LeddarC SDK Tutorial

Written by Caleb Kisby

Table of Contents

Introduction

The LeddarTech SDK provides a variety of functions to handle a Leddar™ LIDAR sensor. In particular, it provides functions to connect to a sensor, get real-time points detected by the sensor, read points from a Leddar record file, and configure the sensor's settings.

The LeddarTech SDK is implemented in both C (LeddarC) and .NET (LeddarNET). For this tutorial, we will be using C. Throughout the tutorial, we may mention changes that must be made in order to convert a Leddar project using C into one using C++. However, we will not discuss such changes for .NET. Differences between the C and .Net implementations is mostly syntactical, but one should consult the official documentation to read more.

This tutorial will cover the concepts and steps necessary for the reader to start programming a project using the LeddarC functions. We aim to give the reader a thorough foundation, so that they may confidently complete such a project with little reference to the scattered documentation.

This tutorial assumes that the reader has already installed the necessary LeddarTech SDK requirements and is able to compile their program using the example Makefile in the LeddarC example code. For more information, consult the official documentation.

The full LeddarTech SDK reference is available in the "LeddarSDK3.2.0_x86_64.tar.gz" zipped file at the LeddarTech support portal: https://support.leddartech.com/ . Further questions not answered in this tutorial may be asked through the LeddarTech support portal "Contact Us" tab.

Basic Concepts

In this section, we discuss basic concepts necessary to understand before connecting to the LIDAR sensor. In particular, we explain the use of `LeddarHandle` and LeddarC return codes.

Leddar Handles

LeddarC communicates with the LIDAR hardware driver through a *handle*. This handle allows the user to request actions from the sensor, such as a connection request, read data request, and a configuration request.

The `LeddarHandle` class represents such a handle. These handles are constructed as follows:

```
LeddarHandle gHandle = new LeddarHandle();
gHandle = LeddarCreate(); // Acts as a constructor
```

```
// Do stuff
LeddarDestroy(gHandle);
```

Here we allocate memory to our handle `gHandle`, and then construct the handle using the `LeddarC` function `LeddarCreate`. When we are finished using the handle, we must remember to deallocate memory with the LeddarC function `LeddarDestroy`.

Warning: A `LeddarHandle` is of type `void*`, i.e. it is already a pointer! This means that we *should not* make a `LeddarHandle*` pointer should we wish to point to the `LeddarHandle`. This is also why we must allocate memory to `gHandle` with `new LeddarHandle`. This is often a point of confusion; always keep in mind that `LeddarHandle` is a pointer.

Note: Alternatively, one may use `LeddarCreateWithConsole` to construct the handle with command line arguments. This is done in the `main` function as follows:

```
int main(int argc, char** argv) {
      LeddarHandle gHandle = new LeddarHandle();
      gHandle = LeddarCreateWithConsole(argc, argv);
      LeddarDestroy(gHandle);
}
```

Error and Return Codes

Every LeddarC function returns an `int` code. These codes are used to test the quality of connection and to catch errors. The expected code is `LD_SUCCESS`, which indicates that the LeddarC function succeeded without a hitch. Other codes include:

```
LD_ACCESS_DENIED      LD_TIMEOUT       LD_START_OF_FILE
LD_END_OF_FILE        LD_NO_RECORD     LD_ALREADY_STARTED
LD_NO_DATA_TRANSFER   LD_NOT_CONNECTED
LD_INVALID_ARGUMENT   LD_ERROR         LD_NOT_ENOUGH_SPACE
```

Most of these error codes are returned by all LeddarC functions, and their meaning is evident in the context of that function. `LD_ERROR` is used for general errors not covered by the remaining codes.

Certain error codes, however, are specific to particular functions. For example, `LD_END_OF_FILE` is returned by LeddarStepForward to indicate that the file reading cannot continue because it has reached the end of the file.

We will discuss specific error codes in more detail when they have the potential to arise.

Connecting to the LIDAR

In this section, we walk through the steps required to connect and disconnect the program to the sensor. In addition, we also discuss the steps for loading a file record before reading data.

The LeddarConnect Function

The LeddarC function used for connecting to the sensor is called `LeddarConnect`. Its signature is the following:

```
LeddarConnect(LeddarHandle aHandle, char* aConnectionType,
      char* aConnectionString)
```

The argument `aHandle` is just our handle discussed above. The function also specifies a connection type, `aConnectionType`. If the sensor is connected via USB, this value should be `"USB"`. If the sensor is connected via a serial port, then this value should be `"SERIAL"`.

The aConnectionString more generally is given as a formatted string with connection information. This is discussed in much detail in the official documentation. However, for our purposes, this value can just be the address of the sensor (default value is '0').

