



SLAMTEC RPLIDAR Low Cost 360 Degree Laser Range Scanner Instruction Manual

[Home](#) » [SLAMTEC](#) » SLAMTEC RPLIDAR Low Cost 360 Degree Laser Range Scanner Instruction Manual

Contents [[hide](#)]

- [1 SLAMTEC RPLIDAR Low Cost 360 Degree Laser Range Scanner](#)
- [2 Introduction](#)
- [3 Demo Application](#)
- [4 SDK Usage and Development](#)
- [5 Retrieving Other Information of an RPLIDAR](#)
- [6 Revision History](#)
- [7 Appendix](#)
- [8 Documents / Resources](#)
- [9 Related Posts](#)



SLAMTEC RPLIDAR Low Cost 360 Degree Laser Range Scanner



Introduction

This document introduces the standard open source version of the RPLIDAR SDK V2.0. The SDK uses Microsoft Visual C++ 2010, Microsoft Visual C++ 2019 and Makefile to compile under the environment of Windows, MacOS (10.x) and Linux. RPLIDAR SDK V2.0 is designed in principle and compatible with v1.x version through macro definition.

SDK Organization

The RPLIDAR standard SDK organized as bellow:

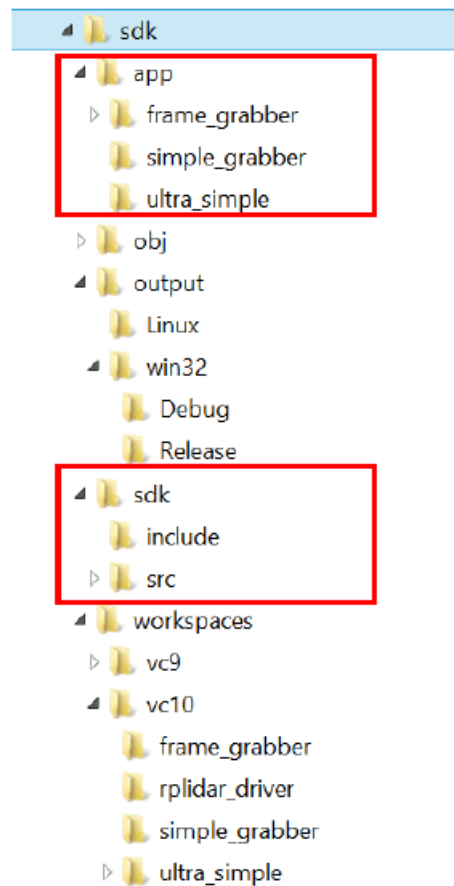


Figure 1-1 The RPLIDAR standard SDK organization

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

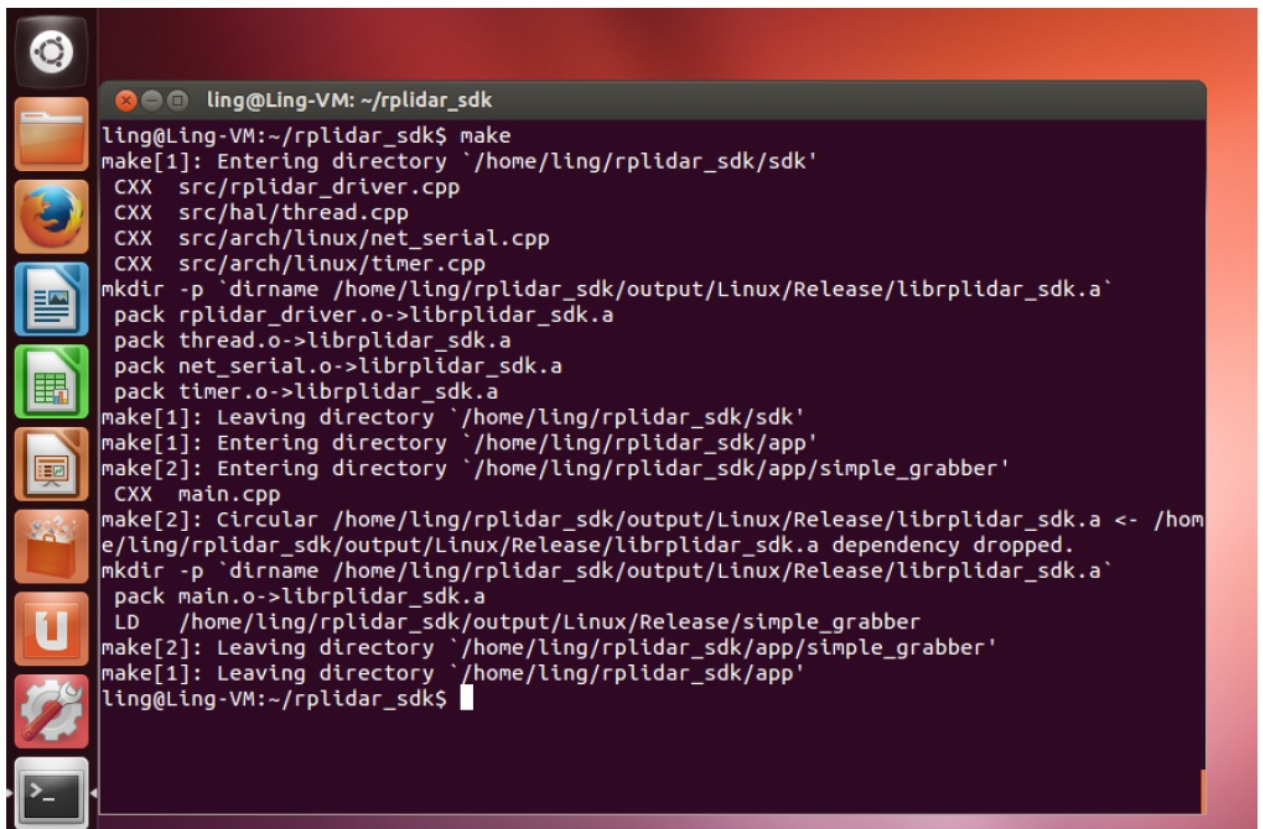
An ultra-simple command line application demonstrates the simplest way to connect to an RPLIDAR device and continuously fetching the scan data and outputting the data to the console. Users can quickly integrate RPLIDAR to their existing system based on this demo application.

