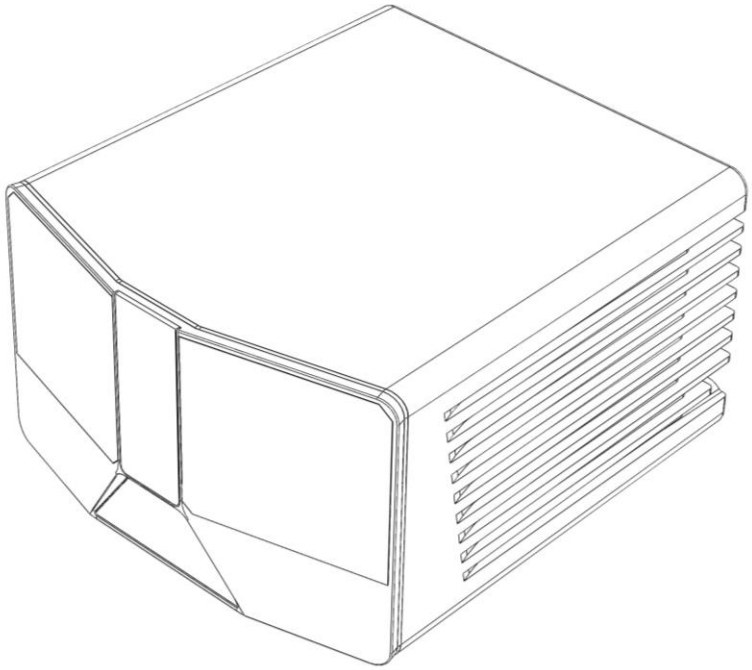
L3CAM USER MANUAL



PRODUCT MODEL	L3CAM
RELEASE	1.0

Disclaimer of Liability

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Beamagine products. To the maximum extent permitted by applicable law: (1) Materials are made available “AS IS” and with all faults, Beamagine hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Beamagine shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Beamagine had been advised of the possibility of the same. Beamagine assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. The products are subject to the terms and conditions of Beamagine’s limited warranty.

Beamagine products are not designed to provide safety functions nor to be integrated as a part of a safety system. You assume sole risk and liability for use of Beamagine products in such critical applications.

Limited Warranty

Beamagine warrants the L3CAM sensor to be free of defects in materials and workmanship for 12 months from the date of shipment. The user should not attempt to service the L3CAM sensor product beyond the description in this manual. Any attempt by the user to repair any Beamagine product under warranty will void the warranty. If the sensor fails during the warranty period, return the laser freight prepaid to Beamagine with a failure description made by the user. Beamagine will, at its option, repair or replace the defective item and return it freight prepaid to the user. Beamagine also offers repair services for products beyond the warranty. The damaged product is then investigated and evaluated at Beamagine resulting in a repair service report and quotation for the customer to consider. When you return products to Beamagine, please ask for the RMA (Return Material Authorization) number. Beamagine does not accept any returns without an RMA number, and/or if the products are mechanically damaged, scratched, burned or insufficiently and inappropriately packaged and especially if opened by unauthorized persons.

REVISION HISTORY

Date	Version	Revision
13/04/2022	1.0	Initial Release
16/05/2022	1.1	Protocol for UDP modified, the timestamp is now uint32_t

TABLE OF CONTENTS

LIST OF TABLES 6
LIST OF FIGURES 7
ACRONYMS 8
DEFINITIONS 9
REFERENCE DOCUMENTS 10
1 L3CAM DESCRIPTION 11
1.1 OVERVIEW 11
1.2 KEY FEATURES 12
1.2.1 Solid-state LIDAR 12
1.2.2 Imaging modes 13
1.2.3 Data fusion and embedded processing (optional features) 15
2 GETTING STARTED 18
2.1 SAFETY WARNINGS AND CAUTIONS 18
2.2 LIST OF MATERIALS 18
2.3 SYSTEM SETUP 18
2.3.1 System connection 18
2.3.2 Windows host configuration 19
2.3.3 Linux host configuration 22
3 API USER GUIDE 25
3.1 LIBRARY OVERVIEW 25
3.2 UDP PROTOCOL AND DATA DESCRIPTION 26
3.2.1 Point cloud 26
3.2.2 Image 28
3.3 RTSP PROTOCOL AND DATA DESCRIPTION 30
3.3.1 Point cloud 31
3.3.2 Image 31
3.4 API FUNCTIONS LIST 32
3.5 API FUNCTIONS DESCRIPTION 33
APPENDIXES 55
APPENDIX 1 READING A POINT CLOUD 55
APPENDIX 2 READING AN IMAGE 56
APPENDIX 3 POLARIMETRIC PIXEL FORMAT 58

APPENDIX 4 RETURN CODES 59

APPENDIX 5 STATUS 60

APPENDIX 6 DATA STRUCTURES 61

APPENDIX 7 L3CAM DIMENSIONS 64

LIST OF TABLES

Table 1. Acronyms....8
Table 2. Definitions....9
Table 3. Reference documents....10
Table 4. libL3Cam ports....26
Table 5. Point cloud frame header....27
Table 6. Data packet structure....27
Table 7. Point structure....27
Table 8. Point cloud frame footer....28
Table 9. Image frame header....29
Table 10. Perception packet....30
Table 11. Point cloud frame footer....30
Table 12. GStreamer pipeline....30
Table 13. API functions....32
Table 14. Library return codes....59
Table 15. Polarimetric camera errors....59
Table 16. Status bits....60
Table 17. sensorTypes enum fields....61
Table 18. laserClass enum fields....61
Table 19. pointCloudColor enum fields....62
Table 20. streamingProtocols enum fields....62
Table 21. sensor struct fields....62
Table 22. l3cam struct fields....63

LIST OF FIGURES

Figure 1. Components and functions of L3CAM. 11
Figure 2. General scheme of the pulsed solid-state MEMS-based L3CAM LIDAR. 12
Figure 3. MEMS raster scan and its associated FOV. 12
Figure 4. Multiple back-reflected hits example whilst scanning a bulky object. 13
Figure 5. Bayer filter mounted on an imaging sensor. 14
Figure 6. Polarimetric imaging examples. 14
Figure 7. Scheme of the microscopic filter structure of the Sony polarsens sensors. 15
Figure 8. Examples of outdoor LWIR thermal images. 15
Figure 9. Multimodal fusion between the L3CAM LIDAR and different imaging modes. 16
Figure 10. Enhanced AI pedestrian perception of an outdoor scene thanks to data fusion. 17
Figure 11. System connection. 19
Figure 12. L3CAM side view. 19
Figure 13. Windows network connections. 19
Figure 14. Access network properties in Windows. 20
Figure 15. Access TCP/IPV4 settings in Windows. 20
Figure 16. Windows TCP/IPV4 settings. 21
Figure 17. Access advanced Eth properties in Windows. 21
Figure 18. Modify jumbo packet size in Windows. 22
Figure 19. Change receive buffers settings in Windows. 22
Figure 20. Access to network configuration in Ubuntu. 23
Figure 21. Configure IP address in Ubuntu. 23
Figure 22. Set jumbo packet size in Ubuntu. 24
Figure 23. Activate the network interface. 24
Figure 24. Configuration of the receiving buffers size. 24
Figure 25. Categories of the functions of the libL3Cam API. 25
Figure 26. L3CAM initialization sequence. 25
Figure 27. Operation sequence. 25
Figure 28. Transmission sequence of point cloud data. 26
Figure 29. Point cloud reference coordinate system. 28
Figure 30. Colour scale for distance (a) and intensity (b). 28
Figure 31. Transmission sequence of image data with results from perception functions. 29
Figure 32. Point cloud and cropping area. 31
Figure 33. Polarimetric Bayer image. See image format in APPENDIX 3. 32
Figure 34. Colour filter array of Lucid PHX050S1. 58
Figure 35. L3CAM front side dimensions. 64
Figure 36, L3CAM mount holes dimensions. 65
Figure 37. L3CAM connector cable dimensions. 66

ACRONYMS

Table 1. Acronyms

ACRONYM	DESCRIPTION
AC/DC	**A**lternating **C**urrent / **D**irect **C**urrent
AI	**A**rtificial **I**ntelligence
API	**A**pplication **P**rogramming **I**nterface
CCD	**C**harge-**C**oupled **D**evice
CMOS	**C**omplementary **M**etal-**O**xide **S**emiconductor
DMD	**D**igital **M**icromirror **D**evice
DOLP	**D**egree **O**f **L**inear **P**olarization
FOV	**F**ield **O**f **V**iew
IP	**I**nternet **P**rotocol
IPV4	**I**nternet **P**rotocol **V**ersion **4**
IR	**I**nfra**R**ed
LIDAR	**LI**ght **D**etection **A**nd **R**anging
LWIR	**L**ong-**W**ave **I**nfrared
MEMS	**M**icro **E**lectro **M**echanical **S**ystem / **S**ensor
MTU	**M**aximum **T**ransmission **U**nit
NIC	**N**etwork **I**nterface **C**ard
NIR	**N**ear **I**nfra**R**ed
RGB	**R**ed-**G**reen-**B**lue
RTSP	**R**eal **T**ime **S**treaming **P**rotocol
SNR	**S**ignal-to-**N**oise **R**atio
SW	**S**oft**W**are
TCP	**T**ransmission **C**ontrol **P**rotocol
TOF	**T**ime **O**f **F**light
TOT	**T**ime **O**ver **T**hreshold
UDP	**U**ser **D**atagram **P**rotocol
VIS	**VIS**ible

DEFINITIONS

Table 2. Definitions

TERM	DESCRIPTION
Data fusion	Data fusion is the process of integrating multiple data sources to produce more consistent, accurate, and useful information than that provided by any individual data source.
DOLP	Measure of the amount of linearly polarized light. The larger the DOLP, the clearer the linear polarization state.
Hit	A hit is a point at which the laser beam pulse reaches an object that reflects it to the LIDAR.
Perception	Machine perception is the capability of a system to interpret data similarly to humans.
TOF	Elapsed time between light events. In the case of L3CAM, TOF is the elapsed time a pulse travels from the laser source, backscatters on a target and comes back to the detector. Thus, it is a direct measure of the distance to the target.
TOT	The elapsed time that the voltage signal caused by an incoming pulse on the photodetector is above a threshold level. Thus, it is a measure of the total energy received since the larger the TOT, the higher the intensity.

REFERENCE DOCUMENTS

Table 3. Reference documents

	DOCUMENT	TITLE	VERSION	DATE
RD 1	L3CAM datasheet	L3CAM Multimode imaging LIDAR datasheet	1.0	06/04/2022
RD 2	GenICam standards	European machine vision association web page.	NA	

1 L3CAM DESCRIPTION

1.1 OVERVIEW

The L3CAM is a multimodal sensor composed essentially of a solid-state LIDAR sensor and up to 3 additional imaging modes (RGB, thermal and/or polarimetric).

The LIDAR system is a patented MEMS-based scanning technology that combines high-resolution 3D imaging, real-time frame rate, and long-range. The most suitable combination for applications related to autonomous vehicles, security, object detection, situational awareness and mapping.

Critical applications, however, require more than a single “eye” to achieve high-reliability levels once the data is processed. This is where the additional imaging modes play a significant role.

The data from the different sensors can be integrated through data fusion algorithms to provide more consistent, accurate and useful information about the environment.

On top of the processing chain, at the higher abstraction level, L3CAM integrates state-of-the-art perception algorithms to perform objects recognition.

L3CAM offers the all in a single, compact and cost-effective device.

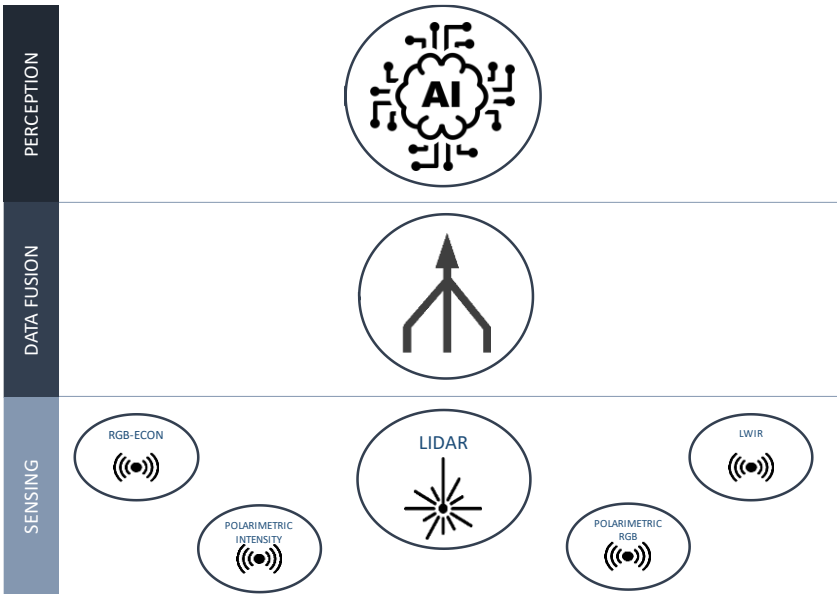


Figure 1. Components and functions of L3CAM.

