

---

**O3R**

**ifm CSR**

**Mar 01, 2022**



# CONTENTS

<b>1 Everything related to the O3R family</b>	<b>1</b>
1.1 Products Description . . . . .	1
1.2 Getting Started . . . . .	6
1.3 Parameters . . . . .	9
1.4 Docker on O3R . . . . .	50
1.5 FAQ - Frequently Asked Questions . . . . .	65
<b>2 ifm3d library</b>	<b>67</b>
2.1 ifm3d Overview . . . . .	67
2.2 Installing the software . . . . .	68
2.3 Basic Library Usage . . . . .	80
2.4 ifm3d - Command Line Tool . . . . .	89
2.5 Python API Reference . . . . .	102
2.6 C++ API Reference . . . . .	116
<b>3 Indices and tables</b>	<b>117</b>
<b>4 ROS wrappers for ifm3d</b>	<b>119</b>
4.1 ifm3d-ros . . . . .	119
4.2 ifm3d-ros2 . . . . .	126
<b>5 Resources available for download</b>	<b>133</b>
5.1 Previous versions of the documentation . . . . .	133
<b>Python Module Index</b>	<b>135</b>
<b>Index</b>	<b>137</b>



---

**CHAPTER  
ONE**

---

## **EVERYTHING RELATED TO THE O3R FAMILY**

### **1.1 Products Description**

#### **1.1.1 FIRMWARE 0.13.13 RELEASE NOTES !!!!PRELIMINARY!!!!**

The following release note provides an overview of the features of the Firmware 0.13.13 Version. Please refer to the ifm O3Rs website [www.ifm3d.com](http://www.ifm3d.com) for further information.

#### **Previous Releases**

There is no previous FW release for image processing platform M03975. The FW-release 0.13.13 does not operate with image processing platform M03903 and their heads.

#### **Compatible Image Processing Platforms**

This firmware release can be applied to the following ifm image processing platform:

article	comment
M03975	preseries sample
M04239	preseries sample, housing change

#### **Supported Heads**

This firmware release supports the following ifm camera articles:

#### **General Features**

- Connectivity: The O3R image processing platform is a multi camera image processing platform.
  - Ports: ifm camera articles can be connected to all six ports (Port0 .. Port5)
  - Ethernet: 1x GB ethernet connection [eth0]

- Data Interface: The ifm3d library is recommended for interfacing with the O3R image processing platform ([download here](#)) The idea of ifm3d is to let developers quickly ramp up and also deploy the image processing platform. Sample programs, that illustrate the various functions and good usage, are provided for the following applications frameworks:
  - C++
  - Python bindings: use `pip install ifm3dpy`, (see [here](#))
  - ROS1 wrapper:
- SSH access: Access to the embedded Linux operating system is enabled through the `oem` user.
- Docker containers: Docker containers can be used to deploy code embedded on the image processing platform. Management of docker containers, like upload, download, autostart, delete is handled through SSH access by the `oem` user.

## ifm Camera Usage

The ifm camera heads are in one of the following states:

- CONF: Configuration state, no data acquisition
- IDLE: pause data acquisition
- RUN: periodic data acquisition @framerate

## 3D-Camera Features

**NEW** In contrast to other TOF-systems the ifm cameras have no ambiguity problem. Objects are measured only within the measurement range, which starts at an offset distance and for the length of the range. Objects before the offset and beyond the measurement range are not detected and more importantly can not disturb the measurement. The range and the offset can be adjusted to the current application. All data acquisitions are carried out in high dynamic mode with multiple exposure times.

- Modes:
  - `standard_range2m`: measurement range [0..2] with `offset=0` or [0.5..2.5] with `offset=0.5` etc.
  - `standard_range4m`: measurement range [0..4] with `offset=0` or [1..5] with `offset=1` etc.
  - `cyclic_4m_2m_4m_2m`: periodic change between 4m and 2m measurement range
- Acquisition Parameters:
  - `framerate`: periodic image acquisition at 1/framerate-intervals
  - `delay`: not functional (timing is currently free-running only)
  - `expLong`: long exposure time of high dynamic acquisition
  - `expShort`: short exposure time of high dynamic acquisition
  - `offset`: shift of measurement range in 0.5m steps
- Available Output Data:

- distance: radial distance between camera and object
  - x: x-coordinate of object pixel
  - y: y-coordinate of object pixel
  - z: z-coordinate of object pixel
  - amplitude: detected signal strength
  - confidence: metainformation for each pixel
  - Metadata: image width [pix], image height [pix], timestamps
- Data Processing:
    - various Thresholds: invalidate pixels on various criteria
    - Filters:
      - \* spatial filters and parametrization
      - \* temporal filters and parametrization

## RGB-Camera Features

The RGB cameras acquire color images of the objects.

- Modes:
  - autoexposure
- Acquisition Parameters:
  - framerate: periodic image acquisition at 1/framerate-intervals
  - delay: not functional (timing is currently free-running only)
- Available Output Data:
  - jpg-encoded RGB image

## Known Issues

- Connectivity: ports must be connected pairwise with the same head-type: [Port0,Port1] [Port2,Port3] [Port4,Port5]
- Frame rate limitation: M03976 is limited to 1..10Hz
- No time synchronization: synchronized data acquisition / image triggering between ports is currently not possible.
- Software triggering in IDLE state is not operational

**Look forward to these features in future releases**

- CAN: messages can be retrieved via a CAN bus system
- [eth1]: enable the second ethernet port
- USB: a mass storage device is accessible for data collection
- avoid crosstalk between multiple cameras by selecting a channel value
- synchronization among different ports - it will be possible to define the acquisition timing for SW-triggered and periodic acquisitions (eg. two heads have a synchronized acquisition)

**Update Firmware Procedure:**

Update procedure:

1. Open <http://192.168.0.69:8080/> in web browser. The SWUpdate web interface is shown.
2. Drag and drop the \*.swu firmware file into the software update-window. The upload procedure starts.

### **1.1.2 Images description**

#### **Description of the available images**

This document gives a high level overview of the images available for the O3R. Receiving certain images can be turned ON/OFF using the schema mask (documentation coming soon).

Note: For more information about the types, sizes and other implementation aspects, please refer to the [ifm3d API documentation](#).

#### **Raw Amplitude image and Amplitude image**

Each pixel of the amplitude matrix denotes the amount of modulated light (i.e., the light from the camera's active illumination) which is reflected by the appropriate object. Higher values indicate higher signal strengths and thus a lower amount of noise on the corresponding distance measurements. The raw amplitude image is not normalized, which can lead to inhomogeneous image impression if a certain pixel is taken from the short exposure time and some of its neighbors are not. Invalid pixels have a value of zero.

The amplitude image is the normalized image over the different exposure times.

### Distance image (radial)

Each pixel of the distance image denotes the ToF distance measured by the corresponding pixel or group of pixels of the imager, along the respective pixel direction. The distance value is corrected by the camera's calibration, excluding effects caused by multi-path interference and multiple objects contributions (e.g., [mixed pixels](#)). The reference point is the center of the back of the camera head's housing. Invalid pixels have a value of zero.

### Distance noise (radial)

The distance noise represent the estimated standard deviation of the distance error, in meters for each pixel. Coming soon in ifm3d library

### Confidence

The confidence image give detail about the validity of each pixel and the reason (if any) why it was invalidated. See details [here](#).

### Reflectivity

The reflectivity image represents the estimated reflectivity in the near infrared spectrum of the objects in the scene. See also the [minimum reflectivity filter](#). coming soon in ifm3d library

### Point cloud (XYZ)

The XYZ image (also called point cloud) is a 3-channel image of the spacial planes X, Y and Z. It uses the Cartesian coordinate system. The value 0 denotes an invalid pixel.

### Unit vectors

The unit vectors are vectors of size 1 that represent each pixel's direction. They are computed from the intrinsic calibration of the camera and the optical model. They are used in combination with the distance image to compute the point cloud.

### JPEG image

This image is the JPEG-encoded RGB image streamed by the 2D imager, when available.

## The confidence image

The confidence image is accessible as part of the data streamed from the O3R device. This image contains information about the validity of each pixel. If a pixel is invalid, the confidence image explains why it has been marked as invalid. The values are as follows:

- 1: CONF\_INVALID - indicates that the pixel is invalid;
- 2: CONF\_SATURATED - the pixel is overexposed/saturated;
- 4: CONF\_BADAMBSYM - the pixel had bad symmetry, probably because of motion (see [symmetry threshold](#));
- 8: CONF\_LOWAMP - amplitude lower than the [minimum amplitude](#), or [distance noise threshold](#) exceeded;
- (16|32): CONF\_EXPINDEX - indicates whether the short, medium or long [exposure](#) is used for this pixel:  $\text{expIndex} = (\text{v} \& \text{CONF_EXPINDEX}) >> 4$  indicates the index of the exposure time used by this pixel where low indices indicate shorter exposures;
- 64: CONF\_INVALID\_RANGE - the pixel is outside of the measurement range;
- 128: CONF\_SUSPECT\_PIXEL - this is a bad pixel on the chip;
- 256: RESERVED
- 512: CONF\_EDGEPIXEL - edge pixels refer to the image edges which are sometimes invalidated by lateral filters;
- 1024: CONF\_UNPLAUSIBLE - pixels remaining after shifting the [offset](#), between the camera and the beginning of the shifted range;
- 2048: CONF\_REFLECTIVITY - the [reflectivity](#) is below the threshold;
- 4096: CONF\_DYNAMIC\_AMPLITUDE - the pixel is probably part of the halo around a very bright object (see the dynamic amplitude threshold (documentation coming soon) and the [stray-light filter](#));
- 16384: CONF\_MIXEDPIXEL - the pixel is a [mixed pixel](#), part of which is measuring the object and the other part the background;
- 32768: CONF\_ISOLATED - an isolated pixel with random amplitude in an area where no amplitude is measured.

This product description contains the Changelog/Release note of the latest hardware/software version.

## 1.2 Getting Started

### 1.2.1 Hardware unboxing

If nobody tampered with your O3R package, you should have following hardware:

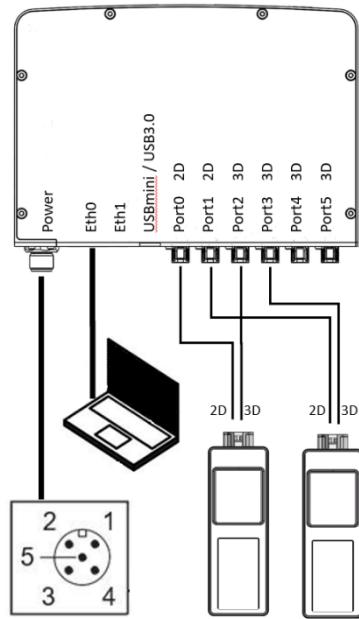
- Camera head (two of them if you ordered the developer kit)



- VPU (Video Processing Unit)
- FPD cables to connect the head(s) to the VPU

You need a strong enough power source: 2.5A and 24V minimum.

1. First, connect the head(s) to the VPU; the only requirement is to connect pairs of same



imager types together, for instance as shown below:

2. Connect power to the VPU - ATTENTION: PIN2 is power! PIN3 is ground
3. Connect the ethernet cable (not included in the package)
4. Wait until you see the ethernet LED flashing before pinging/connecting to the VPU.

That's it about the hardware. Next step: [software installation](#).

## 1.2.2 Software installation instructions

### Network configuration

The default IP address of the VPU is 192.168.0.69. A good first step is to make sure you can connect to the VPU:

```
ping 192.168.0.69
```

If you do not receive an answer from the camera, you can try to adjust your network settings. Your laptop's IP should be within the same subnet's IP-range (for instance, 192.168.0.10).

### Software installation

The ifm3d (ifm3dpy, for python) is the go to library for ifm's 3D cameras. It provides the possibility to easily change parameters, get images and even more. The next steps will cover the installation steps and general usage of ifm3d, and how to unlock the O3R capabilities.

We provide multiple interfaces to use ifm 3D cameras. Follow the links below to find the relevant installation instructions:

- [ifm3dpy](#) (python installation)
- [ifm3d](#) (Linux c++ source build)
- [ifm3d for Windows](#) (Windows c++ source build)

- ifm3d-ros (ifm3d for ROS, source build)
- ifm3d-ros2

This section should help you with the first steps using the O3R.

## 1.3 Parameters

### 1.3.1 Settings Description

#### Acquisition Settings

##### Modes

Variable name	Short description	Min/max values
mode	This parameter designates the measurement range: 2 or 4 meters.	standard_range2m, standard_range4m (default)

Learn more about the modes.

#### Exposure Times

Variable name	Short description	Min/max values
expLong, expShort	These parameters are used to set the exposure times.	1-5000 $\mu$ s

Exposure times are utilized to maximize the number of valid pixels in a scene. The use of multiple exposures permits the camera to operate in “dynamic” environments that require the detection of dark and light objects at both the minimum and maximum ranges.

The proper exposure time for a pixel depends on factors such as the dynamics of the scene and whether the target is moving or stationary. For highly reflective targets or for motion, a short exposure time is best. For targets far away or with low reflectivity, we prefer a high exposure time. As such, it is common that all targets of a scene cannot be properly exposed with a single exposure time. To reduce noise and the number of overexposed/underexposed pixels, we use three exposures for each frame. The “experimental\_high” mode provides two settable exposure times (expLong and expShort) plus a third static exposure (set at 30  $\mu$ s) designed to help detect highly reflective targets in the very near range ( $\sim 1$  m). Note that using a small ratio of exposure times helps reduce noise in transitions regions (where neighboring pixels use different exposure times).

Note: You can find which exposure time is used for each pixel by analyzing the confidence image as detailed [here](#).

## Offset

Variable name	Short description	Min/max values
offset	Shifts the start point of the measured range (see mode)	+/-30 meters

Coded modulation dictates the base range of the camera. (e.g., 0.2 to 2 m). Coded modulation also allows this range to be offset or shifted from its start point. In the example of 0.2–2 m base range, an offset of 1 would lead to a 1.2–3 m range. Continuing this example, an offset of 2 leads to a 2.2–4m range. The offset can be changed frame by frame.

Learn more [here](#).

## Framerate

Variable name	Short description	Min/max values
framerate	Defines the number of frames captured each second	1 to 20 frames per second (FPS)

The FPS highly depends on the applied imager settings (exposure mode and times, filters, etc.). Higher exposure times, for example, negatively impact the overall FPS. The O3R is designed to achieve 20 FPS in the 2 m and 4 m modes, regardless of applied settings. Higher FPS may be achievable by reducing the applied settings.

## Filters

### Maximum Distance Noise

Variable name	Short description	Min/max values
maxDistNoise	Defines the maximum allowable distance noise. Its value represents the standard deviation of the distance value, in meters.	0 to 1. Default 0.05 -> 5 cm noise

Typical filters tend to utilize “broad strokes” when making decisions on which pixel to keep and which to filter. These “broad strokes” may eliminate pixels that are critical to the use case. By utilizing the noise value, we only eliminate pixels with the highest noise value (e.g, ambient light) while preserving the maximum amount of usable data. Applying filters in conjunction with the maximum distance noise filter increases the potential for usable pixels in the scene.

When to change default:

- Lower the max. distance noise value if you are attempting to measure an object with high precision (e.g, box dimensioning).
- Increase the max. distance noise value if it is more important to evaluate all pixels in the scene, regardless of their noise (e.g., obstacle detection).

Learn more [here](#).

## Minimum Amplitude

Variable name	Short description	Min/max values
minAmplitude	Defines the minimum amplitude value required for a pixel to be valid	0 to 1000. Default: 20

A pixel is valid if the energy (amplitude) received is above the defined threshold. The measured amplitude is primarily affected by both the reflectivity of the object and its distance to the camera.

When to change the default:

Lower the default value when the standard targets are known to have low reflectivity (e.g., <10% like matte black targets). A lower amplitude threshold is also valuable when attempting to detect negative obstacles (e.g., stairs). It is recommended to enable a noise filter (temporal or adaptive filter) when lowering the default minimum amplitude.

Learn more [here](#).

## Adaptive Noise Bilateral Filter and Median Filter

Variable name	Short description	Min/max values
anffilterSizeDiv2	Adaptive Noise Bilateral Filter. mask size is $(2*anffilterSizeDiv2+1)^2$ .	0: disable the filter 1: 3x3 2 (default): 5x5 3: 7x7
medianSizeDiv2	Size of the mask for spatial median filtering (the size is $(2*medianSizeDiv2+1)^2$ )	0 (default): disable the filter 1: 3x3 2: 5x5

The adaptive bilateral noise filter reduces distance noise while also preserving object edges. Utilizing a larger number of pixels (e.g., 7x7) in the mask will, in most cases, result in better image quality.

**We recommend that you typically use the bilateral filter because it is more efficient and has a better incorporation of the noise.**

The median filter does not preserve edges as well as the bilateral filter and tends to produce round corners, but being more computationally efficient, could be utilized with “in-motion” use cases (e.g., obstacle detection on mobile robots).

Learn more [here](#)

## Temporal Filter

Variable name	Short description	Min/max values
enableTemporalFilter	Enables the filter	true (default)/false

The temporal filter mitigates distance noise by integrating pixel information over multiple frames. There is no strict limit for the number of frames. Instead, an automatic resetting approach is applied to the pixels.

Although the O3R temporal filter can be used on “in-motion” use cases, it is best suited for static scenes.

Learn more [here](#)

## Mixed Pixel Filtering

Vari-able name	Short description	Min/max values
mixed-Pix-elFil-ter-Mode	Filtering mode (angle or distance)	0: disable the filter, 1 (default): mixed pixel filter is on and uses an angle threshold, 2: mixed pixel filter is on and uses an adaptive delta distance threshold
mixed-Pix-elThresh-oldRad	Threshold given in [rad] for the minimum angle between the surface tangent and the view vector (used if mixedPixelFilterMode=1).	0 to 1.57079. (default 0.15)

Mixed pixels (or “flying pixels”) are pixels that fall partially on a foreground object and partially on an object in the background. Because the physics of indirect ToF do not allow the imager to distinguish partial pixel measurements, the full pixel result is a weighted average distance measurement between the two targets. When viewing the point cloud, these pixels appear “floating”, or not corresponding to any object. The mixed pixel filter removes the mixed pixels from the image.

When to change the default: Mixed pixels fall on the edges of targets. Use cases, such as negative obstacle detection, could take advantage of the additional information provided by these mixed pixels, requiring the filter to be disabled.

Learn more [here](#)

## Symmetry Threshold

Variable name	Short description	Min/max values
enableDynamicSymmetry		true (default)/false
maxSymmetry	Defines the maximum allowed asymmetry for a measured pixel. A pixel with a higher symmetry is discarded.	0 to 1 (default: 0.4)

The raw modulated signal used to perform the distance measurement is designed to be perfectly symmetrical (sent and received). This is true for static applications. If the object is in motion, however, the symmetry of the reflected signal may be altered, leading to “motion blur”. This artifact can be mitigated by allowing “less” symmetry in the measurements.

Note: adjusting this filter for faster motion, or allowing less symmetry, will increase overall distance noise.

[Learn more here](#)

## Stray light

Variable name	Short description	Min/max values
enableStraylight	Turn stray light correction on/off	True (default)/false

Stray light is defined as “unwanted light from the active illumination reaching the imager”. This is typically experienced when there is a very bright object in the FoV. The resulting amplitude of pixels landing on the bright object affect the neighboring “darker” pixels. This is seen as a “halo” around the bright object. This “halo” can affect the measurement of neighboring pixels (even providing a value for pixels where none previously existed). The stray light filter mitigates this physics artifact.

[Learn more here](#)

### 1.3.2 Acquisition settings

#### Modes

#### Description

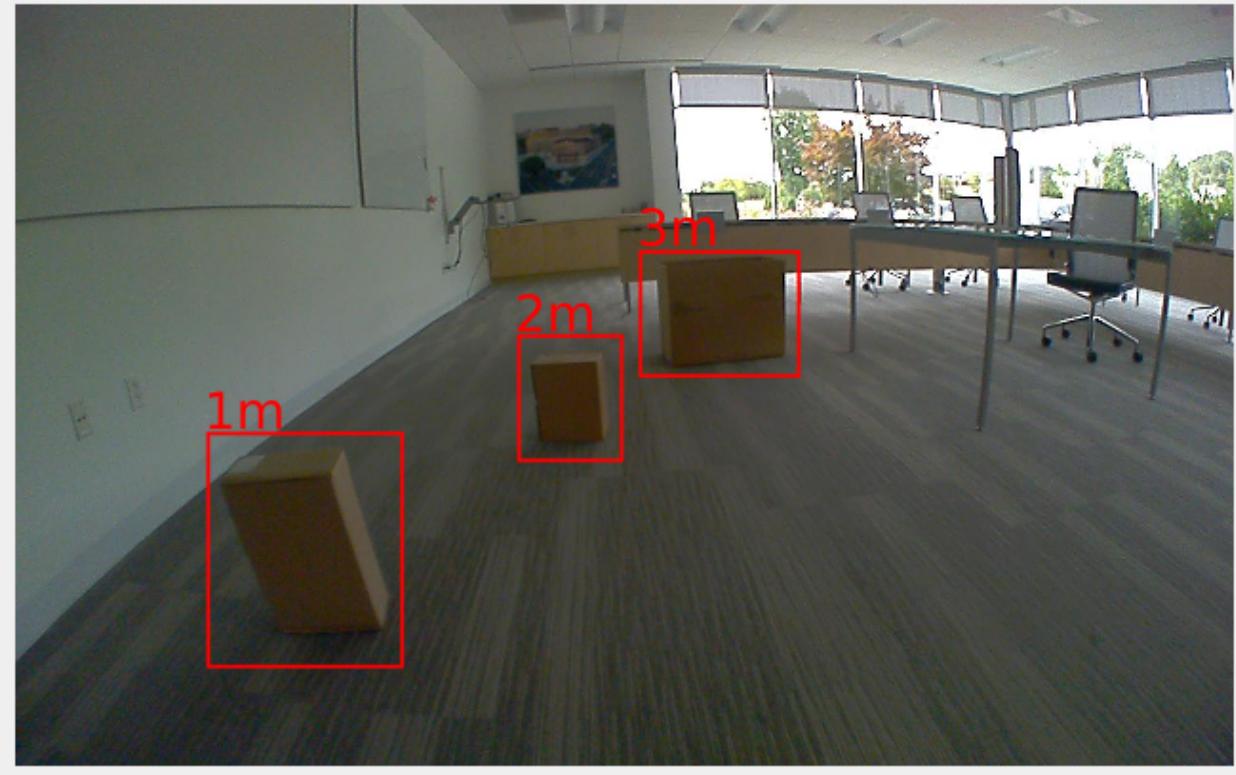
The O3R has the specificity to provide several measurement ranges. A distance measurement is computed only for a subspace of the scene. Elements fully outside of the range are not taken into account and have no impact on the measurement (for instance by causing artifacts like stray-light or Multi-Path Interference(coming soon)).

Note: Objects very close to the beginning or end of the measurement range can still have an impact on the measurement.

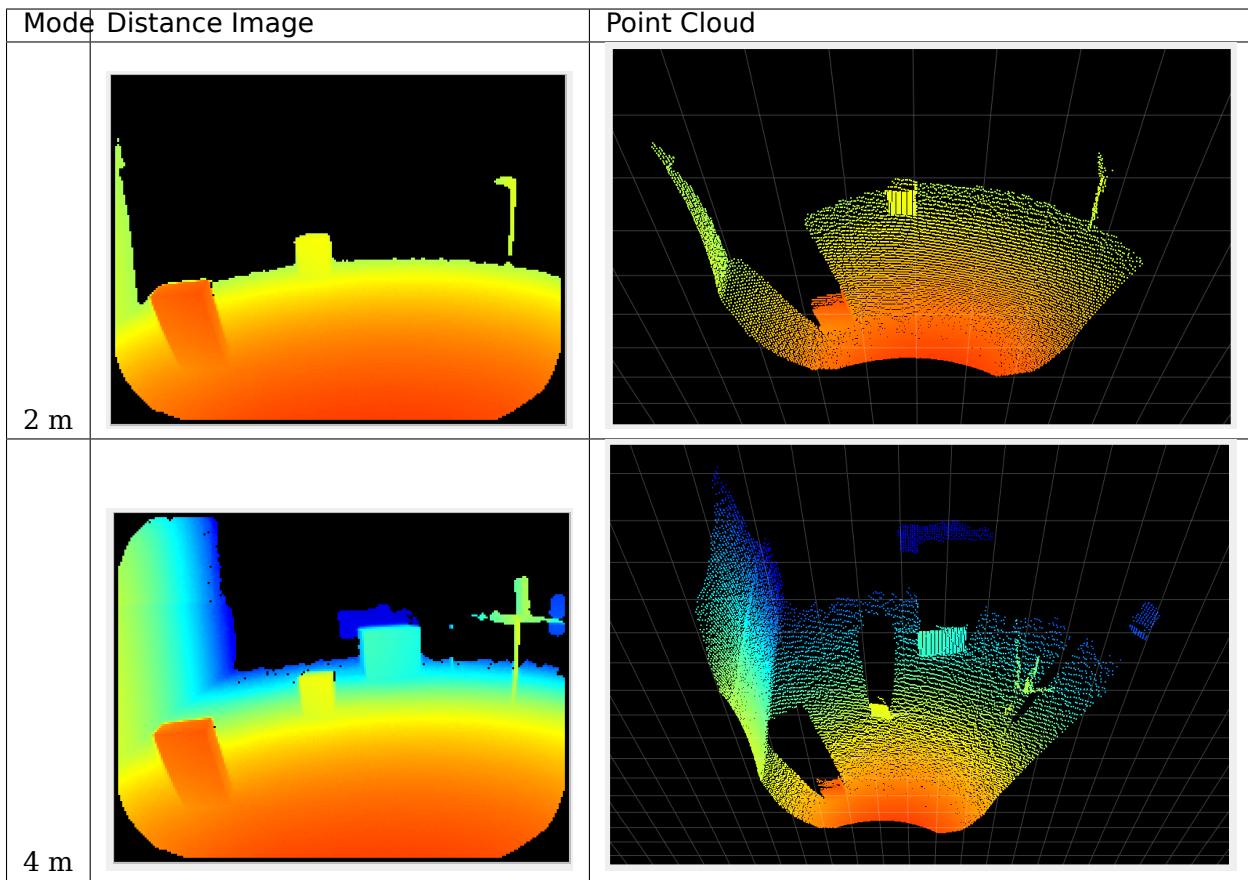
By default, the O3R provides two measurement ranges: two meters and four meters. These ranges use different frequencies to perform the Time-of-Flight (ToF) measurement and therefore show different characteristics, especially in how they are affected by artifacts. The four-meter mode shows higher noise levels than the two-meter mode (around twice as much).

### Example

Let's look at a simple scene: three boxes are placed in front of the camera, one, two, and three meters away (see image below).



The table below shows the computed distance measurement in the distance image view and the point cloud view with the two modes available by default:



The third box, which is three meters away from the camera, is outside of the measurement range when using the two-meter mode; however it is visible when using the four-meter mode.

Note: Using the [offset](#) parameter in combination with the mode is interesting and allows for a lot of flexibility in using the coded modulation ToF technology. We encourage you to investigate strategies using multiple modes in combination with offsets (see our application note (coming soon) on the topic).