simple_grabber

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

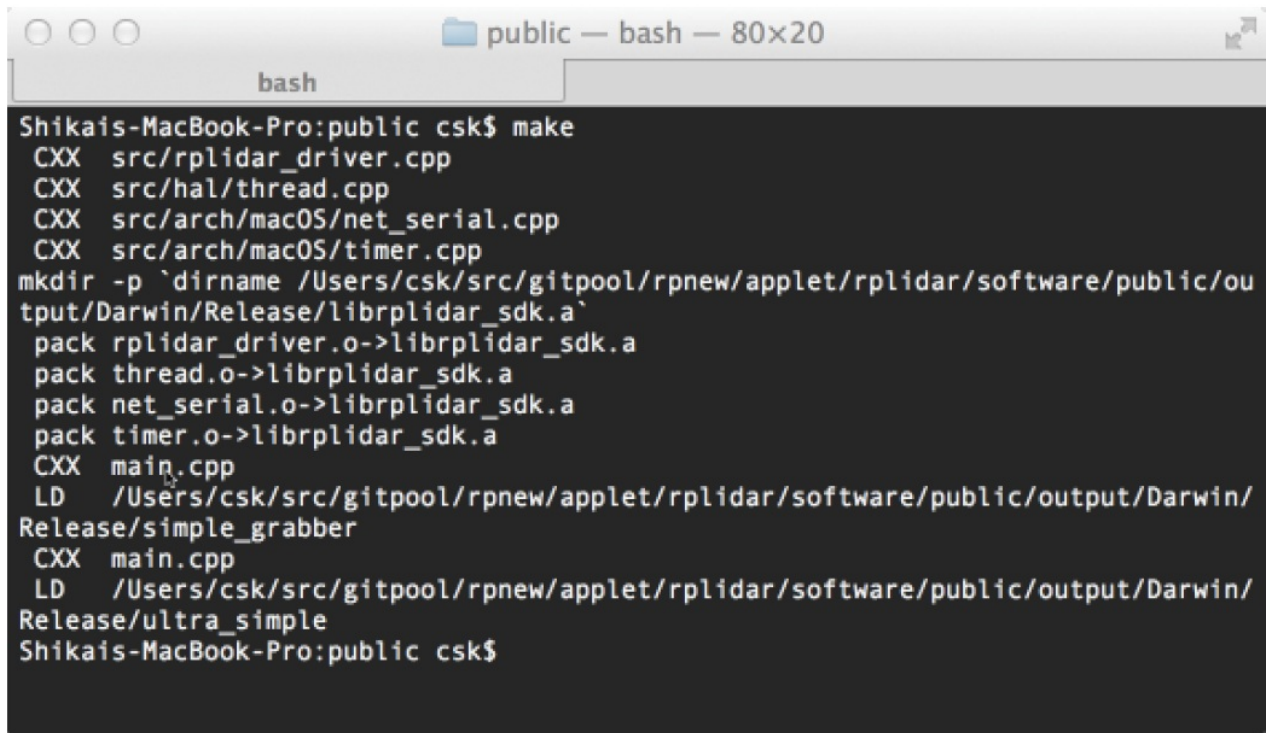
frame_grabber

- Linux
 - `output\Linux\Release`
 - `output\Linux\Debug.`
- MacOS
 - `output\Darwin\Release`
 - `output\Darwin\Debug.`

A terminal window titled 'ling@Ling-VM: ~/rplidar_sdk' showing the execution of a 'make' command. The output shows the compiler (CXX) processing source files in the 'sdk' directory, creating object files and packing them into a library 'librplidar_sdk.a'. It then moves to the 'app' directory, where it compiles 'main.cpp' and links it with the library to create 'simple_grabber'. The process completes successfully.

```
ling@Ling-VM: ~/rplidar_sdk
ling@Ling-VM:~/rplidar_sdk$ make
make[1]: Entering directory `/home/ling/rplidar_sdk/sdk'
CXX src/rplidar_driver.cpp
CXX src/hal/thread.cpp
CXX src/arch/linux/net_serial.cpp
CXX src/arch/linux/timer.cpp
mkdir -p `dirname /home/ling/rplidar_sdk/output/Linux/Release/librplidar_sdk.a`
pack rplidar_driver.o->librplidar_sdk.a
pack thread.o->librplidar_sdk.a
pack net_serial.o->librplidar_sdk.a
pack timer.o->librplidar_sdk.a
make[1]: Leaving directory `/home/ling/rplidar_sdk/sdk'
make[1]: Entering directory `/home/ling/rplidar_sdk/app'
make[2]: Entering directory `/home/ling/rplidar_sdk/app/simple_grabber'
CXX main.cpp
make[2]: Circular /home/ling/rplidar_sdk/output/Linux/Release/librplidar_sdk.a <- /home/ling/rplidar_sdk/output/Linux/Release/librplidar_sdk.a dependency dropped.
mkdir -p `dirname /home/ling/rplidar_sdk/output/Linux/Release/librplidar_sdk.a`
pack main.o->librplidar_sdk.a
LD /home/ling/rplidar_sdk/output/Linux/Release/simple_grabber
make[2]: Leaving directory `/home/ling/rplidar_sdk/app/simple_grabber'
make[1]: Leaving directory `/home/ling/rplidar_sdk/app'
ling@Ling-VM:~/rplidar_sdk$
```