1.2 KEY FEATURES

1.2.1 Solid-state LIDAR

The L3CAM LIDAR sensor is a solid-state device without mechanical moving parts for scanning the environment or large embodiments. Its optomechanical design is based on a MEMS scanner which steers a pulsed IR fibre laser beam towards targets within the FOV of the system and a receptor that gathers the backscattered light to resolve the distance to them from its TOF – time the pulse is travelling from the LIDAR source to the object and backscatters to the detector. This subdivision of the sensor can be appreciated in the following figure.

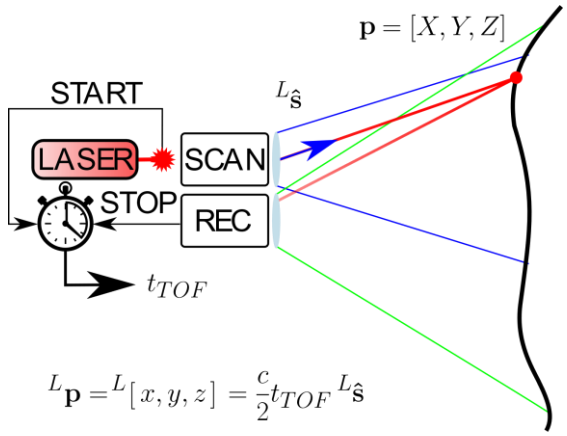


Figure 2. General scheme of the pulsed solid-state MEMS-based L3CAM LIDAR.

Using a MEMS scanner enables reaching high output frame rates (~10 Hz) as well as high-resolution Point Clouds with a large point density. The L3CAM LIDAR offers a rectangular FOV, similar to conventional cameras, by using a raster scan that provides great precision, accuracy and angular resolution in both horizontal and vertical scanning directions (<0.1º). The FOV is cropped to avoid non-linearities in the changes of lines.

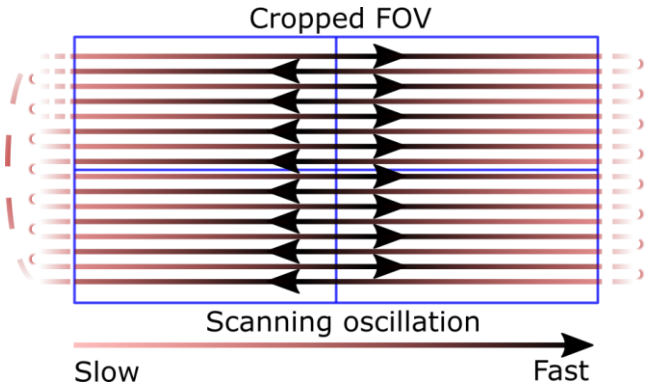


Figure 3. MEMS raster scan and its associated FOV.

Together with the solid-state design, the L3CAM LIDAR sensor offers a patented solar background suppression that enhances its detectivity, thus enlarging its measurement range, by improving the SNR.

It does not only become L3CAM LIDAR robust to sunlight but also self-immune to other devices situated simultaneously together in the same environment. Hence, it is possible to use an array of L3CAM devices for covering a wider FOV without the risk of cross-talking in the union.

Moreover, the L3CAM LIDAR sensor is capable of gathering up to 4 different back-reflected pulses in a single scanning direction because the laser beam slightly broadens as long as it propagates through space due to its divergence (~0.1º). Consequently, a portion of the total emitted energy might back-reflect from either a tiny object or at the contour of a bulky one whereas the remaining one might keep travelling towards further targets producing multiple reflections with different back-reflected intensities measured with the TOT of the measured peak, as Figure 4 demonstrates below.

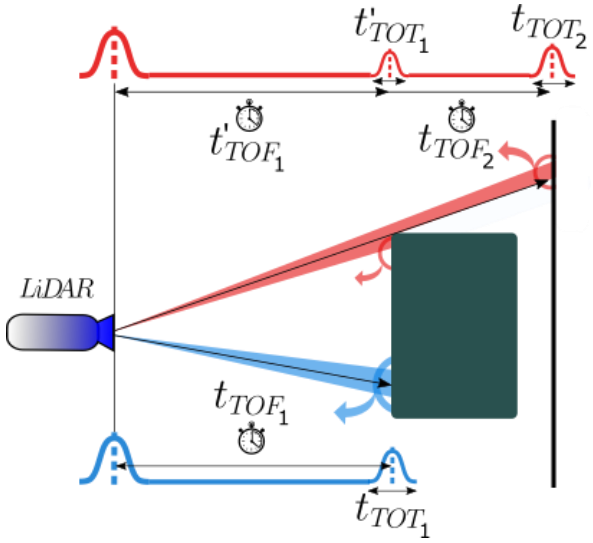


Figure 4. Multiple back-reflected hits example whilst scanning a bulky object.

To summarize, the L3CAM LIDAR sensor offers a high-end imaging performance thanks to the high resolution provided by the MEMS scanner and the enhanced range detectivity by the patented background suppression, resulting in dense output Point Clouds with up to 4 hits.

1.2.2 Imaging modes

Apart from the LIDAR sensor, the L3CAM device provides additional imaging modes that are complementary between them. In such a way, the L3CAM device offers information from different natures and working principles which are referred to as imaging modes. Obtaining information from different imaging modes and sensors is crucial for improving the overall performance and reliability of L3CAM.

Up to date, L3CAM versions can be mounted with up to 3 complementary passive imaging modes:

1.2.2.1 RGB or colour

The conventional colour camera senses the colour information of the environment using a CCD or a CMOS sensor with a Bayer filter (Figure 5). This imaging mode senses the amount of light, thus intensity, in the VIS domain (400 – 700 nm) of the spectrum.

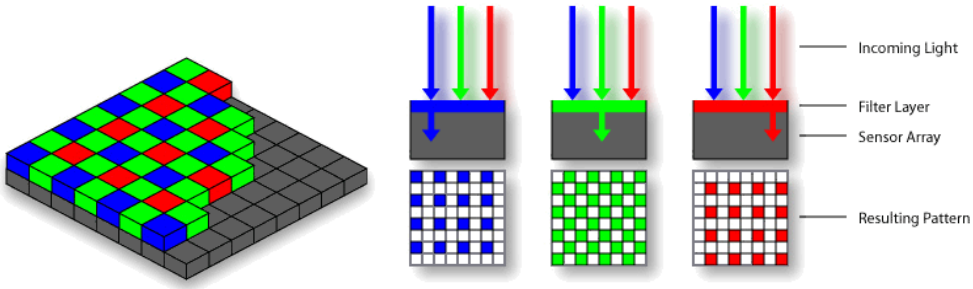


Figure 5. Bayer filter mounted on an imaging sensor.
Image from https://commons.wikimedia.org/wiki/File:BayerPatternFiltration.png

The high-resolution colour images provided by this imaging mode mimic human eye perception, hence the importance of this mode for AI perception and assisted computer visual applications.

1.2.2.2 Polarimetric mono

Polarization is the vectorial property of light related to the geometrical orientation of the transverse wave's oscillations, which is perpendicular to its direction of motion. Light can be unpolarized (radiation from many light sources like the sun) or either linearly, circularly or elliptically polarized. Polarizers are materials that let a certain polarization state pass through, thus they act as filters. Hence, polarimetric cameras sense the polarization state of light using combinations of orthogonal polarizers. In particular, the L3CAM's polarimetric mono camera uses an array of micro-polarizers set as a Bayer filter for sensing four different polarization states in the VIS and slightly in the NIR (>780 nm) domain at once, as Figure 6 presents.

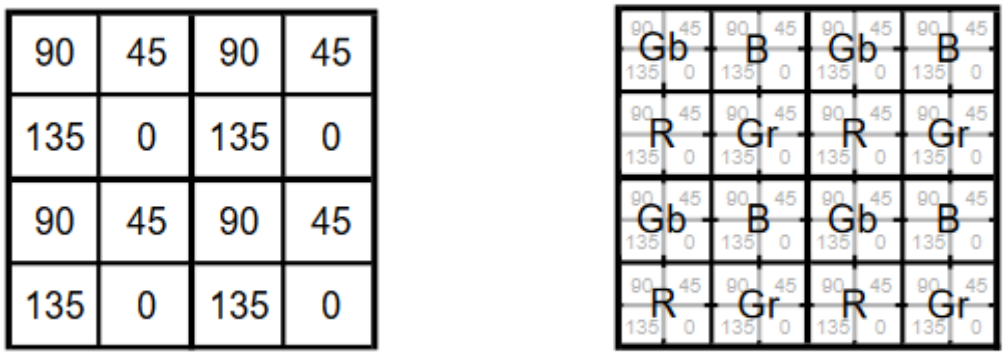
(**a**) (**b**) (**c**)

Figure 6. Polarimetric imaging examples.
*(**a**) Microscopic structure of the Sony IMX264MZR polarsens sensor. (**b**) Conventional colour image.*
*(**c**) Polarimetric image representing the Degree Of Linear Polarization (DOLP) that enhances contrast.*
Images from https://www.sony-semicon.co.jp/e/products/IS/industry/technology/polarization.html.

Polarization is imperceptible for human vision but many animals are sensitive to this light's property because it helps to distinguish between materials. Usually, metals and some artificial materials that keep polarization or polarize in a certain state, clearly differentiate from natural objects that depolarize light. Thus, this imaging mode provides additional contrast of the environment in high resolution as well.

1.2.2.3 Polarimetric colour

Whereas the previous imaging mode senses the total amount of light with a certain polarization state in the whole VIS domain, polarimetric colour cameras are capable of also splitting such information in colours. They combine the micro-polarizers array with a Bayer filter for obtaining the polarization state at the three-primary colour RGB wavelengths as Figure 7 depicts below.



(**a**) (**b**)

*Figure 7. Scheme of the microscopic filter structure of the Sony polarsens sensors. (**a**) Polarimetric mono IMX264MZR. (**b**) Polarimetric colour IMX264MYR.*

In line with the above imaging mode, this one not only provides polarization information to the overall perception but also colour information at once from a unique sensor.

1.2.2.4 LWIR or thermal

Contrary to all the previous imaging modes that work in the VIS and NIR domain, LWIR imaging mode is sensitive to the long-wave IR wavelengths (7 – 14 µm). Consequently, this imaging mode provides thermal information about the environment. Temperature, besides being complementary, offers robustness to lighting conditions and to challenging environments like fog, smoke, rain and dust for example. Firstly, temperature variations in the LWIR range are extremely slow compared to varying illumination conditions and it does not require external illumination since objects always radiate in the LWIR domain. Secondly, scattering media responsible for reducing visibility interacts differently with the LWIR wavelengths than with the VIS and NIR ones due to particle size.

Figure 8. Examples of outdoor LWIR thermal images.

Hence, this imaging mode is robust to external conditions, it does not even need external illumination, and to hazardous conditions where VIS and NIR sensors might fail to work properly.

1.2.3 Data fusion and embedded processing (optional features)

The L3CAM device optionally offers data from all its available sensors precisely and accurately fused, what is called congruent data fusion, according to the acquired product version. This is thanks to an industry-leading multisensory intrinsic calibration and the mechanical robustness and embeddedness of

the device. This novel multimodal calibration process yields low parallax error that outperforms the state-of-the-art data fusion devices for all the available imaging modes.

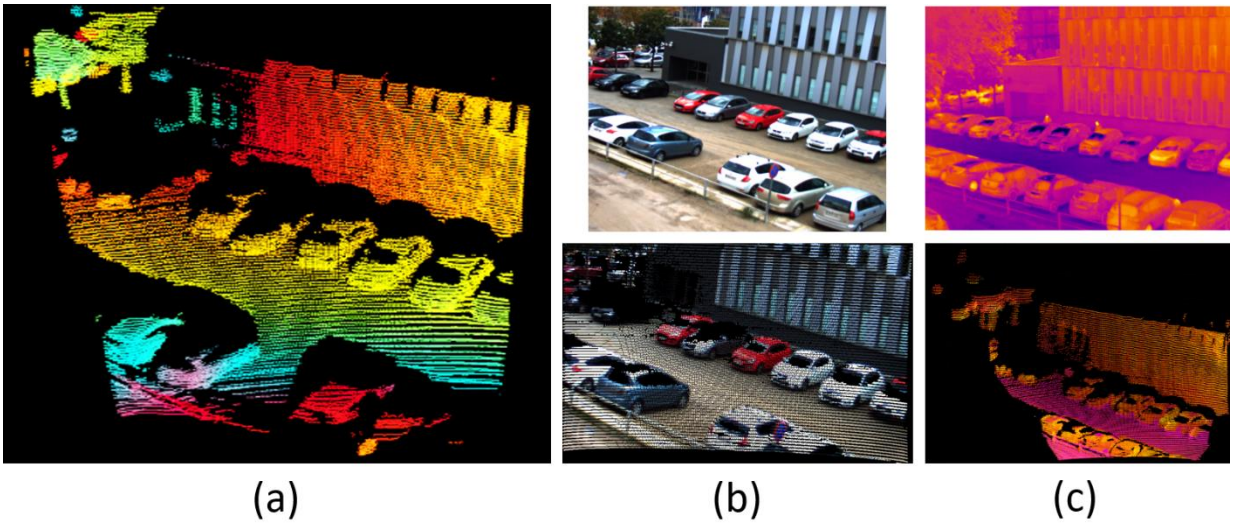


*Figure 9. Multimodal fusion between the L3CAM LIDAR and different imaging modes. (**a**) Original Point Cloud coloured with distance. (**b**) Colour and (**c**) Thermal image with the corresponding fused Point Cloud.*

Thanks to its powerful embedded processing unit, the L3CAM device is capable of offering this congruent data fusion in real-time although it can also be computed offline. This unit synchronously gathers data from all sensors with 10 ns high-resolution timestamps and short time discrepancy. Moreover, depending on the product version, the L3CAM software incorporates advanced and accelerated AI perception algorithms for real-time object detection and tracking that benefit from the congruent data fusion for improving its overall performance and reducing the rate of false alarms.