## Related Topics

- [Offset](#)

## Offset

### Description

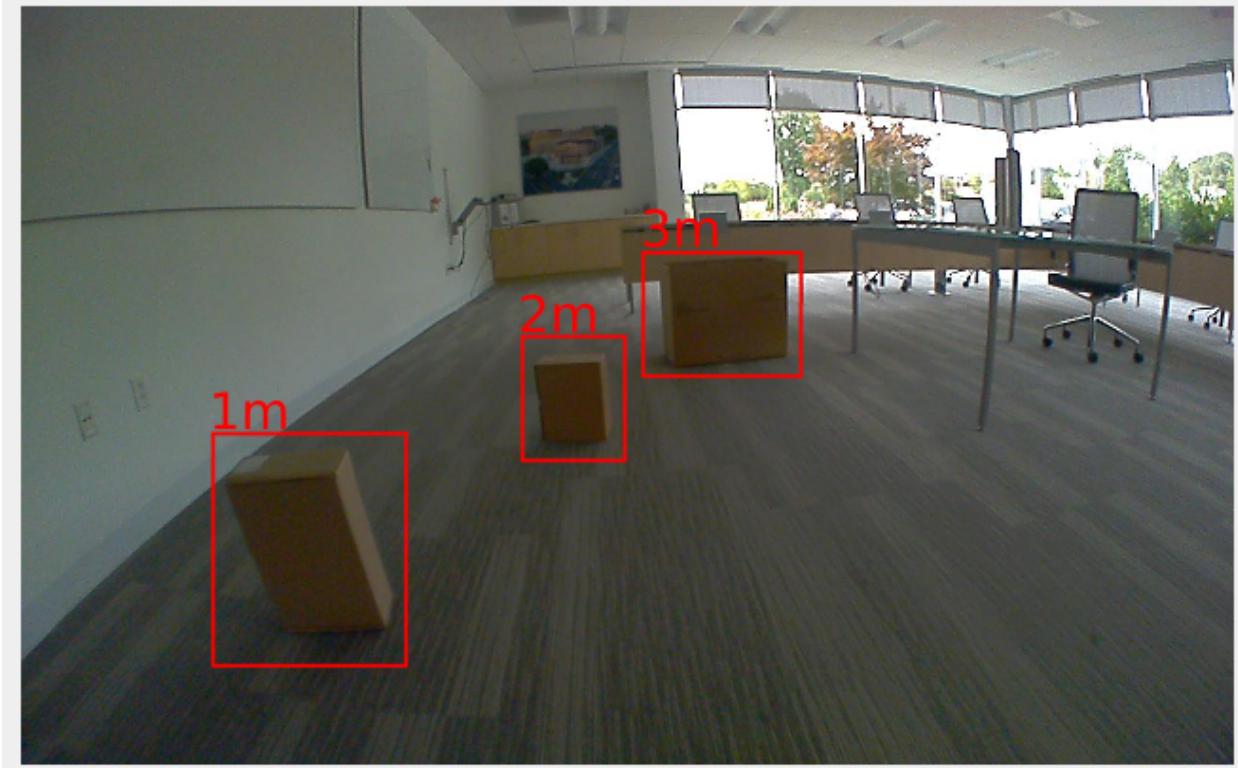
The offset parameter shifts the beginning of the measurement range in space. For instance, when using the 2m mode with an offset of 1m, the O3R will compute distance data for a range between 1 and 3 m from the camera.

Using the offset can allow you to collect distance measurements past the measurement range set by the [mode](#) while taking advantage of the robust point cloud the O3R provides and the specificities of each mode.

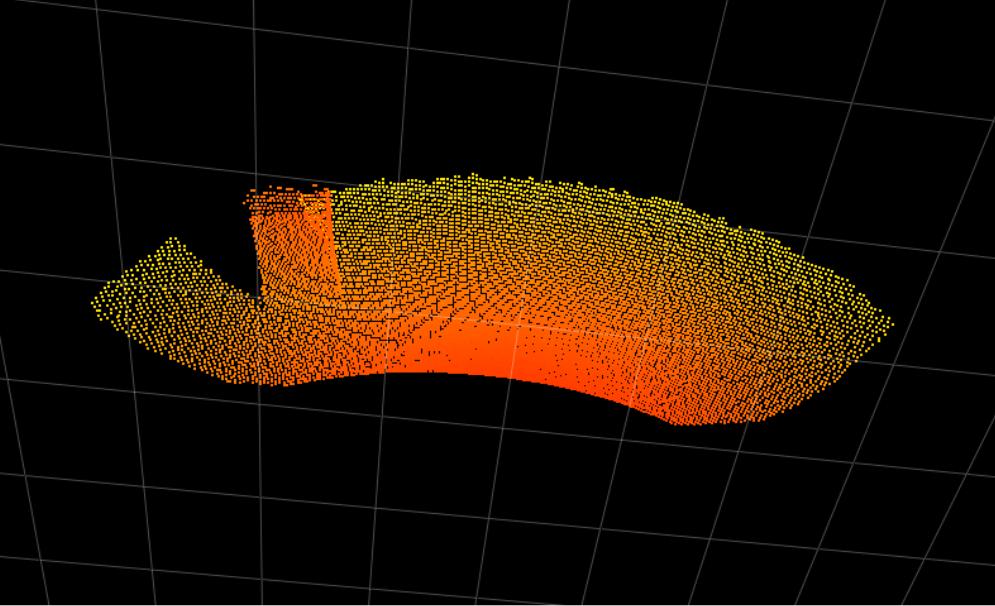
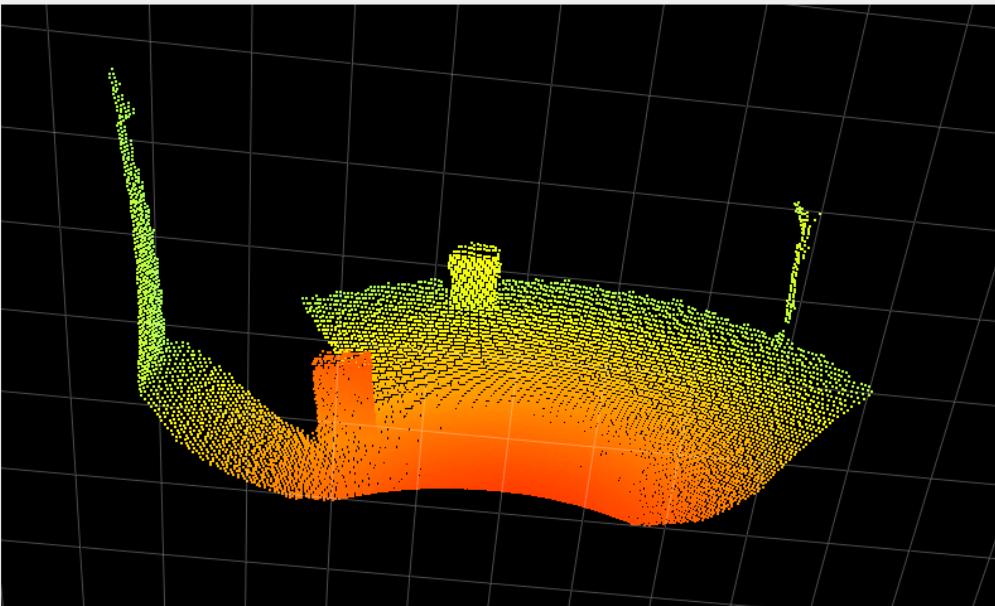
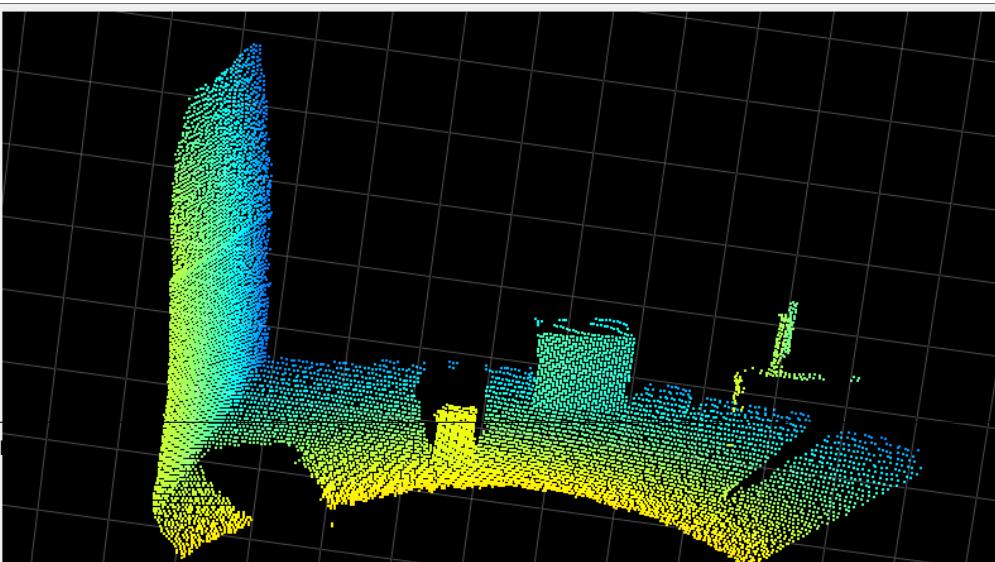
The offset can be set at negative values, which brings the end of the measurement range closer to the camera. This can be useful for mitigating MPI artifacts (coming soon)), for instance, or for avoiding artifacts caused by highly reflective objects (see [stray-light artifacts](#)), by removing the cause of the artifact from the FoV.

### Example

Let's look at the following scene. Three boxes are positioned in front of the camera at about one, two, and three meters away.



We are using the 2m mode, with all the other settings as default. The table below shows the point cloud for multiple values of the offset.

Offset (me- ters)	Point Cloud
-0.5	
0	
1.3. Parameters	

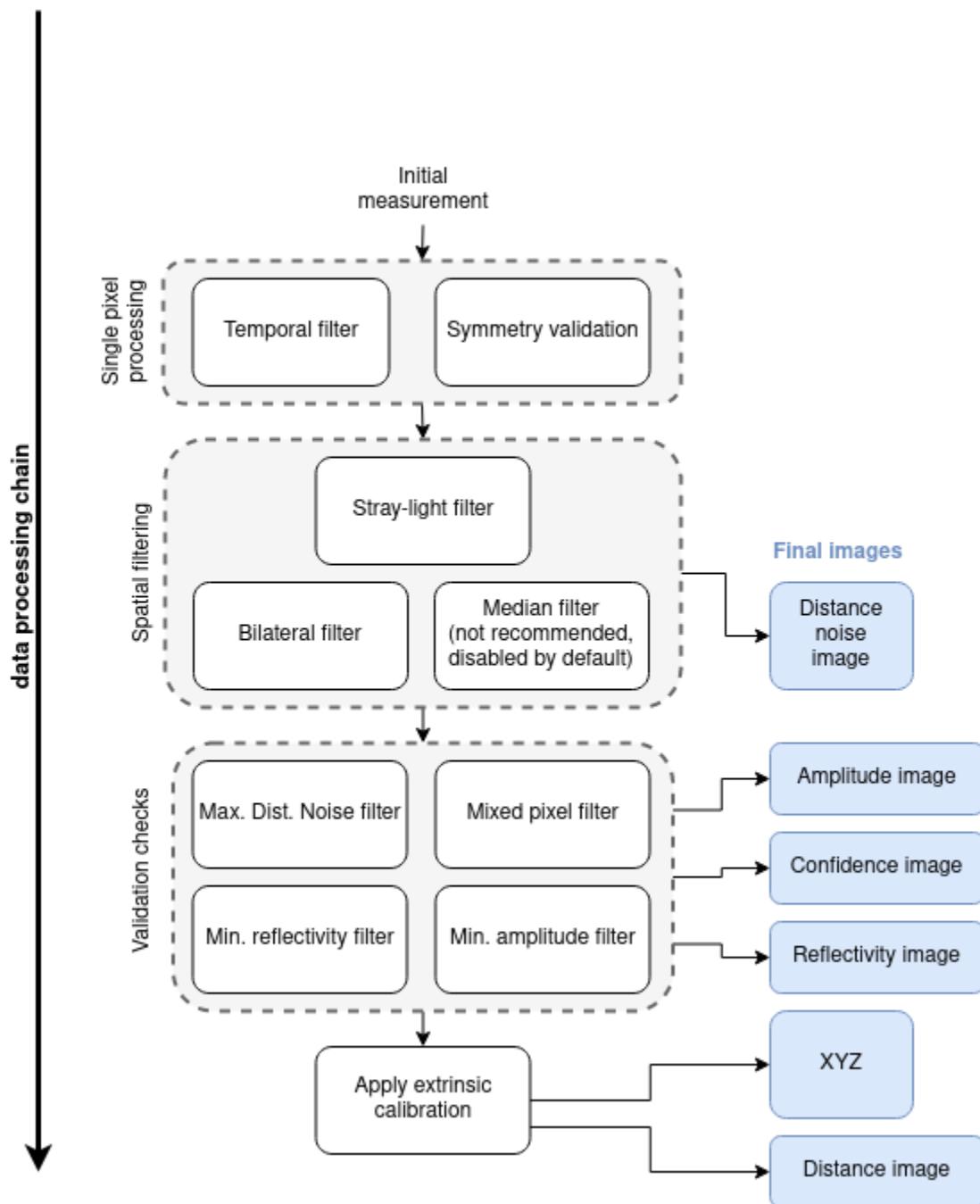
Note: In the last image where the offset is set to 2.5m, we can see that the noise is higher than in the other images. This is due to the distance to the camera, with which the noise increases, and to the fact that the most robust measurement is in the middle of the range, which is from around 3 to 4 m in the case of the last example. The ground in front of the box is outside of the robustness area.

Acquisition parameters change the basic setting on how the camera heads are taking images.

### **1.3.3 Filters**

#### **Filtering Process**

This document details the filtering process applied to the O3R ToF data to produce distance measurements. Each filter listed below can be enabled, disabled or fine-tuned to better fit your application. Please read the detailed documentation for each filter (linked below) for more details.



## Related Topics

- Find an overview of the filters [here](#).
- Find a description of the available images [here](#).

## Maximum Distance Noise

### Abstract

The O3R software estimates distance noise per pixel as well as the distance information per pixel. This distance noise parameter is an estimation of the standard deviation of the radial distance measurement, given in meters. It is based on a noise model built upon the acquired time of flight (ToF) measurements of a single frame. Pixels with a noise value above the threshold `maxDistNoise` are invalidated.

### Description

The O3R camera and software use the ifm ToF technology to measure the distance of objects per pixel. The result are a distance image as and a distance noise image. The distance noise deduction can be interpreted as a standard deviation of the distance measurement in a metric scale. The noise level is dependent on the received signal's amplitude (lower amplitude means greater noise) and on the ambient light level (high ambient light level, especially sunlight, can lead to high noise level).

The distance noise image is processed in the same algorithmic pipeline as the distance image. Any filter applied to the distance image is applied to the distance noise image as well. For example, if filters are activated in the spatial domain (see the [bilateral filter](#)), then they also filter the distance noise image such that the adapted noise image reflects the lowered noise due to lateral filtering.

The parameter `maxDistNoise` is used to invalidate pixels with high noise levels. Higher `maxDistNoise` values will allow noisier pixels to be valid pixels in the point cloud. The maximum allowed value is 1 meter, although we do not recommend using such a high value because the resulting distance measurement will be highly inaccurate in the noisy areas. Low `maxDistNoise` values will result in more noisy pixels being marked as invalid. Values lower than 0.01 meters should always be validated against worst-case expected object and ambient light levels.

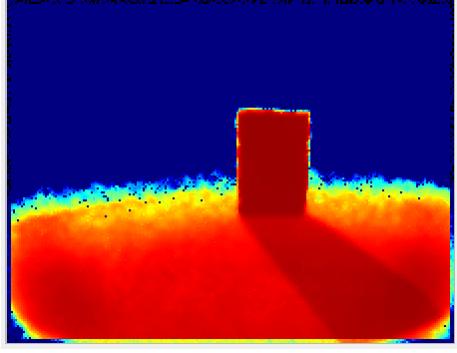
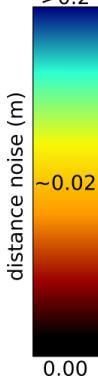
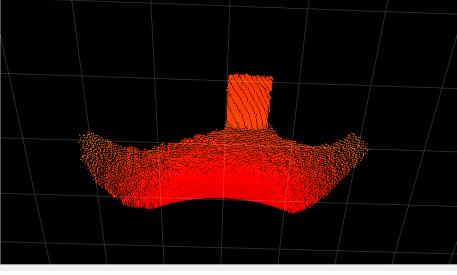
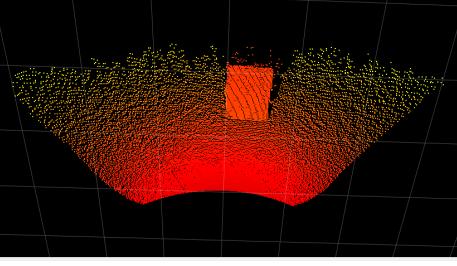
The minimum allowed `maxDistNoise` value is 0.00 meters. This will switch off the validation process based on the estimated distance noise image. The distance noise image is still computed and available to the user.

We suggest you start your experiments with the default values and assess the point cloud quality for your specific use case. The current default value is set to 0.02 meters, which allows for a robust point cloud with negligible noise. For applications where a rich point cloud (i.e., a cloud with more valid pixels) is preferred over accuracy, increasing the noise threshold can be a good idea.

## Example

The following table shows measurements for the same scene with two different distance noise threshold values. The scene consists of a box positioned 1 meter away from the camera, outside in full sunlight. The amount of noise due to the ambient light is high, but it is apparent that we can still get distance values for many pixels by increasing the noise threshold.

Note: For demonstration purposes, we disabled the [temporal filter](#) in these images.

Noise threshold [m]	Distance noise image		Point cloud
0.01			
0.07			

Note: The distance noise image is the same for both noise threshold values. The difference is viewed in the point cloud, where the noise filtering discards more or fewer pixels.

## Related topics

- Bilateral filter (see also the [median filter](#))
- Temporal filter
- Minimum amplitude

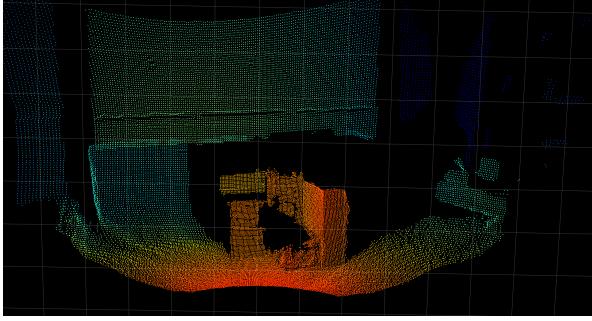
## Minimum Amplitude

### Abstract

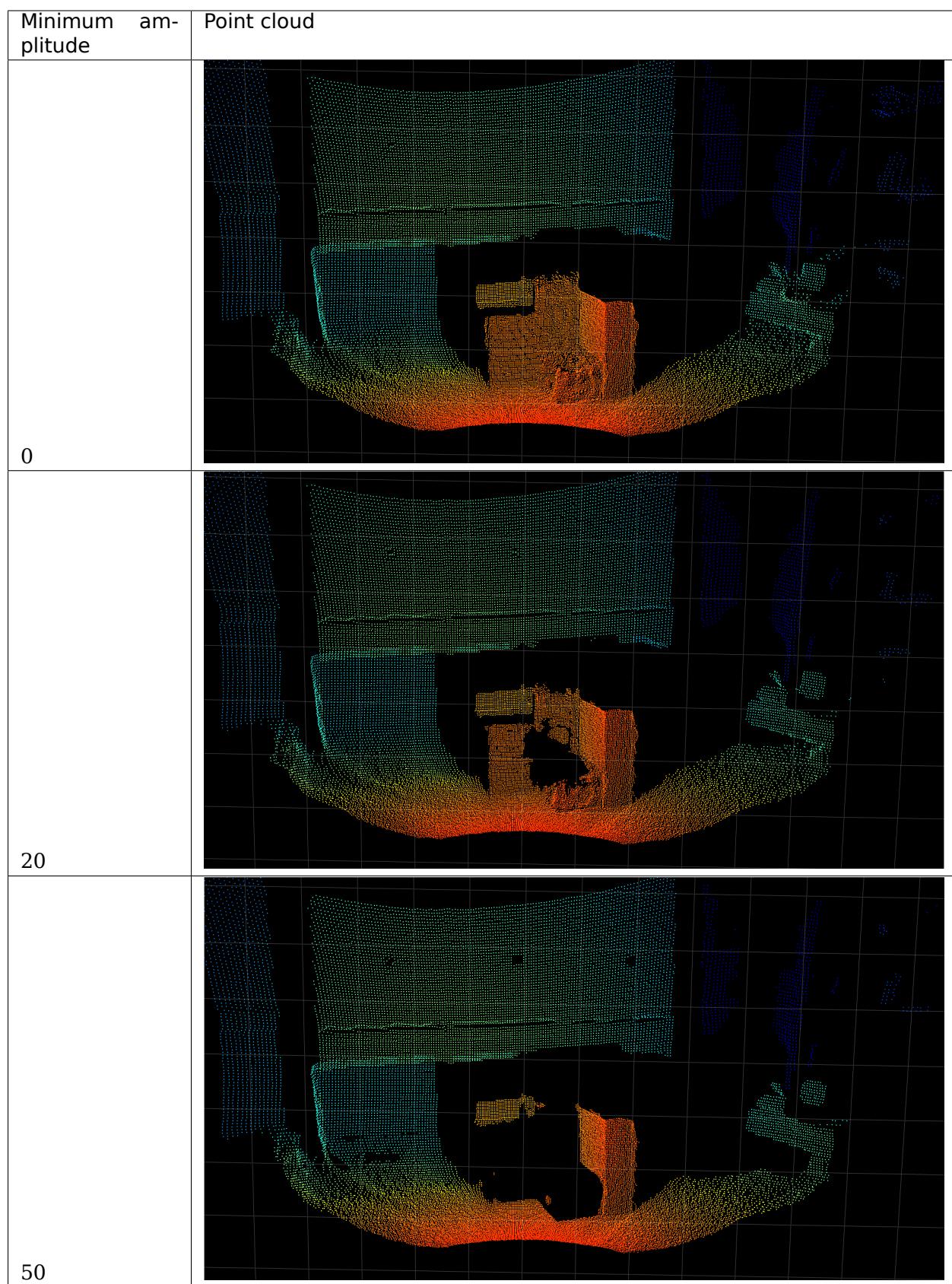
The Minimum amplitude (`diParam.minAmplitude`) parameter invalidates pixels where the amplitude (reflected light) drops below the minimum threshold.

### Description

For each pixel, the amplitude value represents how much light was received by the imager. The `minimum amplitude` parameter provides a threshold that defines when the system should discard low amplitude pixels. The images below show the amplitude image and the point cloud for a scene containing black totes, which are made out of a dark plastic and reflect very little light. Part of the point cloud is missing where the amplitude is below the threshold.

Amplitude image	Point cloud
	

Now, let's see what happens when we change this threshold value. In the table below, we display the same scene measured with different amplitude thresholds. With a value of zero, we can compute the point cloud for the very dark areas. When we increase the threshold to 50, a large part of the point cloud is lost.



In certain cases, such as when black objects are in the field-of-view, changing the default value from 20 to zero can be beneficial because more pixels are valid, leading to a more complete point cloud. Generally speaking, lowering the amplitude leads to more ambient noise and less accuracy in the distance measurement. In this case, we encourage you to test the [filters](#) available with the O3R to mitigate the noise from black objects measurements.

Note: black objects in the visible spectrum are not necessarily black in the near infrared range.

Note: The minimum amplitude threshold is applied to the non-normalized amplitude image. The numerical value of the normalized amplitude image might not correspond to expected values with the set threshold. The normalization factor used in our algorithm is accessible as part of the PCIC output and called `ampNormalizationFactor`.

## Related topics

The minimum amplitude parameter is related to the maximum distance noise parameter: a low amplitude value with a high distance noise value ensures that more pixels are valid but will allow for a noisier measurement, requiring some filtering, for instance with the temporal filter. See the following:

- Distance noise
- Temporal filter
- Bilateral filter

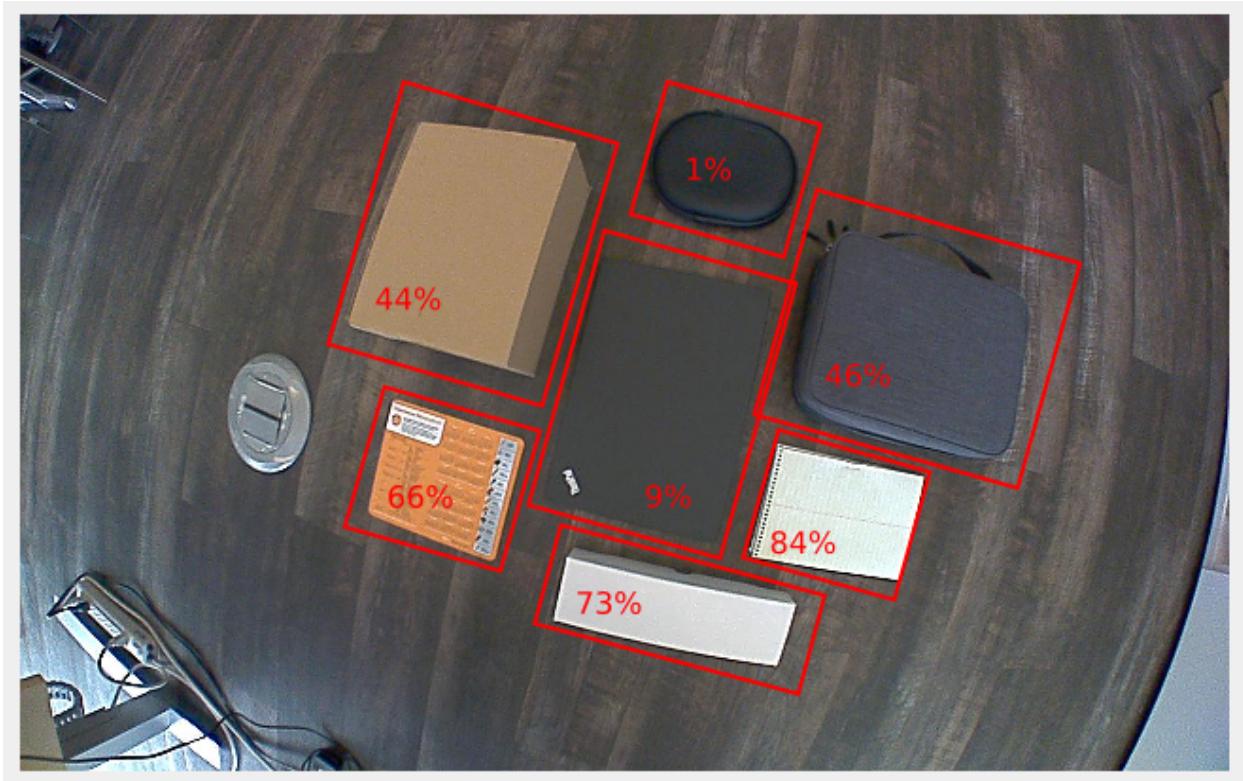
## Minimum Reflectivity

### Description

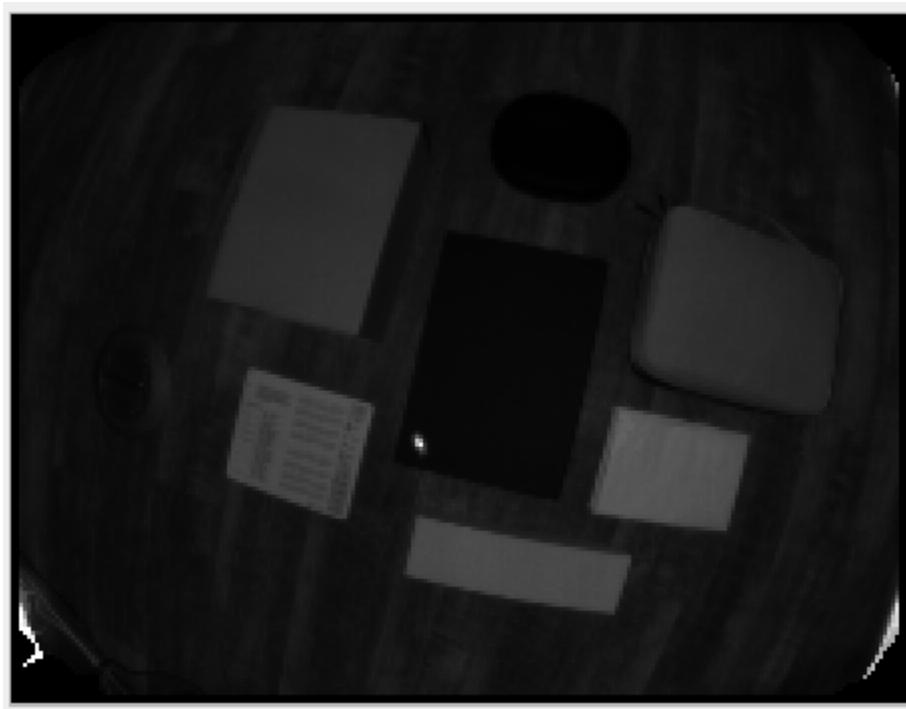
Reflectivity in a near infrared (NIR) spectrum is a characteristic of the material of the object reflecting the light. It depends on its surface material, and the geometric configuration of the scene. The minimum reflectivity (`minReflectivity`) filter invalidates pixels with low amounts of received light (i.e., pixels on objects with a reflectivity below the minimum threshold). The reflectivity is computed from the distance and amplitude images. It can be used to discard unwanted pixels in scenes where an object's reflectivity is known in advance. It can also be used to identify highly reflective objects in a scene; a threshold of 200 would invalidate everything but the retro-reflectors.

