LeddarTech®
Solving Critical Sensing, Fusion and Perception Challenges

INSTALLATION GUIDE

Leddar™ SDK

Software Development Kit

Using LeddarTech Sensors in Windows

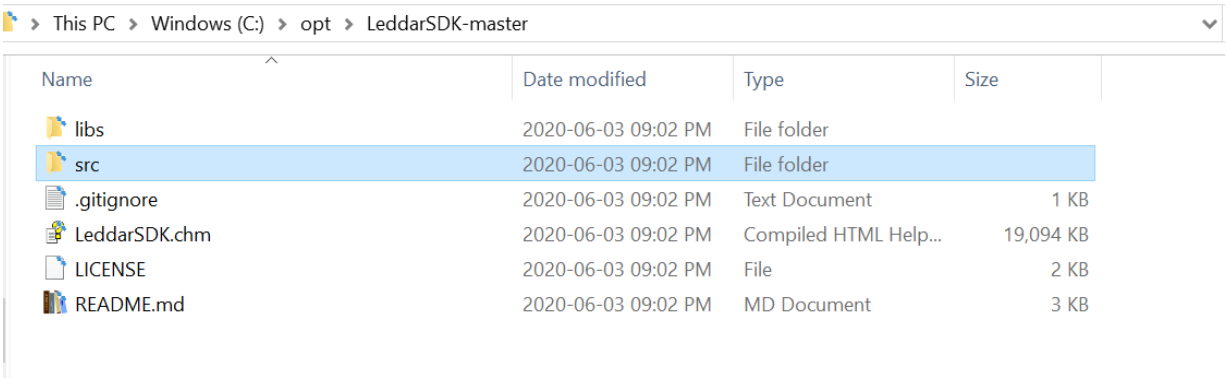
Table of Contents

1. LeddarSDK for Windows

Download the latest version of the SDK from https://github.com/leddartech/LeddarSDK.

Extract the LeddarSDK-master folder.

For the purposes of this Installation Guide, the LeddarSDK-master folder has been created in the **c:\opt** directory for enhanced ROS compatibility.



1.1. Building the SDK in Windows

Before proceeding with the installation of any other package than Python, you first need to build the LeddarSDK solution.

Build process uses cmake (https://cmake.org). The **CMakeLists.txt** file is in the **src** folder. It is recommended to do an out-of-source build in the **src/release** folder.

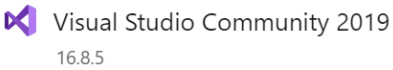
You can choose for what sensor to build the SDK by modifying the cmake project configuration. All sensors are enabled by default.

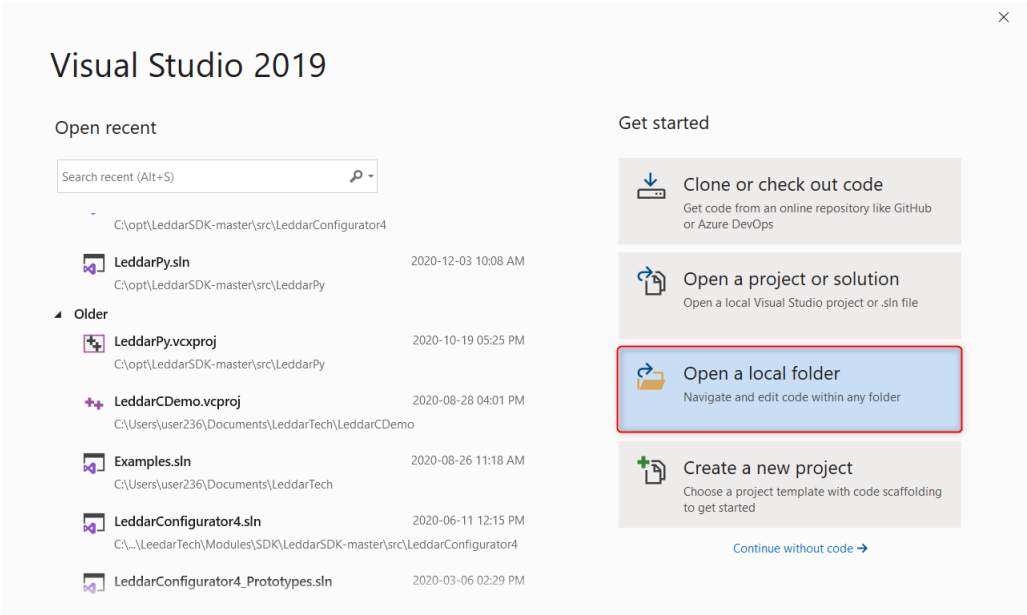
1.1.1. Windows C++ Package

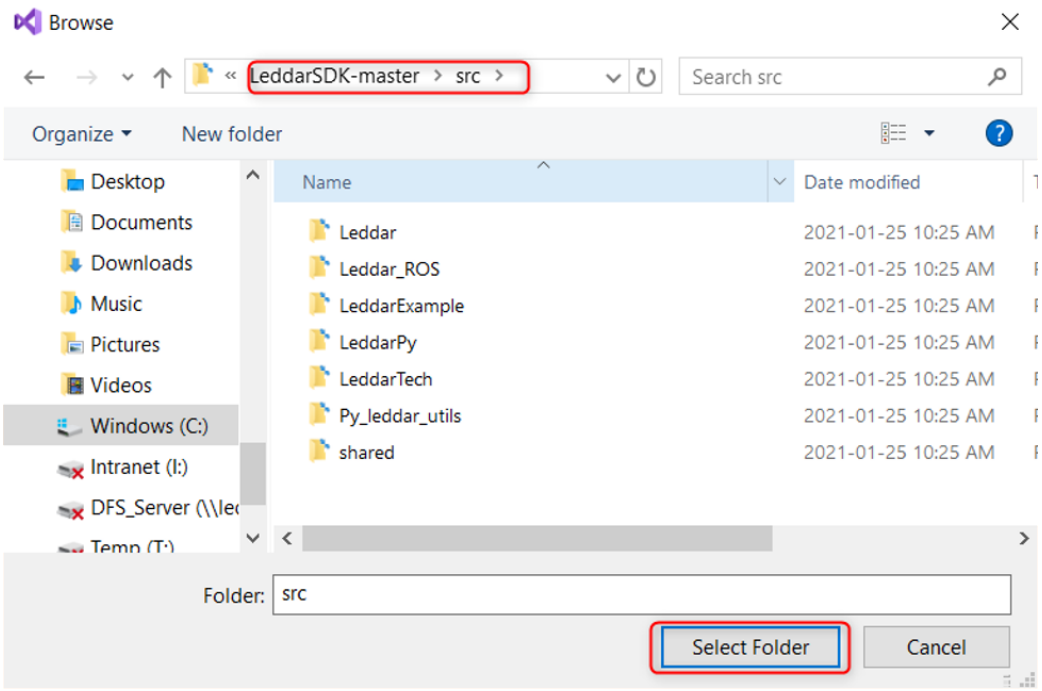
CMake is directly integrated in Visual Studio. Check the documentation here.

Open **Visual Studio 2019 (16.8.5)** and navigate to the folder containing a **CMakeLists.txt** file. In our case:

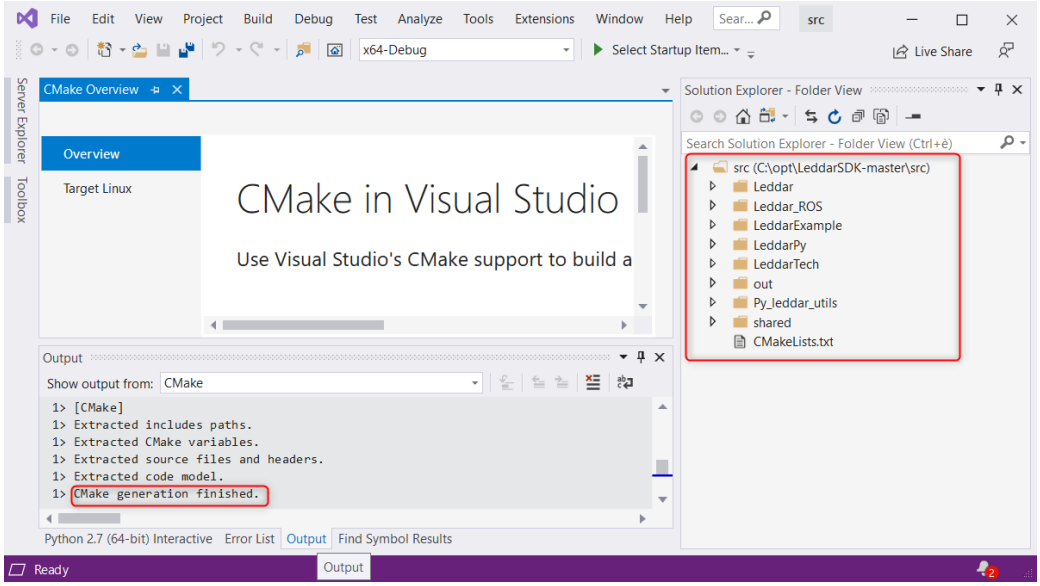
```
.\LeddarSDK-master\src
```





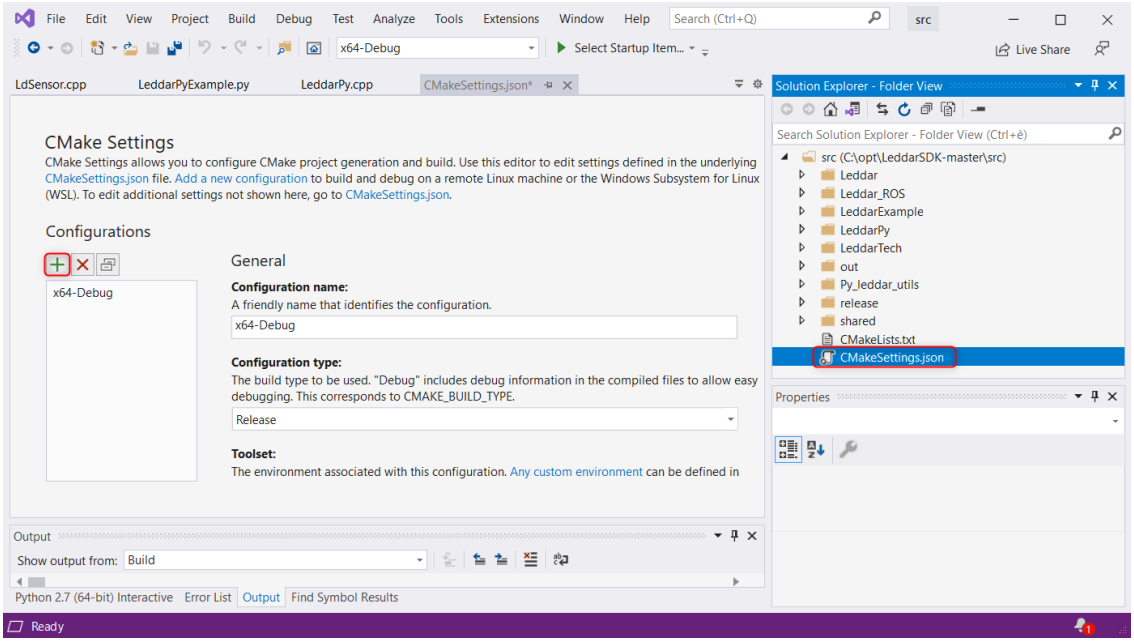


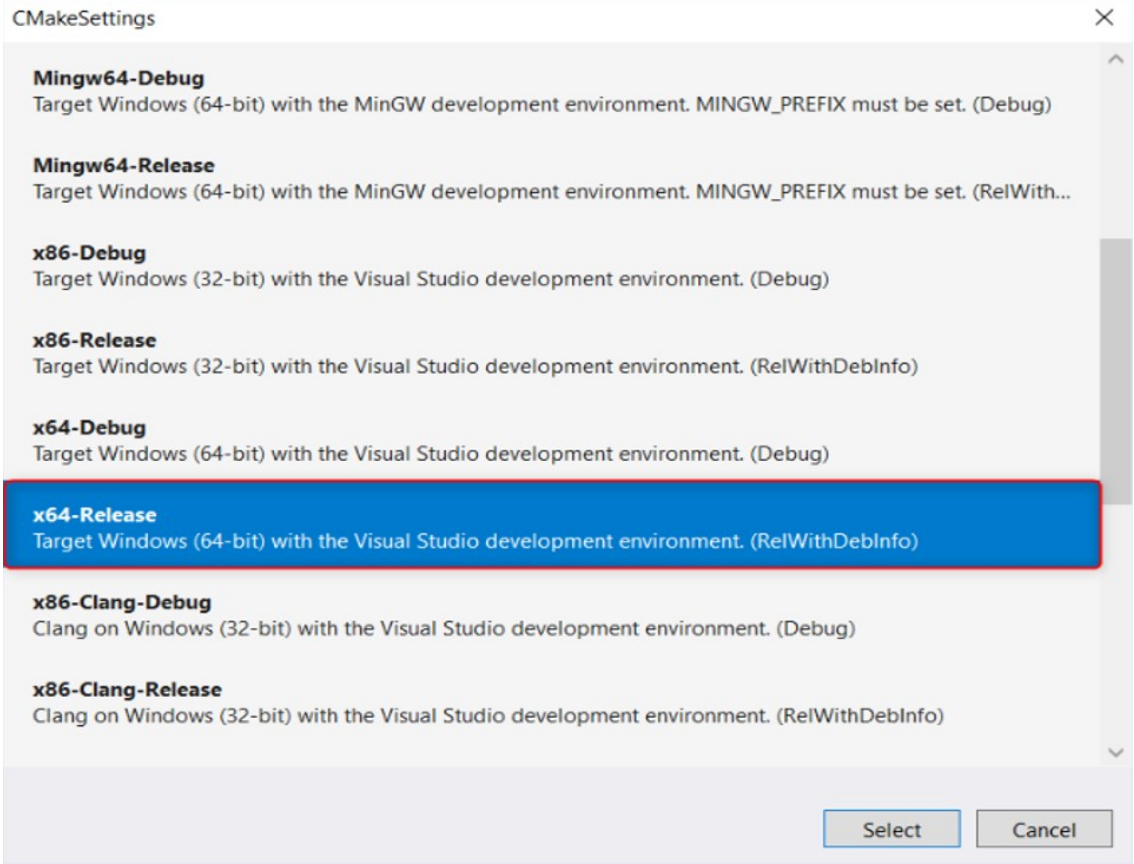
Visual Studio will automatically detect the cmake structure, allowing you to build and debug the solution.

