SLAMTEC

RPLIDAR

Low Cost 360 Degree Laser Range Scanner

Introduction to Standard SDK

Contents

This document introduces the open source RPLIDAR standard SDK. The SDK can be used in Windows, MacOS (10.x) and Linux environment by using Microsoft Visual C++ 2010 and Makefile.

SDK Organization
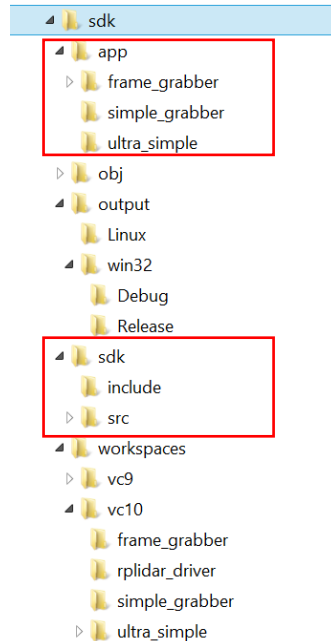
The RPLIDAR standard SDK organized as bellow:

Figure 1-1 The RPLIDAR standard SDK organization

The **workspaces** directory contains VS project files for SDK and related demo applications.

The **sdk** directory contains the external header files(the **include** folder) of RPLIDAR driver application and the internal implementation code of SDK itself(the **src** folder).

The **app** directory contains the related sample code. And SLAMTEC provides the following demo applications in the app directory:

ultra_simple

An ultra-simple command line application demonstrates the simplest way to connect to an RPLIDAR device and continuously fetching the scan data and outputting the data to the console. Users can quickly integrate RPLIDAR to their existing system based on this demo application.

simple_grabber

A command line grab application. Each execution will grab two round of laser data and show as histogram.

frame_grabber

A win32 GUI grab application. When pressing start scan button, it will start scan continuously and show the data in the UI.

For SDK after compilation, there will be two more subfolders in the SDK: **obj** and **output**. The **output** folder contains generated SDK static library (.lib or .a) and demo application executable files (exe or elf). obj folder contains intermediate files generated during compilation.

