**RPLIDAR Interface with NVIDIA Jetson TK1 Development Board**

Amit Vasant Pachore, Bhupendra Naphade, Urvashi Agrawal, Anush Shankar

*Computer Engineering Department, Charles W Davidson College of Engineering*

*San Jose State University, San Jose, CA 95192*

*E-mail:* [*amitvasant.pachore@sjsu.edu*](mailto:amitvasant.pachore@sjsu.edu)*,* [*bhupendra.naphade@sjsu.edu*](mailto:bhupendra.naphade@sjsu.edu)*,* [*Urvashi.agrawal@sjsu.edu*](mailto:Urvashi.agrawal@sjsu.edu)*,* [*anush.shankar@sjsu.com*](mailto:anush.shankar@sjsu.com)

**ABSTRACT**

*This paper describes the interface and working of RPLIDAR with NVIDIA Jetson TK1 development board. The integral components of this paper include driver installation details, Jetson board setup and test results of LIDAR interface. The paper also focuses on a few issues with board bring up and their resolution.*

**1. INTRODUCTION**

LIDAR also known as Light Imaging, Detection and Ranging is the technique of measuring distance to an object by illuminating the object with laser light in Infrared spectrum. Most LIDARs send out laser light and based on the time taken for the light to return to the source, distance measurements are done. Distance is measured with respect to the time delay between transmission of pulse and the detection of the reflected light signal. Modern versions of LIDARs make use of Image processing capabilities to obtain better detection range and reduce cost to considerable amount. LIDARs are the base to 3D environment sensing and have found applications in archeology, agriculture, geology, biology, autonomous vehicles, soil science, atmosphere, meteorology, military and law enforcement.

Accurate environment sensing using LIDAR sensing requires precise measurement of reflected light, processing reflected light at good clock rates to prevent data loss. Currently the most trending nad powerful technology involves the use of GPU, which is capable of efficiently handling intensive computations. GPUs have the ability to parallelize tasks in an effective manner, to allow more time for serial tasks and improving system performance. NVIDIA Jetson TK1 is the latest GPU with NVIDIA Kepler comprising of 192 CUDA cores. The use of a GPU helps achieve better response time for range detection due to powerful computation offered by the architecture. This paper discusses the interface of RPLIDAR with omnidirectional laser scanner, high speed laser triangulation and 360 degree scanning with NVIDIA Jetson TK1 development board.

**2. Installation**

Prior to interfacing RPLIDAR to the NVIDIA Jetson board, interface was established with a personal computer and the tests were performed. This section explains the process involved in the installation and interface of the LIDAR

1. **Installation on Computer**

This is very simple process as we need not compile and install specific device drivers. USB to serial drivers are pre- installed on the Ubuntu for PC. Source code compilation is also a part of the installation before using the RPLIDAR. Below are the steps for source code installation.

1. Download the source code from RPLIDAR website [1].
2. Extract downloaded compressed file to any location.
3. Change your current directory to *Extracted Path/rplidar\_sdk\_v\_x.x.x/sdk/*
4. Compile the SDK and samples programs by running the make command

Before we execute some samples programs, we need to alow access to application programs to the device drives [2].

1. Go to the file directory：*/etc/udev/rules.d/*, and see whether it has the file named *50-usb-serial.rules*
2. If file is not present, create the file with below steps

$ cd /etc/udev/rules.d/

$ sudo touch 50-usb-serial.rules

$ sudo gedit 50-usb-serial.rules

1. Allow USB to serial device driver access to user application programs with this rule: **KERNEL=="ttyUSB0", GROUP="user", MODE="0666"** (Here, the word "user" is your system user name)
2. Run this command:

$ sudo /etc/init.d/udev restart

With above steps, system set up is completed and ready for the application programs to be executed. Change your current directory to …./output/Linux/Release and run the sample programs with below command[2]

$ ./sample\_program\_name /dev/ttyUSB0

1. **Installation on NVIDIA Jetson TK1**

Installation process on Jetson development platform is complex as compared to the computer [3][4][5].