### Example

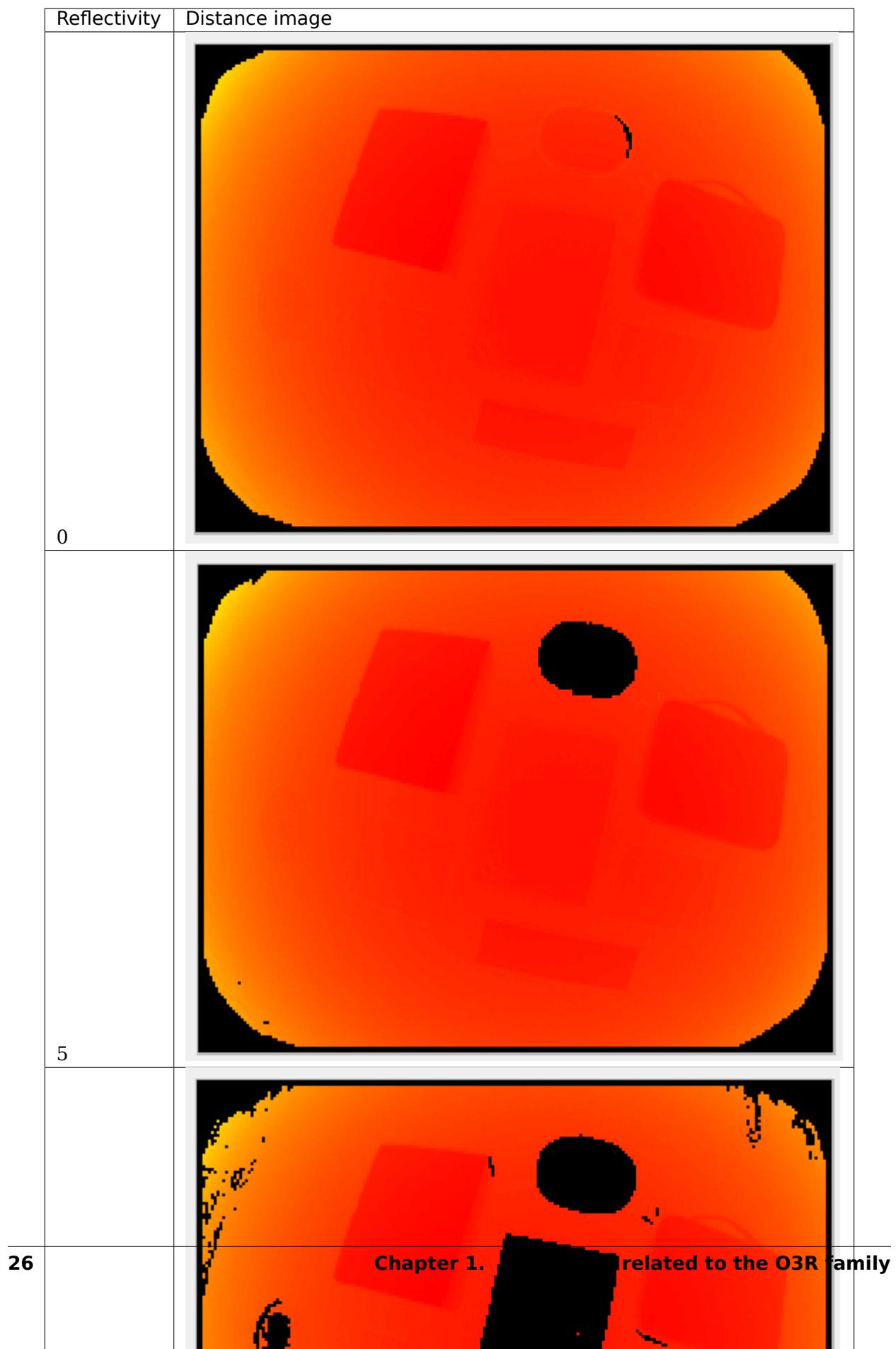
In the image below, we show objects with their measured reflectivity. In this case the reflectivity depends on the material of the object only because their surfaces are seen under the same angle relative to the surface's normal. The estimated reflectivity might vary for the same object seen under varying angles to surfaces normals. Note that this pertains to reflectivity in the NIR spectrum. Some objects might show a higher or lower reflectivity than expected in the visible spectrum.



The corresponding reflectivity image is nothing more than a gray-scale image. Darker shades represent lower reflectivity values:



In the table below, we show the computed distance image for the default settings of the O3R heads with different values for the reflectivity threshold `minReflectivity`:



Note: In the first image, some pixels are missing from the side of the object. This area of the object does not reflect enough light, even when a low reflectivity threshold is allowed.

## Related topics

- Minimum amplitude
- Available images

## Adaptive noise bilateral filter

### Abstract

The O3R software allows for filtering the distance measurement in the spatial domain. The spatial domain of a 3D image can be thought of as the local neighborhood of a pixel, that is, the neighboring pixels X-, Y-, and Z-coordinates. Radial distance information for a pixel is combined with its neighbors' information to form a new distance image with reduced noise.

The bilateral filter is the preferred spatial filter and is enabled by default. It can be applied with different filter mask sizes, which can be set via the parameter `anffFilterSizeDiv2`. Larger filter mask sizes allow for stronger noise reduction.

### Description

This distance bilateral filter is, in its concept, highly similar to a [bilateral filter applied to RGB 2D images](#). A bilateral filter is a nonlinear edge-preserving smoothing filter. The idea is to replace the pixel value with a weighted average of the information from nearby pixels. The weighting is a combination of the spatial kernel and the range kernel. The O3R implementation additionally incorporates the distance noise estimation for calculating the filter weights.

Note: The weighted average is computed by convolution over the spatial domain. The convolution of the original image and the filter mask returns an image reduced by half the filter size at each image border; that is, with a filter mask of 7x7 pixels, the image is stripped of 7 pixels vertically and horizontally (these pixels are marked as invalid).

The bilateral filter is controlled by the parameter `anffFilterSizeDiv2` (turn it off with `anffFilterSizeDiv2 = 0`). `anffFilterSizeDiv2 = 3` sets the filter mask size to a local 7x7 pixel neighborhood.

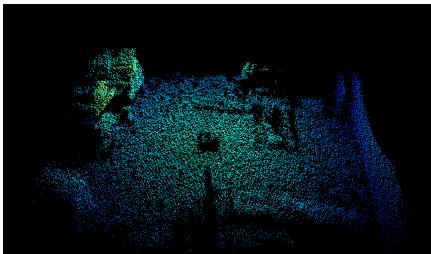
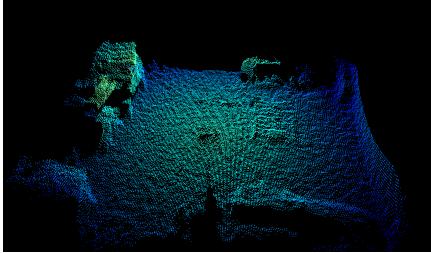
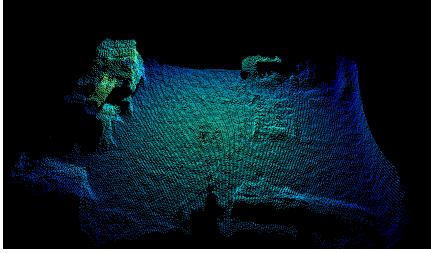
Note: The bilateral filter is preferred over the [median filter](#) because it preserves edge and corner information better (the median filter tends to round corners). It is also possible to apply the bilateral filter with larger filter masks (up to 7x7 pixel masks) compared to 5x5 pixel masks for the median filter. The size of the mask defines how many neighboring pixels are considered when computing a pixel's value.

Invalid pixels will be ignored during the filtering process and therefore have no impact on the surrounding pixels. Invalid pixels will remain invalid after the filtering.

**Example**

The following pictures give an overview of the capabilities of spatial filtering with the bilateral filter for different filter mask sizes. All other filters ([temporal](#) and [median](#)) are deactivated for the purpose of illustrating the bilateral filter's effect. The maximum allowed distance noise is set at 0.2 m for all images. Note that for maximum distance noise values below 0.2 m the point cloud becomes extremely sparse for smaller filter mask sizes (not shown in the following).

The scene shows a view of our lab, containing various typical objects including a black tote in the center of the room. It is a static scene, which makes it simpler to illustrate the filter's effect, but these settings (active bilateral filter and inactive temporal filter) are typical for scenes involving motion. Have a look specifically at the distance noise images in the following table. We can see that the distance noise greatly reduces as the filter mask size increases (the color red denotes negligible noise, whereas blue represents noise of around 1 cm and above).

Bilat- eral filter mask size	Point cloud	Distance (top left), amplitude (top right), distance noise (bottom left), and reflectivity images (bottom right)	
0 (filter dis- abled)		   	
1 (3x3 mask size)		   	
2 (5x5 mask size)		   	
<b>1.3. Parame</b>		   	<b>29</b>

Note: distance information for the black tote in the middle of the image remains extremely hard to compute even with a strong lateral filtering. For better handling of dark objects, have a look at the [minimum amplitude](#), [maximum distance noise](#), and [temporal filter](#)).

## Scenes involving motion

The spatial filtering can be performed in scenes where motion is present: only the parts of the images that are not affected by movement will be filtered. This differentiation is possible because the detection of motion is performed before the spatial filter in the processing pipeline. It is perfectly fine and encouraged to use large filter mask sizes.

Note: this is not true for [temporal filtering](#), which is not best suited to in-motion cases.

## Related topics

- Median filter
- Temporal filter

## (Spatial) Median Filter

### Abstract

The O3R software supports two spatial filters for improving the distance measurements: the median filter and the [bilateral filter](#). **We recommend using the bilateral filter in most cases instead of the median filter because the median filter can have undesirable side effects.**

### Description

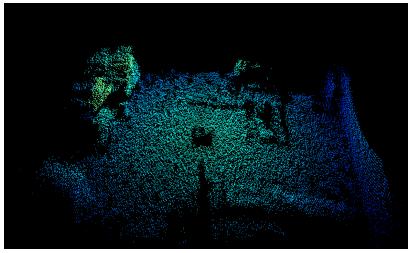
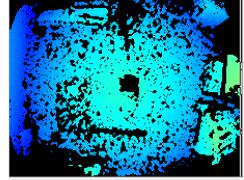
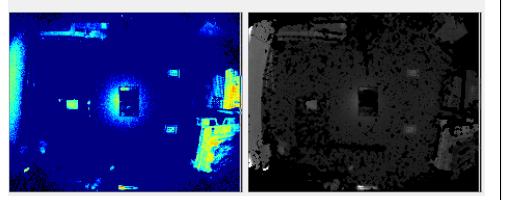
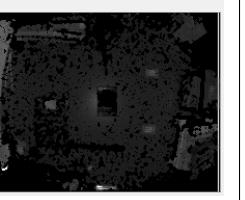
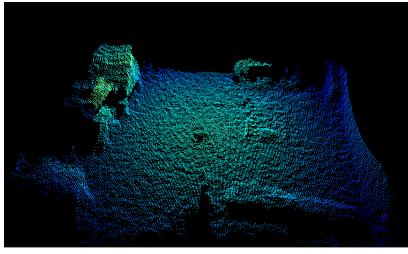
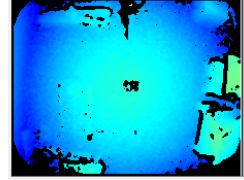
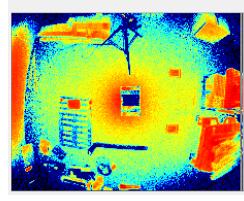
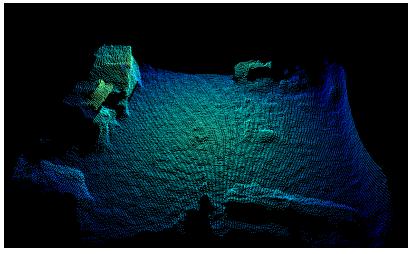
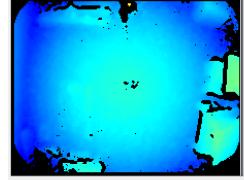
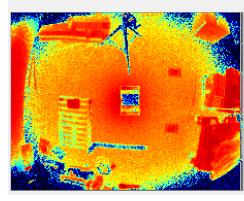
The median filter is conceptually very similar to a [median filter applied to RGB 2D images](#). A median filter is a nonlinear, edge-preserving smoothing filter. It can be thought of as a filter that replaces the value per pixel with the median value of neighboring pixels. The computation is achieved by sliding the filter mask in the spatial domain until it covers the whole image. This filtering technique is robust (i.e., not affected by outliers) and reduces noise while keeping edge information intact. The median filter is applied to the distance image. The distance noise is lowered to heuristically reflect the new noise in the distance image.

The median filter is controlled by the parameter `medianSizeDiv2`; turn it off with `medianSizeDiv2 = 0`. `medianSizeDiv2 = 1` sets the filter mask size to a size of 3 x 3 pixels. `medianSizeDiv2 = 2` is the highest allowed value. It represents a filter mask size of 5 x 5 pixels. Using larger filter mask sizes combines more pixels' distance measurements into the filterer value. The filter's effect will be stronger, resulting in a smoother image.

Note: Invalid pixels are ignored during the filtering process and therefore have no impact on their surrounding pixels. Invalid pixels remain invalid after the filtering.

### Example

Below are images of the same scene with different settings for the median filter. Look more specifically at the distance noise image that shows the amount of noise in the scene—the larger the filter mask size, the lower the noise level. The color red corresponds to negligible noise levels and blue to noise around 1 cm and above. See the [bilateral filter](#) example for comparison with the same scene.

Filter mask size median- SizeDiv2	Point cloud	Distance (top left), amplitude (top right), distance noise (bottom left), and reflectivity (bottom right) images	
0 (filter deactivated)		   	
1 (3 x 3 mask size)		   	
2 (5 x 5 mask size)		   	

## Bilateral vs. median filtering

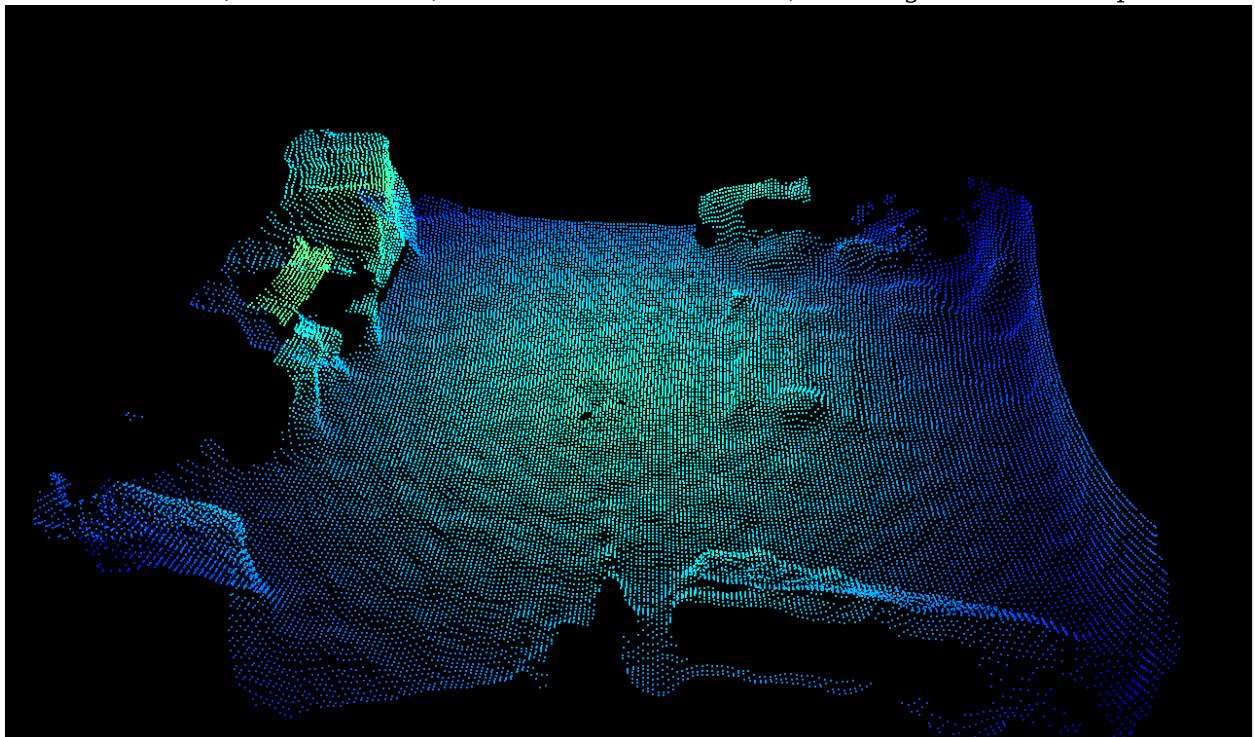
### Disadvantages of the median filter

The median filter is not our spatial filter of choice for two reasons: it does not preserve corners of objects as well as the bilateral filter, and it uses a heuristic method for dealing with the distance noise image. Moreover, the median filter can introduce a bias in the distance image (locally) in some cases, an effect that is not present in the bilateral filter. We recommend using the [bilateral filter](#) in most cases.

### Bilateral and median filters combined

A combination of both spatial filters is rarely required, and we recommend increasing the filter mask size as a first step. However, if the filtering is not strong enough, then one can use both the bilateral and median filters at the same time. This will further reduce local noise levels but can result in bias in larger noise patterns.

To give you an idea, the image below shows the effect of combined bilateral (`anfFilterSizeDiv2=3`) and median (`medianFilterSizeDiv2=2`) filtering for the example



scene.

## Related topics

- [Bilateral filter](#)
- [Temporal filter](#)

## Temporal Filter

### Abstract

The temporal filter filters the data over—you guessed it—time. Each measurement per pixel over several frames is used to produce the filtered value, reducing (distance) noise per pixel. This filter is best suited for static scenes because the objects in the scene are in the same relative positions over multiple frames.

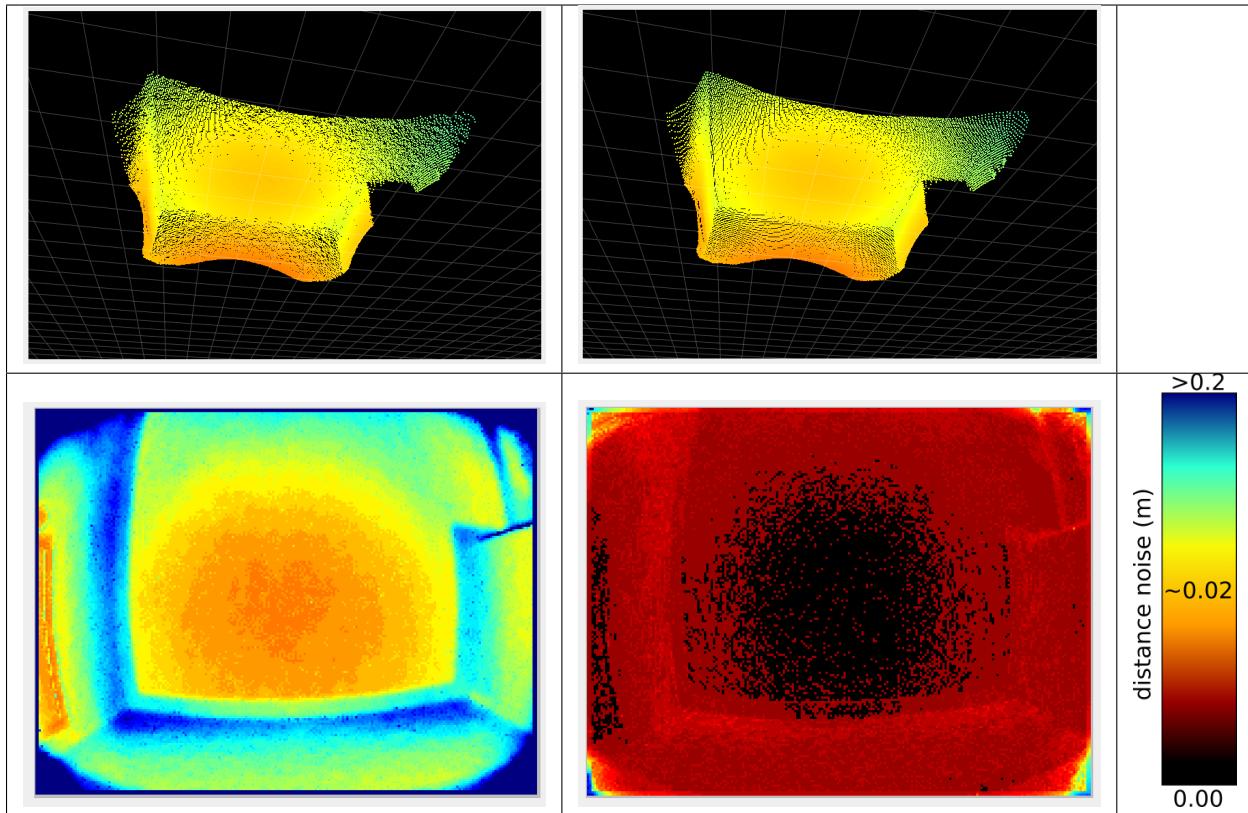
### Description

The temporal filter affects all images—distance image, point cloud, and so on—by reducing the noise. The filtered value for each pixel at a certain time is computed by integrating information over multiple frames. There is no strict limit on how many frames are taken into account for filtering; instead, the filter is automatically reset when necessary, such as when motion is detected.

### Examples

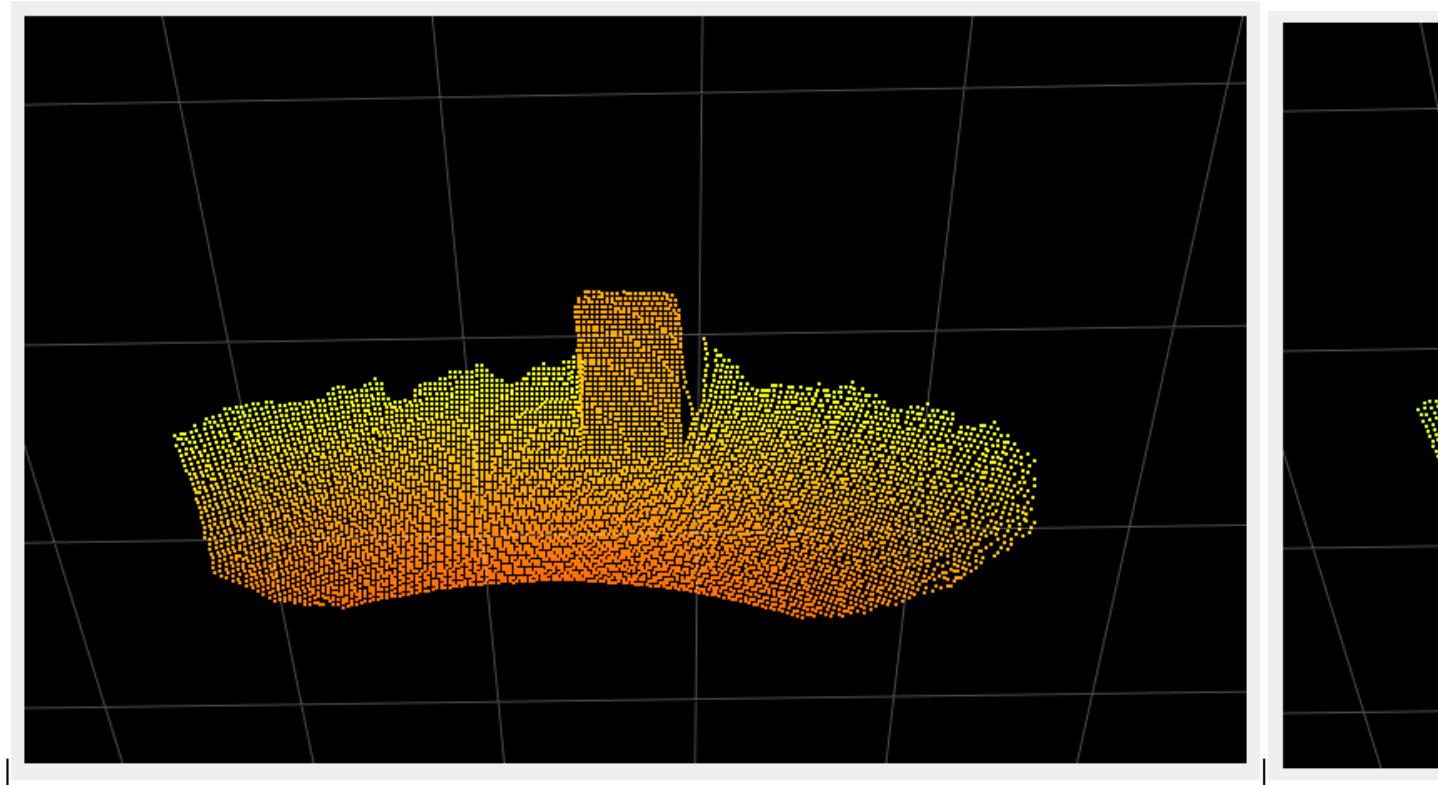
#### Reducing Noise

The primary role of the temporal filter is to reduce noise. The following images show a before/after of a scene measured without (left images) and with (right images) the temporal filter. The two images in the second row represent the distance noise, with the color black representing a negligible amount of noise and blue a noise of around 2 cm. We can see that the noise is greatly reduced by the use of the temporal filter. It is even more visible “live”. Try it!



### Recovering Lost Pixels

Because the filter estimates pixel values over time, a positive side effect is that it gathers more data overall. Certain pixels might reflect too little light because of their distance to the camera or their material, which causes them to be discarded during the filtering process (by the [spatial filter](#), for instance, or the [minimum amplitude filter](#)). However, distance data for these pixels can potentially be computed by collecting light over multiple frames. In the following two images, we compare the same scene without (left image) and with (right image) the filter. We can see that a section of the floor (around 10 cm) at the end of the range (the point at which the pixels return the least amount of light) is not visible without the temporal filter. Note that this section of the floor could also possibly be recovered using the [distance noise](#) with higher values for the distance noise threshold, with the disadvantage of increasing the overall noise.