Build SDK and Demo Applications

If you're developing under Windows, please open VS solution file **sdk_and_demo.sln** under workspaces\vc10 or workspaces\vc9. It contains the SDK project and all demo application projects.
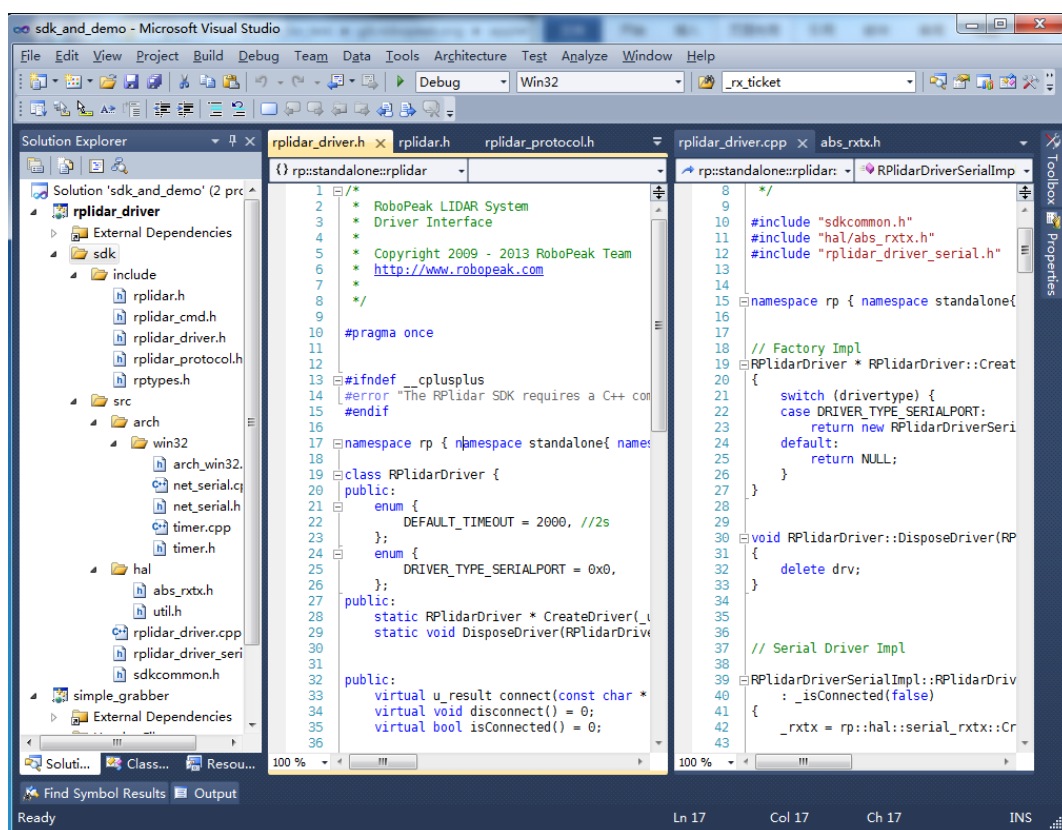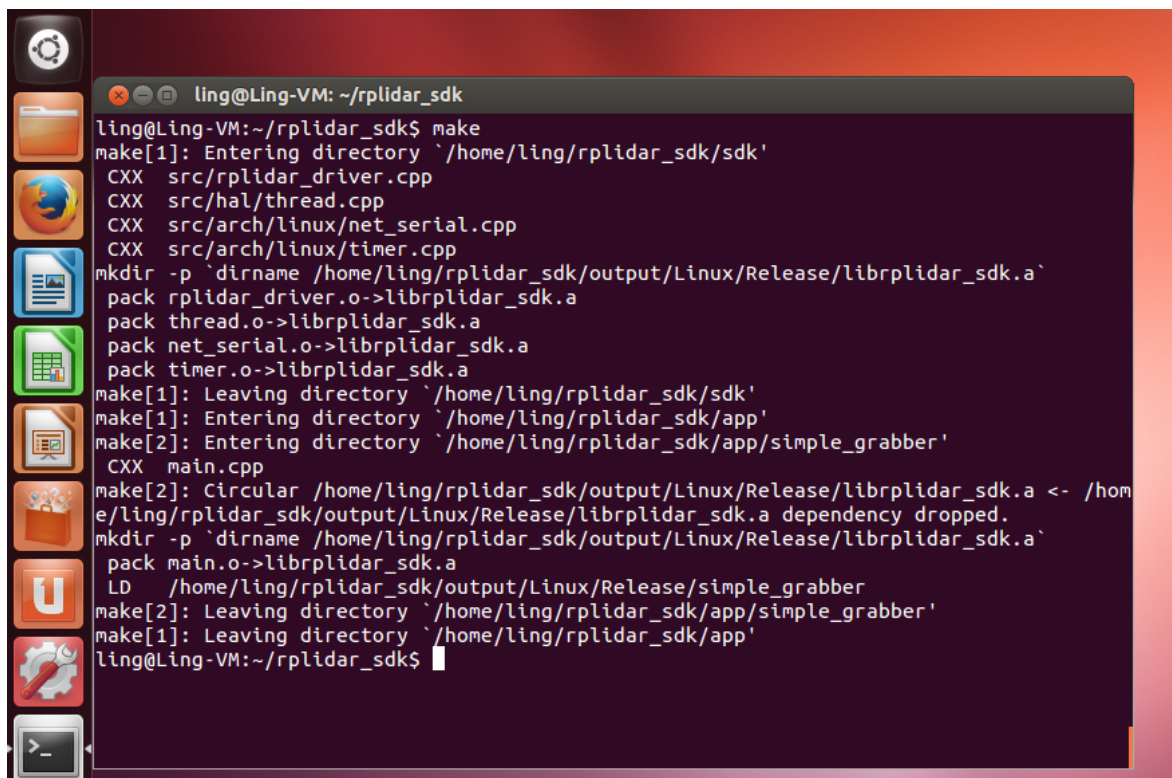

