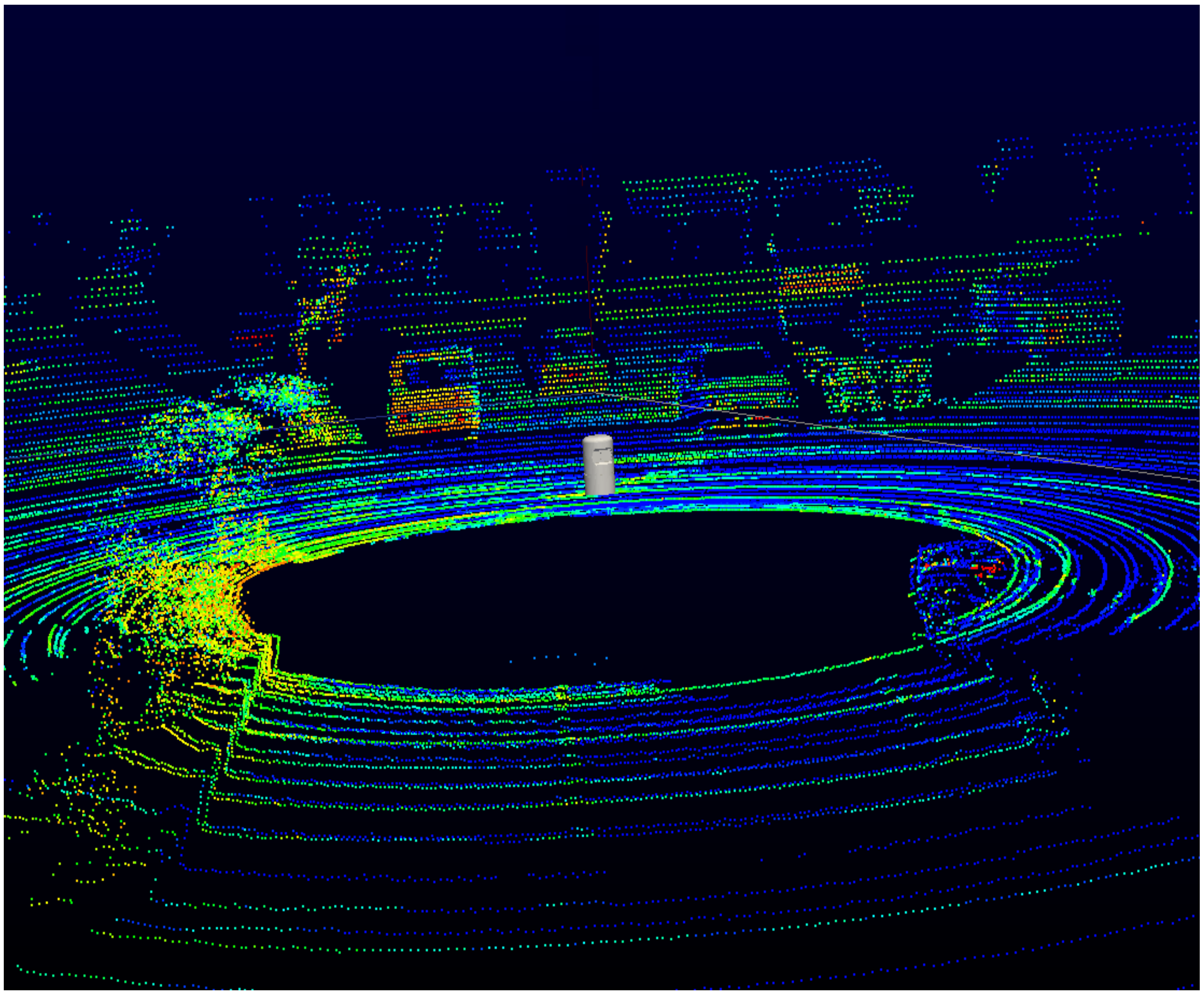
VeloView User Guide

User documentation for the VeloView application and libraries



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

Version: 3.5.

Velodyne LiDAR

Contents

I- Introduction

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

In a first part of the manual we will focus on how to get, visualize and analyze Velodyne sensors data using the graphical interface of VeloView. In a 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

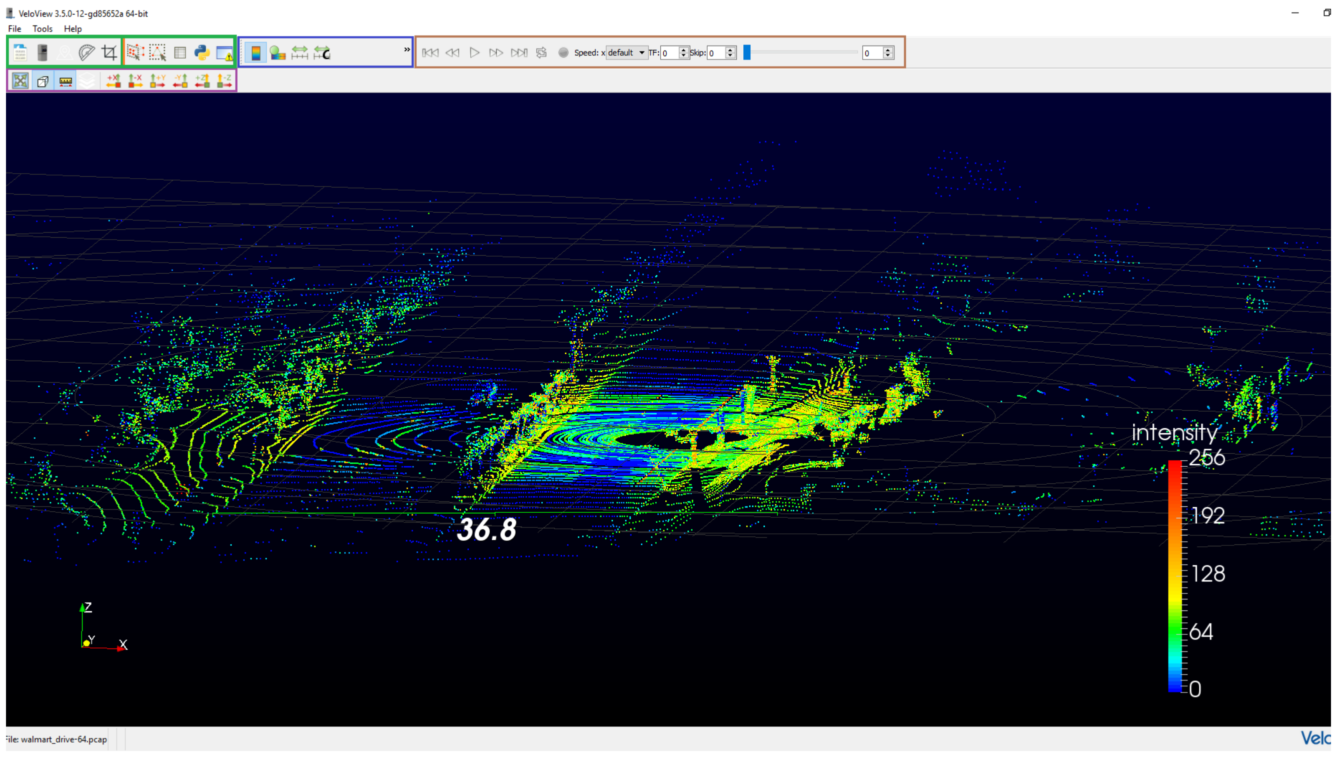


Figure 1 : Global view of VeloView : Green: Basic Control toolbar, blue: Color Control toolbar, brown: Display toolbar, purple: View Control

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

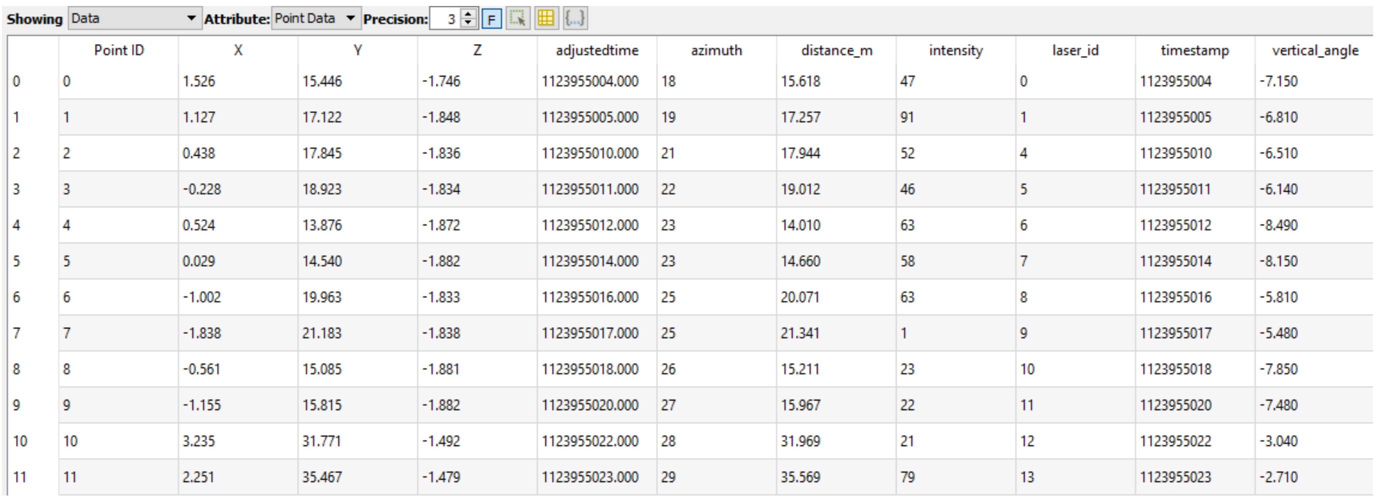
First, we will introduce the VeloView graphical user interface (GUI). The VeloView GUI is composed of four toolbars (introduced in section **D- VeloView toolbars**) and three different views which are called main view, spreadsheet view and sensor track view.

