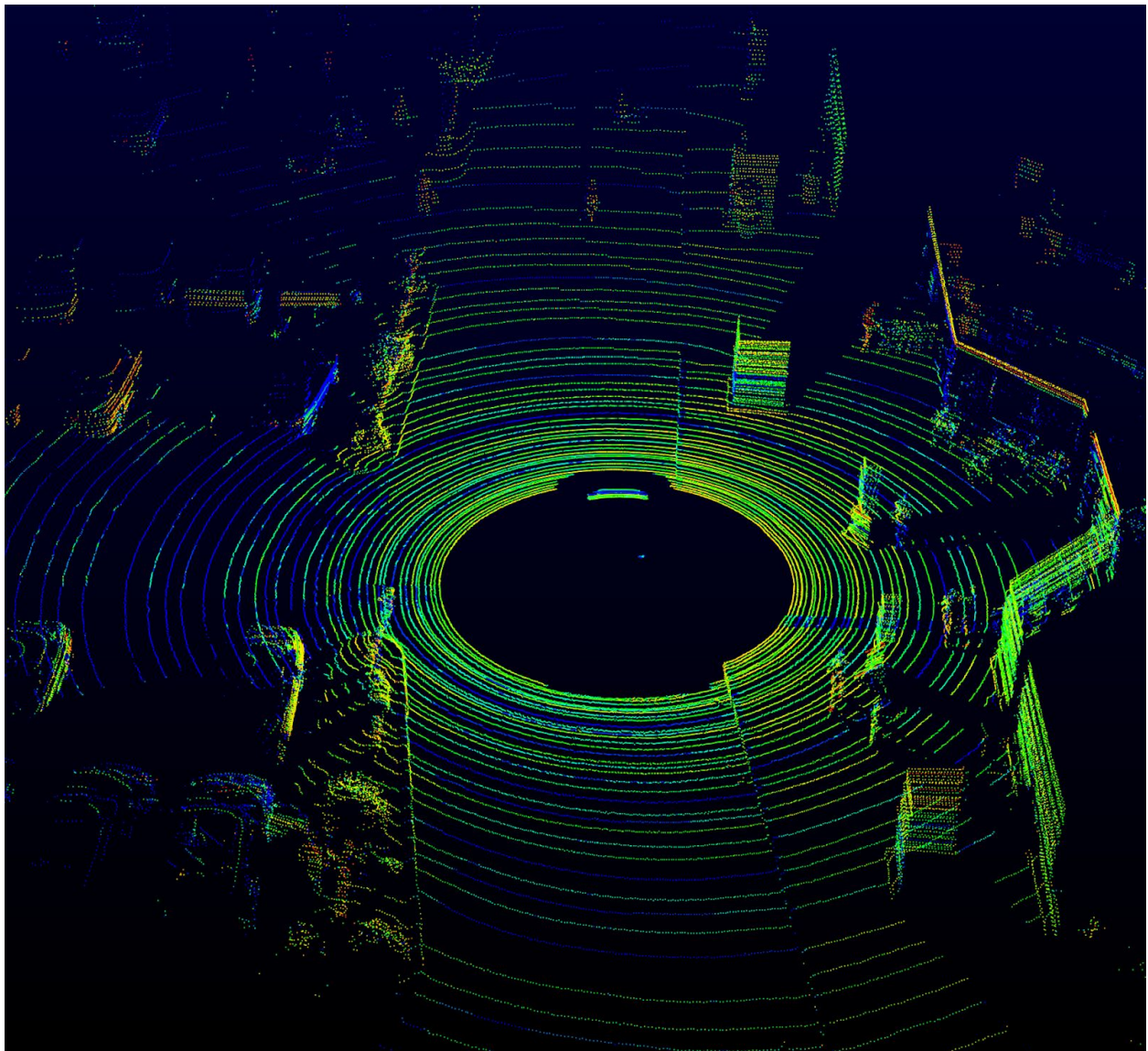


# VeloView User Guide

User documentation for the VeloView application and libraries



Download: <https://www.paraview.org/VeloView>

Copyright: Copyright © 2020, Velodyne Lidar. All rights reserved

Version: 4.1.0

## Contents

<b>I- Introduction</b>	<b>3</b>
Definitions	3
<b>II- Visualize and analyze data</b>	<b>4</b>
A- Global presentation of user interface	4
A-1 Main 3D view	4
A-2 Spreadsheet view	5
B- Live Stream Mode	7
B-1- Open sensor stream and sensor configuration dialog	7
B-2- Visualization & Interaction	10
B-3- Packet loss	10
C- Playback mode (reading .pcap files)	11
D- VeloView toolbars	12
D-1 Basic controls toolbar	12
D-2 Playback controls toolbar	13
D-3 Color controls toolbar	14
D-4 View and camera controls toolbar	15
D-5 Geolocation controls toolbar	16
D-6 Tools Menu	17
<b>III- Python Console</b>	<b>19</b>
A- Introduction	19
B- Presentation of the Python Shell	20
B-1 Introduction to python-VeloView functionalities	20
B-2 Opening the python console	20
B-3 Presentation of the python shell	20
D- VeloView and VTK glossary and data model	21
E- Examples	22
E-1- Get data information	22
E-2- Select points using data	24

E-3- Display a 3D model in the mainview	25
E-4- Start VeloView with a 3D model loaded in the mainview with “- -script”	26
E-5- Handling multiple lidar sensor using python shell	26
F- List of functionalities	28
F-1- Simple Paraview : “lv.smp.”	28
F-2- Qt “lv.QtGui”	28
F-3- Vtk Python	28
F-4- VeloView functions “lv.”	28

## I- Introduction

VeloView is the software provided by Velodyne to visualize, analyze and record data from Velodyne-HDL sensors (and newer devices). The software performs real-time visualization and processing of live captured 3D LiDAR data from Velodyne's sensors (HDL-64E, HDL-32E, VLP-32, VLP-16 (Puck), Puck Lite, Puck HiRes, Alpha Prime). VeloView is able to playback pre-recorded data stored in *.pcap* files, and can record live streams as *.pcap* files. In the next sections of the manual it will be assumed that the reader is familiar with the Velodyne's sensors.

In the first part of the manual, we will focus on how to get, visualize and analyze Velodyne sensor data using the graphical interface of VeloView. In the second part, we will show how it is possible to process the data using python scripting with the Python shell/console.

If you don't have VeloView installed on your computer yet, you can get the installer corresponding with your operative system or the source code here : <https://www.paraview.org/Wiki/VeloView>.

## Definitions

**Frame** : For rotational sensors, the aggregate of all data points acquired during an entire scan (e.g. full 360° sweep). For directional sensors, the aggregate of all data points acquired during a full scan across the field of view, and back.

**GUI** : Graphical User Interface, the software windows and buttons with which the user can interact with mouse and keyboard.

**Pcap** : *.pcap* files are data files containing raw network packets received on some port.

**Python shell** : Python is an interpreter programming language, meaning that it executes the code line by line. The Python shell is the interactive console which is used to execute Python commands provided by the user.

**RPM** : Rotations Per Minute, the current spinning speed of a rotational sensor.

## II- Visualize and analyze data

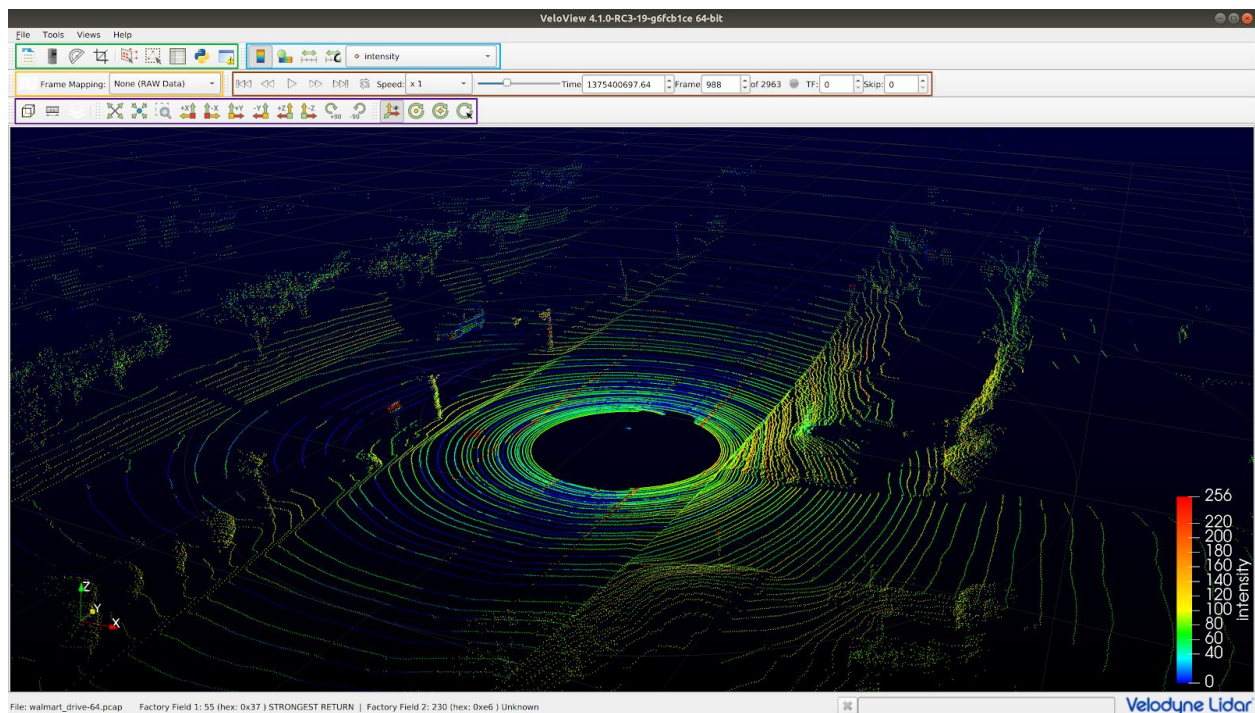
There are two ways to visualize data in VeloView: live stream mode and playback mode. The live stream mode corresponds to the situation where a Velodyne sensor is directly connected to the computer running VeloView. In this case, the live stream mode permits to visualize the current output of the sensor (just like a webcam). The playback mode corresponds to the situation where you had recorded a live stream in .pcap format and you want to replay it.