Figure 1-2 The RPLIDAR Standard SDK Solution File

Now, you can develop SDK and all the demo applications in VS environment easily. There are two ways of compiling: Debug and Release. And you can choose according to your requirement. Depends on your build configure, the generated binary can be found under output\win32\Release or output\win32\Debug.
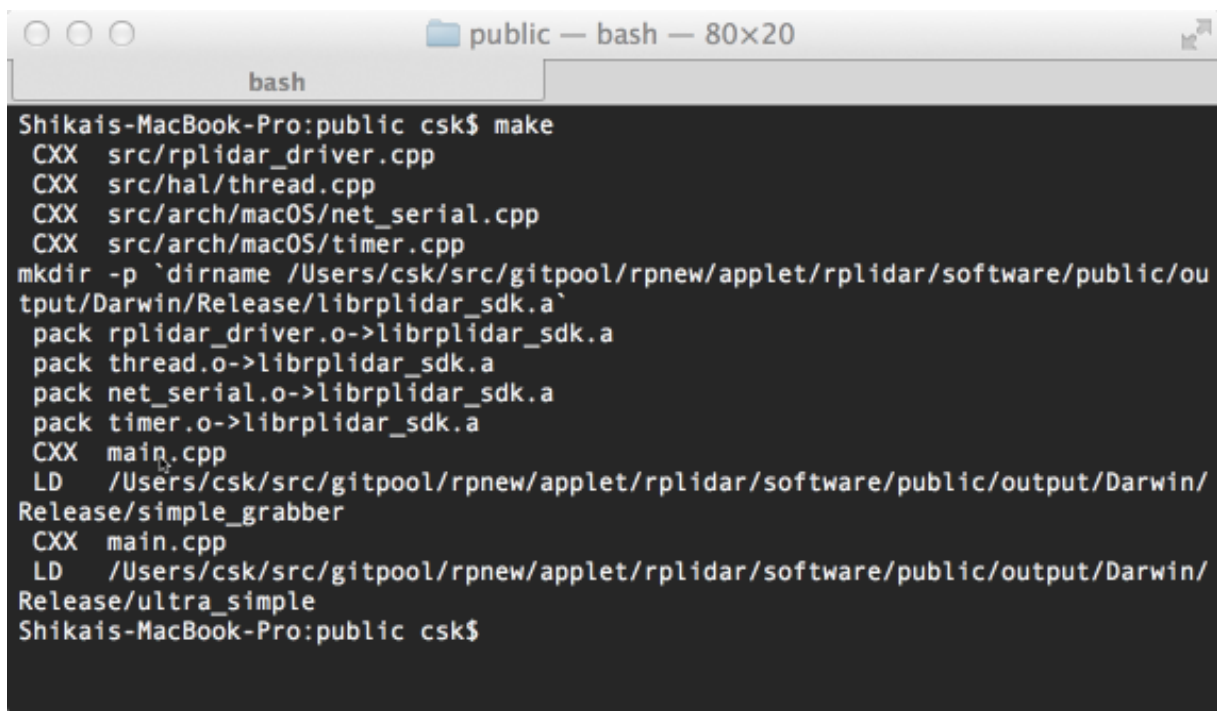
If you're developing under Linux or MacOS, just type "make" under the root of SDK directory to start compiling. It will do Release build by default, and you can also type "make DEBUG=1" to do Debug build. The generated binary can be found under the following folders:

- Linux

    - output\Linux\Release

    - output\Linux\Debug.

- MacOS

    - output\Darwin\Release

    - output\Darwin\Debug.

Figure 1-3 Develop RPLIDAR Standard SDK in Linux

Figure 1-4 Develop RPLIDAR Standard SDK in MacOS

Cross Compile

The SDK build system allows you to generate binaries which run on another platform/system using the cross-compiling feature.

