Libsynexens3 SDK instructions

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

Contents

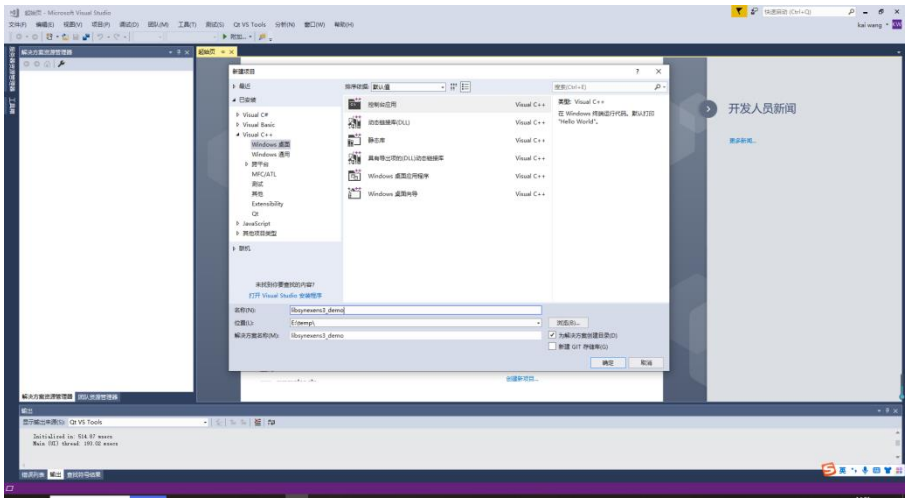
1. Descriptions

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

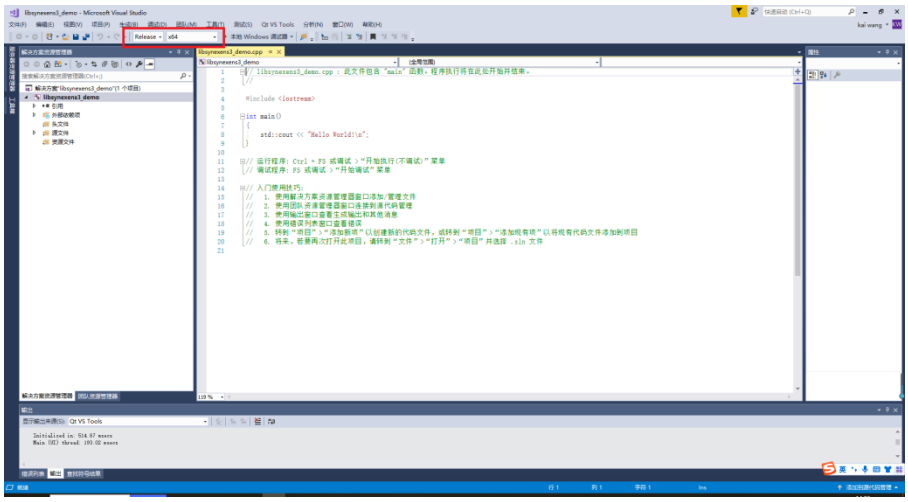
2. Windows environment configuration