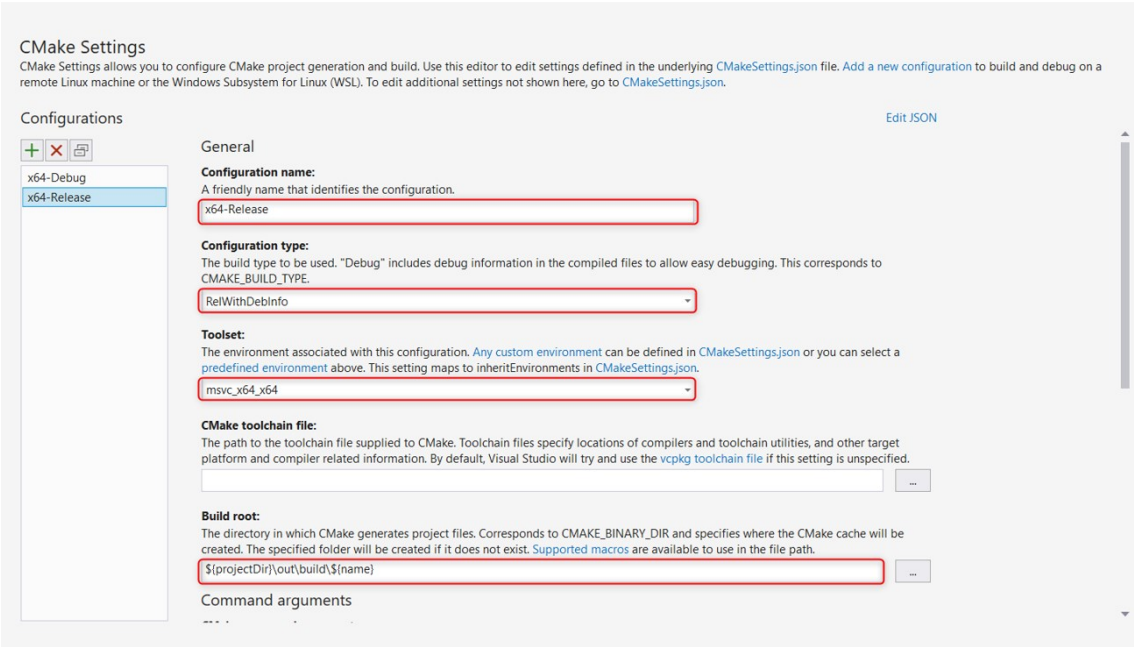


The SDK needs to be built in Release mode.

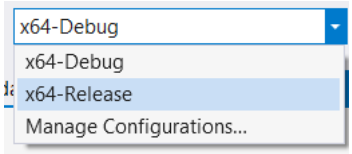
To do so, create a new configuration specifically for the release (e.g., x64-Release) and make sure to select the correct parameters.



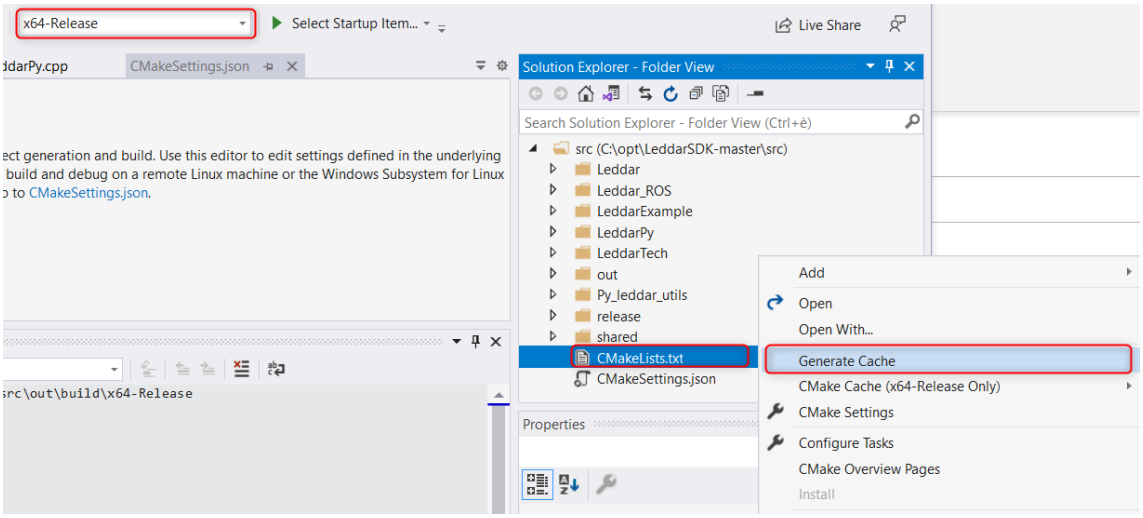


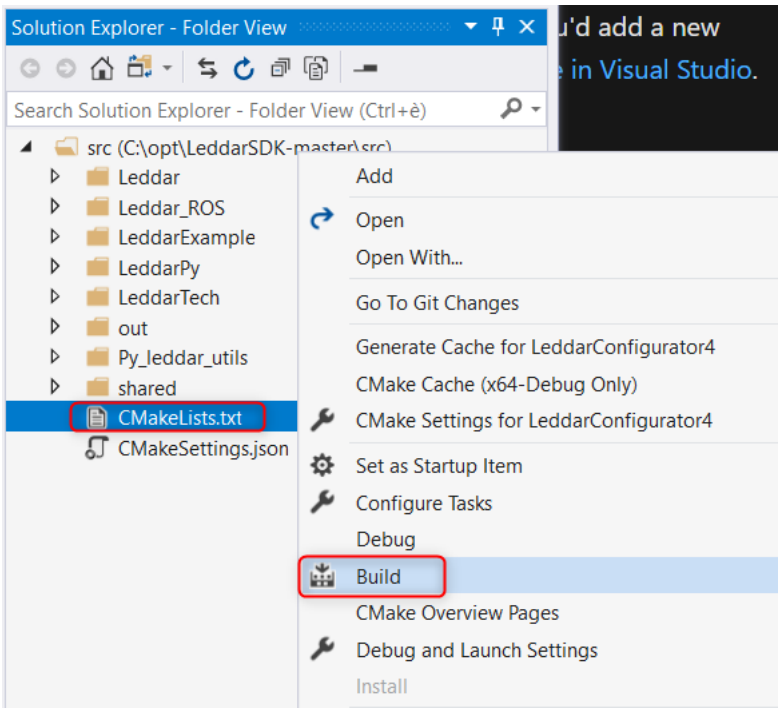


Save your new Cmake settings configuration and, before building the SDK, change the current configuration to the release created previously.



You can generate the cache and build the complete project (release) by right-clicking in CMakeLists.txt and selecting “Generate Cache” and then “Build.”





```
Output
Show output from: Build
[75/79] C:\PROGRA~2\MICROS~2\2019\COMMUN~1\VC\Tools\MSVC\1424~1.283\bin\HostX64\x64\cl.exe  /nologo -DBUI
[76/79] C:\PROGRA~2\MICROS~2\2019\COMMUN~1\VC\Tools\MSVC\1424~1.283\bin\HostX64\x64\cl.exe  /nologo -DBUI
[77/79] cmd.exe /C "cd . && C:\PROGRA~2\MICROS~2\2019\COMMUN~1\VC\Tools\MSVC\1424~1.283\bin\Hostx64\x64\l
[78/79] C:\PROGRA~2\MICROS~2\2019\COMMUN~1\VC\Tools\MSVC\1424~1.283\bin\HostX64\x64\cl.exe  /nologo /TP -
[79/79] cmd.exe /C "cd . && "C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE\Co

Build succeeded.
```

The release can be found in .\LeddarSDK-master\src\out\build\x64-Release or in the folder selected in the configuration phase.

If you are using Python 3.9, you may get the following error:

```
C4996  'PyEval_ThreadsInitialized': deprecated in 3.9   C:\Users\user464\Documents\Dev\SDK-2\LeddarSDK-master\src\
C4996  'PyEval_InitThreads': deprecated in 3.9          C:\Users\user464\Documents\Dev\SDK-2\LeddarSDK-master\src\
```

You can correct the error by commenting the following lines in LeddarPy.cpp:

```
LeddarPy.cpp   LeddarExample.cpp   CMake Overview Pages   LdPropertyIds.h
LeddarPy.dll - x64-Debug (default)   (Global Scope)
353   PyMODINIT_FUNC PyInit_leddar( void )
354   #else
355   #define CREATE_MODULE Py_InitModule3( "leddar", leddar_Methods, "Python wrapper for LeddarTech SDK." );
356   #define RETURN_VALUE(v) ;
357   PyMODINIT_FUNC initleddar( void )
358   #endif
359   {
360       //if( !PyEval_ThreadsInitialized() )
361       //{
362       //    PyEval_InitThreads();
363       //}
364
365       Py_Initialize();
366
367       PyObject *lModule = CREATE_MODULE;
```

The release can be found in .\LeddarSDK-master\src\out\build\x64-Debug.

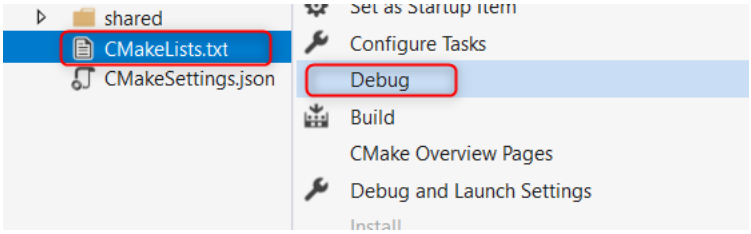
opt > LeddarSDK-master > src > out > build > x64-Debug

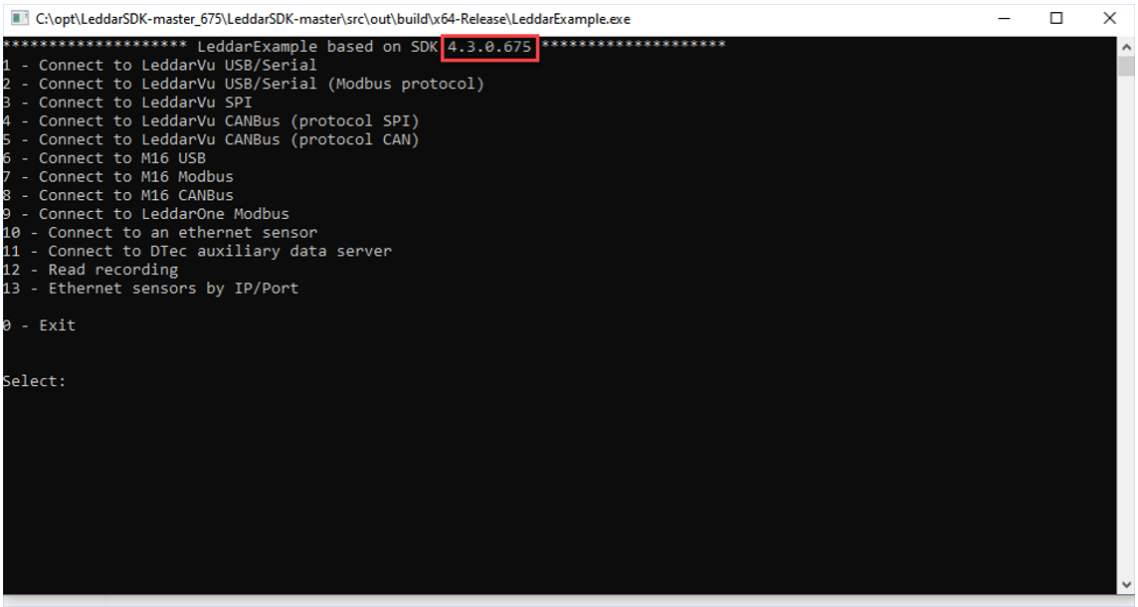
Name	Date modified	Type	Size
.cmake	2021-01-25 10:36 AM	File folder	
CMakeFiles	2021-01-25 10:45 AM	File folder	
Testing	2021-01-25 10:36 AM	File folder	
.ninja_deps	2021-01-25 10:48 AM	NINJA_DEPS File	18 KB
.ninja_log	2021-01-25 10:48 AM	NINJA_LOG File	8 KB
build.ninja	2021-01-25 10:45 AM	NINJA File	83 KB
cmake_install.cmake	2021-01-25 10:36 AM	CMAKE File	2 KB
CMakeCache.txt	2021-01-25 10:36 AM	Text Document	16 KB
LC4.lib	2021-01-25 10:48 AM	Object File Library	30,047 KB
LeddarExample.exe	2021-01-25 10:48 AM	Application	2,184 KB
LeddarExample.ilk	2021-01-25 10:48 AM	Incremental Linker File	12,342 KB
LeddarExample.pdb	2021-01-25 10:48 AM	Program Debug Data...	6,068 KB
rules.ninja	2021-01-25 10:45 AM	NINJA File	4 KB
VSInheritEnvironments.txt	2021-01-25 10:45 AM	Text Document	1 KB

1.1.1.1. Running Windows C++ Package

You can run LeddarExampleModule project by selecting CMakeLists.txt and starting the debug; otherwise, run the **LeddarExampleModule.exe** build after the compilation in the **release** folder.

```
.\LeddarSDK-master\src\out\build\x64-Debug\LeddarExample.exe
```





A simple example to connect and read data from the sensor is present in the SDK.

```
.\LeddarSDK\src\LeddarExample\LeddarExample.cpp
```

You can use the example as a reference for integration into your project.

Running a Leddar™ T16 sensor with C++

To run a Leddar T16 sensor through Ethernet connection, some small modifications to the code are required to **visualize** properly before compiling the SDK.