## Related settings

- Distance noise
- Minimum amplitude
- Spatial filter

## Mixed Pixel Filter

### Abstract

We call mixed pixels pixels resulting from a mixed signal from foreground and background planes (typically, the pixel “lands” partially on an object and partially on its background). Such pixels don’t represent the distance measurement to either object and lie somewhere in between (they appear to be flying, and we sometimes refer to them as flying pixels). The mixed-pixel filter invalidates these pixels. The `mixedPixelFilterMode` setting defines whether this filter is activated and which validation methods is used. `mixedPixelFilterMode = 1` switches to angle validation check (adjust it with `mixedPixelThresholdRad`). `mixedPixelFilterMode = 2` switches to distance based validation check (this mode is inherited from previous algorithm versions and will most likely be deprecated in the future). `mixedPixelFilterMode = 0` switches the filter off completely.

**We recommend either disabling the filter (more precision on objects’ edges) or using the angle based validation method (`mixedPixelFilterMode = 1`) to remove pixels between objects and their backgrounds.**

## Description

The `mixedPixelFilterMode` controls two different methods for invalidation mixed pixels.

### Angle based validation method

The angle based mixed pixel filtering (`mixedPixelFilterMode = 1`) is based on the idea of estimating, for each pixel, the angle between the optical and an approximate tangent plane on the object (at this exact pixel coordinate). If the angle difference is larger than the allowed angle threshold, the pixel is invalidated. The angle threshold of this mode is controlled by the parameter `mixedPixelThresholdRad` (angle in radians).

### Distance based validation method (will be deprecated)

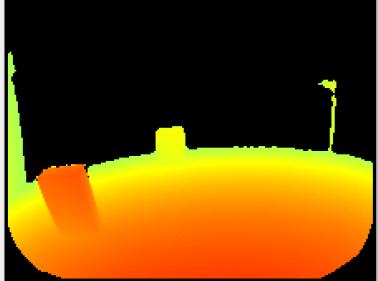
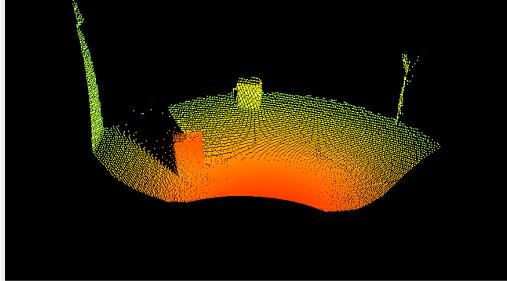
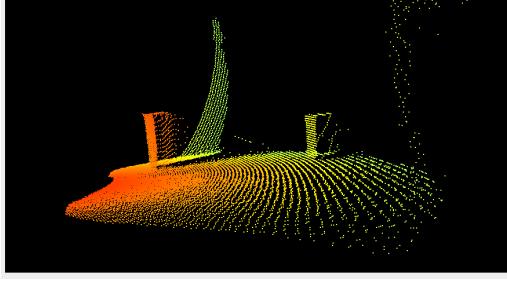
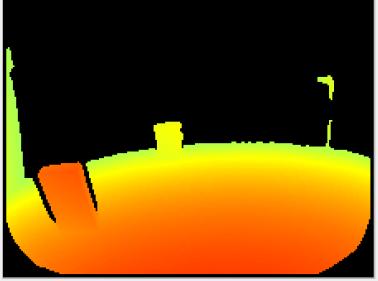
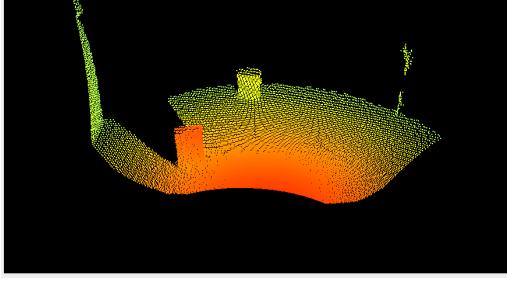
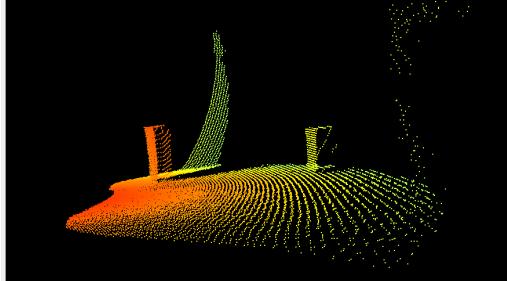
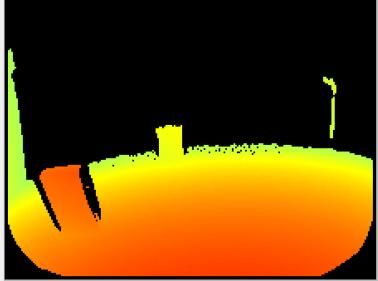
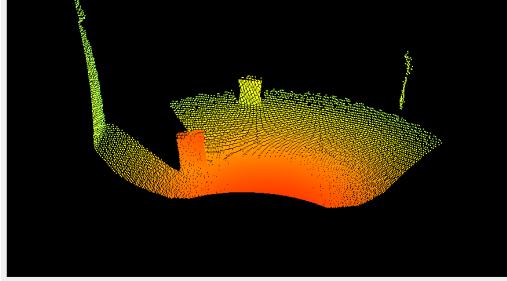
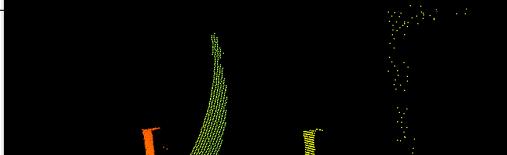
The second version of the mixed pixel (`mixedPixelFilterMode = 2`) filter is centered around the idea of comparing distances in the local neighborhood of a pixel. The distance of the pixel is compared in horizontal and vertical direction against its neighboring pixels' distance values. If the distance differences are outside a threshold (set internally), the pixel is invalidated.

## Examples

### Different angle threshold values

To show the impact of adjusting the mixed pixel filter with the `mixedPixelThresholdRad`, we show a scene where two boxes are placed in front of the camera, at around one and two meters. The table below shows the distance image and the point cloud with the filter inactive and filter in the angle mode with different angle thresholds:

Note: settings `mixedPixelThresholdRad = 0` is equivalent to turning the filter off.

Value of mixedPixelThreshold-dRad	Distance image	Point cloud
0 (equivalent to mixedPixelFilterMode = 0)		
		
0.15 (default)		
		
0.3		
<b>1.3. Parameters</b>		 37

We can see that using higher values for the `mixedPixelThresholdRad` invalidates more pixels. We typically recommend deactivating the filter or using small values for the threshold.

## Symmetry Threshold

### Abstract

The symmetry threshold `maxSymmetry` is used for filtering motion artifacts. Increasing the threshold value leads to more valid pixels around moving objects but also increases the chance of computing incorrect distance measurements for some pixels. Decreasing the threshold will result in invalidation of more pixels because of their estimated symmetry value. In cases with a high ambient noise level, the dynamic symmetry filter should be enabled (with the parameter `enableDynamicSymmetry`) to ensure pixels are not invalidated due to ambient noise.

### Description

The O3R camera heads use the ifm ToF (Time of Flight) technology for measuring the distance to objects. To calculate one single point cloud image, the system takes several independent image frames. These images are correlated over time. This correlation is represented as symmetry value. It can be thought of as the four modulated signals used for performing the raw measurement, being more or less symmetrical to one another.

For low symmetry threshold values, only pixels for which the correlation images are highly symmetrical (i.e., with no or few motion artifacts) are valid. Because of inherent noise, a perfect symmetry is never possible even for static scenes. Increasing the symmetry threshold validates pixels with higher symmetry values, including noisy pixels and potential motion artifacts.

In a high-noise environment, the sensor noise might be propagated to the symmetry image and most of the pixels invalidated because of the symmetry threshold. In these cases, the dynamic symmetry should be activated. The dynamic symmetry ensures that the symmetry threshold of a pixel is at least high enough to prevent invalidation due to sensor noise. It can be thought of as differentiating motion artifacts from ambient noise in the scene.

If the dynamic symmetry is enabled, each pixel gets an individual symmetry value, which is either the `maxSymmetry` setting or the expected symmetry (computed internally) due to noise, whichever one is greater.

### Example

COMING SOON...

Disclaimer: As a vendor of industrial equipment, we always try to mitigate physics artifacts to the best of our abilities. The default settings are chosen with this goal in mind: providing the best experience for most cases. However, the variety of scenes that mobile robots and other applications can encounter makes a “one-fits-all” configuration impossible. With this in mind, we present the outliers in our documentation, challenging but common cases that might require fine-tuning of the camera configuration.

## Stray Light Filter

Scenes including highly reflective objects are common in robotics and industrial use cases. These scenes can present challenges because reflectors introduce an artifact known in optical systems as stray light. In this document, we focus on this phenomenon and present the stray light filter available with the O3R. We analyze some challenging cases and give hints on how to handle specific applications.

NOTE: the scenes presented here are recreated with the intention of highlighting the specific challenges of mobile robots environments here in our office. We detail our experience of trying to “fool” the system, but we are well aware of the limitations of this approach. We always want to hear from firsthand experience, and we gladly welcome your feedback.

## Abstract

Stray light is a phenomenon that exists in any optical system where light reflected from very bright objects is bouncing around in the lens, impacting the measurement. Two main effects of stray light are noticed in 3D point clouds: the creation of a halo around the bright objects, and the apparition of “ghost” pixels in the close range. To mitigate these artifacts, a stray light filter is available (turn it on with `diParam.enableStrayLight`). This filter can be fine-tuned to be more or less conservative, depending on whether the application requires high accuracy or a rich point cloud (adjust `diParam.excessiveCorrectionThreshDist` and `diParam.excessiveCorrectionThreshAmp`).

## Table of Contents:

1. The stray light phenomenon + filter intro
  1. Halo: before/after
  2. Ghost pixels: before/after
2. Fine-tuning of the stray light
3. Conclusion

## The stray light phenomenon

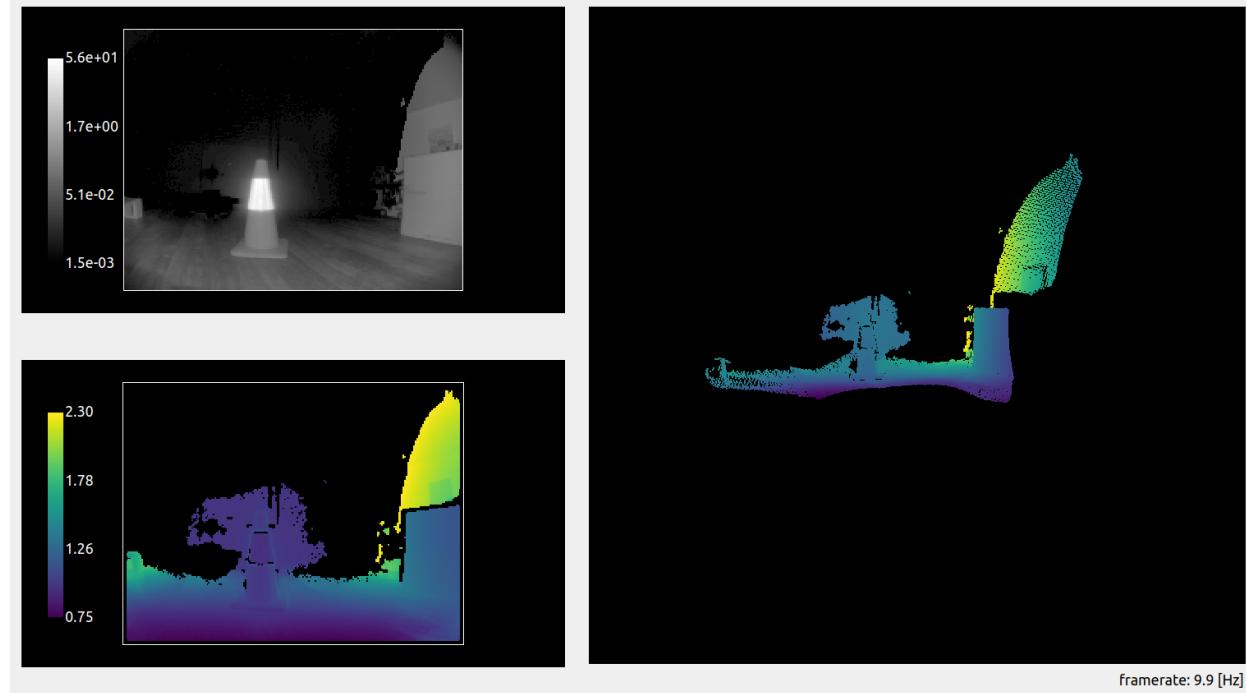
Stray light designates any unwanted light reaching the optical lens of the camera. This light can be reflected light from an object within the field of view or emitted by an object outside the FoV. Stray light exists in any non-perfect optical systems, where an excessive amount of light is reflected on the internal parts of the system (within the lens or other camera components) and eventually reaches a pixel of the imager, interfering with the measurement. Common objects found in warehouses and other industrial environments, like reflective cones or safety vests, are sources of stray light interference.

A typical effect of stray light is to cause a halo of pixels around the reflective object, affecting the measurement of low-signal pixels in the area. But stray light can also affect pixels not in the direct vicinity of the reflector, creating ghosts pixels, typically in the close range, which can make the scene hard to analyze.

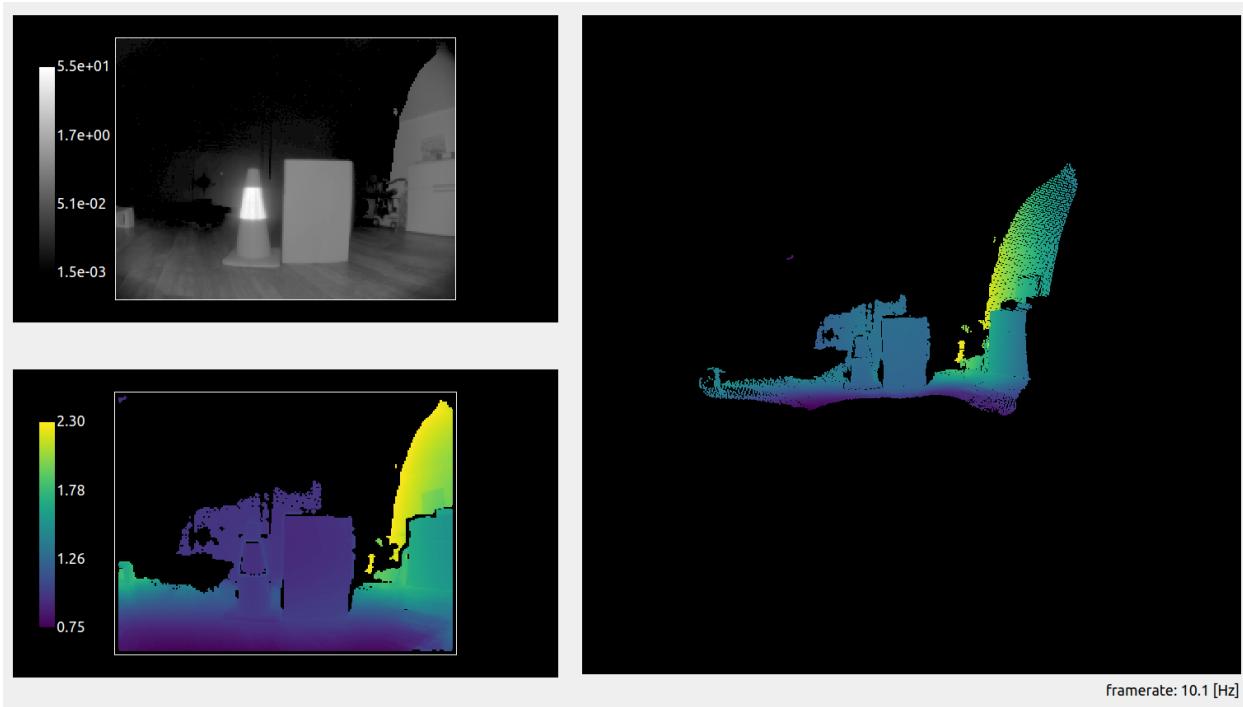
Let’s look at some concrete examples of stray light artifacts. For the purpose of demonstration, we have disabled the built-in O3R stray light filter.

**First case: The stray light halo**

A circulation cone with a reflective band is positioned 1 m in front of the camera. We can observe a stray light halo around the cone: pixels are measured where there should not be anything (the background is out of range in this case).



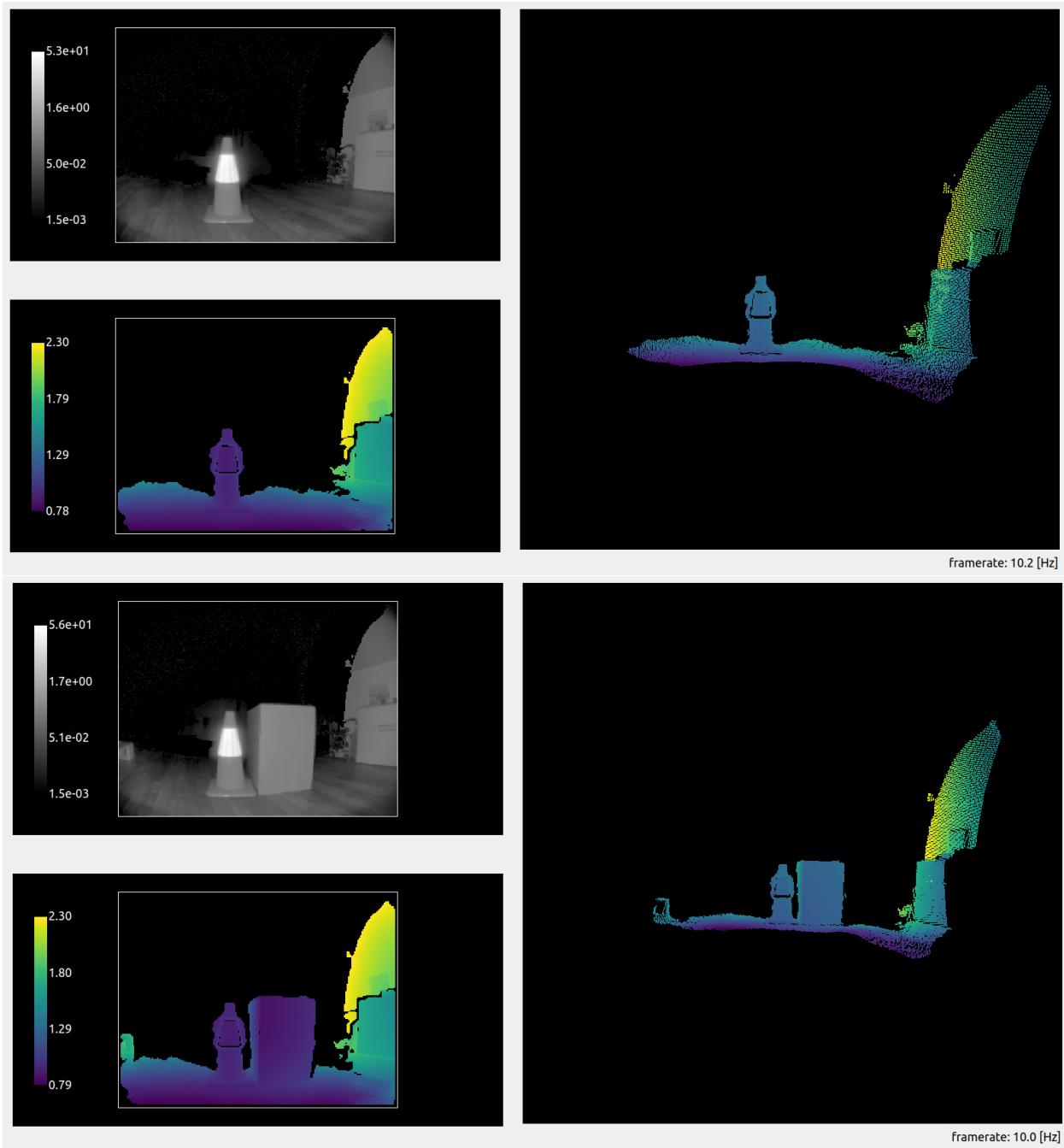
A cardboard box is positioned next to the cone, at the same distance from the camera. The halo is still there and affecting part of the background pixels. However, the measurement of the actual box is accurate and is not affected by the reflective object next to it. Stray light affects the low-signal pixels in the scene more strongly, and, in this case, it does not affect the measurement of the box, which reflects enough light.



### Handling stray light halos

The O3R camera comes with a built-in stray light filter that mitigates stray light artifacts. This filter improves the measurements by correcting the undesired effects of the optical system. Additionally, pixels that are overly affected by stray light are filtered according to set distance and amplitude thresholds (the default distance threshold is set to 8 cm: if stray light affects a pixel measurement more than 8 cm, this pixel will be invalidated).

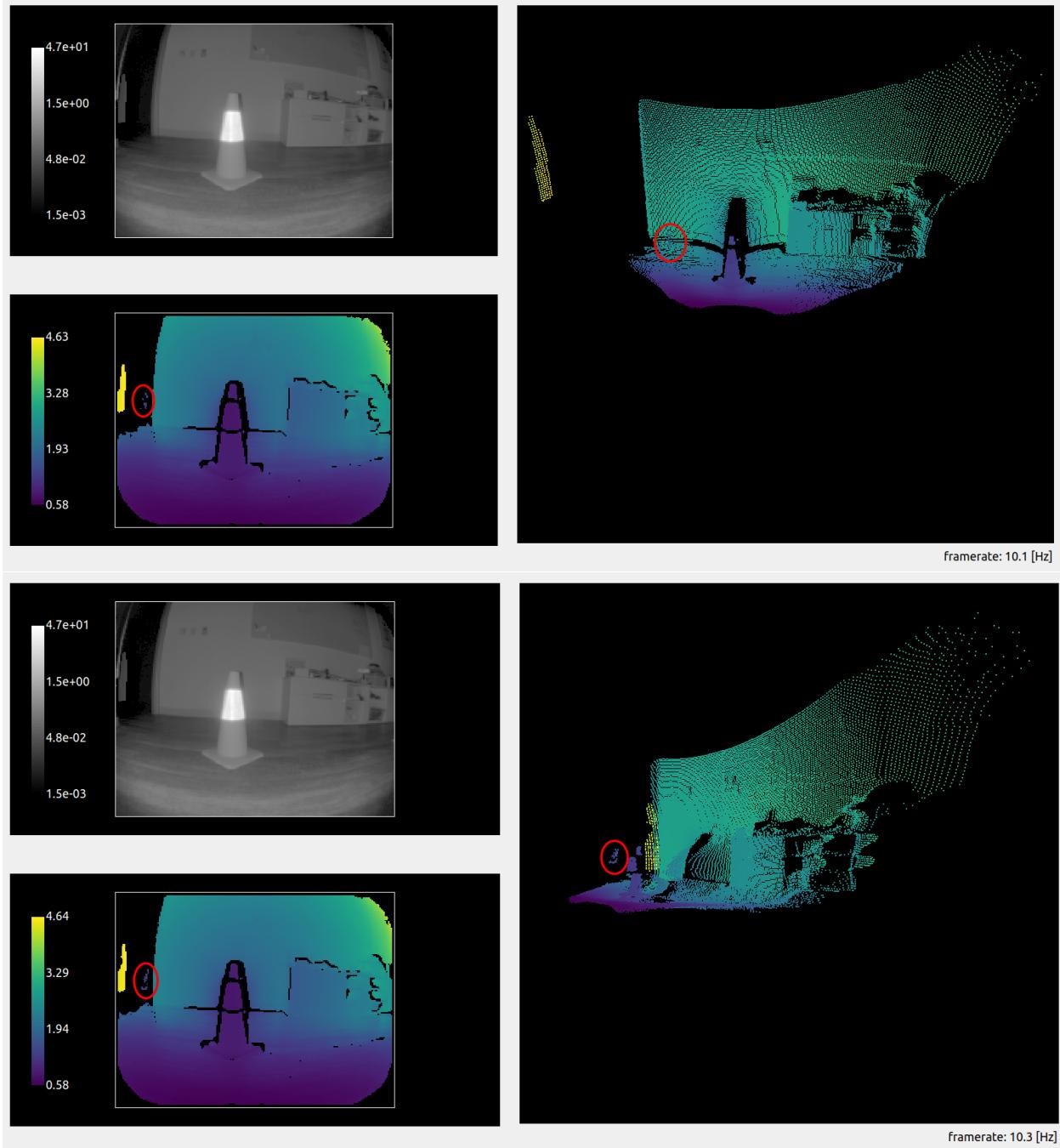
Let's look at the first scene again, but this time with the filter activated. We can see that the halo has been greatly reduced around the reflective part of the cone. A similar result is achieved with the box in the scene.



The stray light filter makes it possible to reduce mismeasured pixels in the vicinity of the reflector.

### Second case: “Ghost” pixels

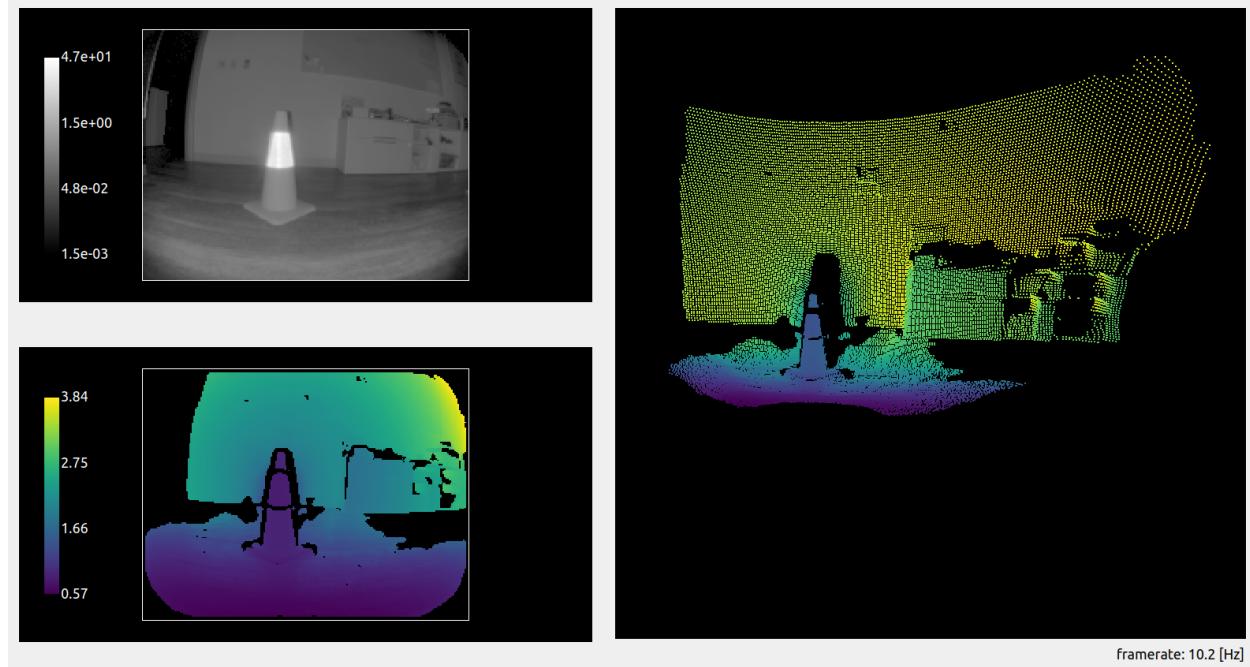
Let's look at a second case of stray light. A reflective cone is placed 1 m in front of the camera. The background is a white wall, at around 2 m. The stray light filter is deactivated.



