Solid-State LiDAR ML-X

User Guide

Version History	Last updated	Description
Release v2.0.0	2023-07-25	• Add multi-echo function
Release v2.1.0	2023-09-07	• Add PTP, F/W update function • Add callback function • Support for Python, ROS2
Release v2.1.1	2024-02-15	• Add laser safety cautions
Release v2.1.2	2024-03-07	• Modify laser safety label • Add caution text
Release v2.3.0	2024-04-05	• Modify IP Changer
Release v2.3.1	2024-06-05	• Add/Modify Python API

Table of Contents

Table of Contents

Chapter 1

Hardware Configuration

1.1. Mechanical Interface

1.1.1 Included Components

ML-X is provided with the following items:

- ML-X 120° Dedicated (Power/Ethernet/Time Sync) Cable
- Sensor AC/DC Power Adapter/Power Cable

(a) ML-X 120°

(b) 120° Dedicated Cable

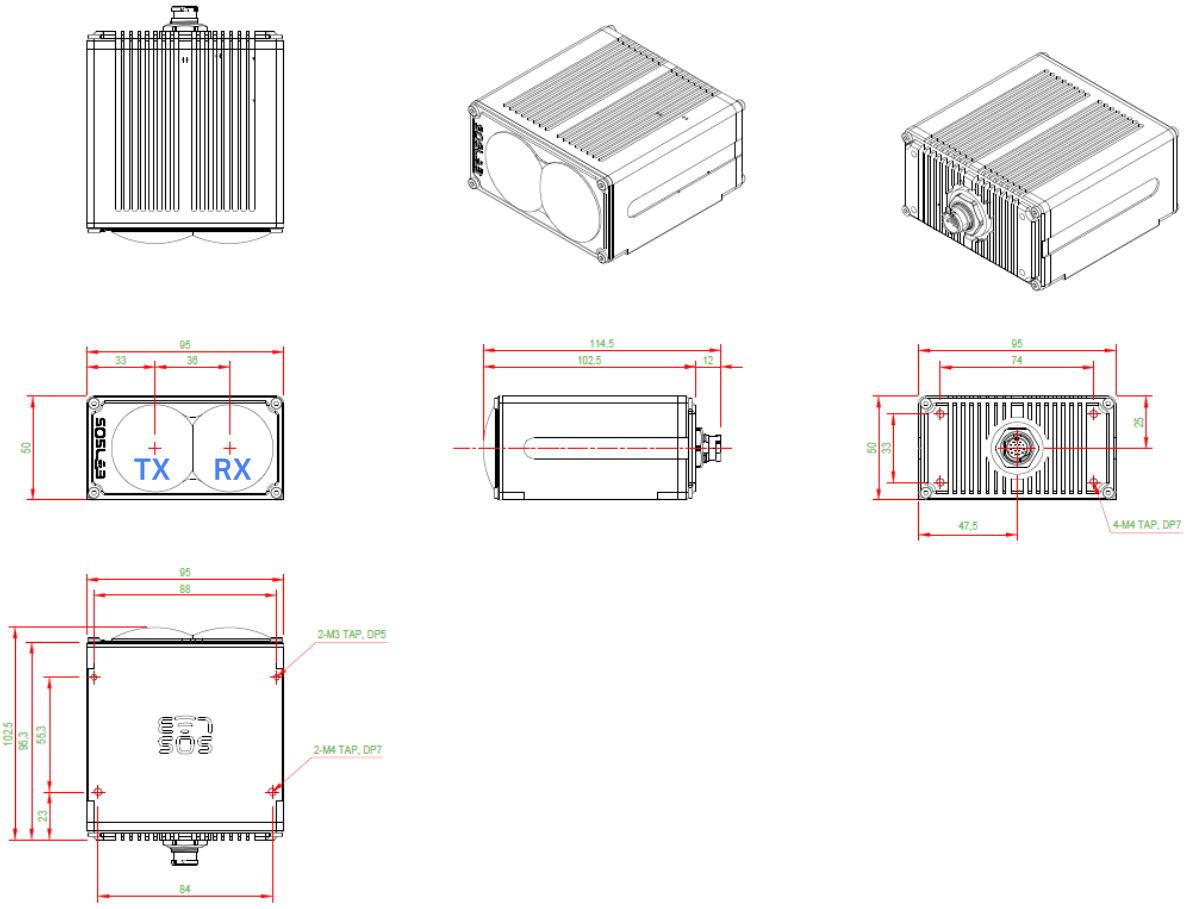
(c) AC/DC Power Adapter

(d) AC/DC Power Cable

1.1. Mechanical Interface

1.1.2 Exterior Mechanical Dimensions

ML-X 120° Dimensions



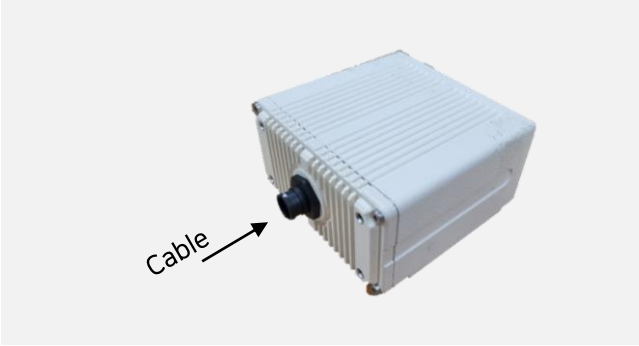
<The sensor has 4×M4 mounting holes>

1.2. Electrical Interface

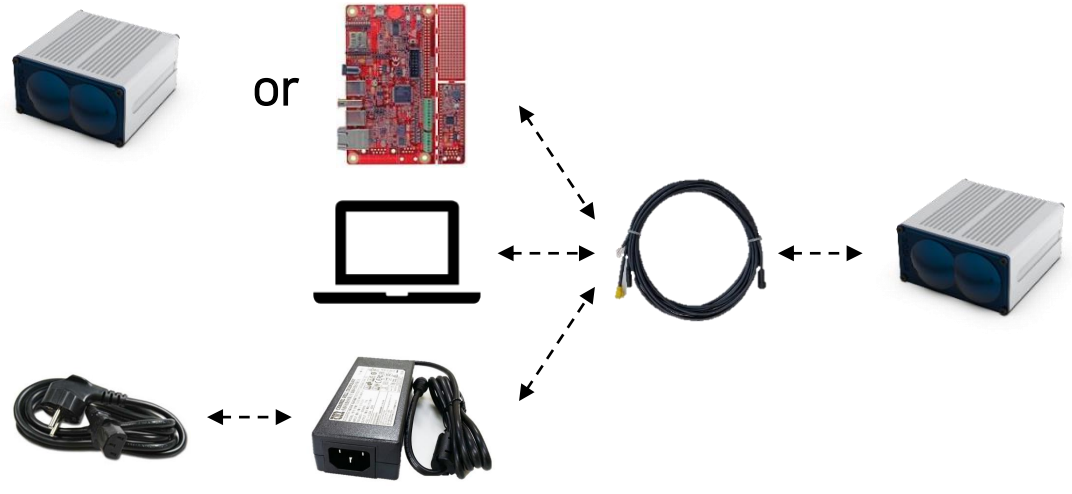
1.2.1 Cable Connection

The sequence for connecting the cables is as follows:

1. Connect the provided sensor to the integrated cable.
2. Connect the Ethernet cable to the PC.
3. If using Timing Sync., connect the provided sensor to the external device.
4. Connect the power cable to the power adapter.



<ML-X 120 ° Cable to Connector>



<ML-X 120° Cable Connection>

1.2.2 IP/Port Information

The default IP/Port information for the provided sensor is as follows:

- Default Local IP : 192.168.1.15
- Default Local Port : 2000
- Default Device IP : 192.168.1.10
- Default Port : 2000

1.2.3 Power Connector

There are two methods for connecting power to the sensor: using the provided power adapter or using an external power cable.

1.2.3.1 Power Adapter Information

The specification for the provided power adapter is as follows:

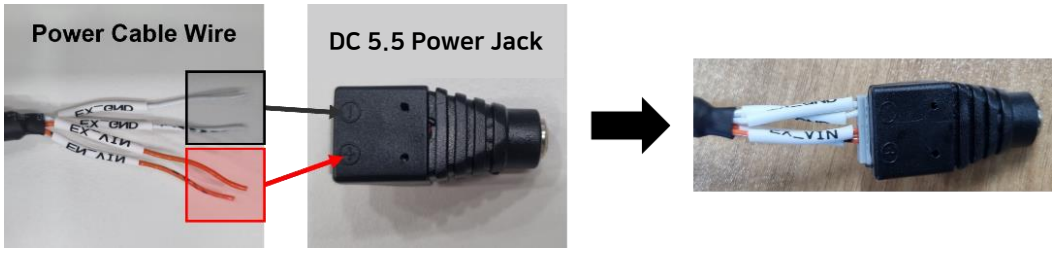
- Rated Input : 100-240V 50/60Hz 1.7A
- Rated Output : +12.0V 5.0A 60.0W

1.2.3.2 External Power Cable

External power cable supplies voltage in the range of 12V~24V and consists of 4 wires.

Power Cable Wire	DC 5.5 전원 잭
White: EX_GND	Negative pole (-)
White+DOT: EX_GND	
Orange: EX_VIN	Positive pole (+)
Orange+DOT: EN_VIN	

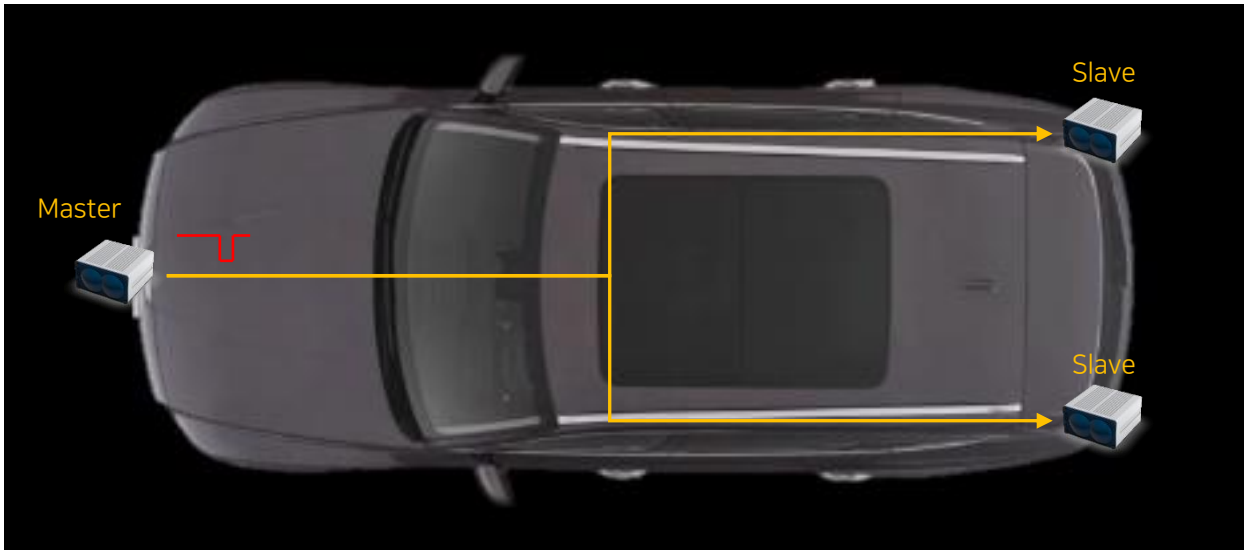
The 4 wires can be connected to the DC 5.5 power jack as follows to be used for power.



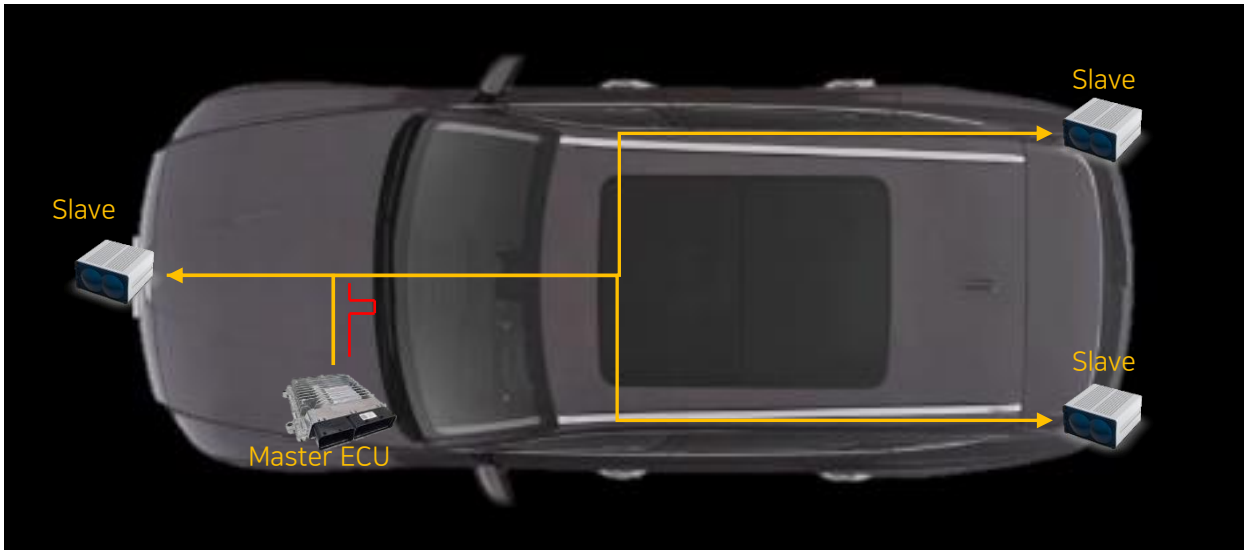
1.2.4 Timing Synchronization

ML-X provides frame synchronization functionality using input/output signals, enabling the configuration of various types of applications as shown in the diagram below.

1) ML-X units can be configured as Master/Slave to implement frame synchronization



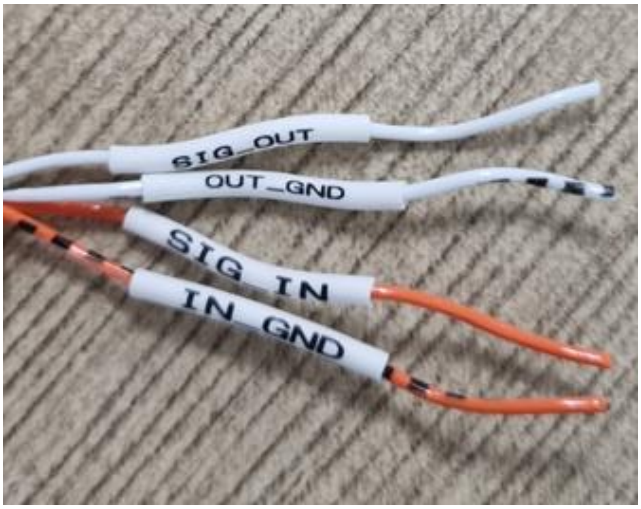
2) ML-X can be synchronized as a Slave by receiving signals from an external device



1.2. Electrical Interface

1.2.4.1 External SYNC IN/OUT Cable

ML-X External Cable provides SYNC IN/OUT ports, which can be used to configure external circuits.

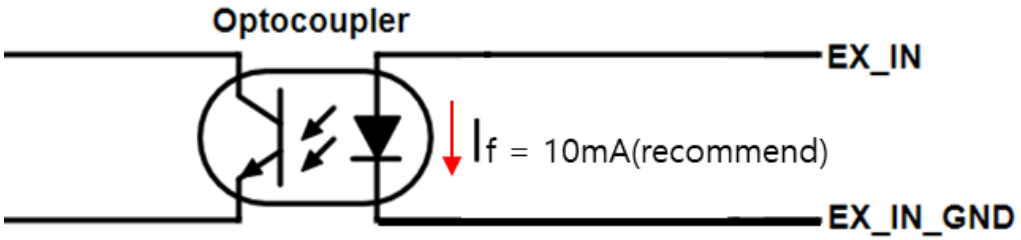


- SIG_IN (Orange, EX_IN)
- IN_GND (Orange+DOT, EX_IN_GND),
- SIG_OUT (White, EX_OUT)
- OUT_GND (White+DOT, EX_OUT_GND)

1.2.4.2 External SYNC IN

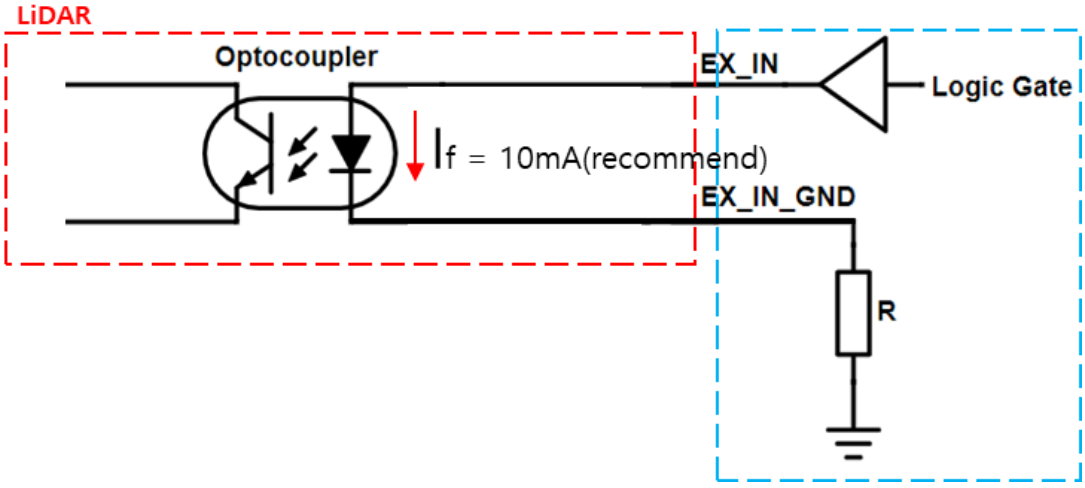
The internal and external circuit configuration of ML-X's sync input are as follows.

1) The internal circuit configuration of ML-X's sync input is as follows.



2) Please refer to the following diagram for the external circuit configuration.

External Circuit



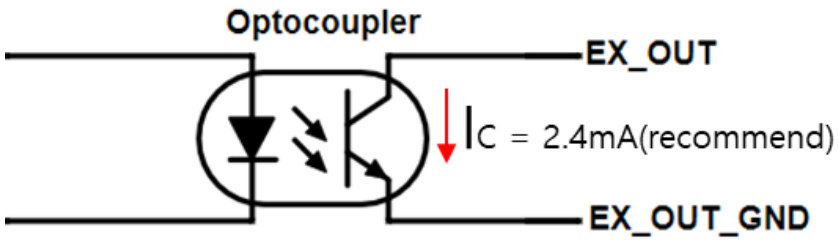
3) Selection of resistance values based on EX_IN Voltage

EX_IN Voltage	External Resistor(R)	Forward Current (I_F)	Recommended I_F
3.3 V (Min)	200	10.25mA	$7.5\ \text{mA} < I_F < 15\ \text{mA}$ $I_F(\text{mA}) = \frac{V - 1.25}{R} * 1000$
5 V	360	10.41 mA	
12 V	1000	10.75 mA	
24 V (Max)	2200	10.34 mA	

1.2.4.3 External SYNC OUT

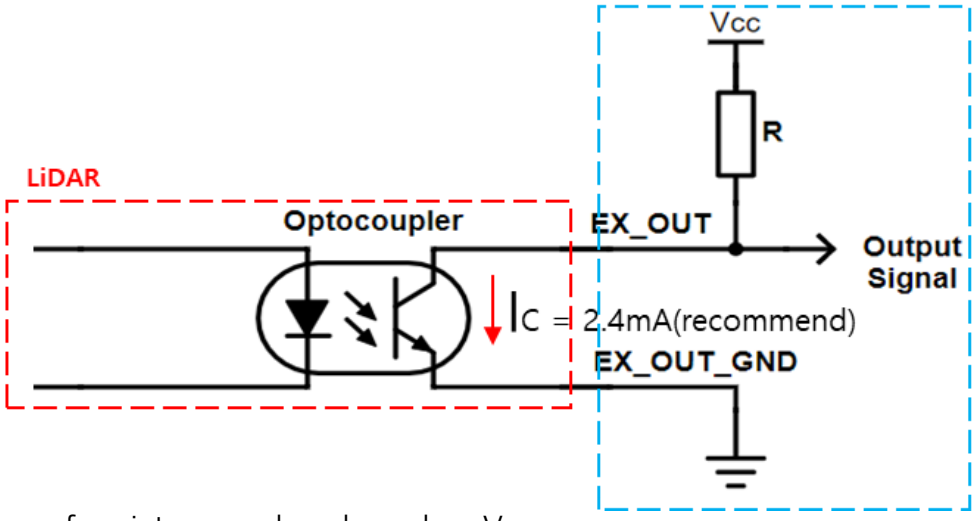
The internal and external circuit configuration of ML-X's sync output are as follows.

1) The internal circuit configuration of ML-X's sync output is as follows.



2) Please refer to the following diagram for the external circuit configuration.

External Circuit



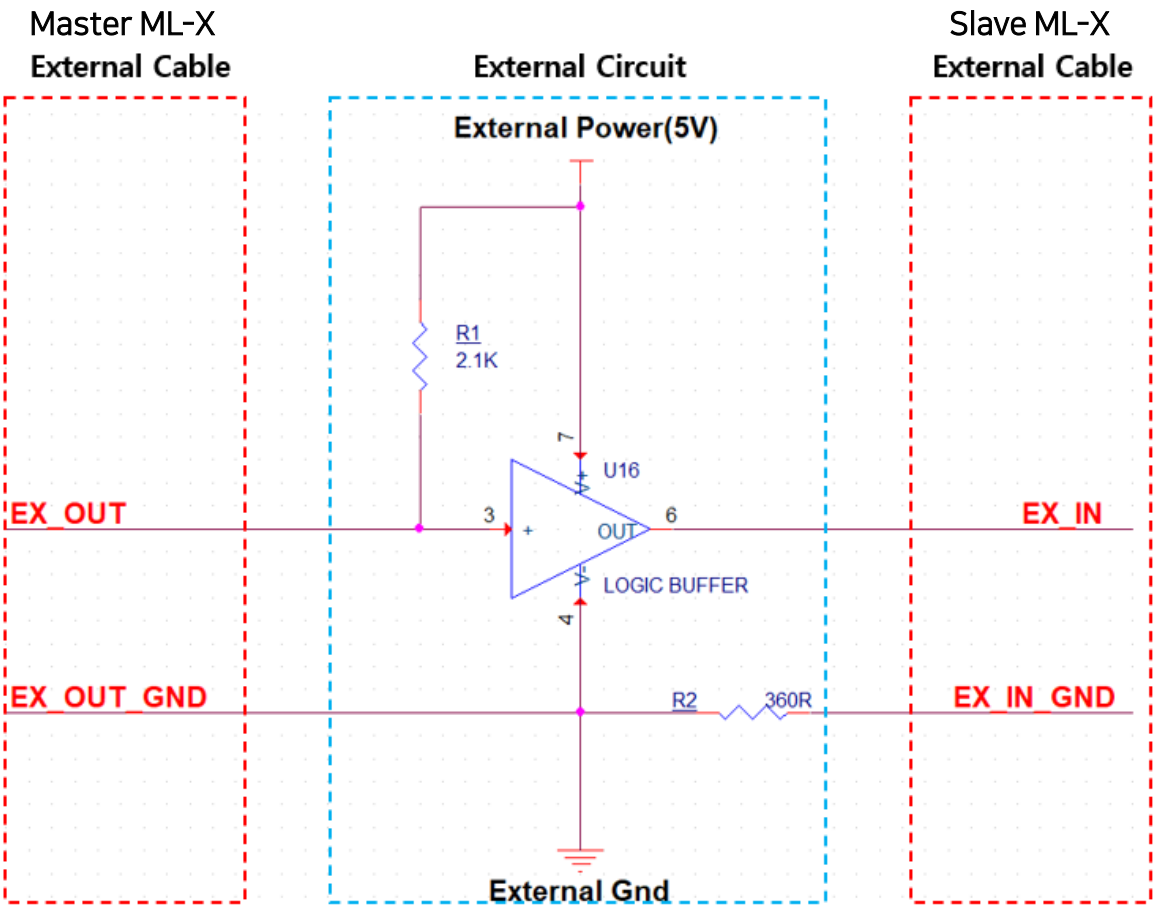
3) Selection of resistance values based on V_{CC}

External Voltage (V_{CC})	External Resistor (R)	Collector Current (I_C)	Recommended I_C
3.3 V (Min)	1375	2.4 mA	$1\ \text{mA} < I_C < 10\ \text{mA}$ $I_C(\text{mA}) = \frac{V_{CC}}{R} * 1000$
5 V	2100	2.4 mA	
12 V	5000	2.4 mA	
24 V (Max)	10000	2.4 mA	

1.2.4.4 Sync IN/OUT Combine

Here is an example of connecting a single master ML-X to a single slave ML-X.