### A- Global presentation of user interface

First, we will introduce the VeloView graphical user interface (GUI). The VeloView GUI is composed of four toolbars (introduced in section **D- VeloView toolbars**) and two different views : the main 3D view, and the spreadsheet view.

#### A-1 Main 3D view

The main view displays the 3D point-cloud in an interactive viewer (see **figure 1**).



**Figure 1** : Main view of VeloView and its toolbars. *green*: Basic controls; *blue*: Color controls; *orange*: Geolocation controls; *brown*: Playback controls; *purple*: View and camera controls.



You can use the mouse/trackpad to move, zoom or rotate the 3D view. Additional keys can be combined with clicks to provide more options, or to replace some if controls are unavailable with your hardware. The following tab sums up the different controls :

	Left click	Middle click	Right click	Wheel (mouse) or two fingers (trackpad)
<b>(no additional key)</b>	Rotate around center	Move the focus point on orthogonal plane	Zoom in/out	Zoom in/out
<b>Ctrl +</b>	Zoom in/out	Rotate around center	Great zoom in/out	Zoom in/out, focusing on cursor's position
<b>Shift +</b>	Rotate around orthogonal plane's normal	Rotate around center	Move the focus point on orthogonal plane	Zoom in/out

## A-2 Spreadsheet view

The spreadsheet view displays the point-cloud sensor data, but as a spreadsheet (see **figure 2**). It is hidden by default, but can be activated by clicking the *Spreadsheet* button in the Basic controls toolbar.

Showing <span>Data (Frame)</span> <span>Attribute: Point Data</span> <span>Precision: 3</span> <span>10</span> <span>10</span> <span>10</span> <span>10</span> <span>10</span>											
	Point ID		Points_m_XYZ		adjustedtime	azimuth	distance_m	intensity	laser_id	timestamp	vertical_angle
0	0	1.526	15.446	-1.746	1123955004.000	18	15.618	47	0	1123955004	-7.150
1	1	1.127	17.122	-1.848	1123955005.000	19	17.257	91	1	1123955005	-6.810
2	2	0.438	17.845	-1.836	1123955010.000	21	17.944	52	4	1123955010	-6.510
3	3	-0.228	18.923	-1.834	1123955011.000	22	19.012	46	5	1123955011	-6.140
4	4	0.524	13.876	-1.872	1123955012.000	23	14.010	63	6	1123955012	-8.490
5	5	0.029	14.540	-1.882	1123955014.000	23	14.660	58	7	1123955014	-8.150
6	6	-1.002	19.963	-1.833	1123955016.000	25	20.071	63	8	1123955016	-5.810
7	7	-1.838	21.183	-1.838	1123955017.000	25	21.341	1	9	1123955017	-5.480
8	8	-0.561	15.085	-1.881	1123955018.000	26	15.211	23	10	1123955018	-7.850
9	9	-1.155	15.815	-1.882	1123955020.000	27	15.967	22	11	1123955020	-7.480
10	10	3.235	31.771	-1.492	1123955022.000	28	31.969	21	12	1123955022	-3.040
11	11	2.251	35.467	-1.479	1123955023.000	29	35.569	79	13	1123955023	-2.710

**Figure 2** : Spreadsheet view. *Options from left to right* : Data to display, Attribute to display, Precision, Toggle scientific representation, Show only selected points, Toggle column visibility, Toggle cell connectivity visibility, Export spreadsheet.

This is useful to check the raw numeric data, to select points on some numeric fields conditions, export this data to CSV file etc.

Notice that the rows are sortable if you click on the column header.

### Here is a brief description of the spreadsheet toolbar controls :

#### **Data to display :**

This dropdown list lets you choose the data you want to display in the spreadsheet. It can be the LiDAR point cloud, but also the GPS track, the LiDAR calibration file, etc.

#### **Attribute to display :**

Data can be associated to points (e.g. 3D coordinates, intensity or timestamp), or to the entire frame (RPM). This list lets you choose which attribute you want to display.

#### **Precision :**

Set the number of decimals to use when displaying non-integer values.

#### **Toggle scientific representation :**

Switch between scientific or fixed-point representation for non integer values.

#### **Show only selected points :**

If enabled, the spreadsheet displays only the points that are selected in the main 3D view.

#### **Toggle column visibility :**

Choose which columns you want to display in the spreadsheet. For example, you may want to hide the *adjustedtime* field because you don't need it.

#### **Toggle cell connectivity visibility :**

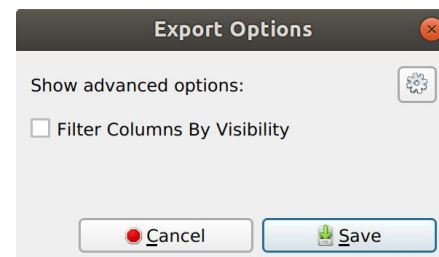
Choose if the *Cell data* attribute should display the point ID. You can probably ignore this setting.

#### **Export spreadsheet :**

Export spreadsheet to a CSV file. This can be useful to save selected data points to disk for analysis, post-processing, plotting graphs, running statistical checks, etc.

When you hit this button, it opens a dialog to let you enter the export file name and choose the save location.

After that, you enter in *Export options* (see **figure 3**). You can select whether to also export the hidden columns that you may have disabled using the *Toggle column visibility* option.



**Figure 3 :** Export spreadsheet option

### About LiDAR data columns :

If you are displaying the LiDAR data in the spreadsheet, here is a brief description about what each columns represents for each point :

- *Point ID* : the ID number within the frame.
- *Points\_m\_XYZ* : the 3D coordinates.

- *adjustedtime* : same as *timestamp*, except that it will not roll over each hour. Because it's a 64 bits integer, It will roll over to 0 after reaching  $(2^{64}) - 1$  (which represents about 500 000 years).
- *azimuth* : the clockwise azimuth angle, in degrees.
- *distance\_m* : range measurement (distance of point to sensor), in meters.
- *intensity* : the reflectivity of the target object, in range [0; 255]. This measurement is calibrated to be independent of laser power and distance. Some LiDAR devices use different mapping for sub-ranges depending on the object reflectivity (e.g. sub-range [0; 100] encodes reflectivity from 0% to 100% for diffuse reflectors, and sub-range [101; 255] reports reflection of retro-reflectors). See the sensor's user manual (section 6.1) for details.
- *laser\_id* : the ID of the laser that acquired this point.
- *timestamp* : the *Top of Hour* timestamp in microseconds at which the laser pulse was fired. This timestamp will roll over each full hour (it will return back to 0 each hour). If GPS time sync with *pps* is set up, the *top hour* will be the last full UTC hour. Otherwise, it begins at LiDAR device's startup.
- *vertical\_angle* : the elevation angle of the laser acquired this point, in degrees.

## B- Live Stream Mode

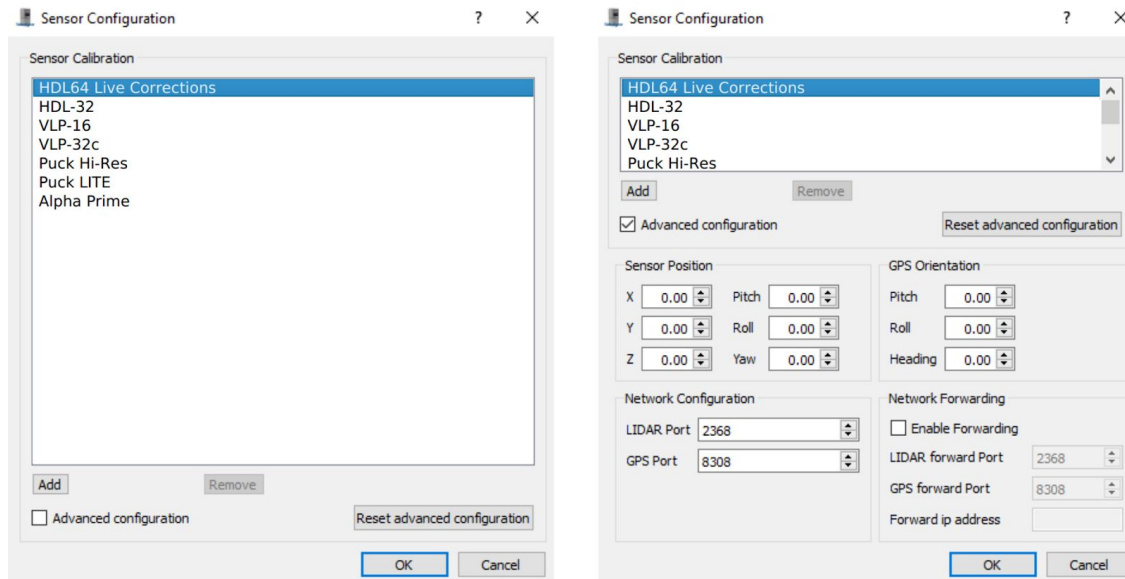
### B-1- Open sensor stream and sensor configuration dialog

Once a compatible (HDL-64E, HDL-32E, VLP-32, VLP-16, Puck, Puck Lite, Puck HiRes, Alpha Prime) Velodyne sensor is connected to your computer and VeloView is running, you can display the output of the sensor using the sensor stream action (see **figure 4**).



**Figure 4** : Sensor stream action

Once the *Sensor Stream* button has been clicked, a configuration dialog named *Sensor Configuration* will pop (see **Figure 5**). You will be able to choose the sensor calibration file corresponding to your sensor. If you click on the advanced configuration checkbox, you will be able to configure the GPS parameters and the network configuration.



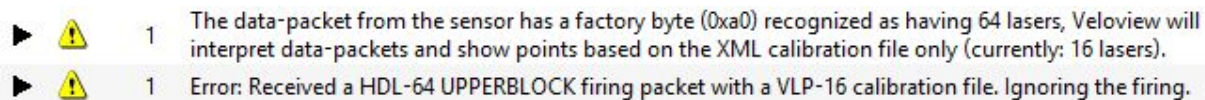
**Figure 5 :** Sensor configuration. *Left* : default option; *Right* : with advanced options.

