interface 19](#_Toc129375297)

[3.4. Algorithm interface 24](#_Toc129375298)

[3.5. Date type 25](#_Toc129375299)

[4. Sample code 30](#_Toc129375300)

[4.1. Get depth frame 30](#_Toc129375301)

[4.2. Get align 31](#_Toc129375302)

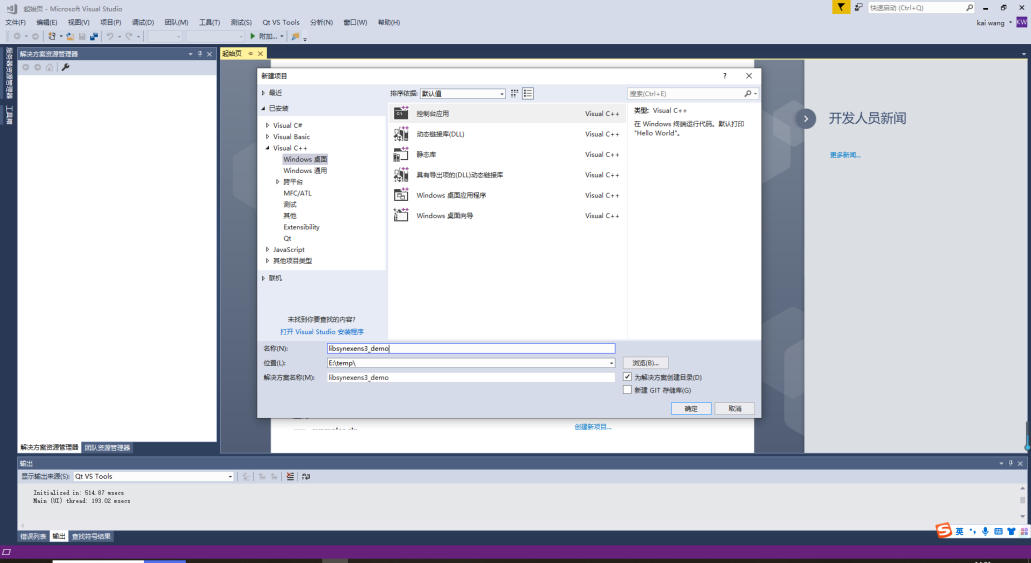
# Descriptions

This document introduces code interfaces and demo code for users, the SDK adapt for CS series camera.

