Week 8 Studio 1 – Let there be light

Core Objectives:

- **C1. Introduction to Lidar**
- **C2. Installing Lidar**
- **C3. The SDK and Sample Program**
- **C4. Writing your own program**
- **C5. Visualizing the data**
- **C6. Tweaking the Visualization**
- **C7. [Optional] Live data**
- **C8. For your exploration**

Preparation (Before the studio):

- Download all the relevant files.
- Transfer "**`rplidar_sdk_v1.5.7.zip`**" and "**`w8s1.zip`**"over to your Pi, unzip them in a convenient location (e.g. Desktop).

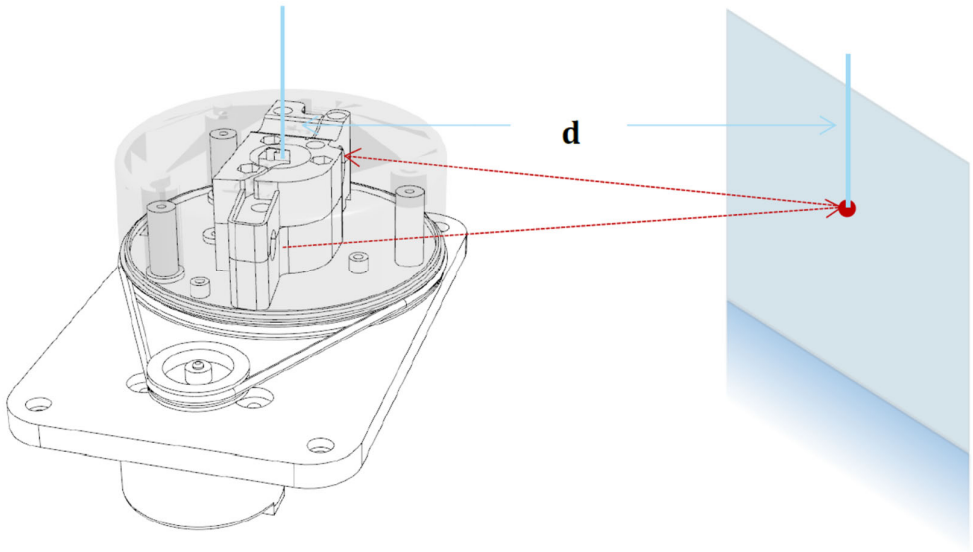
Studio Setup:

- Work with your project team for this studio as you need both the Pi and Lidar.
- Remember to bring the following:
  - Raspberry Pi (with SSH and VNC configured)
  - Ethernet network cable
  - PowerBank (fully charged ☺)
  - [Optional] Power Adapter (Wall plug ➔ USB)

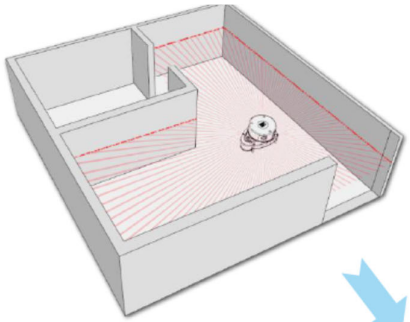
C1. Introduction to Lidar

The Lidar unit used in our course is the **SLAMTEC RPLidar A1**. It is a low cost 360 degree 2D laser scanner (LIDAR) solution developed by the company SLAMTEC. The system can perform 360° environment scans within a 6 meter range. [1]