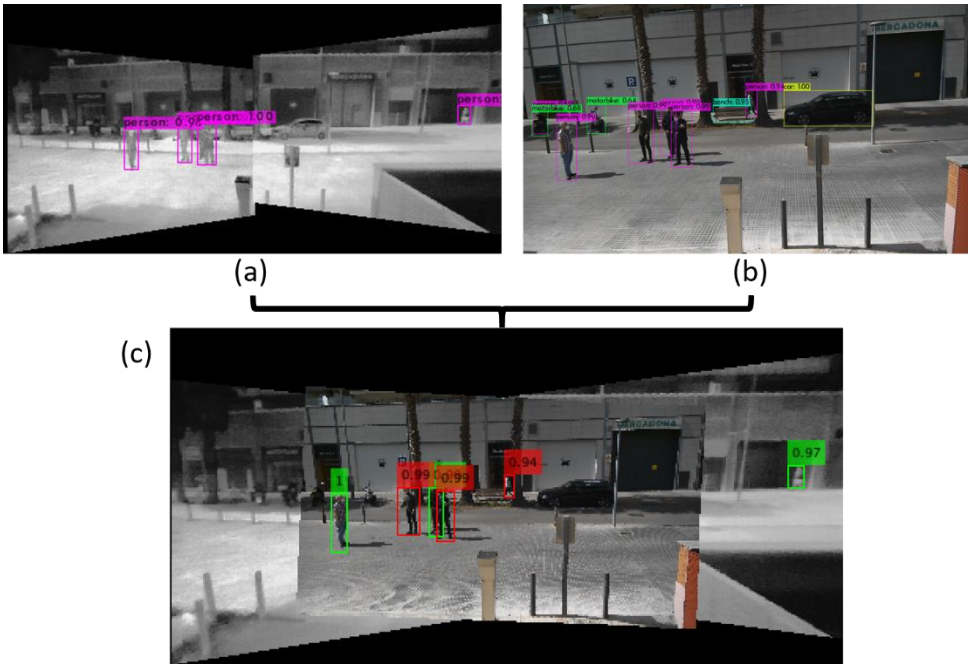
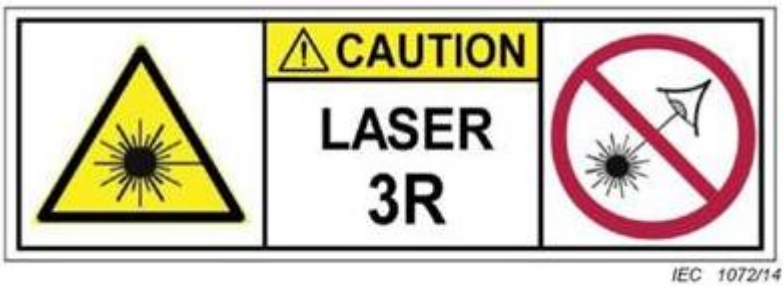


*Figure 10. Enhanced AI pedestrian perception of an outdoor scene thanks to data fusion. (**a**) Perception on a panoramic thermal image. (**b**) Perception on a colour image. (**c**) Perception on a colour + thermal image where the colour of the bounding boxes determines which sensor provides greater detection confidence: (green) thermal detections and (red) colour detections.*

2 GETTING STARTED

2.1 SAFETY WARNINGS AND CAUTIONS

The L3CAM product fulfills the class 3R classification according to the requirements of IEC 60825-1: 2014 / 2007 (Safety of Laser Products).



WARNING:

"**DANGER** - Laser Radiation - Do Not Stare Into Beam or View Directly With Optical Instruments, for example, eye loupes, magnifiers, microscopes or binoculars".

The light source from this sensor emits invisible laser radiation. Avoid direct exposure to the laser light source.

2.2 LIST OF MATERIALS

The following items are delivered with the purchase of L3CAM:

- L3CAM multimodal sensor.
- Connection cable.
- AC/DC converter.
- SW integration library.
- User guide.

2.3 SYSTEM SETUP

Hereinafter we describe the steps required to set up L3CAM, connect it to the system host and configure the network.

2.3.1 System connection

The L3CAM cable to connect the multimodal sensor to the host has a custom connector on the L3CAM side and an RJ45 connector and a power connector on the host side.

The connection of the system is described in the following figures.

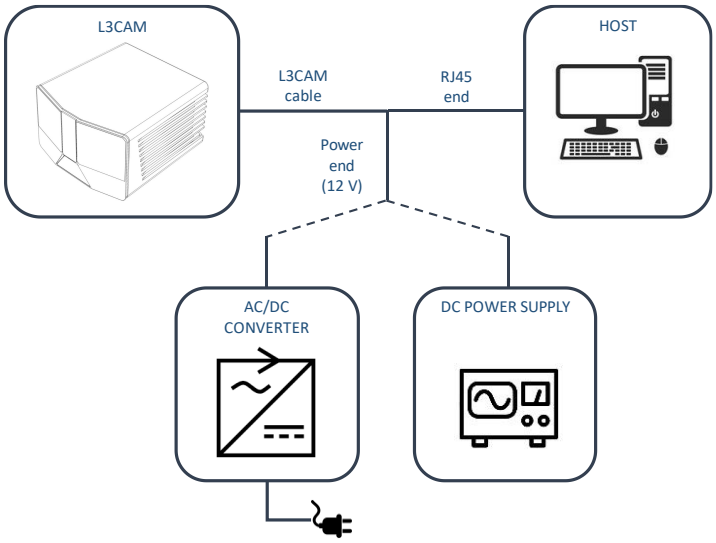


Figure 11. System connection.

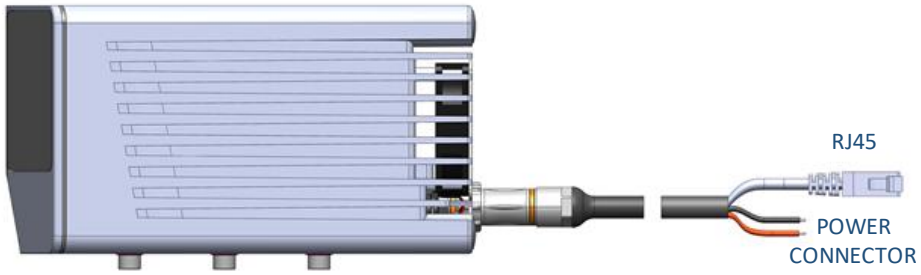


Figure 12. L3CAM side view.

2.3.2 Windows host configuration

To configure the IP address of the device, go to control panel > Network and internet > Network connections:

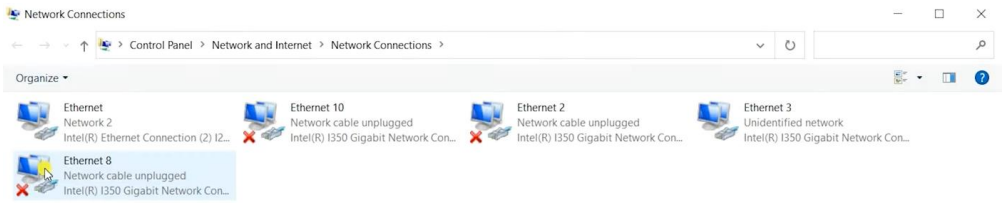


Figure 13. Windows network connections.

Select the NIC where the L3CAM is connected, right-click and open the properties window:

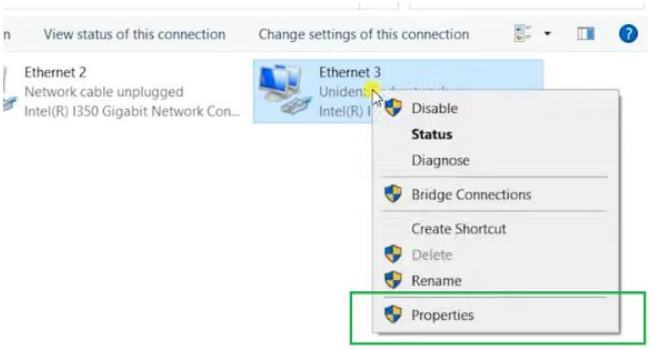


Figure 14. Access network properties in Windows.

Then go to TCP/IPV4 settings:

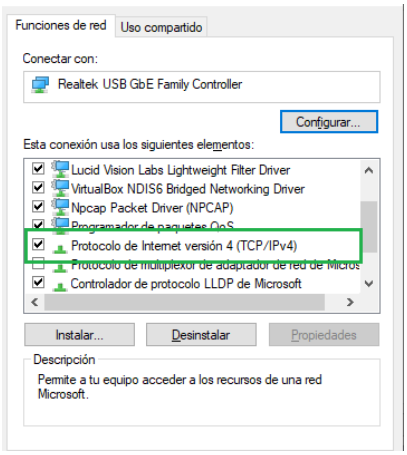


Figure 15. Access TCP/IPV4 settings in Windows.

Select the manual configuration and type the desired address, the netmask and the gateway (optional).

The host IP address has to be in the range 192.168.5.XXX because the L3CAM is configured with netmask size 24. In this example, the configured IP address is 192.168.5.21 and netmask 255.255.255.0. The gateway is left empty.

Click accept to save the changes, this will automatically close the window.

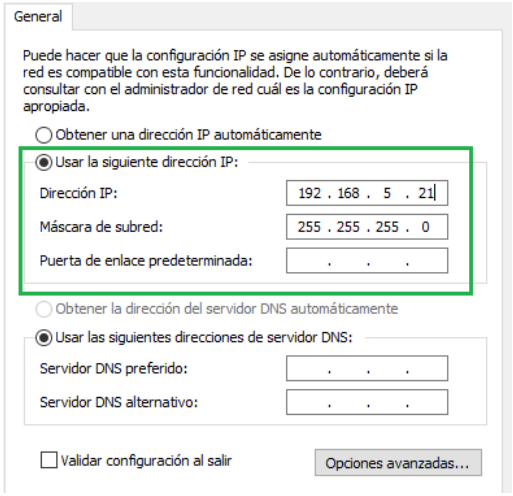


Figure 16. Windows TCP/IPV4 settings.

To enable jumbo frames on the NIC, click on the configure button of the Ethernet interface properties to open the advanced settings.

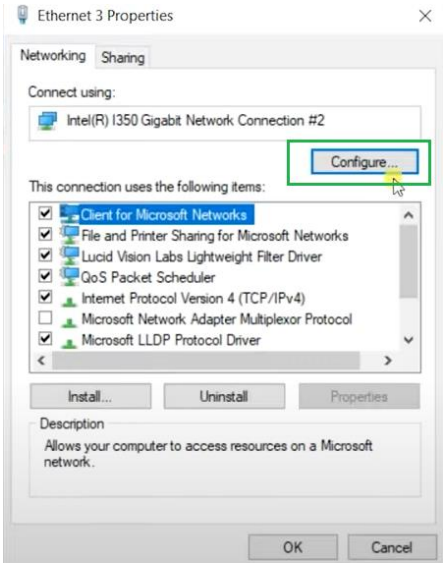


Figure 17. Access advanced Eth properties in Windows.

Then go to the advanced tab, and search the jumbo packet option; modify the value to 9014 bytes:

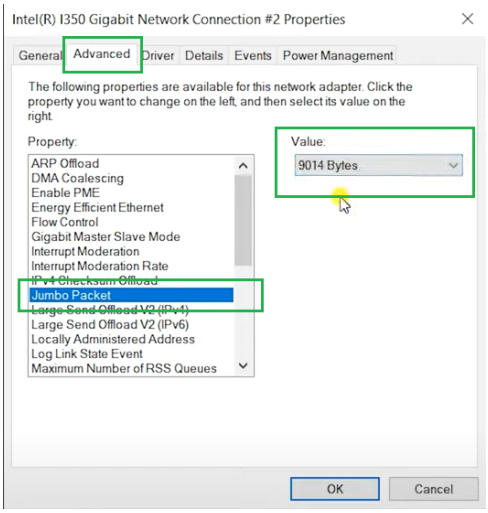


Figure 18. Modify jumbo packet size in Windows.