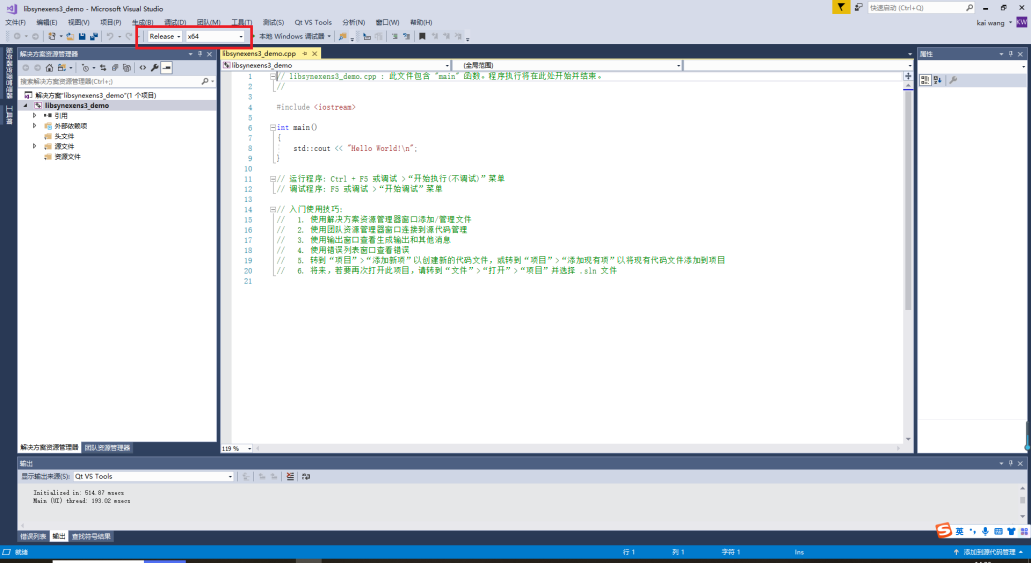
# Windows environment configuration

## Windows environment configuration（vs2017 as example）

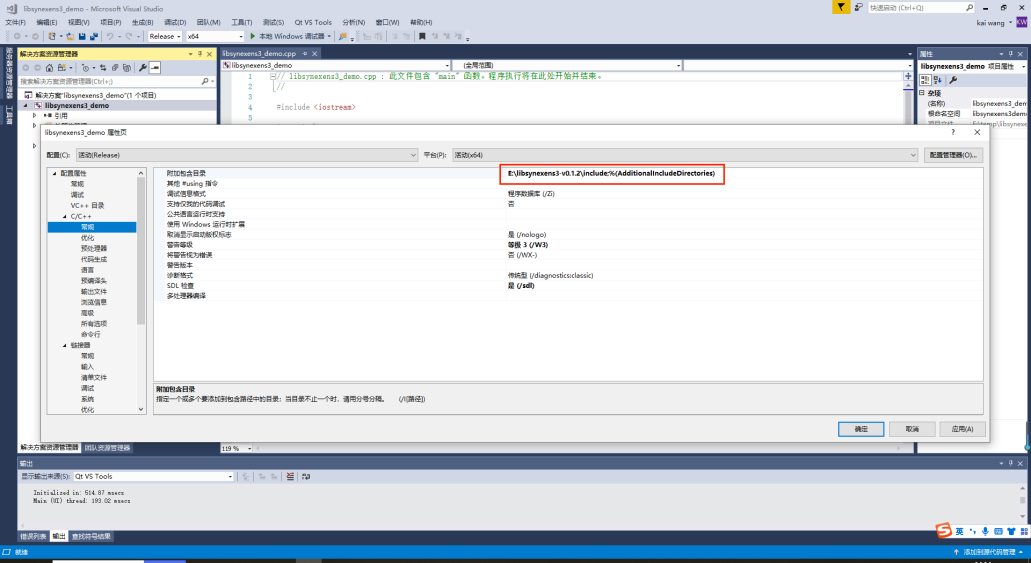
### Create VS project.