Figure 1-3 Develop RPLIDAR Standard SDK in Linux

A terminal window titled 'public — bash — 80x20' showing the execution of a 'make' command. The output shows the compiler (CXX) processing source files in the 'public/output/Darwin/Release' directory, creating object files and packing them into a library 'librplidar_sdk.a'. It then moves to the 'public/output/Darwin/Release' directory, where it compiles 'main.cpp' and links it with the library to create 'ultra_simple'. The process completes successfully.

```
Shikais-MacBook-Pro:public csk$ make
CXX src/rplidar_driver.cpp
CXX src/hal/thread.cpp
CXX src/arch/macOS/net_serial.cpp
CXX src/arch/macOS/timer.cpp
mkdir -p `dirname /Users/csk/src/gitpool/rpnew/applet/rplidar/software/public/output/Darwin/Release/librplidar_sdk.a`
pack rplidar_driver.o->librplidar_sdk.a
pack thread.o->librplidar_sdk.a
pack net_serial.o->librplidar_sdk.a
pack timer.o->librplidar_sdk.a
CXX main.cpp
LD /Users/csk/src/gitpool/rpnew/applet/rplidar/software/public/output/Darwin/Release/simple_grabber
CXX main.cpp
LD /Users/csk/src/gitpool/rpnew/applet/rplidar/software/public/output/Darwin/Release/ultra_simple
Shikais-MacBook-Pro:public csk$
```

Figure 1-4 Develop RPLIDAR Standard SDK in MacOS

Cross Compile

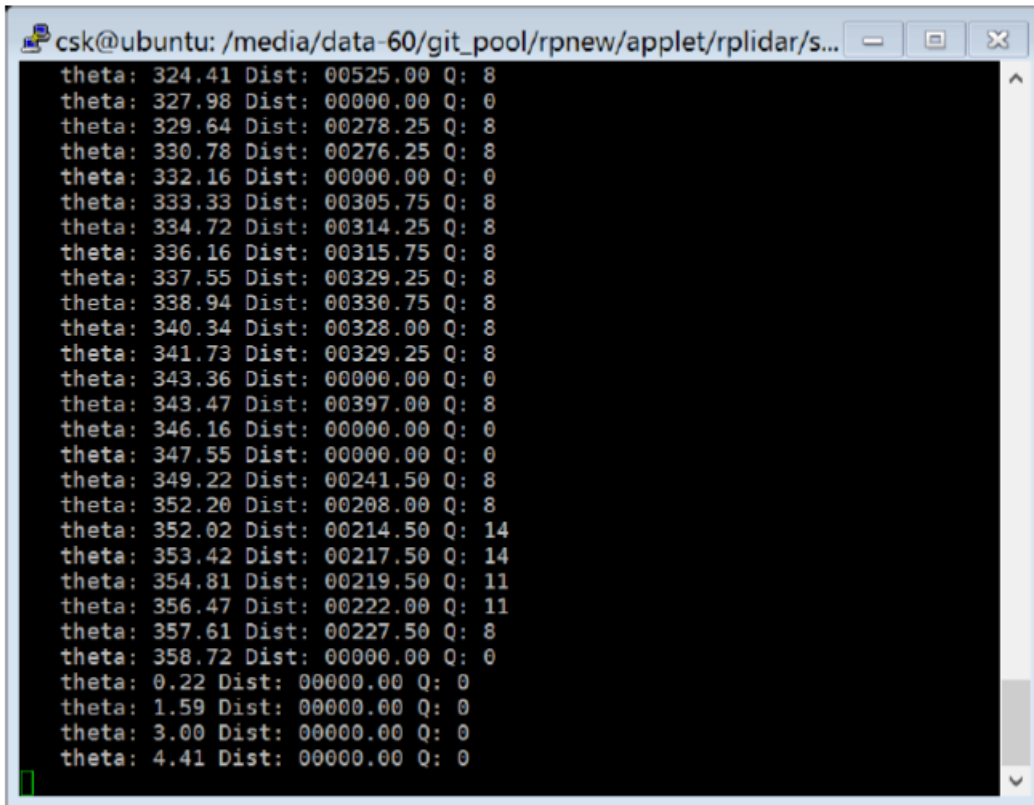
The SDK build system allows you to generate binaries which run on another platform/system using the cross-compiling feature.