Finally, in the same window, search the receive buffers property and change the value to 2048:

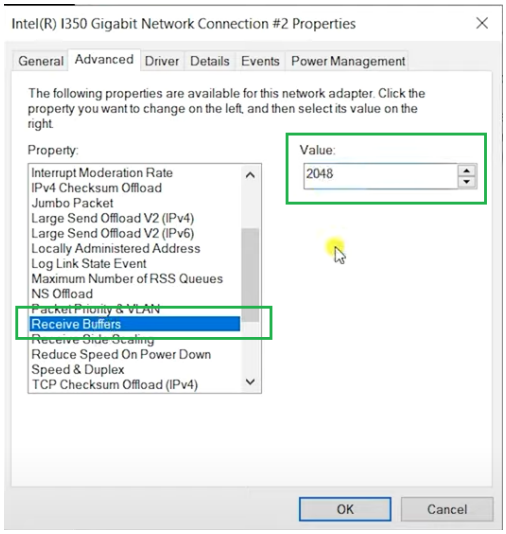


Figure 19. Change receive buffers settings in Windows.

When all the above steps are done, the host will be ready to receive the data streams of the L3CAM.

2.3.3 Linux host configuration

To configure the Ethernet IP address of the host device, go to system configuration > network. From the list select the NIC to be configured and click on Options… this will open the NIC options window.

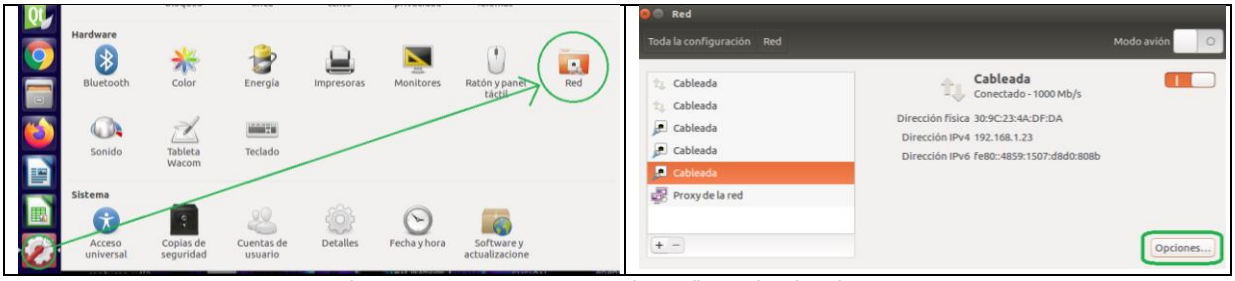


Figure 20. Access to network configuration in Ubuntu.

In the NIC options window go to IPv4 settings, in Method select Manual, then click on the button Add and type the desired IP address, mask and gateway (optional), then click on Save.

The host IP address has to be in the range 192.168.5.XXX because the L3CAM is configured with netmask size 24. In this example the configured IP address is 192.168.5.20.

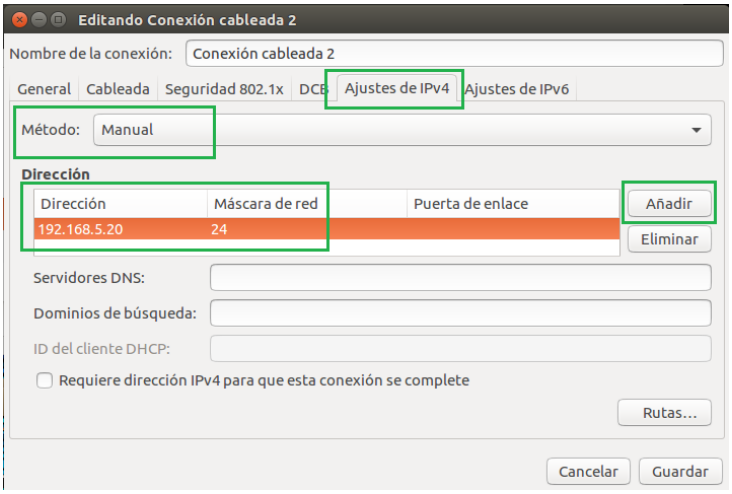


Figure 21. Configure IP address in Ubuntu.

In the current window go to the Ethernet tab and change the MTU value to 9000, this will allow the host device to handle jumbo frames.

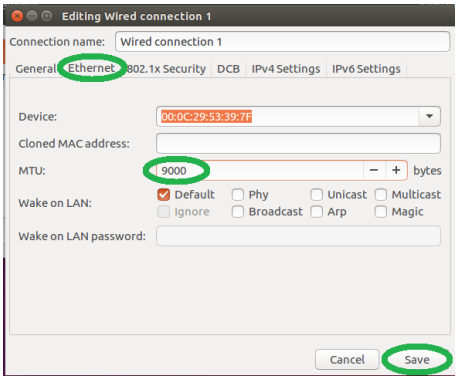


Figure 22. Set jumbo packet size in Ubuntu.

Finally, click the Save button and then restart the NIC configuration with the on/off switch button as shown in the next images.

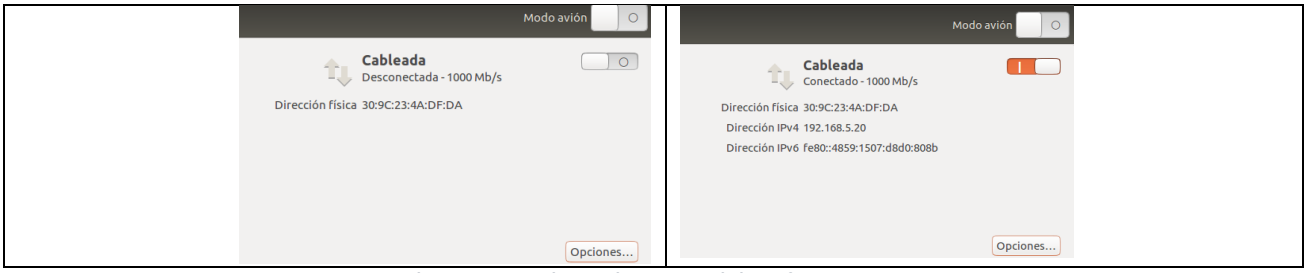


Figure 23. Activate the network interface.

The last step is to configure the receiving buffers size, to do so open a console with ctrl + alt + t, and type the following commands:

```
sudo sh -c "echo 'net.core.rmem_default=268435456' >> /etc/sysctl.conf"
sudo sh -c "echo 'net.core.rmem_max=268435456' >> /etc/sysctl.conf"
sudo sysctl -p
```

```
beamagine@beamagine-l3cam-dev:~$ sudo sh -c "echo 'net.core.rmem_default=268435456' >> /etc/sysctl.conf"
[sudo] contraseña para beamagine:
beamagine@beamagine-l3cam-dev:~$ sudo sh -c "echo 'net.core.rmem_max=268435456' >> /etc/sysctl.conf"
beamagine@beamagine-l3cam-dev:~$ sudo sysctl -p
net.core.rmem_default = 268435456
net.core.rmem_max = 268435456
beamagine@beamagine-l3cam-dev:~$
```

Figure 24. Configuration of the receiving buffers size.

When all the above steps are done, the host will be ready to receive the data streams of L3CAM.

3 API USER GUIDE

3.1 LIBRARY OVERVIEW

The library for the integration of L3CAM in the system host (libL3Cam) has been designed to encapsulate all the complexities of the system, automating many functions to simplify the use of the system. The interface exposed by the API has been carefully designed to facilitate its use.

Before proceeding with the details of the API let's establish the following set of categories of the functions:

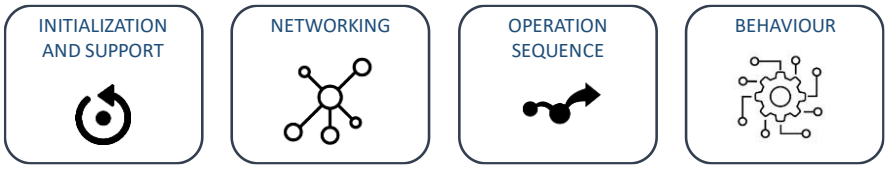


Figure 25. Categories of the functions of the libL3Cam API.

This set of functions is used to initialize and operate L3CAM.

The initialization sequence is described in the following figure.

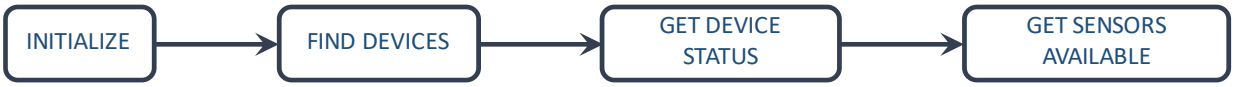


Figure 26. L3CAM initialization sequence.

The operation sequence is as follows:

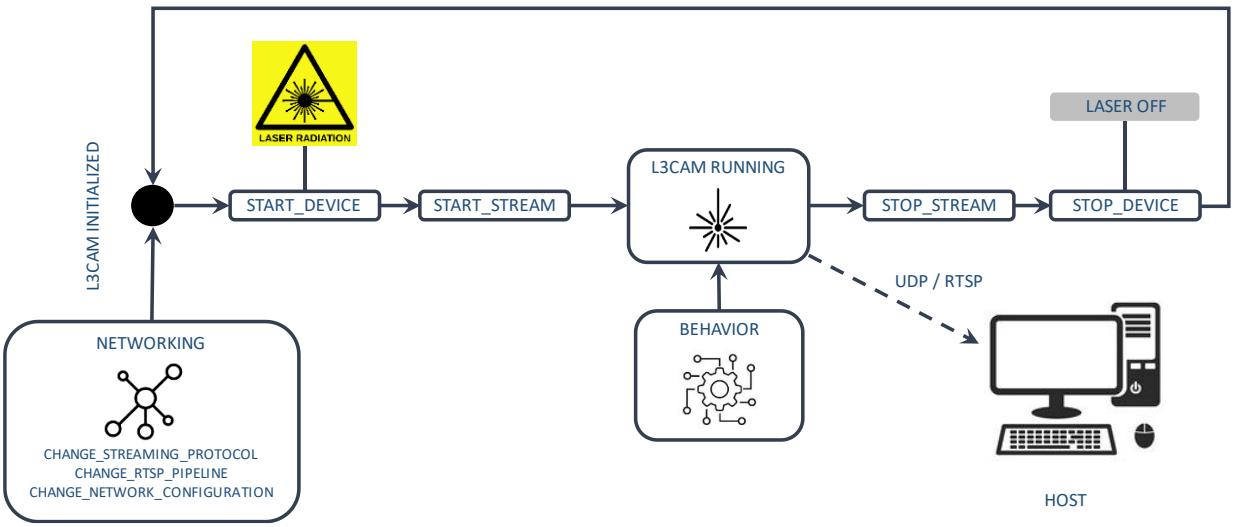


Figure 27. Operation sequence.

After starting the device, we ask it to start streaming to have it fully operational and sending UDP/RTSP packets to the host computer as will be described in the section below. In this state, we can use the functions under the behaviour category to change its comportment.

LibL3Cam uses the following ports for communications

Table 4. libL3Cam ports.

PROTOCOL	PORT	CONFIGURATION
TCP	6000	Fixed
UDP	6050 (LIDAR)	Fixed
UDP	6060 (Polarimetric)	Fixed
RTSP	5040 (LIDAR)	Reconfigurable
RTSP	5030 (Polarimetric)	Reconfigurable

TCP is used internally by lib3Cam and is transparent to the user. However, the system host must have the required port available.

3.2 UDP PROTOCOL AND DATA DESCRIPTION

In this mode, L3CAM sends non-compressed LIDAR and multi-modal sensor data with the UDP protocol. Since each sensor uses a different UDP port as described in Table 4, it is recommended to implement a different thread for each sensor to properly receive the data.

To optimize networking resources it is required to enable jumbo frames as described in section 2.3.

3.2.1 Point cloud

The transmission sequence of a point cloud frame is described in Figure 28.

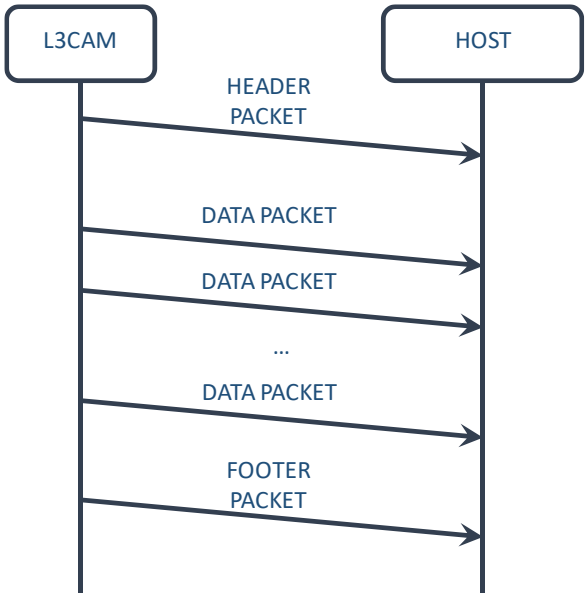


Figure 28. Transmission sequence of point cloud data.