A-1 Main view

The main view is responsible of the 3D display of the point-cloud in an interactive viewer.

A-2 Spreadsheet view

Like the main view, the spreadsheet view displays the point-cloud sensor data. However, the data are not displayed as a point cloud but as a spreadsheet (see **figure 2**).



Showing Data **Attribute:** Point Data **Precision:** 3

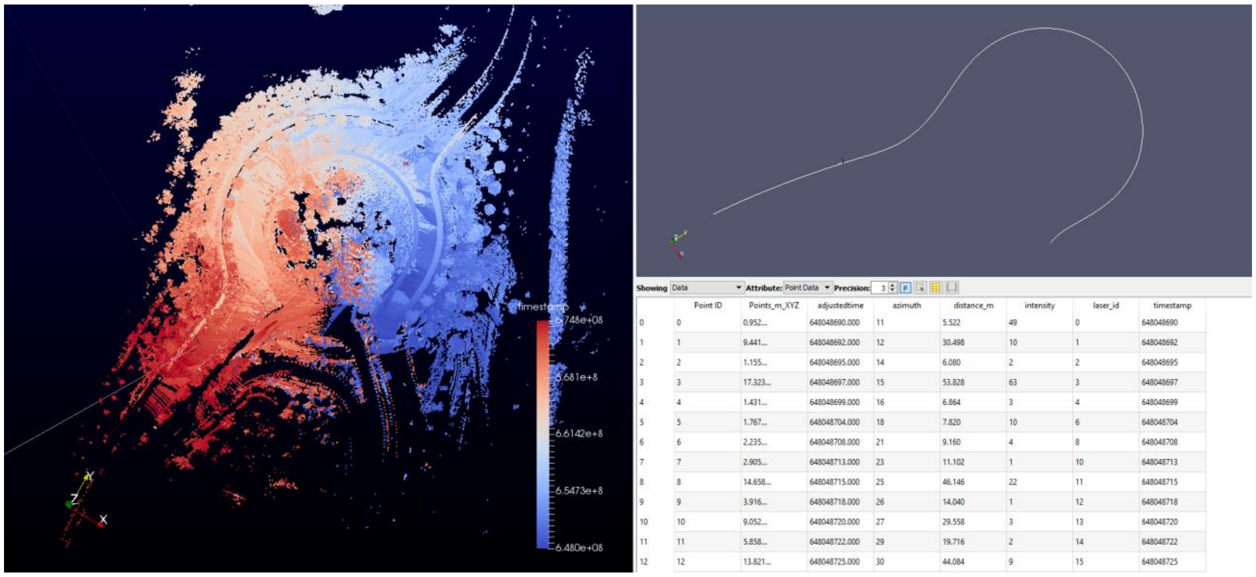
	Point ID	X	Y	Z	adjustedtime	azimuth	distance_m	intensity	laser_id	timestamp	vertical_angle
0	0	1.526	15.446	-1.746	1123955004.000	18	15.618	47	0	1123955004	-7.150
1	1	1.127	17.122	-1.848	1123955005.000	19	17.257	91	1	1123955005	-6.810
2	2	0.438	17.845	-1.836	1123955010.000	21	17.944	52	4	1123955010	-6.510
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

In the spreadsheet view you can select the source of data to be displayed (Velodyne sensor, gps sensor, IMU, measurement grid, …), change the digit precision, show only the selected points.
Notice that the rows are sortable if you click the column header.

D-3 Sensor track view

The Sensor track view displayed the GPS / IMU or slam position and orientation (see **figure 3**)



	Point ID	Points_m_XYZ	adjustedtime	azimuth	distance_m	intensity	laser_id	timestamp
0	0	0.952...	648048690.000	11	5.522	49	0	648048690
1	1	9.441...	648048692.000	12	30.498	10	1	648048692
2	2	1.155...	648048695.000	14	6.080	2	2	648048695
3	3	17.323...	648048697.000	15	53.828	63	3	648048697
4	4	1.431...	648048699.000	16	6.864	3	4	648048699
5	5	1.767...	648048704.000	18	7.820	10	6	648048704
6	6	2.235...	648048708.000	21	9.160	4	8	648048708
7	7	2.905...	648048713.000	23	11.102	1	10	648048713
8	8	14.658...	648048715.000	25	46.146	22	11	648048715
9	9	3.916...	648048718.000	26	14.040	1	12	648048718
10	10	9.052...	648048720.000	27	29.558	3	13	648048720
11	11	5.858...	648048722.000	29	19.716	2	14	648048722
12	12	13.821...	648048725.000	30	44.084	9	15	648048725

Figure 3 : VeloView with the 3 views shown. Left : MainView displaying the point cloud. Top right : sensor track view displaying the gps path. Bottom right : spreadsheet view displaying the selected points.

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) 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**).

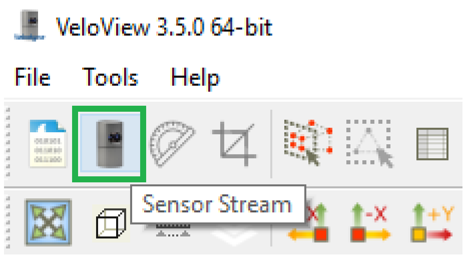
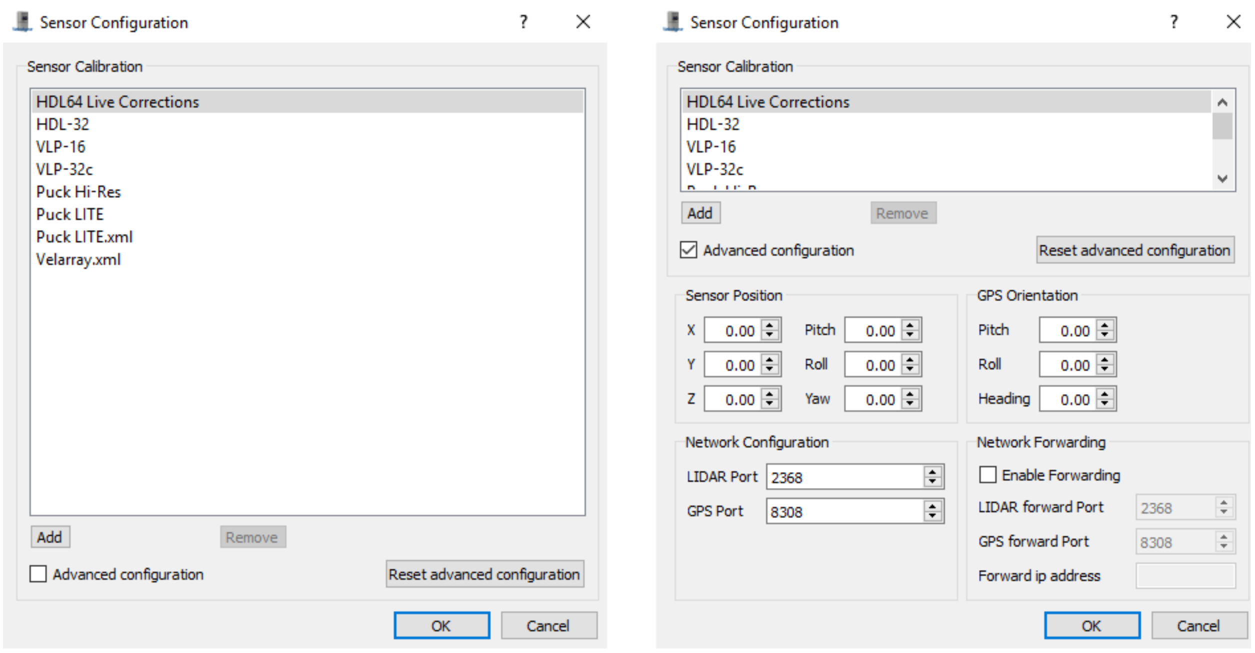


Figure4 : sensor stream action

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; and if you click on the advanced configuration checkbox you will be able to configure the GPS parameters and the network configuration.