2.1. Windows environment configuration（vs2017 as example）

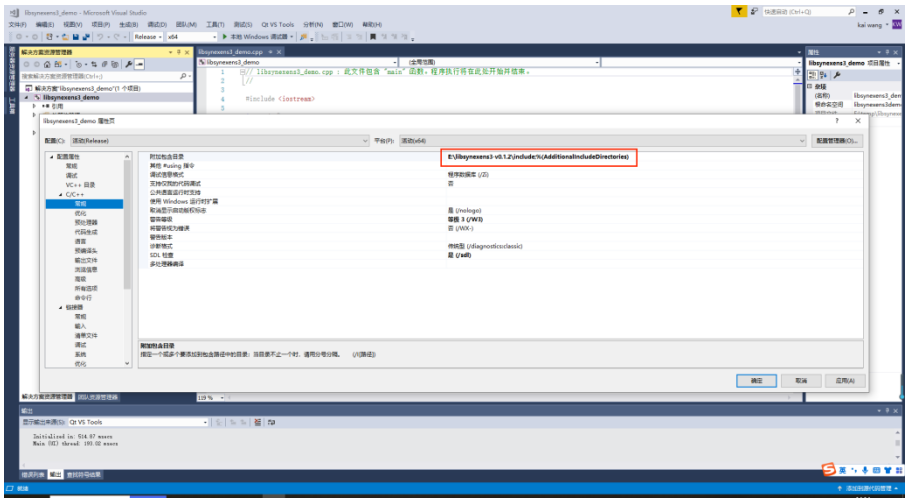
2.1.1. Create VS project.

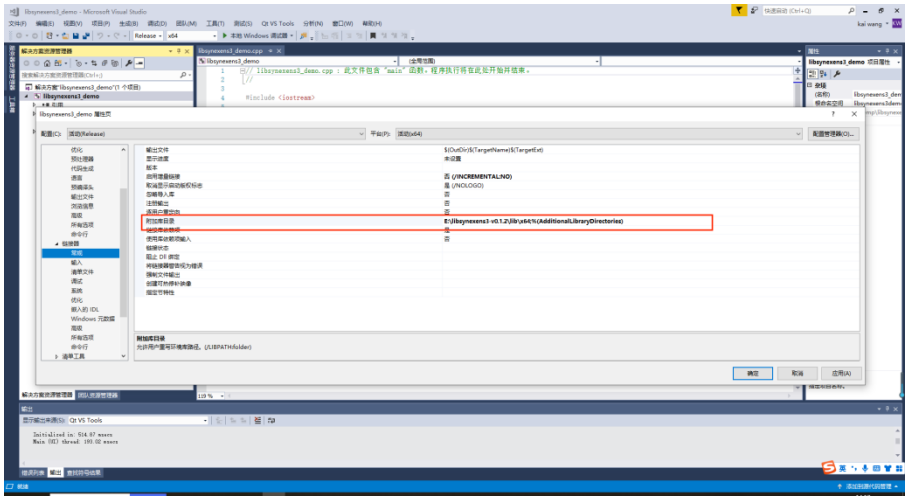


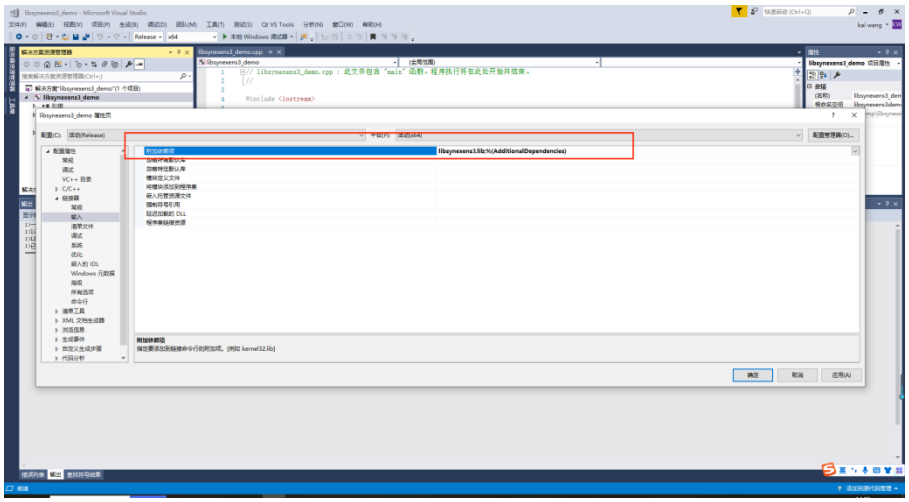
2.1.2. Select the solution and platform corresponding to the SDK library.