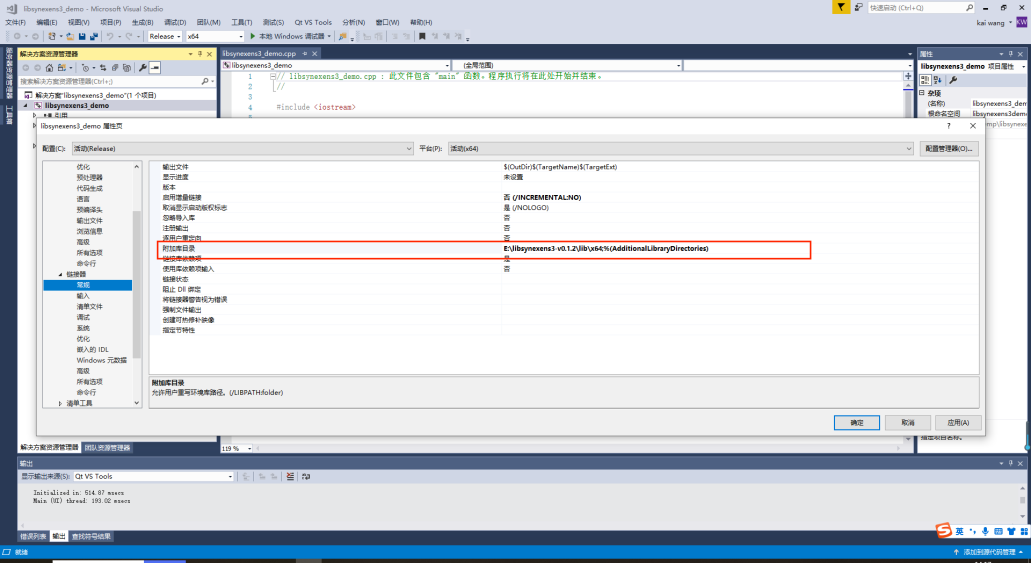


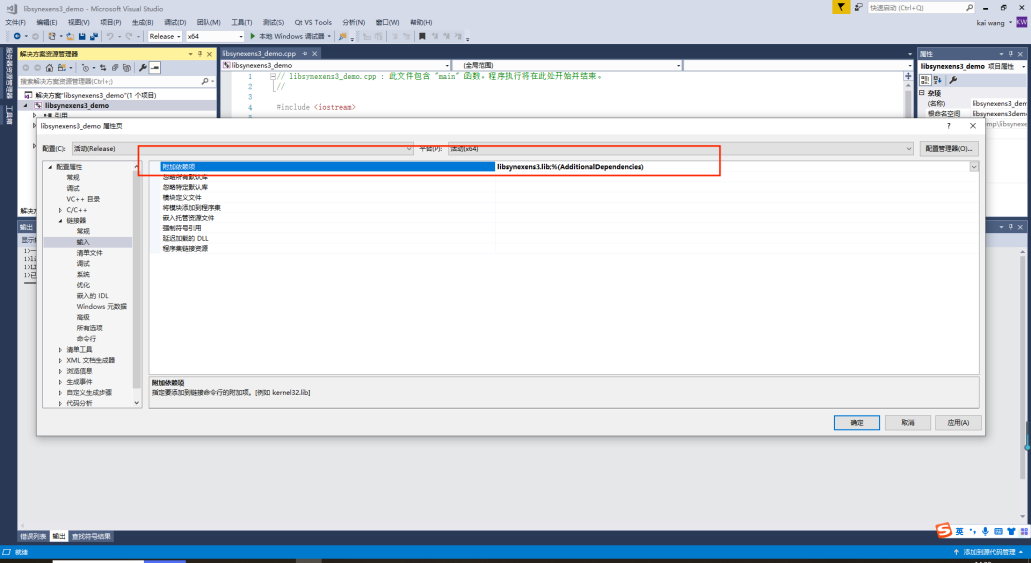
### Select the solution and platform corresponding to the SDK library.



### Configure the header file path and library path of the SDK in the project properties.







### After the configuration is completed, you can enter.

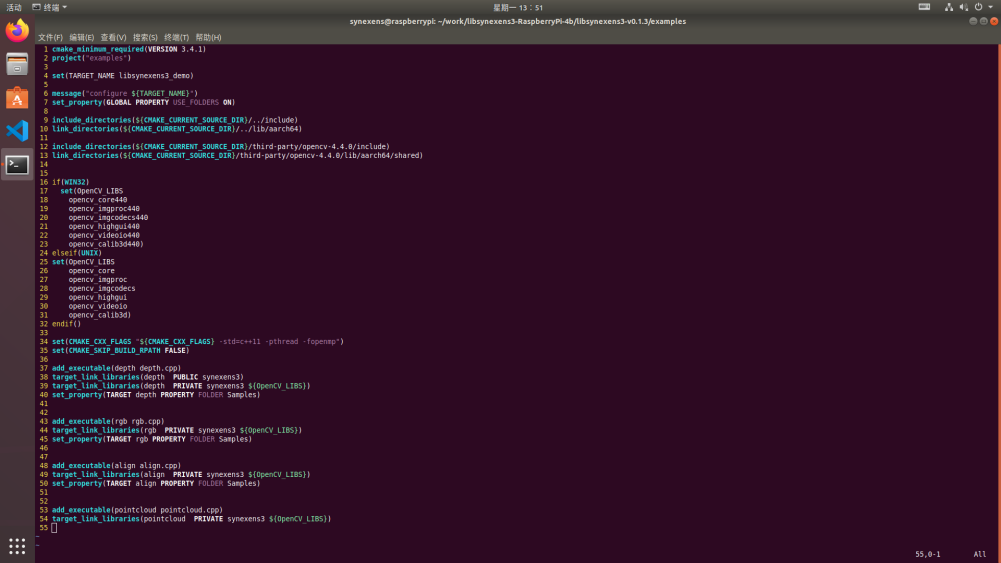
## Ubuntu environment configuration（cmake as example）