Sensor Configuration

Sensor Calibration

HDL64 Live Corrections
HDL-32
VLP-16
VLP-32c
Puck Hi-Res
Puck LITE
Puck LITE.xml
Velarray.xml

☐ Advanced configuration

Sensor Configuration

Sensor Calibration

HDL64 Live Corrections
HDL-32
VLP-16
VLP-32c

☑ Advanced configuration

Sensor Position

X	0.00	Pitch	0.00
Y	0.00	Roll	0.00
Z	0.00	Yaw	0.00

GPS Orientation

Pitch	0.00
Roll	0.00
Heading	0.00

Network Configuration

LIDAR Port	2368
GPS Port	8308

Network Forwarding

☐ Enable Forwarding

LIDAR forward Port	2368
GPS forward Port	8308
Forward ip address	

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

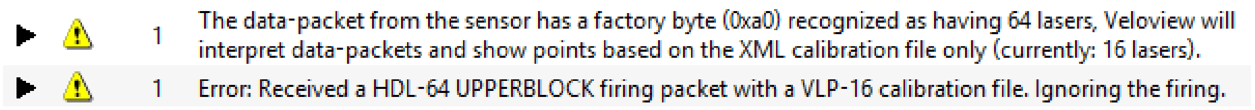
<u>Sensor Configuration:</u>

<u>Sensor Calibration file:</u>

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

Remark: In case you chose a wrong calibration file, here is some hints that indicate that you may have chosen a wrong calibration file:

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



```
▶ ⚠ 1 The data-packet from the sensor has a factory byte (0xa0) recognized as having 64 lasers, Veloview will
       interpret data-packets and show points based on the XML calibration file only (currently: 16 lasers).
▶ ⚠ 1 Error: Received a HDL-64 UPPERBLOCK firing packet with a VLP-16 calibration file. Ignoring the firing.
```

Figure 6 : Error message when number of lasers don't match

- The outputted point cloud should be distorted because of the wrong vertical angles corrections like in **figure 7**.

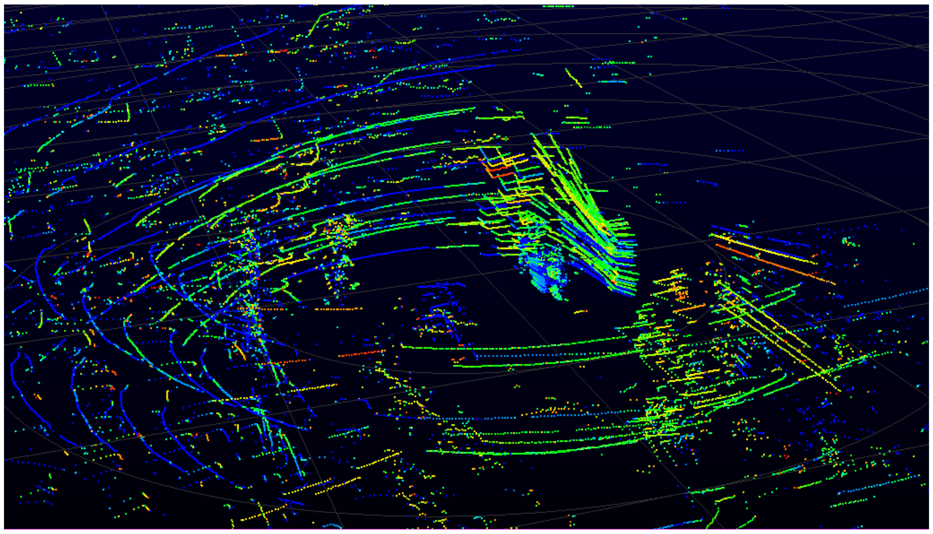
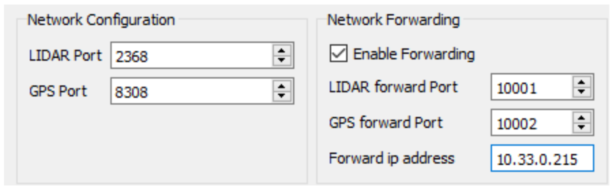


Figure 7 : Distorded point cloud due to wrong configuration file

<u>**GPS parameters (position and orientation):**</u>
In case you are using a GPS associated to the Velodyne sensor you should be careful to the Sensor position and the GPS Orientation. Since the GPS and the Velodyne sensors may not be 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 frame. 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.

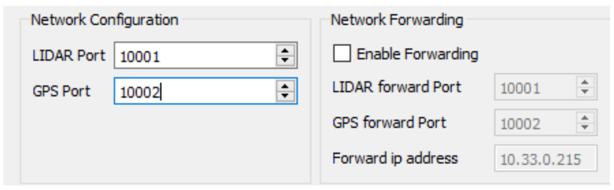
<u>**Network configuration:**</u>
You have the possibility to forward the lidar data to another computer station. For example you can have a first instance of VeloView which receives the sensor's data and forwards the data to another instance of VeloView on another computer at the address 10.33.0.215. To do so, the first instance of VeloView will received the data on the usual ports [2368; 8308] and forward the data to the ip address 10.33.0.215 and ports [10001; 10002] (you can choose other ports here). The second instance of

VeloView will received 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.



Network Configuration
LIDAR Port: 2368
GPS Port: 8308

Network Forwarding
☑ Enable Forwarding
LIDAR forward Port: 10001
GPS forward Port: 10002
Forward ip address: 10.33.0.215



Network Configuration
LIDAR Port: 10001
GPS Port: 10002

Network Forwarding
☐ Enable Forwarding
LIDAR forward Port: 10001
GPS forward Port: 10002
Forward ip address: 10.33.0.215

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 received the forwarded data on the ports [10001; 10002].

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

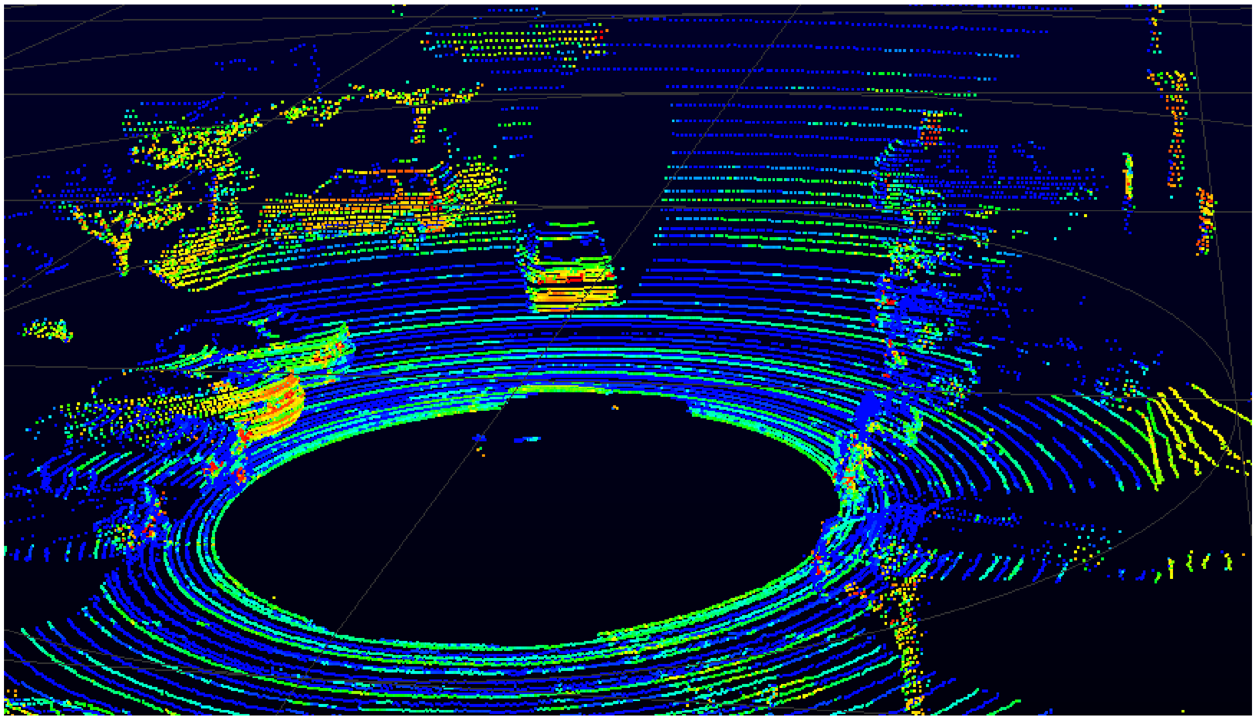


Figure 9 : Ouput of a HDL-64 correctly configured

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**).