1) External Circuit Configuration
    - External Power: 5V

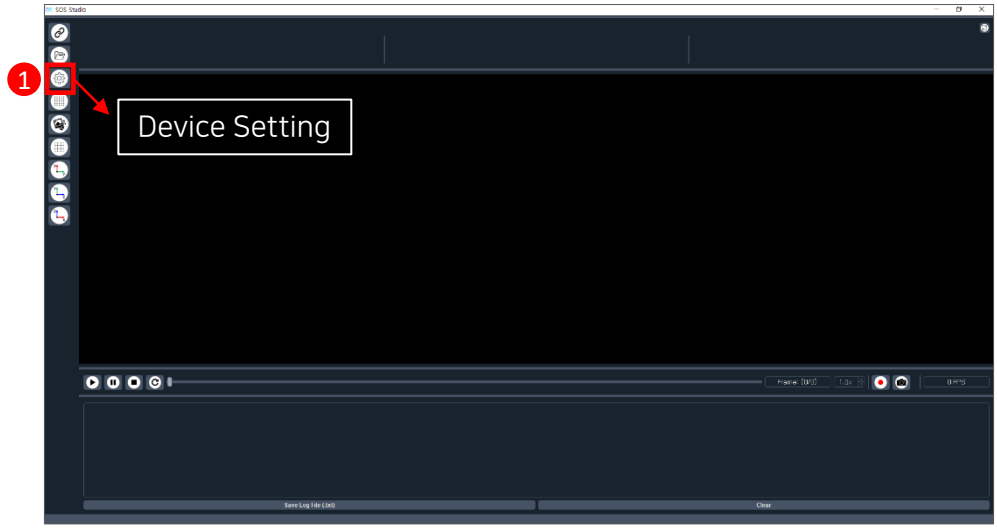


1.2.4.5 SYNC FUNCTION

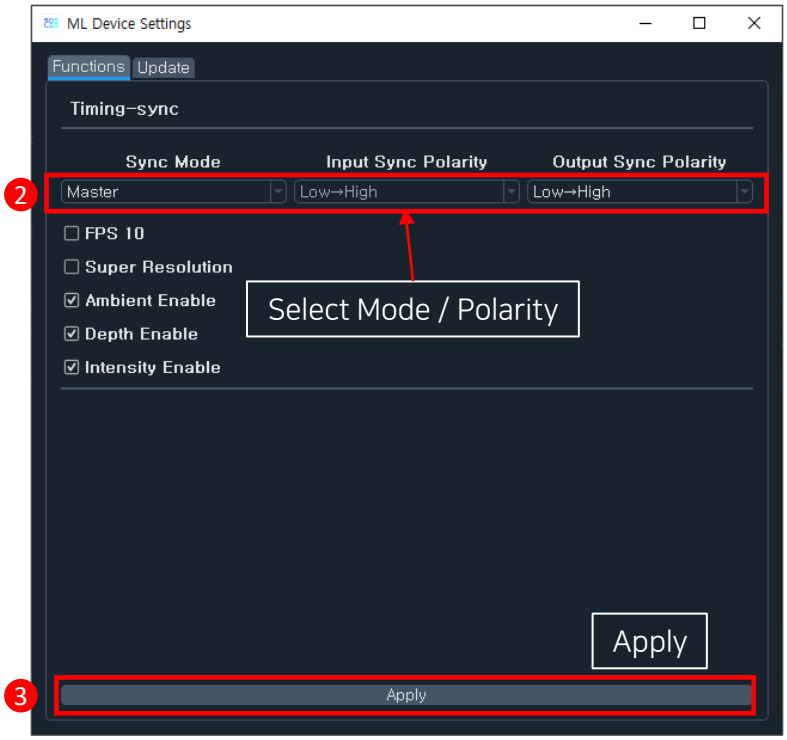
Users can utilize SOS Studio to easily configure the Sync Mode of ML-X as either Master or Slave, granting them full control over the synchronization functionality.

1) Procedure for setting the sync mode

- After launching SOS Studio, click on "Device Setting."

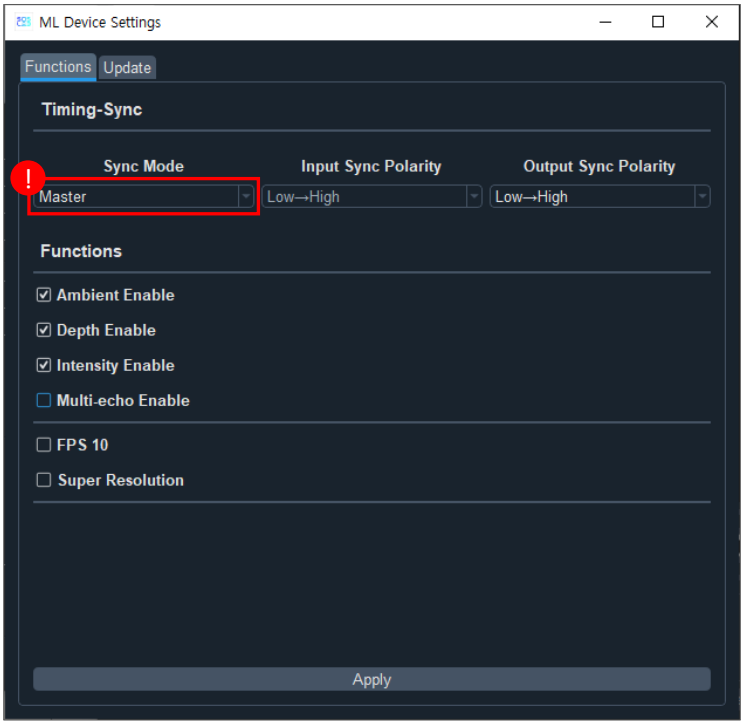


- In the "Functions" tab, user can select and apply the desired mode.

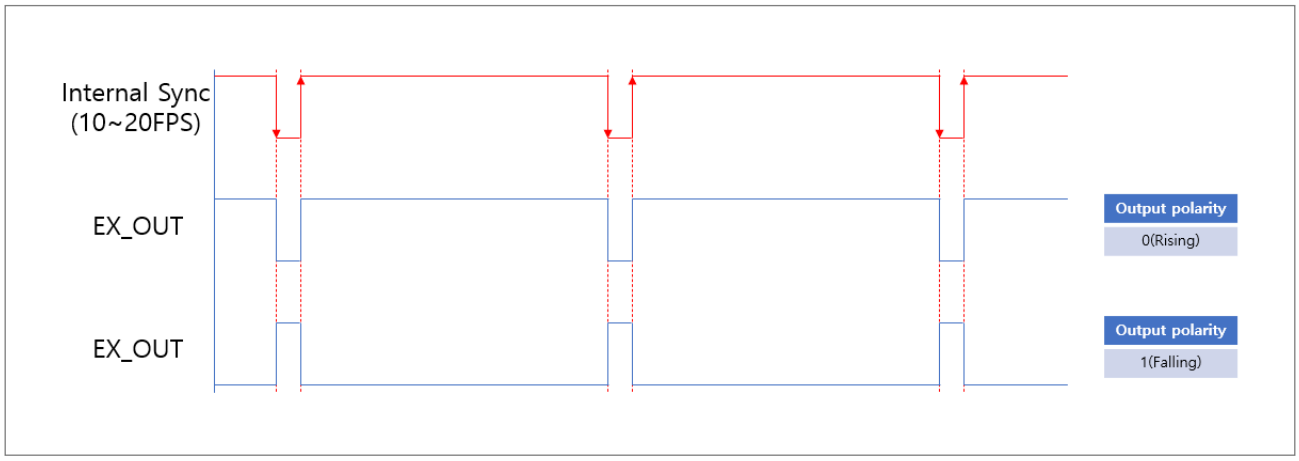


2) MASTER

- When user select "Master" for the sync mode and click on "Apply," ML-X will operate in the master mode.

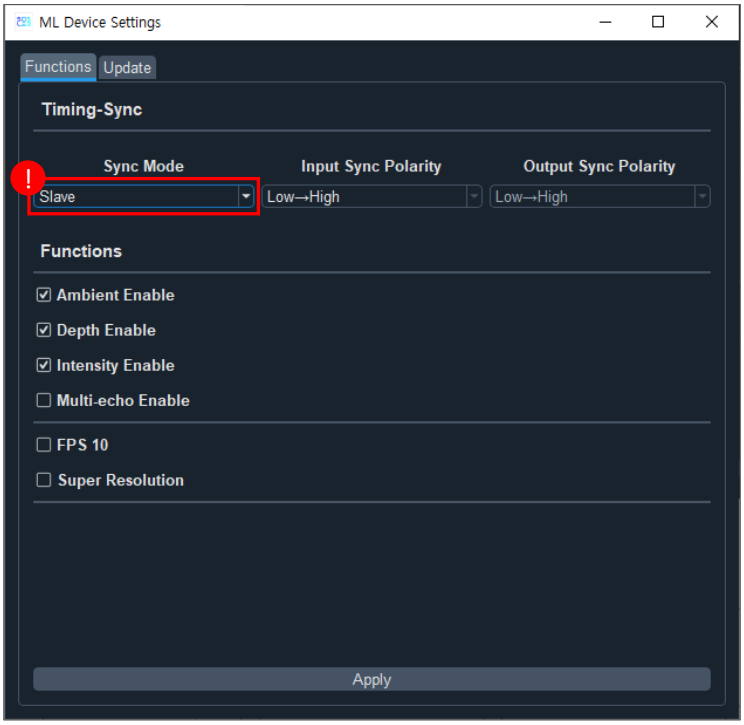


- ML-X outputs synchronization signals for every frame according to the configured frame rate. The polarity of the sync out signal can be changed, allowing for either a High-to-Low or Low-to-High transition.

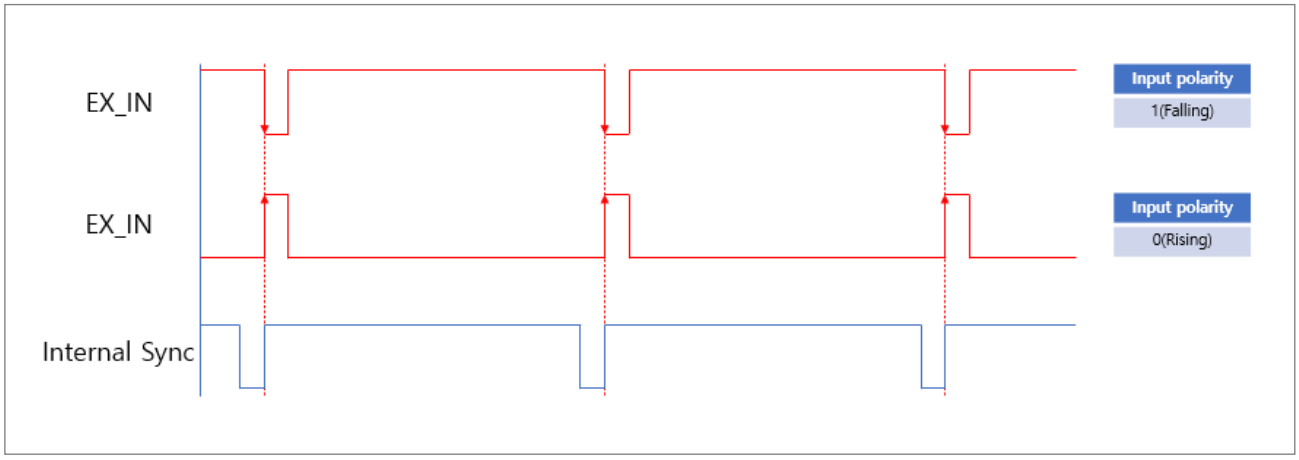


3) SLAVE

- When user select “Slave" for the sync mode and click on "Apply," ML-X will operate in the slave mode.

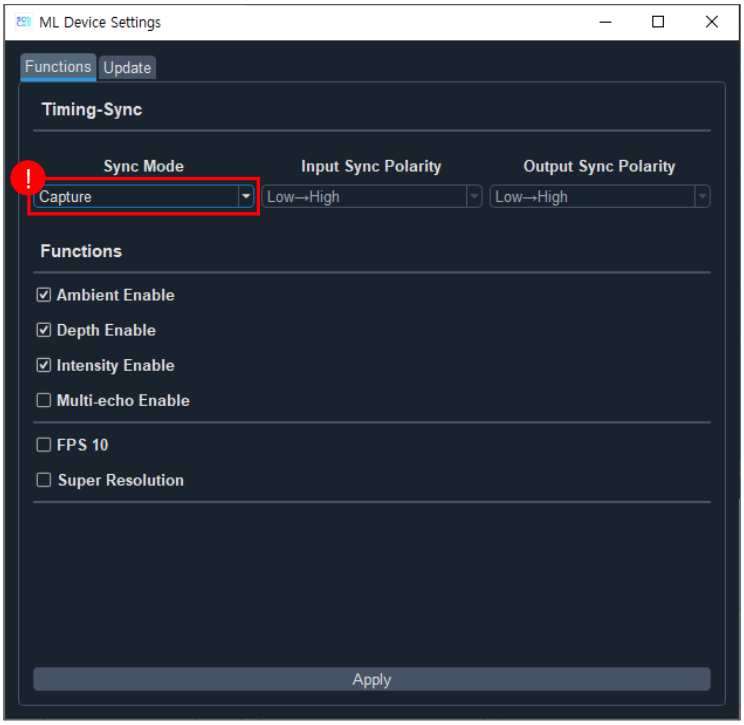


- ML-X operates on a frame-by-frame basis, synchronized with the timing of the incoming synchronization signal. The sync input polarity can be changed, supporting both High-to-Low and Low-to-High transitions.

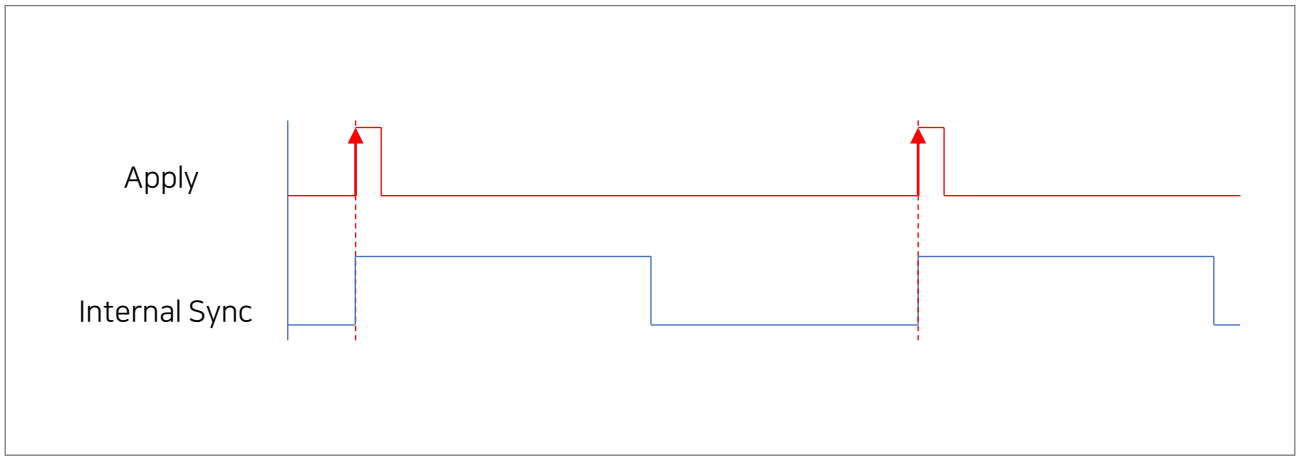


4) CAPUTE

- When user select “Capture" for the sync mode and click on "Apply," ML-X will operate in the capture mode.



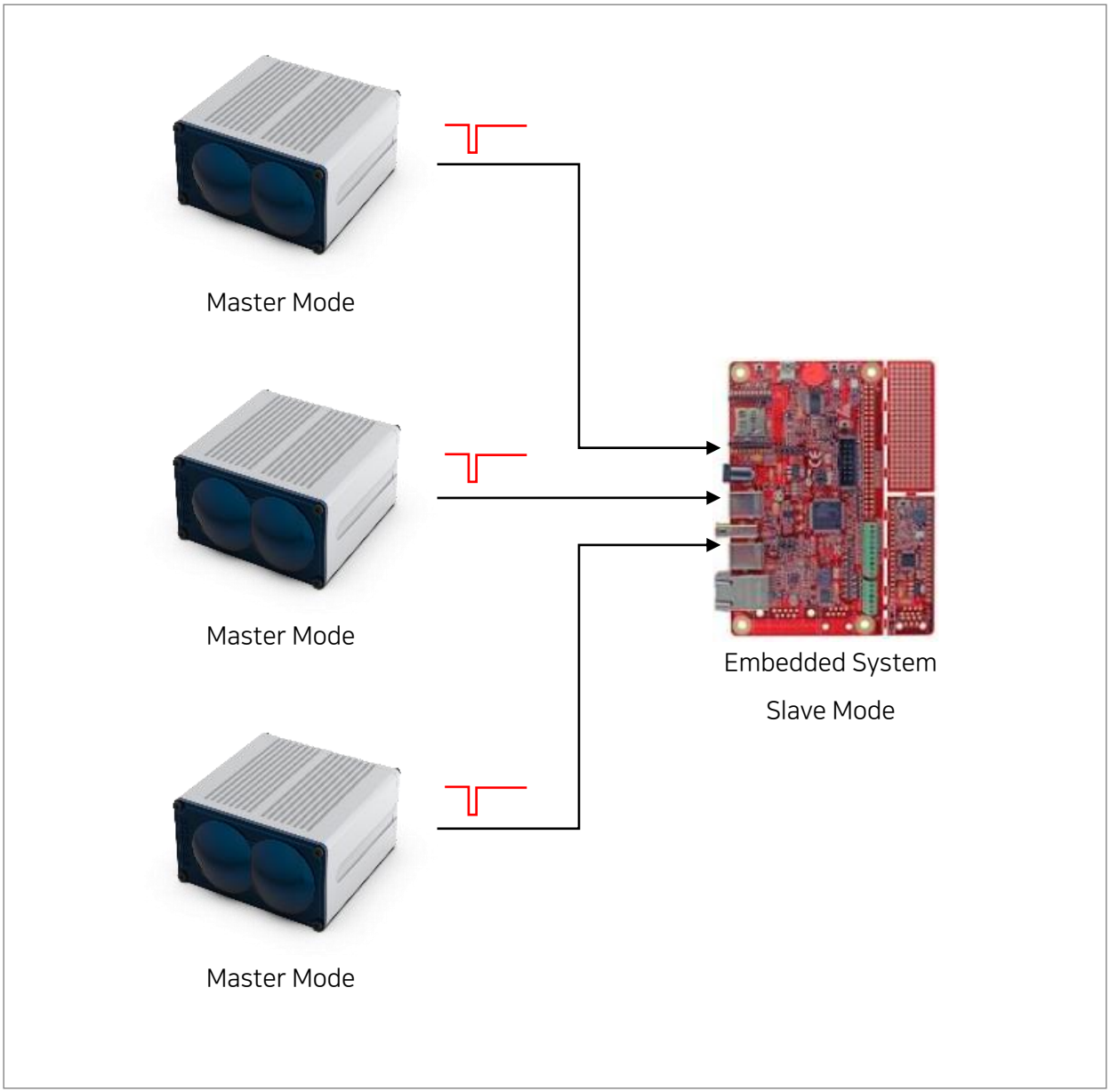
- Each time user click on "Apply," a single internal sync is generated, triggering a scene update.



1.2.4.5 APPLICATION

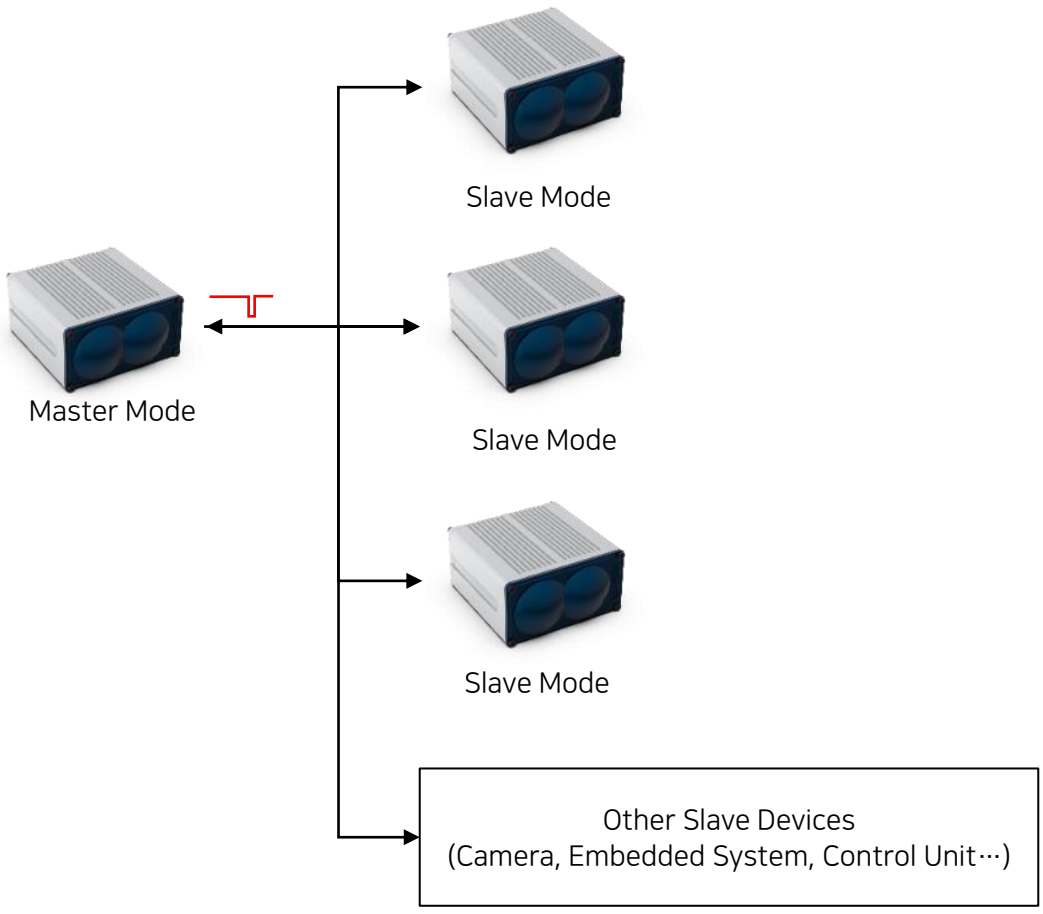
Using input/output synchronization signals, it is possible to configure a synchronized setup with multiple ML-X units or other devices.