### Installation Dependencies

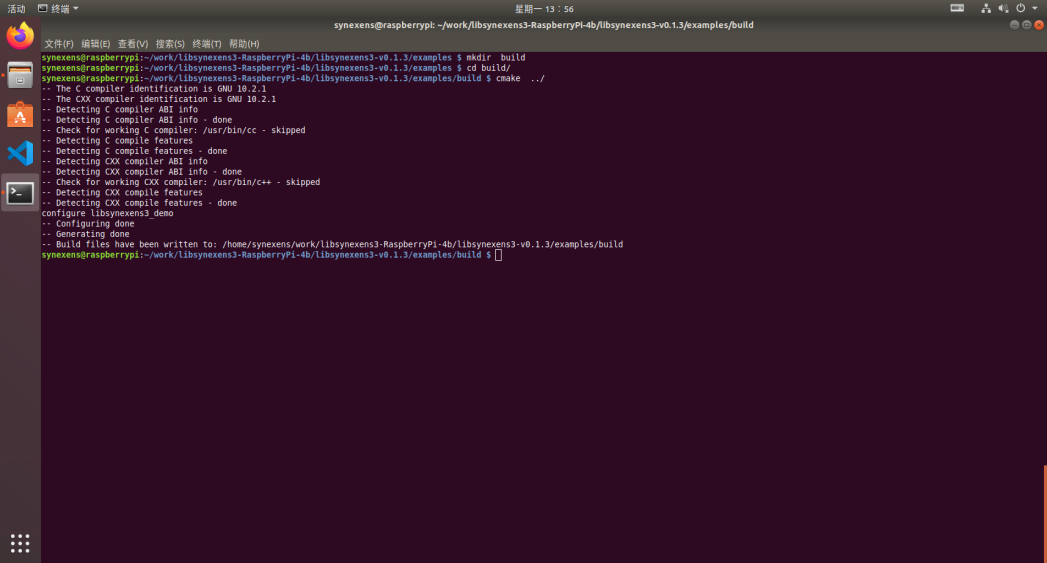
sudo apt-get install libusb-1.0-0-dev

sudo apt-get install libudev-dev

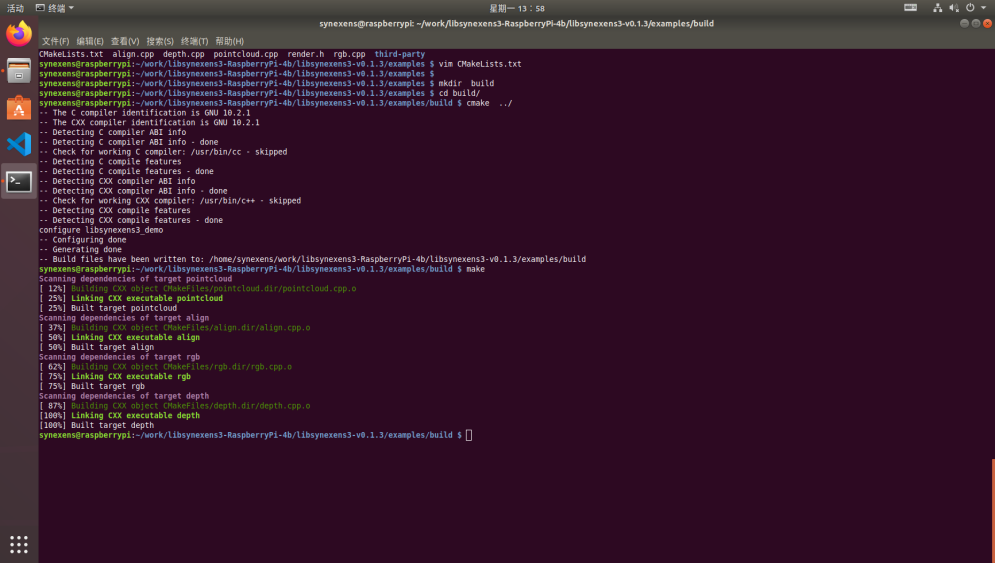
### Compile CmakeLists.txt，the process requires familiarity with cmake syntax