We can see (highlighted in the red circle) that some pixels are measured that do not exist in the real scene (there is no object there). This is a second effect of stray light: nonexistent pixels appear in the close range. This could create false positive measurements, for instance when performing obstacle detection and trigger interruption of service.

## Handling “ghost” pixels

In the second scene, “ghost” pixels appeared in the close range due to stray light. The filter also mitigates this artifact:



The side effect of this is the removal of additional pixels on the scene, as we can see in the floor area. The O3R stray light filter invalidates pixels that are flagged as over-affected by the stray light: pixels whose distance or amplitude measurement is affected over the set threshold. We will see in the next section how to tune this distance threshold.

## Fine tuning the stray light filter

The default settings of the filter are chosen to properly handle most cases of stray light, but some scenes might require fine-tuning it. We focus here on obstacle detection cases that prove challenging with the default settings of the filter.

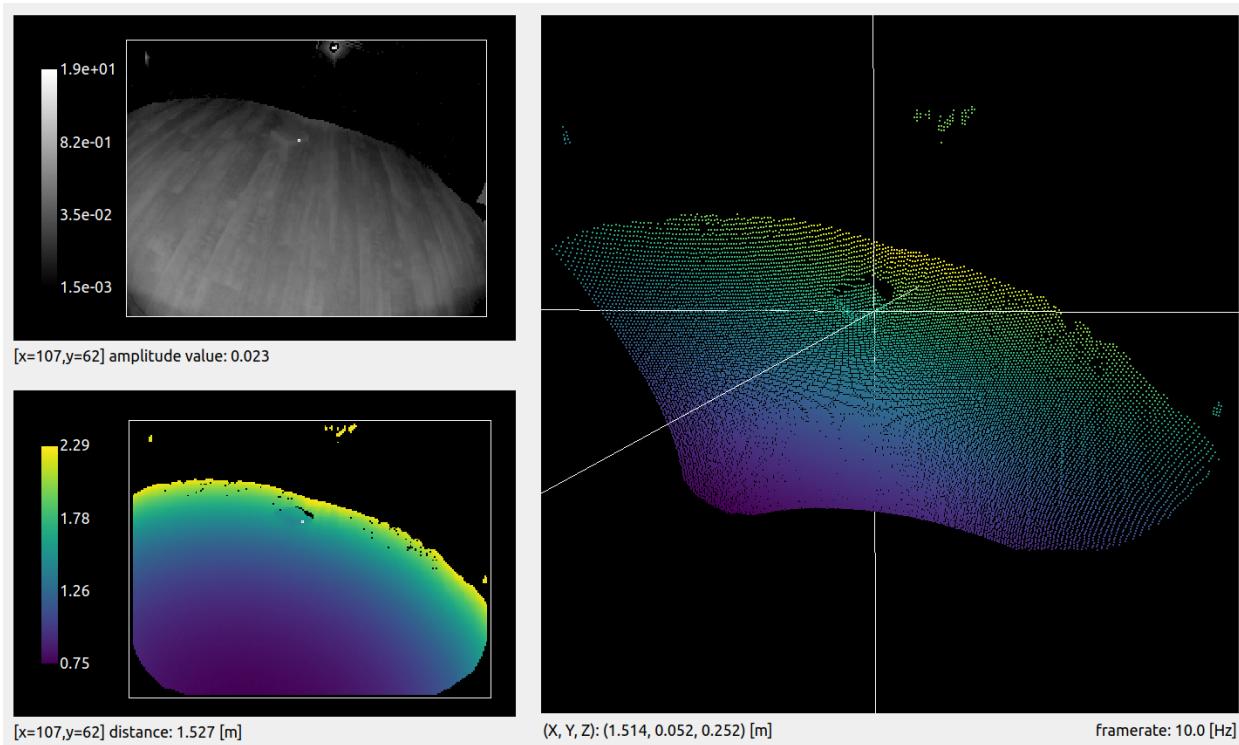
### Obstacle detection

In cases where obstacle detection is the main application, one can accept that stray light affects the scene such that some pixels will be measured as much as 10 cm off their real position, as long as the obstacle is detected and false negatives are avoided. Let’s focus on scenes where the sources of stray light are limited (a single reflector or smaller reflective surfaces).

The interesting setting in this case is the `excessiveCorrectionThreshDist`. This setting defines the distance threshold above which a pixel affected by stray light will be discarded.

Let’s look at a scene with and without stray light. A box (the obstacle) is placed on the floor, and we measure the distance to it.

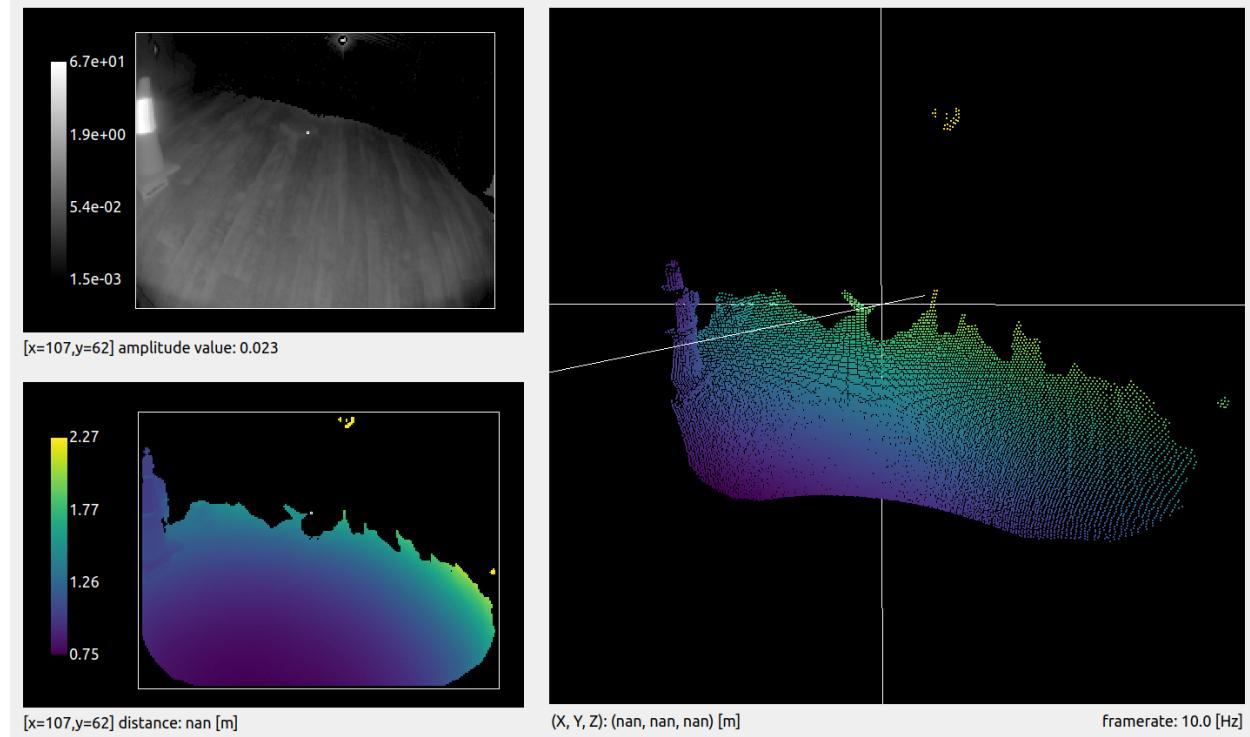
### Scene without reflector:



We choose a reference pixel on the side of the box. The distance measured for this pixel is 1.527 m.

**Scene with reflector:**

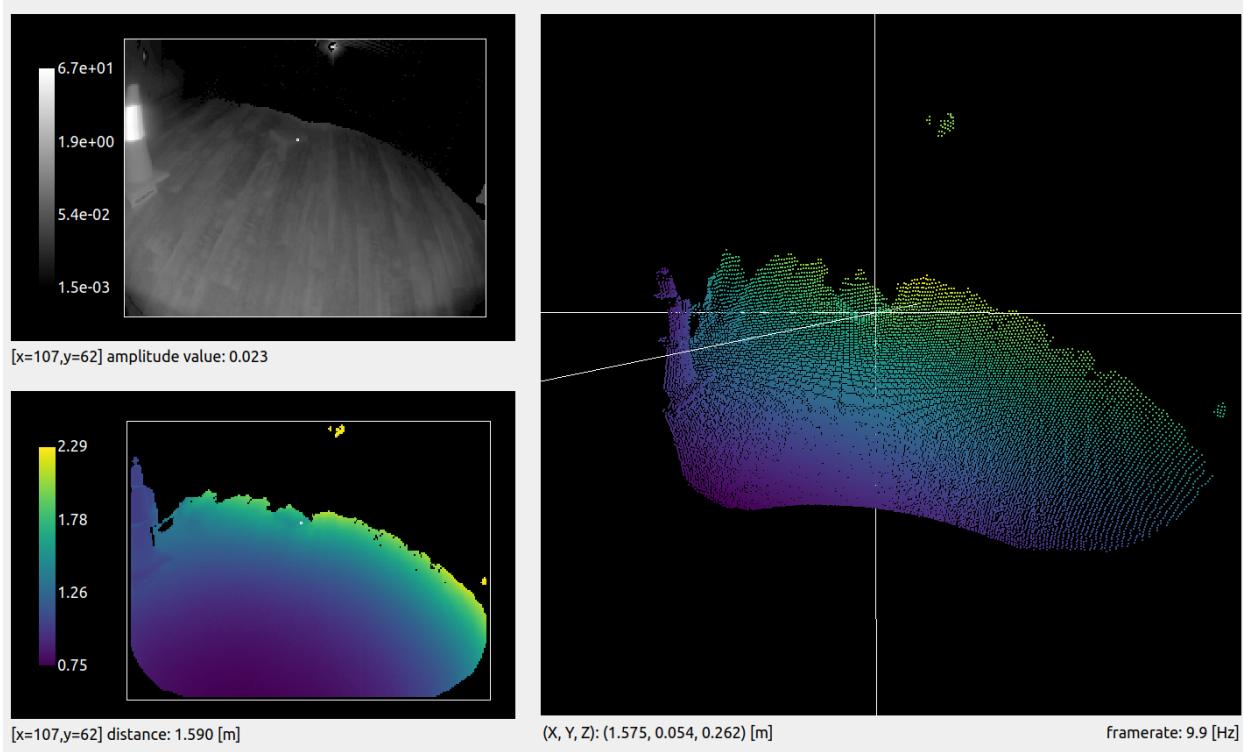
For the exact same scene, we add a reflector to the side of the field of view.



We can see that a whole side of the box disappears. This is due to the stray light that affects the scene: the pixels are marked as invalid by the stray light filter because of their distance measurement being affected.

**Adjusting the distance threshold**

Let's adjust the distance threshold setting, `excessiveCorrectionThreshDist`. Its default value is 0.08 m, which means that any pixel affected more than around 8 cm will be discarded. Let's set it to 0.2 m.



We can see that we are able to recover most of the pixels from the side of the box. However, the distance measured for the same pixel is now 1.590 m, when our reference measurement is 1.527 m. The side of the box is measured 6.3 cm off its actual position. This is acceptable because the robot will be able to drive along the reflector and still detect the presence of an obstacle.

Note: there are multiple filters activated by default with the O3R, which means pixels might be discarded due to a combination of multiple filters. This explains why the pixel was discarded while being only 6.3 cm off, and not 8 cm, as expected with the correction threshold we set.

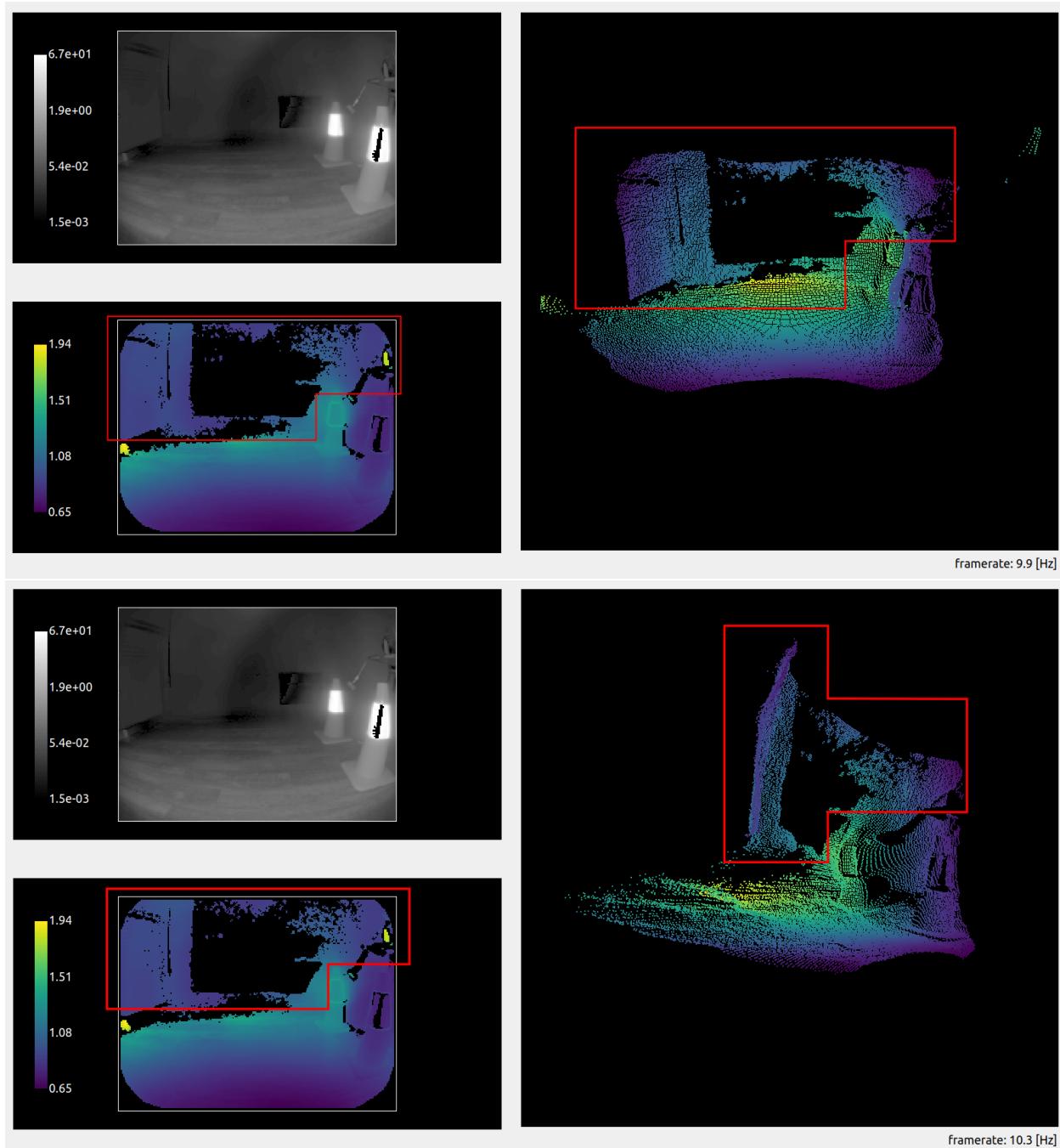
**Warning:** Relaxing the distance threshold value affects which stray light pixels are discarded by the filter. By doing so, you risk allowing some ghost pixels in the scene that might create false positive obstacles.

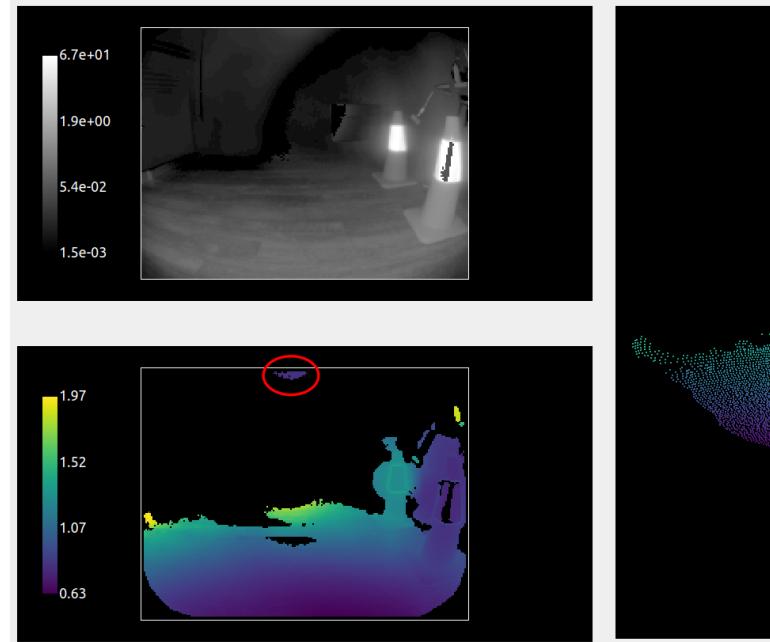
### Multiple reflectors in the scene

Scenes involving multiple reflectors can have pretty dramatic stray light artifacts. In these cases, getting rid of all the stray light ghost pixels might require you to strengthen the filter. Let's analyze a specific example and see how it can be handled.

The scene is set up with two reflectors along the expected path of the robot, and the left side is actually a glass wall. We imagine that the robot is moving forward, and, therefore, it must detect the floor and potential obstacles.

Let's have a look at the scene with the stray light filter deactivated. Pretty dramatic, right? The two reflectors create so much stray light that the whole path of the robot is blocked by ghost pixels (see the highlighted areas in red).

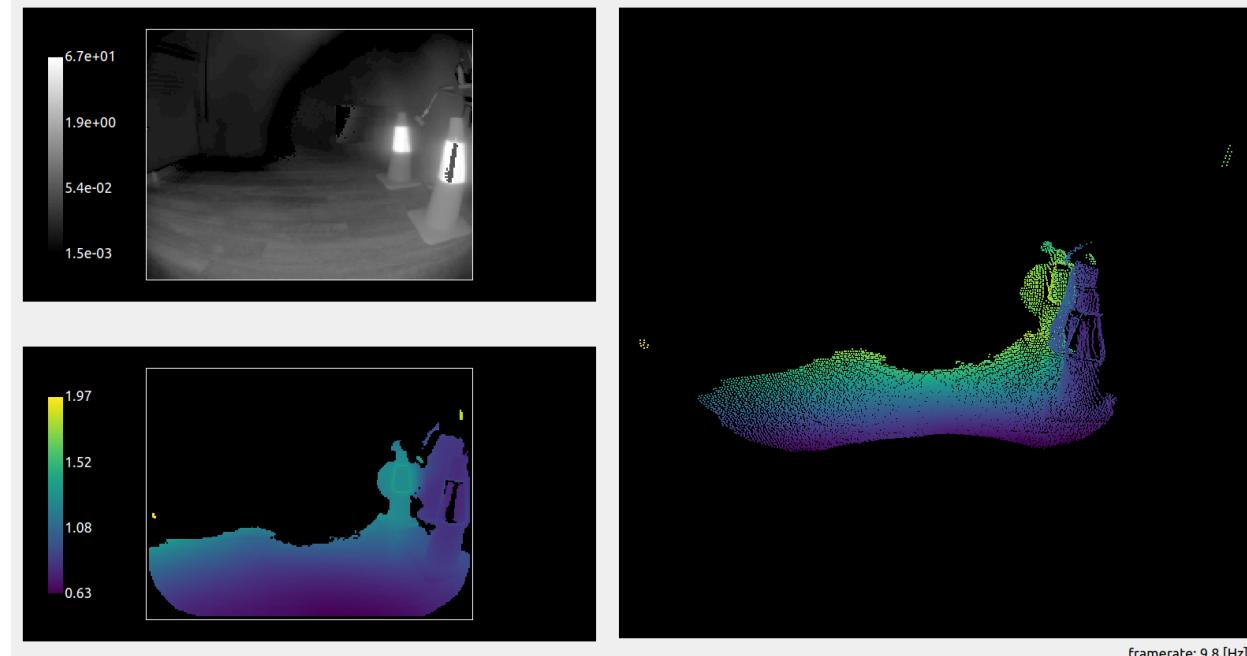




Now let's reactivate the filter and see what we get.

The ghost pixels almost completely disappeared, and the path forward is mostly clear. But we can still see remaining ghost pixels in the close range above the robot (denoted by the red circle). A measurement indicates that these pixels are 0.8 m away from the camera, which could be in the path of the robot if it or its payload reaches this height. The robot would therefore be blocked from moving forward by an obstacle detection algorithm, even though the path is clear.

We can once more adjust the stray light filter to ensure we filter away all the remaining ghost pixels. To do so, we can use the distance threshold, as presented earlier. This time, we are trying to remove extra pixels, so we are going to lower the distance threshold (we set it to 0.05 m). We can see that all the ghost pixels are now gone, and the robot can proceed to its route, obstacle free.



NOTE: This scene could also be handled with the amplitude threshold (`diParam.excessiveCorrectionThreshAmp`). This threshold follows the same concept as the distance threshold, invalidating pixels with an amplitude below the set value. Generally, setting the distance threshold suffices, but we encourage you to play with both to select the best configuration possible for your use case.

## Conclusion

As we saw with the examples detailed above, there is no “one-size-fits-all” configuration for handling stray light in robotics applications. Proper configuration needs to be selected after reviewing the scenes the robot is expected to encounter. The number and position of highly reflective surfaces that will enter the field of view at a certain time affects the necessary strength of the filter. Multiple configurations can be stored on board the O3R, and we encourage bringing intelligence to the use of the platform to adapt the cameras’ configurations to the scene where necessary. In scenes where a high accuracy of the ToF measurement is expected, we recommend avoiding placing reflectors in the scene or placing them reasonably far away from the navigation path of the robot.

The O3R provides several filters which can be used to handle specific ToF artifacts.

This section contains information and a description about the O3R parameters and their effect(s).

## 1.4 Docker on O3R

### 1.4.1 Build and run a docker container for the O3R platform

In this document we guide you through building a container from scratch. We start by building a small container. This container will increase in size and complexity the further we go. We will use a python base image and install the ifm3d (ifm3dpy) library.

If you want to use any of our available docker images or directly build on top of our Dockerfiles, you can jump directly to [this section](#) or check out the list of docker images officially supported by ifm coming soon.

Note that the O3R VPU (Video Processing Unit) is based on an NVIDIA Jetson system (TX2), which is arm64/aarch64 based. Building containers without the right base image will not run on the VPU, an arm64/aarch64 base image is needed. Please read carefully the instructions at the [Nvidia -> GitHub repository](#) for set-up instruction. For running an aarch64 container on a x86-64 host the section [Running or building a container on x86](#) is highly recommended.

#### Contents

- Build and run a docker container for the O3R platform
  - A basic container
    - \* Build the container
      - Troubleshooting: proxies
    - \* Run a container

- \* Save a container
  - Load and start a container
  - Add features to the container
  - Install ifm3d in the container
  - Building on top of the ifm base image

## A basic container

Every Docker container image is built by Docker using a Dockerfile. A docker file contains all the necessary information for building a container image. Most of the Dockerfiles are starting with a base image that is retrieved from the Docker Hub during the build process. Docker will automatically fetch the image for the architecture hosting the build (arm64/aarch64). When building a container for an architecture other than the hosts', the destination architecture needs to be specified in the Dockerfile. The Dockerfile is just a text file named Dockerfile without any file extension (watch out, it is case sensitive). You can use `docker build [path to Dockerfile]` to start the build process.

Our first container will use `arm64v8/python:3.9.6-slim-buster` as the base image. Let's build the first container with that base image.

Dockerfile:

```
#arm64v8 is the pre-requisite for running the container on the VPU.
FROM arm64v8/python:3.9.6-slim-buster
```

## Build the container

Building: To build a container use `docker build [path/to/Dockerfile]`. If an image tag (name) is needed, you can specify it within the docker build command.

```
# Assuming the Dockerfile is located within the same directory:
$ docker build . -t ifm3d
```

Note: For further information about `docker build` refer to the official [docker documentation](#)

Build process:

```
$ docker build . -t ifm3d
Sending build context to Docker daemon 2.048kB
Step 1/1 : FROM arm64v8/python:3.9.6-slim-buster
--> 4770e646d0be
Successfully built 4770e646d0be
Successfully tagged ifm3d:latest
```

If the build was successful, you should be able to use `docker image ls` to display all built images:

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ifm3d	latest	4770e646d0be	5 weeks ago	108MB

## Troubleshooting: proxies

Depending on the network infrastructure, docker might need the proxy information for building the container. You can input them directly when running the command:

```
#$HTTP_PROXY & $HTTPS_PROXY are variables containing the proxy address. E.g.: HTTPS_
↪PROXY=https://[PROXY ADDRESS]
$ docker image build --build-arg http_proxy=$HTTP_PROXY --build-arg https_proxy=
↪$HTTPS_PROXY -t jupyter .
```

You can also define the proxies in the config.json file. You should find the file within the home directory of the user executing docker, in a directory called .docker, which contains config.json. E.g. ~/.docker/config.json. If not available, create and save a config.json file containing the following:

```
{
  "proxies": {
    "default": {
      "httpProxy": "http://192.168.1.12:3128",
      "httpsProxy": "http://192.168.1.12:3128",
      "noProxy": "*.test.example.com,.example2.com,127.0.0.0/8"
    }
  }
}
```

## Run a container

Note: To run a container built for another chip architecture than the host system, you need to use qemu to handle the virtualization. For further information see:

- Docker multi-CPU architecture
- NVIDIA container runtime using qemu

To run the container, we use docker run. We can specify the run command through several arguments: we want to start the container interactively (-it) and with a bash interface (/bin/bash), so we can play around inside the container.

```
$ docker run -it ifm3d /bin/bash
WARNING: The requested image's platform (linux/arm64) does not match the detected
↪host platform (linux/amd64) and no specific platform was requested
root@ee24eff3c797:/#
```

Note: For further information about docker run, refer to the [official documentation](#)

Now we are within the container. The warning tells us that the base image was build for an arm64/aarch64 architecture, which is different from the architecture of the host (amd64).

We should be able to ask for the python version and start a REPL:

```
root@ee24eff3c797:/# python --version
Python 3.9.6
```

```
root@ee24eff3c797:/# python
Python 3.9.6 (default, Jun 29 2021, 19:34:26)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("hello")
hello
>>>
```

## Save a container

The container is working, let's save it so we can share it around. Docker already provides us with the right tool: `docker save`.

```
$ docker save ifm3d > ifm3d.tar
```

## Load and start a container

To reload the content of a previously saved image, use:

```
$ docker load < ifm3d.tar
```

Start the docker container like this on every other device:

```
$ docker run ifm3d
```

Note: The image name might be different than the saved container name. After `docker load`, docker will show the name of the loaded image

## Add features to the container

Until now, our container is not really useful. Let's update the container's kernel, install python packages and create a user (this will improve security). To do that, we need to improve the Dockerfile:

Dockerfile:

```
# This Dockerfile is a documentation example and might not build after a copy/paste ↴
process

#arm64v8 is the pre-requisite for running the container on the VPU.
FROM arm64v8/python:3.9.6-slim-buster

#Security updates
ARG DEBIAN_FRONTEND=noninteractive
RUN apt-get -y update && apt-get -y upgrade

#Create and activate virtual environment. This is not needed right now, but useful ↴
for multistage builds.
RUN python -m venv /opt/venv
ENV PATH="/opt/venv/bin:$PATH"

# Your normal pip installation, within the venv. We also update pip.
```

(continues on next page)

(continued from previous page)

```
RUN pip install -U pip && pip install numpy
#For security reasons, using a "user" is recommended
RUN useradd --create-home pythonuser
USER pythonuser

#Easier to debug the container if issues are happening
ENV PYTHONFAULTHANDLER=1
```

Build process:

```
$ docker build . -t ifm3d
Sending build context to Docker daemon  113.1MB
Step 1/9 : FROM arm64v8/python:3.9.6-slim-buster
--> 4770e646d0be
...
Step 6/9 : RUN pip install -U pip && pip install numpy
--> [Warning] The requested image's platform (linux/arm64) does not match the detected host platform (linux/amd64) and no specific platform was requested
--> Running in bb51c405bbdb
Requirement already satisfied: pip in /opt/venv/lib/python3.9/site-packages (21.1.3)
Collecting pip
  Downloading pip-21.2.2-py3-none-any.whl (1.6 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 21.1.3
    Uninstalling pip-21.1.3:
      Successfully uninstalled pip-21.1.3
Successfully installed pip-21.2.2
...
Step 9/9 : ENV PYTHONFAULTHANDLER=1
--> [Warning] The requested image's platform (linux/arm64) does not match the detected host platform (linux/amd64) and no specific platform was requested
--> Running in 4ea430894bc7
Removing intermediate container 4ea430894bc7
--> 14db5d89303f
Successfully built 14db5d89303f
Successfully tagged ifm3d:latest
```

Note: For easier readability, the build process output was shortened.