1) Multiple Master ML-X → Single Slave Device

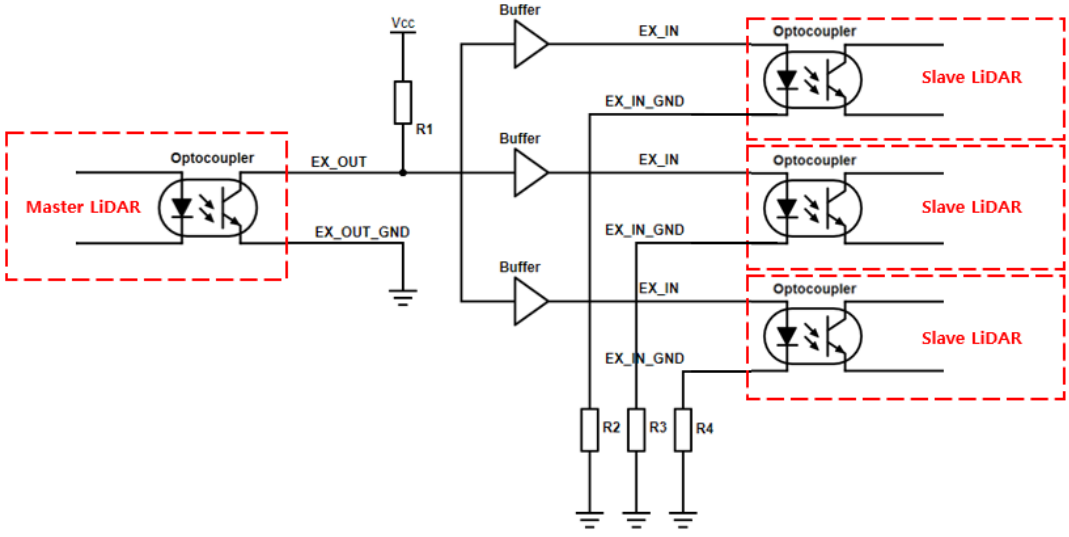


1.2. Electrical Interface

2) Single Master ML-X → Multiple Slave Device

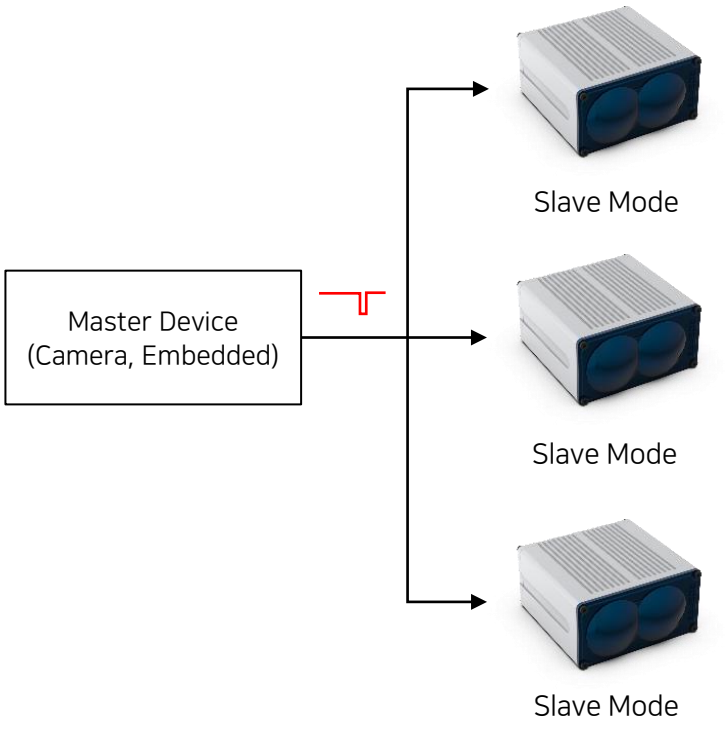


- External Circuit Configuration

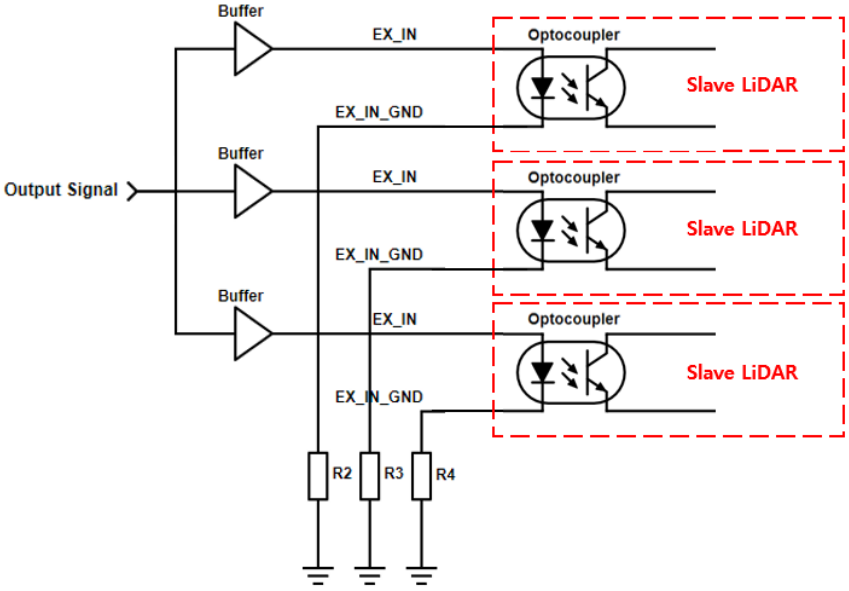


1.2. Electrical Interface

3) Single Master Device → Multiple Slave Device



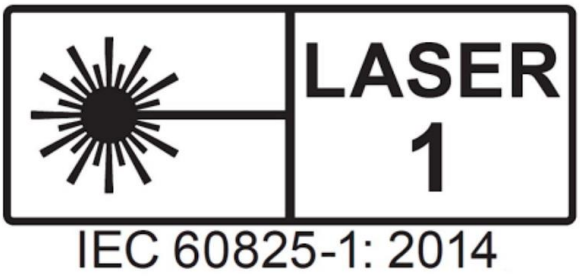
- External Circuit Configuration



1.3. Laser Safety

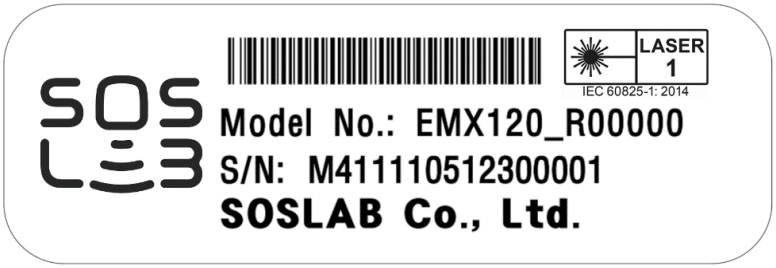
1.3.1 Caution

- Use of controls or adjustments or performance of procedures other than those specified herein may result in hazardous radiation exposure.
- Do NOT stare directly at the product.
- Do NOT disassemble the product without consulting the supplier.
- Avoid applying impact to the product.
- When installing, supply power to the product after cable assembly is completed.



1.3.2 Label

The label below is attached to the bottom of the product.



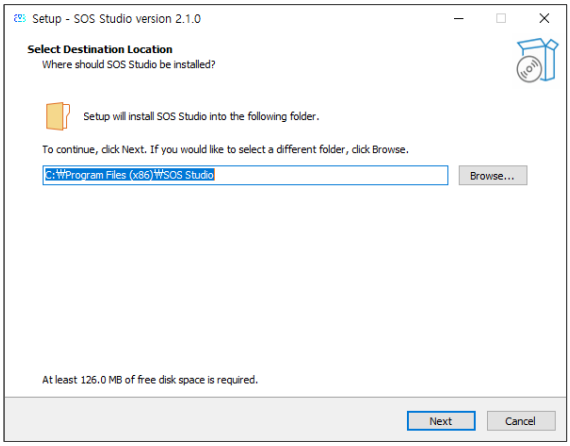
Class 1 Laser Product as stated on the ML-X product attachment label.

Chapter 2

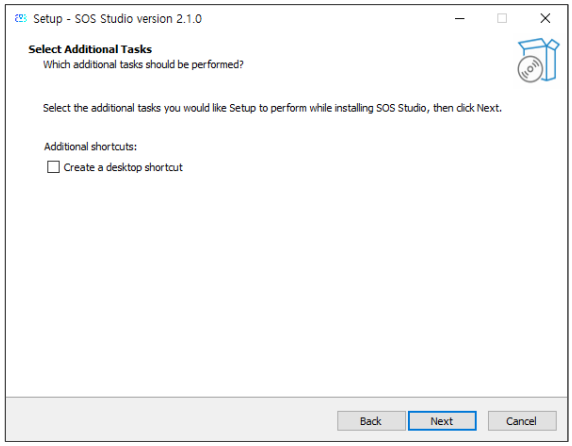
SOS Studio

2.1 Installation

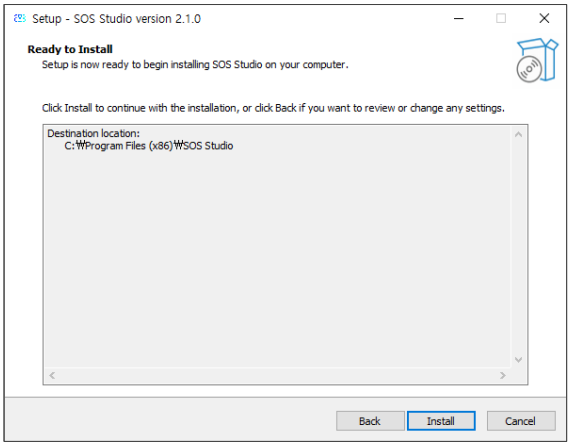
Please execute the installation file "SOS Studio_setup.exe" to begin the installation process. The installation steps are as follows.



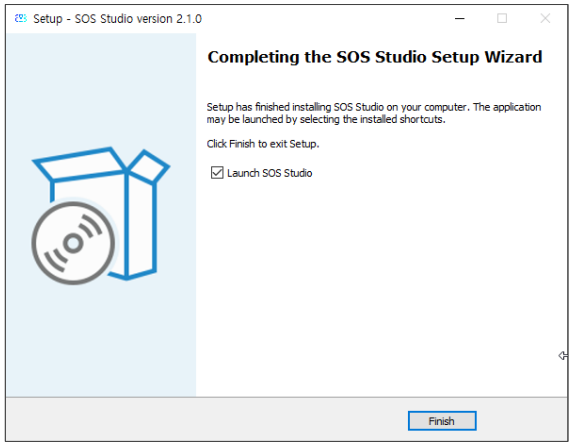
(a) Select Destination Location



(b) Desktop shortcut



(c) Install SOS Studio



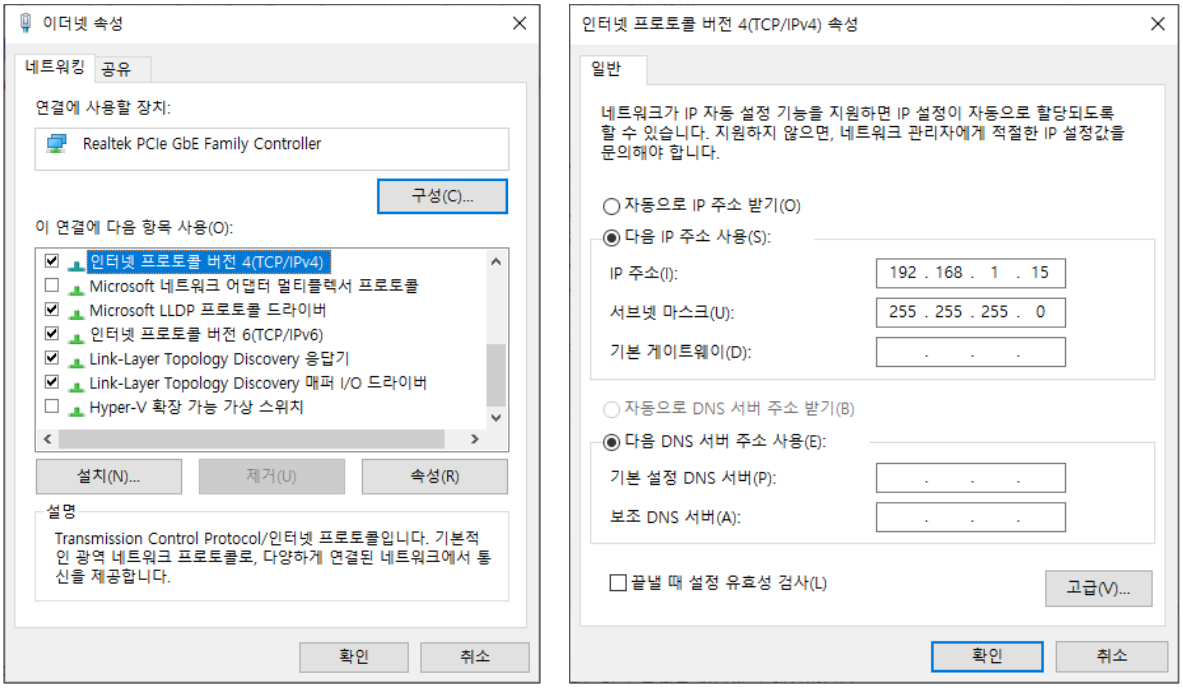
(d) SOS Studio Complete

Once the installation is complete, SOS Studio will be launched automatically.

2.2 TCP/IP Setting

To establish the connection between ML-X LiDAR and the PC, TCP/IP settings need to be configured.

1) Connect the ML-X power cable and use a network cable to connect the PC and LiDAR.
2) Go to Control Panel > Network and Internet > Network and Sharing Center.
3) Click on the active Ethernet connection and open the Properties window.
4) Select Internet Protocol Version 4 (TCP/IPv4) and click on the Properties.
5) In the Properties window, click on the "Use the following IP address" option.
6) Set the IP address (192.168.1.15) and Subnet mask (255.255.255.0) as shown in the figure below.

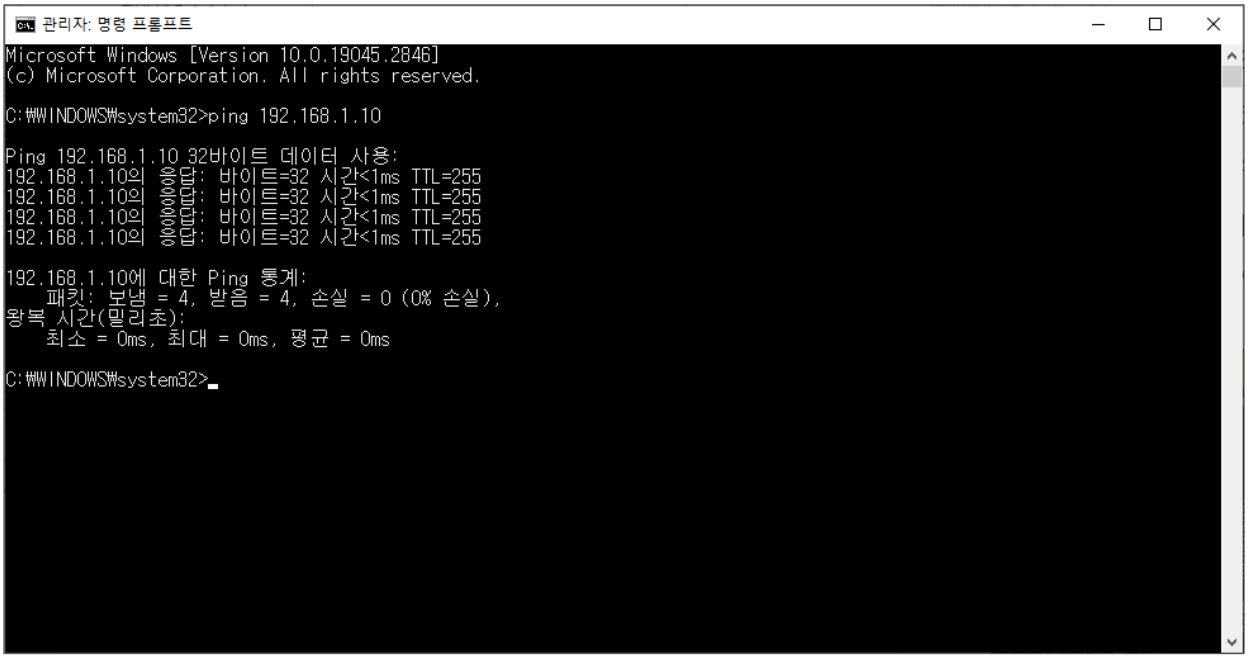


Setting of the Ethernet and Internet Protocol Version 4 (TCP/IPv4)

2.2 TCP/IP Setting

Once the cable connection and IP configuration are completed, user can verify if the PC and ML-X LiDAR are successfully connected by following the Ping test procedure.

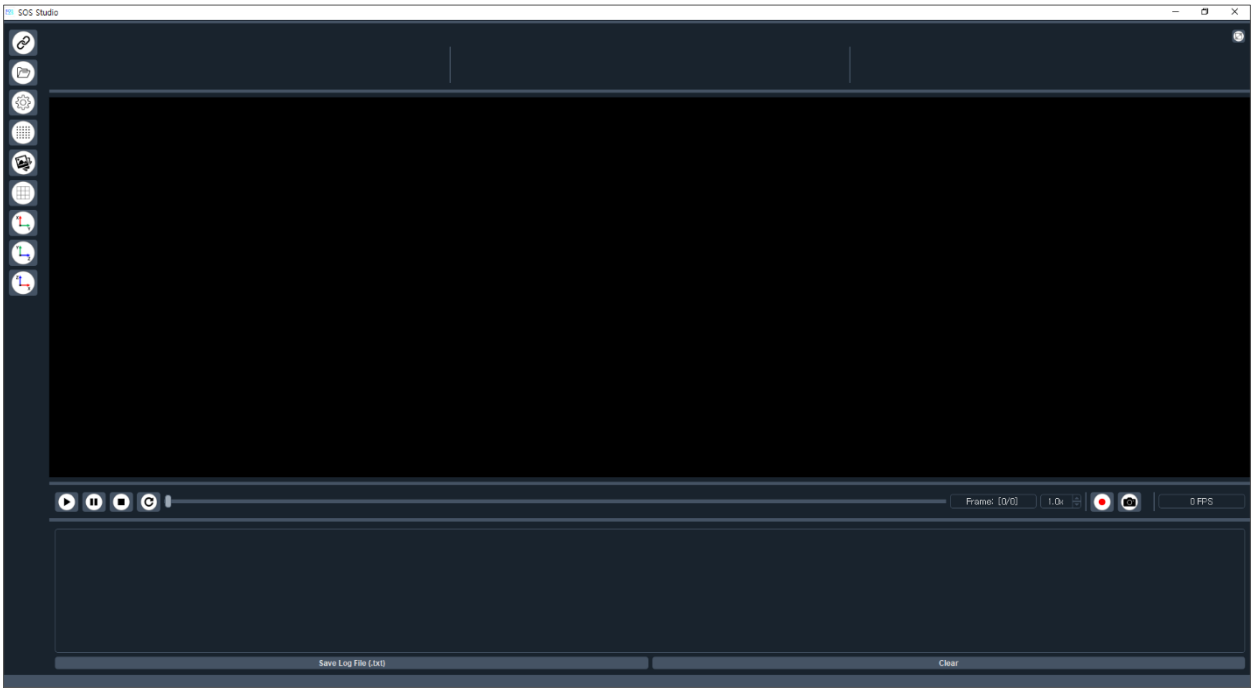
1) Launch the Command Prompt.
2) In the command prompt, enter the command 'ping 192.168.1.10'.
3) If the PC and ML-X are properly connected, user should see below messages.



Example of the Command Prompt window and successful ping test result

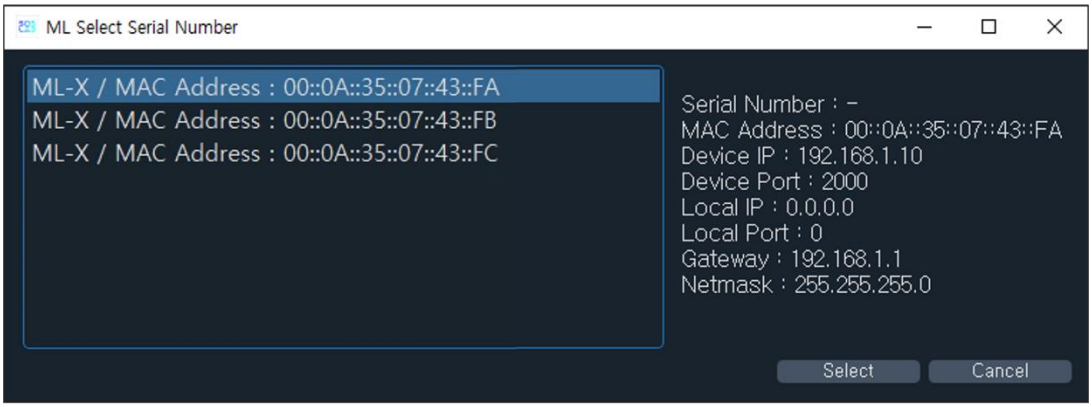
2.3 Connection

This is the SOS Studio execution screen.



SOS Studio Execution Screen

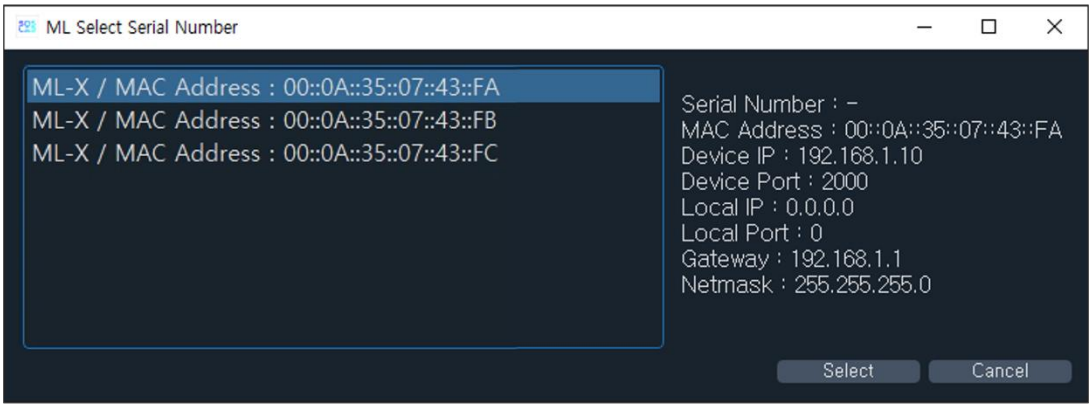
To establish a connection between PC and ML-X, click on the first button on the left in SOS Studio labeled "Connection." This will bring up a list of available ML-X devices that user can connect to.



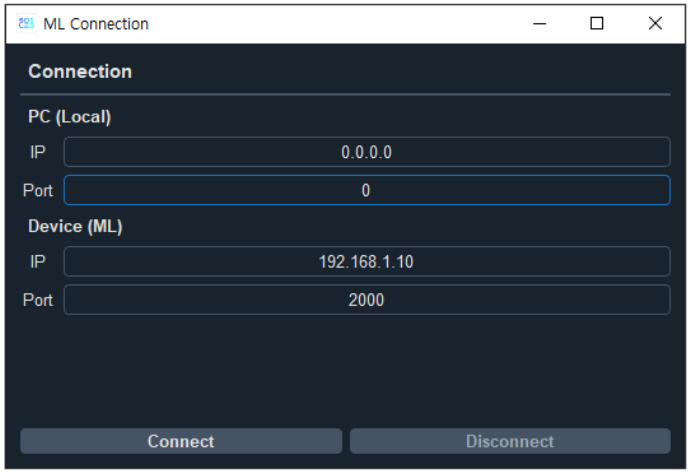
ML-X list

2.3 Connection

After selecting the desired ML-X from the ML-X List, click the "Select" button. This will automatically update the IP and Port of the selected ML-X in the Connection popup window. Click the "Connect" button in the Connection popup window to establish a connection between PC and the ML-X device.



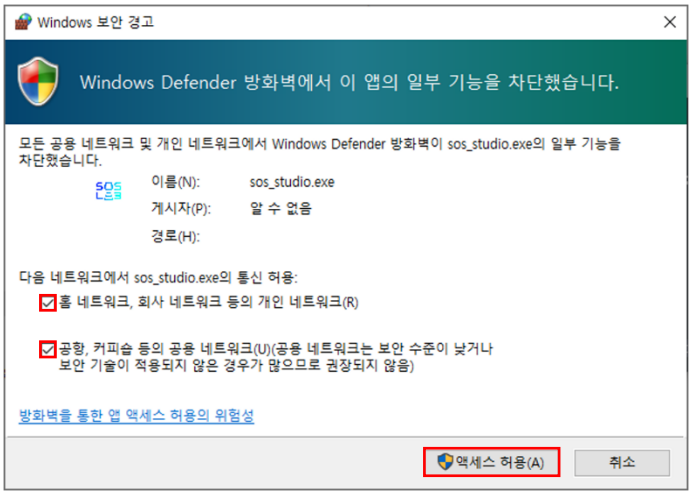
ML-X List



Connection Window

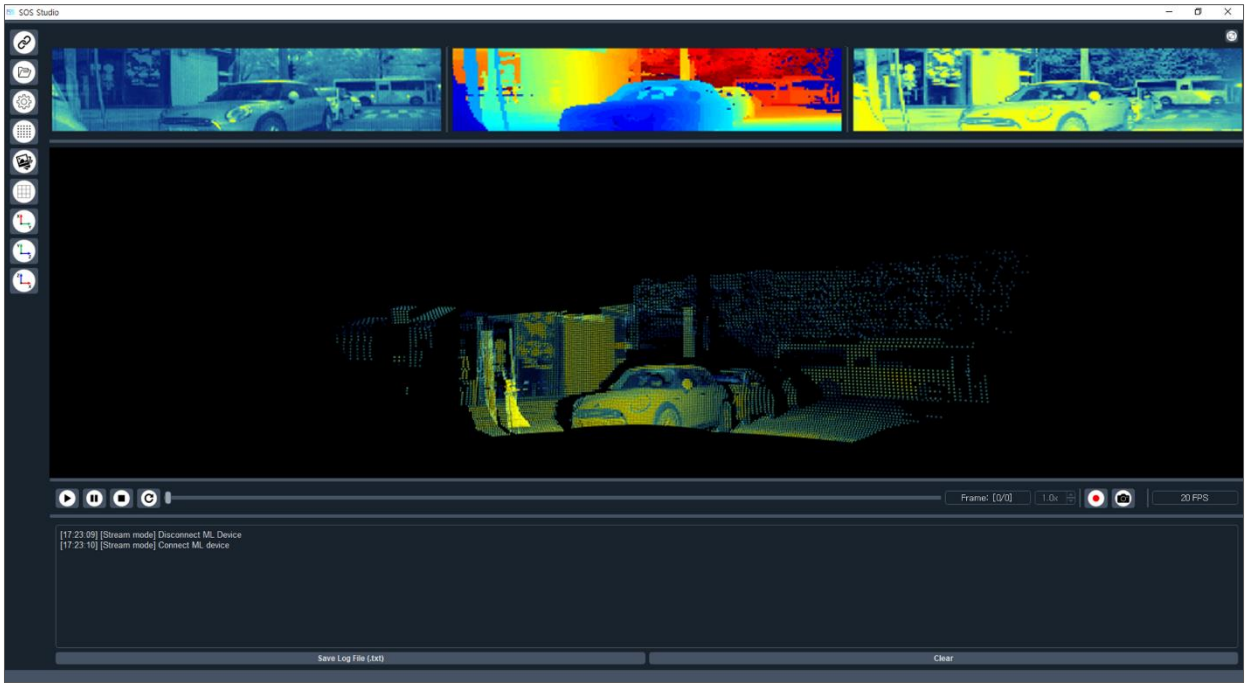
2.3 Connection

When initially connecting the PC and ML-X, user may encounter a Windows Security Alert window as follows. Check the two checkboxes as shown in the image below, then click the "Allow access (A)" button.