### Create Compiled Project File



### make

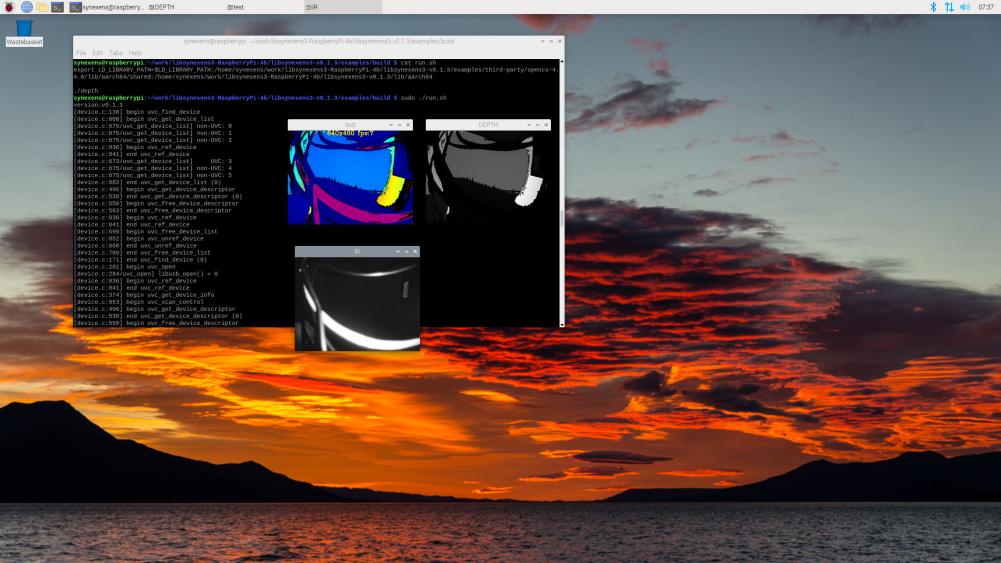


### Execute the executable file to test the effect

LD\_ LIBRARY\_ PATH needs to be configured before executing the program, in order to find the library files that the program depends on, the example has written run.sh script to facilitate program execution.

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

./depth



## Calling process diagram

Currently, only polling mode calls are supported

Start

Create Context

Sy3\_create\_context ()

Query device

Context::query\_device())

Create pipeline

Sy3\_create\_pipeline()

Configure data flow

Config::enable\_stream()

Start-up pipeline

Pipeline::strart()

Get frame gather

Pipleline::wait\_for\_frames()

Close pipeline

Pipeline::stop()

END

# API Reference

## Global interface

### get\_device\_info

Description：Obtain devices information

Grammar：

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

Parameters：

|  |  |  |
| --- | --- | --- |
| Parameters name | Descriptions | Input/Output |
| error | Function execution status | Output |

The return value：

|  |  |
| --- | --- |
| Return value | Description |
| device\_info | Device information |