The build process contains several layers and intermediate container builds (that we use for debugging). You can start the container with the typical commands and check if numpy was installed:

```
$ docker run -it ifm3d:latest /bin/bash
WARNING: The requested image's platform (linux/arm64) does not match the detected host platform (linux/amd64) and no specific platform was requested
pythonuser@319eb5ea67e0:/$ pip freeze
numpy==1.21.1
```

## Install ifm3d in the container

ifm3dpy is the python binding for the ifm3D library. You can install it from source (download it [here](#)) or use the [docker image](#) provided by ifm which can be used on the VPU and contains the ifm3d and ifm3dpy libraries.

The Dockerfile could look similar to this:

```
# This Dockerfile is a documentation example and might not be build after a copy/
→ paste process

FROM ubuntu:20.04 AS build

# if defined, we run unit tests when building ifm3d
ARG run_tests

# if you are running unit tests against a camera at
# a different IP, set it here.
ENV IFM3D_IP 192.168.0.69
ENV DEBIAN_FRONTEND noninteractive

WORKDIR /home/ifm
RUN apt-get update && apt-get -y upgrade
RUN apt-get update && \
    apt-get install -y libboost-all-dev \
        git \
        libcurl4-openssl-dev \
        libgtest-dev \
        libgoogle-glog-dev \
        libxmlrpc-c++8-dev \
        libopencv-dev \
        libpcl-dev \
        libproj-dev \
        python3-dev \
        python3-pip \
        build-essential \
        coreutils \
        findutils \
        cmake \
        locales \
        ninja-build
RUN apt-get clean

# install python
RUN apt-get -y install --no-install-recommends build-essential \
    python3-dev

#install(Update) python packages and dependencies separate - improves Docker caching
→etc.
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# build pybind11 with cmake - but first clone from the official github repo
RUN git clone --branch v2.3.0 https://github.com/pybind/pybind11.git && \
    cd /home/ifm/pybind11 && \
    mkdir -p build && \
    cd build && \
    cmake -DPYBIND11_TEST=OFF .. && \
```

(continues on next page)

(continued from previous page)

```

make && \
make install

# First clone ifm3d repo via username and personal access token into the container
# and than build the ifm3d
# this build include ifm3d pybind for a python access via pybind11
ARG IFM3D_CLONE_REPO
RUN mkdir src && \
  cd src && \
  git clone --branch o3r/main ${IFM3D_CLONE_REPO} && \
  cd ifm3d && \
  echo "Building from current branch" && \
  mkdir build && \
  cd build && \
  cmake -GNinja -DCMAKE_INSTALL_PREFIX=/usr -DBUILD_MODULE_OPENCV=ON -DBUILD_MODULE_\
PCICCLIENT=ON -DBUILD_MODULE_PYBIND11=ON -DPYTHON_EXECUTABLE=/usr/bin/python3 ... && \
  \ \
  ninja && \
  ninja package && \
  ninja repackage
RUN ls -l /home/ifm/src/ifm3d/build/*.deb | grep -iv 'unspecified' | xargs dpkg -i

# multistage to reduce image size, hide secrets and add ifm user
FROM ubuntu:20.04

COPY --from=build /usr /usr

RUN apt-get update \
  && DEBIAN_FRONTEND=noninteractive apt-get install -y && apt-get clean

```

Note: You should leverage the layering from Docker to improve the build speed if you need to build again. Qemu emulates a ARM64 CPU in software on a x86 System which is slow. In case you are planning to build large application from source please consider to run this on a ARM64 based host.

We provide up-to-date images containing the ifm3d library, both on the docker hub [here](#) and on github [here](#). We recommend using the image available on github, as it does not come with rate limits. You can simply pull it like so:

```

$ docker pull ghcr.io/ifm/ifm3d:latest
latest: Pulling from ifm/ifm3d
...
Digest: sha256:f54a5890d75618c5bd21535dfa71e1cd9b1a8515902fb8e1912e6f586e0685a3
Status: Downloaded newer image for ghcr.io/ifm/ifm3d:latest
ghcr.io/ifm/ifm3d:latest

```

Note: Due to easier readability, the pull process output was shortened

Let's try the image and see if we can connect to a (physically connected) VPU:

```

$ docker run -it ghcr.io/ifm/ifm3d:latest
ifm@1f21eb1f98d2:/$ ifm3d dump
{
  "device": {
    "clock": {
      "currentTime": 1581111542490926304,
      ...

```

If this is working, we can also try the ifm3dpy implementation:

```
ifm@8a167fde9edc:/ $ python3
Python 3.6.9 (default, Apr 18 2020, 01:56:04)
[GCC 8.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import ifm3dpy
>>> o3r = ifm3dpy.O3RCamera()
>>> o3r.to_json()
{'device': {'clock': {'currentTime': ...}}
```

### **Building on top of the ifm base image**

Now you want your own container, with your python script to run. Base your Dockerfile simply on the `ghcr.io/ifm/ifm3d:latest` image:

```
FROM ghcr.io/ifm/ifm3d:latest
```

You can now include your application code.

Once the image is built, you can deploy it to the VPU. Read more [here](#).

### **1.4.2 Deploying a container to the VPU**

There are several ways for deploying a container. This documentation focuses on the following two:

- Using `scp`
- Using a local docker registry

Every VPU has two users:

- root - ifm user with all rights
- oem - customer user, this is the only one you have access to.

The first step to access the VPU is to connect to it via SSH.

#### **SSH connection**

To connect to the VPU via ssh, follow these steps:

1. Generate an ssh key-pair
2. Upload the public key to the VPU
3. Connect to the VPU using the passphrase

## 1. Generate ssh key-pair

All user specific ssh keys are located at `~/.ssh`. This is the place where the private key for the connection to the VPU should be stored.

To generate an ssh key-pair, use `ssh-keygen`:

```
$ cd ~/.ssh/
~/ssh$ ssh-keygen -t rsa -b 4096 -C "[email-address]"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/devoegse/.ssh/id_rsa): id_o3r
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
...
```

A passphrase is also needed. After that command, two new keys are generated within the `~/.ssh` directory. With the example above it would be: `id_o3r` & `id_o3r.pub`.

## 2. Upload the public key to the VPU

Uploading the public (`.pub`) ssh key to the VPU is achieved via the `ifm3d` library. The device configuration includes a parameter for authorized keys: `authorized_keys`.

```
"network": {
    "authorized_keys": "",
    "ipAddressConfig": 0,
    "macEth0": "00:04:4B:EA:95:FB",
    "macEth1": "00:02:01:23:33:36",
    "networkSpeed": 1000,
    "staticIPv4Address": "192.168.0.69",
    "staticIPv4Gateway": "192.168.0.201",
    "staticIPv4SubNetMask": "255.255.255.0",
    "useDHCP": false
},
```

To add a new key, the VPU configuration needs to be changed. This can be done with several ways (see configuring the camera). The easiest way in this case is to use the `jq` command:

```
$ ifm3d dump | jq --arg id "$(cat ~/.ssh/id_o3r.pub)" '.device.network.authorized_keys=$id' | ifm3d config
```

- `ifm3d dump` - This command receives the current configuration from the VPU.
- `jq --arg id "$(cat ~/.ssh/id_o3r.pub)"` - This loads the public key into the variable `id` and provides it to the `jq` command
- `'.device.network.authorized_keys=$id'` - Here the json value from `authorized_keys` is changed for the public key within the variable `id`
- `ifm3d config` - The new json is now used to change the configuration of the VPU via `ifm3d config`

### 3. Connect to the VPU using the passphrase

After the key is uploaded, it is possible to connect with ssh and the username oem to the VPU:

```
$ ssh oem@192.168.0.69
The authenticity of host '192.168.0.69 (192.168.0.69)' can't be established.
ECDSA key fingerprint is SHA256:8gjC9za45TTRZNz5JCMwaNJ27BLfsPyDtjBaBQ2vyHw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.0.69' (ECDSA) to the list of known hosts.
o3r-vpu-c0:~$
```

There will be a prompt for the passphrase, configured during step 1.

### SCP

The first way to transfer a container to the VPU is to copy a saved container via scp.

```
path/to/container/folder$ scp ifm3d.tar oem@192.168.0.69:/home/oem/
oem@192.168.0.69's password:
ifm3d.tar                                         100% ↗
→ 108MB   51.5MB/s   00:02
```

The system will ask for a password: oem

To verify if the copy process worked, use the command sync on the VPU after the copying the container.

Note: Use ssh to connect to the VPU - see [SSH connection](#)

Note: The oem user has no write rights outside of his/her home directory. Therefore use /home/oem/ for saving files etc. It is possible to create folders within the oem directory.

Once you copied the container, you can load and start it (see [instructions](#))

### Local docker registry

Due to the fact that proxy servers are sometimes hard to deal with and that disk resources on the VPU is also limited, it might come handy to run a Docker registry in your local network.

#### Create a local Docker registry

The local Docker registry is created by using the container images provided by Docker itself and host them. On the host system (not the VPU) activate a local Docker registry with following commands:

```
$ docker pull registry:latest
# Start the registry and bind the container ports to the host ports
$ docker run -d -p 5000:5000 --name registry registry:latest
```

Note: A local registry might seem complicated at first. For further information refer to the [official documentation](#).

### **Push a container to your local registry**

To push a container to the registry, it is recommended to first tag the image differently. E.g. if the registry is run on the localhost with port 5000, the image tag could be named:

```
docker tag ifm3d localhost:5000/ifm3d
```

Use the normal push command for uploading to the local registry:

```
docker push localhost:5000/ifm3d
```

### **Pull a container from the local registry - host**

If a local Docker registry is running, use `docker pull` to pull the image:

```
docker pull localhost:5000/ifm3d
```

### **Pull a container from the local registry - VPU**

Coming soon

### **Stop the registry**

To stop the registry:

```
docker container stop registry && docker container rm -v registry
```

### **1.4.3 Autostart a container on the VPU**

Once the container(s) has(ve) been transferred to the VPU, you can set up an autostart service to automatically run the containers as start-up. For auto-starting a container, `Docker compose` is used. The VPU already provides a service `.config/systemd/user/oem-dc@.service` which can be used for auto-starting a service: this is what we will use.

#### **Docker compose**

Generate a sample directory and a `docker-compose.yml` file at following destination: `/usr/share/oem/docker/compose/`. E.g. `/usr/share/oem/docker/compose/jupyter/docker-compose.yml`

This file should contain the information for starting the container you need.

## Sample docker-compose.yml

The following docker-compose.yml file would create a service called jupyter, based on the image jupyter and bind the container ports 8888 to the host port 8888 on start.

```
version: "3.3"
services:
  jupyter:
    image: jupyter
    ports:
      - 8888:8888
```

Note: The Docker version on the VPU expects the docker-compose.yml to be either version 2.2 or 3.3. Fur further information refer to [docker compose](#).

## Start the container(s)

A docker-compose.yml can be started via docker-compose up within the docker-compose.yml directory.**TODO: add example of docker compose-up** It is also possible to start the service with systemctl:

```
systemctl --user start oem-dc@jupyter
```

After a few seconds, the service should have started and it is possible to get the status of this service:

```
systemctl --user status oem-dc@jupyter
```

### **TODO: add the result of this cmd**

Another way of seeing all running container is docker ps.

## Auto start the container(s) after a reboot of the VPU

To restart the container automatically, simply enable the service:

```
systemctl --user enable oem-dc@jupyter
```

See Start the container on how to start the container with a docker-compose.yml file

## Save data on consistently on the VPU with a container

### **TODO: move this section to a more appropriate chapter** Coming soon

Data created and saved within a container is only available for the running instance of the container itself. Restarting the container leads to a loss of the previously saved data. Use volumes to avoid this scenario.

#### 1.4.4 Enabling GPU usage on the VPU

##### Using the GPU of the VPU

The VPU provides substantial GPU (Graphical Processing Unit) power to the user. The best way to experience this is using CUDA and the samples from NVIDIA. To do so, we are building a container with the sample files from NVIDIA, push it to the VPU and execute it. This, however is not enough. Docker is not using the GPU power if not specified. We need to activate this too via the right runtime.

##### Dockerfile sample

The following Dockerfile builds the container with the samples from NVIDIA (see <https://github.com/NVIDIA/cuda-samples/tree/master/Samples/deviceQuery>).

Dockerfile:

```
# Base linux for tegra (l4t) amr64/aarch64 image
FROM nvcr.io/nvidia/l4t-base:r32.4.3 AS buildstage

# Install necessary updates + git (for cloning the nvidia samples). Tag v10.2
# specifies the right commit. VPU runs CUDA 10.2
RUN apt-get update && apt-get install -y --no-install-recommends make g++ git && apt-
# get install ca-certificates -y
RUN git clone --depth 1 --branch v10.2 https://github.com/NVIDIA/cuda-samples.git /
# tmp/

# Change into the right directory and install/make the samples
WORKDIR /tmp/Samples/deviceQuery
RUN make clean && make

# Multistage build to reduce the image size on the platform
FROM nvcr.io/nvidia/l4t-base:r32.4.3

# Copy the samples from the buildstage into the final image
RUN mkdir -p /usr/local/bin
COPY --from=buildstage /tmp/Samples/deviceQuery/deviceQuery /usr/local/bin

# Execute the deviceQuery and check for CUDA support. Don't forget the runtime with
# the docker run command
CMD ["/usr/local/bin/deviceQuery"]
```

Building the container:

```
$ docker image build . -t cuda-samples
Sending build context to Docker daemon 875.5MB
Step 1/9 : FROM nvcr.io/nvidia/l4t-base:r32.4.3 AS buildstage
--> c93fc89026d9
...
Successfully tagged cuda-samples:latest
```

After building the container, you can follow the steps from the documentation to test the container on the VPU:

- Save the container: \$ docker save cuda-samples > cuda-samples.tar
- Transfer the container: \$ scp cuda-samples.tar oem@192.168.0.69:/home/oem

- Load the container: `$ docker load < cuda-samples.tar`

### Start the container with the NVIDIA runtime

To use CUDA and the GPU, you have to specify the NVIDIA runtime, either with the `docker run` command, or within the `docker-compose.yml` (see [autostart](#)).

### Using `docker run`

Use the `--runtime nvidia` argument when running your container. The output of the running container should look similar to this:

```
o3r-vpu-c0:~$ docker run -it --runtime nvidia cuda-samples
/usr/local/bin/deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "NVIDIA Tegra X2"
  CUDA Driver Version / Runtime Version           10.2 / 10.2
  CUDA Capability Major/Minor version number:      6.2
  Total amount of global memory:                 3829 MBytes (4014751744 bytes)
  ( 2) Multiprocessors, (128) CUDA Cores/MP:      256 CUDA Cores
  GPU Max Clock rate:                            1300 MHz (1.30 GHz)
  Memory Clock rate:                            1300 Mhz
  Memory Bus Width:                             128-bit
  L2 Cache Size:                                524288 bytes
  Maximum Texture Dimension Size (x,y,z)        1D=(131072), 2D=(131072, 65536), u
  ↪3D=(16384, 16384, 16384)
  Maximum Layered 1D Texture Size, (num) layers   1D=(32768), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers   2D=(32768, 32768), 2048 layers
  Total amount of constant memory:                65536 bytes
  Total amount of shared memory per block:        49152 bytes
  Total number of registers available per block: 32768
  Warp size:                                    32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:          1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                          2147483647 bytes
  Texture alignment:                           512 bytes
  Concurrent copy and kernel execution:        Yes with 1 copy engine(s)
  Run time limit on kernels:                   No
  Integrated GPU sharing Host Memory:         Yes
  Support host page-locked memory mapping:    Yes
  Alignment requirement for Surfaces:         Yes
  Device has ECC support:                     Disabled
  Device supports Unified Addressing (UVA): Yes
  Device supports Compute Preemption:        Yes
  Supports Cooperative Kernel Launch:        Yes
  Supports MultiDevice Co-op Kernel Launch: Yes
  Device PCI Domain ID / Bus ID / location ID: 0 / 0 / 0
  Compute Mode:
```

(continues on next page)

(continued from previous page)

```
< Default (multiple host threads can use ::cudaSetDevice() with device
simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.2, CUDA Runtime Version =
10.2, NumDevs = 1
Result = PASS
```

Note that starting the container without the runtime leads to a FAIL:

```
o3r-vpu-c0:~$ docker run -it cuda
/usr/local/bin/deviceQuery Starting...

CUDA Device Query (Runtime API) version (CUDART static linking)

cudaGetDeviceCount returned 35
-> CUDA driver version is insufficient for CUDA runtime version
Result = FAIL
```

### Use docker-compose to specify the runtime

Specifying the runtime in `docker-compose.yml` is possible for versions above version: "2.3" to get the runtime argument.

```
version: "2.3"
services:
  cuda:
    image: cuda
    runtime: nvidia
```

Start the container using `docker-compose up`:

```
o3r-vpu-c0:~$ docker-compose up
Creating network "oem_default" with the default driver
Creating oem_cuda_1 ... done
Attaching to oem_cuda_1
cuda_1  | /usr/local/bin/deviceQuery Starting...
cuda_1  |
cuda_1  |   CUDA Device Query (Runtime API) version (CUDART static linking)
cuda_1  |
cuda_1  |   Detected 1 CUDA Capable device(s)
cuda_1  |
cuda_1  |   Device 0: "NVIDIA Tegra X2"
cuda_1  |   CUDA Driver Version / Runtime Version      10.2 / 10.2
cuda_1  |   CUDA Capability Major/Minor version number: 6.2
cuda_1  |   Total amount of global memory:            3829 MBytes (4014751744
bytes)
cuda_1  |   ( 2) Multiprocessors, (128) CUDA Cores/MP: 256 CUDA Cores
cuda_1  |   GPU Max Clock rate:                      1300 MHz (1.30 GHz)
cuda_1  |   Memory Clock rate:                       1300 Mhz
cuda_1  |   Memory Bus Width:                        128-bit
cuda_1  |   L2 Cache Size:                          524288 bytes
cuda_1  |   Maximum Texture Dimension Size (x,y,z)  1D=(131072), 2D=(131072,
65536), 3D=(16384, 16384, 16384)
cuda_1  |   Maximum Layered 1D Texture Size, (num) layers 1D=(32768), 2048 layers
```

(continues on next page)

(continued from previous page)

cuda_1	Maximum Layered 2D Texture Size, (num) layers	2D=(32768, 32768), 2048 ↴layers
cuda_1	Total amount of constant memory:	65536 bytes
cuda_1	Total amount of shared memory per block:	49152 bytes
cuda_1	Total number of registers available per block:	32768
cuda_1	Warp size:	32
cuda_1	Maximum number of threads per multiprocessor:	2048
cuda_1	Maximum number of threads per block:	1024
cuda_1	Max dimension size of a thread block (x,y,z):	(1024, 1024, 64)
cuda_1	Max dimension size of a grid size (x,y,z):	(2147483647, 65535, 65535)
cuda_1	Maximum memory pitch:	2147483647 bytes
cuda_1	Texture alignment:	512 bytes
cuda_1	Concurrent copy and kernel execution:	Yes with 1 copy engine(s)
cuda_1	Run time limit on kernels:	No
cuda_1	Integrated GPU sharing Host Memory:	Yes
cuda_1	Support host page-locked memory mapping:	Yes
cuda_1	Alignment requirement for Surfaces:	Yes
cuda_1	Device has ECC support:	Disabled
cuda_1	Device supports Unified Addressing (UVA):	Yes
cuda_1	Device supports Compute Preemption:	Yes
cuda_1	Supports Cooperative Kernel Launch:	Yes
cuda_1	Supports MultiDevice Co-op Kernel Launch:	Yes
cuda_1	Device PCI Domain ID / Bus ID / location ID:	0 / 0 / 0
cuda_1	Compute Mode:	
cuda_1	< Default (multiple host threads can use ::cudaSetDevice() with device ↴simultaneously) >	
cuda_1	deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 10.2, CUDA Runtime ↴Version = 10.2, NumDevs = 1	
cuda_1	Result = PASS	
oem_cuda_1 exited with code 0		

This section explains the handling and deployment of customer specific containers for the O3R.

## 1.5 FAQ - Frequently Asked Questions

Coming soon...

Here you can find different topics related to the O3R product. Containing information about:

- configuring the O3R
- using Docker on the O3R
- getting data from the system
- etc.



## **IFM3D LIBRARY**

### **2.1 ifm3d Overview**

Library and utilities for working with ifm pmd-based 3D ToF Cameras.



#### **2.1.1 Release versions**

☐ Note that the `master` branch is generally in a work in progress state and you probably want to use a tagged `release version` for production.

☐ branch `o3r/main-next` is an early adopters version which may/will contain changes to the interface in the future.

#### **2.1.2 Current Revision**

#### **2.1.3 Organization of the Software**

The ifm3d software is organized into modules, they are:

As of version 0.9.0, we have removed the `viewer` sub-command from the `ifm3d` command line tool (part of the `tools` module). The objective was to lessen the dependencies for the core library. However, a clone of the pre-0.9.0 viewer is available in its own repository: [ifm3d-pcl-viewer](#).

#### **2.1.4 Additional Resources**

- Viewing the Point Cloud
- ROS
- ROS 2

## 2.1.5 Known Issues, Bugs, and our TODO list

Please see the [Github Issue Tracker](#).

## 2.1.6 LICENSE

Please see the file called LICENSE.

# 2.2 Installing the software

## 2.2.1 O3R early adopters

### Installing ifm3d from source

#### Build Dependencies

Additionally, if you plan to build the debian packages and have the dependencies computed for you dynamically (see the note below on the repackage target), you will also need:

- `Python 2.7`
- `readelf` (Part of the `binutils` package)
- `ldd` (Part of the `libc-bin` package)
- `dpkg`

We note that, if you are running on a supported Linux, all of these packages are available through the official debian repositories and should be a simple `apt-get` away from being installed on your machine.

Use the following steps to install all the library dependencies on Debian based systems

```
$ sudo apt-get update && sudo apt-get -y upgrade
$ sudo apt-get update && sudo apt-get install -y
    git \
    jq \
    libcurl4-openssl-dev \
    libgtest-dev libgoogle-glog-dev \
    libxmlrpc-c++8-dev \
    libproj-dev \
    build-essential \
    coreutils \
    cmake
# Only if you wish to build the image and/or opencv modules
$ sudo apt-get update && sudo apt-get install -y libopencv-dev libpcl-dev
# Only if you wish to build the python bindings
$ sudo apt-get update && sudo apt-get install pybind11-dev
```

Note: The package name may differ in different flavours of Linux. Above `apt-get` commands are specific to Debian based systems

## Building From Source

Start with cloning the code from the ifm3d github repository [here](#).

□ The code on the branch o3r/main-next is updated nightly and contains the latest changes to the library. It is typically a work in progress.□ We recommend using tagged versions for your builds, to ensure consistency between builds. The latest tagged version can be found [here](#).

### The default build

By default, the ifm3d build enables the camera, framegrabber, stlimage, and tools modules. Building the software follows the usual cmake idiom of:

```
$ mkdir build
$ cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
$ cmake --build .
$ sudo cmake --build . --target install
```

### Building with PCL and/or OPENCV

ifm3d provides multiple image buffers. The default one, stlimage only relies on standard c++ libraries. The image module relies on opencv and pcl. The opencv modules relies only on openCV. To build either of these use the following:

```
$ mkdir build
$ cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr -DBUILD_MODULE_OPENCV=ON -DBUILD_IN_DEPS=OFF ..
# OR
$ cmake -DCMAKE_INSTALL_PREFIX=/usr -DBUILD_MODULE_IMAGE=ON -DBUILD_IN_DEPS=OFF ..
$ cmake --build .
$ sudo cmake --build . --target install
```

Note: Many ifm3d users ultimately plan to use this library along with its associated ROS wrapper. If this is the case, you need to be sure that the version of OpenCV that you link to in both ifm3d and ifm3d-ros are consistent. To give you some control over that, the build process allows you to explicitly call out which version of OpenCV you wish to use. For example, if you are using OpenCV 2.4, your cmake line above should look something like: \$ cmake -DCMAKE\_INSTALL\_PREFIX=/usr -DFORCE\_OPENCV2=ON ... Similarly, if you are using OpenCV 3, your cmake line above should look something like: \$ cmake -DCMAKE\_INSTALL\_PREFIX=/usr -DFORCE\_OPENCV3=ON ..

## Building the Python Bindings

There are several options available for building and/or installing the `ifm3dpy` module. The most simple one is to install with pip:

```
pip install ifm3dpy
```

For more options, please refer to the python documentation.

## A sumo-build

If you want to build everything:

```
$ mkdir build
$ cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr -DBUILD_MODULE_OPENCV=ON -DBUILD_MODULE_
    ↳PCICCLIENT=ON -DBUILD_MODULE_IMAGE=ON -DBUILDS_IN_DEPS=OFF ..
$ cmake --build .
$ sudo cmake --build . --target install
```

## Building the examples

The examples can be built along with the rest of the library by switching the proper flag on. Assuming you are in the `/build` folder:

```
$ cmake -DCMAKE_INSTALL_PREFIX=/usr -DBUILD_O3R_EXAMPLES=ON ..
$ make
```

## Building debian packages

Alternatively, to build debs to be distributed to multiple runtime machines, you can use the following:

```
$ mkdir build
$ cd build
$ cmake -DCMAKE_INSTALL_PREFIX=/usr ..
$ cmake --build .
$ cmake --build . package
$ cmake --build . repackage
$ sudo dpkg -i ifm3d_0.18.0_amd64-camera.deb
$ sudo dpkg -i ifm3d_0.18.0_amd64-swupdater.deb
$ sudo dpkg -i ifm3d_0.18.0_amd64-framegrabber.deb
$ sudo dpkg -i ifm3d_0.18.0_amd64-stlimage.deb
$ sudo dpkg -i ifm3d_0.18.0_amd64-tools.deb
```

(The version number embedded in the deb file will be dependent upon which version of the `ifm3d` software you are building)

Note: Experienced users may be puzzled by the `repackage` step. This step is used to dynamically compute the debian dependencies for the particular module. Due to how we are partitioning out the software, this approach is necessary vs. the more traditional `CPACK_DEBIAN_PACKAGE_SHLIBDEPS` wrapper around `dpkg-shlibdeps`.

We basically created a version of that tool that exploits a-priori information about the ifm3d environment to properly compute the debian dependencies. If you are building debs on a build machine to be distributed out to various runtime computers, you will certainly want to execute the repackage target so that you are ensured the runtime machines have the proper dependency chain in place.