Windows Security Alert window

Once the connection between the PC and ML-X is established, the Image Viewer and the Point Cloud Viewer in the center become active.

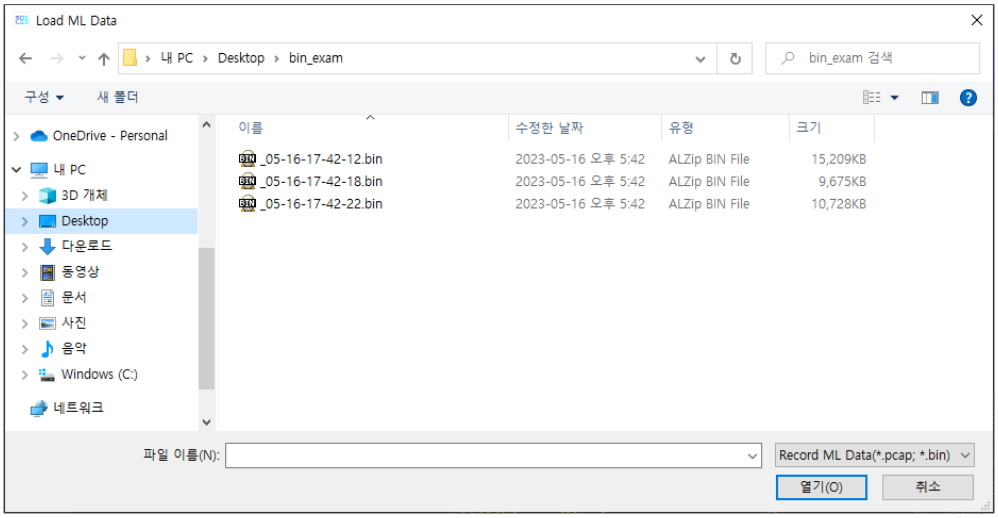


SOS Studio Execution Screen

2.4 Data Load

Data load is used to load stored ML-X data. The functionality for storing data and playing data is described in the **2.10 Play / Record Mode**.

To load ML-X data, click on the second button on the left in SOS Studio labeled "Data Load." This will bring up a window where user can select a file, as shown below. Then, choose the stored ML-X data (.bin) file and click the "Open" button.



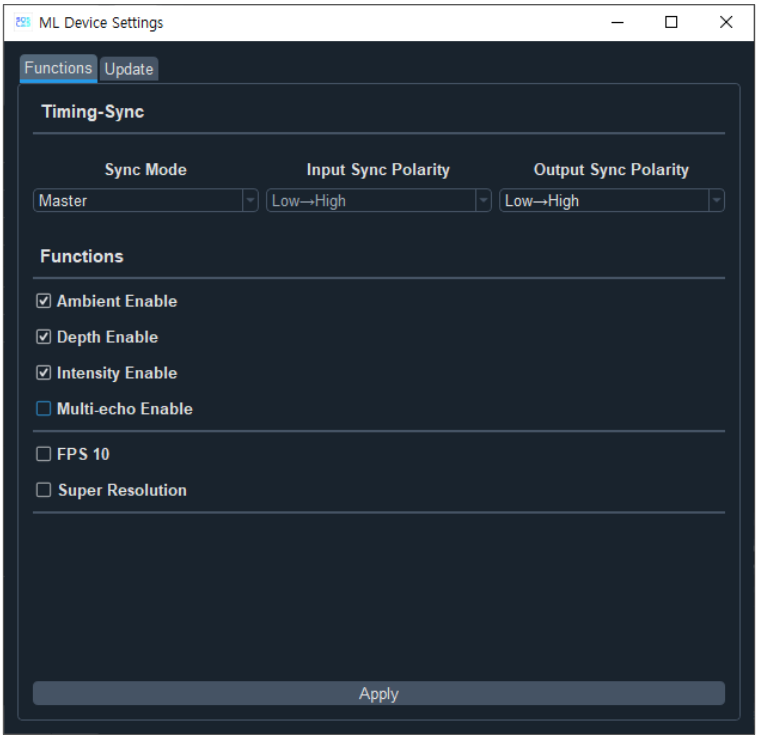
Data Load Window

Once the data loading is complete, user can play the ML-X data using the functionalities provided by the play bar.

2.5 Device Setting

Device setting provides functionalities for the ML-X device, including Timing Sync, Super Resolution on/off, IP change and F/W Update.

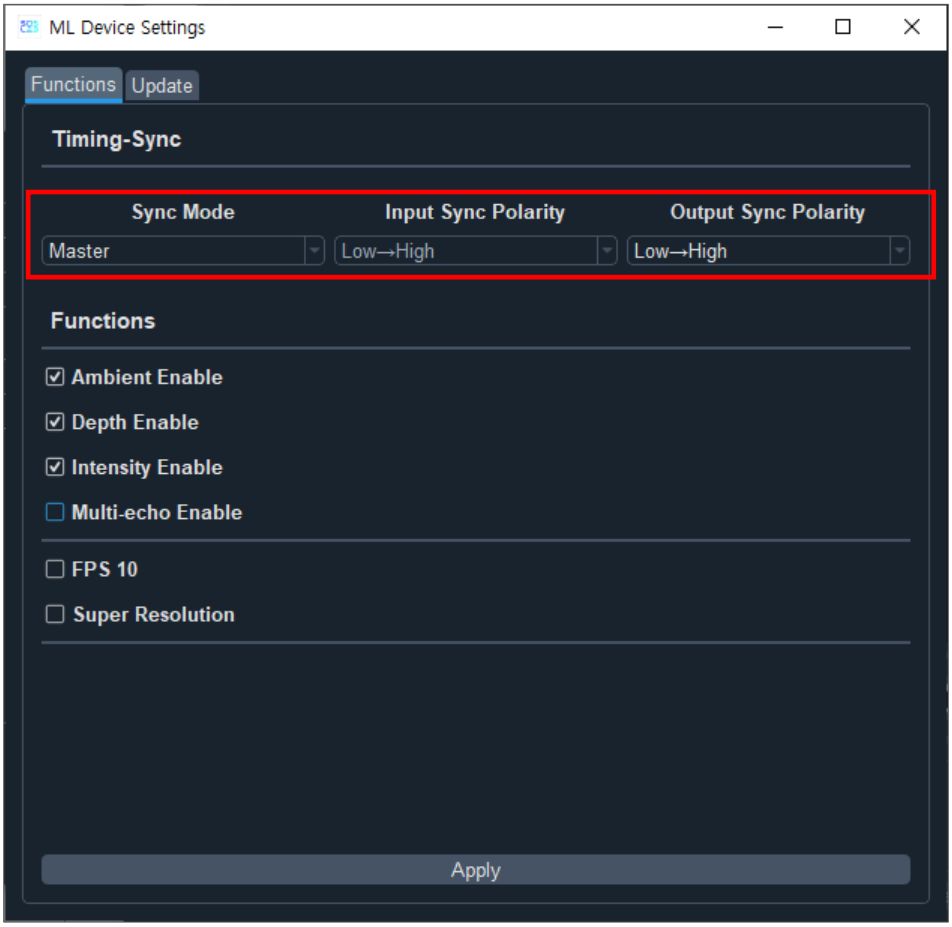
Timing sync and function on/off are provided under the "Functions" tab, while the IP change feature and firmware update functionality are provided under the "Update" tab.



Device Setting Window

2.5.1 Timing Sync

The Timing Sync for the ML-X Device is available under the "Functions" tab. Timing Sync supports three modes: Master mode, Slave mode, and Capture mode. In Master mode, user can configure the Output Sync Polarity, while in Slave mode, user can configure the Input Sync Polarity. After selecting the desired mode and polarity, click the "Apply" button to apply the settings. For more detailed information about the Timing Sync, please refer to pages 9 to 20 of the User Guide.

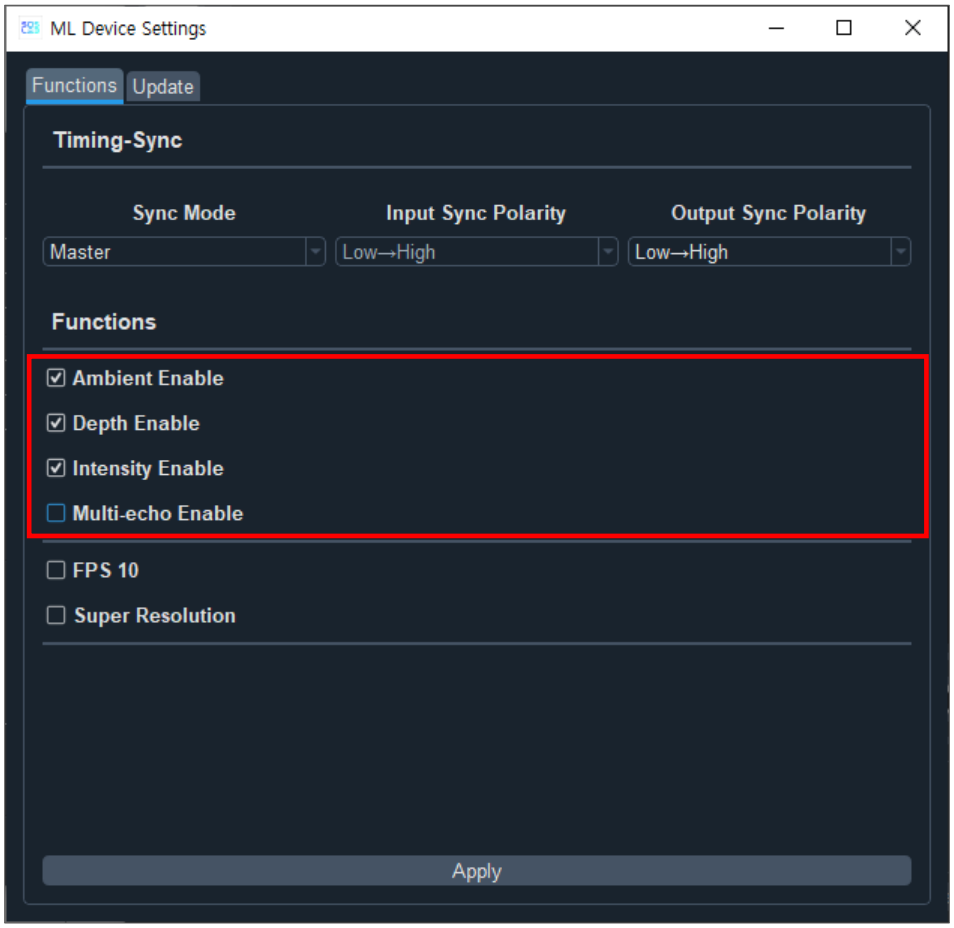


Device Setting Window – Timing sync

2.5.2 Functions on/off

The on/off control for the Ambient, Depth, and Intensity enable of the ML-X Device is available under the "Functions" tab. Each feature has a checkbox button, and when user select the checkbox and click the "Apply" button, the corresponding data will be transmitted. Conversely, if user do not select the checkbox and click the "Apply" button, the corresponding data will not be transmitted. For more detailed information about the Functions, please refer to page 61 of the User Guide.

Functions	Default Setting
Ambient Enable	On
Depth Enable	On
Intensity Enable	On
Multi-echo Enable	Off

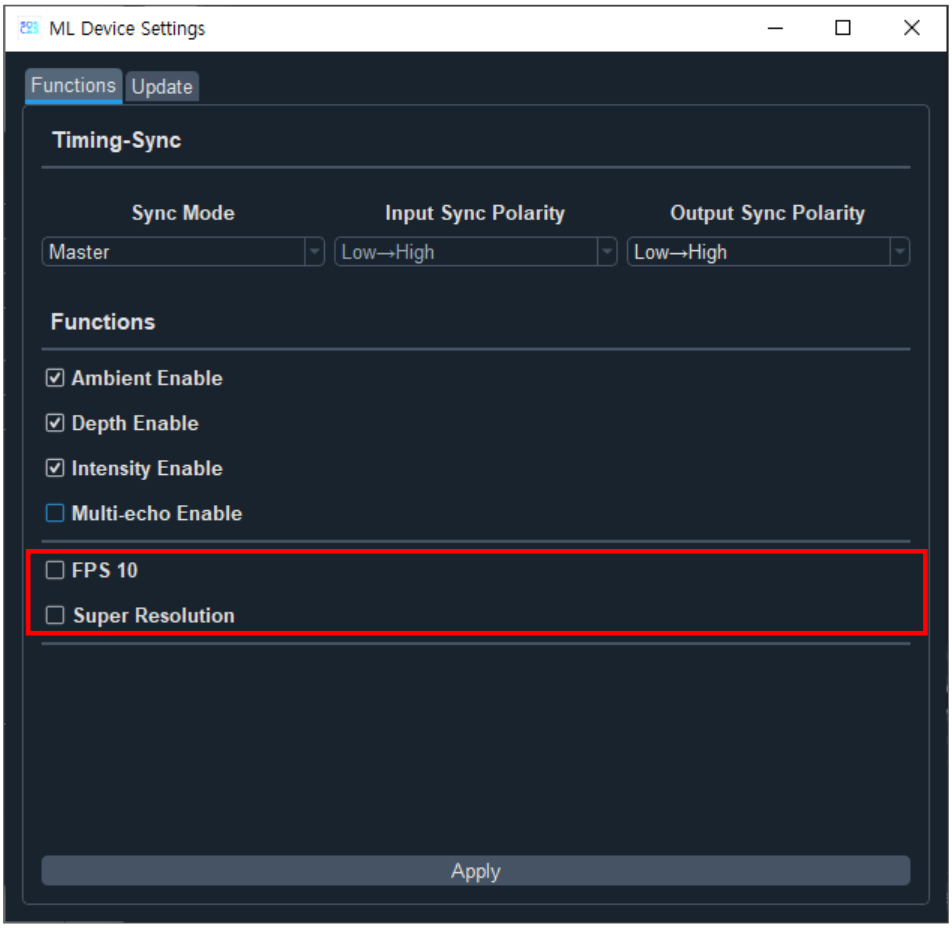


Device Setting Window – Functions On/Off

2.5.2 Functions on/off

The FPS 10 and Super Resolution for the ML-X Device is available under the "Functions" tab. Each feature has a checkbox button, and when user select the checkbox and click the "Apply" button, the feature will be turned on. Conversely, if user do not select the checkbox and click the "Apply" button, the feature will be turned off. For more detailed information about the Functions, please refer to page 60 of the User Guide.

Functions	Default Setting
FPS 10	Off
Super Resolution	Off



Device Setting Window – Functions On/Off

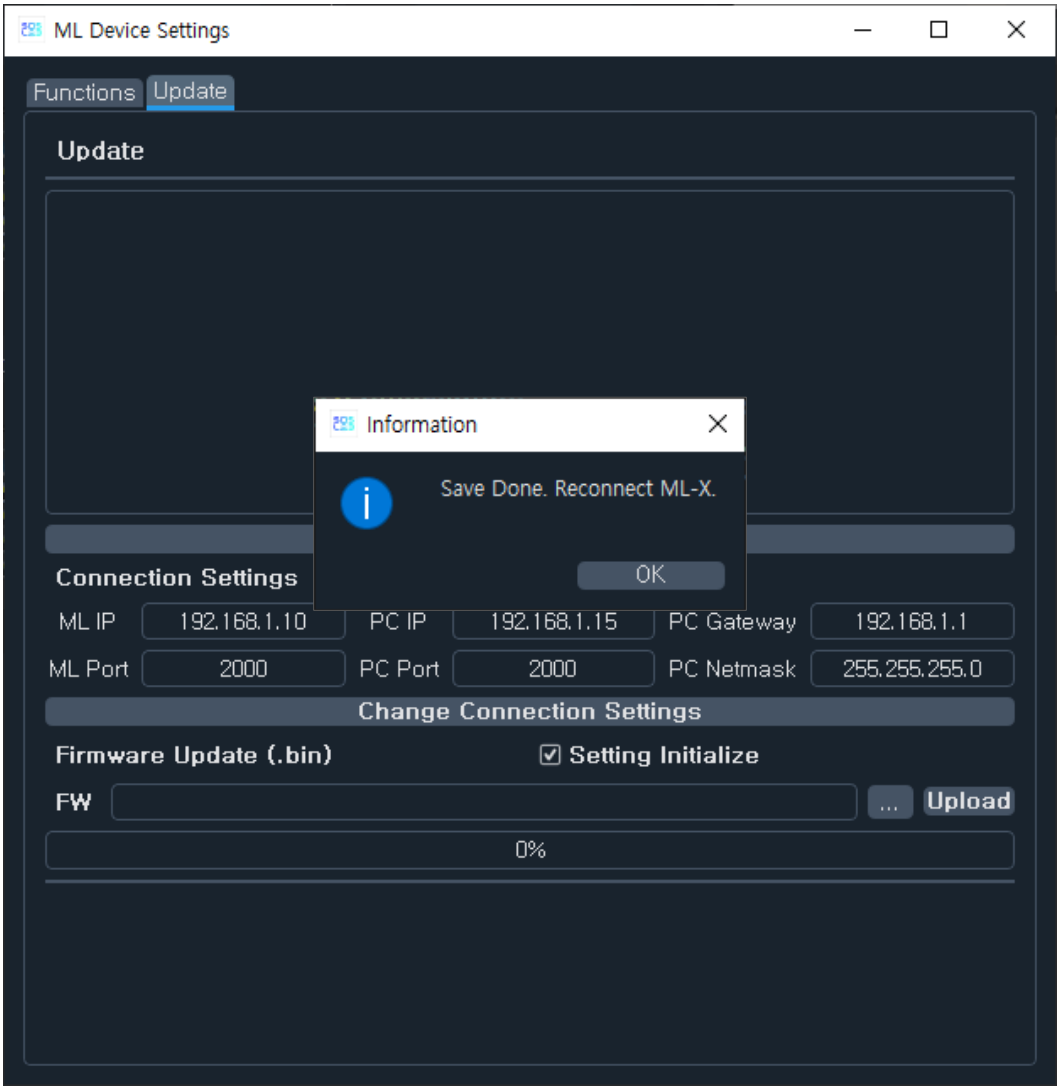
2.5.3 IP Changer

The IP change feature for the ML-X Device is available under the "Update" tab. User can perform the IP change for the ML-X Device using the following steps:

① Enter the new IP for the ML-X Device that the user want to change.

② Click the "Change Connection Settings" button.

③ After the change, reconnect the power cable for the ML-X Device.



IP Changer Result

2.5.4 F/W Update

The F/W Update feature for the ML-X Device is available under the "Update" tab. User can perform the F/W Updates for the ML-X Device using the following steps:

① Click the "…" button.

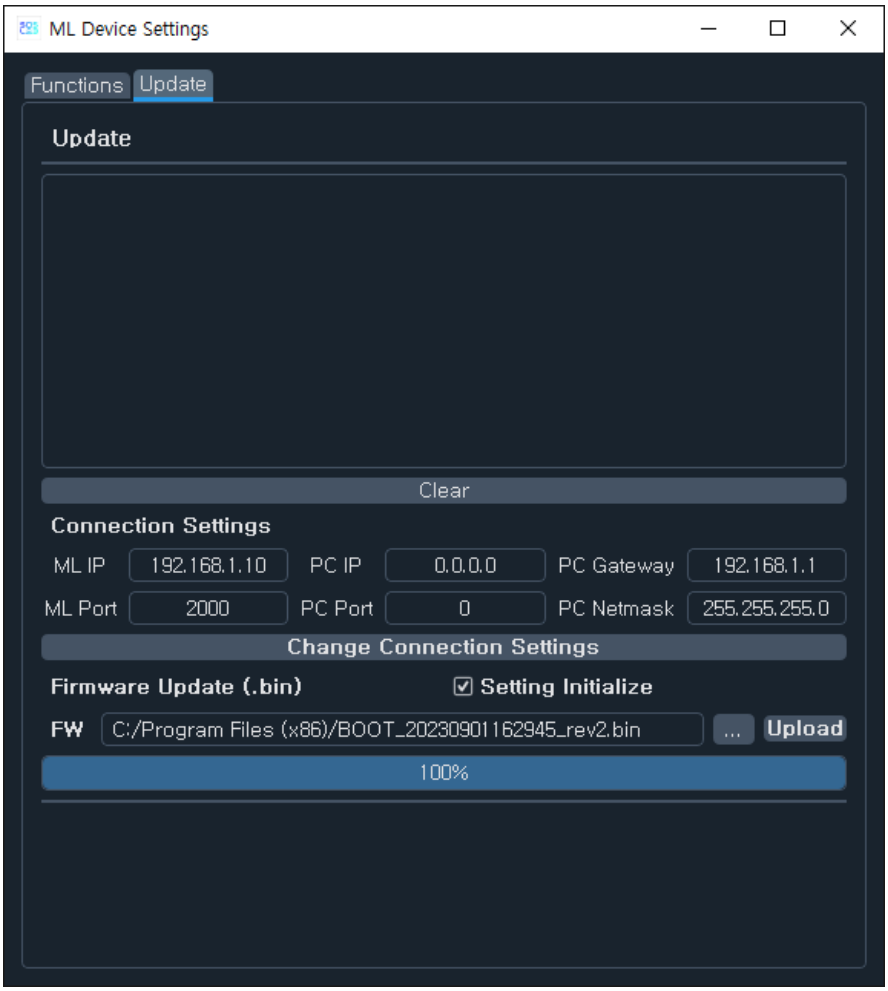
② Select the F/W update files (*.bin).

③ Click the "Upload" button.

④ When the progress bar displays 100%, the F/W Update is complete.

⑤ Reconnect the power cable after the F/W Update is finished.

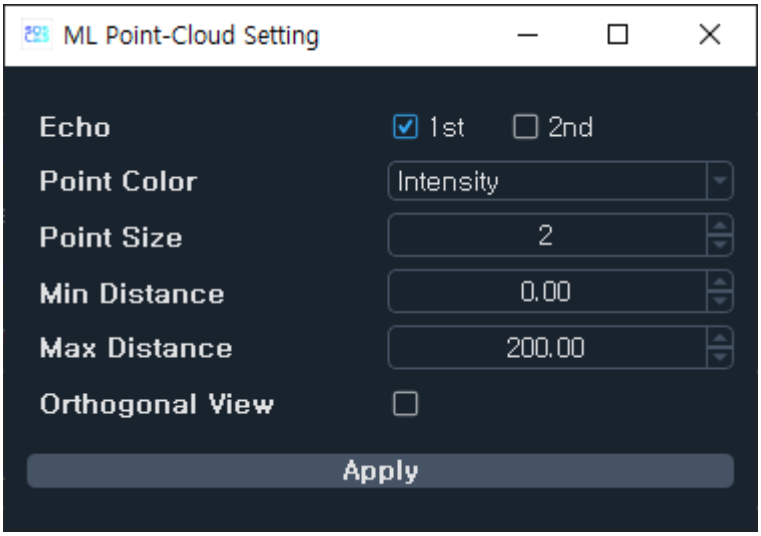
* The F/W Update feature is only provided on the Windows version.



F/W Update Result

2.6 Point Cloud Setting

In the Point Cloud Setting, user can configure the Point Cloud Viewer located in the center of SOS Studio. When user click on the fourth button on the left in SOS Studio labeled "Point Cloud Setting," a popup window will appear as shown below.



Point Cloud Setting Window

The following is an explanation of the available options for Point Cloud Setting.

List	Description
Echo	Visible echo (Generated when multi-echo is enabled.)
Point Color	Point Color (White, Red, Green, Blue, Ambient, Depth, Intensity)
Point Size	Point Size (0~5)
Min Distance	Minimum distance for visible points.
Max Distance	Maximum distance for visible points.
Orthogonal View	Orthogonal View Enable

2.7 Image Setting

In the Image Setting, user can configure the Image Viewer located at the top of SOS Studio. When user click on the fifth button on the left in SOS Studio labeled "Image Setting," a popup window will appear as shown below.