After we are done with the connection, we must use the LeddarC function `LeddarDisconnect`. We will see this used in the example code below.

Warning: In order to successfully establish a connection, the user must have privileges required to access the USB or serial port. Without such privileges, `LeddarConnect` will return `LD_ACCESS_DENIED`. Executing your program with `sudo` will solve this. If the reader does not have root (`sudo`) access on their machine, they should contact their system administrator.

The Usual Routine

The typical routine for establishing a connection with the sensor is as follows. First, one calls `LeddarConnect` to request a connection, and tests if this succeeds (with `LD_SUCCESS`). Then,

one must loop to maintain connection, and check that they are still connected with `LeddarGetConnected` during each iteration (which returns `LD_SUCCESS` if the sensor is still connected).

The following code is typical of this routine:

```
LeddarHandle gHandle = new LeddarHandle();
gHandle = LeddarCreate();

if (LeddarConnect(gHandle, "USB", 0) == LD_SUCCESS) {
      while (LeddarGetConnected(gHandle) == LD_SUCCESS)
            // Do stuff
}

LeddarDisconnect(gHandle);
LeddarDestroy(gHandle);
```

Loading a Record File

If instead one wishes to load data from a previously recorded Leddar record file, the routine is very similar to that of connecting to the sensor. The LeddarC function used for loading records is called `LeddarLoadRecord`. The signature of this function is the following:

```
LeddarLoadRecord(LeddarHandle aHandle, char* aFileName)
```

This signature is much simpler than that for `LeddarConnect` because we do not need to fuss with connection options. Instead, we just specify our handle aHandle and the name of the file to be loaded as a `char* aFilename`. Note that the file must be of type *'ltl'*, i.e. a Leddar record file.

Loading a file for reading is typically done with the following routine, similar to that for connecting to a sensor. Note that we use `LeddarGetRecordLoading` instead of `LeddarGetConnected` to check that the record connection still holds.

```
LeddarHandle gHandle = new LeddarHandle();
gHandle = LeddarCreate();

if (LeddarLoadRecord(gHandle, "filename.ltl") == LD_SUCCESS) {
      while (LeddarGetRecordLoading(gHandle) == LD_SUCCESS)
            // Do stuff
}

LeddarDisconnect(gHandle);
LeddarDestroy(gHandle);
```

Getting Data from the LIDAR

In this section, we explain the standard LeddarC routine for getting live data from the LIDAR: Callback functions. We also discuss getting data from a loaded record file. We conclude with some remarks about an alternative routine for getting data.

Getting Live Data with Callbacks

The primary method used by LeddarC to get live data from the sensor is via a particular Callback function. That is, this function will be "called back" whenever we have new data points from the sensor. We recommend that the reader use the Callback function in the LeddarC example code. In order to use this Callback function, we must implement it ourselves. The standard code for this is the following:

```
static void DataCallback(void *aHandle) {
      ldDetection lDetections[50];
      unsigned int I, j, lCount = LeddarGetDetectionCount(aHandle);

      if (lCount > ARRAY_LEN(lDetections))
            lCount = ARRAY_LEN(lDetections);

      LeddarGetDetections(aHandle, lDetections,ARRAY_LEN(lDetections));

      if (LeddarGetRecordSize(gHandle) != 0)
            printf("%6d ", LeddarGetCurrentRecordIndex(gHandle));

      for (i = 0; i < lCount; ++i)
            printf("%5.2f ", lDetections[i].mDistance);

      puts("");
}
```

This function gets the next set of points from the sensor, and performs basic error checks on it. The `LeddarGetDetections` function is the one that actually gets the next set of points from the sensor. We also print each of the obtained points to console.

We may use the LeddarC `LeddarSetCallback` function to associate this Callback function with the sensor. `LeddarSetCallback` sets `DataCallback` to be executed whenever the sensor detects a new set of points.

So in order to get live data from the sensor, we call `LeddarStartDataTransfer` to start transferring data from the sensor to the program. We then set our Callback function to get the transferred points with `LeddarSetCallback`. Finally, we must stop transferring data with

`LeddarStopDataTransfer` and remove the Callback with `LeddarRemoveCallback`. The following code performs this routine:

```
// We first ensure that we have no error when we begin transfer
CheckError(LeddarStartDataTransfer(gHandle, LDDL_DETECTIONS);
LeddarSetCallback(gHandle, DataCallback, gHandle);

WaitKey();

LeddarStopDataTransfer(gHandle);
LeddarRemoveCallback(gHandle, DataCallback, gHandle);
```

Getting Data from a Record

Another advantage of using this Callback function is that it can be reused to get data from a record file. We perform exactly the same routine as before: start the data transfer, set the Callback, stop data transfer when ready, and then remove the Callback.