NOTE: this feature only works on where using Makefile. 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

Demo Application

ultra_simple

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

A screenshot of a terminal window titled 'csk@ubuntu: /media/data-60/git_pool/rpnew/applet/rplidar/s...'. The terminal displays a series of RPLIDAR scan data points. Each line represents a scan at a specific theta angle, showing the distance (Dist) and quality (Q) of the measurement. The data points are as follows:

```
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. According to the lidar model and configuration, the following three connection modes are available:
 - Connect RPLIDAR to PC by using the provided USB cable and PIC. (The chip transforming the USB to serial port is embedded in the RPLIDAR development kit)
 - Connect RPLIDAR to PC by using the provided Ethernet module. (using TCP/IP protocol)
 - Connect RPLIDAR to PC by using the body network cable. (using UDP protocol)
2. According to the connection mode, the following command is used to start the sample program:
 - **ultra_simple** -channel -serial <com port> [baudrate] <com port>: the serial port # Windows: \\.\com3
 - **MacOS**: /dev/tty.SLAB_USBtoUART
 - **Linux**: /dev/ttyUSB0
 - **[baudrate]**: the communication baudrate
 - **A2**: 115200
 - **A3/S1**: 256000
 - S2: 1000000
 - **ultra_simple** -channel -tcp <ip> [port] <ip>: the IP address of the radar or Ethernet module [port]: Port number 20108
 - **ultra_simple** -channel -udp <ip> [port] <ip>: the IP address of the radar [port] : Port number 8089

simple_grabber

This application demonstrates the process of getting RPLIDAR's serial number, firmware version and healthy

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.

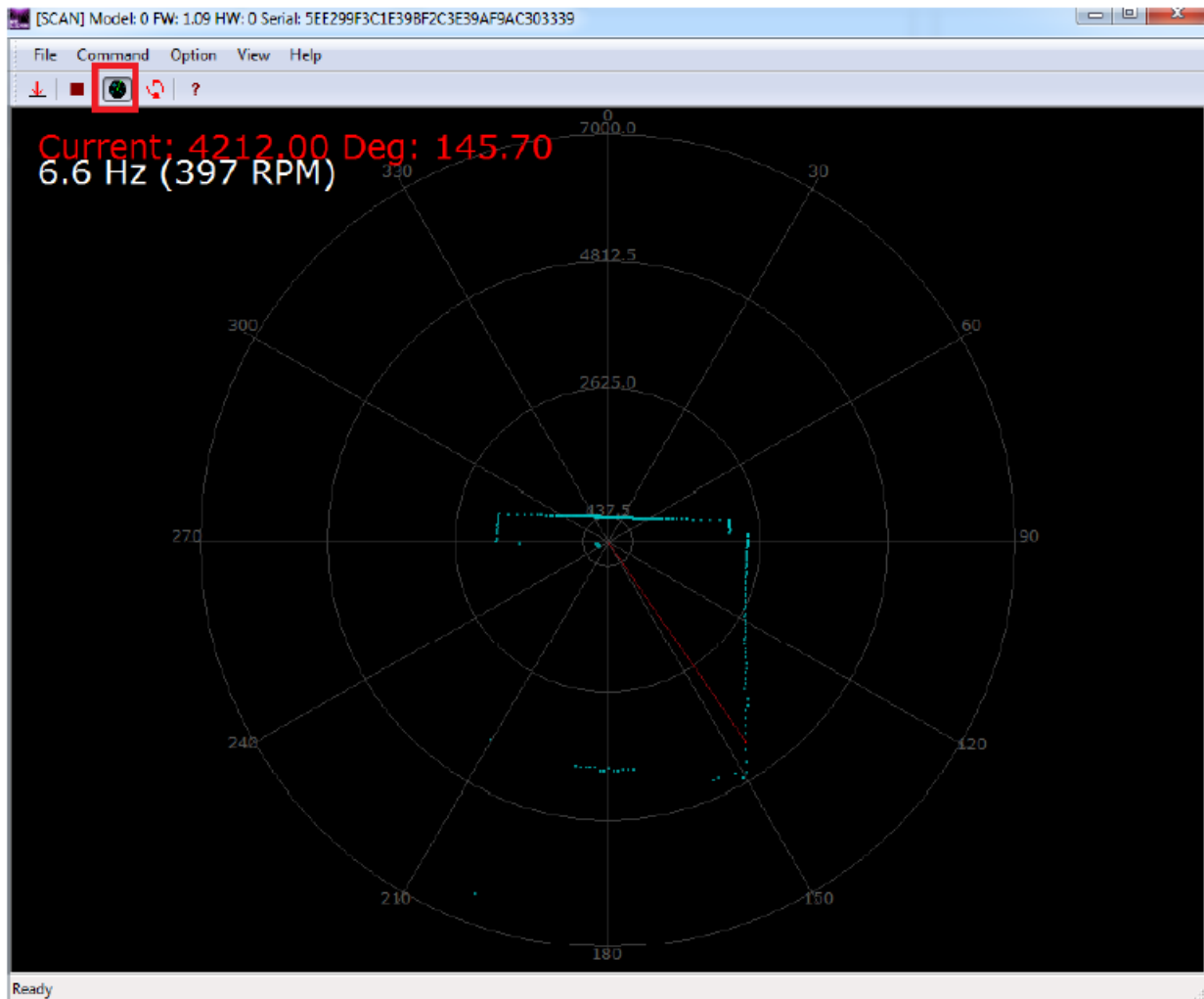


Figure 2-3 frame_grabber Demo Application Data Output

Steps:

1. According to the lidar model and configuration, the following three connection modes are available:
 - Connect RPLIDAR to PC by using the provided USB cable and PIC. (The chip transforming the USB to serial port is embedded in the RPLIDAR development kit)
 - Connect RPLIDAR to PC by using the provided Ethernet module. (using TCP/IP protocol)
 - Connect RPLIDAR to PC by using the body network cable. (using UDP protocol)
2. According to the connection mode, the following command is used to start the sample program:
 - Connect RPLIDAR to PC by using the provided USB cable and PIC. (The chip transforming the USB to serial port is embedded in the RPLIDAR development kit) Select the correct serial port number in the dialog box.

SDK Usage and Development

Click the Start scan button (shown in the red box) to start the scan.