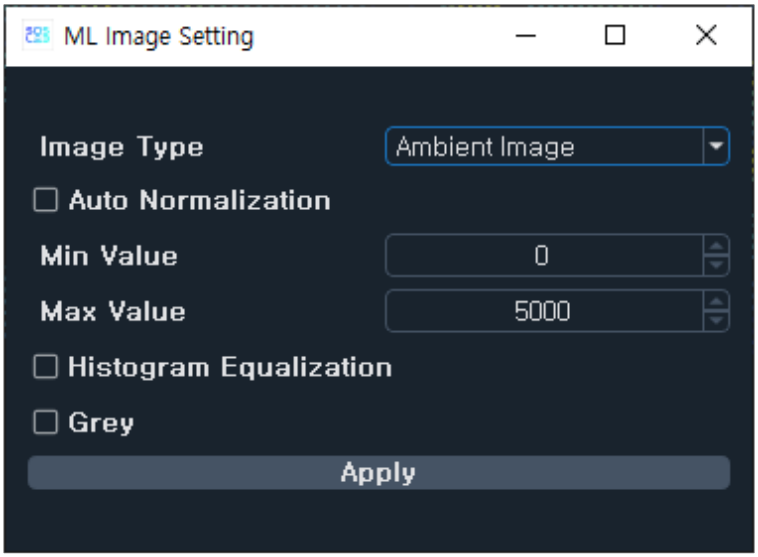


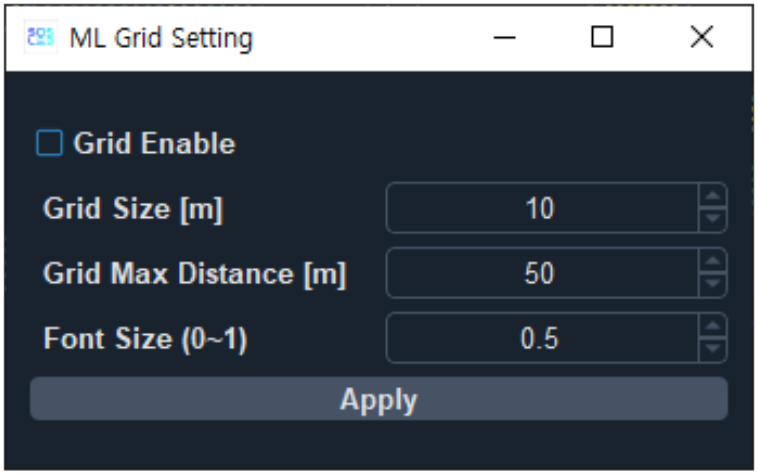
Image Setting Window

The following is an explanation of the available options for Image Setting.

List	Description
Image Type	Select Image (Ambient / Depth / Intensity Image)
Auto Normalization	Normalize the image based on its minimum/maximum values
Min Value	Minimum value for normalization
Max Value	Maximum value for normalization
Histogram Equalization	Histogram equalization enable
Grey	Convert to grey image

2.8 Grid Setting

In the Grid Setting, user can configure the grid for the Point Cloud Viewer located at the center of SOS Studio. When user click on the sixth button on the left in SOS Studio labeled "Grid Setting," a popup window will appear as shown below.



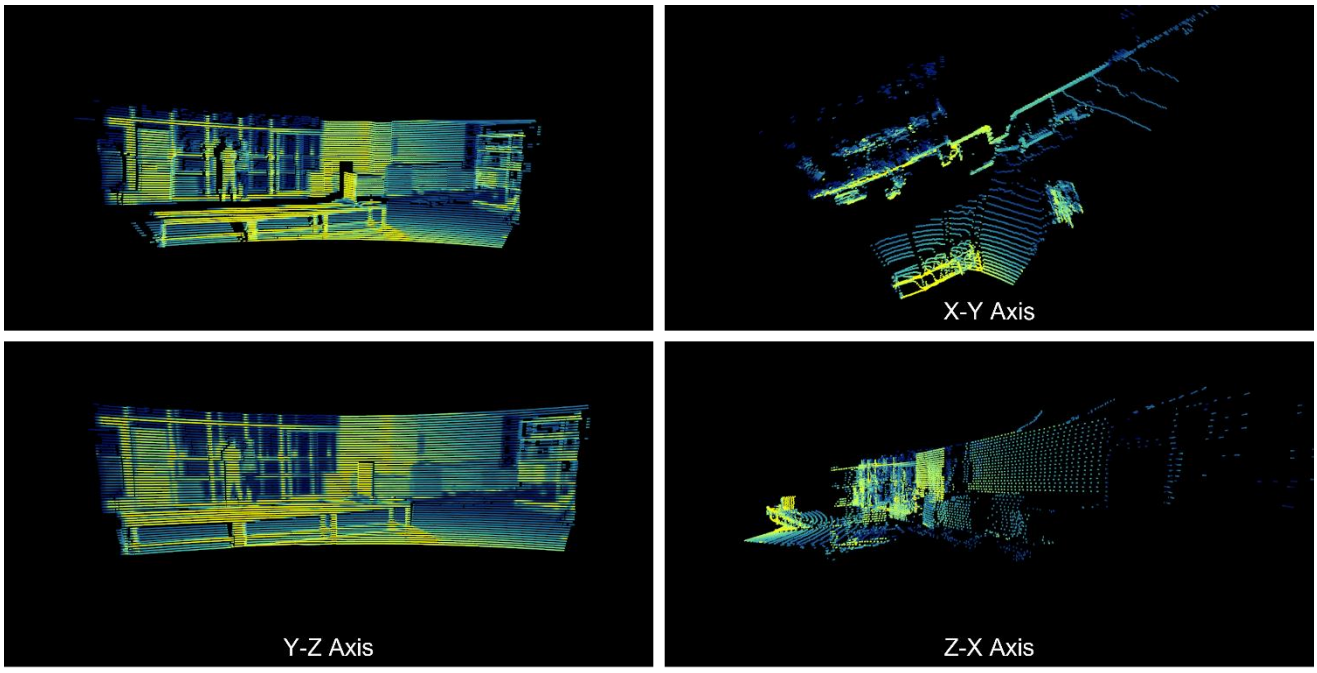
Grid Setting Window

The following is an explanation of the available options for Grid Setting.

List	Description
Grid Enable	Enable grid visualization
Grid Size	Grid spacing size
Grid Max Distance	Grid maximum distance
Font Size	Grid distance display text size

2.9 X-Y / Y-Z / Z-X Axis

 X-Y /  Y-Z /  Z-X Axis provides a functionality to adjust the perspective of the Point Cloud Viewer located at the center of SOS Studio according to the specified axis. When user select each of these axis, the Point Cloud Viewer will be adjusted to the corresponding axis from a viewpoint.



Changing View-Point by Selecting X-Y / Y-Z / Z-X Axis Button

2.10 Play / Record Mode

The Play Bar located below the Point Cloud Viewer provides functionalities for playing ML-X data loaded in the **2.4 Data Load** and recording the data.

SOS Studio Play Bar

The following is an explanation of the available options for Play Bar.

List	Description
Play	Playback of data
Pause	Pause the playback of data
Stop	Stop the playback of data
Repeat	Repeat the playback of data
Scroll Bar	Display of the currently playing frame within the overall frames
Frame	
Speed	Speed of playback
Record	Recording continuously streaming data (.bin)
Capture	Capturing single frame streaming or playback data (.png, .pcd)
FPS	Frame frequency of streaming data

Chapter 3

ML-X LiDAR API

3.1. ML-X API C++ Code Build from Source

The ML-X SDK is an open-source code and can download it through the Github link. (Github link: https://github.com/SOSLAB-github/ML-X_SDK)

The environment required for build the ML-X API is as follows.

- Window 10 / 11
- Ubuntu 18.04 / 20.04 / 22.04
- C++ 11 (GCC 5 / Visual C++ 13) or later
- CMake 3.10 or later

3.1.1. ML-X API C++ Building on Windows

User can generate a Visual Studio Solution file by entering the following command.

```
$ git clone https://github.com/SOSLAB-github/ML-X_SDK.git
$ cd ML-X_SDK/MLX_API/
$ cmake CMakeLists.txt
```

After entering the command, a "libsoslab.sln" will be generated in the folder. Run this solution and change the build type to Release. Build the ALL_BUILD project. Once the build is complete, the following files will be generated in the output/Release folder: "libsoslab_core.lib", "libsoslab_core.dll", "libsoslab_ml.lib", "libsoslab_ml.dll", and "test_ml.exe".

3.1.2. ML-X API C++ Building on Linux

User can build the code by entering the following command.

```
$ git clone https://github.com/SOSLAB-github/ML-X_SDK.git
$ cd ML-X_SDK/MLX_API/
$ cmake CMakeLists.txt -DCMAKE_BUILD_TYPE=Release
$ make
```

After the build is completed, the following files will be generated in the output/Release folder: "libsoslab_core.so", "libsoslab_ml.so", and "test_ml".

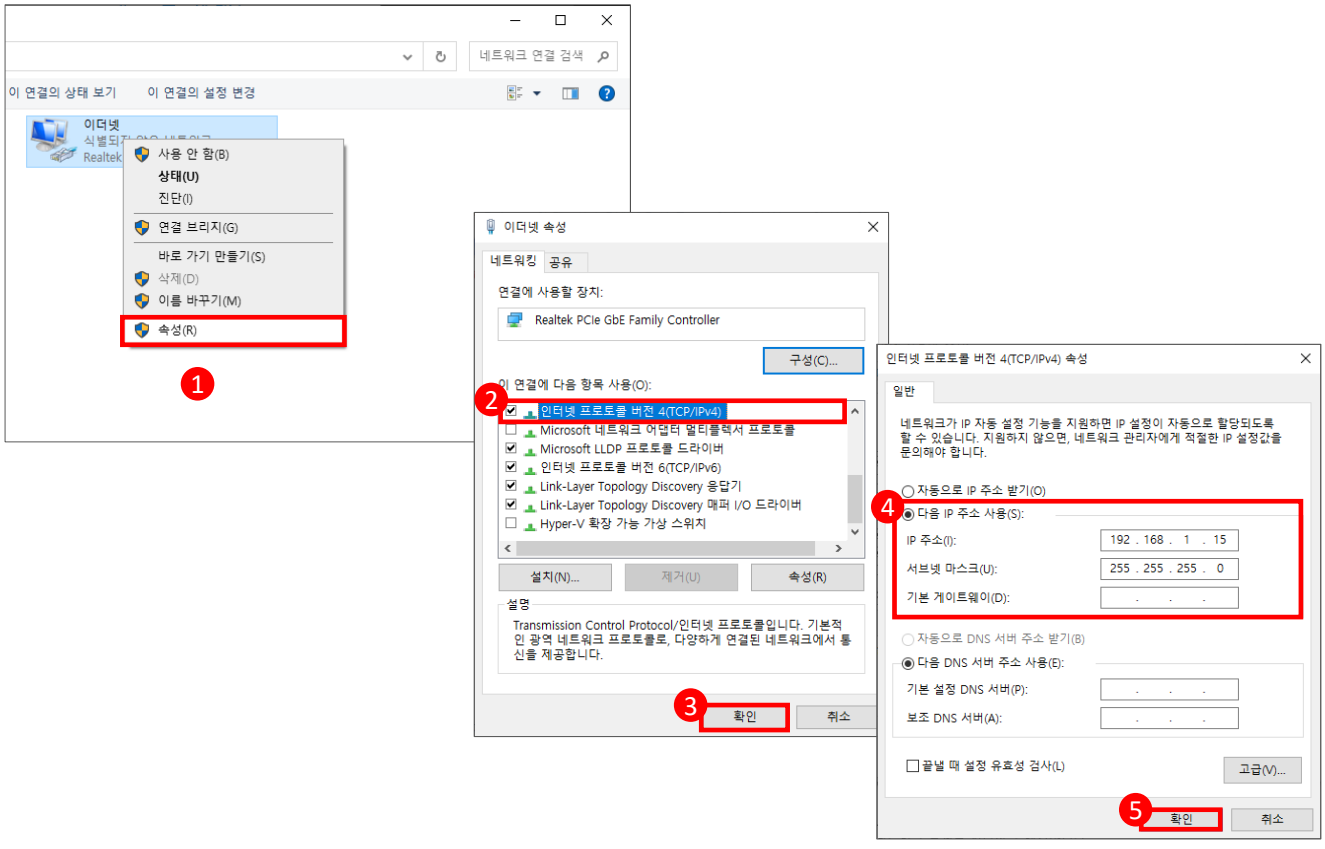
3.1.2. Running the Example Code

After building the source code, user can run the "test_ml" program to save ML-X's point cloud data as a ".csv" file. Before running the program, user need to configure the network settings of the host connected to ML-X.

3.1.2.1. Network Setting on Windows

Open Control Panel and go to Network and Internet. Click on Network Connections. Right-click on the Ethernet connection associated with ML-X and click on the Properties button. Click on Internet Protocol Version 4 (TCP/IPv4), then click on the Properties button. Choose "Use the following IP address" and configure the following settings:

- IP Address: 192.168.1.15
- Subnet Mask : 255.255.255.0

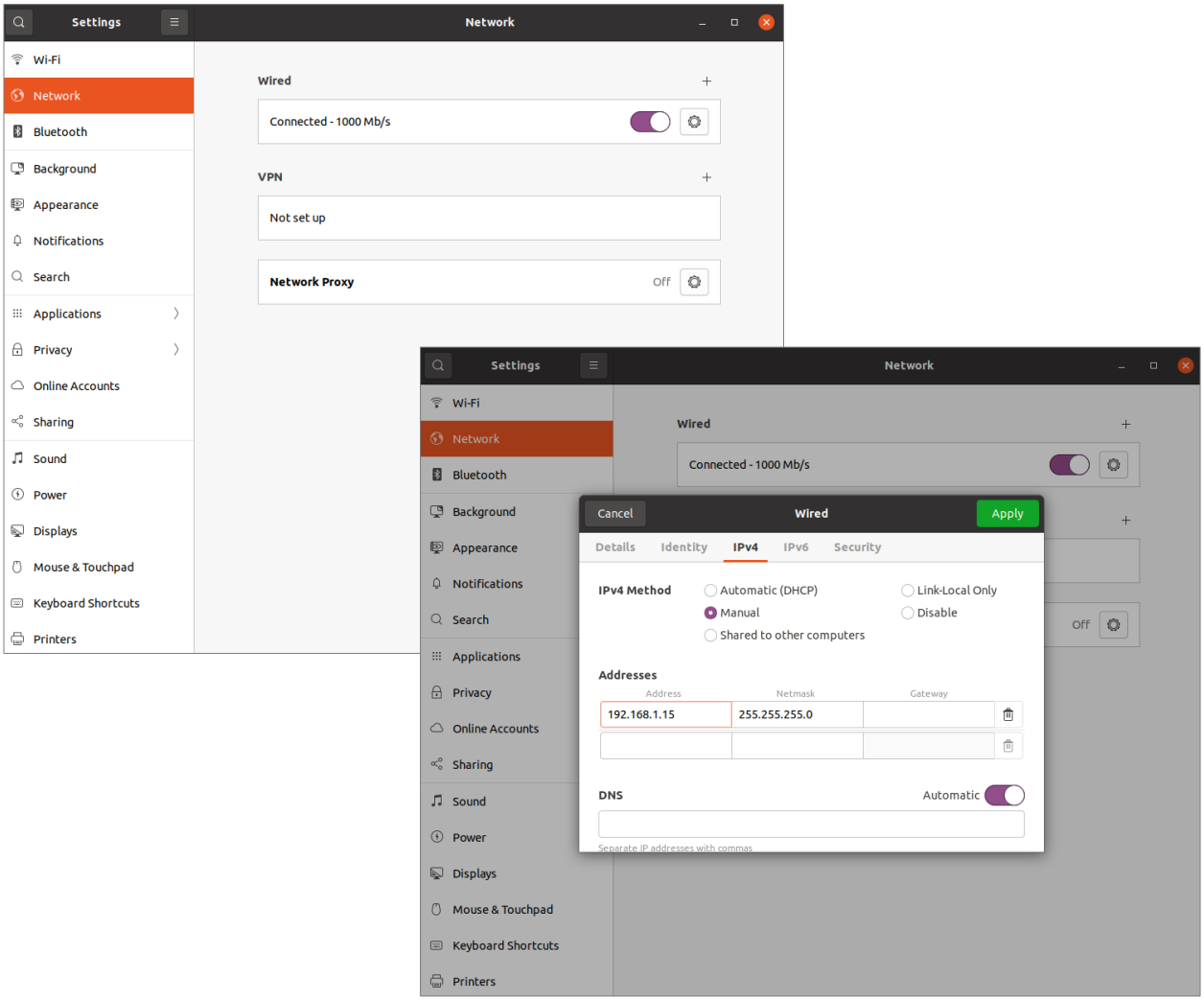


IPv4 Setting @Windows

3.1.2.2. Network Setting on Linux

Open the Settings window and navigate to the Network section. Modify the IPv4 settings as follows:

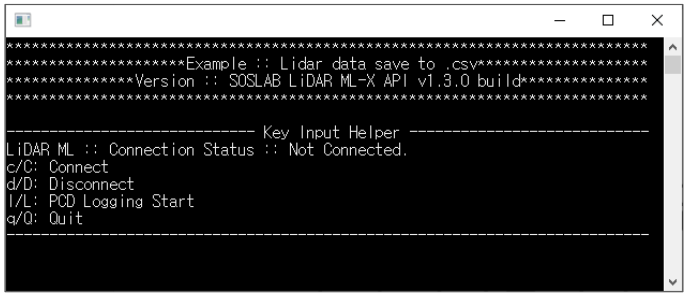
- IPv4 Method – Manual
- Address : 192.168.1.15
- Netmask : 255.255.255.0



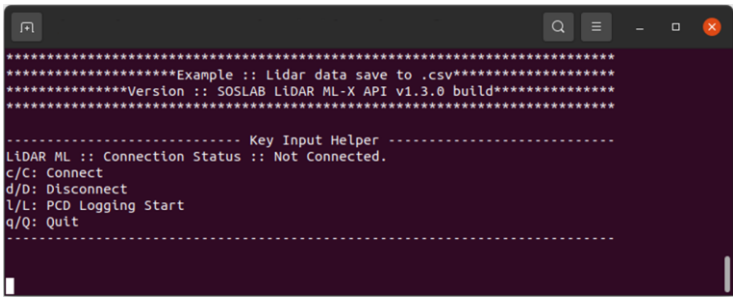
IPv4 Setting @Linux

3.1.2.3. Running the Example Code

Open the terminal (Command Prompt in Windows) and execute the "test_ml" file. When user run the file, user can see the following message displayed.



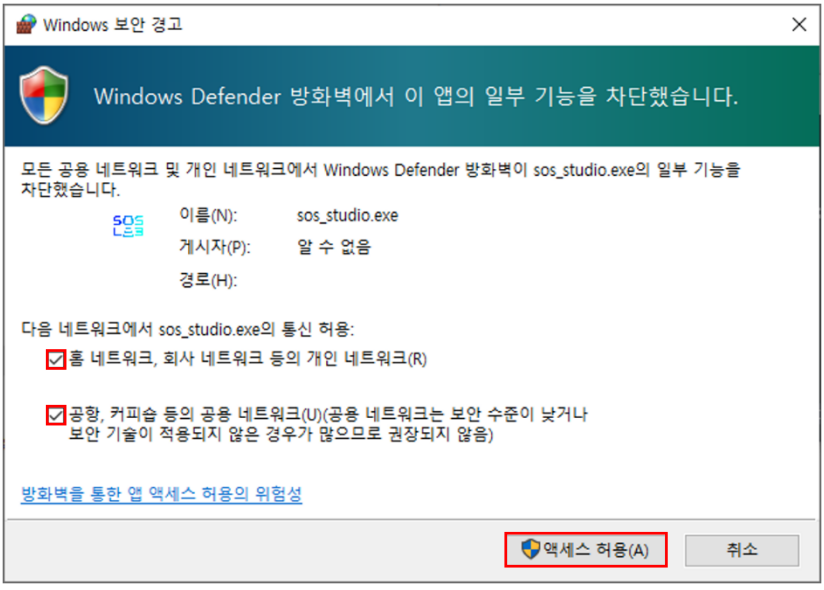
Windows



Linux

- LiDAR ML :: Connection Status :: Not Connected. → ML-X Status
- c/C + enter → Connect ML-X
- d/D + enter → Disconnect ML-X
- l/L + enter → Save the PCD data(.csv)
- q/Q + enter → Quit the Program

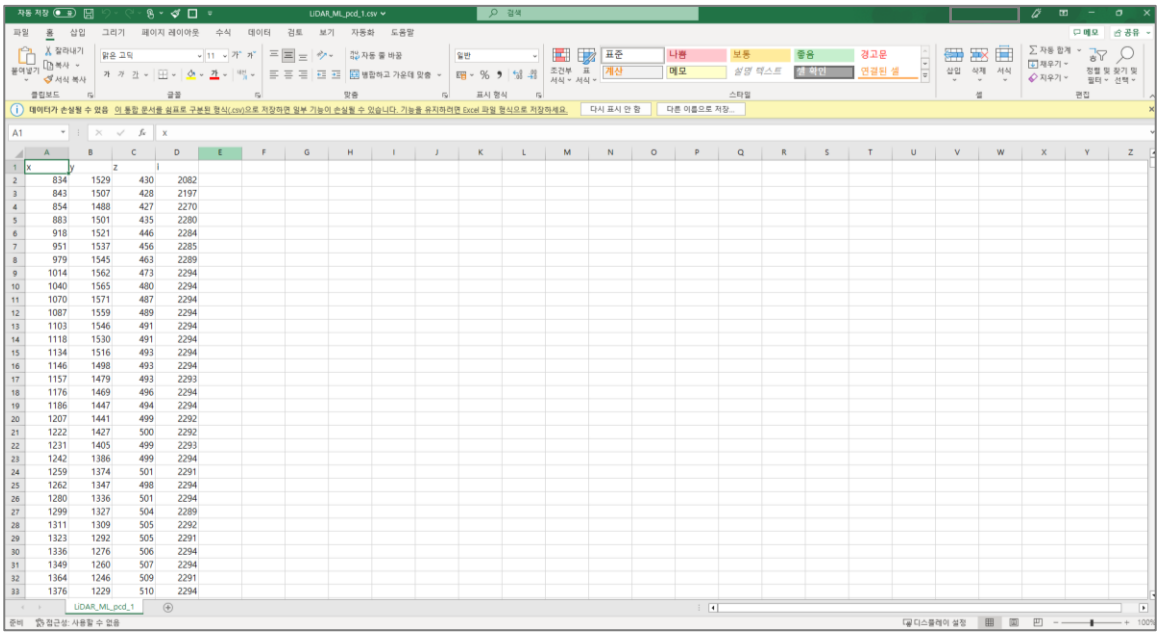
When initially connecting the PC and ML-X, user may encounter a Windows Security Alert window as follows. Check the two checkboxes as shown in the image below, then click the "Allow access (A)" button.



Windows Security Alert window

3.1. ML-X API C++ Code Build from Source

When saving point cloud data as a '.csv' file, it will be stored in the following format:



x	y	z	I
834	1529	430	2082
843	1507	428	2197
854	1488	427	2270
883	1501	435	2280
918	1521	446	2284
951	1537	456	2285
979	1545	463	2289
1014	1562	473	2294
1040	1565	480	2294
1070	1571	487	2294
1087	1559	489	2294
1103	1546	491	2294
1118	1530	491	2294
1134	1516	493	2294
1146	1498	493	2294
1157	1479	493	2293
1176	1469	496	2294
1186	1447	494	2294
1207	1441	499	2292
1222	1427	500	2292
1231	1405	499	2293
1242	1386	499	2294
1259	1374	501	2291
1262	1347	498	2294
1280	1336	501	2294
1299	1327	504	2289
1311	1309	505	2292
1323	1292	505	2291
1336	1276	506	2294
1349	1260	507	2294
1364	1246	509	2291
1376	1229	510	2294

The first column contains the x values, the second column contains the y values, the third column contains the z values, and the fourth column contains the Intensity data.

3.2. ML-X API Python Code

The environment required for build the ML-X API Python is as follows.

- Python 3.7 or later

3.2.1. ML-X API Python

The path for the ML-X Python API is as follows.

- Path : MLX_API/pylibsoslab/libsoslab_ml.py

User can use the Python API by copying the file to the code execution location or importing it using the above path.

3.2.2. ML-X API Python Example Code

The path for the ML-X Python example code is as follows.

- Path : MLX_API/examples/python_ml/test_ml.py

The above example file is a code that visualizes Ambient/Depth/Intensity images transmitted by ML-X, and it requires the following library.