#### **Sensor Calibration file:**

The sensor configuration file depends on the type of sensor you have connected to the computer. The configuration file contains information about the number of lasers, the vertical angle corrections, the intensity correction, distance calibration, etc. If you want to add a specific sensor calibration file you can click on the *Add* button and select your calibration file.

**Remark:** In case you chose a wrong calibration file, here are some hints that indicate that you may have chosen the wrong one:

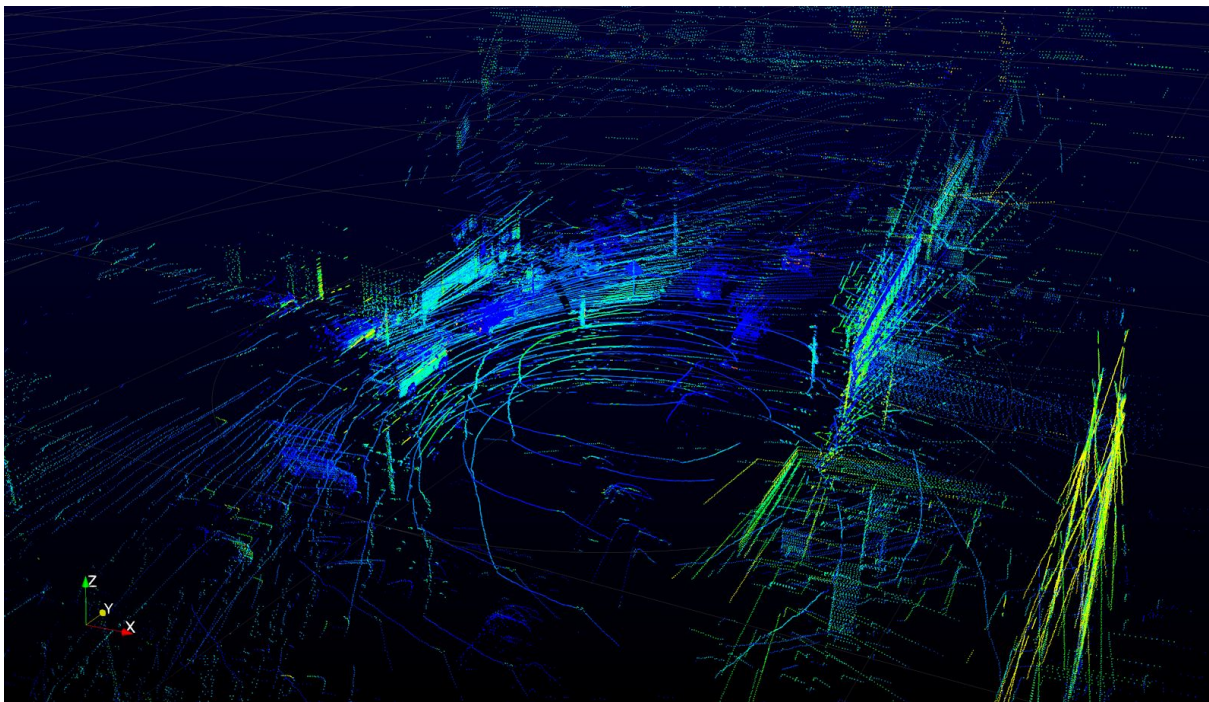
- If the number of lasers does not match between your sensor and the calibration file you may obtain an error like in **figure 6**.



**Figure 6 :** Error message when number of lasers does not match (VLP-16 instead of HDL-64)

- The output point cloud is distorted because of the wrong angle corrections, like in **figure 7**.





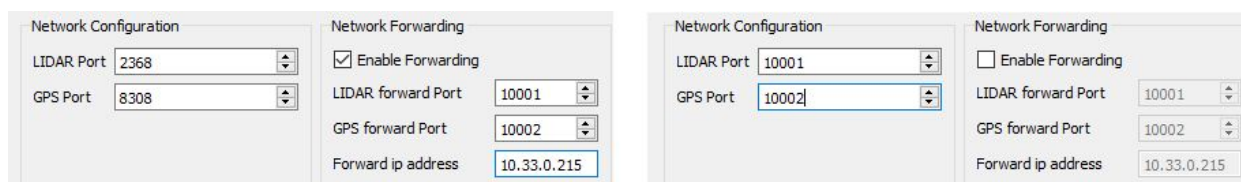
**Figure 7 :** Distorted point cloud due to wrong configuration file (HDL-32 calibration chosen instead of Alpha Prime)

#### **GPS parameters (position and orientation):**

In case you are using a GPS associated with the Velodyne sensor you should be careful about the sensor position and the GPS orientation. Since the GPS and the Velodyne sensors may not be located at the same physical point and exactly aligned you must indicate to VeloView the position and orientation of the Velodyne sensor according to the GPS referential. If the orientation is not indicated, the point cloud will not be georeferenced due to a rotation shift. If the position is not indicated, the result may have some lever arm issues and result in a blurred stabilization of the points clouds. The position is in meters and the orientation in degrees.

#### **Network configuration:**

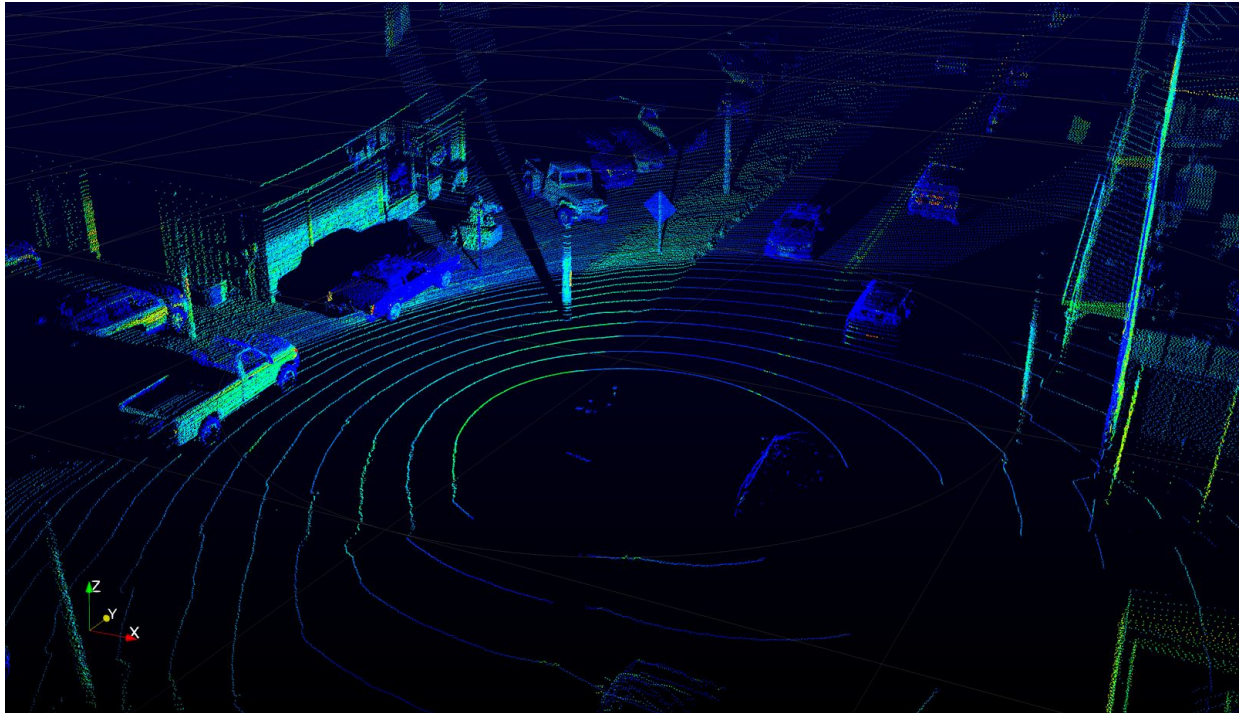
You have the possibility to forward the lidar data to another computer station. For example you can have a first instance of VeloView which receives the sensor's data and forwards it to another instance of VeloView on another computer at the address 10.33.0.215. To do so, the first instance of VeloView will receive the data on the usual ports [2368; 8308] and forward it to the ip address 10.33.0.215 and ports [10001; 10002] (you can choose other ports here). The second instance of VeloView will receive the data on the ports [10001; 10002] (see **figure 8**). Of course, you can forward the data received by the second instance of VeloView if you want.



**Figure 8 :** *Left* : first instance of VeloView which forwards the received data on the address 10.33.0.215

on the ports [10001; 10002]. *Right* : Second instance of VeloView which receives the forwarded data on the ports [10001; 10002].

Once the configuration is correctly done, you should be able to see the point cloud output by your sensor (see **figure 9**).



**Figure 9** : Output of a correctly configured Alpha Prime sensor

## B-2- Visualization & Interaction

While the live stream is running you can pause the stream to a specific frame or record the stream into a *.pcap* file (see **figure 10**).