NOTE: this feature only works with Make build system.
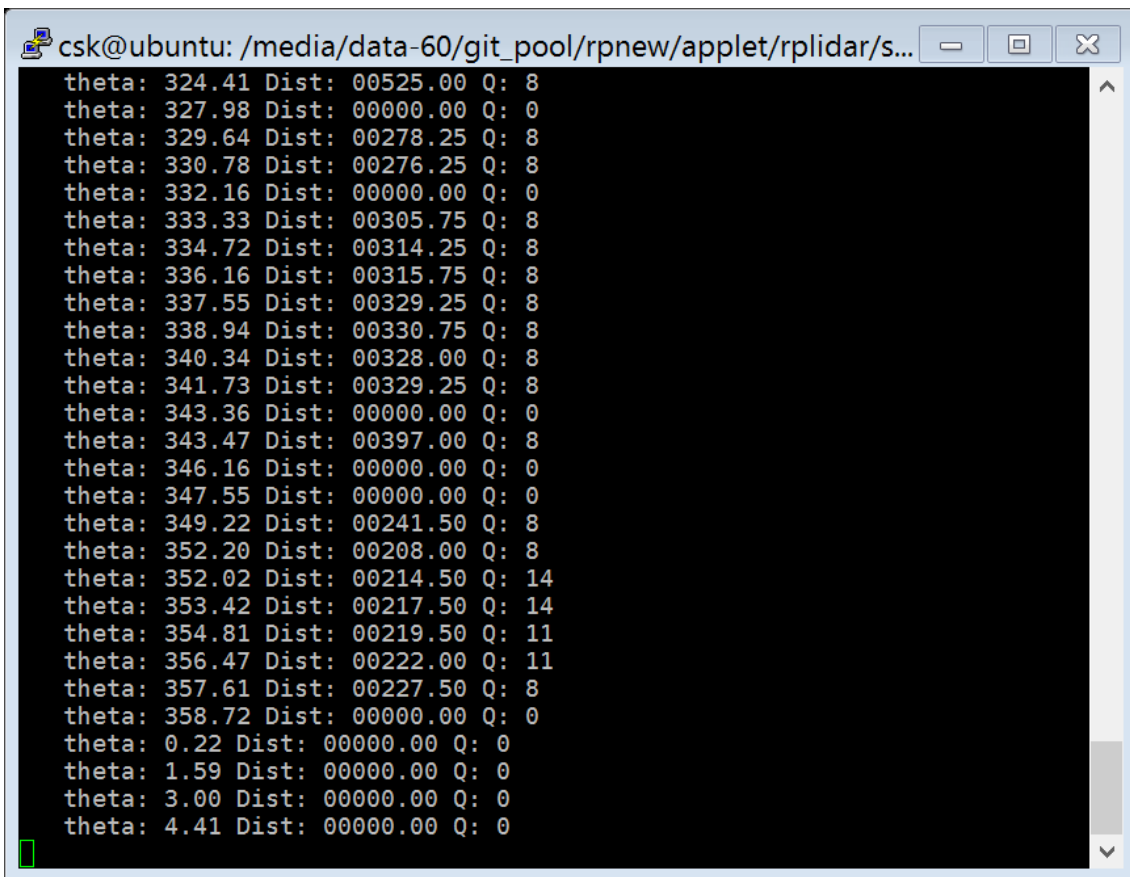
The cross compile process can be triggered by invoking the cross_compile.sh script under the SDK root folder. The common usage is:

CROSS_COMPILE_PREFIX=<COMPILE_PREFIX> ./cross_compile.sh

e.g. `CROSS_COMPILE_PREFIX=arm-linux-gnueabihf ./cross_compile.sh`

ultra_simple

The demo application simply connects to an RPLIDAR device and outputs the scan data to the console:

```
csk@ubuntu: /media/data-60/git_pool/rpnew/applet/rplidar/s...
theta: 324.41 Dist: 00525.00 Q: 8
theta: 327.98 Dist: 00000.00 Q: 0
theta: 329.64 Dist: 00278.25 Q: 8
theta: 330.78 Dist: 00276.25 Q: 8
theta: 332.16 Dist: 00000.00 Q: 0
theta: 333.33 Dist: 00305.75 Q: 8
theta: 334.72 Dist: 00314.25 Q: 8
theta: 336.16 Dist: 00315.75 Q: 8
theta: 337.55 Dist: 00329.25 Q: 8
theta: 338.94 Dist: 00330.75 Q: 8
theta: 340.34 Dist: 00328.00 Q: 8
theta: 341.73 Dist: 00329.25 Q: 8
theta: 343.36 Dist: 00000.00 Q: 0
theta: 343.47 Dist: 00397.00 Q: 8
theta: 346.16 Dist: 00000.00 Q: 0
theta: 347.55 Dist: 00000.00 Q: 0
theta: 349.22 Dist: 00241.50 Q: 8
theta: 352.20 Dist: 00208.00 Q: 8
theta: 352.02 Dist: 00214.50 Q: 14
theta: 353.42 Dist: 00217.50 Q: 14
theta: 354.81 Dist: 00219.50 Q: 11
theta: 356.47 Dist: 00222.00 Q: 11
theta: 357.61 Dist: 00227.50 Q: 8
theta: 358.72 Dist: 00000.00 Q: 0
theta: 0.22 Dist: 00000.00 Q: 0
theta: 1.59 Dist: 00000.00 Q: 0
theta: 3.00 Dist: 00000.00 Q: 0
theta: 4.41 Dist: 00000.00 Q: 0
```