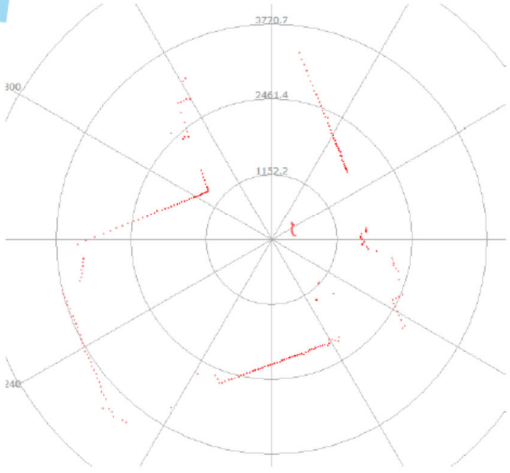
Lidar works on pretty simple principles: The light transmission unit sends out a pulse of light, while the detection unit attempts to capture the returned (reflected) light pulse. The duration elapsed between the transmission and detection as well as the wavelength can then be processed to give the distance of the obstacle. [1]



Now, with a constantly rotating mount, the mechanism described above can be used to "scan" the surrounding and give a 360° map:



*Note：The LIDAR scan image is not directly relative to the environment showed here. Illustrative purpose only.



Let us now attempt to find out more specification details for this particular Lidar unit. Skim through the given RPLidar A1 data sheet and find out the following information:

Minimum and Maximum Scan Rate (how many revolution per second)	
Sample per revolution	
Communication Interface (i.e. the hardware interface)	
Communication Protocol Details (i.e. parameter of the communication)	
Power Requirement (Voltage, Current range for optimal use) (Indicate the range for the scanner and motor unit)	

C2. Installing Lidar

Caution
The Lidar unit is quite "power hungry" and will not work properly if the electrical current supplied is insufficient. Since we are going to connect the Lidar to the Raspberry Pi via a USB port, you need to ensure the Pi unit received sufficient power. Hence, connect the **Raspberry Pi** either to a **wall plug with good adapter** or **the power bank**.

Unbox the RPLidar A1 unit given to your group. As shown below [starting from the left], you should see the **Lidar unit**, "**serial to USB bridge" chip** and a **white serial cable**.

Connect the serial cable to the interface port at the bottom of the Lidar unit, then connect the other end to the "serial to USB Bridge" chip. Plug in the microUSB cable, make sure you use a good quality microUSB cable, e.g. the blue color cable provided for you.

Note: Only connect the Lidar to the Pi unit whenever you need to test it. Although there is no harm in keeping the Lidar unit connected with the Pi, the motor rotating noise do get a bit annoying after a while. ☺

C3. The SDK and Sample Programs

Key Idea:
For hardware components that require intricate communication protocol, the manufacturer usually provide SDK (Software Development Kit) for the users. SDK commonly consists of library / source code which provides commonly performed operations.

The SLAMTEC RPLidar comes with a SDK as well as a few sample programs. Let us try them out in this section.

1. Use **Remote Desktop** (e.g. VNC) to connect your laptop to the Pi. If you haven't setup the use of remote desktop, take a look at Week 7 Studio 1's handout (first activity). (**Note**: activity C4 requires X-windows display, so simple SSH is not sufficient).
2. All the steps below (and subsequent activities) should be performed **on Pi.**
3. Unzip the **`rplidar_sdk_v1.5.7.zip`** on your Pi's desktop if you have not done so. We assume you unzip to the folder **`rplidar_sdk_v1.5.7/`** for the following steps. Adjust the commands accordingly if your folder name is different.
4. Use a **terminal**, go to **`rplidar_sdk_v1.5.7/sdk`** subfolder.
5. Type **`make`**, compilation should start automatically. There may be 1-2 **warning** but no compilation error.
6. Go to **`output/Linux/Release`** subfolder, you should see 3 files:
    a. **`librplidar_sdk.a`** : This is the SDK library object code. We will learn more in the next section.
    b. **`simple_grabber`**, **`ultra_simple`** : These are the sample programs **executable**.

Pit Stop:
We will now run the sample program to verify that the RPLidar unit is working properly.

7. When a new USB device is plugged into Pi, a **device block file** will be created at the **`/dev`** folder. Let us find out the which USB port is uses by our RPLidar unit.

8. Perform "**`ls /dev/ttyUSB*`**" in the terminal. These are the USB devices currently detected on Pi, i.e. if you see "**`cannot access /dev/ttypUSB*……`**" ➔ there are no USB device plugged in yet.