Notes: *The SDK on the Leddar T16 allows basic communication with the sensor as echoes retrieval and configuration setting (GET and SET). However, the T16 Traffic version does not support the camera management nor the motor control (Pan and Tilt) options.*

*Due to the T16's frame rate, the data throughput cannot be viewed properly in the command prompt. Therefore, the detection values when using **LeddarExample.cpp** "seem" to remain constant and not to vary according to the environmental change, e.g., moving the sensor. This behavior is correct considering that the processing speed of an application layer is much higher than the visualization in the command prompt.*

*For a user-friendly visualization in the Windows command prompt, a few changes in **LeddarExample.cpp** are suggested below.*

Add the lines highlighted in yellow and leave the rest of the code as is.

1. In the `Function: displayEchoes`, add a loop to show 1 out of 100 detections.

```
void displayEchoes( const std::string &aSensorName, LeddarDevice::LdSensor *aSensor )
{
   // Retrieves echoes from the device
   bool lNewData = aSensor->GetData();
   //-----------------------------------------------------------------
   //T16 modification to refresh the command prompt in Windows
   static int T16_count = 0;
   if (++T16_count < 100)
      return;
   T16_count = 0;
   //-----------------------------------------------------------------

   static int count = 0;
   LeddarDevice::LdSensorPixell *lPix = dynamic_cast<LeddarDevice::LdSensorPixell *>( aSensor );
   LeddarDevice::LdSensorDTec *lDTec = dynamic_cast<LeddarDevice::LdSensorDTec *>( aSensor );
```

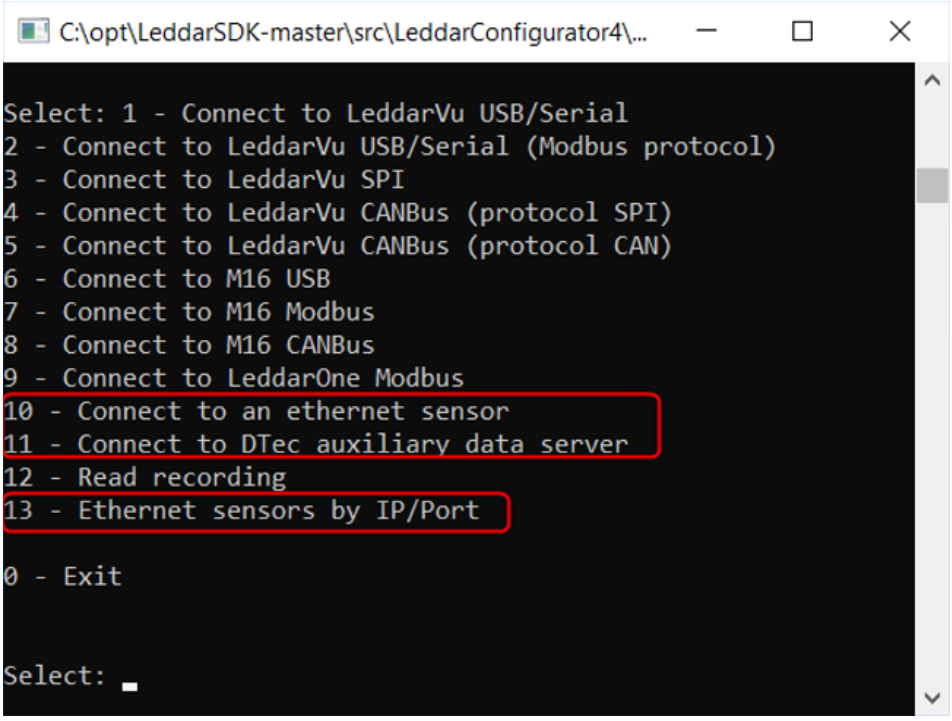
2. In the `Callback Class: LdDisplayer`, add a loop to show 1 out of 100 detections.

```
      else if( aSender == mEchoes ) {
         //-----------------------------------------------------------------
         //T16 modification to refresh the command prompt in Windows
         static int T16_count = 0;
         if (++T16_count < 100)
            return;
         T16_count = 0;
         //-----------------------------------------------------------------
         mEchoes->Lock( LeddarConnection::B_GET );
 std::cout << "Channel\tDistance\tAmplitude - Count = " << mEchoes->GetEchoCount() << " @ " << mEchoes-
>GetTimestamp() << std::endl;
 std::vector<LeddarConnection::LdEcho> &lEchoes = *( mEchoes->GetEchoes() );
```

3. In the `Function: connectedMenu`, modify the wait value, between displayEchoes request:

```
//-----------------------------------------------------------------
      //T16 modification to refresh the command promt in Windows
      LeddarUtils::LtTimeUtils::Wait( 1 );
//-----------------------------------------------------------------
```

Running LeddarExample.cpp, it is possible to connect to the T16 sensor using any of the routines highlighted below.



1.1.2. Windows Python Package

Notes: ***The SDK in Windows is compatible only with Python versions > 3.8****. Be sure you use the correct Python environment during compilation and building.*

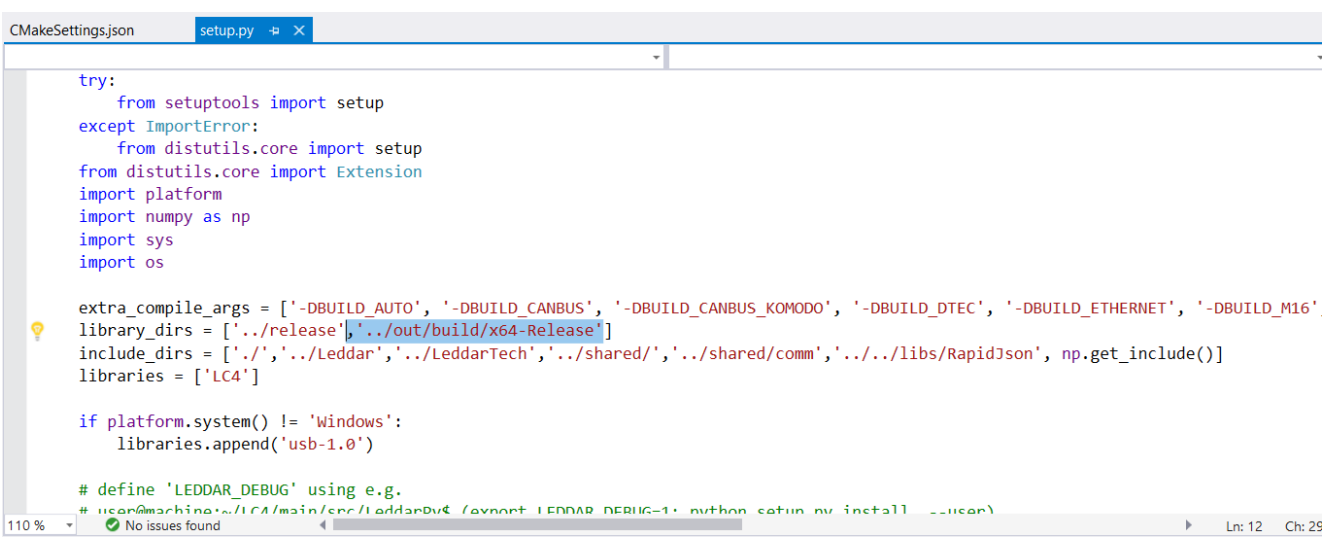
ROS NOETIC for Windows uses Python 3.8.3. *Therefore, you need to compile LeddarPy in the same version of Python to work with Leddar in ROS for Windows.*

If not done previously, download the latest version of Python from https://www.python.org/downloads/

Before building the Python SDK, add the **release** folder information to the setup.py.

By including to the library_dirs the **release** folder and saving the file.

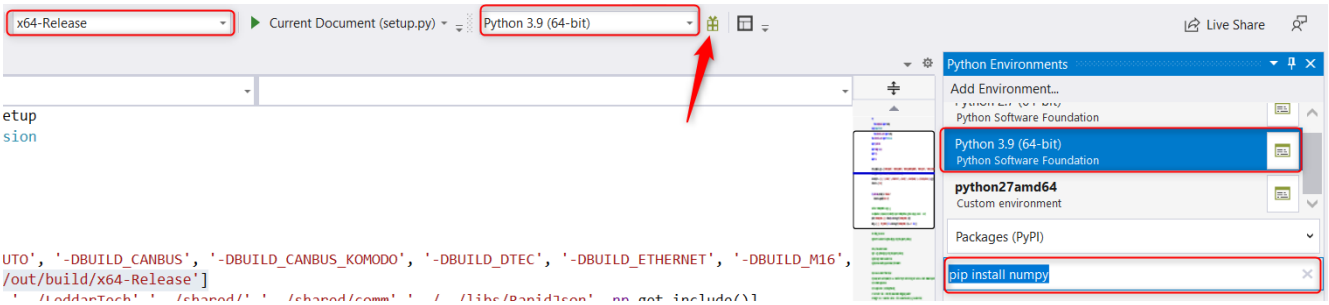
```
library_dirs = ['../release','../out/build/x64-Release']
```



Python SDK uses NumPy libraries; therefore, they need to be installed.

To do so, you can open the Python environments and install NumPy using the command:

```
pip install numpy
```

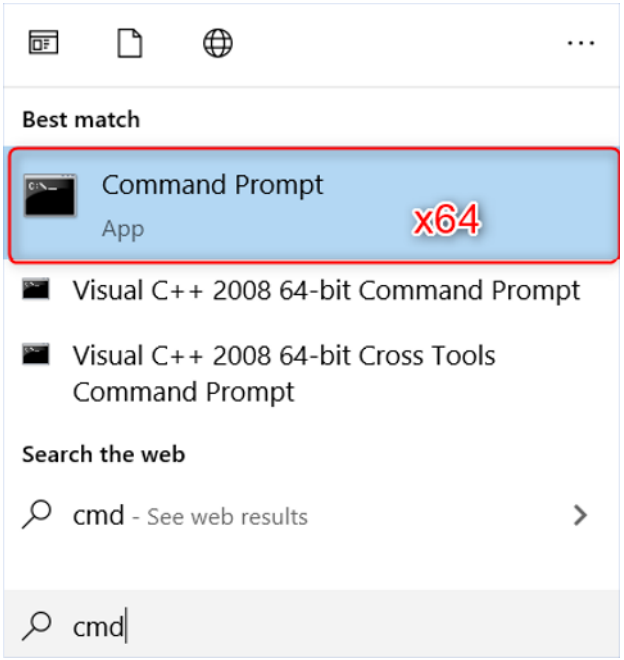


To build the complete Python solution, run a Windows command prompt and navigate to:

```
.\LeddarSDK-master\src\LeddarPy
```

Note: *Be sure you run a command prompt with the same x86 or x64 architecture as the previously compiled SDK.*

In our case, x64:



Then run:

```
cd C:\opt\LeddarSDK-master\src\LeddarPy
py setup.py install --user
```

If you have multiple instances of Python installed, you can use the following to specify the exact version and architecture:

```
py -3.9-64 setup.py install --user
```

Note that the Python egg will be installed in a different path depending on the use of the –user command:

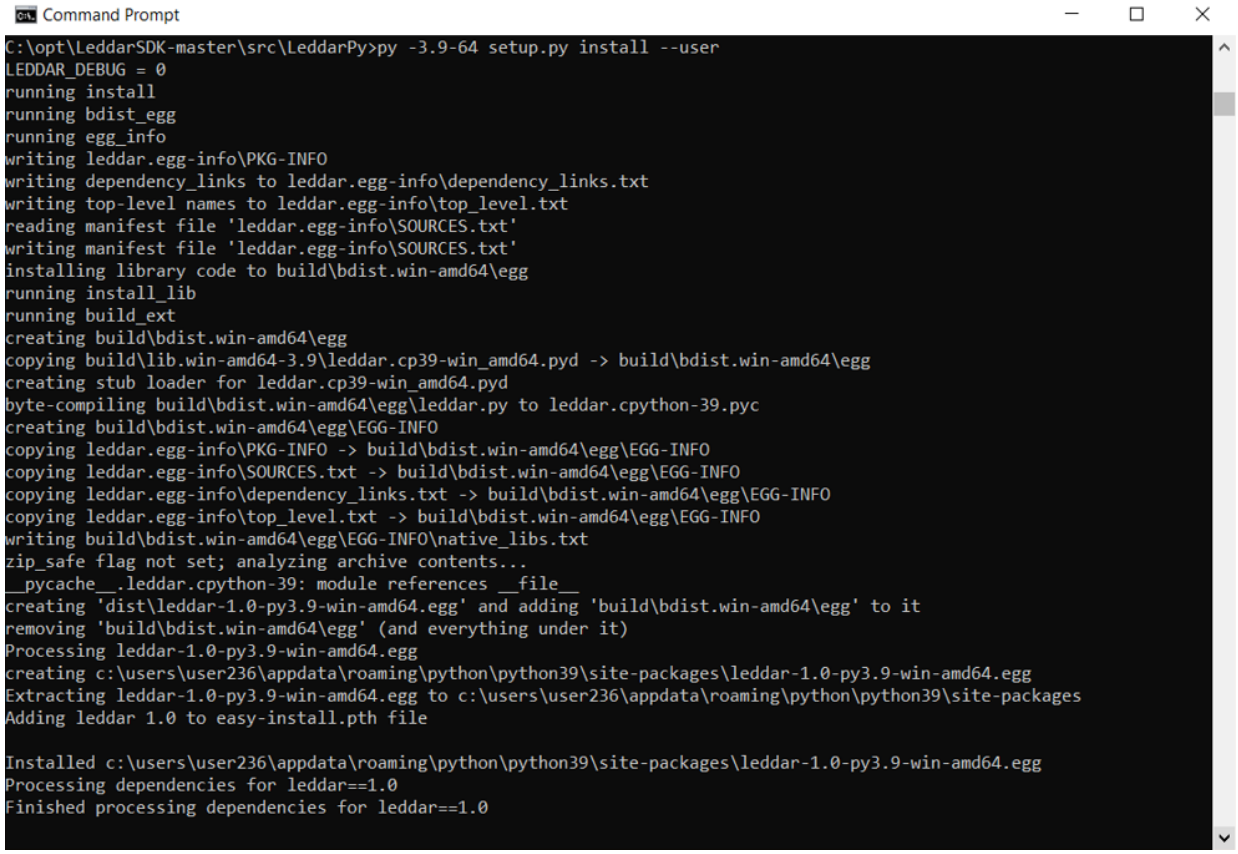
```
py setup.py install
```

"c:\users\user236\appdata**local**\programs\python\python39\lib\site-packages\leddar-1.0-py3.9-win-amd64.egg"