**Figure 10** : *Green* : button to pause the stream, *Red* : button to record the stream

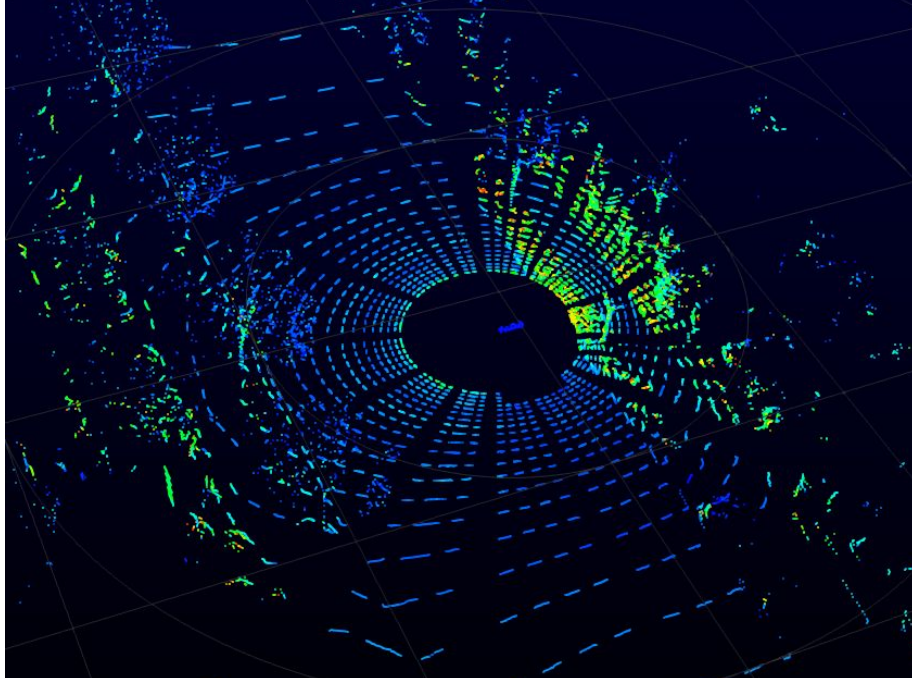
## B-3- Packet loss

Sometimes, some data packets may be lost during a stream. This may happen for various reasons due to hardware, software or network issues :

- underpowered computer
- faulty wired connection to LiDAR device
- too slow network bandwidth
- overloaded network leading to collisions and slower data transmission

- overloaded computer, which is unable to process all received data quickly enough

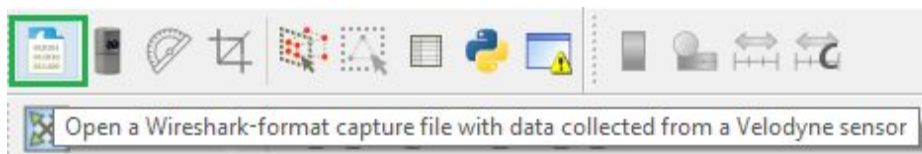
This loss may be impossible to detect by VeloView, but can lead to display or processing issues. **Figure 11** shows an example of partial packet loss during a live stream : notice that some azimuth ranges are missing.



**Figure 11** : Partial packet loss during an HDL-32 live stream

### C- Playback mode (reading *.pcap* files)

If you have a *.pcap* file containing Velodyne sensor data (recorded from VeloView, Wireshark, ...), it is possible to display the data in VeloView with the playback mode using the playback mode button (see **figure 12**).



**Figure 12** : Playback mode action

Once the *Playback Mode* button has been clicked, a configuration dialog named *Sensor Configuration* will pop. For more information about the *Sensor Configuration* dialog and how to set it, please refer to section **B-1- Open sensor stream and sensor configuration dialog**.

In this mode, you can easily interact with frames playback, going back to previous frames, change playing speed, looping over data, etc. All these controls are described in section **D-2 Playback controls toolbar**.



## D- VeloView toolbars

Once you are visualizing data in VeloView, many tools are available to analyze and improve comprehension of the data.



**Figure 13** : VeloView's toolbars. *green*: Basic controls; *blue*: Color controls; *orange*: Geolocation controls; *brown*: Playback controls; *purple*: View and camera controls.

### D-1 Basic controls toolbar

The basic controls toolbar is located at the top left-hand corner of the VeloView GUI.



**Figure 14** : Basic controls. From left to right : Open .*pcap*, Sensor stream, Choose calibration file, Crop returns, Select all points, Select dual returns, Spreadsheet view, Python console, Error console.

#### Open .*pcap*:



Open a .*pcap* file containing recorded Velodyne sensor data.

#### Sensor stream:



Open an UDP socket to receive the data of a connected sensor in real-time.

#### Change the calibration:



Open the calibration dialog to change the calibration parameters (calibration file, GPS parameters and network parameters). Note that Open .*pcap* and Sensor stream buttons will automatically open the calibration dialog before displaying the point cloud. But if you did something wrong in the calibration you can adjust it using this button.

#### Crop returns:



Filter points to only show those that are inside or outside a defined region. The region can be defined using cartesian (x, y and z condition) or spherical (radius, azimuth and vertical angle condition) coordinates (see **figure 15**).

#### Select points:




Select points in a defined region. The region is defined with the mouse. Once drawn, the defined zone and viewpoint are saved, and all the 3D points falling in this zone from this viewpoint will be selected. This works even if you move the camera around or change the time-step.


#### Select dual returns:




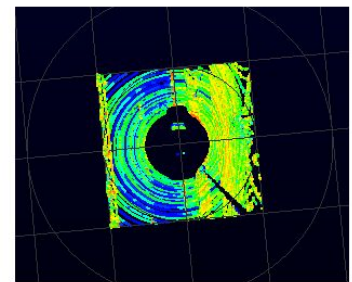
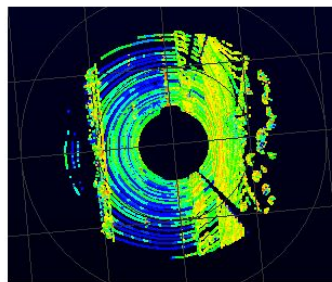
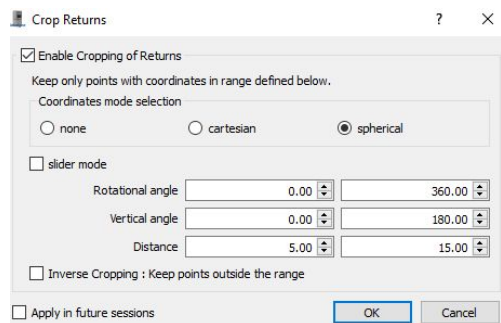
Select the dual returns of all selected points. If

no selection exists, this operation will return the dual returns of all points.

**Error console:**   
Open the error console.

**Spreadsheet view:**   
Open the spreadsheet view.

**Python console:**   
Open the python console (python console is explained in section III- Python Console).



**Figure 15:** *Left* : Crop returns dialog to define the crop region. *Middle* : a region defined in spherical mode (condition over the radius, azimuth and vertical angle). *Right* : a region defined in Cartesian mode (condition over x, y and z).


## D-2 Playback controls toolbar

You can affect the way the data are played in VeloView using the playback control toolbar (**figure 16**).




**Figure 16** : Playback control toolbar. *From left to right* : Go to start, Go previous frame, Start / Stop, Go next frame, Go to end, Loop, Record, Playback speed, Trailing frames, Skip and Select frame slider.


**Go to start:**   
Go to the first frame of the pcap.

**Go previous frame:**   
Go to the previous frame.

**Start / Stop:**   
Start or stop the playback.

**Go to next frame:**   
Go to the next frame.

**Go to end:**   
Go to the last frame of the pcap.

**Loop:**   
Enable or disable the loop mode. Loop mode replays the playback once the last frame is reached.



**Speed:** Speed: x default ▾

Control the speed of the playback. The numbers indicate the multiplicative coefficient according to the real time.

**Trailing frame:** TF: 0 ▴ ▾

Display the last X frames at the same time. It is useful when you have stabilized data (with GPS/IMU or SLAM data for example) (see **figure 17**).

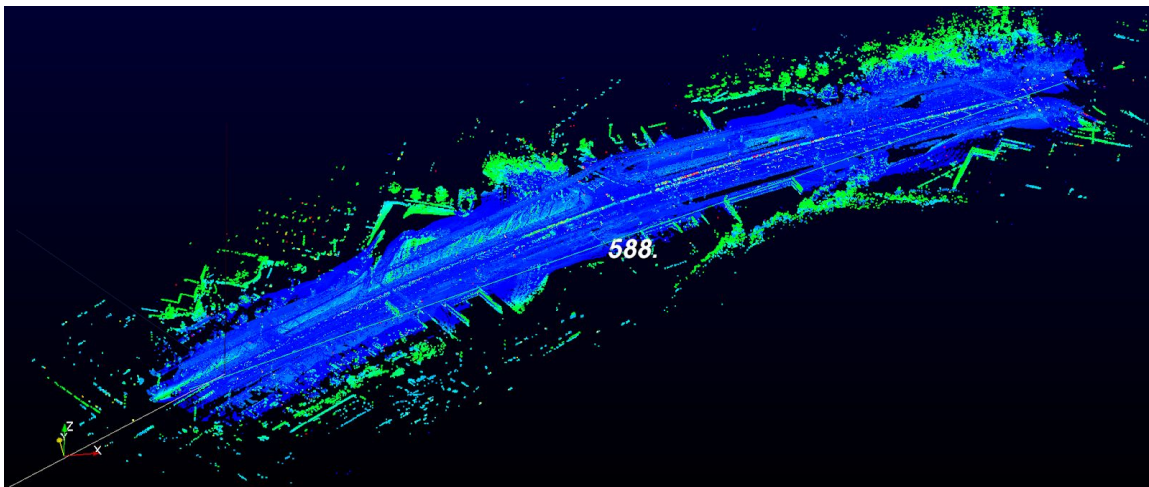
**Skip:** Skip: 0 ▴ ▾

Subsample the shown points by keeping only one

point every X points of the point cloud. It is useful if you are in trailing frame mode with a lot of accumulated point clouds.

**Select frame slider:** 

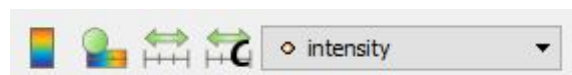
Indicate the current frame. You can move to the frame you want using the slider or the spinbox.



**Figure 17** : Stabilized road section using GPS over 588 meters, visualized using aggregated trailing frames.

### D-3 Color controls toolbar

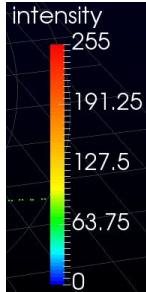
The color control tool (see **figure 18**) is useful to display the information attached to each point. You can choose the information you want to use for color-coding and choose the exact range or color set of the color map.



**Figure 18** : From left to right : Color legend, Edit color map, Fit color range to data, Custom color range, Color-coding data selection

**Color legend:** 

Display/hide the color legend within the main 3D View (see **figure 19**).