- OpenCV

The library can be installed using the following command.

```
$ pip install opencv-python
```

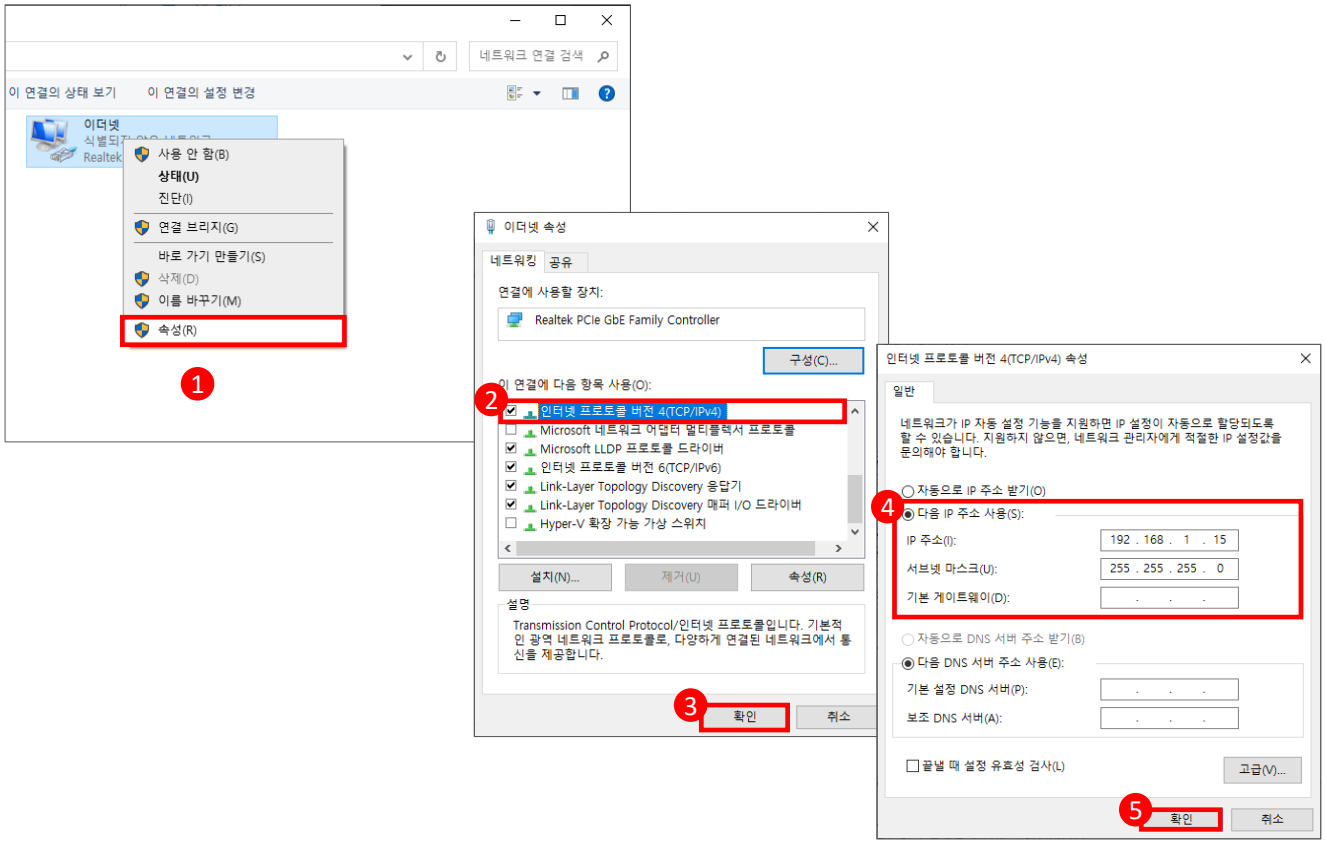
3.2.3. Running the Example Code

After install the wheel file, user can run the "test_ml.py" program to visualize ML-X's ambient/depth/intensity image data. Before running the example program, user need to configure the network settings of the host connected to ML-X.

3.2.3.1. Network Setting on Windows

Open Control Panel and go to Network and Internet. Click on Network Connections. Right-click on the Ethernet connection associated with ML-X and click on the Properties button. Click on Internet Protocol Version 4 (TCP/IPv4), then click on the Properties button. Choose "Use the following IP address" and configure the following settings:

- IP Address: 192.168.1.15
- Subnet Mask : 255.255.255.0

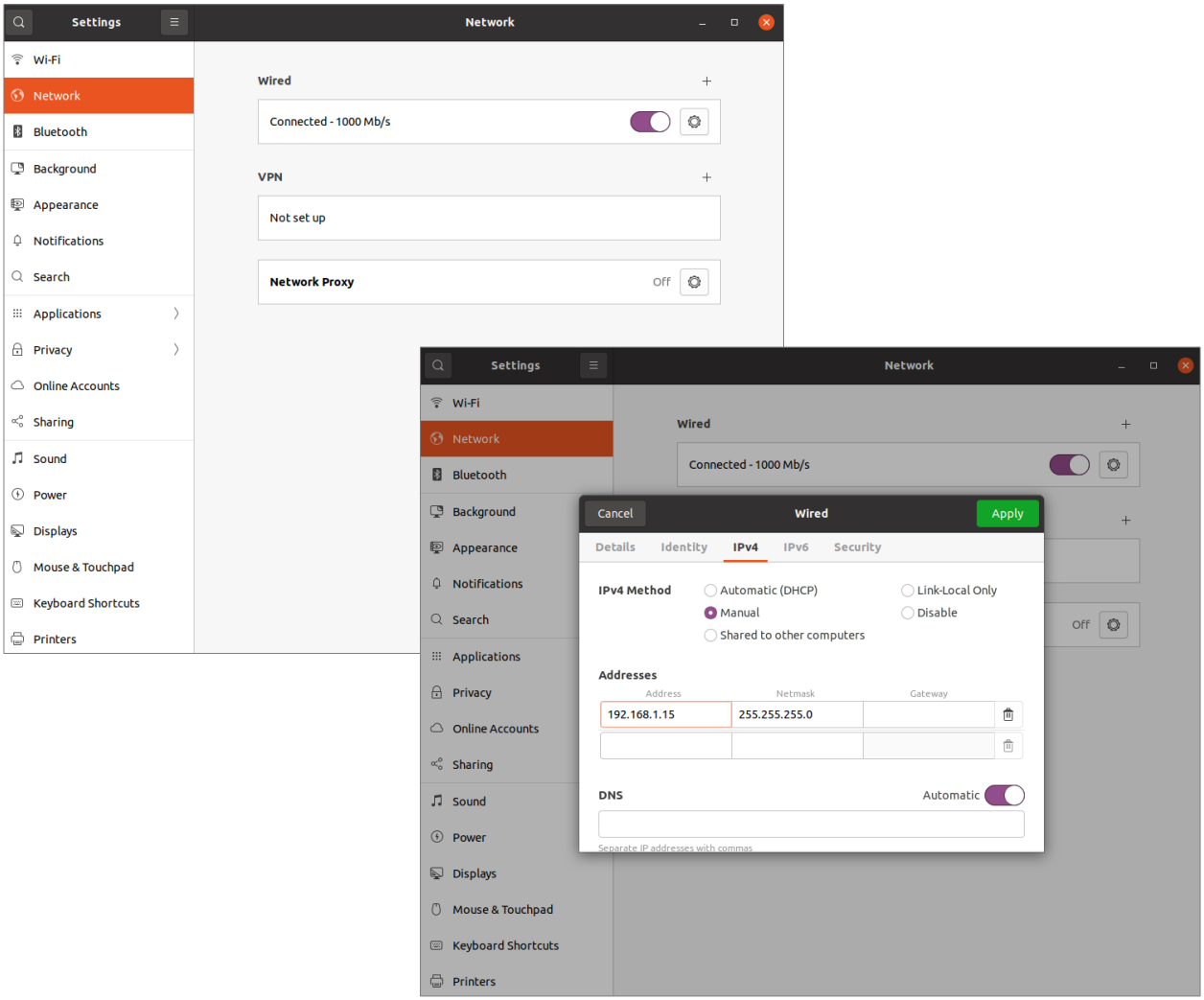


IPv4 Setting @Windows

3.2.3.2. Network Setting on Linux

Open the Settings window and navigate to the Network section. Modify the IPv4 settings as follows:

- IPv4 Method – Manual
- Address : 192.168.1.15
- Netmask : 255.255.255.0

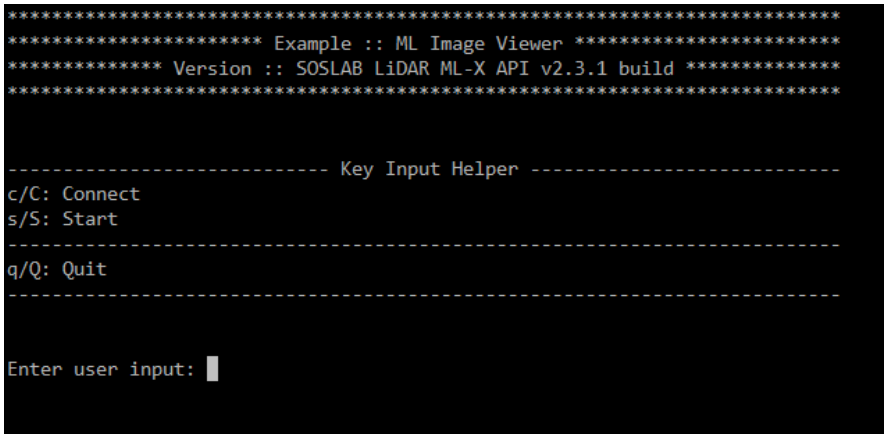


IPv4 Setting @Linux

3.2.3.3. Running the Example Code

User can execute the “test_ml.py” by entering the following command.

```
$ cd MLX_API/examples/python_ml/
$ python test_ml.py
```

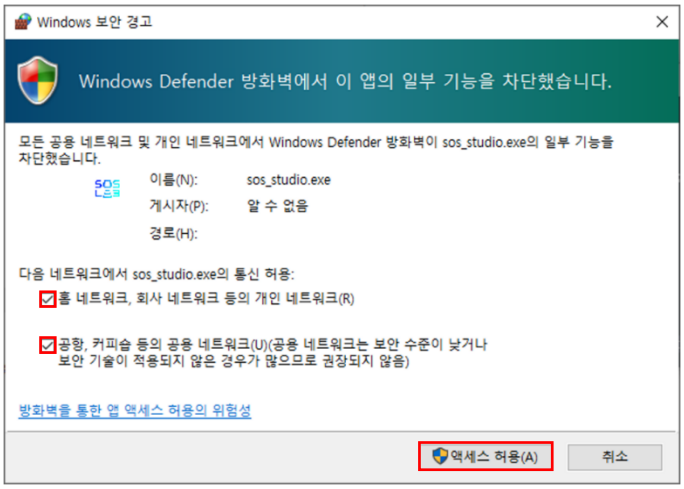


test_ml.py execution

When the user executes test_ml.py, the upper image is displayed, and user can enter the command below:

- c/C + enter → Connect ML-X
- d/D + enter → Disconnect ML-X
- s/S + enter → Visualize ML-X’s data
- q/Q + enter → Quit the Program

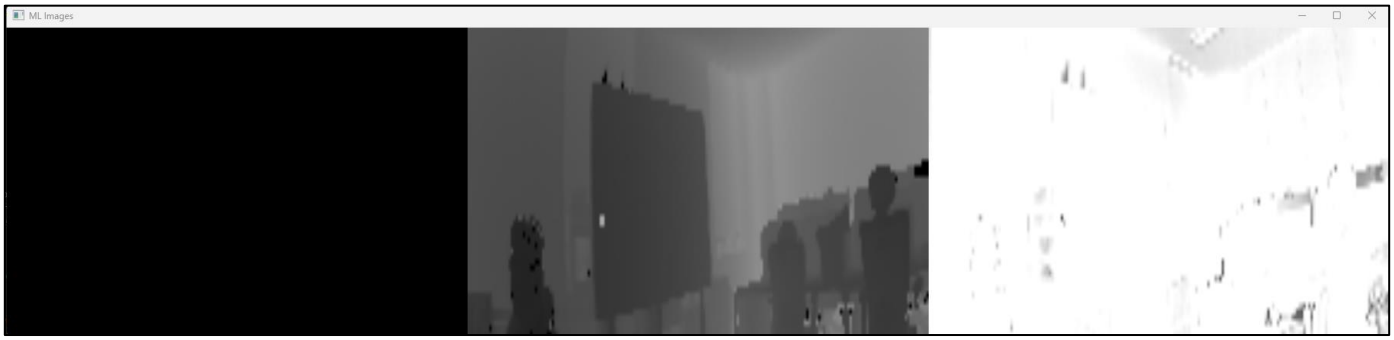
When initially connecting the PC and ML-X, user may encounter a Windows Security Alert window as follows. Check the two checkboxes as shown in the image below, then click the "Allow access (A)" button.



Windows Security Alert window

3.2. ML-X API Python Code

When the user runs the connection to ML-X and executes data visualization, the user can visualize the image data output from ML-X as shown in the image below.



ML-X Data Visualization (Left : Ambient / Middle : Depth / Right : Intensity)

3.2.4. Example Code for Python

The example code includes ML-X connection and data visualization, and the code explanation is as follows.

1) ML-X connection

```
1. lidar_object = libsoslab_ml.LidarML()
2. lidar_data = libsoslab_ml.Scene()

3. ml_ip_address = "192.168.1.10"
4. ml_port_number = 2000
5. lidar_object.connect(ml_ip_address, ml_port_number)
```

Code of ML-X connection

Here is the code for declaring variables and connecting to the ML-X Device using the Python API. Detailed descriptions of the LidarML / Scene classes are provided in the Appendix.

Line	Description
1	Create the LidarML class from the Python API (libsoslab_ml) for communication with ML-X.
2	Create the Scene class from the Python API (libsoslab_ml) to store ML-X data.
3-4	Define the IP (ml_ip_address) and Port number (ml_port_number) variables for ML-X. In the example code, default values are used.
5	Connect to ML-X using the IP/Port numbers defined in lines 3 and 4 as input variables.

3.2.4. Example Code for Python

2) Acquiring raw data

```
1. lidar_object.run()
2. while True:
3.     lidar_data = lidar_object.get_scene()
4.     if lidar_data is not None:
5.         pcd_idx = 0
6.         print(f"Point X(mm): {lidar_data.pointcloud[pcd_idx].x} / Point Y(mm):
    {lidar_data.pointcloud[pcd_idx].y} Point Z(mm): {lidar_data.pointcloud[pcd_idx].z}")
7.         ambient = np.reshape(lidar_data.ambient_image, (lidar_data.rows, 576))
8.         depth = np.reshape(lidar_data.depth_image, (lidar_data.rows, lidar_data.cols))
9.         intensity = np.reshape(lidar_data.intensity_image, (lidar_data.rows,
    lidar_data.cols))
```

Code of acquiring raw data

Here is the code for obtaining raw data from the ML-X Device. Detailed descriptions of the Scene class are provided in the Appendix.

Line	Description
1	Start the ML-X operation using the run function of the LidarML class.
3	Obtain raw data into the lidar_data variable using the get_scene function of the LidarML class.
5-6	Here is the code to access the PointCloud of the raw data. The PointCloud is defined as a List variable named pointcloud in the Scene class. Define the desired index in the pcd_idx variable, and then access the x, y, z data by accessing the lidar_data.pointcloud List variable.
7-8	Here is the code to convert the raw data of Ambient / Depth / Intensity to Numpy variables. Each data type is defined as a List variable named ambient_image, depth_image, and intensity_image in the Scene class.

3.3. ML-X ROS Example Code Build

The ML-X ROS Example Code is compatible with Ubuntu 18.04/20.04 and ROS Melodic / Noetic. It provides code that allows user to run ML-X in a ROS environment.

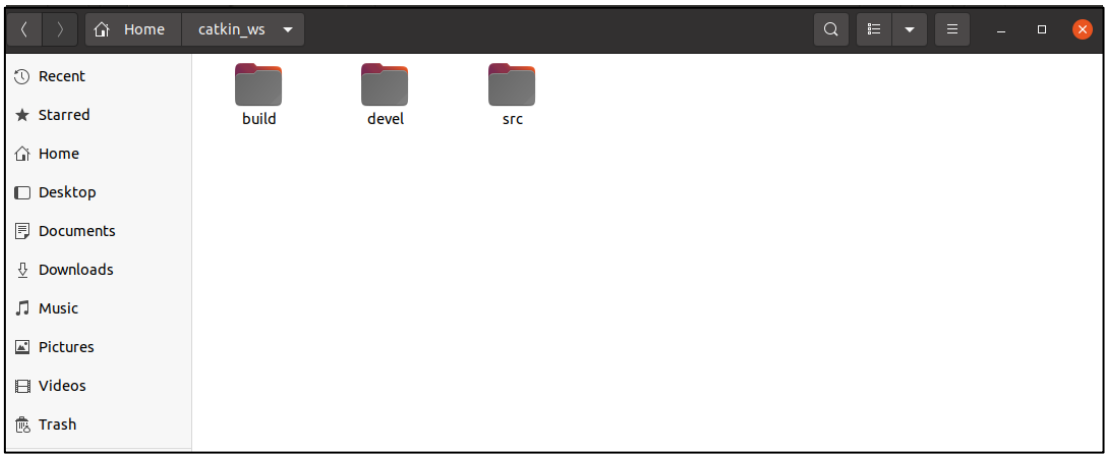
3.3. ML-X ROS Example Code Build

1) Enter the following command in the catkin_ws folder to create the 'ml' package.

```
$ catkin_make
```

```
[100%] Linking CXX executable /home/soslab/catkin_ws/devel/lib/ml/ml
[100%] Built target ml
soslab@soslab-17U790-PA76K:~/catkin_ws$
```

Build ml Package using catkin_make



Build Result of ml Package

2) To add the ml package to ROS environment after building it, enter the following command.

```
$ source ~/catkin_ws/devel/setup.sh
$ rospack find ml
```

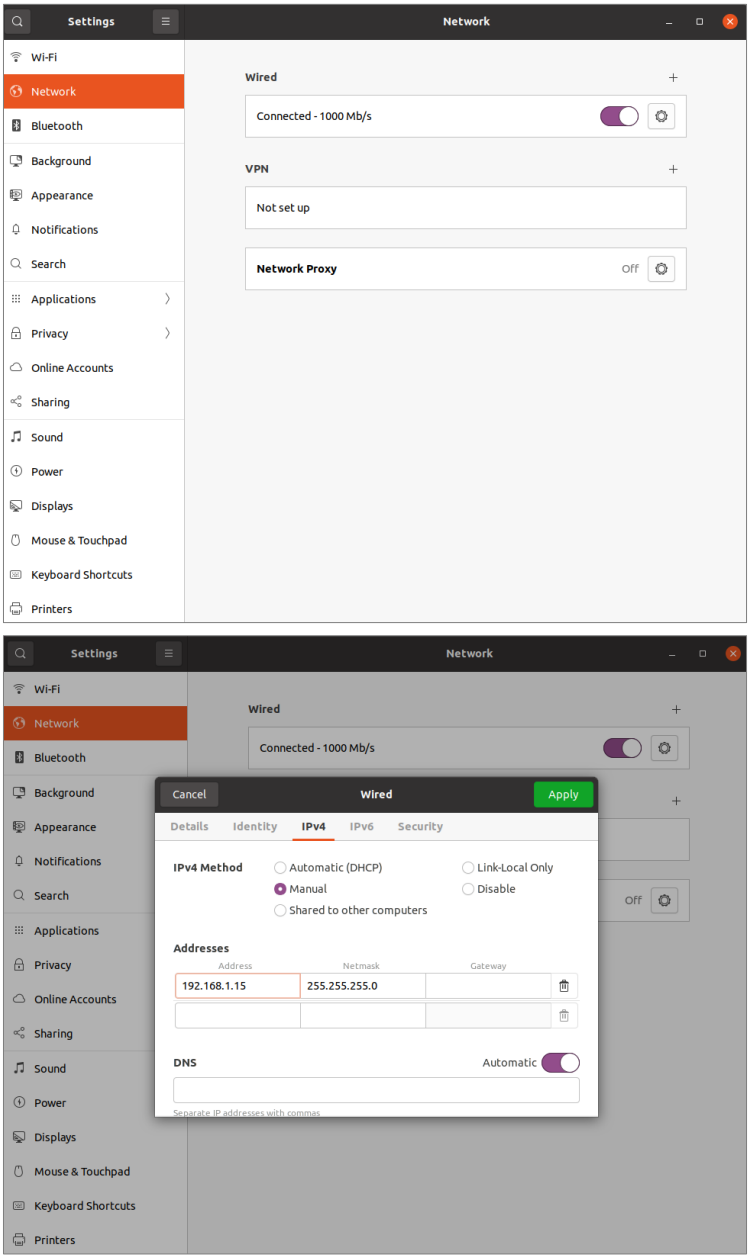
```
soslab@soslab-17U790-PA76K: ~/catkin_ws
soslab@soslab-17U790-PA76K:~/catkin_ws$ source ~/catkin_ws/devel/setup.sh
soslab@soslab-17U790-PA76K:~/catkin_ws$ rospack find ml
/home/soslab/catkin_ws/src/ml
soslab@soslab-17U790-PA76K:~/catkin_ws$
```

Add ml package to ROS environment

3.3. ML-X ROS Example Code Build

3) To establish the connection between the ML-X device and PC, follow these steps to modify the IPv4 settings in the Network section of the Settings window:

- IPv4 Method – Manual
- Address : 192.168.1.15
- Netmask : 255.255.255.0

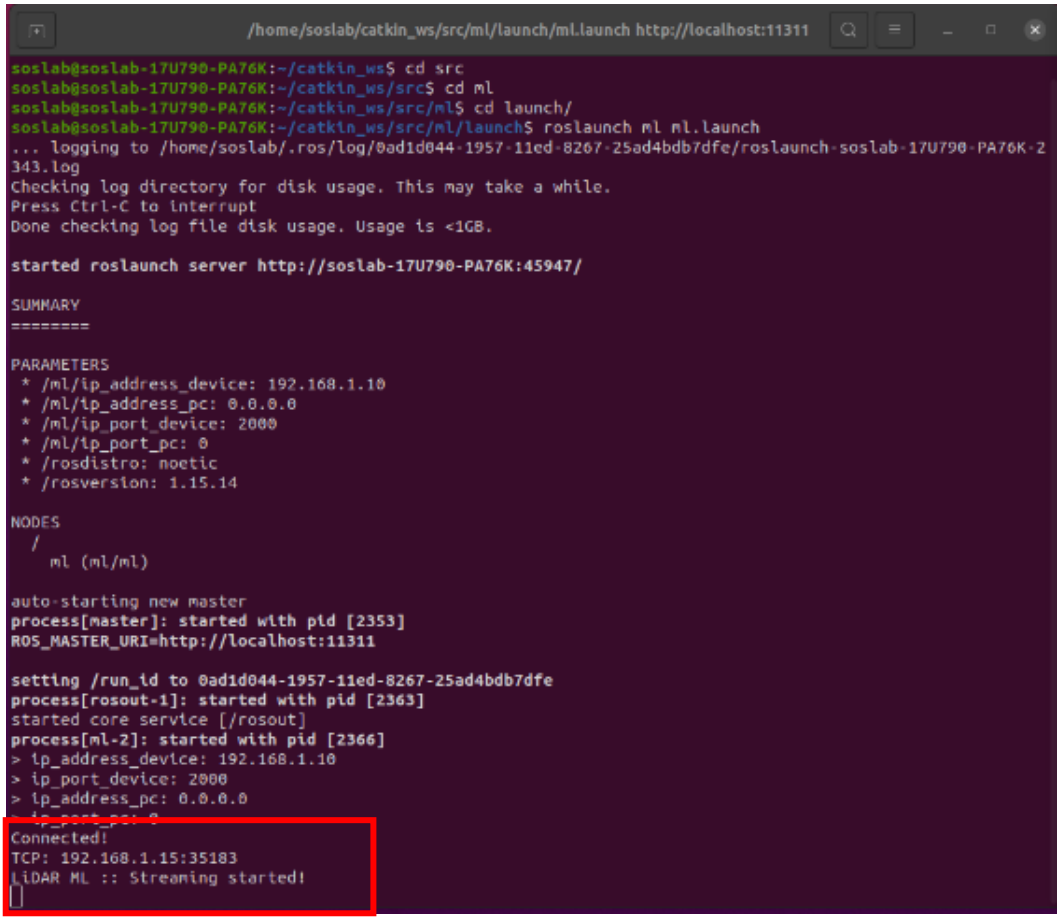


IPv4 Setting @Linux

3.3. ML-X ROS Example Code Build

4) After configuring the network settings, execute the following command to run the ml.launch file.
5) If the connection with the ML-X device is successful, user can see the message "Lidar ML :: Streaming started!", as shown in the provided image.

```
$ cd ~/catkin_ws/src/ml/launch
$ roslaunch ml.launch
```



Connecting and Visualizing ML-X through ROS in Ubuntu

3.4. Example Code for Ubuntu/ROS

The example code includes ML-X connection and data visualization in Rviz, and the code explanation is as follows.

1) Acquiring raw data

```
1.  lidar_ml->register_scene_callback(ml_scene_data_callback, nullptr);
2.  void ml_scene_data_callback(void* arg, SOSLAB::LidarML::scene_t& scene) {
3.      std::vector<uint32_t> ambient = scene.ambient_image;
4.      std::vector<uint16_t> intensity = scene.intensity_image[0];
5.      std::vector<uint32_t> depth = scene.depth_image[0];
6.      std::vector<SOSLAB::point_t> pointcloud = scene.pointcloud[0];
7.      std::size_t height = scene.rows;
8.      std::size_t width = scene.cols;
9.      std::size_t width2 = (scene.cols ==192)?scene.cols*3:scene.cols;
10.  }
11.}
```

Code of Acquiring Raw Data

Here is the code for acquiring raw data from the ML-X Device. The raw data from the ML-X Device is obtained as scene_t, which consists of four 1-dimensional data arrays (ambient, intensity, depth, point cloud). Detailed information about the scene_t structure is provided in the Appendix.