1. Download kernel source from the Jetson download center on Jetson platform
2. Configuring the kernel
3. Copy over the Jetson's existing kernel configuration to the newly-extracted kernel source configuration:

zcat /proc/config.gz > ~/kernel/.config

1. Next build & launch the menuconfig tool to configure the kernel options.

Menuconfig requires ncurses to be installed, hence the apt-get command first.

$ sudo apt-get install ncurses-bin libncurses5-dev

$ make menuconfig

1. Navigate to Device Drivers -> USB Support -> USB Serial Converter Support
2. Choose 'M'odule for “CP210X …… “
3. Get the kernel local version with

$ cat .config | grep LOCALVERSION

1. In make menuconfig, under 'General setup' -> 'Local version'. Set it to '-gdacac96'(local version got from above step) (including the dash).
2. Building modules

$ make prepare

$ make modules\_prepare

$ make M=drivers/usb/serial/

1. Installing the FTDI module

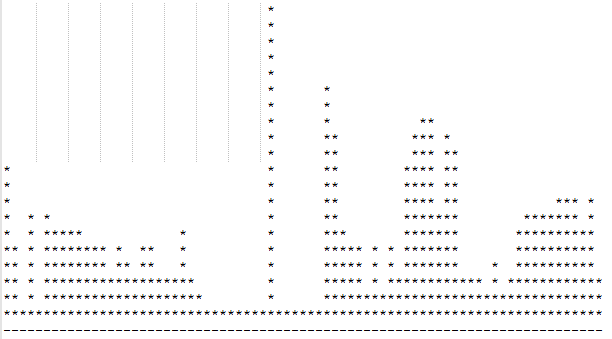
$ sudo cp drivers/usb/serial/cp210x..\*.ko /lib/modules/$(uname -r)/kernel

$ sudo depmod -a

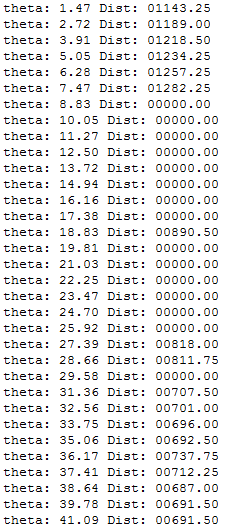
Follow similar steps as explained previously to run sample programs.

**3. Testing and Output**

Once setup was compete, RPLIDAR was tested using the manufacturer provided sample programs. We used the simple\_grabber program to test the output. 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 the scan data and shows the range data as histogram in the command line mode. User can print all scan data if needed [4].



**Figure 1. Output Data Histogram**



**Figure 2. Obstacle Distance at an Angle**

**4. Conclusion**

In conclusion, we successfully interfaced RPLIDAR with a computer and the Jetson development platform. With this experiment, we learned device driver installation on Linux platform. We have come across drawbacks in the current LIDAR mode and problems that should be addressed while designing advanced LIDAR device. From the test results we can conclude that the current available LIDAR is limited for short distance sensing. This shortcoming urges to a design of LIDAR for enhanced distance sensing.

**5. References**

1. "RPLIDAR". *Slamtec.com*. Web. 2 Sept. 2016.
2. "RPLIDAR SDK Manual". *Slamtec.com*. Web. 2 Sept. 2016.
3. "Jetson Tutorials - Program An Arduino". *elinux.org*. N.p., 2015. Web. 2 Sept. 2016.
4. "RPLIDAR Driver Installation Problem On Ubuntu". *ROS.org*. N.p., 2016. Web. 2 Sept. 2016.
5. "Jetson TK1 (R21.1) And Siliconlab Cp210x Driver". *Devtalk.nvidia.com*. Web. 2 Sept. 2016.

**6. Source Code**

/\*

\* RPLIDAR

\* Simple Data Grabber Demo App

\*

\* Copyright (c) 2009 - 2014 RoboPeak Team

\* http://www.robopeak.com

\* Copyright (c) 2014 - 2016 Shanghai Slamtec Co., Ltd.