**Figure 19** : Color legend

#### **Edit color map:**

Edit and change the color mapping of the data.

#### **Rescale data:**

Rescale the color map so that the color range corresponds to the data value bounds

#### **Rescale custom:**

Rescale the color map as you want. It is useful if

you have data with a big range value but the major part of the information is located on subrange values.

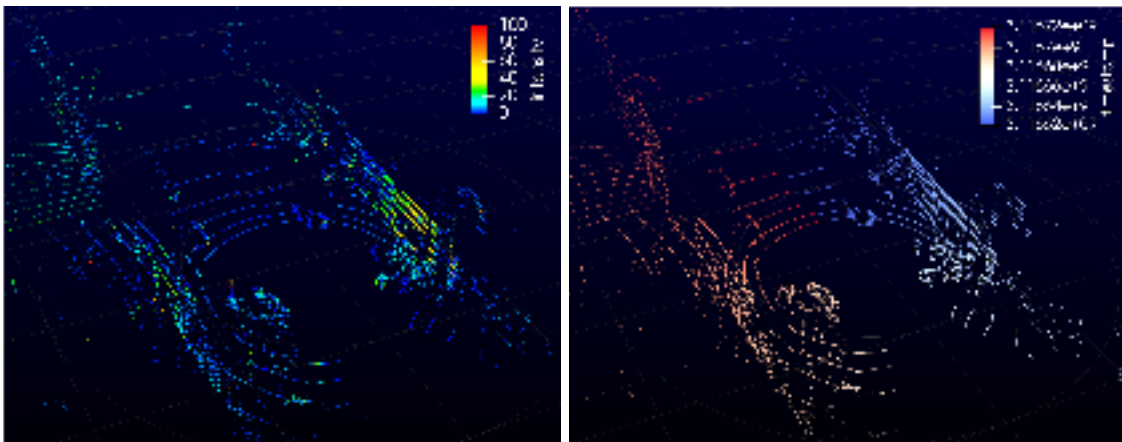
#### **Data selection:**

intensity

Changes the data information used for color coding. By default the intensity of each Lidar Return is displayed. However, you can select other information such as X, Y or Z coordinates, timestamp, laser id... (see **Figure 20**).

- Solid Color
- ◇ X
- ◇ Y
- ◇ Z
- ◇ adjustedtime
- ◇ azimuth
- ◇ distance\_m
- ◇ intensity
- ◇ laser\_id
- ◇ timestamp
- ◇ vertical\_angle

**Figure 20** : Available data information for coloring



**Figure 21** : Same frame displayed using different data information. *Left* : intensity, *Right* : timestamp.

### D-4 View and camera controls toolbar

The view control (see **figure 22**) contains tools to control the camera.



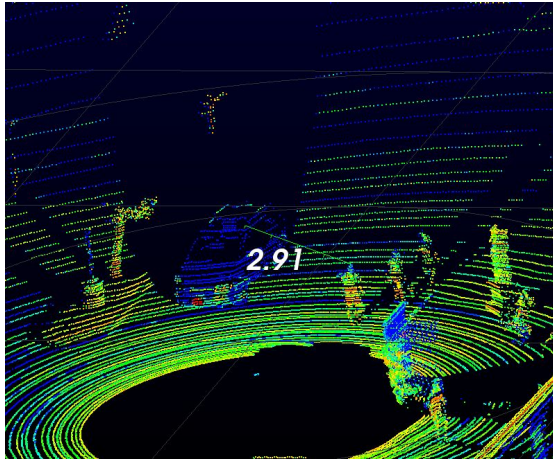
**Figure 22:** View and camera controls toolbar. *From left to right* : Projective/orthogonal view, Measure distance, Plane fit selection; Reset camera, Zoom to data, Zoom to box, Set view direction to +X/-X/+Y/-Y/+Z/-Z, Rotate view by  $\pm 90^\circ$ ; Show orientation axes, Show center, Reset center, Pick center.

### **Orthogonal view:**

Switch from a perspective (smaller far objects) to an orthogonal view (all distances kept).

### **Measure distances:**

*Ctrl + left mouse button* to draw a line and measure its distance. This functionality only works when the orthogonal view is activated (see **figure 23**).



**Figure 23:** Distance between car and pedestrian

### **Plane fit tool:**

Compute the plane that fits the currently selected points.

### **Reset Camera:**

Reset the camera at its initial position and orientation.

### **Zoom to data :**

Zoom to fit the bounding box of the data.

### **Zoom to box :**

Zoom to the user-drawn box.


### **Set view direction to $\pm X$ , $\pm Y$ , $\pm Z$ :**

Set the camera viewing direction to the X, Y or Z axis.

### **Rotate view by $\pm 90^\circ$ :**

Rotate the view by  $90^\circ$  clockwise or  $-90^\circ$  counterclockwise.

### **Show orientation axes :**

Show/hide the orientation axes  frame.

### **Show center :**

Show rotation center. This is the point used as rotation center when rotating the 3D view with the mouse/trackpad.

### **Reset center :**

Reset the rotation center to origin.

### **Pick center :**

Set a new rotation center.

## D-5 Geolocation controls toolbar

The geolocation toolbar (see **figure 24**) can be used to display the GPS / IMU trajectory associated with the LiDAR data (if GPS is connected to the LiDAR device). If this toolbar is disabled, you can activate it in *Views > Toolbars*.



**Figure 24:** Geolocation controls toolbar. *From left to right* : Show the GPS track, Frame mapping.

### **Show the GPS track :**

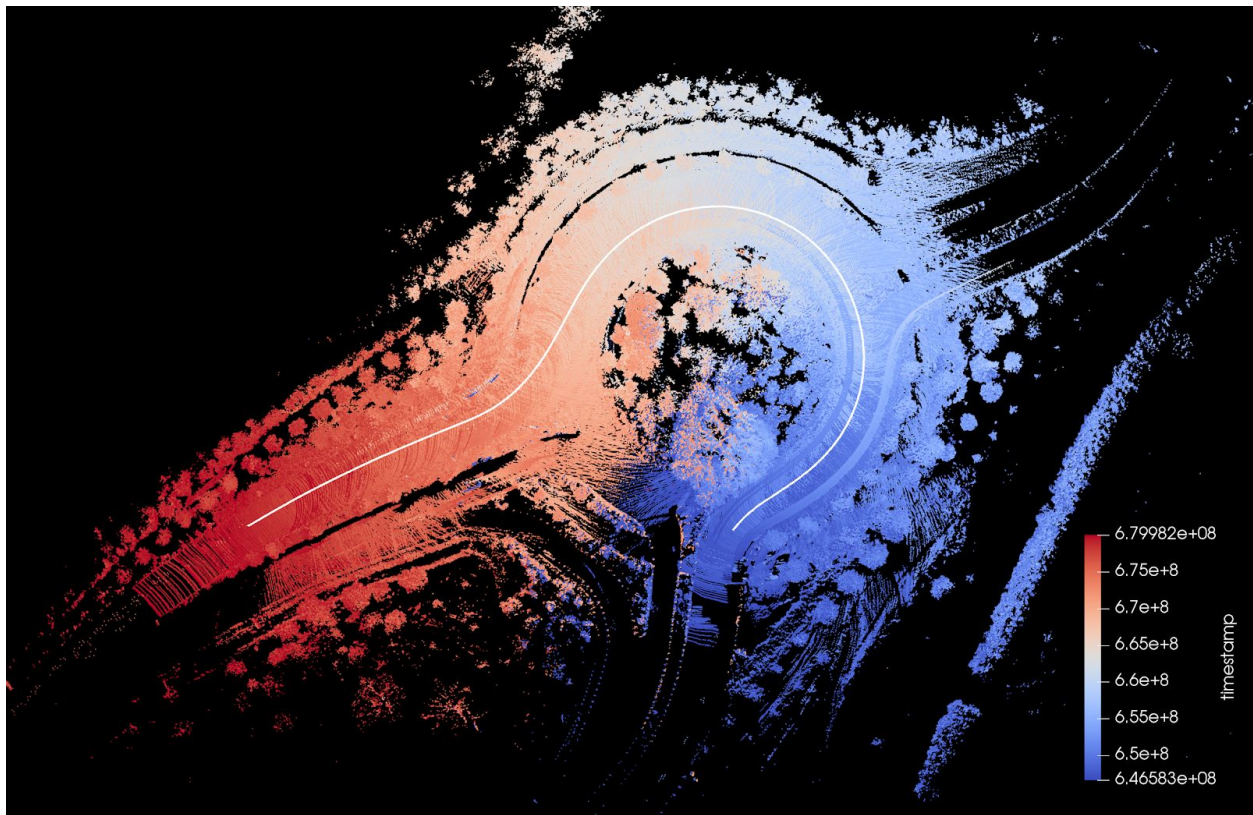


Display the GPS trajectory associated with LiDAR data.

### **Frame mapping :**

Frame Mapping:

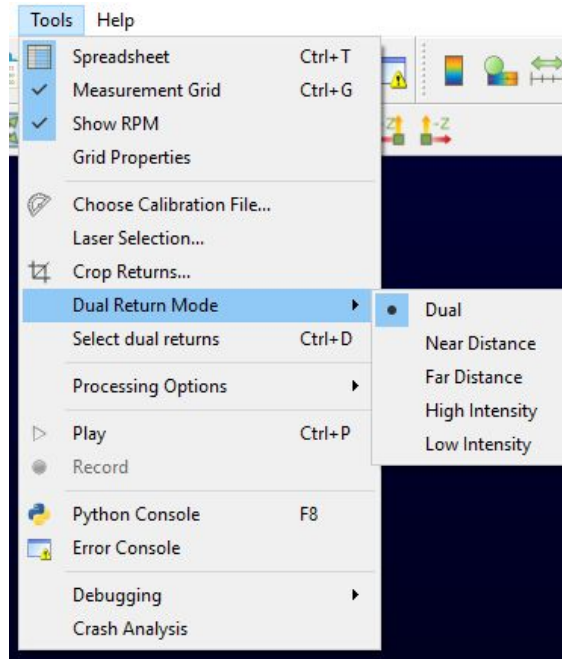
Manage how to match LiDAR frames to GPS pose, and how to transform their coordinates to display them.