- Always enter the correct IP address and port number in the dialog box Select TCP and click the OK button. Click the Start scan button (shown in the red box) to start the scan.
- Always enter the correct IP address and port number in the dialog box. Select UDP and click the OK

button. Click the Start scan button (shown in the red box) to start the scan.

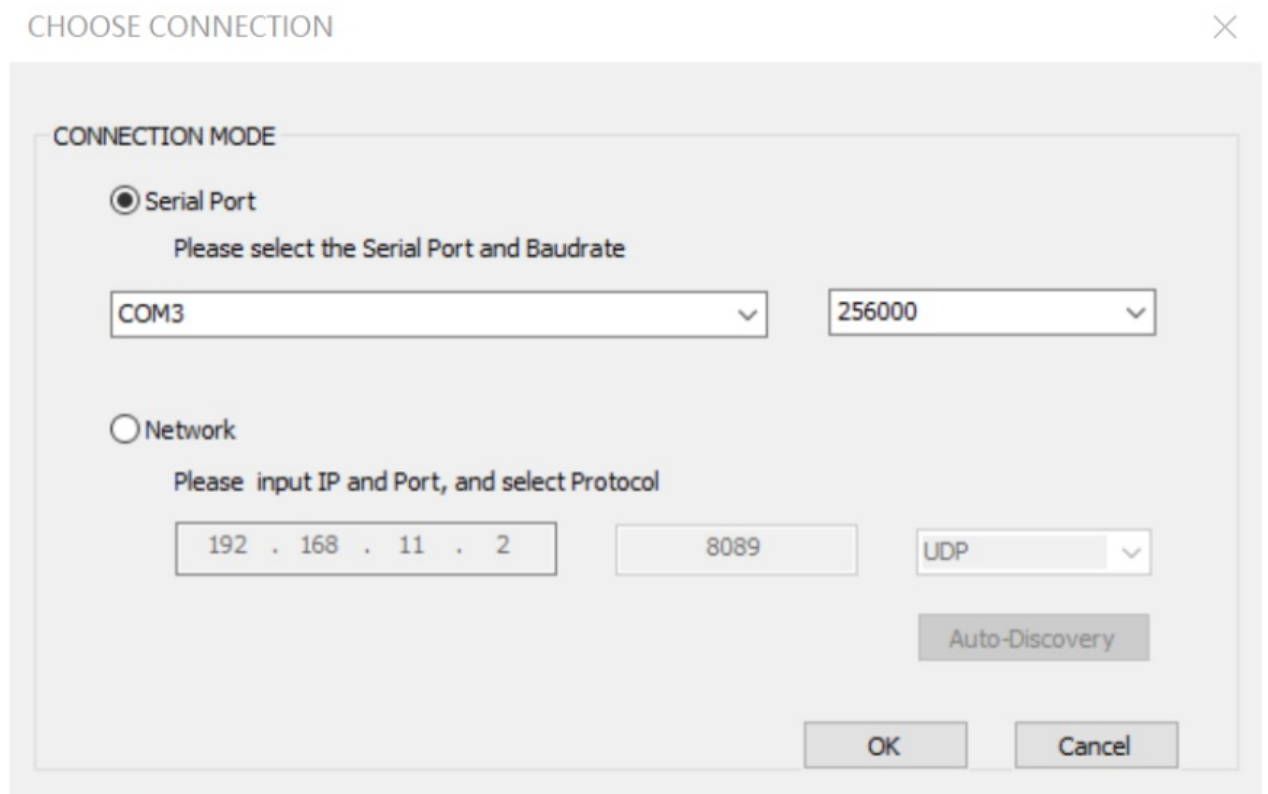


Figure 2-4 frame_grabber Demo Application Dialog

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. PS: To be compatible with earlier SDK, static libraries are still named rplidar_drvier. Please check rplidar_driver.h, sl_lidar_driver.h for details.

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

- sl_lidar.h

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

- sl_lidar_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.

- sl_lidar_protocol.h

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

- `sl_rplidar_cmd.h`

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

- `sl_rptypes.h`

This header defines platform-independent data structures and constants.

SDK Initialization and Termination

The communication between user program and an RPLIDAR device can be made by doing below actions (Details can be checked from sample program, for example, ultra simple):

1. 1) Create a LIDAR driver instance.