9. Plug in the RPLidar unit to the USB Port **on the Pi**.

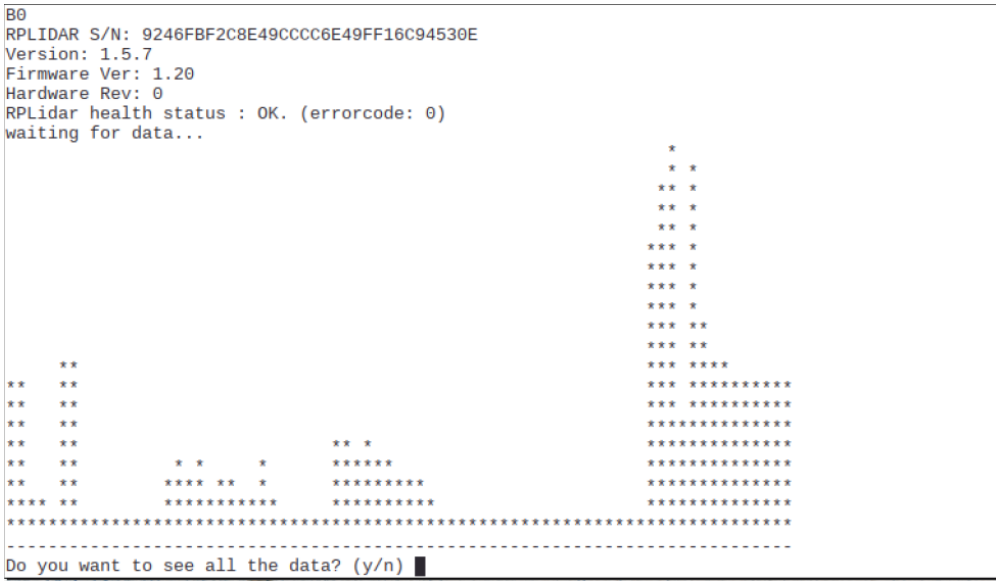
10. Give it a moment (~5 seconds), redo step 8 and you should see a new USB device listed. Take note of the **name of the USB** port. If RPLidar is the only USB device plugged in, then it is likely to be **`/dev/ttyUSB0`**.

11. Ensure you are still in the folder of the compiled sample program executable (i.e. step 6). Execute the following command in the terminal:

    `./simple_grab /dev/<`*`USB port of the RPLidar unit`*`>`

    e.g.
    `./simple_grab /dev/ttyUSB0`

    You should see output similar to the following:



    Important: If you see a histogram that is fully filled with "*", then it is **very likely** that your RPLidar unit does not have sufficient power.

12. We will now browse through the sample source code to find out:
    a. Basic functionalities provided by the SDK.
    b. Basic layout of a user program that utilizes the SDK.

13. Go to **`rplidar_sdk_v1.5.7/sdk/sdk/include`** (note that the two **`sdk/`** is not a typo ☺). Open up the file **`rplidar_driver.h`** and browse through it. Find out the following:

Programming Language used for the SDK?	
What is the method / function to stop the RPLidar motor?	
What is the method to get one complete set of scan data?	
How is the motor speed (i.e. revolution per second) controlled?	

14. Go to **`rplidar_sdk_v1.5.7/sdk/app/simple_grabber/`** . This folder contains the **source code** of the sample program we executed back in step 11.

15. Open up the file **`main.cpp`**, browse the **`main()`** function and pay attention to the following:
    a. How is a RPLidarDriver object created (constructed?)
    b. The steps involved to connect the RPLidar unit via serial port.
    c. The steps to start the scan and extract the scan data.

    Note: We are asking you to get a high level sense of the procedure only. So, don't panic if you are stumped / confused by some part of the code.