## Building ifm3d from source on Windows

This tutorial details how to compile the ifm3d library and its dependencies on a Windows platform using Visual Studio.

### Dependencies

#### Build tools

- CMake v3.11.0 or newer (also available through the Visual Studio installer)
- Git for Windows (also available through the Visual Studio installer)
- Microsoft Visual Studio version 2017 or 2019. The free ‘Community’ edition is sufficient. Be sure to select the ‘Desktop development with C++’ workflow.

### Binary Dependencies for ifm3d::image and ifm3d::opencv optional modules

#### PCL

PCL is available in binary form for Windows platforms via the project’s GitHub releases page. ifm3d is tested against version v1.8.1 which can be downloaded and installed via the following links.

Choose ONE of the following based on your target version of Visual Studio.

- Visual Studio 2017/2019: [PCL-1.8.1-AllInOne-msvc2017-win64.exe](#)

**NOTE:** Opt-in to installing the 3rd party dependencies of PCL. ifm3d also takes a dependency on the boost library. For simplicity/compatibility, this tutorial builds against the version of boost provided along with PCL.

**NOTE:** It is possible to build an ifm3d variant which does not depend on the PCL library. See Appendix A below for instructions.

#### OpenCV 3.4

OpenCV 3.4 is available in binary form for Windows platforms from the [OpenCV Releases](#) page. ifm3d is tested against v3.4.9, available from the following link:

- [OpenCV 3.4.9](#)

Download it and extract to a known location (this tutorial assumes a path of C:\opencv).

## Source Dependencies

ifm3d depends on several additional libraries (curl, xmlrpc-c, glog, and gtest) which are not available as binary packages on Windows.

### Building source dependencies with ifm3d

ifm3d from version 0.90.4 onwards provides BUILT\_IN\_DEPS option to cmake configure command, which fetch required dependencies and build it with ifm3d. On sucessfull first installation user can disable BUILT\_IN\_DEPS option and can use the installed dependencies for future builds of the ifm3d.

**Note:** As gtest is not a part of the build time dependencies, to enable the testing of ifm3d please clone googletest as explained in gtest section.

Following instructions detail how to build ifm3d along with its dependencies.

```
#set the environment variables
set IFM3D_CMAKE_GENERATOR="Visual Studio 16 2019"
set IFM3D_BUILD_DIR=C:\ifm3d
set CONFIG=Release #set to Debug for debug binaries

#make the working dir
mkdir %IFM3D_BUILD_DIR%

# Clone the repository
cd %IFM3D_BUILD_DIR%
git clone https://github.com/ifm/ifm3d.git
cd %IFM3D_BUILD%\ifm3d

# Configure
mkdir build
cd build
cmake -G %IFM3D_CMAKE_GENERATOR% -DCMAKE_WINDOWS_EXPORT_ALL_SYMBOLS=ON -DBUILD_SDK_ ↵PKG=ON -DGTEST_CMAKE_DIR=%IFM3D_BUILD_DIR%\googletest\googletest -Dgtest_force_ ↵shared_crt=TRUE -DCMAKE_PREFIX_PATH=%IFM3D_BUILD_DIR%\install -DCMAKE_BUILD_TYPE= ↵%CONFIG% -DCMAKE_INSTALL_PREFIX=%IFM3D_BUILD_DIR%\install ..

# Build ifm3d and dependencies
cmake --build . --config %CONFIG% --target ALL_BUILD

# install
cmake --build . --config %CONFIG% --target install
```

On successful execution of install step, user can disable the BUILT\_IN\_DEPS flag by appending -DBUILD\_IN\_DEPS=OFF to cmake configure step, this will avoid building dependencies on every clean build.

Note: By default ifm3d::Image and ifm3d::opencv modules are disabled, to enabled these modules use -DBUILD\_MODULE\_IMAGE=ON and -DBUILD\_MODULE\_OPENCV=ON respectively to cmake configure command. Also append the opencv install binary path to -DCMAKE\_PREFIX\_PATH as shown in Building ifm3d section

## Buidling the source dependencies independent of ifm3d

The following instructions detail how to compile them from source for your target.

### Environment Configuration

The following environment variables are used by this tutorial to make customization simpler. Modify them as needed for your environment. You can obtain a list of valid cmake generator strings via `cmake -h`. Again, ifm3d supports version 2017 and newer.

```
set IFM3D_OPENCV_PATH=C:\opencv\build
set IFM3D_CMAKE_GENERATOR="Visual Studio 15 2017 Win64"
set IFM3D_BUILD_DIR=C:\ifm3d
set CONFIG=Release
```

Finally, create the working directory in which ifm3d and its dependencies will be built:

```
mkdir %IFM3D_BUILD_DIR%
```

### curl

```
cd %IFM3D_BUILD_DIR%
git clone --branch curl-7_47_1 https://github.com/curl/curl.git
cd %IFM3D_BUILD_DIR%\curl
mkdir build
cd build
cmake -G %IFM3D_CMAKE_GENERATOR% -DCMAKE_WINDOWS_EXPORT_ALL_SYMBOLS=ON -DCMAKE_
↪INSTALL_PREFIX=%IFM3D_BUILD_DIR%\install ..
cmake --build . --clean-first --config %CONFIG% --target INSTALL
```

### xmlrpc-c

```
cd %IFM3D_BUILD_DIR%
git clone --branch 1.33.14-cmake https://github.com/ifm/xmlrpc-c.git
cd %IFM3D_BUILD_DIR%\xmlrpc-c
mkdir build
cd build
cmake -G %IFM3D_CMAKE_GENERATOR% -DBUILD_SHARED_LIBS=ON -DCMAKE_INSTALL_PREFIX=%IFM3D_
↪BUILD_DIR%\install ..
cmake --build . --clean-first --config %CONFIG% --target INSTALL
```

## glog

```
cd %IFM3D_BUILD_DIR%
git clone --branch v0.3.5 https://github.com/google/glog.git
cd %IFM3D_BUILD_DIR%\glog
mkdir build
cd build
cmake -G %IFM3D_CMAKE_GENERATOR% -DBUILD_SHARED_LIBS=ON -DCMAKE_INSTALL_PREFIX=%IFM3D_
↪BUILD_DIR%\install ..
cmake --build . --clean-first --config %CONFIG% --target INSTALL
```

## gtest

```
cd %IFM3D_BUILD_DIR%
git clone --branch release-1.8.1 https://github.com/google/googletest.git
```

NOTE: gtest is only needed to build and run unit tests. To skip, add -DBUILD\_TESTS=OFF to the cmake configuration command line on the ifm3d library below.

## Building ifm3d

```
# Clone the repository
cd %IFM3D_BUILD_DIR%
git clone https://github.com/ifm/ifm3d.git
cd %IFM3D_BUILD_DIR%\ifm3d

# Configure
mkdir build
cd build
cmake -G %IFM3D_CMAKE_GENERATOR% -DCMAKE_WINDOWS_EXPORT_ALL_SYMBOLS=ON -DBUILD_SDK_-
↪PKG=ON -DGTEST_CMAKE_DIR=%IFM3D_BUILD_DIR%\googletest\googletest -Dgtest_force_-
↪shared_crt=TRUE -DCMAKE_PREFIX_PATH=%IFM3D_BUILD_DIR%\install;%IFM3D_OPENCV_PATH% -
↪DBOOST_ROOT=%IFM3D_BOOST_ROOT% -DBoost_USE_STATIC_LIBS=ON -DCMAKE_BUILD_TYPE=%CONFIG
↪% -DCMAKE_INSTALL_PREFIX=%IFM3D_BUILD_DIR%\install ..

# run tests
cmake --build . --config %CONFIG% --target check

#install
cmake --build . --config %CONFIG% --target INSTALL
```

## Running the ifm3d command line tool

After Building ifm3d, the binary files will be installed at %IFM3D\_BUILD\_DIR%\install\bin. To run the ifm3d tool you need to add this directory to your path. You will also need to add the opencv directory to your path.

If built targeting Visual Studio 2017/2019:

```
set PATH=%IFM3D_BUILD_DIR%\install\bin;%IFM3D_OPENCV_PATH%\x64\vc15\bin;%PATH%
```

After that you should be able to run the ifm3d tool

```
ifm3d
```

## Appendix A: Building without PCL

The ifm3d library offers an alternative image buffer implementation which only depends on OpenCV, thus eliminating a dependency on PCL. In order to build ifm3d without a dependency on PCL, the following modifications to the instructions above are necessary.

### Select the OpenCV Image Container

The ifm3d::ImageBuffer module (has dependency on PCL) must be disabled with the flag -DBUILD\_MODULE\_IMAGE=OFF and the ifm3d::OpenCVBuffer module must be enabled with the flag -DBUILD\_MODULE\_OPENCV=ON. The full cmake configuration command is:

```
cmake -G %IFM3D_CMAKE_GENERATOR% -DCMAKE_WINDOWS_EXPORT_ALL_SYMBOLS=ON -DBUILD_SDK_
PKG=ON -DGTEST_CMAKE_DIR=%IFM3D_BUILD_DIR%\googletest\googletest -Dgtest_force_
shared_crt=TRUE -DCMAKE_PREFIX_PATH=%IFM3D_BUILD_DIR%\install;%IFM3D_OPENCV_PATH% -
DBOOST_ROOT=%IFM3D_BOOST_ROOT% -DBoost_USE_STATIC_LIBS=ON -DCMAKE_BUILD_TYPE=%CONFIG%
% -DCMAKE_INSTALL_PREFIX=%IFM3D_BUILD_DIR%\install -DBUILD_MODULE_IMAGE=OFF -DBUILD_
MODULE_OPENCV=ON ..
```

## Python installation

Note: We recommend for testing purposes to install the ifm3dpy package in a clean python environment first. You can use `python -m venv "venv-name"` to create a new virtual environment.

You can use the official PyPI package to install the ifm3dpy within your virtual environment:

```
pip install ifm3dpy
```

Now, you can check your installation.

### Check the ifm3dpy installation

Let's verify quickly that the installation worked! This command should display the list of packages installed in your environment:

```
pip freeze
```

Open up a python shell with:

```
python.exe
OR
./python.exe
OR
python
```

Then try importing the package:

```
import ifm3dpy
print(ifm3dpy.__version__)
>>>0.91.0
```

You can test the connection from VPU to camera head with following lines:

```
from ifm3dpy import O3RCamera
o3r = O3RCamera()
config = o3r.get() #get the configuration saved on the VPU
```

Using the package json provides an easier tool for displaying JSON-Strings. The configuration from the VPU is always a JSON-String (output below shortened for display purposes).

```
import json
print(json.dumps(config, indent=4))
>>>{
    "device": {
        "clock": {
            "currentTime": 1581090739817663072
        },
        "diagnostic": {
            "temperatures": [],
            "upTime": 94000000000
        },
        "info": {
            "device": "0301",
            "deviceTreeBinaryBlob": "tegra186-quill-p3310-1000-c03-00-base.dtb",
            "features": {},
            "name": "TableTop2",
            "partNumber": "M03975",
            "productionState": "AA",
            "serialNumber": "000201234176",
            "vendor": "0001"
        },
        "network": {
            "authorized_keys": "",
            "ipAddressConfig": 0,
            "macEth0": "00:04:4B:EA:9F:D1",
            "macEth1": "00:02:01:23:41:76",
            "networkSpeed": 1000,
            "staticIPv4Address": "192.168.0.69",
            "staticIPv4Gateway": "192.168.0.201",
            "staticIPv4SubNetMask": "255.255.255.0",
            "useDHCP": false
        },
        "state": {
            "errorMessage": "",
            "errorNumber": ""
        },
        "swVersion": {
            "kernel": "4.9.140-l4t-r32.4+gc35f5eb9d1d9",
            "l4t": "r32.4.3",
            "os": "0.13.13-221",
            "schema": "v0.1.0",
            "swu": "0.15.12"
        }
    }
},
```

(continues on next page)

(continued from previous page)

```

"ports": {
    "port0": {
        "acquisition": {
            "framerate": 10.0,
            "version": {
                "major": 0,
                "minor": 0,
                "patch": 0
            }
        },
        "data": {
            "algoDebugConfig": {},
            "availablePCICOutput": [],
            "pcicTCPPort": 50010
        },
        "info": {
            "device": "2301",
            "deviceTreeBinaryBlobOverlay": "001-ov9782.dtbo",
            "features": {
                "fov": {
                    "horizontal": 127,
                    "vertical": 80
                },
                "resolution": {
                    "height": 800,
                    "width": 1280
                },
                "type": "2D"
            },
            "name": "",
            "partNumber": "M03976",
            "productionState": "AA",
            "sensor": "OV9782",
            "sensorID": "OV9782_127x80_noIllu_Csample",
            "serialNumber": "000000000281",
            "vendor": "0001"
        },
        "mode": "experimental_autoexposure2D",
        "processing": {
            "extrinsicHeadToUser": {
                "rotX": 0.0,
                "rotY": 0.0,
                "rotZ": 0.0,
                "transX": 0.0,
                "transY": 0.0,
                "transZ": 0.0
            },
            "version": {
                "major": 0,
                "minor": 0,
                "patch": 0
            }
        },
        "state": "RUN"
    },
    ...
}

```

## Docker dev containers

Development containers are available. They are built nightly with the latest version of ifm3d available on the o3r/main-next branch. You can pull them using the following command:

```
$ docker pull ghcr.io/ifm/ifm3d:latest
```

For more detailed documentation on using docker containers with the O3R platform, you can refer to this section.

### 2.2.2 Stable release

#### Ubuntu Linux via Apt (amd64/arm64)

□ The provided apt repositories are experimental and shall be used with caution, the version uploaded to the apt repository might change and thus may break your use-case. If you rely on a specific version of the software we do recommend to run your own apt repository or build from source.

We provide apt repositories for the following Ubuntu Linux distributions and architectures:

Add the repository to your sources.list:

```
$ sudo sh -c 'echo "deb [arch=$(dpkg --print-architecture)] https://nexus.ifm.com/
↪repository/ifm-robotics_ubuntu_${lsb_release -sc}_${dpkg --print-architecture}
↪${lsb_release -sc} main" > /etc/apt/sources.list.d/ifm-robotics.list'
```

Add the public key for the repository:

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key
↪8AB59D3A2BD7B692
```

If you experience issues with connecting the key server you can try this alternative which uses curl. This is maybe helpful when you are behind a proxyserver.

```
curl -sSL 'http://keyserver.ubuntu.com/pks/lookup?op=get&search=0x8AB59D3A2BD7B692' | 
↪sudo apt-key add -
```

:exclamation: In case of any name resolution issues, it is worth to check the environment variable `$https_proxy` for proper proxy configuration.

Install the software:

```
$ sudo apt-get update
$ sudo apt-get install ifm3d-camera \
    ifm3d-framegrabber \
    ifm3d-swupdater \
    ifm3d-image \
    ifm3d-opencv \
    ifm3d-pcicclient \
    ifm3d-tools \
    ifm3d-python3 \
    ifm3d-pcl-viewer \
```

## Linux for Tegra

□ The provided apt repositories are experimental and shall be used with caution, the version uploaded to the apt repository might change and thus may break your use-case. If you rely on a specific version of the software we do recommend to run your own apt repository or build from source.

Linux for Tegra is an NVIDIA Linux distribution for the Jetson family of GPU SoC systems. NVIDIA distributes a software package called JetPack with various utilities and libraries optimized for the target hardware. There are a few packages which override the core Ubuntu packages (OpenCV as the primary example). We provide alternate apt repositories for ifm3d built on top of the JetPack libraries rather than the Ubuntu libraries.

Add one of the following repositories based on your desired JetPack/L4T release:

Jetpack 4.4:

```
$ sudo sh -c 'echo "deb https://nexus.ifm.com/repository/ifm-robotics_l4t_jetpack_4_4_
↪arm64 melodic main" > /etc/apt/sources.list.d/ifm-robotics.list'
```

Jetpack 4.3:

```
$ sudo sh -c 'echo "deb https://nexus.ifm.com/repository/ifm-robotics_l4t_jetpack_4_3_
↪arm64 melodic main" > /etc/apt/sources.list.d/ifm-robotics.list'
```

Add the public key for the repository:

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key ↪
↪8AB59D3A2BD7B692
```

Install the software:

```
$ sudo apt-get update
$ sudo apt-get install ifm3d-camera \
    ifm3d-framegrabber \
    ifm3d-swupdater \
    ifm3d-image \
    ifm3d-opencv \
    ifm3d-pciclient \
    ifm3d-tools \
    ifm3d-python3 \
    ifm3d-pcl-viewer \
```

## ROS/ROS2

For users interested in using our ROS bindings, ifm3d and ifm3d-ros are both available in the ROS distribution for Kinetic and Melodic (Noetic coming shortly).

```
$ sudo apt install ros-kinetic-ifm3d
```

or

```
$ sudo apt install ros-melodic-ifm3d
```

For users interested in using our ROS2 bindings, binaries will be included (starting with dashing) very soon. For now, packages must be built from source. Do not use the debian

mirror for ifm3d since (depending on version) ROS2 ships parallel versions of some core libraries (OpenCV, PCL) as compared with standard Ubuntu. ifm3d must be built against the proper dependencies.

## 2.3 Basic Library Usage

### 2.3.1 First steps with ifm3d/ifm3dpy

After installing the ifm3d/ifm3dpy library, you most likely want to receive an image as fast and possible. The next steps will guide you through the process. You will find more information in the [ifm3d API documentation](#).

#### RUN/CONF/IDLE

**The default configuration of all heads is “CONF”. You cannot receive any images before changing to “RUN”.**

To set a head into "RUN", the configuration of the VPU needs to be changed. This change needs to be uploaded to the VPU afterwards.

Please refer to the [this section](#) for more information.

Following steps describe the change of the state of a single head.

```
from ifm3dpy import O3RCamera

o3r = O3RCamera()
config = o3r.get()
config['ports'][‘port0’][‘state’] = “RUN” #Expecting a head on Port 0
o3r.set(config)
```

Note: Depending where your imager is connected, 'port0' might not work.

#### Receive an image

If the head is in "RUN", it is possible to receive images.

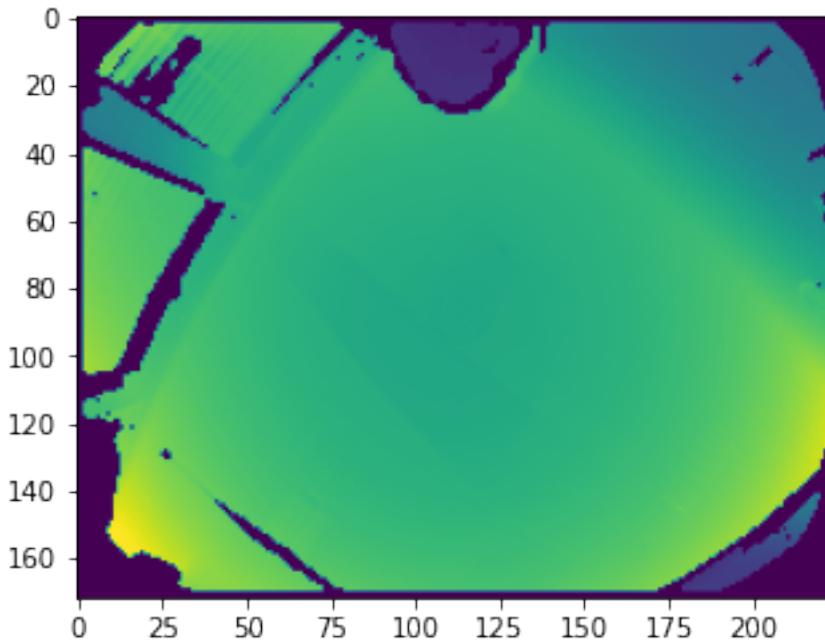
Please refer to [this section](#) for more information.

To display the image directly, we use matplotlib.

```
from ifm3dpy import O3RCamera, FrameGrabber, ImageBuffer
import matplotlib.pyplot as plt

o3r = O3RCamera()
fg = FrameGrabber(o3r, pcic_port=50010) #Expecting a head on Port 0 (Port 0 == 50010)
im = ImageBuffer()

if fg.wait_for_frame(im, 1000):
    plt.imshow(im.distance_image())
```



### 2.3.2 How to: configure the camera

The O3R has multiple parameters that have an influence on the point cloud. Some of them affect the raw measurement and others modify how the data is converted into x,y,z, etc values. These parameters can be changed to better fit your applications and we are going to see how here. You can refer to [this page](#) for a detailed description of each parameter.

There are multiple functions available to read the current configuration of the device and to set a new one. We are using JSON formatting.

For this process, we have to initialize the camera object (please have a look at the code example provided for full details of the imported libraries).

Python

```
o3r = O3RCamera()
```

C++

```
auto cam = std::make_shared<ifm3d::O3RCamera>();
```

Note: if you are using multiple ifm devices (O3D, O3X, O3R), you can use the CameraBase class.

Python

```
cam = CameraBase()
```

C++

If you need to use Device specific functions at a later point you can cast the pointer to the relevant class:

```
auto cam = ifm3d::CameraBase::MakeShared();
auto cam_03R = std::static_pointer_cast<ifm3d::O3RCamera>(cam);
```

### Read the current configuration

The first provided function outputs the current configuration of the device (the VPU and each head currently attached). This function outputs the full configuration, including the parameters set for each camera head, but also other aspects like MAC and IP addresses, etc.

Python

```
conf = cam.get();
```

C++

```
json conf = cam->Get();
```

### Write a new configuration

To write a new configuration to the device, you need to provide said configuration in json formatting. The provided configuration can be a subset or the full configuration.

Python

```
o3r.set({'device':{'info':{'name':'great_o3r'}}})
```

C++

```
cam->Set(R"( {"device": {"info": {"name": "my_o3r"}}})");
```

Note: we use `string` literals for easier readability.

To make the configuration persistent over reboots, you need to use the following function:

Python

```
o3r.save_init()
```

C++

```
cam->SaveInit();
```

### The full example

Python

```
import json

# Define the ifm3d objects for the communication
from ifm3dpy import O3RCamera
o3r = O3RCamera()
```

(continues on next page)

(continued from previous page)

```
# Get the current configuration
config = o3r.get()

# Print a little part from the config to verify the configuration
print(json.dumps(config['device']['swVersion'], indent=4))
# Note: this assumes that a camera is plugged into port 1
print(config['ports']['port1']['state'])

# Let's change the name of the device
o3r.set({'device':{'info':{'name':'great_o3r'}}})
o3r.save_init()

# Double check the configuration
config = o3r.get()
print(config['device']['info']['name'])
```

C++

```
/*
 * Copyright 2021-present ifm electronic, gmbh
 * SPDX-License-Identifier: Apache-2.0
 */

#include <iostream>
#include <iomanip>
#include <memory>
#include <fstream>
#include <ifm3d/camera/camera_o3r.h>
using json = nlohmann::json;

int main(){

    // Create the camera object
    auto cam = std::make_shared<ifm3d::O3RCamera>();

    // Get the current configuration of the camera in JSON format
    json conf = cam->Get();

    // Display the current configuration
    std::cout << std::setw(4) << conf << std::endl;

    // Configure the device from a json string
    cam->Set(json::parse(R"({"device":{"info":{"name": "my_new_o3r"}}})"));
    // Make the configuration persistent
    cam->SaveInit();

    return 0;
}
```

### 2.3.3 How to: receive an image

A primary objective of ifm3d is to make it as simple and performant as possible to acquire pixel data from an ifm 3D camera. Additionally, those data should be encoded in a useful format for performing computer vision and/or robotics perception tasks. A typical ifm3d client program will follow the structure of a control loop whereby images are continuously acquired from the camera and acted upon in some application-specific way.

At the end of this ‘how to’, you should be able to receive images and know the basic usage of the O3RCamera, FrameGrabber and StlImageBuffer classes.

Note: for O3D or O3X devices, simply use the Camera class in place of the O3RCamera in the following code.

#### O3RCamera, FrameGrabber and StlImageBuffer

ifm3d provides three main classes:

- O3RCamera holds the configuration of the camera heads, handles the connection, etc;
- FrameGrabber receives frames (images);
- StlImageBuffer stores the image data.

Instantiating these objects is as follows:

Python

```
cam = O3RCamera()
fg = FrameGrabber(o3r, pcic_port=50012)
im = ImageBuffer()
```

C++

```
auto cam = std::make_shared<ifm3d::O3RCamera>();
auto fg = std::make_shared<ifm3d::FrameGrabber>(cam, ifm3d::DEFAULT_SCHEMA_MASK, 50012);
auto im = std::make_shared<ifm3d::StlImageBuffer>();
```

The O3RCamera class, counter-intuitively, refers to the computing unit (the VPU). It inherits its name from previous ifm 3D devices that only used one camera, with no distinction between sensing and computing units. You can input:

- ip: the IP address of the device;
- xmlrpc\_port: the XML\_RPC port (it is fixed at the moment);
- password: the password to connect to the device (unused for the O3R).

The FrameGrabber stores a reference to the passed in camera shared pointer and starts a worker thread to stream in pixel data from the device. Its inputs:

- cam: The camera instance (the image processing platform) that handles the connection to the camera heads;
- mask: A bitmask encoding the image acquisition schema to stream in from the camera;
- port: Port number of the camera head to grab data from (not the physical port number);

The StlImageBuffer class simply serves to store the received data. It allocates space for each individual image.

Note: instantiating the objects is done the same way for any imager type (2D, 3D, different resolutions, etc).

Note that `ifm3d` encourages the use of `std::shared_ptr` as a means to manage the ownership and lifetimes of the core actors in an `ifm3d` program. Indeed, you will notice that there is no need to explicitly allocate and deallocate memory. To instantiate the `ifm3d::O3RCamera` object (or `ifm3d::Camera`), a call to `ifm3d::O3RCamera::MakeShared()` is made (or `ifm3d::Camera::MakeShared()`), rather than calling `std::make_shared` directly. This wrapper function is used to handle direct hardware probing to determine the type of camera that is connected. For example, this may be an O3R, O3D303, O3X, or something else. Regardless, a `std::shared_ptr` is returned from this call.

## Receive an image

You just need to call the `WaitForFrame` function. Input an `ImageBuffer` object as well as a timeout value in ms. Make sure the camera head is in “RUN” mode.

Python

```
fg.wait_for_frame(im, 1000)
```

C++

```
fg->WaitForFrame(im.get(), 1000);
```

And you’re good to go! Now you can do something with all this data.

Note : The `WaitForFrame` method is called on our `ifm3d::FrameGrabber` pointer. It is passed the raw pointer to the `ifm3d::ImageBuffer` and a timeout in milliseconds. If a new frame from the camera cannot be acquired within the timeout, `WaitForFrame` will return false. For the curious, we pass the raw pointer to the `ImageBuffer` as opposed to the smart pointer because it is more performant (the `std::shared_ptr` reference count does not need to be incremented and decremented with every call to `WaitForFrame` and, semantically, the `FrameGrabber` via the `WaitForFrame` call does not participate in the ownership of the `ImageBuffer`).