The transmission of the frame starts with a header packet, is followed by a sequence of data packets and is finalized with a footer packet.

The header packet is as follows:

Table 5. Point cloud frame header.

FIELD	STX	Total number of points	Ctrl value 1	Ctrl value 2	Timestamp
SIZE (Bytes)	1	4 (int32_t)	4 (int32_t)	4 (int32_t)	4 (uint32_t)
Value	0x02				

where:

- STX is a header packet identifier with value 0x02.
- Total number of points: the number of points that will be sent in the sequence of data packets.
- ctrl value 1: reserved.
- ctrl value 2: reserved.
- Timestamp: an integer that represents the time when the frame was received. The time is in the L3CAM Jetson TX2 time and contains hhmmsszzz. For example, a frame received at 18:32:12:346 will return the timestamp value 183212346;

The data packets are organized as described in the following table:

Table 6. Data packet structure.

FIELD	Number of points in packet	Point 1	Point 2	…	Point N
SIZE (Bytes)	4 (int32_t)	20 (point)	20 (point)	20 (point)	20 (point)

Each data packet contains 400 points (packet size = 8000 + 4 bytes) but the last one, which can contain fewer points depending on the configuration of the LIDAR resolution.

Each point has the following structure:

Table 7. Point structure.

FIELD	X	Y	Z	Intensity	RGB
SIZE (Bytes)	4 (int32_t)	4 (int32_t)	4 (int32_t)	4 (int32_t)	4 (int32_t)

where:

- X, Y, Z are the coordinates of the hit[1] in the coordinate system described in Figure 29;
- Intensity[2];
- RGB is a field that can represent different features depending on the configuration:

[1] See description in the definitions section.
[2] Intensity is the same as the TOT value described in 1.2.

- colour scale[3] for the distance of the hit (*d*),
- colour scale for the intensity of the hit,
- RBG value of the sensor with which the point cloud has been fused.

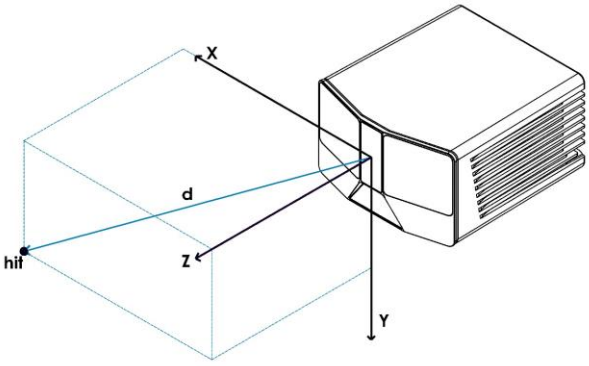
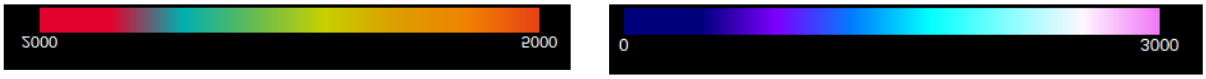


Figure 29. Point cloud reference coordinate system.



(a) (b)

Figure 30. Colour scale for distance (a) and intensity (b).

When all data packets have been sent, L3CAM signals the end of transmission of the point cloud frame with a footer package with STX = 0x03:

Table 8. Point cloud frame footer.

FIELD	**STX**
SIZE (Bytes)	1
Value	0x03

In APPENDIX 1 there is a sample code describing how to read the point cloud frame from the host.

3.2.2 Image

The transmission sequence of an image frame is described in Figure 31. Note that in systems with the perception functions enabled, the image data is complemented with the result of the perception AI algorithms.

[3] Colour scales can be changed with the API.

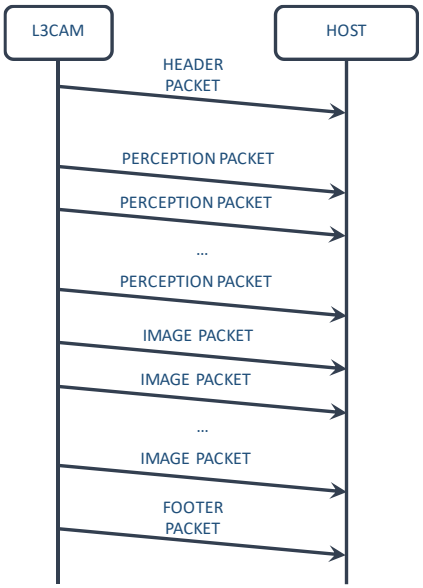


Figure 31. Transmission sequence of image data with results from perception functions.

The transmission begins with a header packet. Then, if the perception functions have detected objects in the image, L3CAM transmits as many perception packets as necessary. The transmission continues with the image packets and ends with a footer packet.

The header packet is as follows:

Table 9. Image frame header.

FIELD	STX	WI	HE	CHANNELS	TIMESTAMP	N_DET
SIZE (Bytes)	1	2 (int16_t)	2 (int16_t)	1 (byte)	4 (uint32_t)	1 (byte)
Value	0x02					

where:

- STX is a header packet identifier with value 0x02;
- WI: image width in pixels;
- WH: image height in pixels;
- Channels: number of channels in the image;
- Timestamp: an integer that represents the time when the frame was received. The time is in the L3CAM Jetson TX2 time and contains hhmmsszzz. For example, a frame received at 18:32:12:346 will return the timestamp value 183212346;
- N_DET: number of object detections.

If the perception functions are enabled and L3CAM detects objects, it sends one perception packet per object detected. Each perception packet contains the 2D coordinates of the bounding rectangle of the

detected object along with the confidence level of the detection. The structure of the perception packet is described in Table 10.

Table 10. Perception packet.

FIELD	Conf	X0	Y0	HE	WI
SIZE (Bytes)	2 (int16_t)	2 (int16_t)	2 (int16_t)	2 (int16_t)	2 (int16_t)

Where:

- Conf: confidence level; range [0-100].
- X0: x coordinate of the origin of the bounding box of the detected object (in pixels).
- Y0: y coordinate of the origin of the bounding box of the detected object (in pixels).
- HE: height of the bounding box (in pixels).
- WI: width of the bounding box (in pixels).

After the perception packets, or in case there are no perception packets, L3CAM starts sending the image packets. The image packets have a size of 8000 bytes. The last image packet can have a smaller size.

The end of transmission (footer) package is:

Table 11. Point cloud frame footer.

FIELD	STX
SIZE (Bytes)	1
Value	0x03

In APPENDIX 2 there is a sample code describing how the read images from the UDP stream.

3.3 RTSP PROTOCOL AND DATA DESCRIPTION

In this mode, L3CAM generates a separate pipeline for each sensor. The internal pipelines are shown in the next table. The pipelines can be modified using the API calls described in the API section.

Table 12. GStreamer pipeline.

SENSOR	GSTREAMER PIPELINE
LIDAR	appsrc ! video/x-raw,format=BGR,width=2440,height=2040 framerate=10/1 ! queue ! rtpvrawpay pt=96 mtu=8950 ! udpsink host=host_ip port=5040
Polarimetric Camera	appsrc ! video/x-raw,format=BGR,width=1440,height=960 framerate=10/1 ! queue ! rtpvrawpay pt=96 mtu=8950 ! udpsink host=host_ip port=5030

3.3.1 Point cloud

The LIDAR image resolution is not a video standard, and to be able to send it over RTSP using GStreamer API it is necessary to add block padding on the right and the bottom of the image before sending it over RTSP.

The result is a 1440x960 pixels image sent over RTSP, and this block padding can be removed in the same pipeline used to receive the RTSP stream as shown in the example.

```
gst-launch-1.0.exe -vvv udpsrc port="5040" caps = "application/x-rtp,
media=(string)video, clock-rate=(int)90000, encoding-name=(string)RAW,
sampling=(string)BGR, depth=(string)8, width=(string)1440, height=(string)960,
colorimetry=(string)BT601-5, payload=(int)96, framerate=10/1" ! rtpvrawdepay !
tee name=t t. ! queue ! videoconvert ! videocrop top=0, left=0, right=48,
bottom=714 ! autovideosink window-x=640, window-y=480, sync=false show-preroll-
frame=false
```

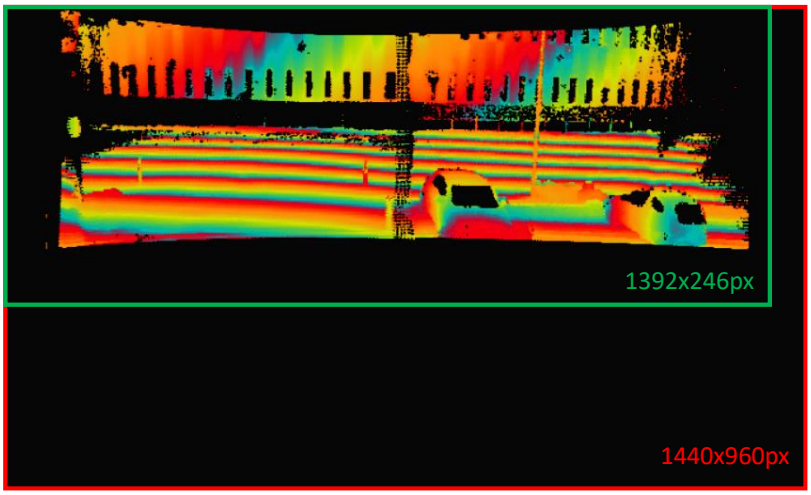


Figure 32. Point cloud and cropping area.

3.3.2 Image

To receive the RTSP video stream from the Polarimetric camera the following pipeline can be used.

```
gst-launch-1.0.exe -vvv udpsrc port="5030" caps = "application/x-rtp,
media=(string)video, clock-rate=(int)90000, encoding-name=(string)RAW,
sampling=(string)BGR, depth=(string)8, width=(string)2448, height=(string)2048,
colorimetry=(string)BT601-5, payload=(int)96, framerate=10/1" ! rtpvrawdepay !
tee name=t t. ! queue ! videoconvert ! autovideosink window-x=640, window-y=480,
sync=false show-preroll-frame=false
```

Figure 33. Polarimetric Bayer image. See image format in APPENDIX 3.

3.4 API FUNCTIONS LIST

The following table lists the functions of the library libL3Cam. Please note that the code to receive image data and point cloud data is not included in the API. It is the responsibility of the user to implement those functions following the examples provided in the appendixes.

Table 13. API functions.

FUNCTION NAME	CATEGORY
getBeamErrorDescription	Initialization and support
GET_VERSION	Initialization and support
INITIALIZE	Networking
FIND_DEVICES	Initialization and support
GET_DEVICE_STATUS	Initialization and support
GET_SENSORS_AVAILABLE	Initialization and support
CHANGE_STREAMING_PROTOCOL	Networking
GET_RTSP_PIPELINE	Networking
CHANGE_RTSP_PIPELINE	Networking
GET_NETWORK_CONFIGURATION	Networking
CHANGE_NETWORK_CONFIGURATION	Networking
START_DEVICE	Operation sequence
STOP_DEVICE	Operation sequence
START_STREAM	Operation sequence
STOP_STREAM	Operation sequence

FUNCTION NAME	CATEGORY
CHANGE_LASER_CLASS	Behaviour
CHANGE_POINTCLOUD_COLOR	Behaviour
CHANGE_POINTCLOUD_COLOR_RANGE	Behaviour
CHANGE_POINTCLOUD_FUSION_SENSOR[4]	Behaviour
CHANGE_POINTCLOUD_COLOR	Behaviour
CHANGE_POINTCLOUD_COLOR_RANGE	Behaviour
CHANGE_DISTANCE_RANGE	Behaviour
SET_LIDAR_DMD_ON	Behaviour
SET_LIDAR_DMD_PATTERNS	Behaviour
SET_POLARIMETRIC_CAMERA_DEFAULT_SETTINGS	Behaviour
CHANGE_POLARIMETRIC_CAMERA_BRIGHTNESS	Behaviour
CHANGE_POLARIMETRIC_CAMERA_BLACK_LEVEL	Behaviour
ENABLE_POLARIMETRIC_CAMERA_AUTO_GAIN	Behaviour
CHANGE_POLARIMETRIC_CAMERA_AUTO_GAIN_RANGE	Behaviour
CHANGE_POLARIMETRIC_CAMERA_GAIN	Behaviour
ENABLE_POLARIMETRIC_CAMERA_AUTO_EXPOSURE_TIME	Behaviour
CHANGE_POLARIMETRIC_CAMERA_AUTO_EXPOSURE_TIME_RANGE	Behaviour
CHANGE_POLARIMETRIC_CAMERA_EXPOSURE_TIME	Behaviour

3.5 API FUNCTIONS DESCRIPTION

getBeamErrorDescription

```
const char * getBeamErrorDescription(int error_code);
```

Description

Returns the description of the error code returned by any of the library functions.