**Figure 25 :** Aggregated frames with GPS track

### **D-6 Tools Menu**

The *Tools* menu provides additional functionalities to visualize and analyze the data. Some functionalities located in the tool menu are also available in one of the controls toolbar we just introduced.



**Figure 26 : Tools menu**

#### **Spreadsheet:**

Open the spreadsheet view.

#### **Measurement Grid:**

Show or hide the measurement grid on the main view.

#### **Show RPM:**

Display the Rotation Per Minute (RPM) of the current frame. If the trailing frame mode is on, the displayed RPM is the one of the last frame.

This spinning speed is an approximation, computed from points timestamps only, and it does not reflect the actual motor RPM. It is computed as such for each frame:

$$RPM = \frac{60}{[Timestamp (seconds) of last shown point] - [Timestamp (seconds) of first shown point]}$$

#### **Grid Properties:**

Open a dialog to change the properties of the measurement grid : number of ticks, dimensions of the grid, size of a tick, ...

#### **Choose Calibration File:**

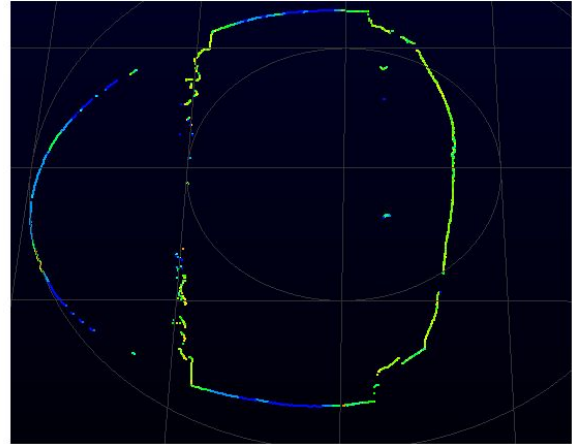
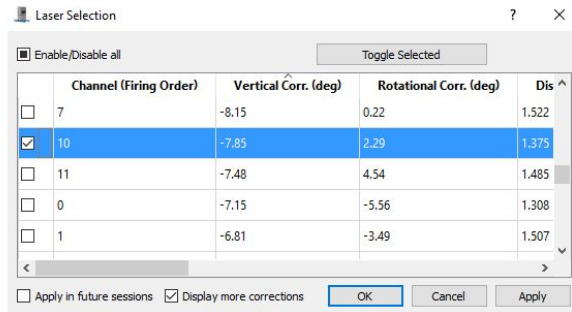
Open the calibration file dialog that we already introduced.

#### **Laser Selection:**

Open a dialog to select and show calibration information of the lasers (see **figure 27**).

This is useful to display only a subset of the lasers, or to know the internal calibration values attached to each laser (such as vertical angle, horizontal angle, ... ).





**Figure 27** : *Left*: laser selection dialog. *Right*: point cloud with only one laser selected

### **Crop Returns:**

Open the crop return dialog (see section **D-1 Basic controls toolbar**).

### **Dual return mode:**

Choose the dual return mode that you are interested in.

### **Processing options:**

Some options that are used to decode the raw sensor packets into 3D points:

- *Apply HDL-64 intensity corrections* : for HDL-64 only, correct the raw intensity values as per device specs (depending on distance and calibration).
- *Apply VLS-128 Fast Rendering*: Use a special mapper to accelerate the rendering, but prevent from using a custom colormap anymore. It is reported to improve rendering on laptops showing live VLS-128 data.
- *Ignore returns with distance equal to 0* : do not display nor add points with coordinates (0, 0, 0).
- *Hide points marked with drop flag* : do not display points received with a drop flag set to ON.
- *Adjust intra-firing timings and azimuth* : interpolate time and azimuth between two firings, using live computed sensor rotation speed and precise per channel firing-pattern timings specified in datasheets.
- *Ignore empty frames* : when reading a *.pcap*, in case a frame has 0 points (possible only if *ignore returns with distance equal to 0* is enabled), it is removed and its timestep is deleted : it behaves as if this frame doesn't exist at all.

## **III- Python Console**

### **A- Introduction**

VeloView is the software provided by Velodyne to visualize, analyze and record data from Velodyne sensors. As with ParaView on which it is based, it is possible to create some task automation pipelines or

small filters using the Python Shell console in VeloView. Another convenient tool is the “*VeloView --script*” command directly derived from ParaView. It is possible to launch VeloView with the following argument “*VeloView --script path\_to\_my\_script.py*” to directly run a python script when VeloView is launched. However, the automation is limited by functions which will require a user interaction such as choosing a sensor calibration. This manual goes through accessing data and programming into the python console. Then, we will show some examples of python programming using VeloView specific functions. Finally, we will expose a non-exhaustive list of VeloView-python features.

## B- Presentation of the Python Shell

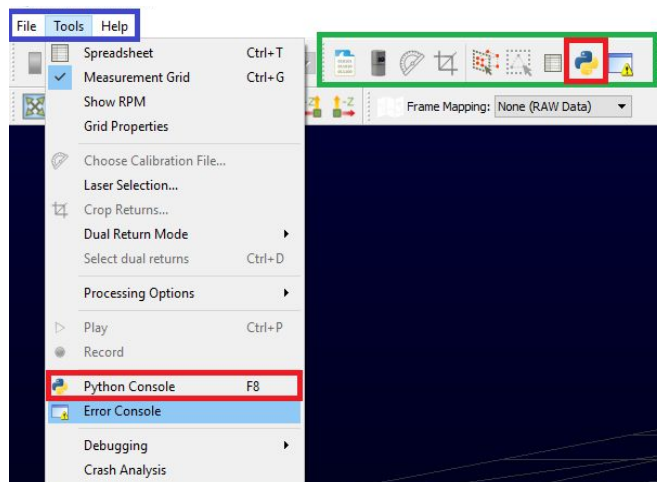
### B-1 Introduction to python-VeloView functionalities

The python console is a python environment available in VeloView which gives you access to some python, paraview, Qt and VeloView functionalities. The VeloView functionalities give access to the data provided by the Velodyne sensors, and gives some basic manipulation and analysis functions. Paraview provides lots of built-in filters to be applied on the data. Qt is in charge of the graphical user interface (GUI), and can be controlled with python too. Finally, you get all the usual Python scripting functionalities for task automation.

### B-2 Opening the python console

Once a VeloView instance is running, the *Python Console* can be opened in three different ways :

1. By clicking on the python icon in the *Basic Controls* bar (see **Figure 28**)
2. By clicking on the python icon in *Tools > Python Console* in the menu bar (see **Figure 28**)
3. By pressing F8 on the keyboard

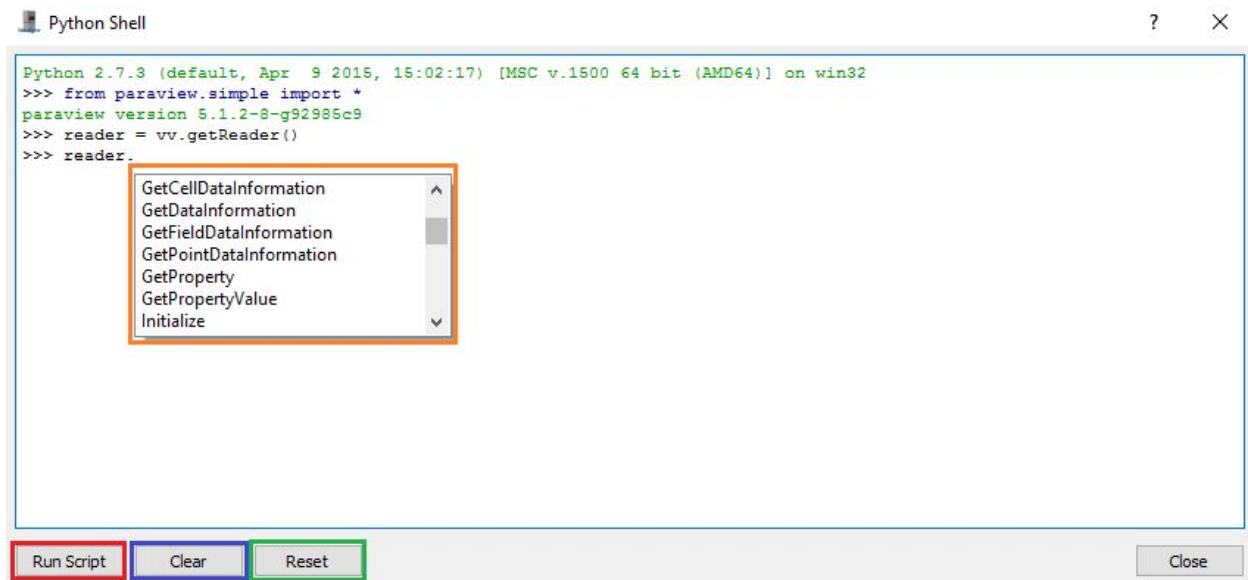


**Figure 28** : How to access the python console. Blue frame represents the menu bar. Green frame represents the *Basic Controls* bar. Red frames represent the accesses to the python console.

### B-3 Presentation of the python shell

Once you opened the python console using one of the three ways presented previously, a python shell should be opened (see **figure 29**). The python console / shell presents 4 different options :

- 1- *Run Script* : Allow you to load and run a python script.
- 2- *Clear* : Clear the console. The information of the console is cleared but the state remains. All your variables and data will not be removed.
- 3- *Reset* : Reset the console. All the variables and data will be removed. Nethertheless, the console won't be cleared. If you want to restart with a fresh new console you can Clear and Reset the console.
- 4- *Tab* : Pressing the *Tab* key in the Python Shell shows the available autocompletion.



**Figure 29** : The Python Shell, *Red*: Run script option, *Blue*: Clear option, *Green*: Reset option, *Orange* : Autocompletion option

## D- VeloView and VTK glossary and data model

In VeloView the data is represented as a VTK *vtkPolyData* object. It is a dataset containing two things: the 3D geometry (the list of 3D points provided by the velodyne sensor) and the extra data associated to these points (intensity, timestamp, azimuth, laser ID, ...). The association of the points and their corresponding data is called the polydata or dataset.