Figure 2-1 ultra_simple Demo Application Data Output

Steps:

1) Connect RPLIDAR to PC by using the provided USB cable. (The chip transforming the USB to serial port is embedded in the RPLIDAR development kit)

2) Start application by using the following command:

o Windows

ultra_simple <com_port>

> **Note:** if the com number is larger than 9, e.g. com11, then you should start application use command like this: **ultra_grabber \\.\com11**

o Linux

ultra_simple <tty device>

e.g. ultra_simple /dev/ttyUSB0.

o Linux

ultra_simple <usb tty device>

e.g. ultra_simple /dev/tty.SLAB_USBtoUART.

simple_grabber

This application demonstrates the process of getting RPLIDAR's serial number, firmware version and healthy status after connecting the PC and RPLIDAR. Then the demo application grabs two round of scan data and shows the range data as histogram in the command line mode. User can print all scan data if needed.

Figure 2-2 simple_grabber Demo Application Data Output

Steps:

1) Connect RPLIDAR to pc using provided USB cable. (USB to serial chip embedded in RPLIDAR development kit)

2) Start application use command: simple_grabber <com number>

o Windows

```
simple_grabber <com_port>
```

Note: if the com number is larger than 9, e.g. com11, then you should start application use command like this: `ultra_grabber \\.\com11`

o Linux

`simple_grabber <tty device>`

e.g. `ultra_simple /dev/ttyUSB0.`

o Linux

`simple_grabber <usb tty device>`

e.g. `ultra_simple /dev/tty.SLAB_USBtoUART.`

frame_grabber

This demo application can show real-time laser scan data in the GUI with 0-360 degree environment range data. Note, this demo application only has win32 version.
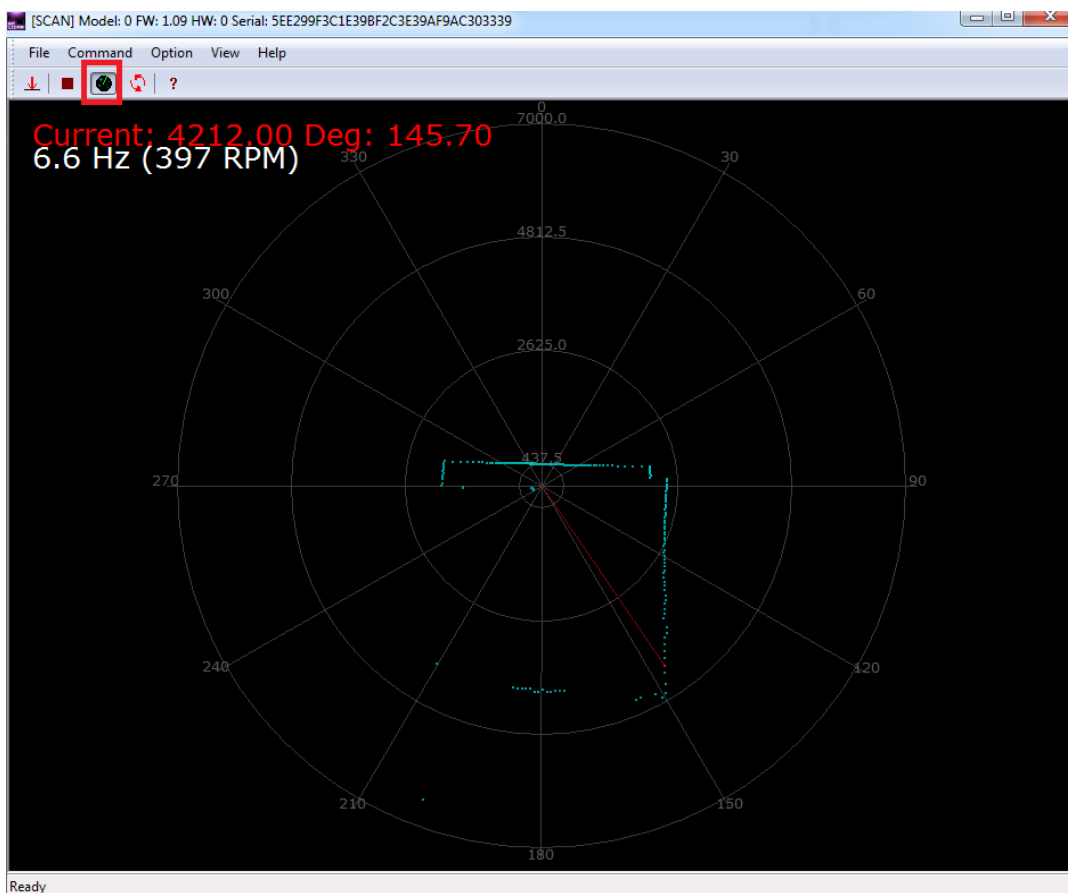

