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, bhupendra.naphade@sjsu.edu, Urvashi.agrawal@sjsu.edu, 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

a. 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
```

3. 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)

4. 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
```

b. 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
   a. Copy over the Jetson's existing kernel configuration to the newly-extracted kernel source configuration:
   ```
   zcat /proc/config.gz >
   ~/kernel/.config
   ```

3. 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
   ```

4. Navigate to Device Drivers -> USB Support -> USB Serial Converter Support

5. Choose 'M'odule for "CP210X …… "

6. Get the kernel local version with
   ```
   $ cat .config | grep
   LOCALVERSION
   ```

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

8. Building modules
   ```
   $ make prepare
   $ make modules_prepare
   $ make M=drivers/usb/serial/
   ```

9. 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].

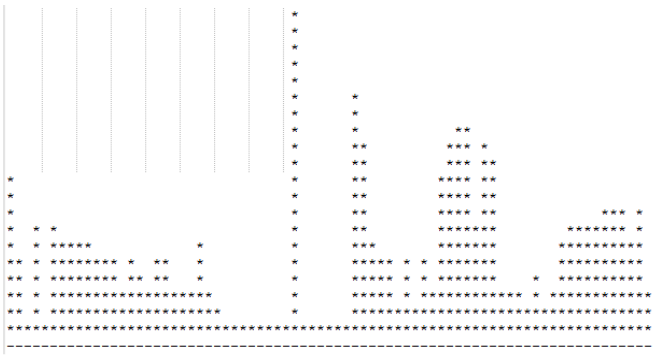

Figure 1. Output Data Histogram

```
theta: 1.47 Dist: 01143.25
theta: 2.72 Dist: 01189.00
theta: 3.91 Dist: 01218.50
theta: 5.05 Dist: 01234.25
theta: 6.28 Dist: 01257.25
theta: 7.47 Dist: 01282.25
theta: 8.83 Dist: 00000.00
theta: 10.05 Dist: 00000.00
theta: 11.27 Dist: 00000.00
theta: 12.50 Dist: 00000.00
theta: 13.72 Dist: 00000.00
theta: 14.94 Dist: 00000.00
theta: 16.16 Dist: 00000.00
theta: 17.38 Dist: 00000.00
theta: 18.83 Dist: 00890.50
theta: 19.81 Dist: 00000.00
theta: 21.03 Dist: 00000.00
theta: 22.25 Dist: 00000.00
theta: 23.47 Dist: 00000.00
theta: 24.70 Dist: 00000.00
theta: 25.92 Dist: 00000.00
theta: 27.39 Dist: 00818.00
theta: 28.66 Dist: 00811.75
theta: 29.58 Dist: 00000.00
theta: 31.36 Dist: 00707.50
theta: 32.56 Dist: 00701.00
theta: 33.75 Dist: 00696.00
theta: 35.06 Dist: 00692.50
theta: 36.17 Dist: 00737.75
theta: 37.41 Dist: 00712.25
theta: 38.64 Dist: 00687.00
theta: 39.78 Dist: 00691.50
theta: 41.09 Dist: 00691.50
```

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_SYNCBI
T) ?"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::DR
IVER_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;
}
```