The available readers in VeloView read the data from the network (streaming reader) or from *.pcap* files (pcap reader) and provide the *vtkPolyData* corresponding to the data. Hence, when we get the output of a reader in the Python Shell we get a *vtkPolyData* object. The structure of a *vtkPolyData* can be found here : <http://www.vtk.org/doc/release/5.4/html/a01260.html>.

What should be retained is that the geometric points are accessible through the method *“.GetPoints()”* and the extra data are stored in arrays accessible through the method *“.GetPointData()”*.

About the *“.GetClientSideObject()”*:

Short story: You don't need to understand the client/server paradigm: just base yourself on examples and auto-completion.

Long story: Usually the VeloView functions will provide a servermanager containing the object you want.

It is because VeloView is directly derived from the Paraview Client / Server architecture. When you get a reader with the function “*lv.getReader()*” for instance, you will get a servermanager instance on the reader. If you want to get the output or have access to the method of the object you must use the “*GetClientSideObject()*” function. For instance, if you want to get the *vtkPolyData* output of the reader you must write “*getReader().GetClientSideObject().GetOutput()*”.

## E- Examples

### E-1- Get data information

The first example will show you how to get the point cloud generated by the Velodyne sensor.

**Step 1 :** Open the provided *.pcap* example: “*roundabout.pcap*”

**Step 2 :** Open the Python Shell

**Step 3 :** Write these python code lines :

```
reader = lv.getReader()           # Get the .pcap reader
cloudInfo = reader.GetClientSideObject().GetOutput() # Get the output of the reader stored in a vtkPolyData object
points = cloudInfo.GetPoints()    # Get the 3D points list
points.GetPoint(10)               # Coordinates of the 10th point
times = cloudInfo.GetPointData().GetArray("timestamp") # Get the time information of the points
times.GetValue(10)                # Value of the time information for the 10th point
```

**Explanation :** In this example we get the point cloud associated to the first frame of the “*roundabout.pcap*” data, and we print the coordinates and timestamp values of the 10th point of the 1st frame.

1- As you can see, all the available functions in VeloView-python are stored in the module named “*lv*”. For instance, if you want to access the function stored in the module *vtk*, you must write : *lv.vtk.DesiredFunction()*.

**Note :** For retro-compatibility reasons, the alias “*vv*” can also be used instead of “*lv*”.

2- *getReader()* is the VeloView function which gives you the *.pcap* reader. The *.pcap* reader is a *vtk* filter whose output is the dataset of the current frame. The output dataset is stored in a *vtkPolyData* object. It contains the 3D points list and the extra data associated with these points.

3- The method *GetPointData()* of the *vtkPolyData* class gives you access to the data arrays stored in the dataset. Using the method *GetArray(“ArrayName”)*, you will have access to the data of the corresponding array. The exact names of the arrays available in the point cloud are visible in the spreadsheet view (see **Figure 30**). The available arrays depend on the data (type of Velodyne sensor, GPS availability, IMU availability, extra data, ...).

Showing Data ▾ Attribute: Point Data ▾ Precision: 3 ▾ F [ ] [ ] [ ]

	Point ID	X	Y	Z	adjustedtime	azimuth	distance_m	intensity	laser_id	timestamp	vertical_angle
0	0	0.036	5.372	-1.440	646182671.000	38	5.562	50	0	646182671	-15.000
1	1	0.207	30.451	0.532	646182673.000	39	30.456	10	1	646182673	1.000
2	2	0.042	5.967	-1.378	646182675.000	40	6.124	2	2	646182675	-13.000
3	3	0.571	77.843	4.080	646182677.000	42	77.952	63	3	646182677	3.000
4	4	0.051	6.765	-1.315	646182680.000	43	6.892	3	4	646182680	-11.000
5	5	0.062	7.723	-1.223	646182684.000	46	7.820	10	6	646182684	-9.000
6	6	0.076	9.060	-1.112	646182689.000	48	9.128	4	8	646182689	-7.000
7	7	0.095	10.888	-0.953	646182694.000	50	10.930	1	10	646182694	-5.000
8	8	0.126	13.587	-0.712	646182698.000	53	13.606	1	12	646182698	-3.000
9	9	0.179	18.288	-0.319	646182703.000	56	18.292	1	14	646182703	-1.000
10	10	0.064	5.380	-1.442	646182726.000	68	5.570	50	0	646182726	-15.000
11	11									646182728	1.000
12	12									646182730	-13.000
13	13									646182733	3.000
14	14									646182735	-11.000
15	15									646182737	5.000
16	16									646182740	-9.000
17	17									646182744	-7.000
18	18									646182749	-5.000
19	19									646182753	-3.000
20	20									646182758	-1.000
21	21									646182781	-15.000
22	22									646182783	1.000
23	23									646182786	-13.000
24	24									646182788	3.000
25	25									646182790	-11.000

Python Shell

```

Python 2.7.3 (default, Apr 9 2015, 15:02:17) [MSC v.1500 64 bit (AMD64)] on win32
>>> from paraview.simple import *
paraview version 5.1.2-8-g92985c9
>>> reader = vv.getReader()

>>> cloudInfo = reader.GetClientSideObject().GetOutput()
>>> points = cloudInfo.GetPoints()
>>> times = cloudInfo.GetPointData().GetArray("timestamp")
>>> times.GetValue(10)
646182726
>>> points.GetPoint(10)
(0.06385207921266556, 5.379827976226807, -1.4416221380233765)
>>> points.GetPoint(10) # Coordinates of the 10 th point
(0.06385207921266556, 5.379827976226807, -1.4416221380233765)
>>>

```

Run Script Clear Reset Close

Figure 30 : Python shell and Spreadsheet view



## E-2- Select points using data

In this example we will select points that have a returned intensity over a defined threshold, and save the result in a .csv file.

**Step 1 :** Open the provided .pcap example: “roundabout.pcap”

**Step 2 :** Open the Python Shell

**Step 3 :** Write these python code lines :

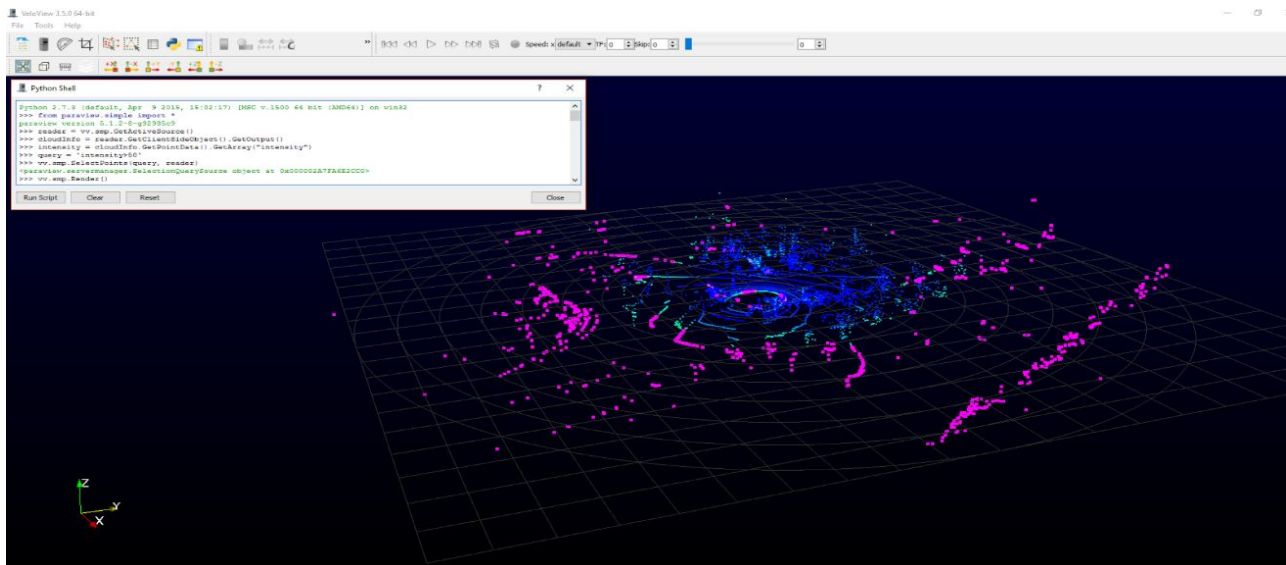
```
reader = lv.smp.GetActiveSource()           # Get the .pcap reader
query = 'intensity > 50'                   # Query selecting points whose intensity is over 50
lv.smp.SelectPoints(query, reader)          # Select the points which satisfies the query
lv.smp.Render()                             # Render the result
lv.saveCSVCurrentFrameSelection("D:\save.csv") # Save the selected points in .csv format
```

### Explanation :

1- *lv.smp.GetActiveSource()* : here we get the .pcap reader with the function *GetActiveSource()*. As you can see this function belongs to the module “smp” which is the module “paraview.simple” where all the paraview functions are stored. *GetActiveSource()* returns the active source, in our example the active source is the .pcap reader.

2- *smp.Render()* refreshes the active source once the data process is done. If you want to refresh another view (spreadsheet view or main view) you have to set the active source with *smp.SetActiveSource()*.

3- *lv.saveCSVCurrentFrameSelection("D:\save.csv")* saves the selected points in a .csv file located at the indicated path.



**Figure 31 :** Result of example #2. Pink points are the selected points.

### E-3- Display a 3D model in the mainview

In this example we load and display a 3D model in the main view

**Step 1 :** Open the provided .pcap example: “roundabout.pcap”

**Step 2 :** Open the Python Shell

**Step 3 :** Write these python code lines :

```
Model = lv.smp.OpenDataFile("D:/PythonManualData/BMWX54.obj")
ModelOriented = lv.smp.Transform(Input = Model)
ModelOriented.Transform = 'Transform'
ModelOriented.Transform.Scale = [1.0/30.0, 1.0/30.0, 1.0/30.0]
ModelOriented.Transform.Rotate = [90, -102, 45]
ModelOriented.Transform.Translate = [-0.69, -0.25, -1.80]
lv.smp.Show(ModelOriented)
lv.smp.Render()
```

#### Explanation :

1- *smp.OpenDataFile("D:/PythonManualData/BMWX54.obj")* open the .obj model.