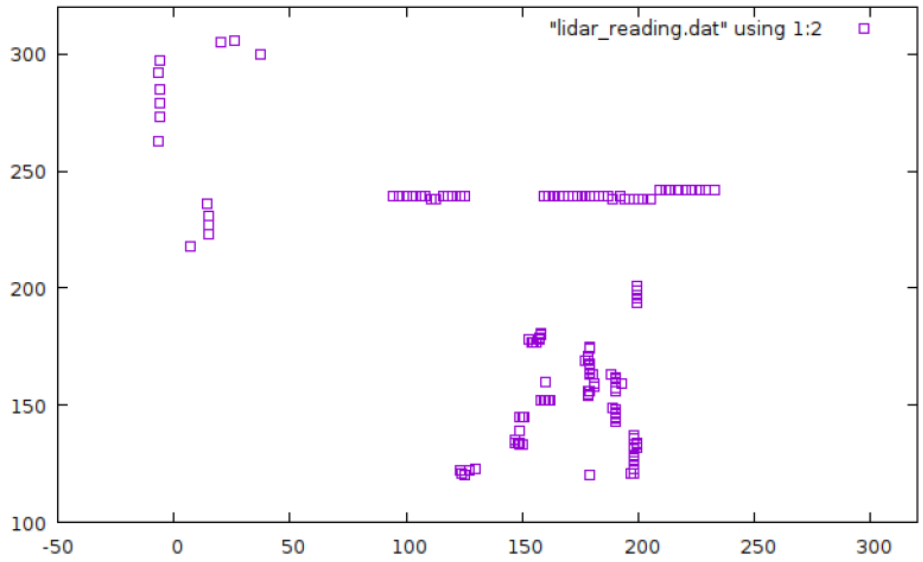
16. Browse the **`plot_histogram()`** function, answer the following:

What does each of the bar in the histogram represents?	
What does the height of each of the bar represents?	

C4. Writing your own program

Key Idea:
We will try to write a "simple" program that interacts with RPLidar unit. The key learning point is to understand how to link a user source code with the SDK Library.

1. The histogram representation of the scan data is quite awkward and hard to understand. It would be much nicer if we can show the scan data in a proper 2D plot like:



You can imagine the RPLidar unit is at the center of the plot, the points are obstacles detected in the surrounding 360° area. We will attempt to produce this 2D plot now!

Credit:
Code / idea were adapted from the sample programs developed by SLAMTEC.

2. Unzip the **`w8s1.zip`** if you have not done so. We assume you unzip the file into a folder **`w8s1/`** in the following steps. Adjust the command accordingly if your folder name is different.

3. For convenience, the following are already performed for you:
   - The header (includes) files are copied into **`w8s1/Include/`**
   - The compiled SDK library object file in **`w8s1/Lib/`**

   If you are developing on your own in future, remember to place these files at a location that is accessible to your source code.

4. Open up the **w8s1.cpp**, take a look at the function **capture_and_display()**, pay attention to the **printf()** statement in the **for-loop.** That **printf()** statement prints out the scan data with minimum processing on screen.

5. Compile the given **w8s1.cpp** as follows:

```
g++ w8s1.cpp Lib/librplidar_sdk.a -lpthread -lm
```

   - "**g++**" invokes the **C++** compiler.
   - "**Lib/librplidar_sdk.a**" indicates the library object file for static linking.
   - "**-lpthread -lm**" instructs the compiler to link with the ***pthread*** and **math** libraries.