\* http://www.slamtec.com

\*

\*/

/\*

\* This program is free software: you can redistribute it and/or modify

\* it under the terms of the GNU General Public License as published by

\* the Free Software Foundation, either version 3 of the License, or

\* (at your option) any later version.

\*

\* This program is distributed in the hope that it will be useful,

\* but WITHOUT ANY WARRANTY; without even the implied warranty of

\* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

\* GNU General Public License for more details.

\*

\* You should have received a copy of the GNU General Public License

\* along with this program. If not, see <http://www.gnu.org/licenses/>.

\*

\*/

#include <stdio.h>

#include <stdlib.h>

#include "rplidar.h" //RPLIDAR standard sdk, all-in-one header

#ifndef \_countof

#define \_countof(\_Array) (int)(sizeof(\_Array) / sizeof(\_Array[0]))

#endif

#ifdef \_WIN32

#include <Windows.h>

#define delay(x) ::Sleep(x)

#else

#include <unistd.h>

static inline void delay(\_word\_size\_t ms){

while (ms>=1000){

usleep(1000\*1000);

ms-=1000;

};

if (ms!=0)

usleep(ms\*1000);

}

#endif

using namespace rp::standalone::rplidar;

void print\_usage(int argc, const char \* argv[])

{

printf("Simple LIDAR data grabber for RPLIDAR.\n"

"Usage:\n"

"%s <com port> [baudrate]\n"

"The default baudrate is 115200. Please refer to the datasheet for details.\n"

, argv[0]);

}

void plot\_histogram(rplidar\_response\_measurement\_node\_t \* nodes, size\_t count)

{

const int BARCOUNT = 75;

const int MAXBARHEIGHT = 20;

const float ANGLESCALE = 360.0f/BARCOUNT;

float histogram[BARCOUNT];

for (int pos = 0; pos < \_countof(histogram); ++pos) {

histogram[pos] = 0.0f;

}

float max\_val = 0;

for (int pos =0 ; pos < (int)count; ++pos) {

int int\_deg = (int)((nodes[pos].angle\_q6\_checkbit >> RPLIDAR\_RESP\_MEASUREMENT\_ANGLE\_SHIFT)/64.0f/ANGLESCALE);

if (int\_deg >= BARCOUNT) int\_deg = 0;

float cachedd = histogram[int\_deg];

if (cachedd == 0.0f ) {

cachedd = nodes[pos].distance\_q2/4.0f;

} else {

cachedd = (nodes[pos].distance\_q2/4.0f + cachedd)/2.0f;

}

if (cachedd > max\_val) max\_val = cachedd;

histogram[int\_deg] = cachedd;

}

for (int height = 0; height < MAXBARHEIGHT; ++height) {

float threshold\_h = (MAXBARHEIGHT - height - 1) \* (max\_val/MAXBARHEIGHT);

for (int xpos = 0; xpos < BARCOUNT; ++xpos) {

if (histogram[xpos] >= threshold\_h) {

putc('\*', stdout);

}else {

putc(' ', stdout);

}

}

printf("\n");

}

for (int xpos = 0; xpos < BARCOUNT; ++xpos) {

putc('-', stdout);

}

printf("\n");

}

u\_result capture\_and\_display(RPlidarDriver \* drv)

{

u\_result ans;

rplidar\_response\_measurement\_node\_t nodes[360\*2];

size\_t count = \_countof(nodes);

printf("waiting for data...\n");

// fetech extactly one 0-360 degrees' scan

ans = drv->grabScanData(nodes, count);

if (IS\_OK(ans) || ans == RESULT\_OPERATION\_TIMEOUT) {

drv->ascendScanData(nodes, count);

plot\_histogram(nodes, count);

printf("Do you want to see all the data? (y/n) ");

int key = getchar();

if (key == 'Y' || key == 'y') {

for (int pos = 0; pos < (int)count ; ++pos) {

printf("%s theta: %03.2f Dist: %08.2f \n",

(nodes[pos].sync\_quality & RPLIDAR\_RESP\_MEASUREMENT\_SYNCBIT) ?"S ":" ",

(nodes[pos].angle\_q6\_checkbit >> RPLIDAR\_RESP\_MEASUREMENT\_ANGLE\_SHIFT)/64.0f,

nodes[pos].distance\_q2/4.0f);