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







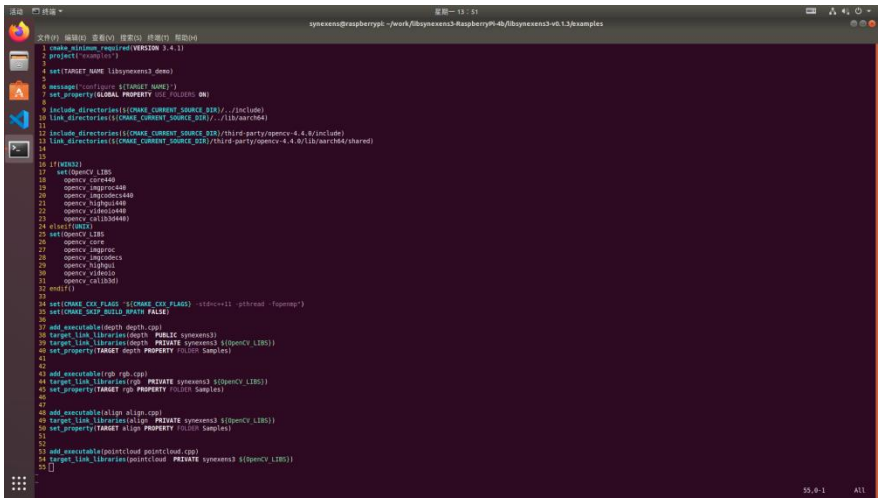
2.1.4. After the configuration is completed, you can enter.

2.2. Ubuntu environment configuration（cmake as example）

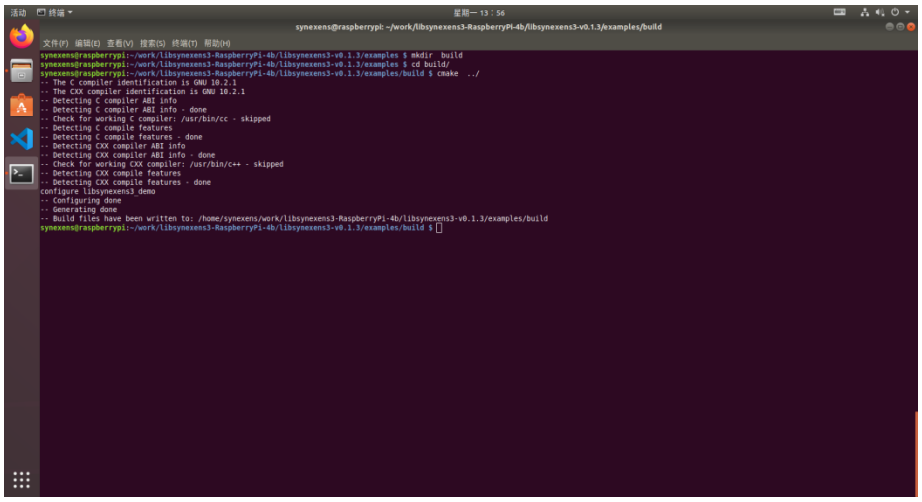
2.2.1. Installation Dependencies

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

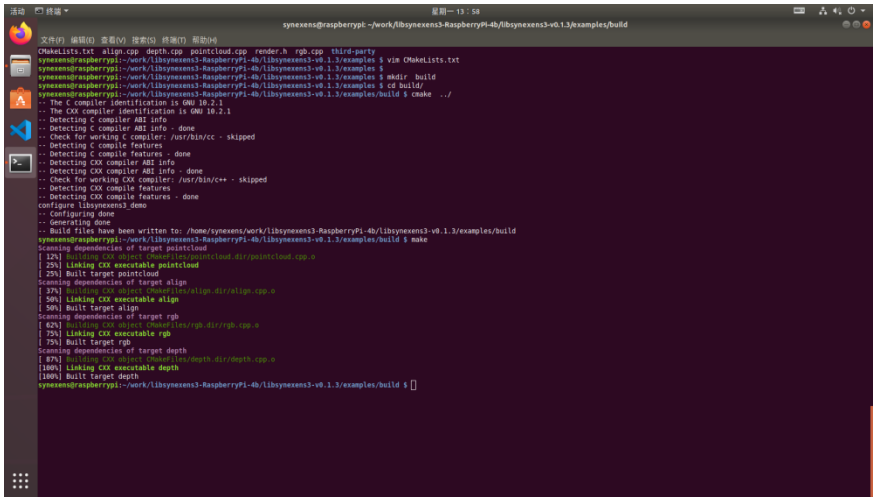
2.2.2. Compile CmakeLists.txt，the process requires familiarity with cmake syntax