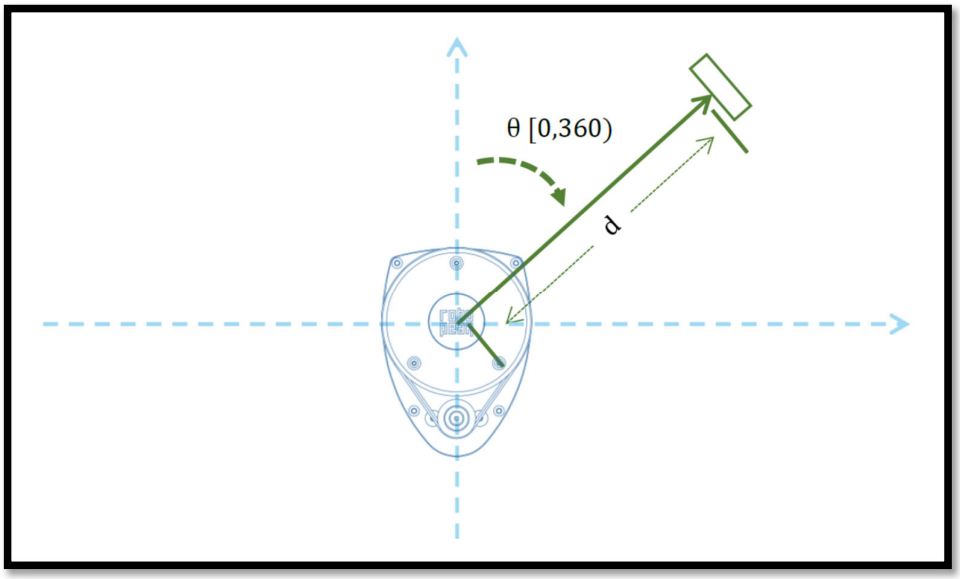
6. Execute the compiled binary:

```
./a.out /dev/<USB port for Lidar>
e.g.
./a.out /dev/ttyUSB0
```

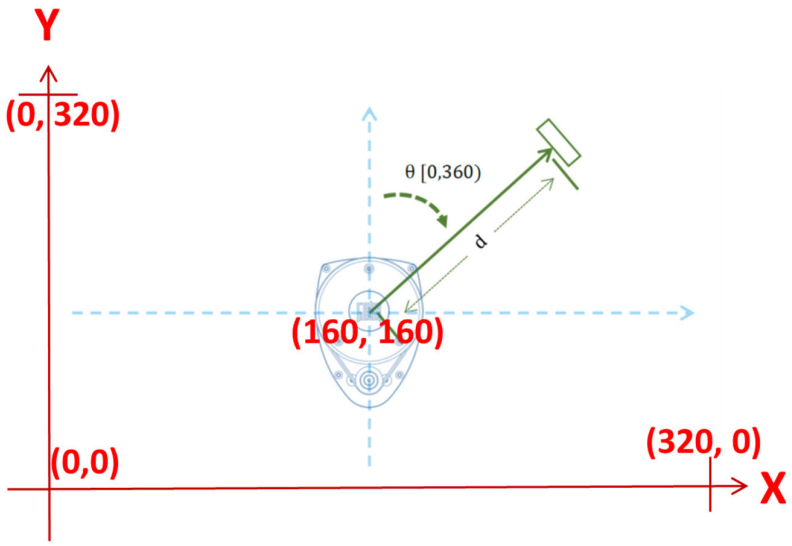
   You should see the raw scan data on screen. The data is shown as:

   [**Theta:** <Angle> **d:** <Distance in millimetre (mm)]

   For your reference, the orientation of the data is illustrated below:



7. We can "easily" transform the raw data into 2D coordinate by using the following idea:
    - The visualized 2D plane is a canvas with 320 x 320 pixels (arbitrarily chosen as it fits well for most display size).
    - The bottom left of the canvas is the origin (0, 0).
    - The Lidar is at the center of the canvas, i.e. (160, 160) in this case.
    - The obstacle detected can then be **calculated as an offset from (160, 160) .**



You only need to fill in **two lines of source code (actually only two math expressions) at line 123 with the "//TODO: ...." Tag.** (Hint: I hope you still remember your trigonometry .....). Show the instructor or your TA of the expression before proceeding.

8. Compile your program and run it. The transformed data is stored in "**`lidar_reading.dat`**". We will now proceed to visualize the data.