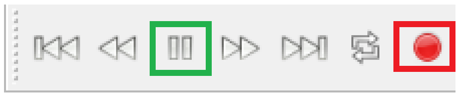


Figure 10 : green : button to pause the stream, red : button to record the stream

C- Playback mode (.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 11**).

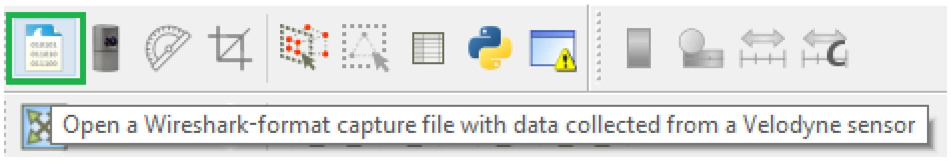

Open a Wireshark-format capture file with data collected from a Velodyne sensor

Figure 11 : 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.**

D- VeloView toolbars

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

D-1 Basic controls toolbar

The basic controls toolbar is located to the top right-hand corner of the VeloView GUI (see **figure 12**).

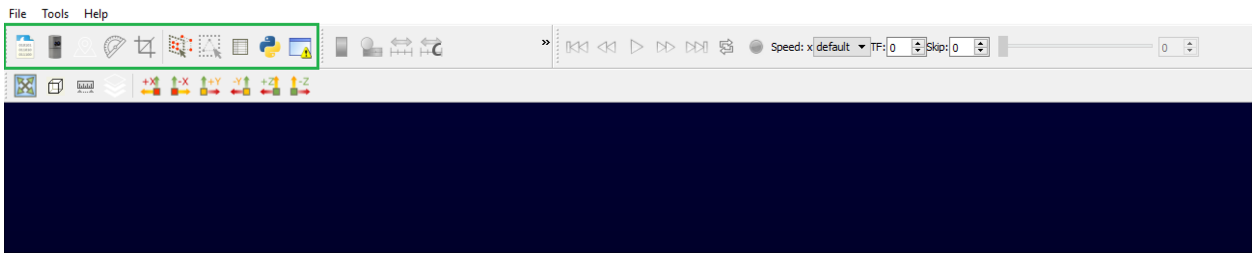


Figure 12 : Localization of the basic controls toolbar

Basic controls toolbar overview

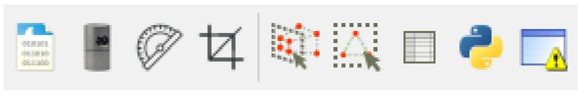


Figure 13 : Basic controls. From left to right : open .pcap, sensor stream, Choose calibration file, Crop returns, Select all points, Select dual returns, Spreadsheet view, python console, Error console

Open .pcap:

Open a .pcap file containing recorded Velodyne sensor data

Sensor stream:

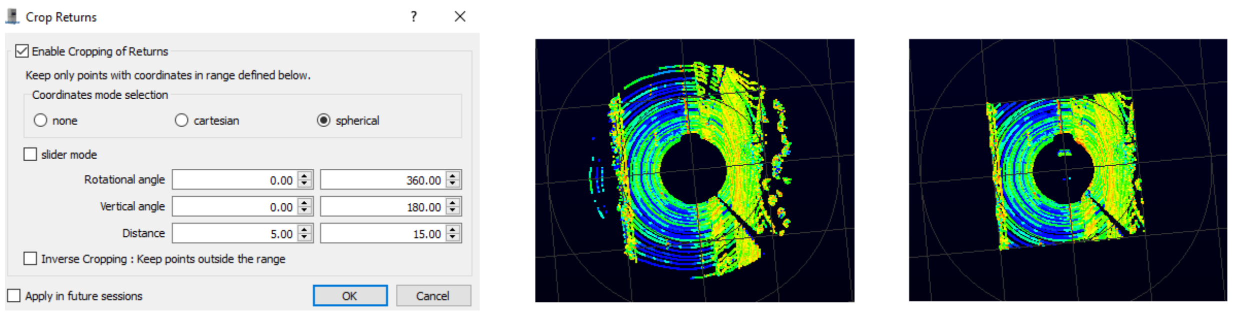
Open an UDP socket to receive the data of a connected sensor in real-time.

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 button 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 14**).



Crop Returns

☑ Enable Cropping of Returns

Keep only points with coordinates in range defined below.

Coordinates mode selection

none cartesian [✓ spherical]

☐ slider mode

Rotational angle	0.00	360.00
Vertical angle	0.00	180.00
Distance	5.00	15.00

☐ Inverse Cropping : Keep points outside the range

☐ Apply in future sessions

Figure 14 : Left : Crop return 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).

Select points:

Select points in a defined region. The region is defined with the mouse. Once drawn, the defined zone and viewpoint is 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 each selected points. If no selection exists, this operation will return the dual returns of all points.

Spreadsheet view:

Open the spreadsheet view.

Python console:

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

Error console:

open the error console

D-2 Display controls toolbar

You can affect the way the data are displayed in VeloView using the playback control tool bar (see **figure 15**).

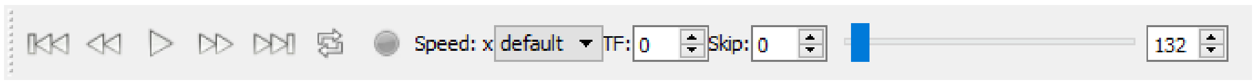


Figure 15 : playback control tool bar. From left to right : Go to start, Go previous frame, Start / Stop, Go next frame, Go to end, Loop, speed, Trailing frame, Skip and Select frame slider.

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

Trailing frame:
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 16**).

Skip:
Subsamples the points shown by keeping only one point every X point of the point cloud. It is useful if you are in trailing frame mode with a lot of point cloud accumulated.

Select frame slider:
Indicates the current frame. You can move to the frame you want using the slider or the spinbox.

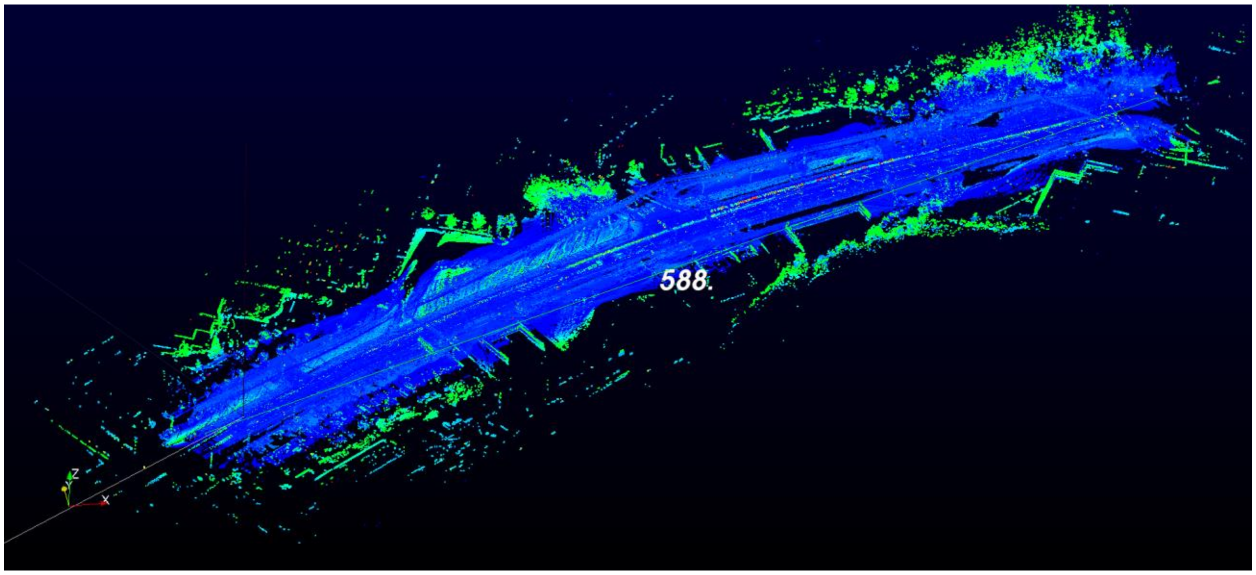


Fig 16 : Road section stabilized using GPS over 588 meters visualized using trailing frame.

D-3 Color controls toolbar

The color control tool (see **figure 16**) 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 color map or range of colored data.

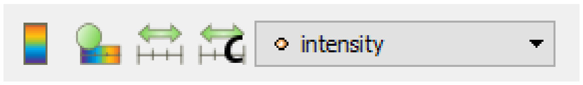


Figure 16 : from left to right : Color legend, Edit color map, Fit color range to data, Custom color range, color-coding data selection

Data selection:

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 17**).

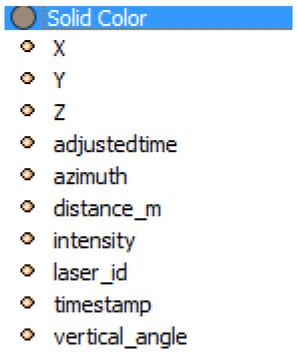


Figure 17 : avalaible data information for color-coding

Color legend:

Display the color legend within the Main 3D View (see **figure 18**).