```
ILidarDriver * drv = *createLidarDriver();
```

Create a corresponding communication channel based on different connections of Lidar.

- Connect RPLIDAR to PC by using the provided USB cable and PIC.
- param device Serial port device
- e.g. on Windows, it may be com3 or \\.\com10
- on Unix-Like OS, it may be /dev/ttyS1, /dev/ttyUSB2, etc
- param baudrate Baudrate
- Please refer to the datasheet for the baudrate (maybe 115200 or 256000)
- `std::string device = "com3"; int baudrate = 115200; Result<IChannel*> channel = createSerialPortChannel("/dev/ttyUSB0", 115200);`
- Connect RPLIDAR to PC by using the provided Ethernet module. (using TCP/IP protocol) `Result<IChannel*> channel = createTcpChannel("192.168.11.2", 20108);`
- Connect RPLIDAR to PC by using the body network cable. (using UDP protocol) `Result<IChannel*> channel = createTcpChannel("192.168.11.2",8089);`

An ILidarDriver instance can communicate with only one RPLIDAR in the system at the same time. However a user program can create any several ILidarDriver instances to communicate with any several RPLIDAR devices. Once user programs finish operations, all previously created lidar driver instances should be released explicitly using the following static function in order to free system memory.

- `delete drv;`
- `delete channel;`

Connecting to an RPLIDAR

After creating an ILidarDriver 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.

- `auto res = (*lidar)->connect(*channel);`
- The function returns `RESULT_OK` for success operation.

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

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
<code>startScan()</code>	Request the RPLIDAR core to start measurement scan operation and send out result data continuously SDK will use Express Scan mode automatically, when it is supported and the program invokes the <code>startScan()</code> by using the default parameters
<code>startScanExpress()</code>	Force the RPLIDAR core to start measurement scan operation in Express Scan mode. If the RPLIDAR firmware does not support Express Scan mode, the function will fail the execution.
<code>stop()</code>	Request the RPLIDAR core to stop the measurement scan operation.
<code>grabScanDataHg()</code>	Grab a complete 360-degrees' scan data sequence.
<code>ascendScanData()</code>	Rank the scan data from <code>grabScanData()</code> as the angle increases.

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 `grabScanDataHg()` function to fetch. User programs can use the `grabScanDataHg()` 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 `grabScanDataHg()` call, the internal data cache will be cleared to ensure the `grabScanDataHg()` won't get duplicated data. In case a complete 360-degrees' scan sequence hasn't been available at the time when `grabScanDataHg()` 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. 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
<code>getHealth()</code>	Get the healthy status of an RPLIDAR
<code>getDeviceInfo()</code>	Retrieve the device information, e.g. serial number, firmware version etc, from an RPLIDAR
<code>getFrequency()</code>	Calculate an RPLIDAR's scanning speed from a complete scan sequence.

Revision History

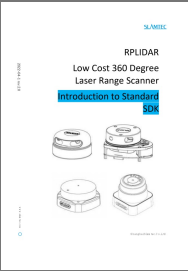
Date	Description
2013-3-5	Initial version
2014-1-25	Added Linux content and updated related information
2014-3-8	Added description for the ultra_simple demo application
2014-7-25	Added description about developing under MacOS environment Added description about the cross compiling
2016-4-12	Added description about the newly added firmware interface of 1.5.1
2016-5-3	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
2017-5-15	Release 1.0 Version
2022-4-1	Release 2.0 Version

Appendix

Image and Table Index

• FIGURE 1-1 THE RPLIDAR STANDARD SDK ORGANIZATION	3
• FIGURE 1-2 THE RPLIDAR STANDARD SDK SOLUTION FILE	4
• FIGURE 1-3 DEVELOP RPLIDAR STANDARD SDK IN LINUX	5
• FIGURE 1-4 DEVELOP RPLIDAR STANDARD SDK IN MACOS	6
• FIGURE 2-1 ULTRA_SIMPLE DEMO APPLICATION DATA OUTPUT	7
• FIGURE 2-2 SIMPLE_GRABBER DEMO APPLICATION DATA OUTPUT	9
• FIGURE 2-3 FRAME_GRABBER DEMO APPLICATION DATA OUTPUT	11
• FIGURE 2-4 FRAME_GRABBER DEMO APPLICATION DIALOG	12
• FIGURE 3-1RPLIDAR FUNCTIONS RELATED TO MEASUREMENT SCAN OPERATION.....	16
• FIGURE 3-2RPLIDAR FUNCTIONS RELATED TO RETRIEVING OTHER INFORMATION OPERATION	17
◦ Copyright (c) 2009-2013 RoboPeak Team	
◦ Copyright (c) 2013-2022 Shanghai Slamtec Co., Ltd.	

Documents / Resources

	<p>SLAMTEC RPLIDAR Low Cost 360 Degree Laser Range Scanner [pdf] Instruction Manual</p> <p>RPLIDAR Low Cost 360 Degree Laser Range Scanner, RPLIDAR, Low Cost 360 Degree Laser Range Scanner, Laser Range Scanner, Range Scanner, Scanner</p>
--	--