OR:

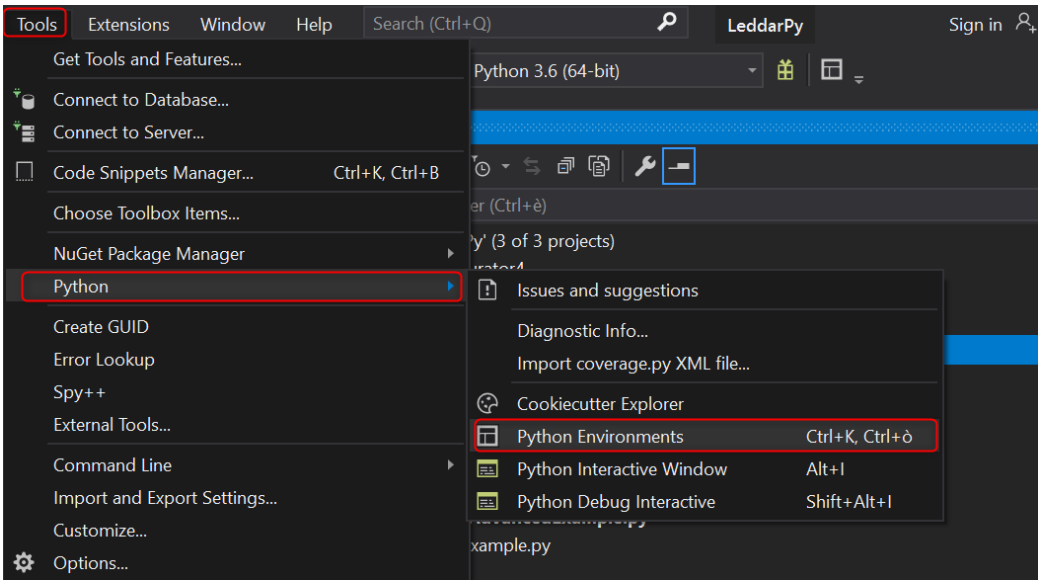
```
py setup.py install --user
```

“c:\users\user236\appdata**roaming**\python\python39\site-packages\leddar-1.0-py3.9-win-amd64.egg”



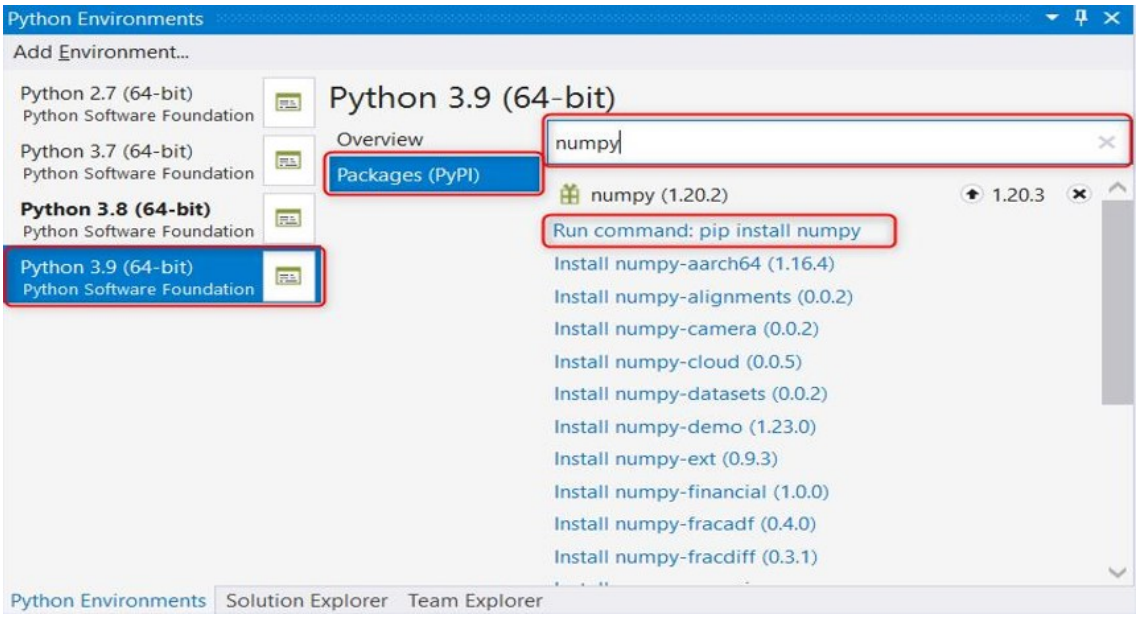
If you get an error in the compilation, you might need to install NumPy libraries.

To do so, navigate to the Python environment in Visual Studio:



Then select the Python version you are using (in this case, **Python 3.9 (64-bit) – Packages**) and search for numpy.

Then click **Run command: pip install numpy**.



1.1.2.1. Running Windows Python Package

To run LeddarPyExample or LeddarPyAdvancedExample in Visual Studio 2019, you need to:

1. Modify LeddarPyExample or LeddarPyAdvancedExample connection parameters according to the sensor to use.
2. Set up the correct Python environment.

1.1.2.2. Modifying LeddarPyExample or LeddarPyAdvancedExample Connection Parameters

In `.\LeddarSDK-master\src\LeddarPy\`, you will find LeddarPyExample.py and LeddarPyAdvancedExample.py

By opening it with your favorite text editor or in Visual Studio, you will find in the code specification about the connection method.

Select the correct connection method according to your sensor and communication parameters.

```
##Use one of this connection method

#Ethernet
# sensor_list = leddar.get_devices("Ethernet")
# print(leddar.device_types["Ethernet"])
# dev.connect('192.168.0.2', leddar.device_types["Ethernet"], 48630)
# connection = dev.connect('192.168.0.2', leddar.device_types["Ethernet"], 48630)

#M16 Usb
# sensor_list = leddar.get_devices("Usb")
# dev.connect(sensor_list[0]['name'], leddar.device_types["Usb"])

##For any sensors with modbus serial communication (LeddarOne, Vu8 or M16)
# sensor_list = leddar.get_devices("Serial")
# dev.connect(sensor_list[0]['name'], leddar.device_types["Serial"])

##For Vu8/M16 sensors with CAN BUS communication
# dev.connect(Baud rate (kbps), Type of sensor, Tx ID (optionnal), Rx ID (optionnal))
```

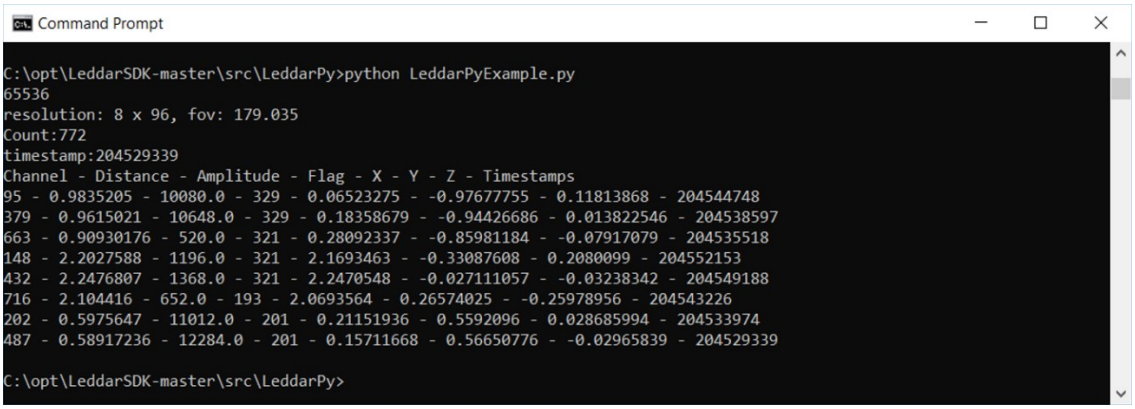
```
# dev.connect('1000', leddar.device_types["M16Komodo"])
# dev.connect('1000', leddar.device_types["Vu8Komodo"])
```

Running a Leddar Pixell with Python

To run a Leddar Pixell through Ethernet connection, comment the other connection methods, leaving only Ethernet enabled and change the IP parameter according to the sensor IP. The port must always be set at 48630.

```
#Ethernet
sensor_list = leddar.get_devices("Ethernet")
print(leddar.device_types["Ethernet"])
connection = dev.connect('192.168.0.2', leddar.device_types["Ethernet"], 48630)
```

You can run the example if the Python environment is properly set.



Running a Leddar M16 through USB with Python

Tu run a Leddar M16 through USB connection, comment the other connection methods, leaving only M16 USB enabled.

```
#M16 Usb
sensor_list = leddar.get_devices("Usb")
dev.connect(sensor_list[0]['name'], leddar.device_types["Usb"])
```

You can run the example if the Python environment is properly set.

```
Command Prompt
C:\opt\LeddarSDK-master\src\LeddarPy>python LeddarPyExample.py
resolution: 1 x 16, fov: 48.0
Count:19
timestamp:71735
Channel - Distance - Amplitude - Flag - X - Y - Z - Timestamps
0 - 2.1024628 - 515.8306 - 9 - 1.9424223 - 0.80457765 - 0.0 - 0
1 - 2.002411 - 529.9684 - 9 - 1.8875556 - 0.66841847 - 0.0 - 0
2 - 1.966095 - 538.0653 - 9 - 1.8851306 - 0.55840117 - 0.0 - 0
3 - 1.9485321 - 539.8217 - 9 - 1.894694 - 0.4548758 - 0.0 - 0
4 - 1.9226074 - 539.4639 - 9 - 1.8904132 - 0.35036737 - 0.0 - 0
5 - 1.8835449 - 538.0569 - 9 - 1.8674309 - 0.24585195 - 0.0 - 0
6 - 1.8999634 - 539.1158 - 9 - 1.8941064 - 0.14906941 - 0.0 - 0
7 - 1.9021301 - 535.8698 - 9 - 1.9014783 - 0.049791962 - 0.0 - 0
8 - 1.9139252 - 539.01044 - 9 - 1.9132693 - -0.05010072 - 0.0 - 0
9 - 1.9105988 - 540.744 - 9 - 1.904709 - -0.14990385 - 0.0 - 0
10 - 1.8931732 - 540.46515 - 9 - 1.8769768 - -0.2471087 - 0.0 - 0
11 - 1.5547943 - 569.2357 - 9 - 1.5287591 - -0.28333876 - 0.0 - 0
12 - 1.6832123 - 276.03186 - 1 - 1.636705 - -0.3929381 - 0.0 - 0
13 - 0.9228058 - 101.77336 - 1 - 0.8848044 - -0.262091 - 0.0 - 0
13 - 6.181183 - 7.8982697 - 64 - 5.92664 - -1.7555507 - 0.0 - 0
14 - 0.9254303 - 39.32258 - 1 - 0.872349 - -0.308915 - 0.0 - 0
14 - 5.66626 - 7.388626 - 64 - 5.3412514 - -1.8914363 - 0.0 - 0
15 - 0.83966064 - 23.688744 - 1 - 0.7757453 - -0.32132423 - 0.0 - 0
15 - 5.530777 - 11.714348 - 64 - 5.1097717 - -2.1165366 - 0.0 - 0

C:\opt\LeddarSDK-master\src\LeddarPy>
```

Running a Leddar M16, Vu8, or LeddarOne Modbus with Python

To run a LeddarOne, Vu8 or M16 sensor with Modbus connection, comment the other connection methods, leaving only Modbus serial communication enabled, as it is possible to have multiple COM ports available. To select the correct one, try to modify the code as follows:

```
##For any sensors with modbus serial communication (LeddarOne, Vu8 or M16)
sensor_list = leddar.get_devices("Serial")
for port in sensor_list:
    print("COM ports avaliable: ", port)
dev.connect(sensor_list[1]['name'], leddar.device_types["Serial"])
```

The item highlighted in yellow must correspond to the COM the sensor is connected to. In our case:

```
sensor_list[0] → Point to COM3
sensor_list[1] → Point to COM11 (where the sensor is connected)
```

If the Python environment is properly set, you can run the example.

```
Command Prompt

C:\opt\LeddarSDK-master\src\LeddarPy>python LeddarPyExample.py
COM ports avaliable:  {'name': '\\\\.\\COM3', 'type': 2, 'address': '\\\\.\\COM3'}
COM ports avaliable:  {'name': '\\\\.\\COM11', 'type': 2, 'address': '\\\\.\\COM11'}
resolution: 1 x 8, fov: 45.0
Count:7
timestamp:1952365
Channel - Distance - Amplitude - Flag - X - Y - Z - Timestamps
7 - 2.1678467 - 121.07143 - 1 - 2.0411232 - -0.7303256 - 0.0 - 0
6 - 2.0451355 - 171.16179 - 1 - 1.9838454 - -0.49692738 - 0.0 - 0
5 - 2.0409393 - 194.2578 - 1 - 2.0188491 - -0.299468 - 0.0 - 0
4 - 2.0401459 - 200.94426 - 1 - 2.0376885 - -0.10010521 - 0.0 - 0
3 - 2.0255127 - 187.65327 - 1 - 2.023073 - 0.0993872 - 0.0 - 0
2 - 2.0070343 - 194.06447 - 1 - 1.9853112 - 0.2944931 - 0.0 - 0
1 - 1.9000702 - 28.018726 - 64 - 1.8431275 - 0.4616794 - 0.0 - 0

C:\opt\LeddarSDK-master\src\LeddarPy>
```

Running a Leddar T16 with Python

To run a Leddar T16 sensor through Ethernet connection, comment the other connection methods, leaving only Ethernet enabled and change the IP parameter according to the sensor IP. The port must always be set at 48620.

```
#Ethernet
sensor_list = leddar.get_devices("Ethernet")
print(leddar.device_types["Ethernet"])
dev.connect('192.168.0.2', leddar.device_types["Ethernet"], 48620)
connection = dev.connect('192.168.0.2', leddar.device_types["Ethernet"], 48620)
```