Parameters

error_code	Integer with the returned error of any API call.

Returns

Char pointer with the description of the error returned by the library.

Example

[4] Only supported in system configurations with the data fusion enabled.

```
char *desc = NULL;
int32_t error = L3CAM_OK;

error = INITIALIZE();
desc = getBeamErrorDescription(error);
printf("Initialize response %d - %s \n", error, desc);
```

GET_VERSION

```
const char * GET_VERSION();
```

Description

Returns the version of the library.

Returns

Char pointer with the version of the library.

Example

```
char *version = NULL;
version = GET_VERSION();
printf("Library version %s \n", version);
```

INITIALIZE

```
int INITIALIZE();
```

Description

This function initializes the library. Internally starts the TCP port for communications with an L3Cam device, and starts the discovery thread to find L3Cam devices in the network.

Example

```
char *desc = NULL;
int32_t error = L3CAM_OK;

error = INITIALIZE();
desc = getBeamErrorDescription(error);
printf("Initialize response %d - %s \n", error, desc);
```

FIND_DEVICES

```
int FIND_DEVICES (l3cam devices[], int *num_devices);
```

Description

Call this function to get the available devices found in the host network, the current version of the library only allows communications with a single L3CAM device, so the l3cam array should be initialized with one item as described in the example.

Parameters

devices []	Array to store the available devices in the network with l3cam structure.
num_devices	Pointer to integer where the number of devices found is returned.

Example

```
int num_devices = 0;
l3cam devices[1];
char *desc = NULL;

int32_t error = GET_DEVICES(devices, &num_devices);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Get devices response %d - %s \n", error, desc);
}

if(num_devices > 0){
    printf("ip address of device  %s\n", devices[0].ip_address);
}
```

GET_DEVICE_STATUS

```
int GET_DEVICE_STATUS (l3cam device, int32_t *system_status);
```

Description

Call this function to get the current device status. See status definition for more information.

Parameters

device	Structure of an available device to execute the function.
system_status	Pointer to an integer (32 bit) where the status is returned.

Example

```
int32_t system_status = 0;
char *desc = NULL;

int32_t error = GET_DEVICE_STATUS(devices[0], &system_status);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Get device status response %d - %s \n", error, desc);
}

printf("System status %d\n", system_status);
```

GET_SENSORS_AVAILABLE

```
int GET_SENSORS_AVAILABLE (l3cam device, sensor sensors [],
                           int *num_sensors);
```

Description

Call this function to get the available sensors and the information of each sensor in the device. The number of sensors available will vary depending on the L3CAM model.

Parameters

device	Structure of an available device to execute the function.
sensors[]	Sensor struct array to store the sensors data.
num_sensors	Pointer to store the number of sensors available in the device

Example

```
int num_sensors = 0;
sensor av_sensors[6];
char *desc = NULL;

int32_t error = GET_SENSORS_AVAILABLE(devices[0], av_sensors, &num_sensors);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Get sensors available response %d - %s \n", error, desc);
}
```

CHANGE_STREAMING_PROTOCOL

```
int CHANGE_STREAMING_PROTOCOL (l3cam device, sensor *sensor_id);
```

Description

Call this function to change the streaming protocol for the desired sensor. See 3.2 for more information.

Parameters

device	Structure of an available device to execute the function.
sensor_id	Pointer to the sensor to change the protocol

Example

```
char *desc = NULL;
av_sensors[0].protocol = protocol_rtsp;

int32_t error = CHANGE_STREAMING_PROTOCOL(devices[0], &av_sensors[0]);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change streaming protocol response %d - %s \n", error, desc);
}
```

GET_RTSP_PIPELINE

```
int GET_RTSP_PIPELINE(l3cam device, sensor sensor_id, char **pipeline);
```

Description

Call this function to get the current RTSP pipeline of a specific sensor.

Parameters

device	Structure of an available device to execute the function.
sensor_id	The sensor to get the RTSP pipeline
pipeline	Pointer to a pointer with the pipeline string

Example

```
char *desc = NULL;
char *rtsp_pipeline = NULL;

int32_t error = GET_RTSP_PIPELINE(devices[0], av_sensors[0], &rtsp_pipeline);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Get RTSP pipeline response %d - %s \n", error, desc);
}
else{
    print("%s \n", rtsp_pipeline);
}
```

CHANGE_RTSP_PIPELINE

```
int CHANGE_RTSP_PIPELINE(l3cam device, sensor sensor_id,
                         char *pipeline);
```

Description

Call this function to change the current RTSP pipeline of the specific sensor.

Parameters

device	Structure of an available device to execute the function.
sensor_id	The sensor to change the RTSP pipeline
pipeline	Pointer with the new pipeline string

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;

error = CHANGE_RTSP_PIPELINE(devices[0], av_sensors[0], "new_pipeline");

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Initialize response %d - %s \n", error, desc);
}
```

GET_NETWORK_CONFIGURATION

```
int GET_NETWORK_CONFIGURATION(l3cam device, char **ip_address,
                              char **netmask, char **gateway);
```

Description

Call this function to get the current network configuration of the L3CAM device.

Parameters

device	Structure of an available device to execute the function.
ip_address	Pointer to pointer where the IP address is returned
netmask	Pointer to pointer where the netmask is returned as string
gateway	Pointer to pointer where the gateway is returned

Example

```
char *desc = NULL;
char *address = NULL;
char *netmask = NULL;
char *gateway = NULL;

int32_t error = L3CAM_OK;
error = GET_NETWORK_CONFIGURATION(devices[0], &address, &netmask, &gateway);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Get network configuration response %d - %s \n", error, desc);
}else{
    printf("address %s netmask %s gateway %s \n", address, netmask, gateway);
}
```

CHANGE_NETWORK_CONFIGURATION

```
int CHANGE_NETWORK_CONFIGURATION(l3cam device, char *ip_address,
                                 char *netmask, char *gateway);
```

Description

Call this function to change the L3Cam device network configuration.

Parameters

device	Structure of an available device to execute the function.
ip_address	Pointer with the new IP address
netmask	Pointer with the new netmask as a string
gateway	Pointer with the new gateway

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = CHANGE_NETWORK_CONFIGURATION(devices[0],
                                     "192.168.5.15","255.255.255.0","0.0.0.0");

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change network response %d - %s \n", error, desc);
}
```

START_DEVICE

```
int START_DEVICE (l3cam device);
```

Description

Call this function to initialize the device, it initializes all the internal sensors and the streaming protocols.

Parameters

device	Structure of an available device to execute the function.

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = START_DEVICE(devices[0]);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Start device response %d - %s \n", error, desc);
}
```

STOP_DEVICE

```
int STOP_DEVICE (l3cam device);
```

Description

Call this function to stop the device. It stops the internal sensors and streaming protocols.

Parameters

device	Structure of an available device to execute the function.

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = STOP_DEVICE(devices[0]);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Stop device response %d - %s \n", error, desc);
}
```

START_STREAM

```
int START_STREAM (l3cam device);
```

Description

Call this function to start the streaming of the device. Each sensor will stream with the selected streaming protocol. See 3.1 and 3.2 for more information.

Parameters

device	Structure of an available device to execute the function.

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = START_STREAM(devices[0]);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Start stream response %d - %s \n", error, desc);
}
```

STOP_STREAM

```
int STOP_STREAM (l3cam device);
```

Description

Call this function to stop the sensors from streaming.

Parameters

device	Structure of an available device to execute the function.

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = STOP_STREAM(devices[0]);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Stop stream response %d - %s \n", error, desc);
}
```

CHANGE_LASER_CLASS

```
int CHANGE_LASER_CLASS (l3cam device, int laser_class);
```

Description

Call this function to change the behaviour of the laser, this function has to be used carefully. If the laser is set to CLASS 3R special protection has to be used by the user. See 2.1 for more information.

Parameters

device	Structure of an available device to execute the function.
laser_class	Integer with the new laser class. See APPENDIX 6 for more information.

Example

```
char *desc = NULL;

int32_t error = CHANGE_LASER_CLASS(devices[0], laser_class_3R);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change laser class response %d - %s \n", error, desc);
}
```

CHANGE_POINTCLOUD_COLOR

```
int CHANGE_POINTCLOUD_COLOR(l3cam device, int visualization_color);
```

Description

Call this function to change the colour representation of the point cloud. See 3.2 and APPENDIX 6 for more information about point cloud colours.

Parameters

device	Structure of an available device to execute the function.
visualization_color	Integer with the colour of the point cloud.

Example

```
char *desc = NULL;
int32_t error = CHANGE_POINTCLOUD_COLOR(devices[0], RAINBOW_Z);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change pointcloud color response %d - %s \n", error, desc);
}
```

CHANGE_POINTCLOUD_COLOR_RANGE

```
int CHANGE_POINTCLOUD_COLOR_RANGE(l3cam device, int max_value, int min_value);
```

Description

Call this function to change the colour range representation of the point cloud. See 3.2 and APPENDIX 6 for more information about point cloud colours.

Parameters

device	Structure of an available device to execute the function.
max_value	Maximum value for the colour representation.
min_value	Minimum value for the colour representation.

Example

```
char *desc = NULL;
int32_t error = CHANGE_POINTCLOUD_COLOR_RANGE(devices[0], 5000, 0);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change pointcloud color range response %d - %s \n", error, desc);
}
```

CHANGE_DISTANCE_RANGE

```
int CHANGE_DISTANCE_RANGE(l3cam device, int min_distance, int max_distance);
```

Description

Call this function to apply a distance filter to the point cloud. The resulting point cloud will show the points delimited between the minimum distance and the maximum distance.

Parameters

device	Structure of an available device to execute the function.
min_distance	Minimum distance of the point cloud, the value is in millimetres (mm)
max_distance	Maximum distance of the point cloud, the value is in millimetres (mm)

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
//!filter the pointcloud between 2 meters and 30 meters
error = CHANGE_DISTANCE_RANGE(devices[0], 2000, 30000);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change distance range response %d - %s \n", error, desc);
}
```

SET_LIDAR_DMD_ON

```
int SET_LIDAR_DMD_ON(l3cam device);
```

Description

Call this function to change the DMD behaviour to use the full array of mirrors. This function will make the LIDAR lose noise filtering capabilities.

Parameters

device	Structure of an available device to execute the function.

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = SET_LIDAR_DMD_ON(devices[0]);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Set DMD ON response %d - %s \n", error, desc);
}
```

SET_LIDAR_DMD_PATTERNS

```
int SET_LIDAR_DMD_PATTERNS(l3cam device, PARAMS);
```

Description

Call this function to return to “normal” LIDAR behaviour. The DMD will return to patterns mode.

Parameters

device	Structure of an available device to execute the function.

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = SET_LIDAR_DMD_PATTERNS(devices[0]);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Set DMD Patterns response %d - %s \n", error, desc);
}
```

SET_POLARIMETRIC_CAMERA_DEFAULT_SETTINGS

```
int SET_POLARIMETRIC_CAMERA_DEFAULT_SETTINGS(l3cam device);
```

Description

Call this function to change all the polarimetric camera parameters to the system defaults.

Parameters

device	Structure of an available device to execute the function.

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = SET_POLARIMETRIC_CAMERA_DEFAULT_SETTINGS(devices[0]);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Set polarimetric camera default parameters response %d - %s \n",
            error, desc);
}
```

CHANGE_POLARIMETRIC_CAMERA_BRIGHTNESS

```
int CHANGE_POLARIMETRIC_CAMERA_BRIGHTNESS(l3cam device, int brightness);
```

Description

Call this function to change the camera brightness.

Parameters

device	Structure of an available device to execute the function.
brightness	Integer with the new brightness value. The range value is 0 - 255

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = CHANGE_POLARIMETRIC_CAMERA_BRIGHTNESS(devices[0], 210);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change polarimetric brightness response %d - %s \n", error, desc);
}
```

CHANGE_POLARIMETRIC_CAMERA_BLACK_LEVEL

```
int CHANGE_POLARIMETRIC_CAMERA_BLACK_LEVEL(l3cam device, float black_level);
```

Description

Call this function to change the black level of the polarimetric camera.

Parameters

device	Structure of an available device to execute the function.
black_level	Float with the new black level value. The range value is 0 – 12.5

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = CHANGE_POLARIMETRIC_BLACK_LEVEL(devices[0], 8.5);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change polarimetric black level response %d - %s\n", error, desc);
}
```

ENABLE_POLARIMETRIC_CAMERA_AUTO_GAIN

```
int ENABLE_POLARIMETRIC_CAMERA_AUTO_GAIN(l3cam device, bool enabled);
```

Description

Call this function to enable or disable the auto-gain feature of the polarimetric camera.

Parameters

device	Structure of an available device to execute the function.
enabled	Boolean to enable/disable the auto gain feature. true to enable auto gain and false to disable it.