C5. Visualizing the data

What are we doing?
C/C++, though versatile, does not have simple "built-in" means to visualize data. Fortunately, there are many options available on Raspbian / Linux platform. For our purpose, we have chosen **gnuplot**, which is a simple yet powerful application to plot data. Think of it as the Excel plotting function without the bulk of other non-critical features.

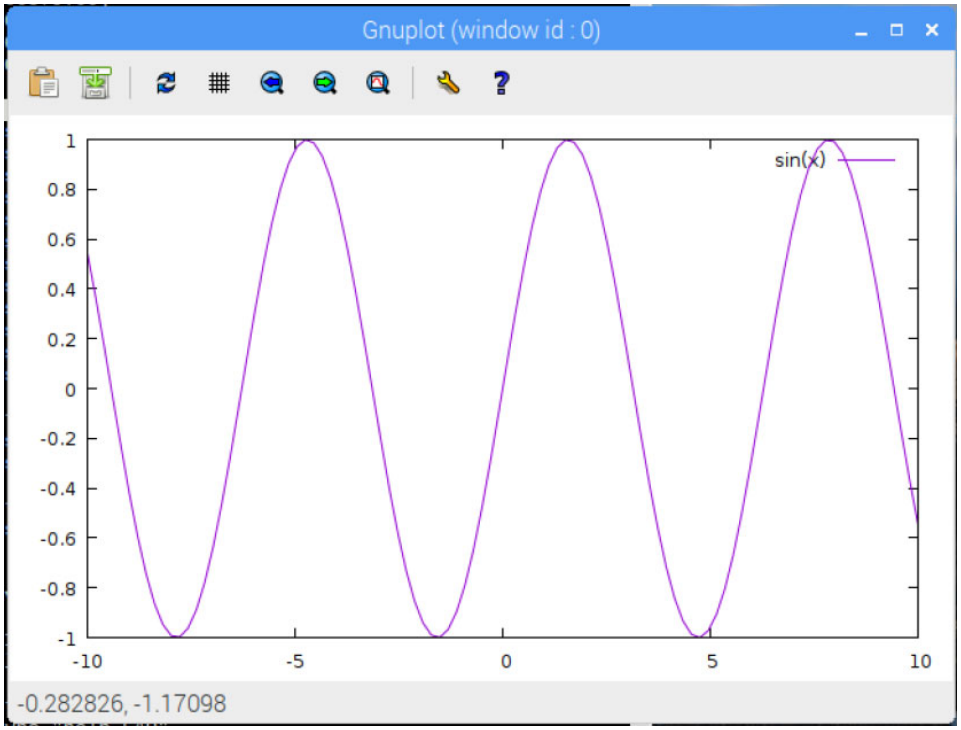
1. Install **gnuplot** by:

```
sudo apt-get install gnuplot-x11
```

2. Verify **gnuplot** installation by:

```
gnuplot
gnuplot > plot sin(x)        //note:  "gnuplot >" is the prompt
```

You should see the following:



3. Let us plot the data gathered from our program in C4. Close the previous plot, then enter the following in the gnuplot prompt:

```
gnuplot > plot [0:320] [0:320] "lidar_reading.dat" using 1:2 with points pointtype 4
```

You should see a plot similar to C4, step 1.

4. As it is quite painful to type the long command, we have written it into a "gnuplot file", which is simply a set of gnuplot commands saved in a text file.
    a. Exit **gnuplot** by pressing **<Ctrl-D>** or enter "**quit**".
    b. On your normal prompt, enter:

```
gnuplot lidar_plot.plt --persist
```

    - **lidar_plot.plt** is the "gnuplot file". You can open it using any text editor to take a look.
    - **--persist** instructs gnuplot to keep the plot window open. If you leave out this flag, the plot window will be closed as soon as gnuplot exits.
5. We now have a way to visualize the scan data from lidar, the steps are essentially:
    a. Execute program from C4 to output the transformed scan data.
    b. Execute **gnuplot** to show the information.