You can run the example if the Python environment is properly set:

```
LeddarPyExample.py
```

```
Command Prompt

C:\opt\LeddarSDK-master\src\LeddarPy>python LeddarPyExample.py
65536
resolution: 1 x 16, fov: 36.4
Count:16
timestamp:31064
Channel - Distance - Amplitude - Flag - X - Y - Z - Timestamps
0 - 1.6959229 - 125.525635 - 9 - 1.6212772 - 0.49760872 - 0.0 - 0
1 - 1.3292389 - 165.00491 - 9 - 1.2852134 - 0.33926806 - 0.0 - 0
2 - 1.3028107 - 178.16006 - 9 - 1.2718673 - 0.28225735 - 0.0 - 0
3 - 1.2259521 - 174.04199 - 9 - 1.2064344 - 0.21788709 - 0.0 - 0
4 - 1.187088 - 126.571045 - 9 - 1.1756432 - 0.1644413 - 0.0 - 0
5 - 1.1983643 - 126.19562 - 9 - 1.192465 - 0.11876109 - 0.0 - 0
6 - 1.2003632 - 126.286156 - 9 - 1.1982348 - 0.071450606 - 0.0 - 0
7 - 1.2214355 - 126.17817 - 9 - 1.2211949 - 0.024247715 - 0.0 - 0
8 - 1.1994019 - 126.23457 - 9 - 1.1991655 - -0.023810307 - 0.0 - 0
9 - 1.2239227 - 124.68193 - 9 - 1.2217525 - -0.07285297 - 0.0 - 0
10 - 1.3440399 - 125.6788 - 9 - 1.3374234 - -0.13319793 - 0.0 - 0
11 - 1.4406738 - 126.13734 - 9 - 1.4267842 - -0.19956927 - 0.0 - 0
12 - 1.457962 - 141.78345 - 9 - 1.4347506 - -0.25912192 - 0.0 - 0
13 - 1.4758148 - 175.17027 - 9 - 1.4407624 - -0.31973913 - 0.0 - 0
14 - 1.5309296 - 155.92041 - 9 - 1.4802238 - -0.39074656 - 0.0 - 0
15 - 1.670517 - 125.60435 - 9 - 1.5969896 - -0.49015424 - 0.0 - 0
```

Notes: *The SDK on the Leddar T16 allows basic communication with the sensor as echoes retrieval and configuration setting (GET and SET). However, the T16 Traffic version does neither support the camera management nor the motor control (Pan and Tilt) options.*

Due to the T16's frame rate, the data throughput cannot be viewed properly in the command prompt. Therefore, the detection values when using ***LeddarPyAdvancedExample.py*** *"seem" to remain constant and not to vary according to the environmental change, e.g., moving the sensor. This behavior is correct as the processing speed of an application layer is much higher than the visualization in the command prompt.*

For a user-friendly visualization in the Windows command prompt, a few changes in ***LeddarPyAdvancedExample.py*** *are suggested below.*

1. Add a global variable and implement a loop to show 1 out of 100 detections.

```
import leddar
import time

global T16_count
T16_count = 0

leddar.enable_debug_trace(True)

#Callback functions for the data thread
def echoes_callback(echoes):
    data = echoes["data"]

    #To avoid display all the ECHOES
    global T16_count
    if (T16_count < 100) :
        #print("Count:" + str(T16_count))
        T16_count = T16_count + 1
        return
    T16_count = 0

   #To avoid display of too much lines
    increment = 1
    if len(data) > 100 :
        increment = 100

    print("Count:" + str(len(data)))
```

2. Modify the delay between two requests to be a small value:

```
#Optional : set the delay between two request to the sensor
dev.set_data_thread_delay(1)
```

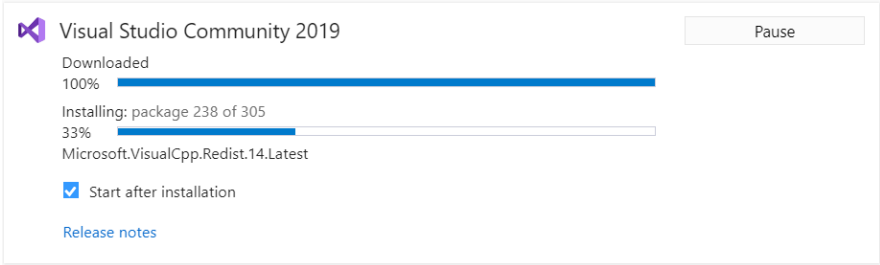
You can find additional information in section "1.1.2.2 Modifying LeddarPyExample or LeddarPyAdvancedExample Connection Parameters."

2. Installing ROS for Windows

Go to https://wiki.ros.org/Installation/Windows.

Follow the required steps.

In step 3, remember to include “Desktop development with C++” workload.



Steps 5.2 and 6.1 are optional and are not required for working with Leddar sensors.

After completing step 7, close and reopen the ROS command prompt.

You are now ready to start using ROS.

You can check this by opening as Administrator (step 6) and running:

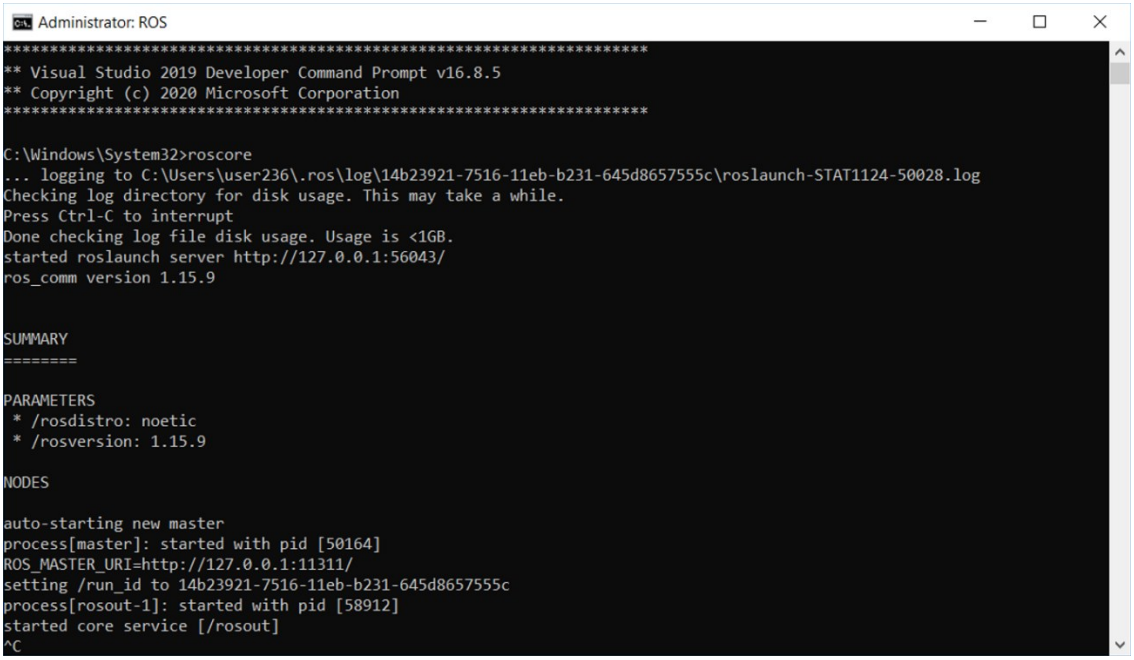
```
roscore
```

If you get errors regarding any missing dll, reinstall ROS by running in the ROS command prompt:

```
choco install ros-noetic-desktop_full -y --execution-timeout=0 --force
```

Then relaunch:

```
roscore
```



3. Running Leddar Sensors in ROS for Windows

You will find the ROS driver for Leddar sensors in LeddarSDK-master. Navigate to .\LeddarSDK-master\src\Leddar_Ros

Note: ***ROS NOETIC for Windows uses Python 3.8.3.*** *Therefore, you need to compile LeddarPy in the same version of Python to work with Leddar sensors in ROS for Windows.*

3.1. Installing LeddarPy and NumPy in ROS

Note: *It is mandatory to update LeddarPy as follows to use the latest Leddar_ROS driver.*

Using the ROS command prompt

Update NumPy by running:

```
py -m pip install --upgrade numpy
py -3.8 -m pip install --upgrade numpy
```

Then, check where Python is installed by running:

```
where python
python --version
```

Python in the Windows directory and ROS related directory will appear:

```
c:\opt\catkin_ws>python --version
Python 3.8.3

c:\opt\catkin_ws>where python
C:\opt\ros\noetic\x64\python.exe
```

Typically:

```
C:\opt\ros\noetic\x64\python.exe
```

Using the ROS command prompt

3.1.1. Building LeddarPy

Navigate to the **LeddarPy** folder by running:

```
cd C:\opt\LeddarSDK-master\src\LeddarPy
```

Build it in Python 3.8.3 by running:

```
py -3.8-64 setup.py build
```

The build library will be created in “LeddarSDK-master\src\LeddarPy\build\lib.win-amd64-3.8”

Copy the **leddar.cp38-win_amd64.pyd** file into C:\opt\ros\noetic\x64.

```
cd C:\opt\LeddarSDK-master\src\LeddarPy
copy build\lib.win-amd64-3.8\leddar.cp38-win_amd64.pyd C:\opt\ros\noetic\x64
```

3.2. Preparing the catkin Workspace

Using File Explorer, create a new folder under c:\opt\

```
catkin_ws
```

Create a sub-folder inside catkin_ws

```
src
```

Alternatively, using the ROS command prompt, run:

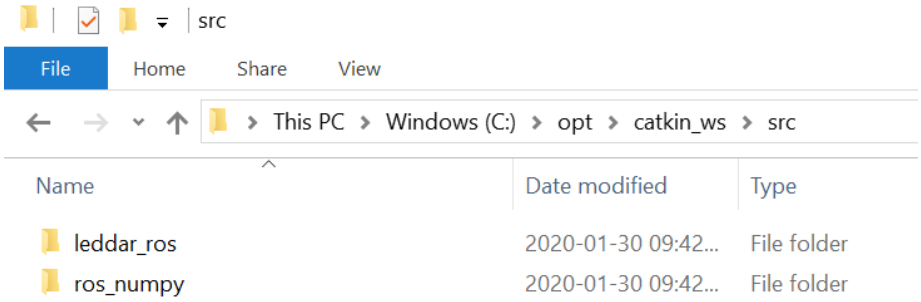
```
cd c:\opt\
mkdir catkin_ws
cd catkin_ws
mkdir src
```

Open the ROS command prompt and run:

```
cd c:\opt\catkin_ws\src
catkin_init_workspace
```

With Windows Explorer, copy from any other catkin workspace into the **src** folder:

```
leddar_ros
ros_numpy
```



Using the ROS command prompt, run:

```
cd c:\opt\catkin_ws
catkin_make
```

```
Administrator: ROS
/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.28.293
33/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Using CATKIN_DEVEL_PREFIX: C:/opt/catkin_ws/devel
-- Using CMAKE_PREFIX_PATH: C:\opt\ros\noetic\x64\tools\vcpkg\installed\x64-windows;c:\opt\ros\noetic\x64
-- This workspace overlays: c:/opt/ros/noetic/x64
-- Found PythonInterp: C:/opt/ros/noetic/x64/python.exe (found suitable version "3.8.3", minimum required is "3")
-- Using PYTHON_EXECUTABLE: C:/opt/ros/noetic/x64/python.exe
-- Using default Python package layout
-- Found PY_em: C:\opt\ros\noetic\x64\lib\site-packages\em.py
-- Using empy: C:/opt/ros/noetic/x64/lib/site-packages/em.py
-- Using CATKIN_ENABLE_TESTING: ON
-- Call enable_testing()
-- Using CATKIN_TEST_RESULTS_DIR: C:/opt/catkin_ws/build/test_results
-- Found gtest: gtests will be built
-- Using Python nosetests: C:/opt/ros/noetic/x64/Scripts/nosetests.exe
-- catkin 0.8.9
-- BUILD_SHARED_LIBS is on
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 2 packages in topological order:
-- ~~  - leddar_ros
-- ~~  - ros_numpy
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'leddar_ros'
-- ==> add_subdirectory(leddar_ros)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- leddar_ros: 1 messages, 0 services
-- +++ processing catkin package: 'ros_numpy'
-- ==> add_subdirectory(ros_numpy)
-- Configuring done
-- Generating done
-- Build files have been written to: C:/opt/catkin_ws/build
####
#### Running command: "ninja -j8 -l8" in "c:\opt\catkin_ws\build"
####
[10/10] Linking CXX executable catkin_generated\windows_wr...Project__setup_util.py_exec_install_python\_setup_util.exe

c:\opt\catkin_ws>
```

Close and re-open your ROS command prompt.