Figure 2-3 frame_grabber Demo Application Data Output

Steps:

1) Connect RPLIDAR to pc using provided USB cable. (USB to serial chip embedded in RPLIDAR development kit)

2) Choose correct com port number through com port selection dialog.

3) Press start scan button (marked in red in the figure above) to start.

Assumption

We strongly recommend developers learn RPLIDAR's communication protocol and working mode before starting development by using the RPLIDAR SDK.

This document assumes developer has related knowledge about C++ development.

SDK usage

The RPLIDAR standard SDK provides static library for developers to integrated the SDK feature into their existing project. If the dynamic library is required, the developers can also set it by easily modifying the project configurations.

When developing via RPLIDAR SDK, developers only need to include SDK's external header files (under sdk\include) into their own source code and link the application with SDK's static library (rplidar_driver.lib or rplidar_driver.a).

For VS developers, they can also include the SDK's VC project into their own project and set the related project dependence. For Linux developers, please refer to the simple_grabber's Makefile for detailed settings.

Runtime consistency

For windows developers: the SDK static library c uses VC10 MD C runtime library. If your project used different C runtime library may lead to compilation failure or unpredictable behavior. Then please change SDK settings accordingly.

SDK Headers

- o rplidar.h

Usually, you only need to include this file to get all functions of RPLIDAR SDK.

- o rplidar_driver.h

This header defines the SDK core drive interface: the RPLidarDriver class. Please refer to demo applications ultra_simple or simple_grabber to understand how to use it.

- o rplidar_protocol.h

This header defines low-level data structures and constants for RPLIDAR protocol.

- o   rplidar_cmd.h

This header defines request/answer data structures and constants for RPLIDAR protocol.

- o   rptypes.h

This header defines platform-independent data structures and constants.

SDK Initialization and Termination

User programs are required to create an RPlidarDriver instance via SDK before communicating with an RPLIDAR device. It can be implemented by using the following static function is used:

RPlidarDriver *RPlidarDriver::CreateDriver (_u32 drivertype)

Each RPlidarDriver instance is bound to only one RPLIDAR device at the same time. But user programs can freely allocate arbitrary number of RPlidarDriver instances and make them communicates with multiple RPLIAR devices concurrently.

Once user programs finish operations, all previously created RPlidarDriver instances should be released explicitly using the following static function in order to free system memory.

RPlidarDriver::DisposeDriver(RPlidarDriver * drv)

Connecting to an RPLIDAR

After creating an RPlidarDriver instance, the user program should invoke the `connect()` function firstly to open the serial port and connect with the RPLIDAR device. All RPLIDAR operations require invoking the `connect()` function first.

u_result RPlidarDriver::connect(const char * port_path, _u32 baudrate, _u32 flag = 0)

The function returns RESULT_OK for success operation.

When invoking this interface, by default, the SDK will call `stopMotor` to stop the rotating of motor. And the SDK will call startMotor to start the rotating of motor before taking measurement.

Once the user program finishes operation, it can call the `disconnect()` function to close the connection and release the serial port device.