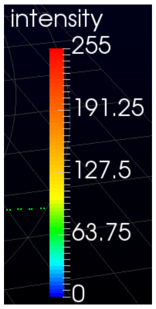


Figure 18 : color legend

Edit color map:
Edit and change the color mapping of the data.

Rescale data:
Rescale the color map so that the color extremities correspond to the data value extremities

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 a subrange values.

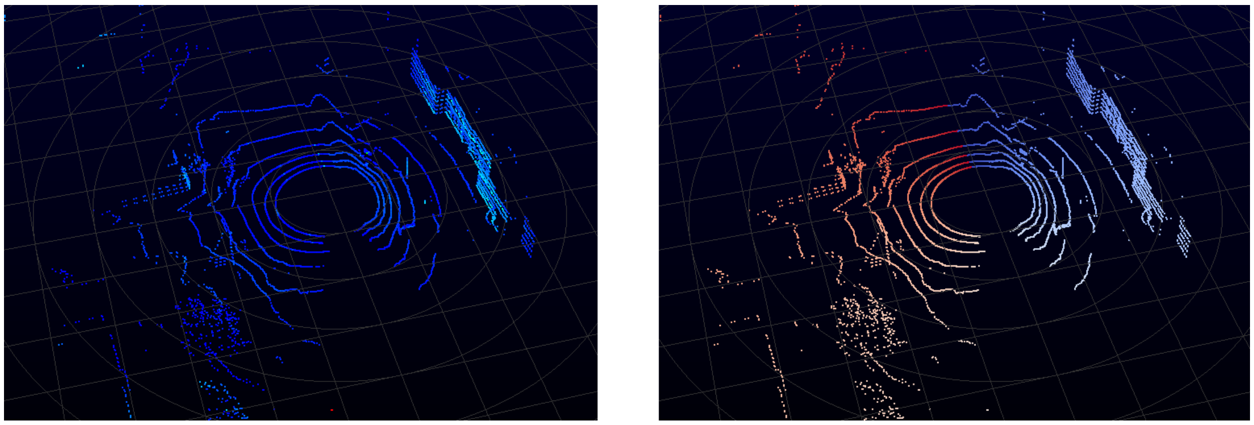


Figure 19 : same data displayed with different data information. Left : intensity, Right : azimuth

D-4 View Controls toolbar

The view control (see **figure 20**) contains tools to control the camera.

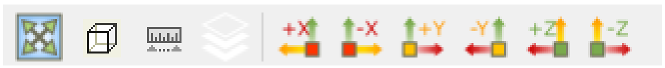


Figure 20: View Control. From left to right : Reset camera, Orthogonal view, measure distances, plane fit selection, set view direction to +x, -x, +y, -z, +z, -y

Reset Camera:
Reset the camera at its initial position and orientation

Orthogonal view:
Switch from a perspective to an orthogonal view

Measure distances:

Ctrl + left button of the mouse to draw a line and measure its distance. This functionality only works when the orthogonal view is activated (see **figure 21**)

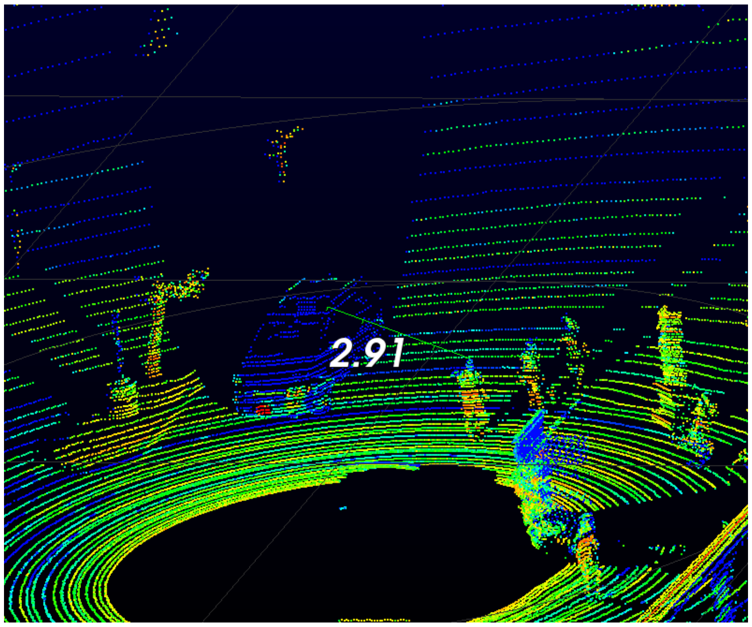


Figure 21: distance between car and pedestrian in meters

Plane fit tool:

Compute the plane which fits the currently selected points.

Set view direction to +x, -x, +y, -y, +z, -z:

Set the camera viewing direction to the x, y or z axis.

D-5 Tools Menu

The tool 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.

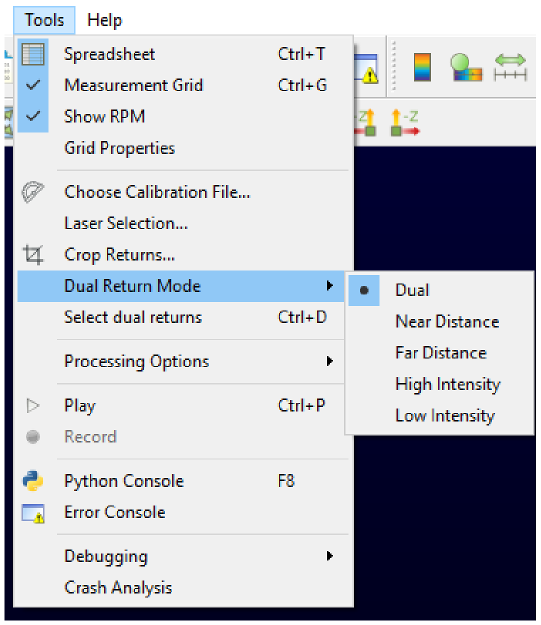


Figure 22 : Tool Menu

<u>Spreadsheet:</u>
Open the spreadsheet view

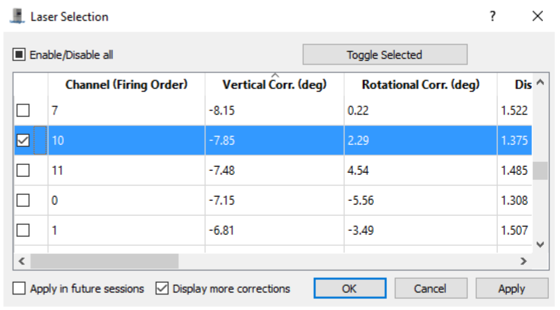
<u>Measurement Grid:</u>
Show or hide the measurement grid on the main view

<u>Show RPM:</u>
Display the Rotation Per Minute (RPM) of the current frame. If the trailing frame mode is on, the RPM displayed is the one of the last frame.

<u>Grid Properties:</u>
Open a dialog to change the properties of the measurement grid : number of ticks, size of the grid, size of a tick, …

<u>Choose Calibration File:</u>
Open the calibration file dialog that we already introduced

<u>Laser Selection:</u>
Open a dialog to select and show calibration information of the lasers (see **figure 23**).
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, …).



	Channel (Firing Order)	Vertical Corr. (deg)	Rotational Corr. (deg)	Dis
☐	7	-8.15	0.22	1.522
✓	10	-7.85	2.29	1.375
☐	11	-7.48	4.54	1.485
☐	0	-7.15	-5.56	1.308
☐	1	-6.81	-3.49	1.507

☐ Apply in future sessions ✓ Display more corrections

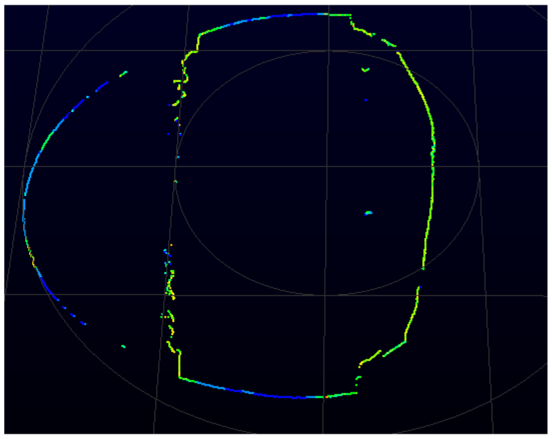


Figure 23 : Left laser selection dialog. Right: point cloud with only one laser selected

<u>Crop Returns:</u>
Open the crop return dialog (see section **D-1 Basic controls toolbar**)

<u>Dual return mode:</u>
Choose the dual return mode that you are interested on.