Line	Description
2	Register the callback function. The callback function is called every time scene data is acquired.
3	Declare a SOSLAB::LidarML::scene_t structure to store Raw Data with the variable name "scene" as an argument.
4-7	Obtain the ambient, intensity, depth, and point cloud data from the scene_t structure named 'scene'. The variable names and types for each data are provided in the Appendix.
8-10	Obtain the height and width values for the Depth and Intensity images from the scene_t structure named 'scene'. The width2 variable represents the width value for the Ambient image

3.4. Example Code for Ubuntu/ROS

2) Rviz - Publish Ambient, Depth, Intensity Image

```
1. ros::NodeHandle nh("~");
2. image_transport::ImageTransport it(nh);
3. image_transport::Publisher pub_ambient = it.advertise("ambient_color", 1);
4. sensor_msg::ImagePtr msg_ambient;
5. cv::Mat ambient_image
6. ambient_image(scene.rows, scene.cols, CV_32SC1, ambient.data());
7. ambient_image.convertTo(ambient_image, CV_8UC1, (255.0 / 2000),0);
8. msg_ambient = cv_bridge::CvImage(std_msg::Header(), "rgb8",
   ambient_image).toImageMsg();
9. pub_ambient.publish(msg_ambient);
```

Code for publish Ambient, Depth, and Intensity Images in Rviz

Here is the code for converting and publishing the Ambient, Depth, and Intensity data obtained from the ML-X Device as images for visualization in Rviz. The code provided above is for the conversion and creation of Ambient image publishing.

Line	Description
1-4	To create a Publisher Node for sending messages, I create objects such as ImageTransport and ImagePtr from ROS. The Publisher provides ambient data on the topic 'ambient'
5-6	To visualize the Ambient data from the scene_t structure as an image using OpenCV, here is the process of converting it to the cv::Mat format. Initialize a cv::Mat object with the input values of Height (scene.rows), Width (scene.cols), Ambient data type (CV_32SC1), and the Ambient data values.
7	To visualize the image, the values are converted from the range of the maximum value (2000) and the minimum value (0) to the 8-bit (uchar) format ranging from 0 to 255.
8	To store the OpenCV's cv::Mat object as a message in the Publisher Node, here is the process of converting it to ImagePtr.
9	The publish function of the Publisher object with the topic specified as 'ambient_color' is called, using the ImagePtr variable as the input argument.

The publishing process for each data is the same as the Ambient publishing process, and the image conversion types are as follows:

- Ambient : CV_32SC1
- Depth : CV_32SC1
- Intensity : CV_16UC1

3) Rviz - Publish Point Cloud

```
1. typedef pcl::PointCLoud<pcl::PointXYZRGB> PointCloud_T
2. static const char* DEFAULT_FRAME_ID = "map"

3. ros::NodeHandle nh("~");
4. ros::Publisher pub_lidar = nh.advertise<PointCloud_T>("pointcloud",10);

5. PointCloud_T::Ptr msg_pointcloud(new PointCloud_T);
6. msg_pointcloud->header.frame_id = DEFAULT_FRAME_ID
7. msg_pointcloud->width = width;
8. msg_pointcloud->height = height;
9. msg_pointcloud->points.resize(scene.pointcloud.size())
10.for (int i = 0; i < scene.pointcloud.size(); i++) {
11.    msg_pointcloud->points[i].x = pointcloud[i].x/1000.0;
12.    msg_pointcloud->points[i].y = pointcloud[i].y/1000.0;
13.    msg_pointcloud->points[i].z = pointcloud[i].z/1000.0;
14.}
15.pcl_conversions::toPCL(ros::Time::now(), msg_pointcloud->header.stamp);
16.pub_lidar.publish(msg_pointcloud);
```

Code for publish Point Cloud in Rviz

Here is the code for publishing the point cloud data obtained from the ML-X Device for visualizing the point cloud in Rviz in a ROS environment.

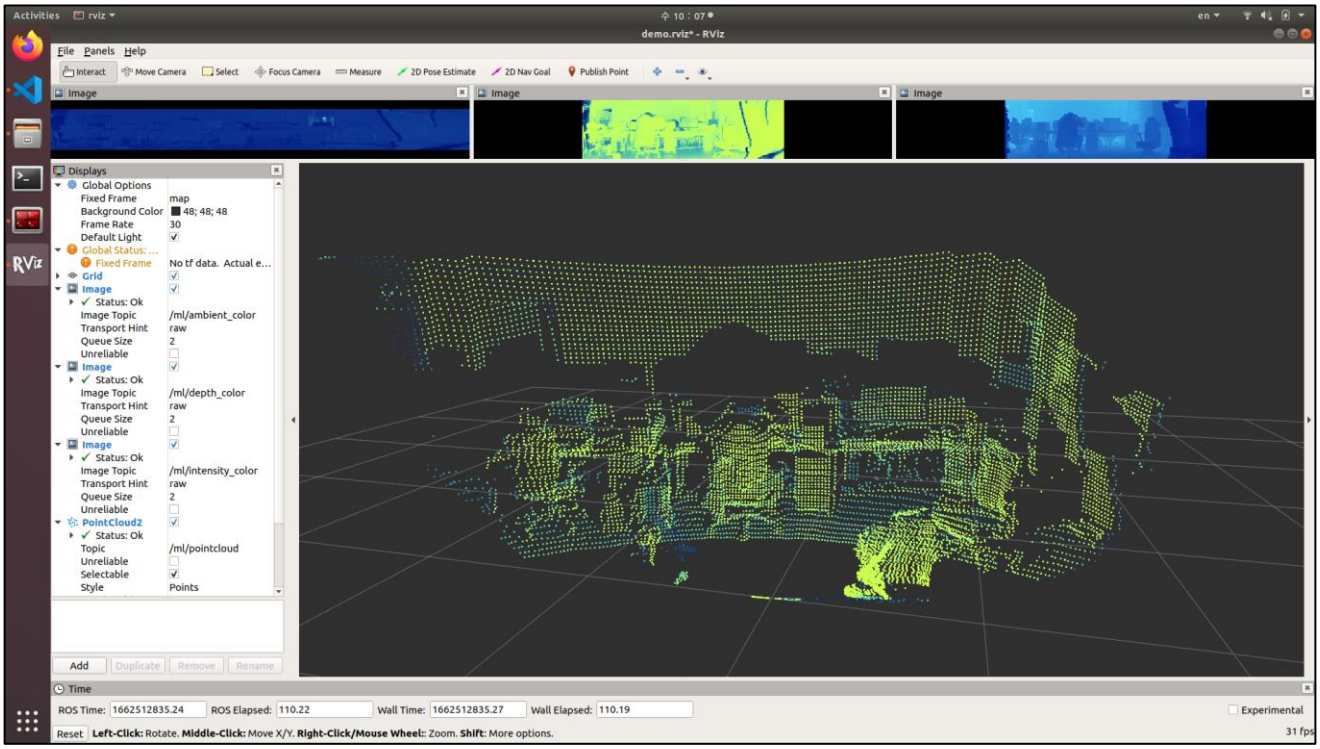
Line	Description
3-4	To create a Publisher Node for sending messages, I create a ros::Publisher object from ROS. The Publisher provides point cloud on the topic 'pointcloud'.
5-9	Define a message variable and initialize its data size and name.
10-14	copy the point cloud data from the scene_t structure to the msg_pointcloud variable and change the distance unit from millimeters (mm) to meters (m).
15-16	After saving the timestamp of the Point Cloud, the publish function of the Publisher object with the topic specified as 'pointcloud' is called, using the PointCloud_T::Ptr variable as the input argument to complete the publish

4) Rviz - Visualization

To visualize it in Rviz, execute the example code and connect to the ML-X Device by entering the following command.

```
$ cd ~/catkin_ws
$ catkin_make
$ source ~/catkin_ws/devel/setup.sh
$ cd ~/catkin_ws/src/ml/launch
$ roslaunch ml ml.launch
```

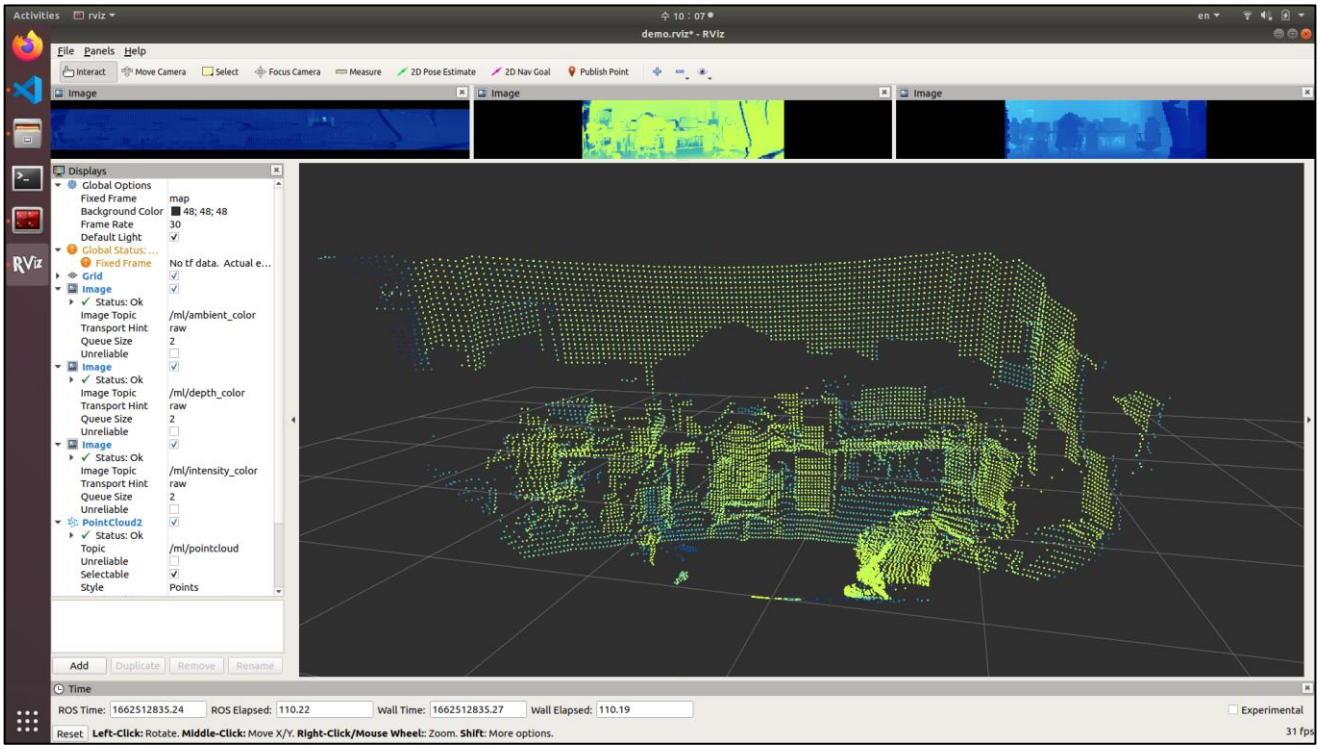
If user click the Add button in the bottom left corner, as shown in the image below, user will see a window similar to the image below. By clicking the 'By topic' menu at the top of that window, user can see all the topics that are being published



Rviz program execution screen

3.4. Example Code for Ubuntu/ROS

To visualize the Ambient, Depth, and Intensity images, select 'Image' in the Topic (/ambient, /depth, /intensity) menu and click the OK button. This will allow user to visualize the images for each topic. For Point Cloud data, select 'PointCloud2' in the Topic (/pointcloud) menu and click the OK button to visualize it.



Rviz base ML-X Data (Ambient/Depth/Intensity Images, Point Cloud) Visualization

5) Rviz – Multi-LiDAR Visualization

To enable multi-LiDAR visualization in Rviz, user need to configure the IP address of each ML-X device differently. In the multi-LiDAR example, the IP settings for ML-X are as follows:

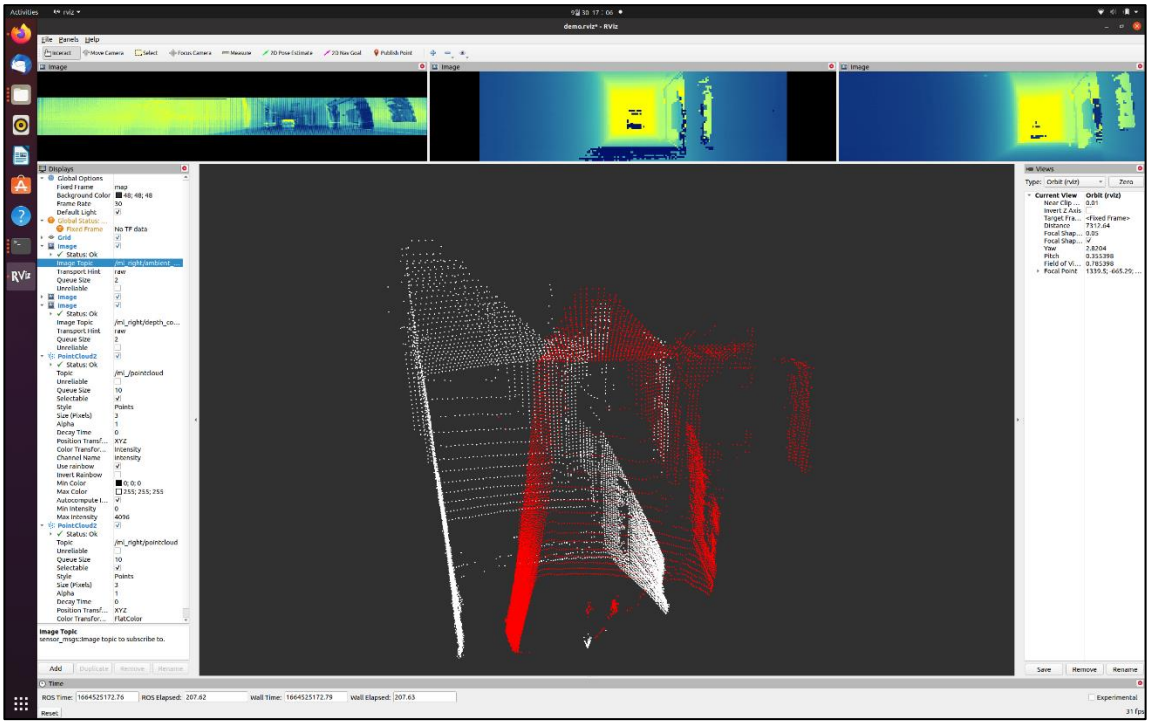
- ML-X IP #1 : 192.168.1.10
- ML-X IP #2 : 192.168.1.11

After changing the IP, execute the example code by entering the following command.

```
[Terminal #1]
$ cd ~/catkin_ws
$ catkin_make
$ source ~/catkin_ws/devel/setup.sh
$ cd ~/catkin_ws/src/ml/launch
$ roslaunch ml ml.launch

[Terminal #2]
$ cd ~/catkin_ws/src/ml/launch
$ roslaunch ml ml_second.launch
```

If user click the Add button in the bottom left corner, as shown in the image below, can see a window like the image below. By clicking the 'By topic' menu at the top of that window, can see the Multi-LiDAR data being transmitted through the 'ml' and 'ml_second' topics.



Multi-LiDAR Visualization

3.5. ML-X ROS2 Example Code Build

The environment required for build the ML-X ROS2 is as follows.

- Ubuntu 18.04
- ROS2 Foxy

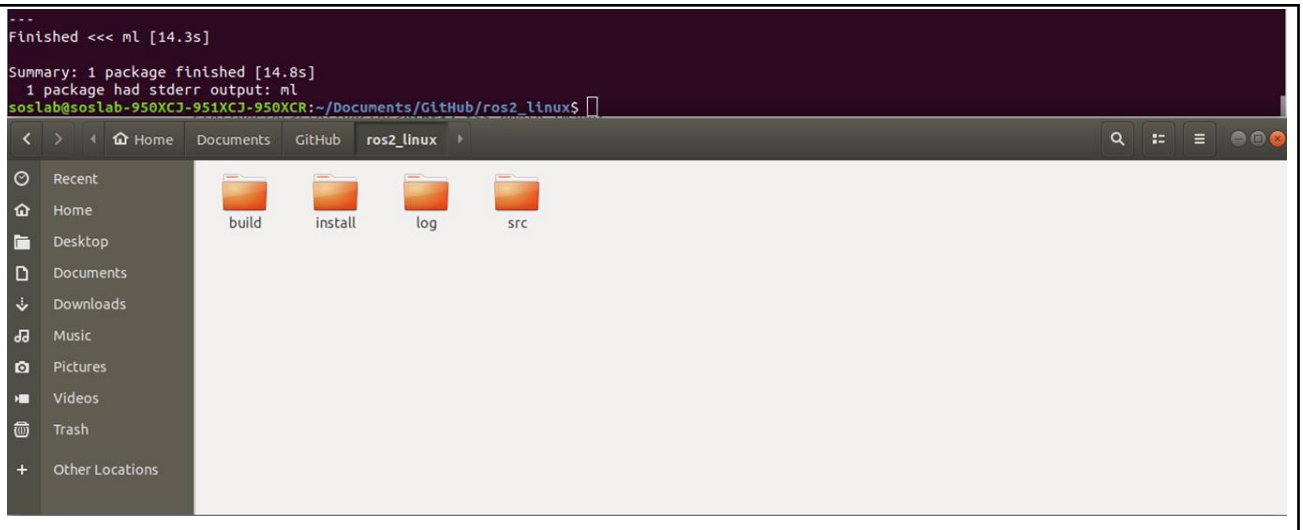
3.5. ML-X ROS2 Example Code Build

1) From the ROS2 installation folder location, enter the command below to set up the ROS2 environment.

```
$ source ${PATH - ROS2}/install/setup.bash
```

2) Navigate to the folder where user downloaded the ML-X ROS2 Example code and build the code.

```
$ cd ${PATH – ros2_ml}
$ colcon build
```



ML ROS2 Build Result

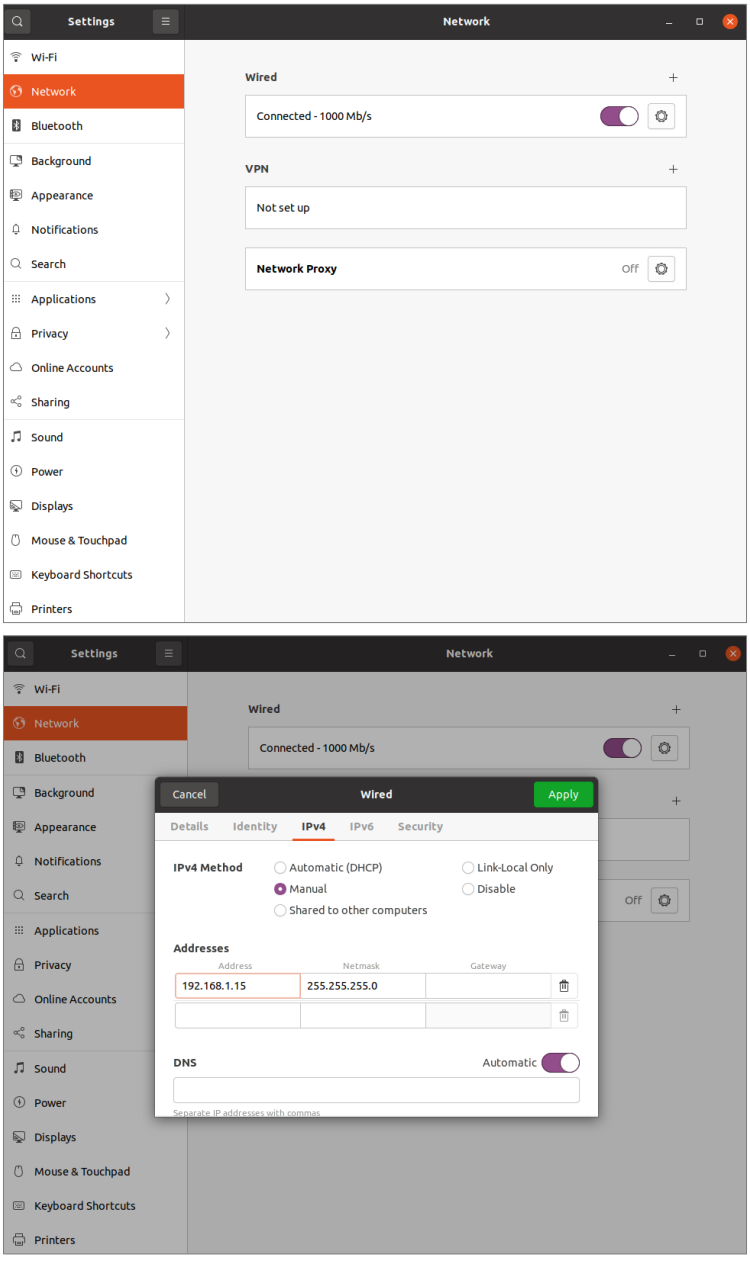
3) Set up the ML-X environment.

```
$ source ./install/setup.bash
```

3.5. ML-X ROS2 Example Code Build

Open the Settings window and navigate to the Network section. Modify the IPv4 settings as follows:

- IPv4 Method – Manual
- Address : 192.168.1.15
- Netmask : 255.255.255.0

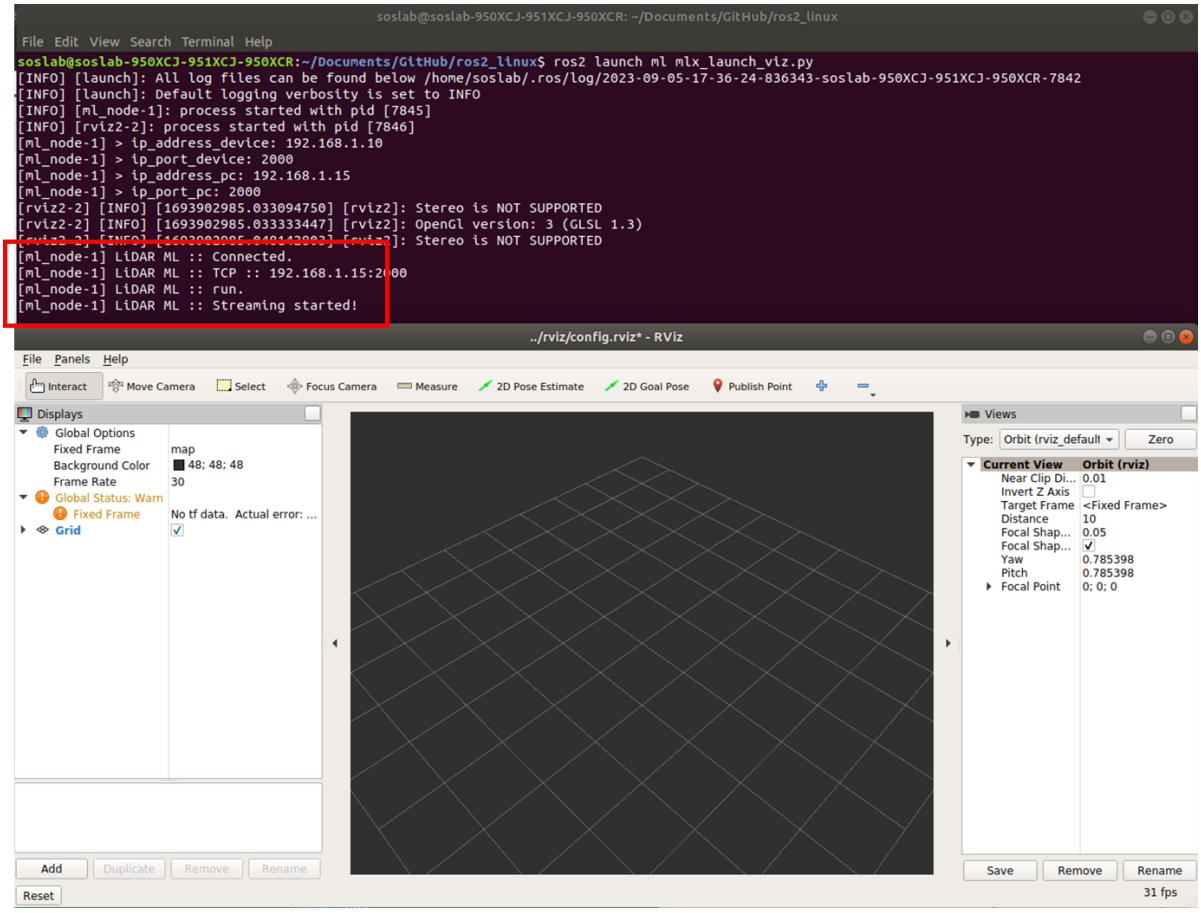


IPv4 Setting @Linux

3.5. ML-X ROS2 Example Code Build

- After setting up the network, enter the command below to run the mlx_launch_viz.py file and connect the PC to the ML-X Device.
- Once the ML-X Device is successfully connected, user can see the output "Lidar ML :: Streaming started!" in the terminal window as shown in the figure below.