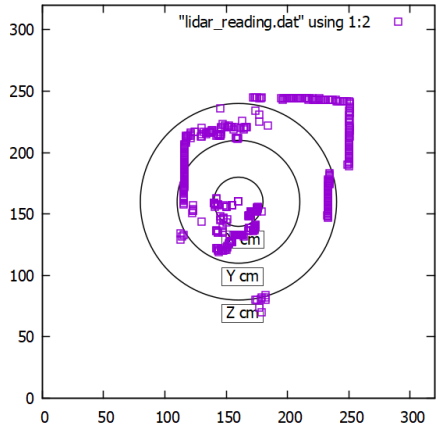
C6. Tweaking the Visualization

What are we doing?
gnuplot is a pretty powerful tool. You can use built-in commands to change the output and adds useful features. We will show you one simple way to augment your visualization from C5 in this section.

1. On your normal prompt, enter:

```
gnuplot lidar_plot_tweak.plt --persist
```

   You should see a plot similar to the following:



2. Notice that there are now three concentric circles with labels at the bottom (note: the first circle's label is slightly obscured in the plot shown). These circles are there to give some indications of distance of obstructions detected by lidar.

3. Take a look of the file "**lidar_plot_tweak.plt**", you should see that the label's text is defined by these lines:

```
set label "X cm" at 150,130 boxed
set label "Y cm" at 150,100 boxed
set label "Z cm" at 150,70 boxed
```

4. Find out the actual distance of the 3 circles and change the label accordingly. (hint: you can do this empirically by measuring real objects or take a look at how distance is represented in the source code). Show the TAs the updated plot with the labels updated properly.

5. With the correct distance value now, you can try to use the lidar to detect your surrounding. A few more ideas to improve the visualization for you to consider:
    a. Change the circle radius so that they are measuring more useful distances?
    b. Change the display canvas size (it is 320 x 320 at the moment) so that you can read more details easily?
    c. Add other artefacts to enhance the visual? e.g. add a triangle to show the position of the lidar unit?

C7. [Optional] Live data

1. You can improve on C5 by two simple tweaks:
    a. Update the scan data output file regularly (say every second, or every time the user type enter)
    b. Instruct the gnuplot to replot the graph regularly.

2. For (1a), you can consider to place the capture_and_output() function in a loop so that the output file is updated regularly.

3. For (1b), we have provided a simple "gnuplot file" with the instruction to replot every 3 seconds. To run it, enter:

```
gnuplot lidar_plot_live.plt
```

   This allows gnuplot to re-process the data file every 3 seconds. So, if your program can now output "live" data, you will be able to see the visualization in gnuplot "live" too!

4. Remember to use the distance values found in C6 so that the live visualization is more useful.

C8. For your own exploration

It is now time to assure you that you wont be coding at the C/C++ level for the Lidar interaction for Alex. ☺ As the data read from Lidar needs to be fed into SLAM algorithm for environment map generation, there is no need for you to directly interface with Lidar.

However, don't conclude that this studio is "useless" and a "waste of time"! This studio intends to show you the typical underlying processes involved in interfacing with a hardware component from C/C++ program. Hopefully when you use the high level software components later in the project, you wont treat the whole process as "blackbox" or worse, assume it is "black magic". ☺

Browsing through the SDK source code is also **extremely educational**, especially for your understanding on serial communication. If you want to learn more, start from the excellent protocol documentation of the RPLidar [3].

References / Resources:

1. SLAMTEC RPLidar Datasheet (LD108_SLAMTEC_rplidar_datasheet_A1M8_v1.0_en.pdf)
2. SLAMTEC RPLidar SDK Manual(LR002_SLAMTEC_rplidar_sdk_v1.0_en.pdf)
3. SLAMTEC RPLidar Communication Protocol Documentation(LR001_SLAMTEC_rplidar_protocol_v1.0_en.pdf)
4. GNUPlot documentation(http://www.gnuplot.info/documentation.html)