}

}

} else {

printf("error code: %x\n", ans);

}

return ans;

}

int main(int argc, const char \* argv[]) {

const char \* opt\_com\_path = NULL;

\_u32 opt\_com\_baudrate = 115200;

u\_result op\_result;

if (argc < 2) {

print\_usage(argc, argv);

return -1;

}

opt\_com\_path = argv[1];

if (argc>2) opt\_com\_baudrate = strtoul(argv[2], NULL, 10);

// create the driver instance

RPlidarDriver \* drv = RPlidarDriver::CreateDriver(RPlidarDriver::DRIVER\_TYPE\_SERIALPORT);

if (!drv) {

fprintf(stderr, "insufficent memory, exit\n");

exit(-2);

}

rplidar\_response\_device\_health\_t healthinfo;

rplidar\_response\_device\_info\_t devinfo;

do {

// try to connect

if (IS\_FAIL(drv->connect(opt\_com\_path, opt\_com\_baudrate))) {

fprintf(stderr, "Error, cannot bind to the specified serial port %s.\n"

, opt\_com\_path);

break;

}

// retrieving the device info

////////////////////////////////////////

op\_result = drv->getDeviceInfo(devinfo);

if (IS\_FAIL(op\_result)) {

if (op\_result == RESULT\_OPERATION\_TIMEOUT) {

// you can check the detailed failure reason

fprintf(stderr, "Error, operation time out.\n");

} else {

fprintf(stderr, "Error, unexpected error, code: %x\n", op\_result);

// other unexpected result

}

break;

}

// print out the device serial number, firmware and hardware version number..

printf("RPLIDAR S/N: ");

for (int pos = 0; pos < 16 ;++pos) {

printf("%02X", devinfo.serialnum[pos]);

}

printf("\n"

"Firmware Ver: %d.%02d\n"

"Hardware Rev: %d\n"

, devinfo.firmware\_version>>8

, devinfo.firmware\_version & 0xFF

, (int)devinfo.hardware\_version);

// check the device health

////////////////////////////////////////

op\_result = drv->getHealth(healthinfo);

if (IS\_OK(op\_result)) { // the macro IS\_OK is the preperred way to judge whether the operation is succeed.

printf("RPLidar health status : ");

switch (healthinfo.status) {

case RPLIDAR\_STATUS\_OK:

printf("OK.");

break;

case RPLIDAR\_STATUS\_WARNING:

printf("Warning.");

break;

case RPLIDAR\_STATUS\_ERROR:

printf("Error.");

break;

}

printf(" (errorcode: %d)\n", healthinfo.error\_code);

} else {

fprintf(stderr, "Error, cannot retrieve the lidar health code: %x\n", op\_result);

break;

}

if (healthinfo.status == RPLIDAR\_STATUS\_ERROR) {

fprintf(stderr, "Error, rplidar internal error detected. Please reboot the device to retry.\n");

// enable the following code if you want rplidar to be reboot by software

// drv->reset();

break;

}

drv->startMotor();

// take only one 360 deg scan and display the result as a histogram

////////////////////////////////////////////////////////////////////////////////

if (IS\_FAIL(drv->startScan( /\* true \*/ ))) // you can force rplidar to perform scan operation regardless whether the motor is rotating

{

fprintf(stderr, "Error, cannot start the scan operation.\n");

break;

}

if (IS\_FAIL(capture\_and\_display(drv))) {

fprintf(stderr, "Error, cannot grab scan data.\n");

break;

}

} while(0);

drv->stop();

drv->stopMotor();

RPlidarDriver::DisposeDriver(drv);

return 0;

}