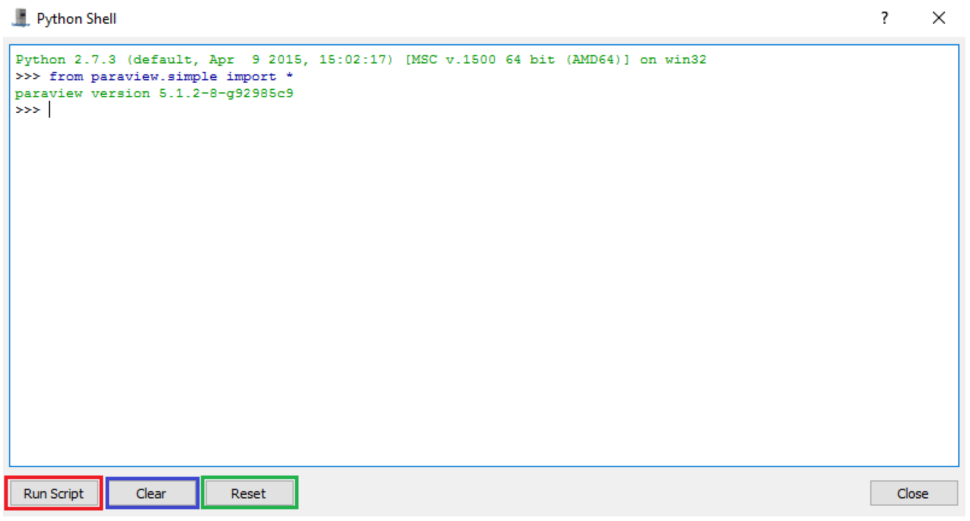
<u>Processing options:</u>
Some options that are used to decode the raw sensor packets into 3D points:

- Apply HDL-64 intensity corrections,
- Ignore returns with distance equals to 0,
- Adjust intra firing timings and azimuth (interpolate time and azimuth between two firings, using live computed sensor rotation speed)
- Ignore empty frames.

III- Python Console

A- Introduction

VeloView is the software provided by Velodyne to visualize, analyze and record data from velodyne-HDL 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” directly derived from ParaView. It is possible to launch VeloView with the following argument “VeloView –script path_to_my_script” to directly run a python script when VeloView is launched. However, the automation is limited by functions which will required an user interaction such as choosing a sensor calibration This manual goes through accessing data and programing 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 functionalities.



```
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
>>> |
```

B- Presentation of the Python Shell

B-1 Introduction to python-VeloView functionalities

The python console is a python environment available into VeloView which give you access to some python, paraview, Qt and VeloView functionalities. The VeloView functionalities give access to the data provided by the Velodyne-HDL 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 differents ways :

1- by clicking on the python icon in the “Basic Controls” bar (see **Figure 21**)
2- by clicking on the python icon in “Tools > Python Console” in the Menu Bar (see **Figure 21**)
3- by pressing F8 on the keyboard

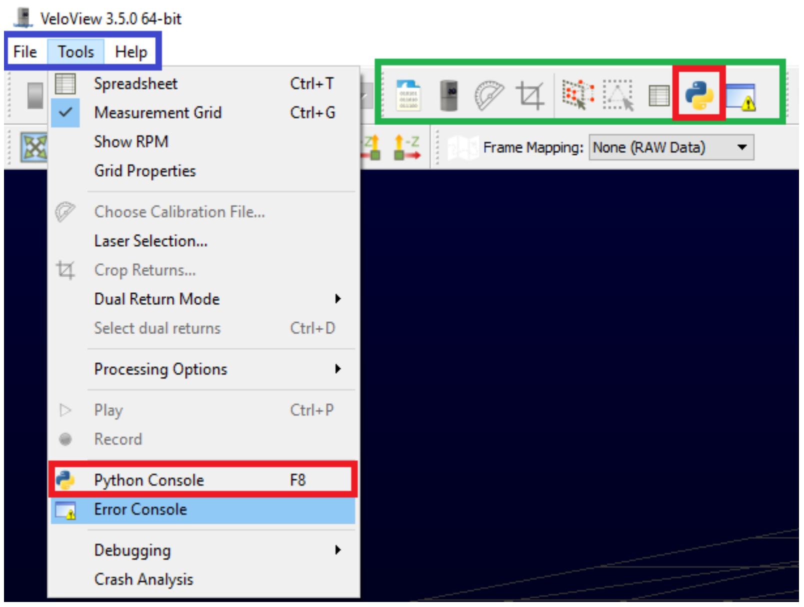


Figure 21 : How to access the python console. Blue frame represents the Menu Bar. Green frame represents the "Basic Controls" bar. Red frames represent the accesses to the python console

B-3 Presentation of the python shell

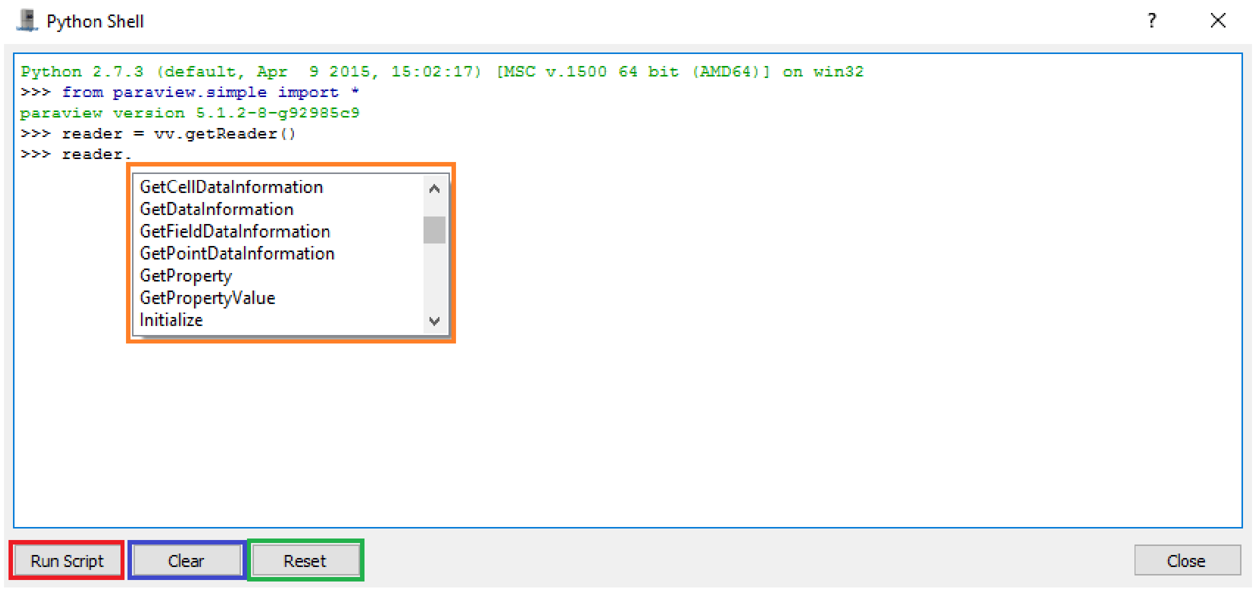
Once you opened the python console using one of the three ways presented previously, a python shell should be opened (see **figure 22**). The python console / shell presents 4 differents options :

1- Run Script : Allow you to load and run a python script

2- Clear : Clear the console. The information of the console are 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 the Python Shell shows you the available autocompletion.



```
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()
>>> reader.
```

Figure 22 : The Python Shell, Red: Run script option, Blue: Clear option, Green: Reset option, Orange : Autocompletion option

D- VeloView and VTK glossary and datamodel

In VeloView the data are represented as a VTK "vtkPolyData" class. 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 readers available in VeloView read the data from the network (streaming reader) or from .pcap files (pcap reader) and provides 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()".

On the "GetClientSideObject()":
TL;DR: 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 "vv.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 "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-HDL sensor.