Motor Start and Stop Control

The following functions are related to the motor control:

Function Name	Brief Introduction
startMotor()	Request RPLIDAR to start the motor rotating. For RPLIDAR A1, this interface will enable DTR to set the motor start rotating by default. For RPLIDAR A2, this interface will start the motor by using the default duty cycle and configure the rotating speed.
stopMotor()	Request RPLIDAR to stop the motor rotating.

Figure 3-1 RPLIDAR Functions Related to Motor Control

Note: as described in the previous section, the RPLIDAR SDK will call `stopMotor` to stop the rotating of motor by default when invoking the interface `connect`. And before the RPLIDAR start taking distance measurement, the `startMotor` is required to be called to start the motor rotating.

Measurement Scan and Data Acquiring

The following functions are related to the measurement scan operation and help user programs to acquire the measurement data:

Function Name	Brief description
startScan()	Request the RPLIDAR core to start measurement scan operation and send out result data continuously. If the Express Scan mode is supported, and the program invokes the startScan() by using the default parameters
startScanNormal()	Force the RPLIDAR core to start measurement scan operation in Scan mode
startScanExpress()	Force the RPLIDAR core to start measurement scan operation in Express Scan mode. It the RPLIDAR firmware not supports Express Scan mode, the function will fail the execution.
stop()	Request the RPLIDAR core to stop the me as asurement scan operation.
grabScanData()	Grab a complete 360-degrees' scan data sequence.
ascendScanData()	Rank the scan data from grabScanData() as the angle inscreases.

Figure 3-2 RPLIDAR Functions related to Measurement Scan Operation

The `startScan()` function will start a background worker thread to receive the measurement scan data sequence sent from RPLIDAR asynchronously. The received data sequence is stored in the driver's internal cache for the `grabScanData()` function to fetch.

User programs can use the `grabScanData()` function to retrieve the scan data sequence previously received and cached by the driver. This function always returns a latest and complete 360-degrees' measurement scan data sequence. After each `grabScanData()` call, the internal data cache will be cleared to ensure the `grabScanData()` won't get duplicated data.

In case a complete 360-degrees' scan sequence hasn't been available at the time when `grabScanData()` is called, the function will wait until a complete scan data is received by the driver or the given timeout duration is expired. User programs can tune this timeout value to meet different application requirements.

Note: the `startScan()` and `stop()` functions don't control the scanning motor of the RPLIDAR directly. The host system should control the scanning motor to rotate or stop via the PWM pin.

Please refer to the comments in the header files and the implementation of SDK demo applications for details.

Retrieving Other Information of an RPLIDAR

The user program can retrieve other information of an RPLIDAR via the following functions. Please refer to the comments in the header files and the implementation of SDK demo applications for details.

Function Name	Brief description
getHealth()	Get the healthy status of an RPLIDAR
getDeviceInfo()	Retrieve the device information, e.g. serial number, firmware version etc, from and RPLIDAR
getFrequency()	Calculate an RPLIDAR's scanning speed from a complete scan sequence.
checkExpressScanSupported()	Check whether the RPLIDAR supports Express Scan mode
getSampleDuration_uS()	Get the the single measurement duration in standard scan mode and express scan mode respectively. The unit is micro second.

Figure 3-3 RPLIDAR Functions related to Retrieving Other Information Operation

The Operations Related to the Accessory Board

For the accessory board included in some RPLIDAR development kits, since the board supports speed control via PWM, this feature can be implemented by invoking the following functions:

Function Name	Brief description
checkMotorCtrlSupport ()	Check whether the accessory board supports the PWM control. Please refer to the SDK header files notes for details about the usage.
setMotorPWM ()	Send specific PWM duty cycle to the accessory board to control the motor rotating speed of the RPLIDAR. Please refer to the SDK header files notes for details about the usage.

Figure 3-4 RPLIDAR Functions related to the Accessory Board

Date	Description
2013-3-5	Initial version
2014-1-25	Added Linux content and updated related information
2014-3-8	Added description for the ultra_simple demo application
2014-7-25	Added description about developing under MacOS environment
	Added description about the cross compiling
2016-4-12	Added description about the newly added firmware interface of 1.5.1
2016-05-03	Added description about the newly added interface of 1.5.2
	Added new interface startMotor/stopMotor
	Updated connect interface and set stopMotor called by default
2017-05-15	Release 1.0 Version

Image and Table Index