2.2.3. Create Compiled Project File



2.2.4. make

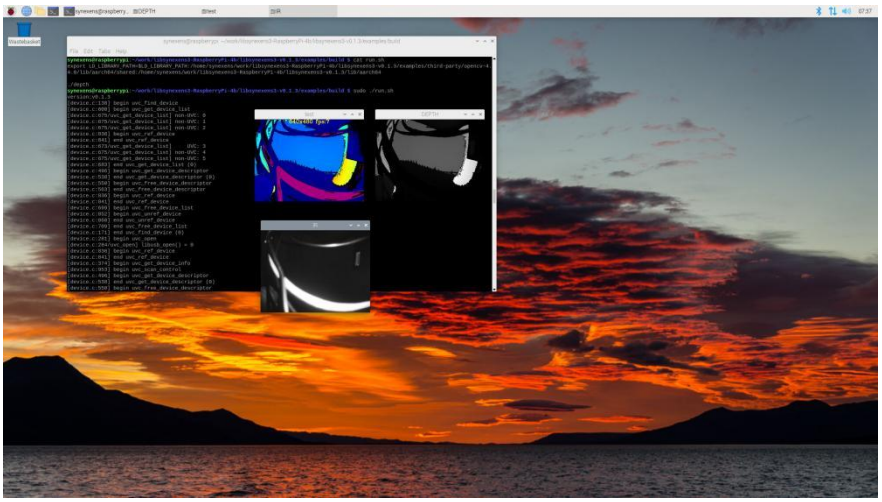


2.2.5. Execute the executable file to test the effect

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.

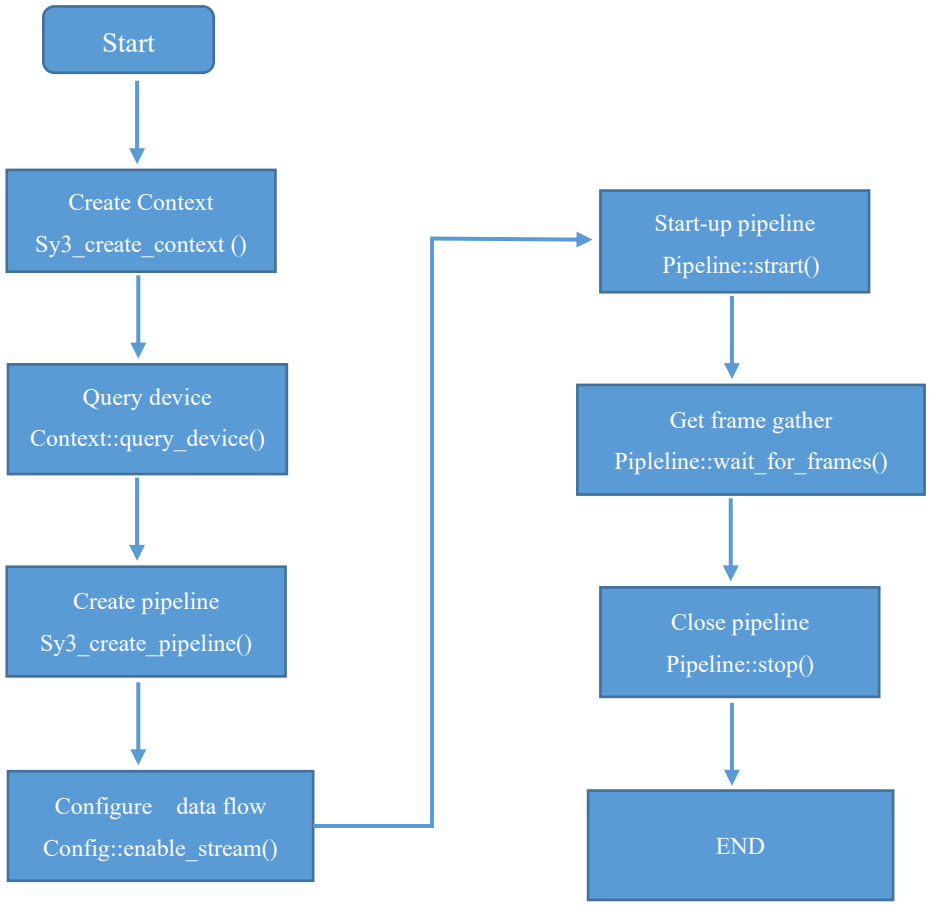
```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:../lib/:../../../third-party/ubuntu18.04_x64/opencv-4.4.0/lib/x64/shared/

./depth
```