The `LeddarGetDetections` method used in `DataCallback` gets the points pointed to by a Leddar frame. That is, if we wish to read all of the points in a record, we must move the frame forward through the file. Moving the frame is done through the functions `LeddarStepForward`, `LeddarStepBackward`, and `LeddarMoveRecordTo`. We provide examples of the usage of each, in the context after the Callback has been set:

```
LeddarStepForward(gHandle);
LeddarStepBackward(gHandle);
LeddarMoveRecordTo(gHandle, 0); // Move frame to beginning of file
```

Getting Live Data without Callbacks

Although using the `DataCallback` function is highly recommended by the official documentation, it is not strictly necessary. One may invoke `LeddarGetDetections` on its own. This is useful for situations in which Callback functions are highly inconvenient, or for situations in which the user would like to handle data getting themselves.

The official documentation warns, however, that one "is not guaranteed to get coherent data." This is because points that are covered on the sensor do not register, resulting in fewer than the 16 expected points. In addition, occasionally the sensor detects *more* than 16 points! Some error checking similar to that used in `DataCallback` should be used to ensure that one gets coherent points.

`LeddarGetDetections`, used on its own, has the following signature:

```
LeddarGetDetections(LeddarHandle aHandle, LdDetection* aDetections,
      LeddarU32 aLength);
```

Note that `aDetections` is an array of detection points, and `aLength` is its length (default should be 16, since we have 16 points to detect). This array is initially empty, until `LeddarGetDetections` stores the detections in `aDetections`.

In order to iterate until the sensor does not detect another set of points, we may use the LeddarC function `LeddarWaitForData`. `LeddarWaitForData` waits a certain specified time for more data, and returns `LD_SUCCESS` if it obtains more data within that time.

```
while (LeddarWaitForData(gHandle, 200) == LD_SUCCESS) {
      LeddarGetDetections(gHandle, lDetections, ARRAY_LEN(lDetections))
}
```

Warning: Some of the LIDAR configuration variables, such as those dealing with the framerate of the sensor streaming, may only work with the Callback method. `LeddarSetCallback` might adjust its rate according to such a configuration setting. In this Callback-free method, we do *not* account for different stream rates!

Configuring the LIDAR

Now that we have walked through how to set up a connection and read data from the sensor, we will now discuss configuration settings for the sensor.

Basic Functions

LeddarC provides functions for modifying and accessing configuration variables. To modify a variable, one must use the `LeddarSetProperty` function, followed by the `LeddarWriteConfiguration` function to commit that change. To access a variable, one must simply use the `LeddarGetProperty` function. We discuss each of these functions in turn.

```
LeddarSetProperty(LeddarHandle aHandle, LeddarU32 aId,
      LeddarU32 aIndex, double aValue)
```

`LeddarSetProperty` takes in our handle aHandle, the integer ID of the variable to change `aId`, and the new value `aValue` for that variable. In addition, some array properties require an index `aIndex`. `aIndex` should be set to '0' for single properties. The property is modified, and `LeddarSetProperty` returns only an error code.

```
LeddarGetProperty(LeddarHandle aHandle, LeddarU32 aId,
      LeddarU32 aIndex, double* aValue)
```

`LeddarGetProperty` takes in the same handle aHandle, and gets the value of the variable with ID `aId`. Note that this function does not return anything, but instead stores the resulting value in the pointer `aValue`. This function has a similar `aIndex` argument for array properties.

```
LeddarWriteConfiguration(LeddarHandle aHandle)
```

This function commits all changes made to the sensor configuration. This function *must* be called, otherwise no changes will be made.

Common Configurable Variables

There are a number of different sensor variables that one may wish to change or access – too many to list here! We will describe common or useful variables, listed by their ID (PID), and give their reasonable values. The complete list is available in the official documentation on the LeddarProperties.h file.

PID_BASE_POINT_COUNT: This variable specifies the number of points that we wish to use with our sensor. The Leddar LIDAR sensor default is 16 points, but this variable allows us to use sensors with more or fewer points. The value of this variable must be the number of points one wishes to use.

PID_OVERSAMPLING_EXPONENT: This variable changes the number of oversampling cycles. Increasing this enhances the accuracy of the point set, but reduces the measurement rate because we are performing more oversampling. This value must be a power of two, no more than 1024.

PID_THRESHOLD_OFFSET: This variable changes the amplitude threshold of the LIDAR beams. Higher values decrease the range of the beams. Additionally, higher values reduce the sensitivity of the beams. This value must be a decimal number from 0 to 2. The default value of 0 is recommended.