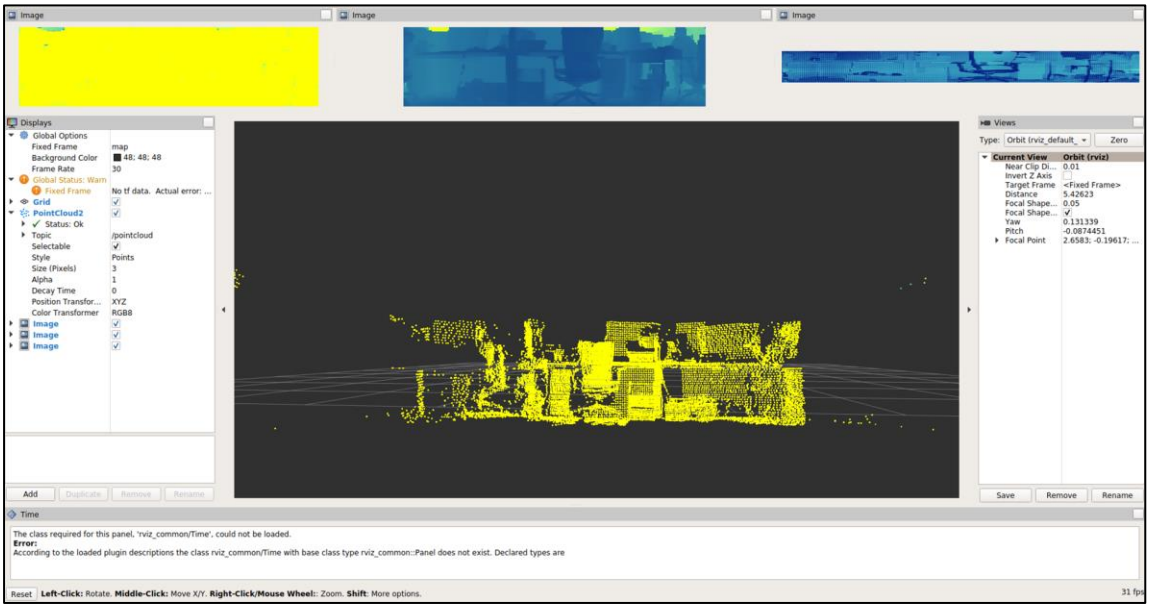
$ ros2 launch ml mlx_launch_viz.py



Connecting and visualizing the ML-X Device through ROS2

3.5. ML-X ROS2 Example Code Build

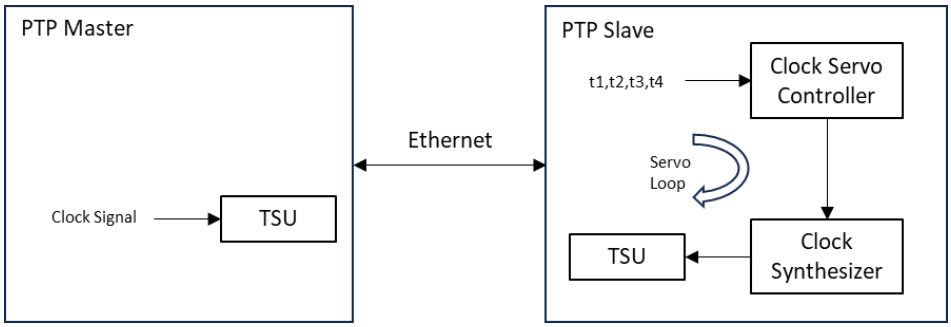
- In Rviz2, click on File – Open Config from the top toolbar.
- By opening the {PATH – ros2_ml}/src/ml/rviz/config.rviz file, user can visualize the ML-X data in Rviz2.



Connecting and visualizing the ML-X Device through ROS2

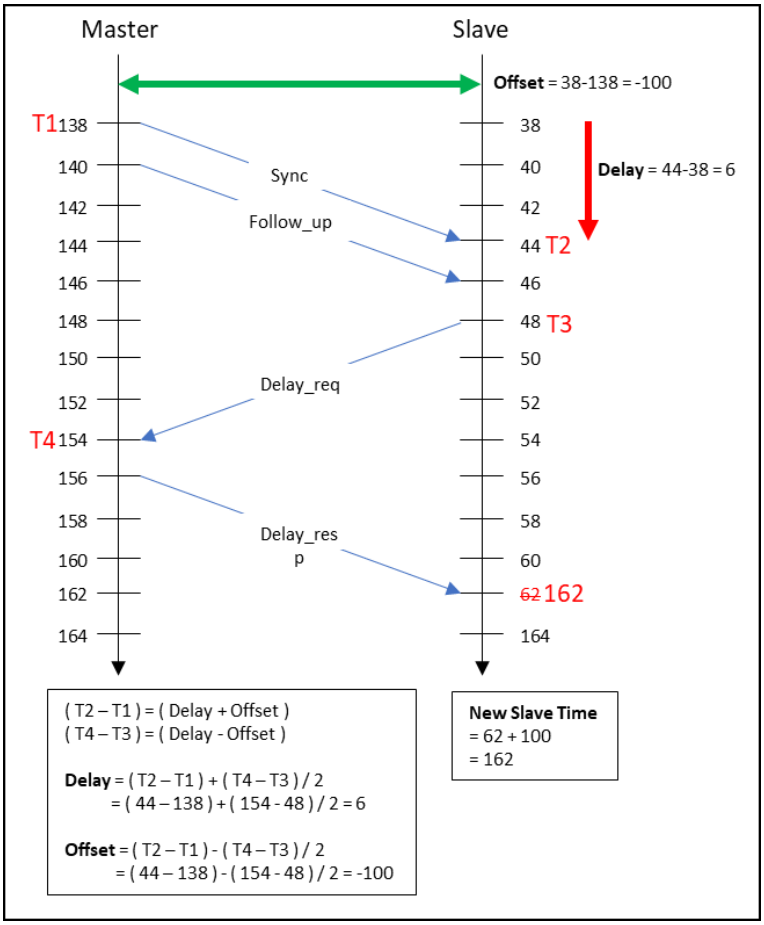
3.6. PTP (Precision Time Protocol)

PTP (Precision Time Protocol) is an IEEE 1588 standard time transfer protocol that enables precise synchronization between devices within a network. In the context of ML-X, it ensures that the timestamp generated by ML-X matches the timestamp of the master device with nanosecond [ns] accuracy.



PTP Block Diagram

Depending on the device operating as the master, PTP may exhibit differences in delay and offset performance due to hardware variations such as ethernet performance and the accuracy of the grand master clock.



PTP: Delay and Offset

3.6. PTP (Precision Time Protocol)

1) PTP Requirement Installation

To use the PTP function of ML-X, the following environment is required, and the packages needed to operate the PTP function can be installed via the command below.

- ML-X Firmware version: v1.5 or above v2.1
- Linux
- linuxptp: Package supporting the PTP function
- ethtool: Package to check Ethernet interface functions

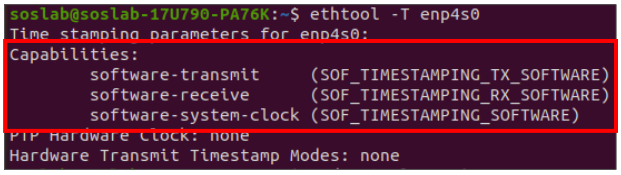
```
$ sudo apt update
$ sudo apt install linuxptp ethtool
```

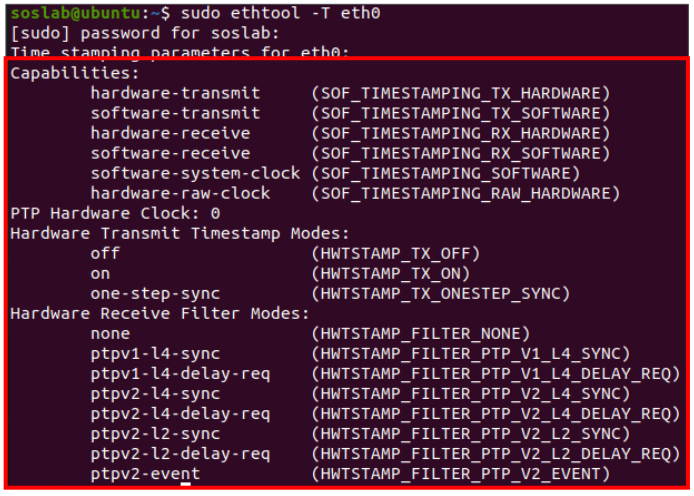
2) Verification of PTP Capability of the Hardware

To verify if the user's hardware supports the PTP function, enter the command below. (User can check the Ethernet name using the ifconfig function.)

```
$ sudo ethtool –T [Ethernet name]
```

If the three items (software-transmit / software-receive / software-system-clock) under Capabilities are displayed as shown in the left image, only the Software based PTP function is possible. If values are displayed under the Hardware Transmit / Receive items as shown in the right image, User can use the Hardware based PTP function.





(L) Software-based PTP operable / (R) Hardware-based PTP operable

3.6. PTP (Precision Time Protocol)

3) Operate PTP

After verifying the operability of both Software-based PTP and Hardware-based PTP, user can activate the PTP function using the command below.

```
$ sudo ptp4l –S –i [Ethernet Name] –m
```

Software based PTP Operating Command

```
$ sudo ptp4l –i [Ethernet Name] –m & sudo phc2sys –w –s [Ethernet Name] –O 0 -m
```

Hardware based PTP Operating Command

4) Example Code for visualize Timestamp

User can visualize the timestamp of ML-X through the code below. If the PTP function is not executed, an unsynchronized timestamp will be displayed. After running the PTP function, user can check the synchronized timestamp of the ML-X.

```
1.  void ml_scene_data_callback(void* arg, SOSLAB::LidarML::scene_t& scene) {
2.      uint64_t timestamp = scene.timestamp[55];
3.      time_t sec = timestamp >> 32;
4.      int nsec = timestamp & 0xFFFFFFFF;

5.      struct tm* ml_tm = localtime(&sec);
6.      std::cout << "time: " << 1900 + ml_tm->tm_year << "."
7.      << ml_tm->tm_mon + 1 << "." << ml_tm->tm_mday << " ";
8.      std::cout << ml_tm->tm_hour << ":" << ml_tm->tm_min << ":"
9.      << ml_tm->tm_sec << ":" << nsec << std::endl;
10.}
```

```
time: 1970.1.1 9:0:17:163885048
time: 1970.1.1 9:0:17:213877640
time: 1970.1.1 9:0:17:263879104
time: 2023.9.6 22:20:38:981331335
time: 2023.9.6 22:20:39:31210543
time: 2023.9.6 22:20:39:81208359
```

: Before applying PTP

: After applying PTP

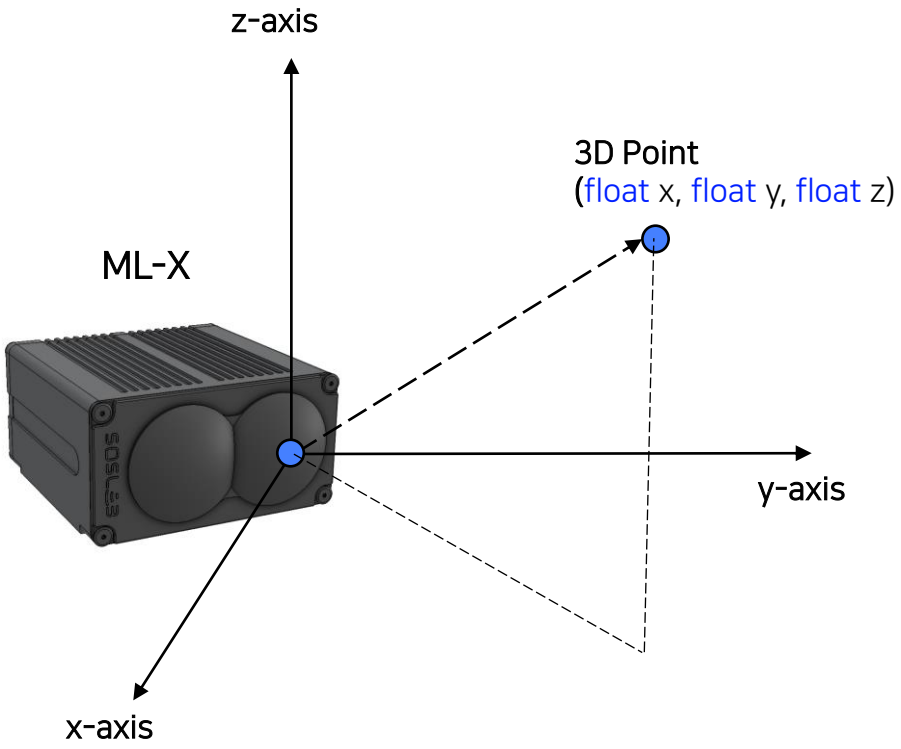
Time synchronization result using PTP

Appendix

A.1. ML-X API - Classes

Classes

Variable	SOSLAB::point_t
Header	#include "soslab_typedef.h"
Description	Structure corresponding to 3D point in coordinate
Member Variables	Description of Member Variables
float x	The x-coordinate of a point (unit: mm)
float y	The y-coordinate of a point (unit: mm)
float z	The z-coordinate of a point (unit: mm)



Cartesian Coordinate System of 3D Point Cloud

Classes

Variable	SOSLAB::ip_setting_t
Header	#include "soslab_typedef.h"
Description	Structure to store IP address and port
Member Variables	Description of Member Variables
std::string ip_address	IP Address
int port_number	Port Number

A.1. ML-X API - Classes

Classes

Variable	SOSLAB::LidarMl::scene_t
Header	#include "ml/libsoslab_ml.h"
Description	Structure of Raw Data
Member Variables	Description of Member Variables
std::vector<uint64_t> timestamp	Time stamp of Raw Data [size : rows] [63:32] Second, [31:0] NanoSecond
uint8_t frame_id	Frame Index [Maximum : 255]
uint16_t rows	Height of Raw Data
uint16_t cols	Width of Raw Data
std::vector<uint32_t> ambient_image	Ambient Data
std::vector<std::vector<uint32_t>> depth_image	Depth Data [1st vector size : # echo, 2nd vector size : rows x cols]
std::vector<std::vector<uint16_t>> intensity_image	Intensity Data [1st vector size : # echo, 2nd vector size : rows x cols]
std::vector<std::vector<point_t>> pointcloud	Point cloud Data [1st vector size : # echo, 2nd vector size : rows x cols]

A.1. ML-X API - Classes

Classes

Variable	SOSLAB::LidarMl
Header	#include "ml/libsoslab_ml.h"
Description	ML-X Device Connection
Functions	
bool connect(const ip_settings_t ml, const ip_settings_t local);	
• Description : Connect between the local PC and the ML-X Device • Input : IP address and port number for the local PC and ML-X Device • Output : Connection status as a boolean variable (returning true if connected)	
bool disconnect();	
• Description : Disconnect between the local PC and the ML-X Device • Output : Disconnection status as a boolean variable (returning true if disconnected)	
bool run();	
• Description : Run the ML-X Device • Output : Run status as a boolean variable (returning true if it is running)	
bool stop();	
• Description : Stop the ML-X Device • Output : Stop status as a boolean variable (returning true if it is stopping)	
bool get_scene(scene_t& scene);	
• Description : Acquire raw data using the input variable 'scene' • Input : scene_t& scene • Output : Acquire the status as a boolean variable (returning true if is acquired)	
bool register_scene_callback(receive_scene_callback_t callback, void* arg);	
• Description : Register a callback function to acquire Scene data • Input (1) : Function: receive_scene_callback that processes the scene_t variable - typedef void(*receive_scene_callback_t)(void* arg, scene_t& scene_) • Input (2) : Class pointer or nullptr • Output : Acquire the register status as a boolean variable (returning true if is registered)	

A.1. ML-X API - Classes

Classes

Variable	SOSLAB::LidarMl
Header	#include "ml/libsoslab_ml.h"
Description	ML-X Play Mode
Functions	
void record_start(std::string filepath)	
• Description : Start the recording ML-X data • Input : Save location of the recording file (.bin)	
void record_stop()	
• Description : Stop the recording ML-X data	
void play_start(const std::string filepath)	
• Description : Play the recording ML-X data • Input : Load location of the recording file (.bin)	
void play_stop()	
• Description : Stop the playing ML-X data	
bool get_scene(scene_t& scene, uint64_t index);	
• Description : Retrieve the data of a specific frame from the recording file using the input variables 'scene' and frame number • Input (1): Variable to store the raw data (scene_t& scene) • Input (2): Frame (uint64_t index) • Output : scene_t variable that stores the raw data from the ML-X Device	
void get_file_size(uint64_t& size)	
• Description : Load the recording file and return the total number of frames • Output : Total Frame	

A.1. ML-X API - Classes

Classes

Variable	SOSLAB::LidarMl
Header	#include “ml/libsoslab_ml.h"
Description	ML-X Functions
Functions	
bool fps10(bool en)	
• Description : Control the On/Off state of the ML-X FPS 10Hz • Input : 10 FPS On (true) / Off (false)	
bool depth_completion(bool en)	
• Description : Control the On/Off state of Super Resolution • Input : Super Resolution On (true) / Off (false)	

A.1. ML-X API - Classes

Classes

Variable	SOSLAB::LidarMl
Header	#include “ml/libsoslab_ml.h"
Description	ML-X Functions
Functions	
bool ambient_enable(bool en)	
• Description : Control the On/Off state of the transmission of Ambient data • Input : Ambient Enable (true) / Disable (false)	
bool depth_enable(bool en)	
• Description : Control the On/Off state of the transmission of Depth data • Input : Depth Enable (true) / Disable (false)	
bool intensity_enable(bool en)	
• Description : Control the On/Off state of the transmission of Intensity data • Input : Intensity Enable (true) / Disable (false)	
bool multi_echo_enable(bool en)	
• 설명 : Control the On/Off state of the transmission of multi-echo data • Input : Multi-echo Enable (true) / Disable (false)	

Classes

Variable	libsoslab_ml.Point()
Module	libsoslab_ml
Description	Class corresponding to 3D point in coordinate
Member Variables	Description of Member Variables
x : float	The x-coordinate of a point (unit: mm)
y : float	The y-coordinate of a point (unit: mm)
z : float	The z-coordinate of a point (unit: mm)

Variable	libsoslab_ml.Scene()
Module	libsoslab_ml
Description	Class of raw data
Member Variables	Description of Member Variables
timestamp : List[float]	Time stamp of Raw Data [size : rows]
rows : int	Height of Raw Data
cols : int	Width of Raw Data
ambient_image : List[float]	Ambient Data
depth_image : List[float]	Depth Data
intensity_image : List[float]	Intensity Data
pointcloud : List[Point]	Point cloud Data

A.2. ML-X API – Python Classes

Classes

Variable	libsoslab_ml.LidarML()
Module	libsoslab_ml
Description	ML-X Device Connection
Functions	
connect(self, ml_ip : str, ml_port : int) → bool	
• Description : Connect between the local PC and the ML-X Device • Input : IP address and port number for the local PC and ML-X Device • Output : Connection status as a bool variable (returning true if connected)	
disconnect(self)	
• Description : Disconnect between the local PC and the ML-X Device	
run(self)	
• Description : Run the ML-X Device	
stop(self)	
• Description : Stop the ML-X Device	
get_scene(self) → Scene	
• Description : Acquire raw data • Output : Scene variable included raw data	

A.2. ML-X API – Python Classes

Classes

Variable	libsoslab_ml.LidarML()
Module	libsoslab_ml
Description	ML-X Functions
Functions	
fps10(self, enable : bool) → bool	
• Description : Control the On/Off state of the ML-X FPS 10Hz • Input : 10 FPS On (true) / Off (false)	
depth_completion(self, enable : bool) → bool	
• Description : Control the On/Off state of Super Resolution • Input : Super Resolution On (true) / Off (false)	
ambient_enable(self, enable : bool) → bool	
• Description : Control the On/Off state of the transmission of Ambient data • Input : Ambient Enable (true) / Disable (false)	
depth_enable(self, enable : bool) → bool	
• Description : Control the On/Off state of the transmission of Depth data • Input : Depth Enable (true) / Disable (false)	
intensity_enable(self, enable : bool) → bool	
• Description : Control the On/Off state of the transmission of Intensity data • Input : Intensity Enable (true) / Disable (false)	

A.3.1 Normal Packet Structure (192 pixels)

The basic packet structure of UDP communication is as follows.

Byte

7	6	5	4	3	2	1	0
header							
timestamp (sec[63:32], nsec[31:0])							
status							
-					row_num	frame_id	type
ambient[1]				ambient[0]			
...				...			
ambient[575]				ambient[574]			
-		intensity[0][echo0]		range[0][echo0]			
point_cloud[0][echo0] (x[20:0], y[41:21], z[62:42])							
...							
-		intensity[191][echo0]		range[191][echo0]			
point_cloud[191][echo0] (x[20:0], y[41:21], z[62:42])							

A.3. UDP Protocol Packet Structure

UDP Protocol Basic Packet Structure (192 pixels)				
Byte	Name	Sub Name	Type	Description
0~7	header	-	char	Header: "LIDARPKT"
8~15	timestamp	-	uint64_t	Timestamp. Unit: 1[s] + 1[ns]
16~23	status	-	uint64_t	Sensing Data
24	type	-	uint8_t	Normal Packet: 0x01
25	frame_id	-	uint8_t	Frame Count Increase in order with frame
26	row_number	-	uint8_t	Row: 0~55
27~31	rsvd	-	uint8_t	Reserved
32~2335	ambient_data	-	uint32_t	576 pixels
2336~2339 2340~2341 2342~2343 2344~2351	lidar_data [0][echo 0]	range intensity rsvd point_cloud(x, y,z)	uint32_t uint16_t uint16_t int64_t	Distance: 0~262143 [mm] Intensity Reserved Point cloud (x, y, z). Unit [mm] [20:0] x, [41:21] y, [62:42] z
...	...	...	...	...
5392~5395 5396~5397 5398~5399 5400~5407	lidar_data [191][echo 0]	range intensity rsvd point_cloud (x,y,z)	uint32_t uint16_t uint16_t int64_t	Distance: 0~262143 [mm] Intensity Reserved Point cloud (x, y, z). Unit [mm] [20:0] x, [41:21] y, [62:42] z

A.3.2 Depth Completion Packet Structure (576 pixels)

The packet structure for Depth completion in UDP communication is as follows.

Byte

7	6	5	4	3	2	1	0
header							
timestamp (sec[63:32], nsec[31:0])							
status							
-					row_num	frame_id	type
ambient[1]				ambient[0]			
...				...			
ambient[575]				ambient[574]			
-		intensity[0][echo0]		range[0][echo0]			
point_cloud[0][echo0] (x[20:0], y[41:21], z[62:42])							
...							
-		intensity[575][echo0]		range[575][echo0]			
point_cloud[575][echo0] (x[20:0], y[41:21], z[62:42])							

A.2. UDP Protocol Packet Structure

UDP Protocol Depth Completion Packet Structure (576 pixels)				
Byte	Name	Sub Name	Type	Description
0~7	header	-	char	Header: "LIDARPKT"
8~15	timestamp	-	uint64_t	Timestamp. Unit: 1[s] + 1[ns]
16~23	status	-	uint64_t	Sensing Data
24	type	-	uint8_t	Depth Completion Packet: 0x11
25	frame_id	-	uint8_t	Frame Count Increase in order with frame
26	row_number	-	uint8_t	Row: 0~55
27~31	rsvd	-	uint8_t	Reserved
32~2335	ambient_data	-	uint32_t	576 pixels
2336~2339 2340~2341 2342~2343 2344~2351	lidar_data [0][echo 0]	range intensity rsvd point_cloud(x, y,z)	uint32_t uint16_t uint16_t int64_t	Distance: 0~262143 [mm] Intensity Reserved Point cloud (x, y, z). Unit [mm] [20:0] x, [41:21] y, [62:42] z
...	...	...	...	...
11536~11539 11540~11541 11542~11543 11544~11551	lidar_data [575][echo 0]	range intensity rsvd point_cloud (x,y,z)	uint32_t uint16_t uint16_t int64_t	Distance: 0~262143 [mm] Intensity Reserved Point cloud (x, y, z). Unit [mm] [20:0] x, [41:21] y, [62:42] z

A.4.1 TCP Protocol Packet Structure

TCP communication is structured in JSON format. It is composed of commands sent from the PC and responses received from the ML-X. The structure in JSON format is as shown in the table below

TCP Protocol Packet Structure	
JSON Format	Description
{ "command": "run" }	Device Run
{ "command": "stop" }	Device Stop

Solid-State LiDAR ML-X

User Guide

Thank you