Update the catkin environment by running:

```
c:\opt\catkin_ws\devel\setup.bat
cd c:\opt\catkin_ws
catkin_make
```

```
Administrator: ROS
**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.8.5
** Copyright (c) 2020 Microsoft Corporation
**********************************************************************

C:\Windows\System32>c:\opt\catkin_ws\devel\setup.bat

C:\Windows\System32>cd c:\opt\catkin_ws

c:\opt\catkin_ws>catkin_make
Base path: c:\opt\catkin_ws
Source space: c:\opt\catkin_ws\src
Build space: c:\opt\catkin_ws\build
Devel space: c:\opt\catkin_ws\devel
Install space: c:\opt\catkin_ws\install
####
#### Running command: "ninja build.ninja" in "c:\opt\catkin_ws\build"
####
ninja: no work to do.
####
#### Running command: "ninja -j8 -l8" in "c:\opt\catkin_ws\build"
####
[1/1] cmd.exe /C "cd /D C:\opt\catkin_ws\build\leddar_ros ...leddar_ros C:/opt/catkin_ws/src/leddar_ros/msg/Specs.msg "

c:\opt\catkin_ws>
```

3.3. Running Leddar Pixell With ROS for Windows

Update the catkin environment using the ROS command prompt:

```
c:\opt\catkin_ws\devel\setup.bat
cd c:\opt\catkin_ws
catkin_make
```

Launch Pixell example: To avoid modifying the **.launch** file, it is possible to send all parameters directly in the command line:

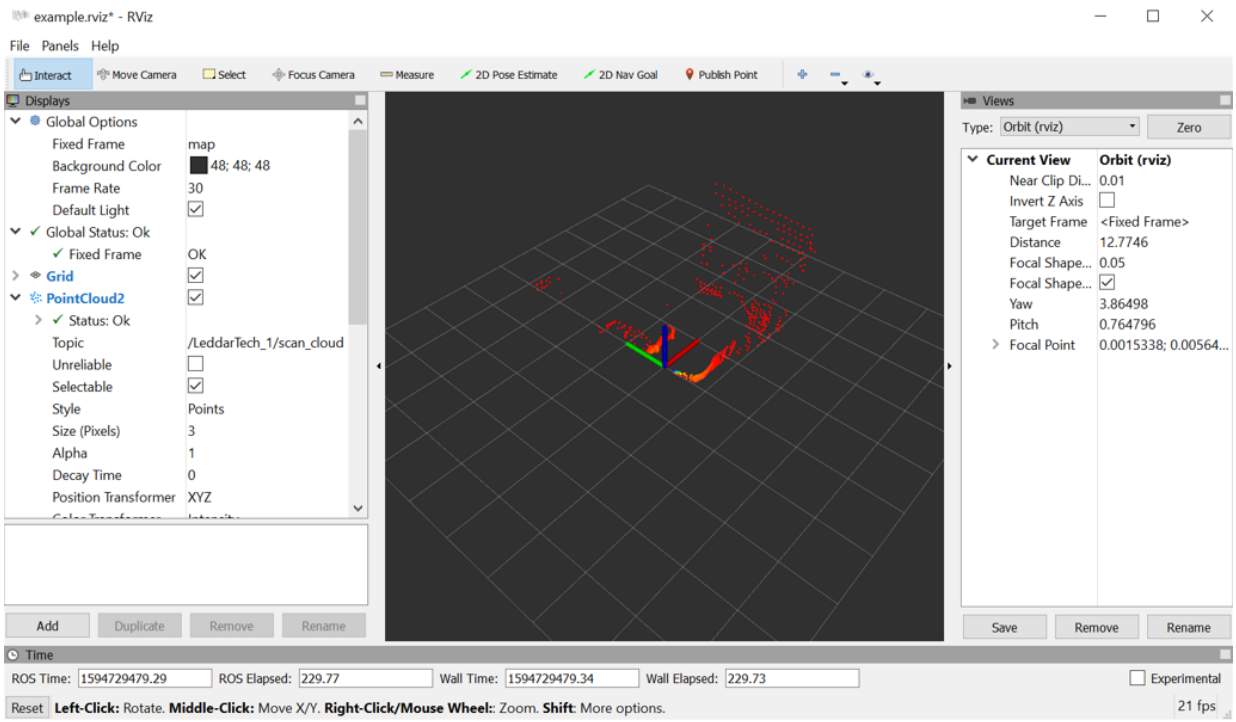
```
roslaunch leddar_ros example.launch param1:=192.168.0.2 device_type:=Ethernet param3:=48630
param4:=0
```

Alternatively, it is possible to modify the example.launch to reflect the required parameters:

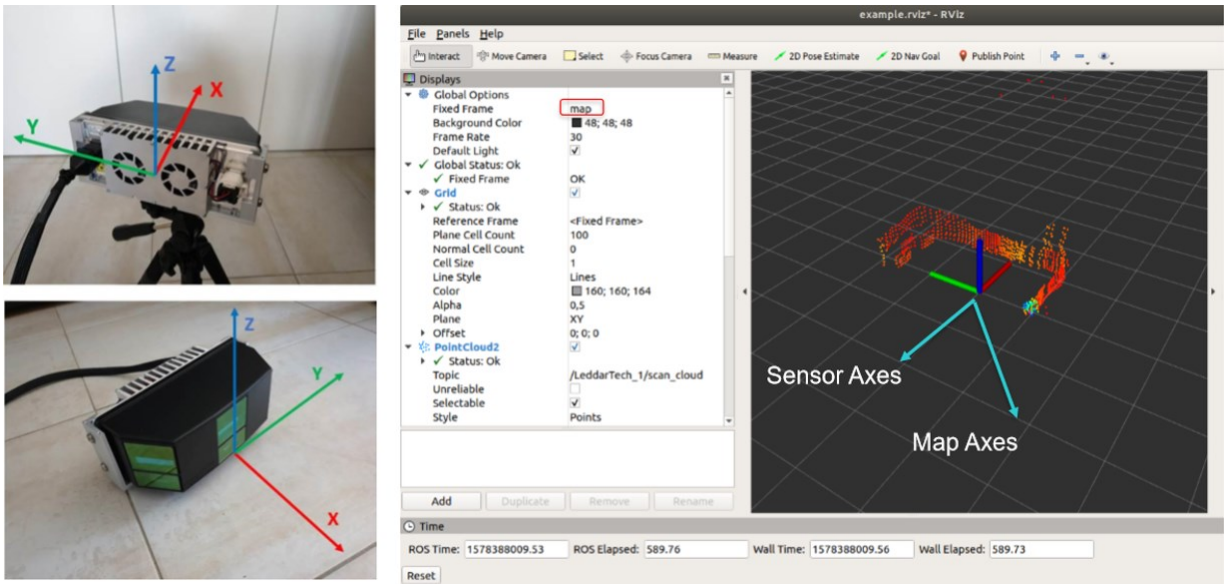
```
<arg name="param1" default="192.168.0.2" />
<arg name="device_type" default="Ethernet" />
<arg name="param3" default="48630"/>
<arg name="param4" default="0"/>
```

Then run the launch file:

```
roslaunch leddar_ros example.launch
```



Below is a diagram of the representation in Rviz according to the sensor orientation:



It is possible to move the sensor axes positions with regards to the map axes position using **device.xml**. See the sections highlighted in blue below.

3.3.1. Leddar Pixell ROS Code Examples

"`leddar_ros\launch\`**`example.launch`**"

```
<launch>
 <!-- Connection info, see python leddar.Device.connect for more info -->
 <arg name="param1" default="192.168.0.2" />
 <!-- [All] value returned by GetDeviceList['name'],
  [Serial] serial port com,
  [USB] serial number,
  [SPI-FTDI] FTDI cable id (use get_devices("SpiFTDI")),
  [CANBus komodo] Baudrate in kbps,
  [Ethernet] Ip address -->
 <arg name="device_type" default="Ethernet" />
 <!-- (optional but recommended) Device type | connection type -
  Both are mandatory for CANBus protocol (from leddar.device_types dictionnary) -->
 <arg name="param3" default="48630"/>
 <!-- (optional) [Serial] modbus address (default 1),
  [CANBus komodo] Tx (default 0x750) ,
    [Ethernet] port (default 48630) -->
  <arg name="param4" default="0"/>
  <!-- param4: (optional) [Serial] baudrate (default 115200),
    [CANBus komodo] Rx (default 0x740),
    [Ethernet] Communication timeout -->

  <include file="$(find leddar_ros)/config/device.xml" ns="LeddarTech_1" pass_all_args="true">
  </include>

  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find leddar_ros)/rviz/example.rviz"
output="screen"/>
</launch>
```

“leddar_ros\config**device.xml**”

```
<launch>
 <!-- Connection info, see python leddar.Device.connect for more info -->
 <arg name="param1" default="192.168.0.2" />
 <!-- [All] value returned by GetDeviceList['name'],
  [Serial] serial port com,
  [USB] serial number,
  [SPI-FTDI] FTDI cable id (use get_devices("SpiFTDI")),
  [CANBus komodo] Baudrate in kbps,
  [Ethernet] Ip address -->
 <arg name="device_type" default="Ethernet" />
 <!-- (optional but recommended) Device type | connection type -
  Both are mandatory for CANBus protocol (from leddar.device_types dictionnary) -->
 <arg name="param3" default="48630"/>
 <!-- (optional) [Serial] modbus address (default 1),
  [CANBus komodo] Tx (default 0x750) ,
  [Ethernet] port (default 48630) -->
 <arg name="param4" default="0"/>
 <!-- param4: (optional) [Serial] baudrate (default 115200),
  [CANBus komodo] Rx (default 0x740),
  [Ethernet] Communication timeout -->

 <arg name="frame_id" default="sensor_frame" />
 <arg name="parent_frame_id" default="map" />

 <!-- Position of the Sensor <arg name="position" default="X Y Height" /> -->
 <arg name="position" default="0 0 0" />
 <!-- Euler_angles in ROS are (yaw, pitch, roll) in rad
  rotate about fixed X-axis by Roll
  rotate about fixed Y-axis by Pitch
  rotate about fixed Z-axis by Yaw
 -->
  <arg name="rotation" default="0 0 0" />

 <!-- Quaternion -->
 <!--<arg name="rotation" default="-0.707106781 0 0 0.707106781" />  -->

 <node pkg="leddar_ros" type="device.py" name="sensor">
  <param name="device_type" value="$(arg device_type)" />
  <param name="frame_id" value="$(arg frame_id)" />
  <param name="param1" value="$(arg param1)" />
  <param name="param3" value="$(arg param3)" />
  <param name="param4" value="$(arg param4)" />
 </node>

 <node pkg="tf" type="static_transform_publisher" name="leddar_1_to_map"
    args="$(arg position) $(arg rotation) $(arg parent_frame_id) $(arg frame_id) 10" />

</launch>
```

3.4. Running Leddar Vu8 With ROS for Windows

Update the catkin environment using the ROS command prompt:

```
c:\opt\catkin_ws\devel\setup.bat
cd c:\opt\catkin_ws
catkin_make
```

Launch Vu8 example: To avoid modifying the **.launch** file, it is possible to send all parameters in the command line:

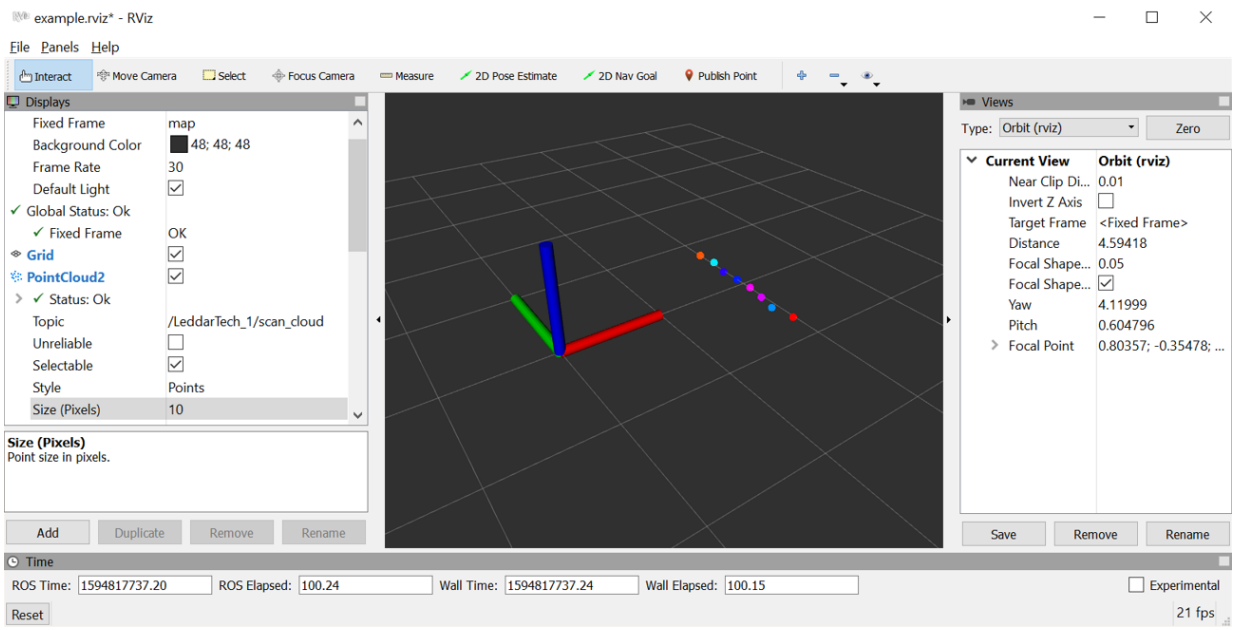
```
roslaunch leddar_ros example.launch param1:=\\.\COM11 device_type:=Serial param3:=1
param4:=0
```

Alternatively, it is possible to modify the example.launch to reflect the required parameters:

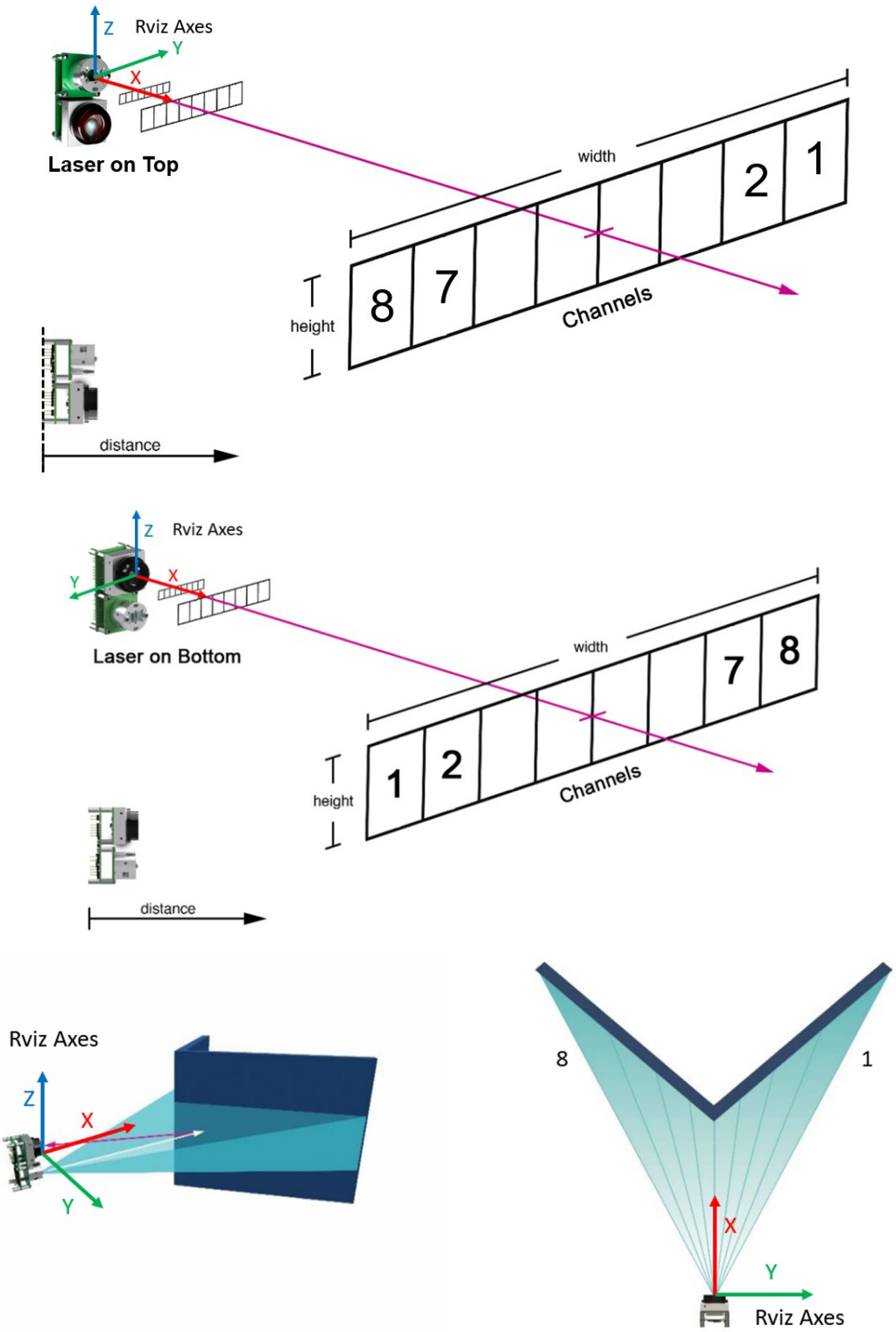
```
<arg name="param1" default="\\.\COM11"/>
<arg name="device_type" default="Serial" />
<arg name="param3" default="1"/>
<arg name="param4" default="0"/>
```

Then run the launch file:

```
roslaunch leddar_ros example.launch
```



Below is a diagram of the representation in Rviz according to the sensor orientation:



It is possible to move the sensor axes positions with regards to the map axes position using **device.xml**. See the sections highlighted in blue below.

3.4.1. Leddar Vu8 ROS Code Examples

"leddar_ros\launch\example.launch"

```
<launch>
  <!-- Connection info, see python leddar.Device.connect for more info -->
  <arg name="param1" default="\\.\COM11" />
  <!-- [All] value returned by GetDeviceList['name'],
    [Serial] serial port com,
    [USB] serial number,
    [SPI-FTDI] FTDI cable id (use get_devices("SpiFTDI")),
    [CANBus komodo] Baudrate in kbps,
    [Ethernet] Ip address -->
  <arg name="device_type" default="Serial" />
  <!-- (optional but recommended) Device type | connection type -
    Both are mandatory for CANBus protocol (from leddar.device_types dictionnary) -->
  <arg name="param3" default="1"/>
  <!-- (optional) [Serial] modbus address (default 1),
    [CANBus komodo] Tx (default 0x750) ,
    [Ethernet] port (default 48630) -->
  <arg name="param4" default="0"/>
  <!-- param4: (optional) [Serial] baudrate (default 115200),
    [CANBus komodo] Rx (default 0x740),
    [Ethernet] Communication timeout -->

  <include file="$(find leddar_ros)/config/device.xml" ns="LeddarTech_1" pass_all_args="true">
  </include>

  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find leddar_ros)/rviz/example.rviz" output="screen"/>
</launch>
```

"leddar_ros\config\device.xml"

```
<launch>
  <!-- Connection info, see python leddar.Device.connect for more info -->
  <arg name="param1" default="\\.\COM11" />
  <!-- [All] value returned by GetDeviceList['name'],
    [Serial] serial port com,
    [USB] serial number,
    [SPI-FTDI] FTDI cable id (use get_devices("SpiFTDI")),
    [CANBus komodo] Baudrate in kbps,
    [Ethernet] Ip address -->
  <arg name="device_type" default="Serial" />
  <!-- (optional but recommended) Device type | connection type -
    Both are mandatory for CANBus protocol (from leddar.device_types dictionnary) -->
  <arg name="param3" default="1"/>
  <!-- (optional) [Serial] modbus address (default 1),
    [CANBus komodo] Tx (default 0x750) ,
    [Ethernet] port (default 48630) -->
  <arg name="param4" default="0"/>
```

```
<!-- param4: (optional) [Serial] baudrate (default 115200),
  [CANBus komodo] Rx (default 0x740),
  [Ethernet] Communication timeout -->

<arg name="frame_id" default="sensor_frame" />
<arg name="parent_frame_id" default="map" />

<!-- Position of the Sensor <arg name="position" default="X Y Height" /> -->
<arg name="position" default="0 0 0" />
<!-- Euler_angles in ROS are (yaw, pitch, roll) in rad
  rotate about fixed X-axis by Roll
  rotate about fixed Y-axis by Pitch
  rotate about fixed Z-axis by Yaw
-->
 <arg name="rotation" default="0 0 0" />

<!-- Quaternion -->
<!--<arg name="rotation" default="-0.707106781 0 0 0.707106781" />  -->

<node pkg="leddar_ros" type="device.py" name="sensor">
  <param name="device_type" value="$(arg device_type)" />
  <param name="frame_id" value="$(arg frame_id)" />
  <param name="param1" value="$(arg param1)" />
  <param name="param3" value="$(arg param3)" />
  <param name="param4" value="$(arg param4)" />
</node>

<node pkg="tf" type="static_transform_publisher" name="leddar_1_to_map"
    args="$(arg position) $(arg rotation) $(arg parent_frame_id) $(arg frame_id) 10" />

</launch>
```

3.5. Running Leddar M16 With ROS for Windows

Use the ROS command prompt update catkin environment:

```
c:\opt\catkin_ws\devel\setup.bat
cd c:\opt\catkin_ws
catkin_make
```

Launch M16 (connected through USB) example: To avoid modifying the **.launch** file, it is possible to send all parameters directly in the command line:

```
roslaunch leddar_ros example.launch param1:=AK47027 device_type:=Usb param3:=1 param4:=0
```

Note: *AK47027 is the unit's serial number.*

For a Leddar **IS16 connected through USB**, the only change is the SN:

```
roslaunch leddar_ros example.launch param1:=AM05002 device_type:=Usb param3:=1 param4:=0
```

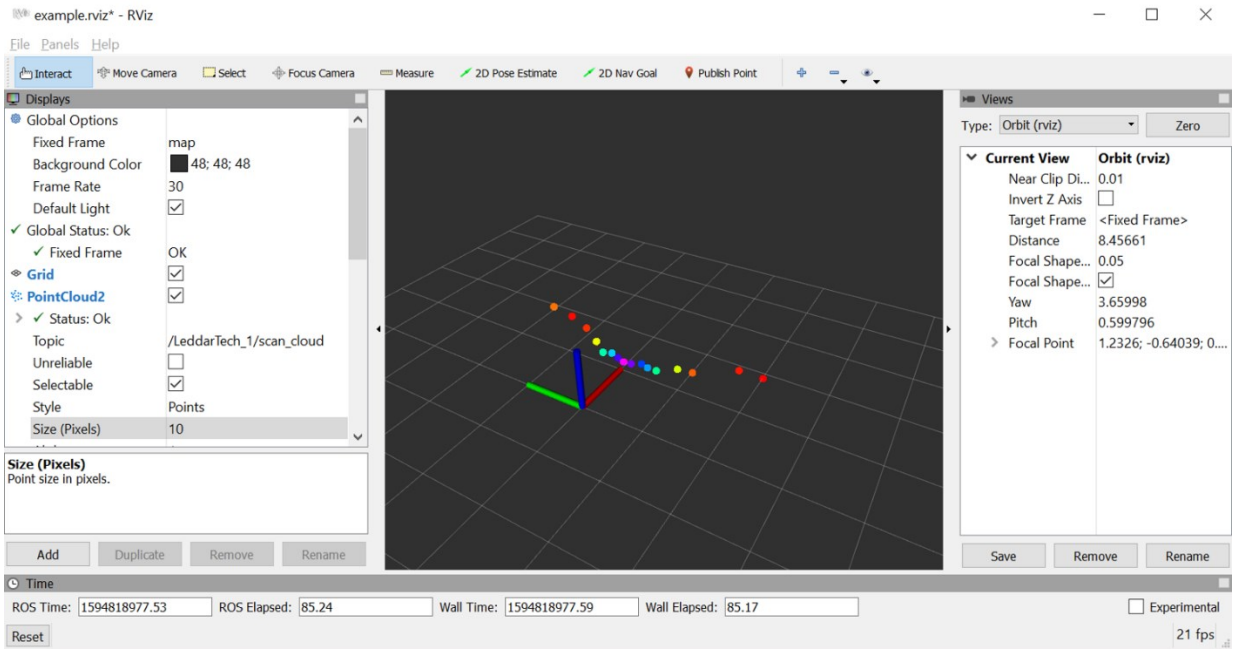
Alternatively, it is possible to modify the example.launch to reflect the required parameters:

```
<arg name="param1" default="AK47027"/>
<arg name="device_type" default="Usb" />
```

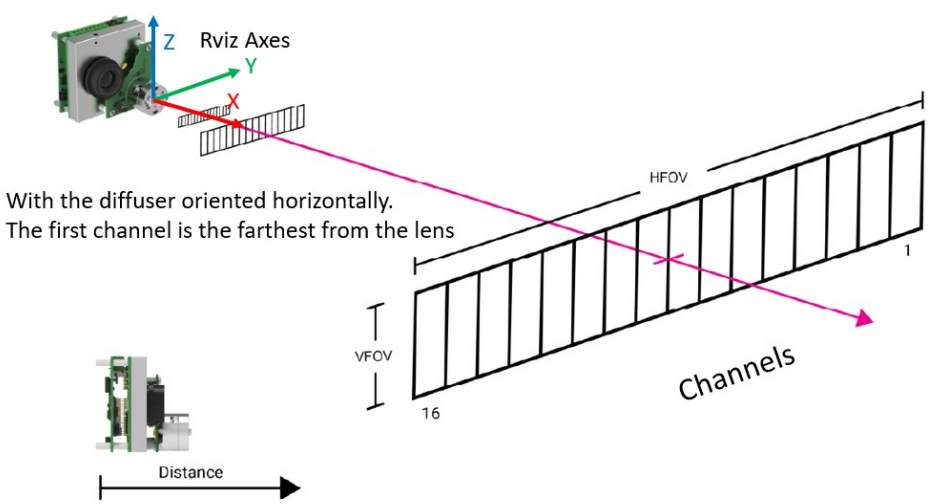
```
<arg name="param3" default="1"/>
<arg name="param4" default="0"/>
```

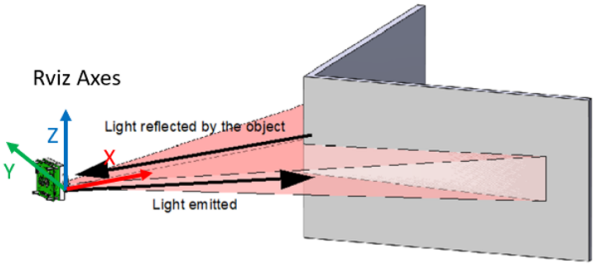
Then run the launch file:

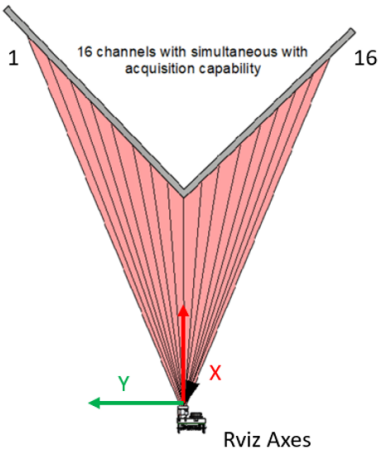
```
roslaunch leddar_ros example.launch
```



Below is a diagram of the representation in Rviz according to the sensor orientation:







It is possible to move the sensor axes positions with regards to the map axes position using **device.xml**. See the sections highlighted in blue below.