Step 1 : Open the provided .pcap example: “roundabout.pcap”
Step 2 : Open the Python Shell
Step 3 : Write these python code lines :

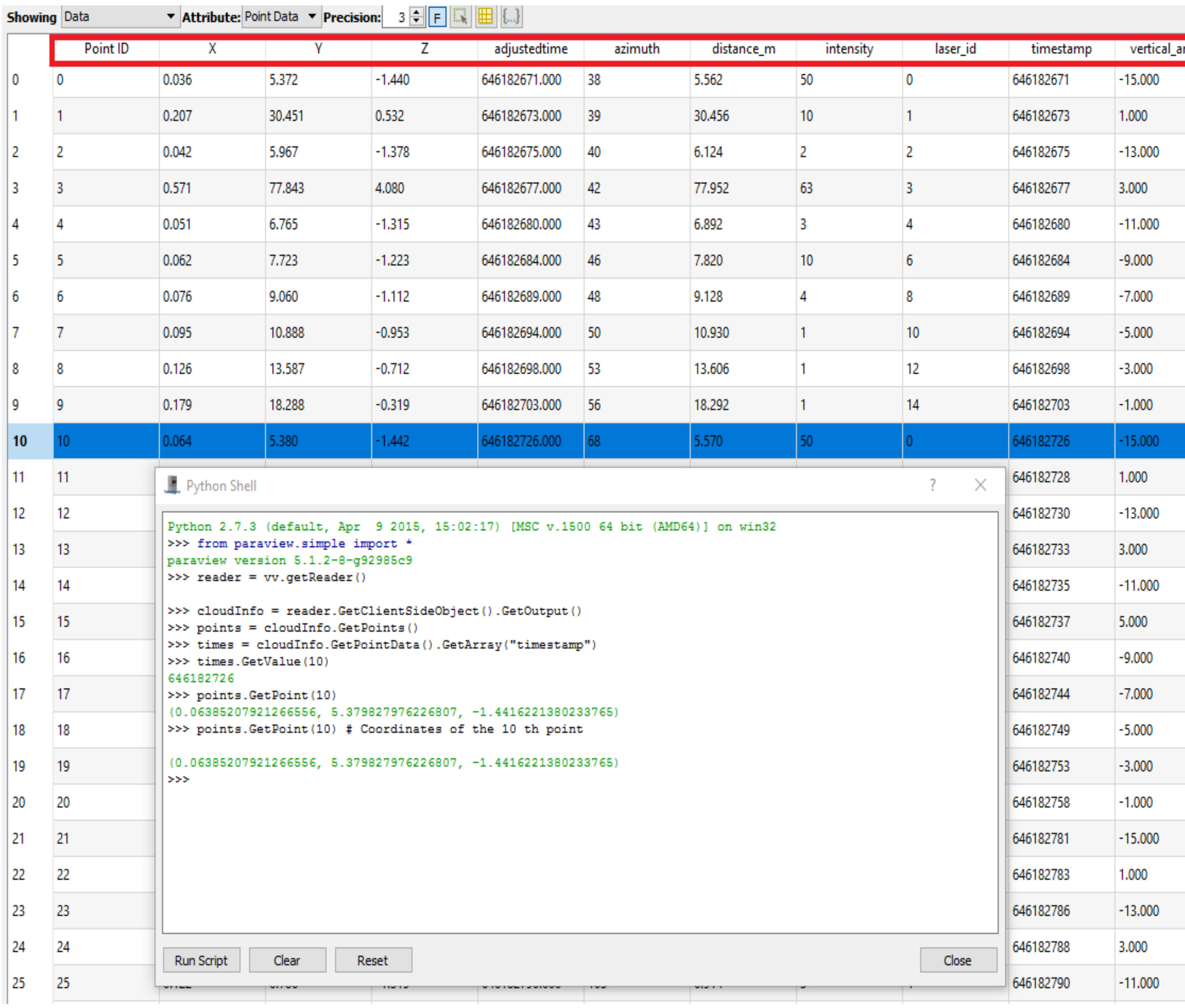
```
reader = vv.getReader() # Get the .pcap reader
cloudInfo = reader.GetClientSideObject().GetOutput() # Get the output of the reader -> dataset stored in a
vtkPolyData class
points = cloudInfo.GetPoints() # Get the 3D points list
times = cloudInfo.GetPointData().GetArray("timestamp") # Get the time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

Explanation : In this example we get the point cloud associated to the first frame of the “roundabout.pcap” data.

1- As you can see, all the function available in VeloView-python are stored in the module named “vv”. For instance, if you want to access to the function stored in the module vtk, you must write : vv.vtk.FunctionWanted.

2- getReader is the VeloView function which give you the .pcap reader. The .pcap reader is a vtk filter whose output is the dataset of the current frame. The outputted dataset is stored in a vtkPolyData class. It contains the 3D points list and the extra data associated to these points

3- The method GetPointData() of the vtkPolyData class give 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 23**). The arrays available depend on the data (Type of Velodyne sensor, if GPS is available or not, IMU available or not, extra data, …).



Showing Data Attribute: Point Data Precision: 3

	Point ID	X	Y	Z	adjustedtime	azimuth	distance_m	intensity	laser_id	timestamp	vertical_a
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 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)
>>>
```

Figure 3 : Python shell + Spreadsheet view

E-2- Select points using data

In this example we will select points with 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 :

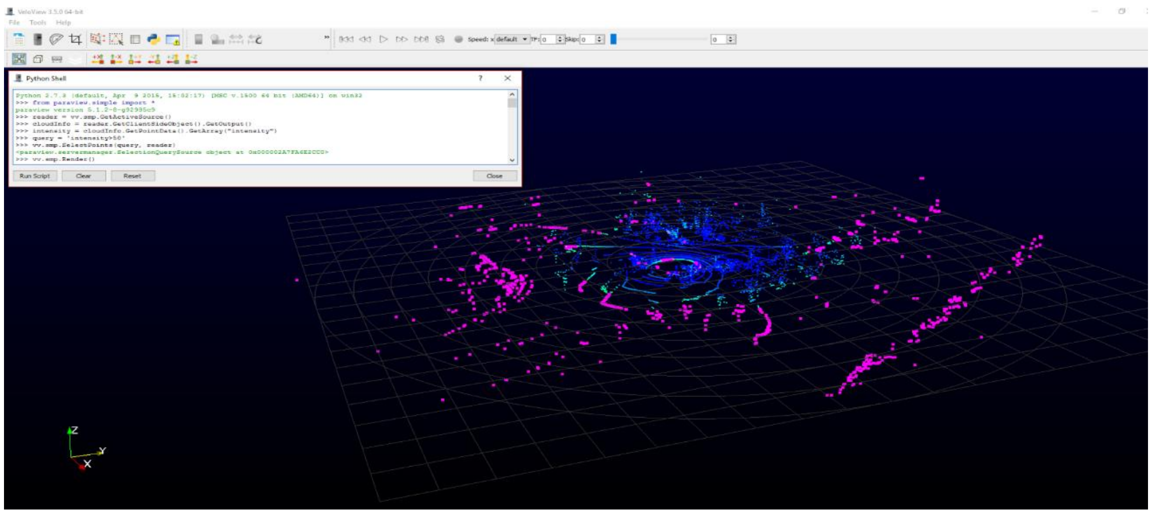
```
reader = vv.smp.GetActiveSource() # Get the .pcap reader
cloudInfo = reader.GetClientSideObject().GetOutput()
myintensity = cloudInfo.GetPointData().GetArray("intensity") # Get the intensity array
query = 'intensity>50' # query to select points having their intensity data array value over 50
vv.smp.SelectPoints(query, reader) # Select the points which satisfies the query
vv.smp.Render() # render the result
vv.saveCSVCurrentFrameSelection("D:\save.csv") # Save the selected points in .csv format
```

Explanation :

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

2- smp.Render() refresh the active source once the data process is done. If you want to refresh another view (spreadsheet view, overhead view, Main view) you have to set the active source with smp.SetActiveSource.

3- vv.saveCSVCurrentFrameSelection("D:\save.csv") saves the selected points in a .csv file located in the indicated filename.



```
Python 2.7.3 (default, Apr  9 2016, 16:02:17) [MSC v.1600 64 bit (AMD64)] on win32
>>> from paraview.simple import *
paraview version 5.1.2-8-g92985c9
>>> reader = vv.smp.GetActiveSource()
>>> cloudInfo = reader.GetClientSideObject().GetOutput()
>>> intensity = cloudInfo.GetPointData().GetArray("intensity")
>>> query = 'intensity>50'
>>> vv.smp.SelectPoints(query, reader)
<paraview.servermanager.SelectionQuerySource object at 0x000002A7FA6E2CC0>
>>> vv.smp.Render()
```

Figure 24 : Result of example #2. Pink points are the selected points

E-3- Manipulation of the GUI

In this example we add a new value available to the display speed button in the GUI.

Step 1 : Open the provided .pcap example: “roundabout.pcap”
Step 2 : Open the Python Shell
Step 3 : Write these python code lines :

```
speedButton = vv.app.PlaybackSpeed
speedButton.addItem("156")
speedButton.setCurrentIndex(speedButton.count - 1)
```

Figure 25 show the obtained result.