2- *Scale, Rotate, Translate* : Since the model is given in an arbitrary frame, we must rescale and set the orientation of the model to fit our data.

3- *Show(), Render()* : We show the result in the main view and then render the result to display the model.

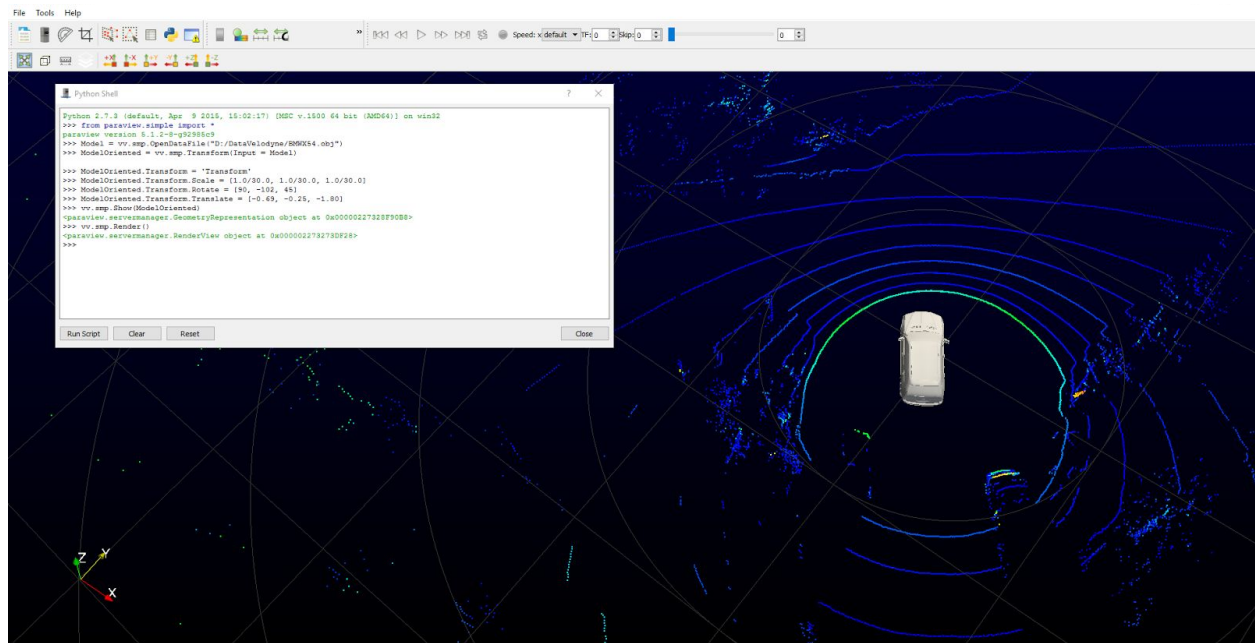


Figure 32 : The model is added in the main view

#### E-4- Start VeloView with a 3D model loaded in the mainview with “- -script”

In this example we load and display a 3D model in the mainview directly when VeloView is launched using the --script command-line option. We use the script LoadModel.py written in the previous example **E-3- Display a 3D model in the mainview**.

**Step 1 :** Open a terminal in the folder containing VeloView executable. You can do that on Windows by pressing the key  + R and type “cmd”.

**Step 2 :** launch VeloView from the terminal with the command:

```
./VeloView --script='D:\PythonManualData\LoadModel.py'
```

Of course, you need to replace “D:\PythonManualData\LoadModel.py” with the path to the file LoadModel.py. You also must replace the model filename contained in the file LoadModel.py.

You should now see VeloView start, and display the 3D model already loaded within the main view as in the previous example.

#### E-5- Handling multiple lidar sensor using python shell

In this example we show how to display the stream of several lidars in VeloView.

**Step 1:** Open the Python Shell

**Step 2:** Instantiate as many stream as you want :

```
lidar = LidarStream()  
...  
lidar = LidarStream()
```

**Step 3:** Fill all the information needed by the Stream for all stream you have instantiate

The mandatory fields are :

- *ListeningPort* : Port number where the corresponding lidar send its lidar packets
- *CalibrationFile* : Calibration file corresponding to the sensor
- *Interpreter* : Have to be the ‘Velodyne Meta Interpreter’ (which selects the appropriate one)

1) You can set them using the User Interface (After Step 2) :

- Open the “Pipeline Browser” and the “Properties” panel (in the “Views” menu)
- Select in the “Pipeline Browser” the stream to which you want to set the parameters
- Fill the “Listening Port” information, select the calibration file corresponding to your data and select the “Velodyne Meta Interpreter”
- Press “Apply” then “Start”

2) You can set these parameters using python :

```
calibrationFile = " "      # Path to calibration file of the lidar
lidarPort = 2368          # Port number where the lidar send its lidar packets
N = 3                    # Number of lidar stream to instantiate

for i in range(N):
    lidar = LidarStream()
    lidar.CalibrationFile = calibrationFile
    lidar.Interpreter = 'Velodyne Meta Interpreter'
    lidar.ListeningPort = lidarPort + i
    lidar.UpdatePipeline()
    Show(lidar)
    lidar.Start()
```

**Step 4:** Fill the optional information you need :

- 1) Throw the UI : With the “Pipeline Browser” and the “Properties” panel (in the “Views” menu):
  - Select the stream that you want
  - Set the optional parameters you want (“Enable Advanced Array”, “Sensor Transform”, “Is forwarding”, ...)

2) Apply a transform using python :

```
calibrationFile = " "      # Path to calibration file of the lidar
lidarPort = 2368          # Port number where the lidar send its lidar packets
N = 3                    # Number of lidar stream to instantiate

for i in range(N):
    lidar = LidarStream()
    lidar.CalibrationFile = calibrationFile
    lidar.Interpreter = 'Velodyne Meta Interpreter'
    lidar.Interpreter.SensorTransform = 'Transform2'
    lidar.Interpreter.SensorTransform.Scale = [1.0/2.0, 1.0/2.0, 1.0/2.0]
    lidar.Interpreter.SensorTransform.Rotate = [0 , 90 * i, 0]
    lidar.Interpreter.SensorTransform.Translate = [i, i, i]
    lidar.ListeningPort = lidarPort + i
    lidar.UpdatePipeline()
    Show(lidar)
    lidar.Start()
```

## F- List of functionalities

This section gives a non-exhaustive list of available functionalities in Python-VeloView.

### F-1- Simple Paraview : “lv.smp.”

All functionalities provided by the module *paraview.simple* are available using “lv.smp.”. You can find the description and documentation of the functionalities here :

<https://kitware.github.io/paraview-docs/latest/python/paraview.simple.html>

### F-2- Qt “lv.QtGui”

All functionalities provided by the module *QtGui* are available using “lv.QtGui.”. You can find the description and documentation of the functionalities here :

<https://www.riverbankcomputing.com/static/Docs/PyQt5/>

### F-3- Vtk Python

All functionalities provided by the module *vtk* are available using “lv.vtk.”. You can find the description and documentation of the functionalities here :

<http://www.vtk.org/Wiki/VTK/Examples/Python>

### F-4- VeloView functions “lv.”

Following is a non-exhaustive list of common functions that are available in the module “lv.”. For more functionalities, don’t hesitate to use auto-completion to get access to all available features.

**getReader()** : Return the .pcap reader which contains the point cloud of the current frame.

**openSensor()** : Instantiate a UDP socket to receive data from velodyne-HDL sensor.

**openPCAP(filename)** : Create a .pcap reader to read the data contained in the file located at *filename*.

**hideMeasurementGrid()** : Hide the measurement grid.

**showMeasurementGrid()** : Show the measurement grid.

**saveCSVCurrentFrame(filename)** : Export the current frame into a .csv file.

**saveCSVCurrentFrameSelection(filename)** : Export the current selection of the current frame into a .csv file.

**recordFile(filename)** : Record the data received in live stream.

**stopRecording()** : Stop the recording.

**startStream()** : Start the listening of the UDP socket.

**stopStream()** : Stop the listening of the UDP socket.

**getSensor()** : Get the live streaming reader. The output of this reader is the 100 last received point clouds.

**getPosition()** : Get the position reader. The output of this reader is the GPS / IMU information contained in the data.

**onCropReturns(show = True)** : Open the crop dialog box.

**resetCamera()** : Reset the camera.

**saveScreenshot(filename)** : Save a screenshot.

**showGrid()** : Show the grid.

**hideGrid()** : Hide the grid.

**getMainWindow()** : Get the main view (the view where the point cloud is displayed).

**toggleSelectDualReturn()** : Select the dual return of the current selection.



**setFilterToDual()** : Show only the dual return.  
**setFilterToDistanceNear()** : Show nearest returns.  
**setFilterToDistanceFar()** : Show farthest returns.  
**setFilterToIntensityHigh()** : Show highest intensity returns.  
**setFilterToIntensityLow()** : Show lowest intensity returns.  
**setTransformMode(mode)** : Change the transform mode : 0) raw, 1) absolute, 2) relative.  
**lv.app.reader** : Get the .pcap reader.  
**lv.app.sensor** : Get the streaming reader.

**lv.app.distanceResolutionM** : Distance resolution of the sensor.  
**lv.app.trailingFramesSpinBox** : GUI spinbox which determines the number of trailing frames.  
**lv.app.geolocationToolBar** : GUI geolocation toolbar.  
**lv.app.mainView** : The main view (the one displaying the point cloud).  
**lv.app.gridProperties** : Properties of the grid.  
**lv.app.grid** : The displayed grid in the main view.  
**lv.app.scene** : Displayed scene in the main view.

Additional information can be found in your **Velodyne sensor's user manual**, available here:  
<https://velodynelidar.com/downloads/#manuals>. See Appendix E.

For questions, issues, and feedback, use the **Contact Technical Support form** here:  
<https://velodynelidar.com/contact-us/>.