3.5.1. Leddar M16/IS16 ROS Code Examples

"`leddar_ros\launch\`**`example.launch`**"

```
<launch>
 <!-- Connection info, see python leddar.Device.connect for more info -->
 <arg name="param1" default="AK47027" />
 <!-- [All] value returned by GetDeviceList['name'],
  [Serial] serial port com,
  [USB] serial number,
  [SPI-FTDI] FTDI cable id (use get_devices("SpiFTDI")),
  [CANBus komodo] Baudrate in kbps,
  [Ethernet] Ip address -->
 <arg name="device_type" default="Usb" />
 <!-- (optional but recommended) Device type | connection type -
  Both are mandatory for CANBus protocol (from leddar.device_types dictionnary) -->
 <arg name="param3" default="1"/>
 <!-- (optional) [Serial] modbus address (default 1),
  [CANBus komodo] Tx (default 0x750) ,
  [Ethernet] port (default 48630) -->
 <arg name="param4" default="0"/>
 <!-- param4: (optional) [Serial] baudrate (default 115200),
  [CANBus komodo] Rx (default 0x740),
  [Ethernet] Communication timeout -->

 <include file="$(find leddar_ros)/config/device.xml" ns="LeddarTech_1" pass_all_args="true">
 </include>

 <node pkg="rviz" type="rviz" name="rviz" args="-d $(find leddar_ros)/rviz/example.rviz" output="screen"/>
</launch>
```

“leddar_ros\config**device.xml**”

```
<launch>
 <!-- Connection info, see python leddar.Device.connect for more info -->
 <arg name="param1" default="AK47027" />
 <!-- [All] value returned by GetDeviceList['name'],
  [Serial] serial port com,
  [USB] serial number,
  [SPI-FTDI] FTDI cable id (use get_devices("SpiFTDI")),
  [CANBus komodo] Baudrate in kbps,
  [Ethernet] Ip address -->
 <arg name="device_type" default="Usb" />
 <!-- (optional but recommended) Device type | connection type -
  Both are mandatory for CANBus protocol (from leddar.device_types dictionnary) -->
 <arg name="param3" default="1"/>
 <!-- (optional) [Serial] modbus address (default 1),
  [CANBus komodo] Tx (default 0x750) ,
  [Ethernet] port (default 48630) -->
 <arg name="param4" default="0"/>
 <!-- param4: (optional) [Serial] baudrate (default 115200),
  [CANBus komodo] Rx (default 0x740),
  [Ethernet] Communication timeout -->

 <arg name="frame_id" default="sensor_frame" />
 <arg name="parent_frame_id" default="map" />

 <!-- Position of the Sensor <arg name="position" default="X Y Height" /> -->
 <arg name="position" default="0 0 0" />
 <!-- Euler_angles in ROS are (yaw, pitch, roll) in rad
  rotate about fixed X-axis by Roll
  rotate about fixed Y-axis by Pitch
  rotate about fixed Z-axis by Yaw
 -->
  <arg name="rotation" default="0 0 0" />

 <!-- Quaternion -->
 <!--<arg name="rotation" default="-0.707106781 0 0 0.707106781" />  -->

 <node pkg="leddar_ros" type="device.py" name="sensor">
  <param name="device_type" value="$(arg device_type)" />
  <param name="frame_id" value="$(arg frame_id)" />
  <param name="param1" value="$(arg param1)" />
  <param name="param3" value="$(arg param3)" />
  <param name="param4" value="$(arg param4)" />
 </node>

 <node pkg="tf" type="static_transform_publisher" name="leddar_1_to_map"
    args="$(arg position) $(arg rotation) $(arg parent_frame_id) $(arg frame_id) 10" />

</launch>
```

3.6. Running Leddar T16 With ROS for Windows

Use the ROS command prompt update catkin environment:

```
c:\opt\catkin_ws\devel\setup.bat
cd c:\opt\catkin_ws
catkin_make
```

Launch T16 example: To avoid modifying the **.launch** file, it is possible to send all parameters directly in the command line:

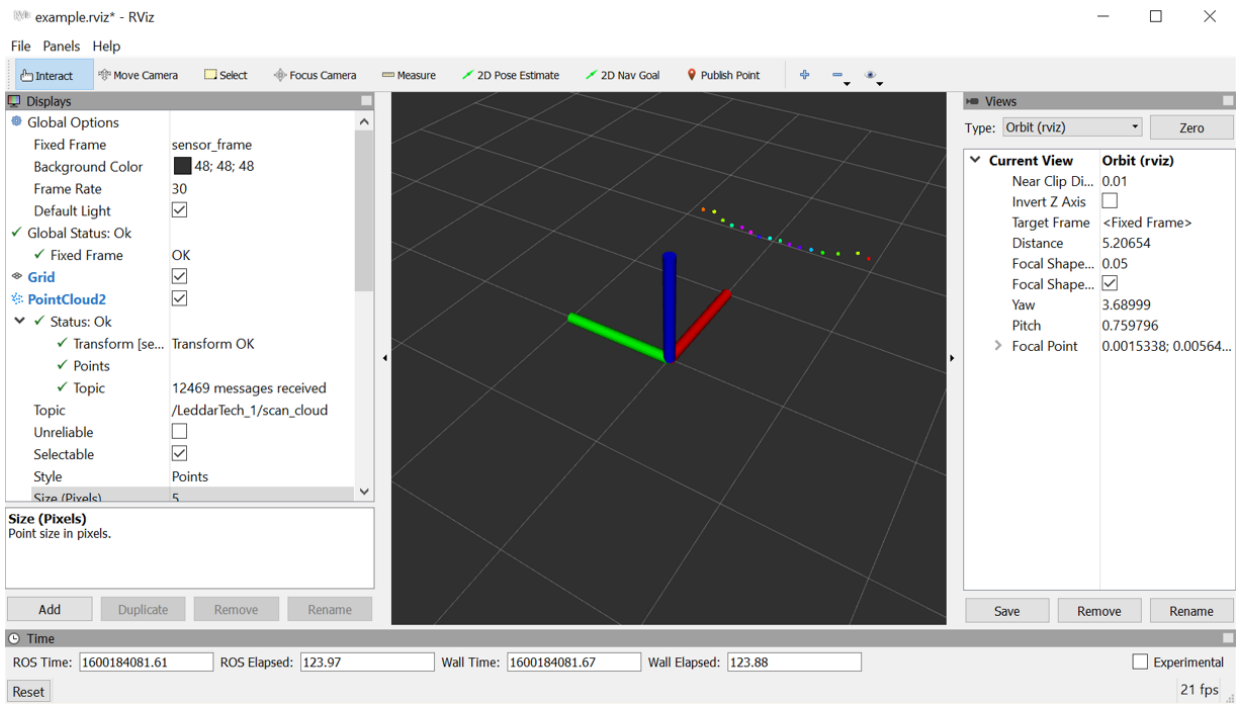
```
roslaunch leddar_ros example.launch param1:=192.168.0.2 device_type:=Ethernet param3:=48620
param4:=0
```

Alternatively, it is possible to modify the example.launch to reflect the required parameters:

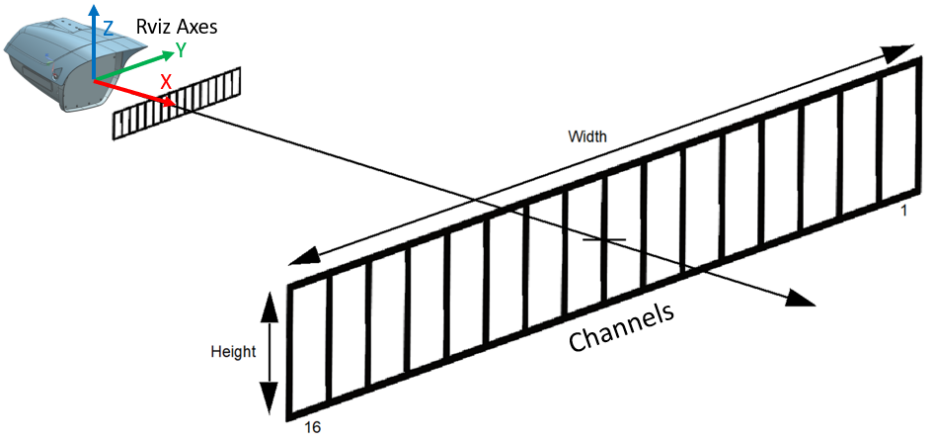
```
<arg name="param1" default="192.168.0.2" />
<arg name="device_type" default="Ethernet" />
<arg name="param3" default="48620"/>
<arg name="param4" default="0"/>
```

Then run the launch file:

```
roslaunch leddar_ros example.launch
```



Below is a diagram of the representation in Rviz according to the sensor orientation:



It is possible to move the sensor axes positions with regards to the map axes position using **device.xml**. See the sections highlighted in blue below.

3.6.1. Leddar T16 ROS Code Examples

"`leddar_ros\launch\`**`example.launch`**"

```
<launch>
  <!-- Connection info, see python leddar.Device.connect for more info -->
  <arg name="param1" default="192.168.0.2" />
  <!-- [All] value returned by GetDeviceList['name'],
    [Serial] serial port com,
    [USB] serial number,
    [SPI-FTDI] FTDI cable id (use get_devices("SpiFTDI")),
    [CANBus komodo] Baudrate in kbps,
    [Ethernet] Ip address -->
  <arg name="device_type" default="Ethernet" />
  <!-- (optional but recommended) Device type | connection type -
    Both are mandatory for CANBus protocol (from leddar.device_types dictionnary) -->
  <arg name="param3" default="48620"/>
  <!-- (optional) [Serial] modbus address (default 1),
    [CANBus komodo] Tx (default 0x750) ,
    [Ethernet] port (default 48630) -->
  <arg name="param4" default="0"/>
  <!-- param4: (optional) [Serial] baudrate (default 115200),
    [CANBus komodo] Rx (default 0x740),
    [Ethernet] Communication timeout -->

  <include file="$(find leddar_ros)/config/device.xml" ns="LeddarTech_1" pass_all_args="true">
  </include>

  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find leddar_ros)/rviz/example.rviz" output="screen"/>
</launch>
```

"`leddar_ros\config\`**`device.xml`**"

```
<launch>
  <!-- Connection info, see python leddar.Device.connect for more info -->
  <arg name="param1" default="192.168.0.2" />
  <!-- [All] value returned by GetDeviceList['name'],
    [Serial] serial port com,
    [USB] serial number,
    [SPI-FTDI] FTDI cable id (use get_devices("SpiFTDI")),
    [CANBus komodo] Baudrate in kbps,
    [Ethernet] Ip address -->
  <arg name="device_type" default="Ethernet" />
  <!-- (optional but recommended) Device type | connection type -
    Both are mandatory for CANBus protocol (from leddar.device_types dictionnary) -->
  <arg name="param3" default="48620"/>
  <!-- (optional) [Serial] modbus address (default 1),
    [CANBus komodo] Tx (default 0x750) ,
    [Ethernet] port (default 48630) -->
  <arg name="param4" default="0"/>
  <!-- param4: (optional) [Serial] baudrate (default 115200),
    [CANBus komodo] Rx (default 0x740),
    [Ethernet] Communication timeout -->

  <arg name="frame_id" default="sensor_frame" />
  <arg name="parent_frame_id" default="map" />

  <!-- Position of the Sensor <arg name="position" default="X Y Height" /> -->
  <arg name="position" default="0 0 0" />
  <!-- Euler_angles in ROS are (yaw, pitch, roll) in rad
    rotate about fixed X-axis by Roll
    rotate about fixed Y-axis by Pitch
    rotate about fixed Z-axis by Yaw
  -->
   <arg name="rotation" default="0 0 0" />

  <!-- Quaternion -->
  <!--<arg name="rotation" default="-0.707106781 0 0 0.707106781" />  -->

  <node pkg="leddar_ros" type="device.py" name="sensor">
    <param name="device_type" value="$(arg device_type)" />
    <param name="frame_id" value="$(arg frame_id)" />
    <param name="param1" value="$(arg param1)" />
    <param name="param3" value="$(arg param3)" />
    <param name="param4" value="$(arg param4)" />
  </node>

  <node pkg="tf" type="static_transform_publisher" name="leddar_1_to_map"
      args="$(arg position) $(arg rotation) $(arg parent_frame_id) $(arg frame_id) 10" />

</launch>
```

LeddarTech®

LeddarTech® has made every effort to ensure that the information contained in this document is accurate. Any information herein is provided "AS IS." LeddarTech shall not be liable for any errors or omissions herein or for any damages arising out of or related to the information provided in this document. LeddarTech reserves the right to modify design, characteristics and products at any time, without notice, at its sole discretion.

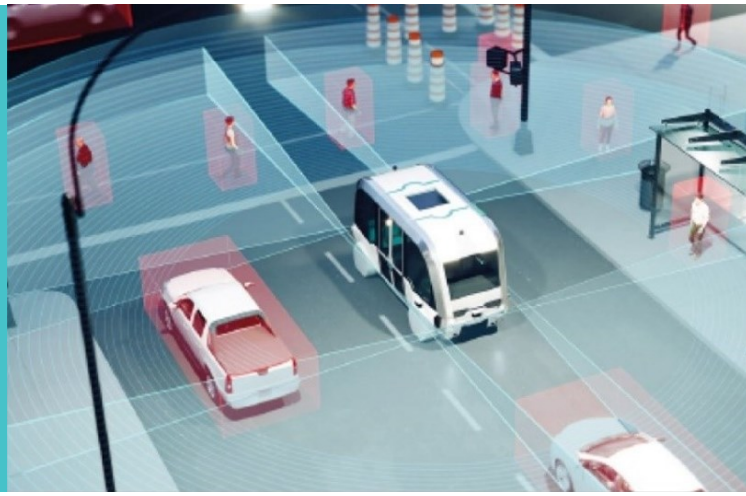
LeddarTech does not control the installation and use of its products and shall have no liability if a product is used for an application for which it is not suited. You are solely responsible for (1) selecting the appropriate products for your application, (2) validating, designing and testing your application and (3) ensuring that your application meets applicable safety and security standards.

Furthermore, LeddarTech products are provided only subject to LeddarTech's Sales Terms and Conditions or other applicable terms agreed to in writing. By purchasing a LeddarTech product, you also accept to carefully read and to be bound by the information contained in the User Guide accompanying the product purchased.

About LeddarTech

LeddarTech is a leader in environmental sensing platforms for autonomous vehicles and advanced driver assistance systems. Founded in 2007, LeddarTech has evolved to become a comprehensive end-to-end environmental sensing company by enabling customers to solve critical sensing and perception challenges across the entire value chain of the automotive and mobility market segments. With its LeddarVision™ sensor-fusion and perception platform and its cost-effective, scalable and versatile LiDAR development solution for automotive-grade solid-state LiDARs based on the LeddarEngine™, LeddarTech enables Tier 1-2 automotive system integrators to develop full-stack sensing solutions for autonomy level 1 to 5. These solutions are actively deployed in autonomous shuttles, trucks, buses, delivery vehicles, smart cities/factories and robotaxi applications. The company is responsible for several innovations in cutting-edge automotive and mobility remote-sensing applications, with over 100 patented technologies (granted or pending) enhancing ADAS and autonomous driving capabilities.

For more information: sales@leddartech.com



LeddarTech®

CANADA – USA – AUSTRIA – FRANCE – GERMANY – ITALY – ISRAEL – HONG KONG – CHINA

Head Office
4535, boulevard Wilfrid-Hamel, Suite 240
Québec (Québec) G1P 2J7, Canada
leddartech.com

Phone: + 1-418-653-9000
Toll-free: 1-855-865-9900