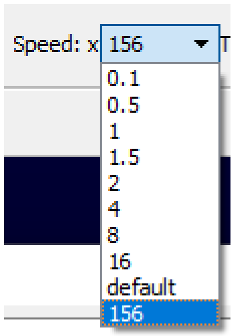


Figure 25 : New option available and setted : x156 speed

E-4- 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 :

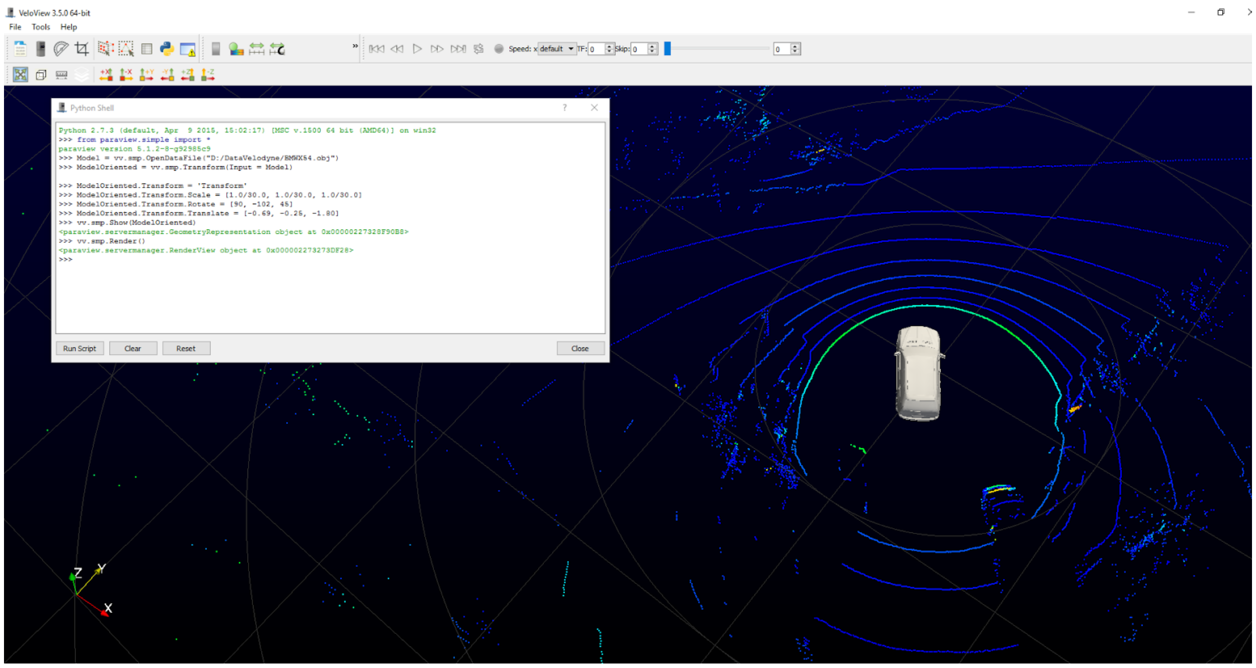
```
Model = vv.smp.OpenDataFile("D:/PythonManualData/BMWX54.obj")
ModelOriented = vv.smp.Transform(Input = Model)
ModelOriented.Transform = 'Transform'
ModelOriented.Transform.Scale = [1.0/30.0, 1.0/30.0, 1.0/30.0]
ModelOriented.Transform.Rotate = [90, -102, 45]
ModelOriented.Transform.Translate = [-0.69,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



```
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
>>> Model = vv.smp.OpenDataFile("D:/DataVelodyne/BMWX54.obj")
>>> ModelOriented = vv.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]
>>> vv.smp.Show(ModelOriented)
<paraview.servermanager.GeometryRepresentation object at 0x0000027328F90B8>
>>> vv.smp.Render()
<paraview.servermanager.RenderView object at 0x000002273273DF28>
>>>
```

Figure 26 : The model is added in the main view

E-5- 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.

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 must 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.

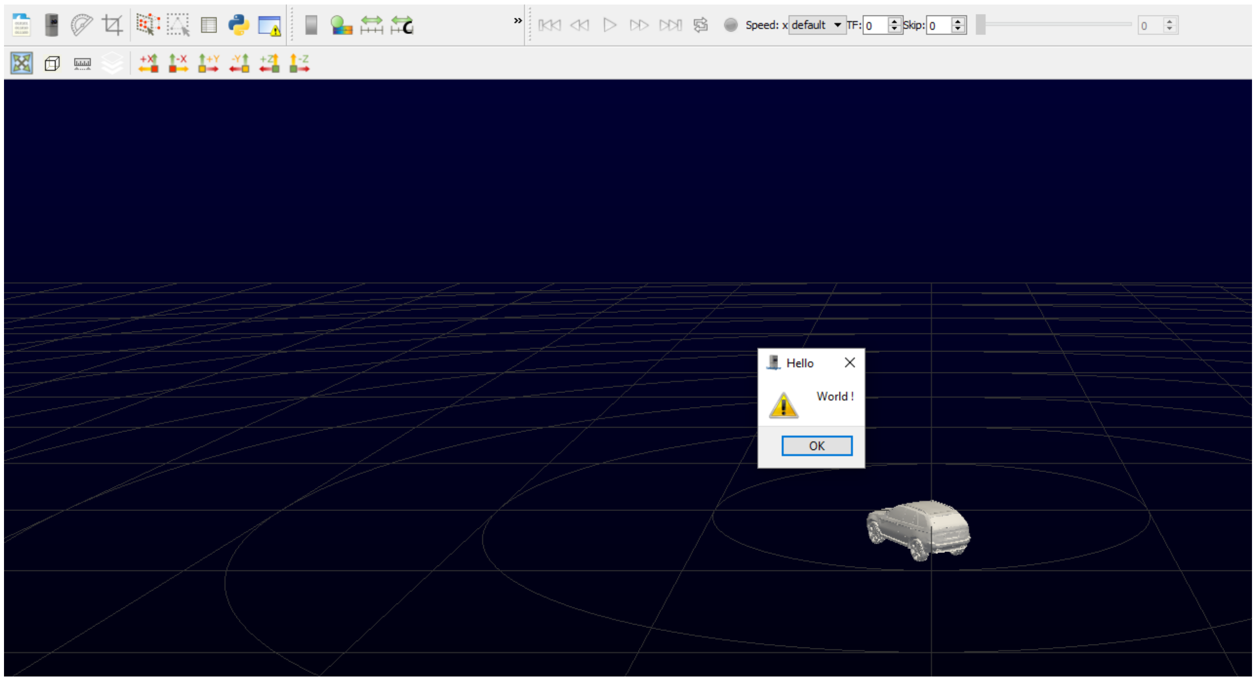


Figure 28 : The model is loaded thanks to python script launched with --script tool

F- List of functionalities

Here we will give a non-exhaustive list of all functionalities available in Python-VeloView

F-1- Simple Paraview : “vv.smp.”

All functionalities provided by the module paraview.simple are available using “vv.smp.”. You can find the description and documentation of the functionalities here : https://www.paraview.org/ParaView3/Doc/Nightly/www/py-doc/paraview.simple.html

F-2- Qt “vv.QtGui”

All functionalities provided by the module QtGui are available using “vv.QtGui.”. You can find the description and documentation of the functionalities here : http://pyqt.sourceforge.net/Docs/PyQt4/qtgui.html

F-3- Vtk Python

All functionalities provided by the module vtk are available using “vv.vtk.”. You can find the description and documentation of the functionalities here : http://www.vtk.org/Wiki/VTK/Examples/Python

F-4- VeloView functions “vv.”

All these functions are available on the module “vv.” :

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 filename

hideMeasurementGrid() : Hide the measurement grid

showMeasurementGrid(): Show the measurement grid

saveCSVCurrentFrame(filename) : export the current frame into a .csv file format

saveCSVCurrentFrameSelection(filename) : export the current selection of the current frame into a .csv file format

seekForward(): move to the next frame

seekBackward() : move to the previous frame

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

gotoNext() : go to the next frame

gotoPrevious : go to the previous frame

gotoEnd() : go to the last frame

gotoStart() : go to the first frame

updatePosition() : Refresh data of the current frame (RPM, GPS transformation, …)

getSensor() : get the live streaming reader. The output of this reader is the 100 last received pointcloud

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 farest returns

setFilterToIntensityHigh() : Show highest intensity returns

setFilterToIntensityLow() : Show lowest intensity returns

setTransformMode(mode) : change the transform mode 0 - raw, 1 - absolute, 2 - relative

showRPM() : show the rpm

Vv.app.reader : get the .pcap reader

vv.app.sensor : get the streaming reader

vv.app.distanceResolutionM : Distance resolution of the sensor

vv.app.trailingFramesSpinBox : GUI spinbox which determines the number of trailing frame

vv.app.geolocationToolBar : GUI geolocation toolbar

vv.app.overheadView : the overhead view (the one displaying the gps trajectory)

vv.app.mainView : the main view (the one displaying the point cloud)

vv.app.gridProperties : Properties of the grid

Vv.app.grid : The gris displayed in the main view

vv.app.targetFps : the target fps in playback mode

vv.app.scene : scene displayed in the main view

vv.app.PlaybackSpeed : GUI combo box which determines the playback speed

vv.app.pointsSkipSpinBox : GUI spin box which determines the number of points to skip

Velodyne LiDAR

If you have any problems with the existing software, please contact:

Michael Parkin mparkin@velodyne.com

If you would like new features or a specific version of VeloView which suits to your requirements, please contact:

Bastien Jacquet bastien.jacquet@kitware.com