### query\_device

Descriptions：query device

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| device | Device pointer |

### sy3\_create\_pipeline

Description：Create routes

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| ctx | Context currently in use | Input |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| pipeline | Pipeline pointer |

### sy3\_create\_config

Description: Creates a configuration parameter pointer

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| config | Configuration parameter pointer |

### sy3\_get\_device\_info

Description: Get device information

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| device | Device pointer | Input |
| info | Info Enumeration | Input |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| const char\* | Corresponding device information string |

Note: The call is invalid until the device is found successfully

## Basic interface

### device::get\_sensor

Description: Get the sensor pointer

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters | Description | Input/Output |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| sensor \* | sensor pointer |

### device::get\_type

Description: Get device type

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters | Description | Input/Output |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| sy3\_device\_type \* | Device type |

### device::get\_support\_stream

Description: Get the list of data streams supported by the device

Grammar:

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

Parameters：

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| const std::vector<sy3\_stream> | Data flow list |

### device::get\_support\_format

Description: Get supported data formats

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| const std::vector<sy3\_format> | Data flow list |

### device::get\_support\_format

Description: Get the format supported by the specified data stream of the device

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| stream | Data flow type | Input |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| const std::vector<sy3\_format> | Data format list |

### config::enable\_stream

Description: Enables the specified data flow

Grammar:

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

Parameters：

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| stream | Data type to enable | Input |
| width | Width of data stream image | Input |
| height | Height of data stream image | Input |
| error | Function execution status | Output |

Return value: None

Note: This function is only called valid before the pipeline is started. It is invalid when called during the pipeline running.

### config::disable\_stream

Description: Disables the specified data flow

Grammar:

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

Parameters：

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| stream | Type of data flow to disable | Input |
| error | Function execution status | Output |

Return value: None

### config::disable\_all\_streams

Description: Disable all data flows

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| error | Function execution status | Output |

Return value: None

### pipeline::start

Description: Start pipeline

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| cfg | Pipeline configure | Input |
| error | Function execution status | Output |

Return value: None

### pipeline::get\_process\_engin

Description: Get the algorithm instance pointer

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| process\_engine\* | Algorithm instance pointer |

### pipeline::stop

Description: stop pipeline

Grammar:

void stop(sy3\_error &error);

Parameters：

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| error | Function execution status | Output |

Return value: None

### pipeline::wait\_for\_frames

Description: Get pipeline frame set

Grammar：

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

Parameters：

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| timeout\_ms | overtime | Input |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| frameset \* | Frame gather |

### pipeline::get\_device

Description: Get the current device

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| const device \* | Device pointer |

### frameset::get\_depth\_frame

Description: Get depth frame

Grammar:

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

Parameters: none

The return values

|  |  |
| --- | --- |
| Return value | Description |
| depth\_frame \* | Depth frame data pointer |

### frameset::get\_ir\_frame

Description: Get ir frame

Grammar:

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

Parameters: none

The return values

|  |  |
| --- | --- |
| Return value | Description |
| ir\_frame \* | Ir frame data pointer |

### frameset::get\_raw\_frame

Description: Get device information

Grammar:

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

Parameters: none

The return value

|  |  |
| --- | --- |
| Return value | Description |
| raw\_frame \* | raw frame data pointer |

Note: At present, raw\_ Frame data is only open to internal calibration, and external users cannot obtain raw data

### frame::get\_width

Description: Get the width of image

Grammar:

const int get\_width();

Parameters: none

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| const int | Width |

### frame::get\_height

Description: Get the height of image

Grammar:

const int get\_height() ;

Parameters: none

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| const int | Height |

### frame::get\_type

Description: Get the type of frame

Grammar:

const sy3\_stream get\_type() ;

Parameters: none

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| const sy3\_stream | Frame type |

### depth\_frame::get\_data

Description: Obtain the frame image array, store the data in the form of uint16 type two-dimensional array (uint16 [h] [w]), and return the array pointer to the caller as the return value. The specific data format is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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] |