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = ENABLE_POLARIMETRIC_CAMERA_AUTOGAIN(devices[0], true);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Enable polarimetric auto gain response %d - %s \n", error, desc);
}
```

CHANGE_POLARIMETRIC_CAMERA_AUTO_GAIN_RANGE

```
int CHANAGE_POLARIMETRIC_CAMERA_AUTO_GAIN_RANGE(l3cam device, float min_gain,
                                                float max_gain);
```

Description

Call this function to change the range of the polarimetric camera auto-gain feature. When the auto-gain feature is enabled, the camera will adjust it using the range of the specified values.

Parameters

device	Structure of an available device to execute the function.
min_gain	Float with the minimum gain value for the auto-gain feature. The range is 0 to 48.0 dB
max_gain	Float with the maximum gain value for the auto-gain feature. The range is 0 to 48.0 dB

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = CHANGE_POLARIMETRIC_CAMERA_AUTO_GAIN_RANGE(devices[0], 20.5, 35.0);

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change polarimetric auto gain range response %d - %s \n", error,
            desc);
}
```

CHANGE_POLARIMETRIC_CAMERA_GAIN

```
int CHANGE_POLARIMETRIC_CAMERA_GAIN(l3cam device, float gain);
```

Description

Call this function to manually set the gain of the polarimetric camera. The auto gain feature must be disabled to use this function

Parameters

device	Structure of an available device to execute the function.
gain	The value of the gain. The range is 0 – 48.0 dB

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
//! Disable the polarimetric autogain
error = ENABLE_POLARIMETRIC_CAMERA_AUTO_GAIN(devices[0], false)

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Disable polarimetric auto gain response %d - %s \n", error, desc);
}
else{
    error = CHANGE_POLARIMETRIC_CAMERA_GAIN(devices[0], 30.0);
    if(error != BEAM_NO_ERROR){
        desc = getBeamErrorDescription(error);
        printf("Change polarimetric camera gain response %d - %s \n", error,
                desc);
```

```
    }
}
```

ENABLE_POLARIMETRIC_CAMERA_AUTO_EXPOSURE_TIME

```
int ENABLE_POLARIMETRIC_CAMERA_AUTO_EXPOSURE_TIME(l3cam device, bool enabled);
```

Description

Call this function to enable or disable the auto-exposure time feature of the polarimetric camera.

Parameters

device	Structure of an available device to execute the function.
enabled	Boolean to enable/disable the auto-exposure time feature. True to enable auto exposure time and false to disable it.

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;

error = ENABLE_POLARIMETRIC_CAMERA_AUTO_EXPOSURE_TIME(devices[0], true);
if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Enable polarimetric auto exposure time response %d - %s \n",
            error, desc);
}
```

CHANGE_POLARIMETRIC_CAMERA_AUTO_EXPOSURE_TIME_RANGE

```
int CHANGE_POLARIMETRIC_CAMERA_AUTO_EXPOSURE_TIME_RANGE(l3cam device,
                                    float min_exposure, float min_exposure);
```

Description

Call this function to change the auto-exposure time values range. When the auto-exposure time feature is enabled, the value will vary in the range of the specified values.

Parameters

device	Structure of an available device to execute the function.
min_exposure	Float with the minimum exposure time value for the auto exposure time feature. The range is 33.456us to 1000000.0us
max_exposure	Float with the maximum exposure time value for the auto exposure time feature. The range is 33.456us to 1000000.0us

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
error = CHANGE_POLARIMETRIC_CAMERA_AUTO_EXPOSURE_TIME_RANGE(devices[0], 100.0,
                                                            500.0);
if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Change Polarimetric auto exposure time response %d - %s \n",
           error, desc);
}
```

CHANGE_POLARIMETRIC_CAMERA_EXPOSURE_TIME

```
int CHANGE_POLARIMETRIC_CAMERA_EXPOSURE_TIME(l3cam device,
                                             float exposure_time);
```

Description

Call this function to change the polarimetric camera exposure time. The auto-exposure time must be disabled to be able to use this function.

Parameters

device	Structure of an available device to execute the function.
exposure_time	Float with the exposure time value. The range is 33.456us to 1000000.0us

Example

```
char *desc = NULL;

int32_t error = L3CAM_OK;
//! Disable the polarimetric auto exposure time
error = ENABLE_POLARIMETRIC_CAMERA_AUTO_EXPOSURE_TIME(devices[0], false)

if(error != BEAM_NO_ERROR){
    desc = getBeamErrorDescription(error);
    printf("Disable polarimetric auto exposure time response %d - %s \n",
            error, desc);
}
else{
    error = CHANGE_POLARIMETRIC_CAMERA_EXPOSURE_TIME(devices[0], 300.0);
    if(error != BEAM_NO_ERROR){
        desc = getBeamErrorDescription(error);
        printf("Change polarimetric camera exposure time response %d - %s \n",
                error, desc);
    }
}
```

APPENDIXES

APPENDIX 1 READING A POINT CLOUD

```
void udpReceiverController::readPointcloud(){

    int points_received = 1;
    socklen_t socket_len = sizeof(m_socket);
    char* buffer = NULL;
    buffer = (char*)malloc(64004);

    while(1)
    {
       int size_read = recvfrom(m_socket_descriptor, buffer, 64004, 0, (struct
       sockaddr*)&m_socket, &socket_len);
       if(size_read == 17){
            memcpy(&m_pointcloud_size, &buffer[1], 4);
            m_pointcloud_data = (int32_t*)malloc(   sizeof(int32_t)*(((m_pointcloud_size) *5)
            + 1));
            memcpy(&m_pointcloud_data[0], &m_pointcloud_size, sizeof(int32_t));
            int32_t suma_1, suma_2;
            memcpy(&suma_1, &buffer[5], sizeof(int32_t));
            memcpy(&suma_2, &buffer[9], sizeof(int32_t));
            memcpy(&m_timestamp, &buffer[13], sizeof(uint32_t));
            m_is_reading_pointcloud = true;
            points_received = 1;
        }
        else if(size_read == 1){
            m_is_reading_pointcloud = false;
            int32_t *data_received = (int32_t*)malloc(sizeof(int32_t)*
            (m_pointcloud_size*5)+1);

            memcpy(&data_received[0], &m_pointcloud_data[0],sizeof(int32_t)*
            ((m_pointcloud_size*5)+1));

            emit pointcloudReadyToShow(data_received);
            free(m_pointcloud_data);
            points_received = 1;
        }
        else if(size_read > 0){
            if(m_is_reading_pointcloud){
                int32_t points = 0;
                memcpy(&points, &buffer[0], 4);
                memcpy(&m_pointcloud_data[points_received],&buffer[4],
                (sizeof(int32_t)*(points*5)));
                points_received+=points*5;
            }
        }
    }
    free(buffer);
}
```

APPENDIX 2 READING AN IMAGE

```
struct boxImage{
    int16_t x;
    int16_t y;
    int16_t height;
    int16_t width;
};

struct detectionImage{
    boxImage box;
    uint16_t confidence;
};

void udpReceiverController::readRgbImage(){
    m_image_buffer = NULL;
    int bytes_count = 0;
    socklen_t socket_len = sizeof(m_socket);
    char* buffer;
    buffer = (char*)malloc(64000);
    detectionImage curr_det;

    while(1){
        int size_read = recvfrom(m_socket_descriptor, buffer, 64000, 0,
                                 (struct sockaddr*)&m_socket, &socket_len);

        if(size_read == 11){

            memcpy(&m_image_height, &buffer[1], 2);
            memcpy(&m_image_width, &buffer[3], 2);
            memcpy(&m_image_channels, &buffer[5], 1);
            memcpy(&m_timestamp, &buffer[6], sizeof(uint32_t));
            memcpy(&m_image_detections, &buffer[10], 1);
            m_image_data_size = m_image_height*m_image_width*m_image_channels;
            m_image_buffer = (char*)malloc(m_image_data_size);
            m_is_reading_image = true;
            m_image_ready = false;
            m_detections.clear();
            bytes_count = 0;

        }else if(size_read == 1 || bytes_count == m_image_data_size ){

            m_is_reading_image = false;
            m_image_ready = true;
            bytes_count = 0;
            uint8_t* image_pointer = (uint8_t*)malloc(m_image_data_size);
            memcpy(image_pointer, m_image_buffer, m_image_data_size);

            emit imageRgbReadyToShow(image_pointer, m_image_height, m_image_width,
m_image_channels, m_detections);
            free(m_image_buffer);
            m_image_detections = 0;
```

```
        }else if(size_read != -1){
            if(m_image_detections > 0){
                //!read detections packages
                memcpy(&curr_det.confidence, &buffer[0], 2);
                memcpy(&curr_det.box.x, &buffer[2], 2);
                memcpy(&curr_det.box.y, &buffer[4], 2);
                memcpy(&curr_det.box.height, &buffer[6], 2);
                memcpy(&curr_det.box.width, &buffer[8], 2);
                m_detections.push_back(curr_det);
                --m_image_detections;
                continue;
            }
            if(m_is_reading_image){
                memcpy(&m_image_buffer[bytes_count], buffer, size_read);
                bytes_count+=size_read;
            }
        }
    }
    free(buffer);
}
```

APPENDIX 3 POLARIMETRIC PIXEL FORMAT

L3CAM supports one Lucid PHX050S1 polarized colour camera. These cameras have unprocessed BayerRG pixel formats. The data is transmitted by the camera following the GenICam standard's Pixel Format naming convention [RD 2]. L3CAM forwards the data using the same format.

The colour filter array of the PHX050S1 is as follows:

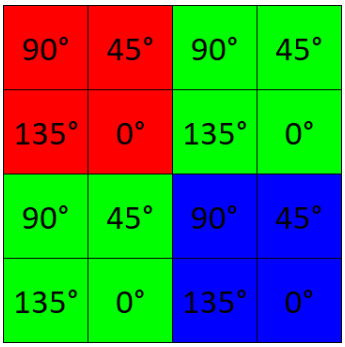


*Figure 34. Colour filter array of Lucid **PHX050S1**.*

APPENDIX 4 RETURN CODES

Table 14. Library return codes.

RETURN CODE	DESCRIPTION
0	L3CAM_OK – The API call executed successfully
21	L3CAM_NO_SENSORS_AVAILABLE – The L3CAM device was not able to find any device attached to it.
210	L3CAM_TIMEOUT_ERROR – The library timed out waiting for a response from L3CAM
211	L3CAM_ERROR_OUT_OF_MEMORY – The library has no available memory
220	L3CAM_UNDEFINED_LASER_CLASS – The user input of laser class is erroneous
222	L3CAM_UNDEFINED_SENSOR – The sensor input is erroneous
223	L3CAM_VALUE_OUT_OF_RANGE – The user input for a value is erroneous
224	L3CAM_SENSOR_NOT_AVAILABLE – The sensor input is erroneous
230	L3CAM_ERROR_CREATING_TCP_CLIENT_SOCKET – Unable to create a socket to connect with the L3CAM TCP server
231	L3CAM_ERROR_INITIALIZING_WSA - *Windows only, error initializing the WSA subsystem.
232	L3CAM_ERROR_CONNECTING_WITH_TCP_SERVER – The library failed to connect with the L3CAM TCP server
233	L3CAM_ERROR_SENDING_TCP_MESSAGE – Error while trying to send a message over TCP to L3CAM
234	L3CAM_ERROR_CREATING_TCP_SERVER_SOCKET – Error creating the Library TCP server
235	L3CAM_EROR_BINDING_SOCKET – Error when binding library TCP socket, check local ports and system privileges.
236	L3CAM_ERROR_STARTING_TCP_SERVER – The library is unable to start the local TCP server.
237	L3CAM_ERROR_ACCEPTING_TCP_CLIENT – The local TCP server was unable to accept the L3CAM connection

Table 15. Polarimetric camera errors.

ERROR CODE	DESCRIPTION
30	ERROR_POL_STD_EXCEPTION
31	ERROR_POL_RUNTIME_EXCEPTION
32	ERROR_POL_GENERIC_EXEPTION
33	ERROR_POL_TIMEOUT_EXCEPTION
34	ERROR_POL_UNDEFINED_ERROR
35	ERROR_POL_PERCEPTION_EXCEPTION

APPENDIX 5 STATUS

The L3CAM device returns the device status encoded in a 16-bit unsigned integer, each bit corresponds to a possible status. The bit 0 is the less significant bit of the integer.

Table 16. Status bits.

BIT	Description
0	UNDEFINED STATUS
1	BOOTING
2	ALARMS AT BOOTING
3	ERRORS AT BOOTING
4	BOOTING UNCONFIGURED
5	STARTED
6	STARTED WITH ALARMS
7	STARTED WITH ERRORS
8	STARTED UNCONFIGURED
9	STOPPING
10	STOPED
11	STOPED WITH ALARMS
12	UNUSED
13	UNUSED
14	UNUSED
15	UNUSED

APPENDIX 6 DATA STRUCTURES

Enum sensorTypes

```
typedef enum sensorTypes {
    sensor_lidar = 1,
    sensor_econ_rgb,
    sensor_pol,
    sensor_thermal,
}sensorTypes;
```

This enum contains the possible sensors available in the L3CAM.

Table 17. sensorTypes enum fields.

Parameter	Description
Sensor_lidar	Identifier for LIDAR sensor
Sensor_econ_rgb	Identifier for RGB camera
Sensor_pol	Identifier for Polarimetric camera (Mono or RGB)
Sensor_thermal	Identifier for Thermal camera

Enum laserClass