2.3. Calling process diagram

Currently, only polling mode calls are supported



3. API Reference

3.1. Global interface

3.1.1. 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

3.1.2. 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

3.1.3. 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

3.1.4. 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

3.1.5. 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

3.2. Basic interface

3.2.1. 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

3.2.2. 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

3.2.3. 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

3.2.4. 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

3.2.5. 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

3.2.6. 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.

3.2.7. 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

3.2.8. 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

3.2.9. 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

3.2.10. 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

3.2.11.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

3.2.12.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

3.2.13.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

3.2.14.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

3.2.15.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

3.2.16.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

3.2.17.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

3.2.18.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

3.2.19.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

3.2.20.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:

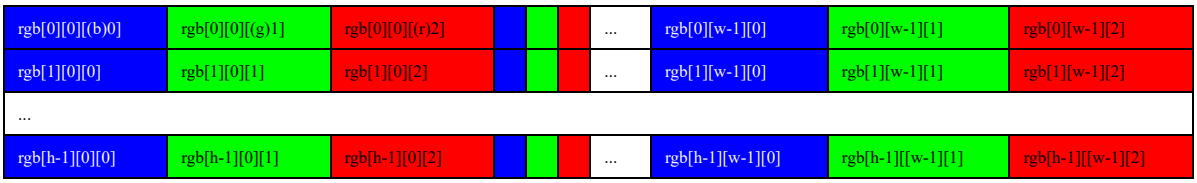
Return value	Description

void *	Frame data pointer

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

3.2.21.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:



Grammar:
uint8_t *apply_colormap();

Parameters: none

The return value:

Return value	Description
uint8_t *	rgb image numeric pointer

3.2.22.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

3.2.23.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.

3.2.24.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

3.2.25.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

3.3. SENSOR control interface

3.3.1. 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

3.3.2. 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

3.3.3. 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

3.3.4. 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

3.3.5. 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

3.3.6. 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);

3.4. Algorithm interface

3.4.1. 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

3.4.2. 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.

3.5. Date type

3.5.1. 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

3.5.2. sy3_device_type

```
enum sy3_device_type
{
    DEVICE_CS30,
    DEVICE_CS20,
} sy3_device_type;
```

Parameters：

Parameter description
DEVICE_CS30
DEVICE_CS20

3.5.3. 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

3.5.4. 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

3.5.5. sy3_format

```
struct sy3_format {
    sy3_stream stream;
    int width;
    int height;
}sy3_format;
```

Parameters:

Parameters description
stream
width
height

3.5.6. 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]

3.5.7. 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

4. Sample code

4.1. 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;
}
```

4.2. 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;}
```