Grammar：

void \*get\_data();

Parameters: none

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| void \* | Frame data pointer |

Note: The return value type is void \*, and you need to manually convert it to uint16\*

### depth\_frame::apply\_colormap

Description: Obtain the data mapped rgb (BGR format), store the data in the form of uint8 type 3D array (uint8 [h] [w] [3]), and return the array pointer to the caller as the return value. The specific data format is as follows:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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] |

Grammar:

uint8\_t \*apply\_colormap();

Parameters: none

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| uint8\_t \* | rgb image numeric pointer |

### frame::get\_profile

Description: Get frame configure

Grammar：

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

Parameters: none

The return values

|  |  |
| --- | --- |
| Return value | Description |
| const stream\_profile \* | Frame configuration pointer |

### frame::dump

Description: Save frames locally for debugging

Grammar:

int dump(const char \*filenam) ;

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| filenam | File name to save | Input |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| int | execution result |

Note: This function will not automatically create folders. If there are folders in the path that have not been created, please create them manually.

### points::get\_points

Description: Obtain 3D point cloud data, which is stored in the form of float type 3D array float [h] [w] [3]. The array pointer is returned to the caller as a return value. The specific data format is as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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] |

Grammar：

float \*get\_points();

Parameters: none

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| float \* | Point cloud array pointer |

### points::get\_length

Description: Get point cloud length

Grammar:

int get\_length();

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| int | Point cloud length |

## SENSOR control interface

### sensor::set\_option

Description: configure sensor function attribute value

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| option | Function Item Enumeration | Input |
| value | Set value | Input |
| error | Execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| int | whether or not success |

### sensor::set\_option

Description: Configure the extreme value of the sensor function item

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| option | Function Item Enumeration | Input |
| max | Max | Input |
| min | Min | Input |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| int | whether or not success |

### sensor::get\_option

Description: Get the extreme value range of sensor function attribute

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| option | Function Item Enumeration | Input |
| max | Max | Input |
| min | Min | Input |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| int | whether or not success |

### sensor::get\_option

Description: Get the extreme value of sensor function attribute

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| option | Function Item Enumeration | Input |
| value | Attribute Value | Input |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| int | whether or not success |

### sensor::set\_filter\_value

Description: filter function

Grammar:

set\_filter\_value(FilterType filter\_type,FILTER\_THRESHOLD

threshold\_value,int num)

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| filter\_type | Filter type | Input |
| threshold\_value | Filter type | Input |
| num | Filter parameters length | Input |

### sensor::get\_filter\_value

Description: filter function

Grammar:

set\_filter\_value(FilterType filter\_type,FILTER\_THRESHOLD

threshold\_value,int num)

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| filter\_type | Filter type |  |
| threshold\_value | Filter type |  |
| num | Filter parameters length |  |

Filter function parameters setting description:

Sample: Amplitude filter 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;

Sample: Median filter 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;

Sample: Gauss filter 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;

Sample: Edge filter 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;

Sample: Speckle filter 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;

Sample: Sobel filter 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;

Sample: Edge filter 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;

Sample: Okada filter 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;

Parameters range table:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Parameters interface | Parameters1  Min | Parameters1  Max | Parameters1  recommended value | Parameters2  Min | Parameters2  Max | Parameters2  recommended value |
| 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 |  |  |  |

Filter call sequence:

CS20: Median , Edge, Speckle, Median

DepthFilter(depth, FilterType::MEDIAN);

DepthFilter(depth, FilterType::EDGE);

DepthFilter(depth, FilterType::MEDIAN);

CS30: SOC run Median, Edge, Median

DepthFilter(depth, FilterType::MEDIAN);

DepthFilter(depth, FilterType::EDGE);

DepthFilter(depth, FilterType::MEDIAN);

Call Speckle , Median again in other platforms

DepthFilter(depth, FilterType::SPECKLE);

DepthFilter(depth, FilterType::MEDIAN);