```
enum laserClass {
    laser_class_1 = 0,
    laser_class_3R,
};
```

Table 18. laserClass enum fields.

Parameter	Description
Laser_class_1	Class 1 lasers or systems cannot emit accessible laser radiation in excess of the applicable Class 1 AEL for any exposure times within the maximum duration inherent in the design or intended use of the laser. Class 1 lasers are exempt from all beam-hazard control measures.
Laser_class_3R	A Class 3R laser is considered safe if handled carefully, with restricted beam viewing.

Enum pointCloudColor

```
enum pointCloudColor{
    RAINBOW = 0,
    RAINBOW_Z,
    INTENSITY,
};
```

Table 19. pointCloudColor enum fields.

Parameter	Description
RAINBOW	Colour that represents the spherical distance of the point
RAINBOW_Z	Colour that represents the distance on the Z-axis of the point
INTENSITY	Colour that represents the intensity of the point

Enum streamingProtocols

```
typedef enum streamingProtocols {
    protocol_raw_udp = 1,
    protocol_rtsp,
}streamingProtocols;
```

This enum describes the available streaming protocols for each sensor.

Table 20. streamingProtocols enum fields.

Parameter	Description
Protocol_raw_udp	Value to enable Raw UDP streaming mode
Protocol_rtsp	Value to enable the RTSP streaming mode

Struct sensor

```
typedef struct sensor{
    streamingProtocols protocol;
    sensorTypes sensor_type;
    uint8_t sensor_status;
    uint8_t image_type;
    bool perception_enabled;
}sensor;
```

This structure describes the parameters of each sensor in the L3CAM.

Table 21. sensor struct fields.

Parameter	Description
Protocol	Streaming protocol of the sensor
Sensor_type	Identify the sensor type

Parameter	Description
Sensor_status	Integer with the current sensor status
Image_type	Identify the current image type that the sensor is streaming
Perception_enabled	Boolean that shows if the perception algorithm is enabled in the sensor

Struct l3cam

```
typedef struct l3cam{
    char ip_address[16];
    char serial_number[10];
    uint8_t model;
}l3cam;
```

This structure describes the parameters of each L3CAM device available in the network.

Table 22. l3cam struct fields.

Parameter	Description
Ip_addres	The IP address of the device
Serial_number	The serial number of the device
Model	The model of the device

APPENDIX 7 L3CAM DIMENSIONS

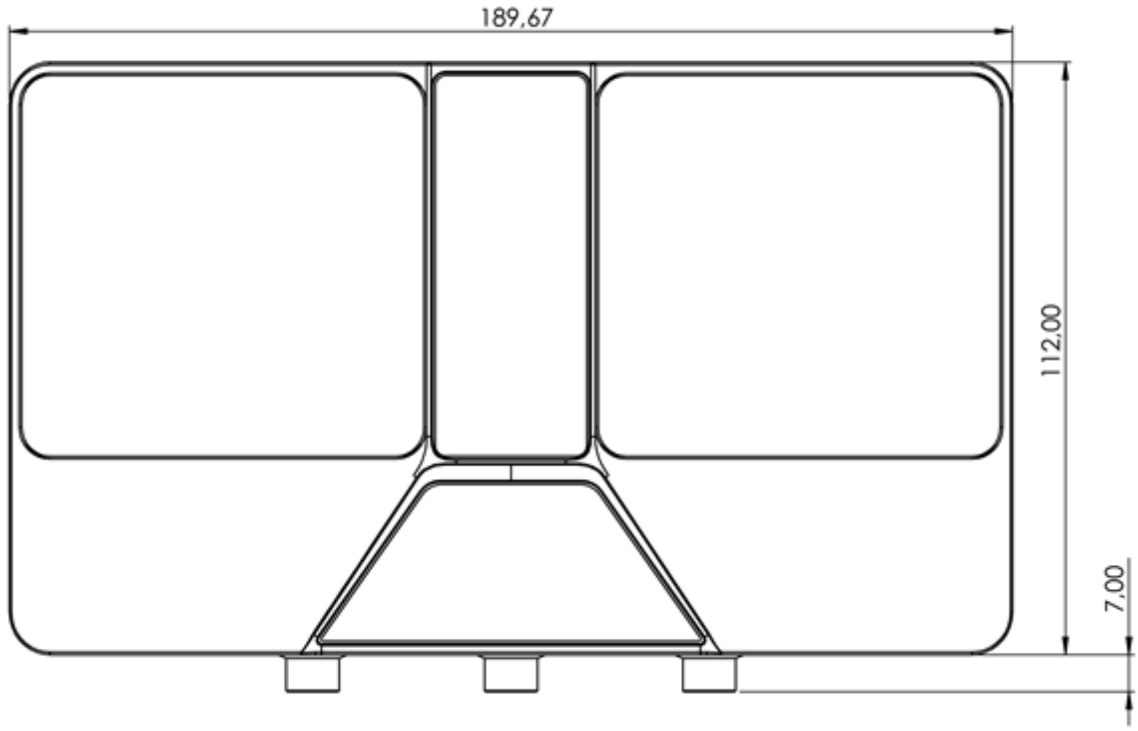


Figure 35. L3CAM front side dimensions.

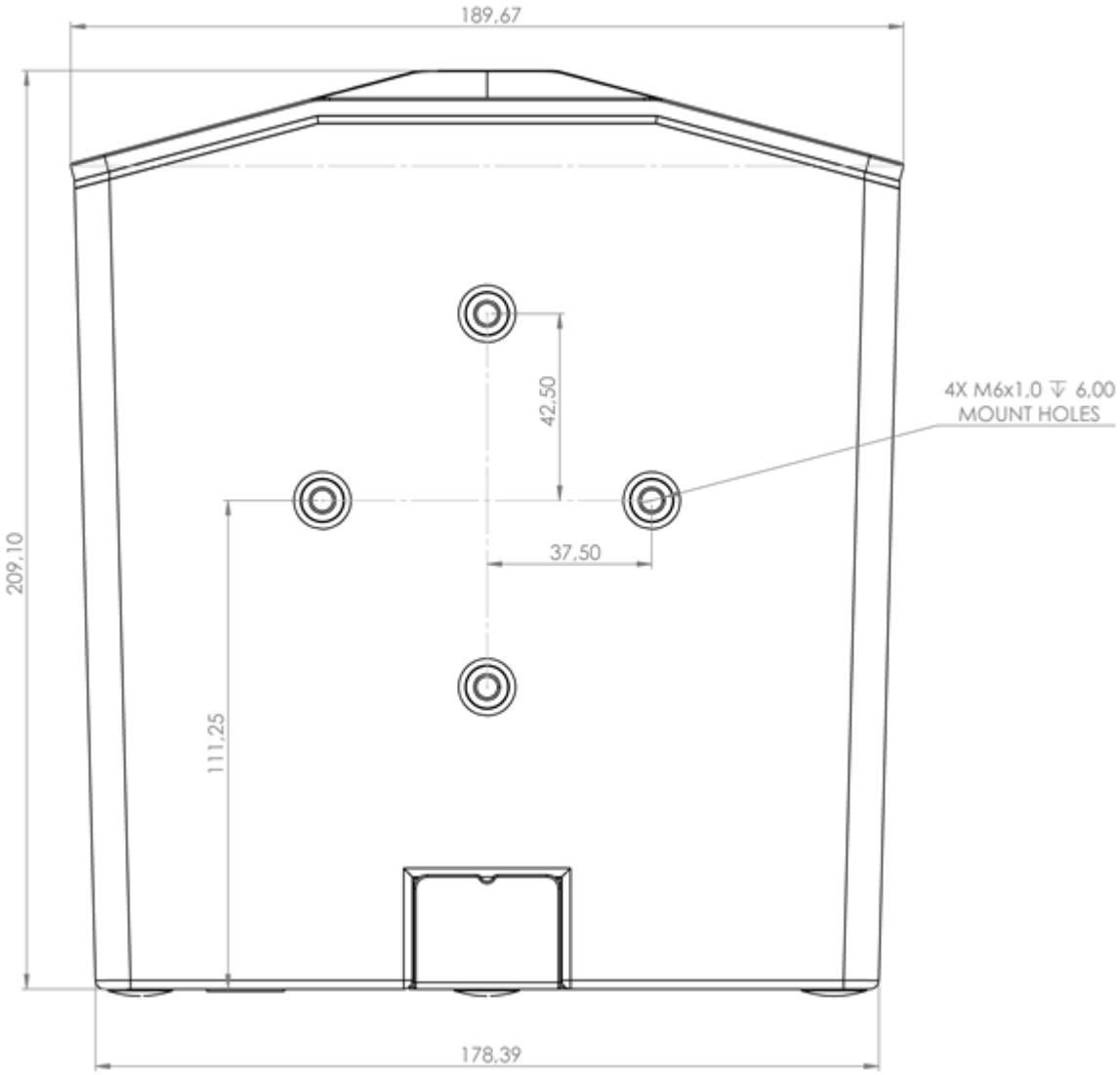


Figure 36, L3CAM mount holes dimensions.

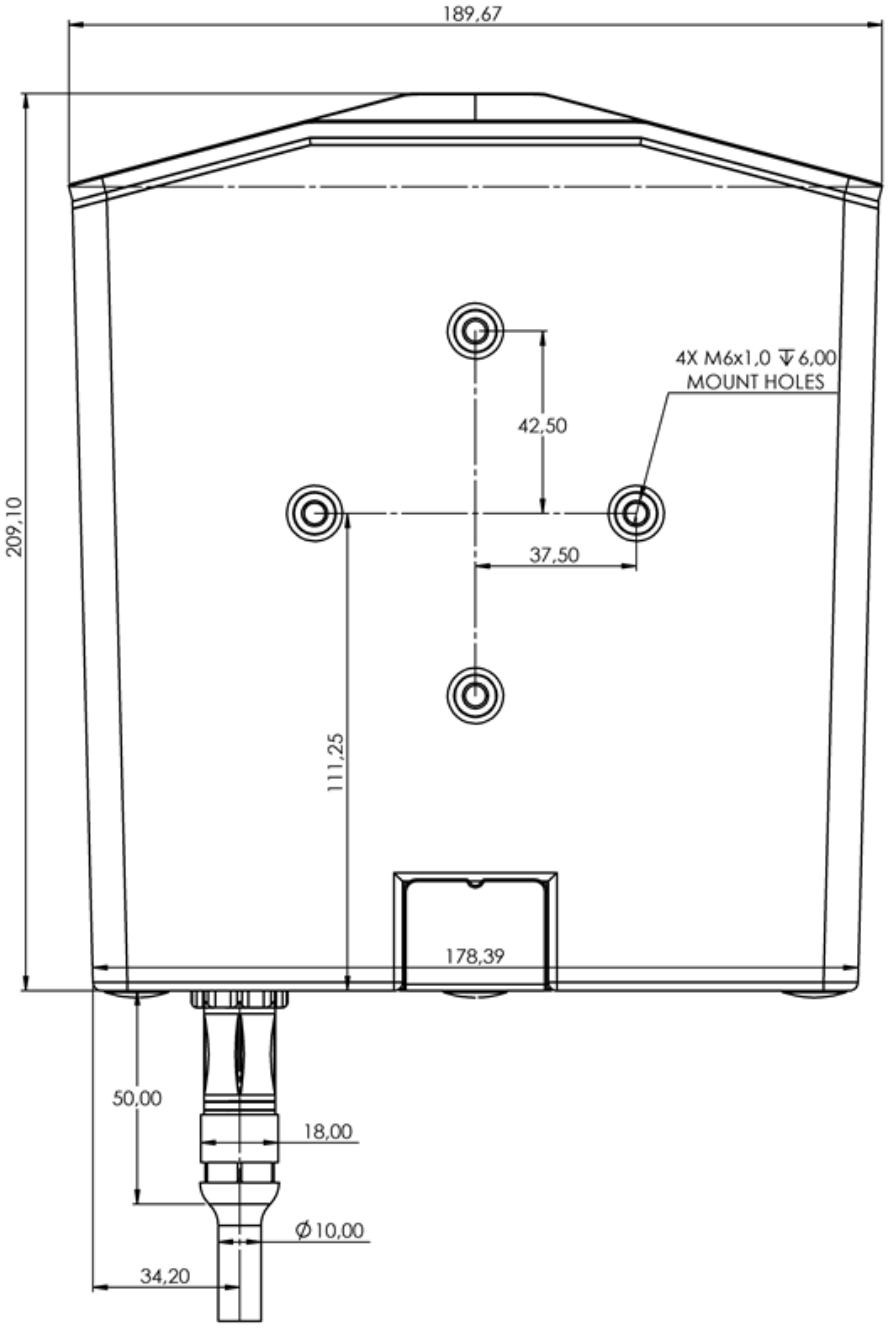


Figure 37. L3CAM connector cable dimensions.