## Algorithm interface

### process\_engine::comptute\_points

Description: Compute point clouds

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| depth | Depth frame to be converted to point cloud | Input |
| error | Function execution status | Output |

The return value:

|  |  |
| --- | --- |
| Return value | Description |
| points | Points cloud pointer |

Note: The function has applied for memory for the point cloud internally. If the point cloud data is no longer needed, it needs to be released manually, that is, call delete points

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

Description: rgbd alignment

Grammar:

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

Parameters:

|  |  |  |
| --- | --- | --- |
| Parameters name | Description | Input/Output |
| depth | Depth pointer | Input |
| rgb | rgb pointer | Input |
| error | Function execution status | Output |

The return values:

|  |  |
| --- | --- |
| Return value | Description |
| frameset \* | Aligned rgb and depth sets |

Note: Currently, only mapping from rgb resolution 1920x1080 to depth resolution 640x480 is supported. The align rgb resolution and align depth resolution of the mapping output are 1920x1080 and 1920x1080 respectively.

## Date type

### 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;

Parameters:

|  |
| --- |
| Parameters name |
| SUCCESS |
| INVALID\_PID |
| INVALID\_VID |
| DEVICE\_NOT\_FOUND |
| INVALID\_FORMAT |
| INCONSISTENCY\_RES |
| OPEN\_FAILED |
| NOT\_IMPLEMENTED |
| INVALID\_INSTANCE |

### sy3\_device\_type

enum sy3\_device\_type

{

DEVICE\_CS30,

DEVICE\_CS20,

} sy3\_device\_type;

Parameters：

|  |
| --- |
| Parameter description |
| DEVICE\_CS30 |
| DEVICE\_CS20 |

### 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;

Parameters:

|  |
| --- |
| Parameters description |
| 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\_stream

enum sy3\_stream

{

SY3\_STREAM\_NONE,

SY3\_STREAM\_DEPTH=2,

SY3\_STREAM\_RGB,

SY3\_STREAM\_IR,

SY3\_STREAM\_COUNT,

} sy3\_stream;

Parameters:

|  |
| --- |
| Parameters description |
| SY3\_STREAM\_NONE |
| SY3\_STREAM\_DEPTH |
| SY3\_STREAM\_RGB |
| SY3\_STREAM\_IR |

### sy3\_format

struct sy3\_format {

sy3\_stream stream;

int width;

int height;

}sy3\_format;

Parameters:

|  |
| --- |
| Parameters description |
| stream |
| width |
| height |

### intrinsics

struct sy3\_intrinsics{

int width;

int height;

float ppx;

float ppy;

float fx;

float fy;

float coeffs[5];

} sy3\_intrinsics;

Parameters:

|  |
| --- |
| Parameters description |
| width |
| height |
| ppx |
| ppy |
| fx |
| fy |
| coeffs[5] |

### 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;

Description:

|  |  |
| --- | --- |
| Parameters description | Description |
| SY3\_OPTION\_EXPOSURE | Exposure time; unit: us |
| SY3\_OPTION\_EXPOSURE\_RANGE | Exposure time range |
| SY3\_OPTION\_DISTANCE\_RANGE | Displayed distance range |
| SY3\_OPTION\_DEFAULT\_DISTANCE\_RANGE | Default distance range |
| SY3\_OPTION\_RGB\_IMAGE\_FLIP | Rgb flip |
| SY3\_OPTION\_RGB\_IMAGE\_MIRROR | Rgb mirror |
| SY3\_OPTION\_TOF\_IMAGE\_FLIP | Tof flip |
| SY3\_OPTION\_TOF\_IMAGE\_MIRROR | Tof mirror |
| SY3\_OPTION\_DEPTH\_IMAGE\_MIRROR | depth mirror |
| SY3\_OPTION\_DEPTH\_IMAGE\_FILTER | depth filter |

# Sample code

## Get depth frame

//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;

}

## Get align

//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;}