## Access the data

Accessing the received data is done through the `StlImageBuffer`. Different data types are available depending on whether the camera is a 2D or a 3D camera. Simply access the image like so:

Python

```
# For 3D data:
dist = im.distance_image();
# For 2D data:
rgb = im.jpeg_image();
```

C++

```
// For 3D data:
auto dist = im->DistanceImage();
// For 2D data:
auto rgb = im->JPEGImage();
```

## The full example

Python

```
from ifm3dpy import O3RCamera, ImageBuffer, FrameGrabber

# Initialize the objects
o3r = O3RCamera('192.168.0.69')
port='port2'
fg = FrameGrabber(o3r, pcic_port=50012)
im = ImageBuffer()

# Get a frame
if fg.wait_for_frame(im, 500)==False:
    raise ValueError #Exception('fg-timeout on ' + port + ' reached')

# Read the distance image and display a pixel in the center
dist = im.distance_image()
(width, height) = dist.shape
print(dist[width//2,height//2])
```

C++

```
/*
 * Copyright 2021-present ifm electronic, gmbh
 * SPDX-License-Identifier: Apache-2.0
 */
#include <iostream>
#include <ifm3d/camera/camera_o3r.h>
#include <ifm3d/fg.h>
#include <ifm3d/stlimage.h>
#include <ifm3d/fg/distance_image_info.h>

int main(){

    /////////////////
    // Declare the objects:
    ///////////////
    // Declare the device object (one object only, corresponding to the VPU)
    auto cam = std::make_shared<ifm3d::O3RCamera>();
    // Declare the FrameGrabber and ImageBuffer objects.
    // One FrameGrabber per camera head (define the port number).
    const auto FG_PCIC_PORT = cam->Get()["/ports/port2/data/pcicTCPPort"_json_
    pointer];
    auto fg = std::make_shared<ifm3d::FrameGrabber>(cam, ifm3d::DEFAULT_SCHEMA_MASK,_
    FG_PCIC_PORT);
    auto im = std::make_shared<ifm3d::StlImageBuffer>();

    /////////////////
    // Get a frame:
    ///////////////
    if (! fg->WaitForFrame(im.get(), 3000))
    {
        std::cerr << "Timeout waiting for camera!" << std::endl;
        return -1;
    }
}
```

(continues on next page)

(continued from previous page)

```
///////////
// Example for 3D data:
///////////
auto dist = im->DistanceImage();

std::cout << dist.height() << " " << dist.width() << std::endl;

return 0;
}
```

### 2.3.4 How to: receive data from multiple heads

```
/*
 * Copyright 2021-present ifm electronic, gmbh
 * SPDX-License-Identifier: Apache-2.0
 */

/*How to: receive data from multiple heads
One feature of the O3R platform is to enable the use of multiple camera heads of ↴
different types (2D, 3D, various resolutions, etc).
In this example, we show how to retrieve the pcic port number for each head ↴
connected to the VPU along with its type and create `FrameGrabber` and ↴
`ImageBuffer` objects for each.
*/
#include <ifm3d/camera/camera_o3r.h>
#include <iostream>
#include <iomanip>
#include <ifm3d/fg.h>
#include <ifm3d/stlimage.h>

// This function formats the timestamps for proper display
// a.k.a converts to local time
std::string formatTimestamp(ifm3d::TimePointT timestamp)
{
    using namespace std::chrono;
    std::time_t time = std::chrono::system_clock::to_time_t(
        std::chrono::time_point_cast<std::chrono::system_clock::duration>(
            timestamp));

    milliseconds milli = duration_cast<milliseconds>(
        timestamp.time_since_epoch() - duration_cast<seconds>(
            timestamp.time_since_epoch()));

    std::ostringstream s;
    s << std::put_time(std::localtime(&time), "%Y-%m-%d %H:%M:%S")
      << ":" << std::setw(3) << std::setfill('0') << milli.count();

    return s.str();
}
```

(continues on next page)

(continued from previous page)

```

int main(){

    // Declare the camera object
    auto cam = std::make_shared<ifm3d::O3RCamera>();
    // Retreive ports configuration
    json conf = cam->Get();
    // Initialize the structures
    std::vector<ifm3d::FrameGrabber::Ptr> fgs;
    auto im = std::make_shared<ifm3d::StlImageBuffer>();

    std::cout << "Available connections:" << std::endl;

    for (const auto& port : conf["ports"].items())
    {
        // Create lists of connected PCIC ports along with types
        nlohmann::json::json_pointer p1("/ports/" + port.key() + "/data/pcicTCPPort");
        const auto pcic = conf[p1];
        nlohmann::json::json_pointer p2("/ports/" + port.key() + "/info/features/type");
        const auto type = conf[p2];
        //Display connected port with type
        std::cout << "Port: " << port.key()
            << "\t PCIC: " << pcic
            << "\t Type: " << type << std::endl;
        // Create list of FrameGrabber and ImageBuffer objects for connected ports
        auto fg = std::make_shared<ifm3d::FrameGrabber>(cam, ifm3d::DEFAULT_SCHEMA_
MASK, pcic);
        fgs.push_back(fg);
    }

    // Grab frames from each heads
    // The timestamp has two parts, the timestamp in seconds and the timestamp in
    //nanoseconds
    for (auto fg : fgs)
    {
        if (! fg->WaitForFrame(im.get(), 3000))
        {
            std::cerr << "Timeout waiting for camera!" << std::endl;
            return -1;
        }

        std::cout << "Timestamp of frame "
            << std::setw(2) << std::setfill('0')
            << ":" << formatTimestamp(im->TimeStamp())
            << std::endl;
    }

    return 0;
}

```

## 2.4 ifm3d - Command Line Tool

### 2.4.1 Overview

ifm3d ships with a command line tool of the same name. The ifm3d command line tool is used to both introspect the state of a camera as well as mutate parameters. To carry out a particular task, you evoke one of the ifm3d subcommands. To get a listing of available subcommands, you can pass the `--help` option.

```
$ ifm3d --help
ifm3d: version=0.20.0

Usage:
ifm3d [<global options>] <command> [<args>]

global options:
-h, --help           Produce this help message and exit
--ip arg            IP address of the sensor (default: 192.168.0.69)
--xmlrpc-port arg  XMLRPC port of the sensor (default: 80)
--password arg      Password for establishing an edit-session with the
                    sensor (default: )
```

These are common commands used in various situations:

<code>app-types</code>	List the application types supported by the sensor.
<code>config</code>	Configure sensor settings from a JSON description of the desired sensor state. See also `dump'.
<code>cp</code>	Create a new application on the sensor, bootstrapped from a copy of an existing one.
<code>discover</code>	Discover ifm devices on the network.
<code>dump</code>	Serialize the sensor state to JSON.
<code>export</code>	Export an application or whole sensor configuration into a format compatible with ifm Vision Assistant.
<code>hz</code>	Compute the actual frequency at which the FrameGrabber is running.
<code>imager-types</code>	List the imager types supported by the sensor.
<code>import</code>	Import an application or whole sensor configuration that is compatible with ifm Vision Assistant's export format.
<code>jitter</code>	Collects statistics on framegrabber (and optionally, image construction) jitter.
<code>ls</code>	Lists the applications currently installed on the sensor.
<code>passwd</code>	Sets the password on the sensor.

(continues on next page)

(continued from previous page)

reboot	Reboot the sensor, potentially into recovery mode (no recovery mode for O3R). Recovery mode is useful for putting the sensor into a state where it can be flashed with new firmware.
reset	Reset the sensor to factory defaults.
rm	Deletes an application from the sensor.
schema	Construct and analyze image acquisition schema masks.
swupdate	Perform a firmware update on the camera. Please ensure that the camera is booted to recovery beforehand.
time	Get/set the current time on the camera.
trace	Get trace messages from the internal camera trace buffer.

For bug reports, please see:  
<https://github.com/ifm/ifm3d/issues>

As it is reported in the help output above, the `ifm3d` command line program accepts 1) a set of global arguments which control the particular platform you wish to communicate with; 2) a subcommand; and 3) arguments to the subcommand. To get a listing of the particular arguments accepted by a subcommand, you can pass the `--help` option to the subcommand. For exemplary purposes, let's list the options accepted by the `cp` subcommand.

```
$ ifm3d cp --help
Usage:
  ifm3d [<global options>] cp [<cp options>]

  global options:
    -h, --help            Produce this help message and exit
    --ip arg              IP address of the sensor (default: 192.168.0.69)
    --xmlrpc-port arg    XMLRPC port of the sensor (default: 80)
    --password arg        Password for establishing an edit-session with the
                          sensor (default: )

  cp options:
    --index arg           Index of source application to copy (default: -1)
```

As is shown above, `cp` takes a source application index to copy from. Note that the concept of applications is deprecated for the O3R platform.

We now walk through a couple of simple examples of using `ifm3d`. This is not an exhaustive tutorial on `ifm3d` but rather intended to give a sense of how to use the tool. The concepts apply broadly to all of the subcommands.

## 2.4.2 Camera and Imager Configuration

Configuring the parameters of an ifm 3D camera is accomplished in ifm3d in one of two ways: 1) via the `ifm3d` command line tool; 2) via the `ifm3d` library's camera module API. We show below how to do so with the command line tool. Please refer to the `../examples/o3r/index` for instructions on configuring the camera through `ifm3d` library.

The primary mechanism for using the `ifm3d` command line tool to configure an ifm 3D camera is to utilize the `dump` and `config` subcommands to `ifm3d`. The `dump` command serializes the camera state to JSON and prints it to `stdout`, while the `config` subcommand consumes (either from a file or `stdin`) the same JSON serialization but interprets that JSON as a desired state for the camera. Said another way, `ifm3d config` will make a best effort attempt to have the actual camera hardware reflect the camera state encoded in the JSON stream passed to it.

The remainder of this document will contain a set of examples and associated narrative in hopes of demonstrating how to leverage `ifm3d` to configure your 3D camera. For purposes of this document, an ifm O3D303 will be utilized. However, the techniques shown here apply to any supported `ifm3d` camera (e.g., O3X). Additionally, since the camera state is serialized via JSON, some of the examples below will utilize the `jq` command line JSON processor to build up Linux pipelines to carry out a specific task. The usage of `jq` is not required. Standard Linux tools (`grep`, `sed`, `awk`, `perl`, `python`, etc.) could also be used or a single pipeline can be decomposed into multiple commands whereby data are serialized to a file, edited, then I/O redirected into `ifm3d config` in discrete steps. Again, the remainder of this document will assume `jq` is available. (To install `jq` on Ubuntu: `sudo apt-get install jq`).

### Dump

Serializing the current state of the camera is accomplished through the `ifm3d dump` command. Exemplary output is shown below:

O3D example

```
$ ifm3d dump
{
  "ifm3d": {
    "Apps": [
      {
        "Description": "",
        "Id": "476707713",
        "Imager": {
          "AutoExposureMaxExposureTime": "10000",
          "AutoExposureReferencePointX": "88",
          "AutoExposureReferencePointY": "66",
          "AutoExposureReferenceROI": "{\"ROIs\": [{\"id\":0,\"group\":0,\"type\":\"\\
→\"Rect\", \"width\":130, \"height\":100, \"angle\":0, \"center_x\":88, \"center_y\":66}]}",
          "AutoExposureReferenceType": "0",
          "Channel": "0",
          "ClippingBottom": "131",
          "ClippingCuboid": "{\"XMin\": -3.402823e+38, \"XMax\": 3.402823e+38, \"YMin\\
→\": -3.402823e+38, \"YMax\": 3.402823e+38, \"ZMin\": -3.402823e+38, \"ZMax\": 3.
→402823e+38}",
          "ClippingLeft": "0",
          "ClippingRight": "175",
          "ClippingTop": "0",
          "ContinuousAutoExposure": "false",
        }
      }
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```

    "EnableAmplitudeCorrection": "true",
    "EnableFastFrequency": "false",
    "EnableFilterAmplitudeImage": "true",
    "EnableFilterDistanceImage": "true",
    "EnableRectificationAmplitudeImage": "false",
    "EnableRectificationDistanceImage": "false",
    "ExposureTime": "5000",
    "ExposureTimeList": "125;5000",
    "ExposureTimeRatio": "40",
    "FrameRate": "5",
    "MaxAllowedLEDFrameRate": "23.2",
    "MinimumAmplitude": "42",
    "Resolution": "0",
    "SpatialFilter": {},
    "SpatialFilterType": "0",
    "SymmetryThreshold": "0.4",
    "TemporalFilter": {},
    "TemporalFilterType": "0",
    "ThreeFreqMax2FLineDistPercentage": "80",
    "ThreeFreqMax3FLineDistPercentage": "80",
    "TwoFreqMaxLineDistPercentage": "80",
    "Type": "under5m_moderate",
    "UseSimpleBinning": "false"
  },
  "Index": "1",
  "LogicGraph": "{\"IOMap\": {\"OUT1\": \"RFT\", \"OUT2\": \"AQUFIN\"}, \"blocks\":\n  \"B00001\": {\"pos\": {\"x\": 200, \"y\": 200}, \"properties\": {}, \"type\": \"PIN_EVENT_IMAGE_ACQUISITION_FINISHED\"},\n  \"B00002\": {\"pos\": {\"x\": 200, \"y\": 75}, \"properties\": {}, \"type\": \"PIN_EVENT_READY_FOR_TRIGGER\"},\n  \"B00003\": {\"pos\": {\"x\": 600, \"y\": 75}, \"properties\": {\"pulse_duration\": 0}, \"type\": \"DIGITAL_OUT1\"},\n  \"B00005\": {\"pos\": {\"x\": 600, \"y\": 200}, \"properties\": {\"pulse_duration\": 0}, \"type\": \"DIGITAL_OUT2\"}},\n  \"connectors\": {\"C00000\": {\"dst\": \"B00003\", \"dstEP\": 0, \"src\": \"B00002\", \"srcEP\": 0},\n  \"C00001\": {\"dst\": \"B00005\", \"dstEP\": 0, \"src\": \"B00001\", \"srcEP\": 0}}},\n  \"Name\": \"Sample Application\",\n  \"PcicEipResultSchema\": {\"layouter\": \"flexible\", \"format\": { \"dataencoding\": \"binary\", \"order\": \"big\" }, \"elements\": [ { \"type\": \"string\", \"value\": \"star\", \"id\": \"start_string\" }, { \"type\": \"records\", \"id\": \"models\", \"elements\": [ { \"type\": \"int16\", \"id\": \"boxFound\" }, { \"type\": \"int16\", \"id\": \"width\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"height\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"length\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"xMidTop\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"yMidTop\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"zMidTop\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"yawAngle\" }, { \"type\": \"int16\", \"id\": \"qualityLength\" }, { \"type\": \"int16\", \"id\": \"qualityWidth\" }, { \"type\": \"int16\", \"id\": \"qualityHeight\" } ] }, { \"type\": \"string\", \"value\": \"stop\", \"id\": \"end_string\" } ] },\n  \"PcicPnioResultSchema\": {\"layouter\": \"flexible\", \"format\": { \"dataencoding\": \"binary\", \"order\": \"big\" }, \"elements\": [ { \"type\": \"string\", \"value\": \"star\", \"id\": \"start_string\" }, { \"type\": \"records\", \"id\": \"models\", \"elements\": [ { \"type\": \"int16\", \"id\": \"boxFound\" }, { \"type\": \"int16\", \"id\": \"width\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"height\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"length\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"xMidTop\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"yMidTop\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"zMidTop\", \"format\": { \"scale\": 1000 } }, { \"type\": \"int16\", \"id\": \"yawAngle\" }, { \"type\": \"int16\", \"id\": \"qualityLength\" }, { \"type\": \"int16\", \"id\": \"qualityWidth\" }, { \"type\": \"int16\", \"id\": \"qualityHeight\" } ] }, { \"type\": \"string\", \"value\": \"stop\", \"id\": \"end_string\" } ] },\n  \"continues on next page\"\n}

```

(continued from previous page)

```

    "PcicTcpResultSchema": "{ \"layouter\": \"flexible\", \"format\": { \
    \"dataencoding\": \"ascii\" }, \"elements\": [ { \"type\": \"string\", \"value\": \
    \"star\", \"id\": \"start_string\" }, { \"type\": \"blob\", \"id\": \"normalized_ \
    amplitude_image\" }, { \"type\": \"blob\", \"id\": \"distance_image\" }, { \"type\\ \
    \": \"blob\", \"id\": \"x_image\" }, { \"type\": \"blob\", \"id\": \"y_image\" }, { \
    \"type\": \"blob\", \"id\": \"z_image\" }, { \"type\": \"blob\", \"id\": \" \
    confidence_image\" }, { \"type\": \"blob\", \"id\": \"diagnostic_data\" }, { \
    \"type\": \"string\", \"value\": \"stop\", \"id\": \"end_string\" } ] }",
        "TemplateInfo": "",
        "TriggerMode": "1",
        "Type": "Camera"
    },
],
"Device": {
    "ActiveApplication": "1",
    "ArticleNumber": "03D303",
    "ArticleStatus": "AD",
    "Description": "",
    "DeviceType": "1:2",
    "EnableAcquisitionFinishedPCIC": "false",
    "EthernetFieldBus": "0",
    "EthernetFieldBusEndianness": "0",
    "EvaluationFinishedMinHoldTime": "10",
    "ExtrinsicCalibRotX": "0",
    "ExtrinsicCalibRotY": "0",
    "ExtrinsicCalibRotZ": "0",
    "ExtrinsicCalibTransX": "0",
    "ExtrinsicCalibTransY": "0",
    "ExtrinsicCalibTransZ": "0",
    "IODebouncing": "true",
    "IOExternApplicationSwitch": "0",
    "IOLogicType": "1",
    "IPAddressConfig": "0",
    "ImageTimestampReference": "1520963818",
    "Name": "New sensor",
    "OperatingMode": "0",
    "PNIODeviceName": "",
    "PasswordActivated": "false",
    "PcicProtocolVersion": "3",
    "PcicTcpPort": "50010",
    "SaveRestoreStatsOnApplSwitch": "true",
    "ServiceReportFailedBuffer": "15",
    "ServiceReportPassedBuffer": "15",
    "SessionTimeout": "30",
    "TemperatureFront1": "3276.7",
    "TemperatureFront2": "3276.7",
    "TemperatureIMX6": "40.7179985046387",
    "TemperatureIllu": "47.5",
    "UpTime": "1.619722222222222"
},
"Net": {
    "MACAddress": "00:02:01:40:7D:96",
    "NetworkSpeed": "0",
    "StaticIPv4Address": "192.168.0.69",
    "StaticIPv4Gateway": "192.168.0.201",
    "StaticIPv4SubNetMask": "255.255.255.0",
}

```

(continues on next page)

(continued from previous page)

```

    "UseDHCP": "false"
},
"Time": {
    "CurrentTime": "1520963816",
    "NTPServers": "",
    "StartingSynchronization": "false",
    "Stats": "",
    "SynchronizationActivated": "false",
    "Syncing": "false",
    "WaitSyncTries": "2"
},
"-": {
    "Date": "Mon May 7 11:55:08 2018",
    "HWInfo": {
        "Connector": "#!02_A300_C40_02452814_008023176",
        "Diagnose": "#!02_D322_C32_03038544_008023267",
        "Frontend": "#!02_F342_C34_17_00049_008023607",
        "Illumination": "#!02_I300_001_03106810_008001175",
        "MACAddress": "00:02:01:40:7D:96",
        "Mainboard": "#!02_M381_003_03181504_008022954",
        "MiraSerial": "0e30-59af-0ef7-0244"
    },
    "SWVersion": {
        "Algorithm_Version": "2.0.45",
        "Calibration_Device": "00:02:01:40:7d:96",
        "Calibration_Version": "0.9.0",
        "Diagnostic_Controller": "v1.0.69-9dbc4ca5ef-dirty",
        "ELDK": "GOLDENEYE_YOCTO/releases%2F03D%2FRB_1.20.x-7-  
06d9c894636352a6c93711c7284d02b0c794a527",
        "IFM_Recovery": "unversioned",
        "IFM_Software": "1.20.1138",
        "Linux": "Linux version 3.14.34-rt31-yocto-standard-00009-ge4ab4d94f288-dirty  
(jenkins@dettlx190) (gcc version 4.9.2 (GCC) ) #1 SMP PREEMPT RT Tue Mar 13  
16:06:07 CET 2018",
        "Main_Application": "unknown"
    },
    "ifm3d_version": 900
}
}
}
}

```

**O3R example**

```

$ ifm3d dump
{
  "device": {
    "clock": {
      "currentTime": 1581108383428570624
    },
    "diagnostic": {
      "temperatures": [],
      "upTime": 17739000000000
    },
    "info": {
      "device": "0301",
      "deviceTreeBinaryBlob": "tegra186-quill-p3310-1000-c03-00-base.dtb",
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

    "features": {},
    "name": "",
    "partNumber": "M03975",
    "productionState": "AA",
    "serialNumber": "000201234159",
    "vendor": "0001"
  },
  "network": {
    "authorized_keys": "",
    "ipAddressConfig": 0,
    "macEth0": "00:04:4B:EA:9F:35",
    "macEth1": "00:02:01:23:41:59",
    "networkSpeed": 1000,
    "staticIPv4Address": "192.168.0.69",
    "staticIPv4Gateway": "192.168.0.201",
    "staticIPv4SubNetMask": "255.255.255.0",
    "useDHCP": false
  },
  "state": {
    "errorMessage": "",
    "errorNumber": ""
  },
  "swVersion": {
    "kernel": "4.9.140-l4t-r32.4+gc35f5eb9d1d9",
    "l4t": "r32.4.3",
    "os": "0.13.13-221",
    "schema": "v0.1.0",
    "swu": "0.15.12"
  }
},
"ports": {
  "port0": {
    "acquisition": {
      "framerate": 10.0,
      "version": {
        "major": 0,
        "minor": 0,
        "patch": 0
      }
    },
    "data": {
      "algoDebugConfig": {},
      "availablePCICOutput": [],
      "pcicTCPPort": 50010
    },
    "info": {
      "device": "2301",
      "deviceTreeBinaryBlobOverlay": "001-ov9782.dtbo",
      "features": {
        "fov": {
          "horizontal": 127,
          "vertical": 80
        },
        "resolution": {
          "height": 800,
          "width": 1280
        }
      }
    }
  }
}

```

(continues on next page)

(continued from previous page)

```

        },
        "type": "2D"
    },
    "name": "",
    "partNumber": "M03933",
    "productionState": "AA",
    "sensor": "OV9782",
    "sensorID": "OV9782_127x80_noIllu_Csample",
    "serialNumber": "000000000280",
    "vendor": "0001"
},
"mode": "experimental_autoexposure2D",
"processing": {
    "extrinsicHeadToUser": {
        "rotX": 0.0,
        "rotY": 0.0,
        "rotZ": 0.0,
        "transX": 0.0,
        "transY": 0.0,
        "transZ": 0.0
    },
    "version": {
        "major": 0,
        "minor": 0,
        "patch": 0
    }
},
"state": "RUN"
},
"port2": {
    "acquisition": {
        "exposureLong": 5000,
        "exposureShort": 400,
        "framerate": 10.0,
        "offset": 0.0,
        "version": {
            "major": 0,
            "minor": 0,
            "patch": 0
        }
    },
    "data": {
        "algoDebugConfig": {},
        "availablePCICOutput": [],
        "pcicTCPPort": 50012
    },
    "info": {
        "device": "3101",
        "deviceTreeBinaryBlobOverlay": "001-irs2381c.dtbo",
        "features": {
            "fov": {
                "horizontal": 60,
                "vertical": 45
            },
            "resolution": {
                "height": 172,

```

(continues on next page)

(continued from previous page)

```

        "width": 224
    },
    "type": "3D"
},
"name": "",
"partNumber": "M03933",
"productionState": "AA",
"sensor": "IRS2381C",
"sensorID": "IRS2381C_60x45_4x2W_60x45_C2",
"serialNumber": "000000000280",
"vendor": "0001"
},
"mode": "standard_range4m",
"processing": {
    "diParam": {
        "anfFilterSizeDiv2": 2,
        "enableDynamicSymmetry": true,
        "enableStraylight": true,
        "enableTemporalFilter": true,
        "excessiveCorrectionThreshAmp": 0.3,
        "excessiveCorrectionThreshDist": 0.08,
        "maxDistNoise": 0.02,
        "maxSymmetry": 0.4,
        "medianSizeDiv2": 0,
        "minAmplitude": 20.0,
        "minReflectivity": 0.0,
        "mixedPixelFilterMode": 1,
        "mixedPixelThresholdRad": 0.15
    },
    "extrinsicHeadToUser": {
        "rotX": 0.0,
        "rotY": 0.0,
        "rotZ": 0.0,
        "transX": 0.0,
        "transY": 0.0,
        "transZ": 0.0
    },
    "version": {
        "major": 0,
        "minor": 0,
        "patch": 0
    }
},
"state": "RUN"
}
}
}

```

In the example above, we serialize the entire state of the camera. This is useful to, for example, save to a file, edit, and push out to one or more cameras. However, sometimes we need to look at the camera configuration to simply answer a question we may have. For example, if we wanted to see which version of the firmware the camera is running we could issue the following command.

```
$ ifm3d dump | jq .ifm3d._.SWVersion.IFM_Software
"1.20.1138"
```

It follows that the entire JSON serialized configuration may be further processed either programmatically or manually via a text editor.

## Config

Mutating parameters on the camera is done by creating a desired camera state encoded in JSON compliant to the output produced by `ifm3d dump`. For example, if we wanted to change the framerate of the first application on the camera we could do the following.

1. Before changing the framerate, let's see what it is currently set to:

(NOTE: This step is not necessary. We do this to illustrate the state of the camera prior to mutating the parameter).

```
$ ifm3d dump | jq .ifm3d.Apps[0].Imager.FrameRate  
"5"
```

We see the current framerate is 5 fps.

1. Let's set it to 10 fps:

```
$ ifm3d dump | jq '.ifm3d.Apps[0].Imager.FrameRate="10"' | ifm3d config
```

1. Let's check to make sure that our configuration has persisted.

```
$ ifm3d dump | jq .ifm3d.Apps[0].Imager.FrameRate  
"10"
```

Let's now break down what we did in this single Linux pipeline.

```
$ ifm3d dump | jq '.ifm3d.Apps[0].Imager.FrameRate="10"' | ifm3d config
```

First we dump the entire state of the camera to JSON, process the JSON in-line via `jq` to set the `FrameRate` to 10 fps, then pipe the resulting output to `ifm3d config` which accepts the new (mutated) JSON stream on `stdin` and carries out the necessary network calls to mutate the camera settings and persist them.

This is the basic paradigm that can be followed to tune just about any parameter on the camera. To carry out more complex configuration tasks (e.g., changing several parameters at once), the dump can be saved to a file, edited via a text editor, then fed into `ifm3d config` to perform the configuration. It is also important to point out that `ifm3d config` does not need the entire `ifm3d` JSON object to operate correctly. Snippets are valid. For example, if we wanted to set the framerate back to 5, we could do this:

```
$ echo '{"Apps": [{"Index": "1", "Imager": {"FrameRate": "5"}}]} | ifm3d config
```

Let's validate that it worked:

```
$ ifm3d dump | jq .ifm3d.Apps[0].Imager.FrameRate  
"5"
```

In summary, the primary concept in configuring your camera via `ifm3d` is that the `dump` sub-command can be used to access the current camera state while the `config` subcommand is used to declare and mutate the camera into a desired state – assuming the desired state is valid.

## 2.4.3 Examples

### Creating new applications



Let's first list the applications on the camera.

```
$ ifm3d ls
[
  {
    "Active": true,
    "Description": "",
    "Id": 476707713,
    "Index": 1,
    "Name": "Sample Application"
  }
]
```

Now, let's create a new application whose settings will be bootstrapped from the application at index 1.

```
$ ifm3d cp --index=1
$ ifm3d ls
[
  {
    "Active": true,
    "Description": "",
    "Id": 476707713,
    "Index": 1,
    "Name": "Sample Application"
  },
  {
    "Active": false,
    "Description": "",
    "Id": 476707714,
    "Index": 2,
    "Name": "Sample Application"
  }
]
```

Now, let's create a new application from scratch (bootstrapped with camera-default settings).

```
$ echo '{"Apps":[]}' | ifm3d config
$ ifm3d ls
[
  {
    "Active": true,
    "Description": "",
    "Id": 476707713,
    "Index": 1,
    "Name": "Sample Application"
  },
  {
    "Active": false,
    "Description": "",
    "Id": 476707714,
```

(continues on next page)

(continued from previous page)

```

    "Index": 2,
    "Name": "Sample Application"
},
{
  "Active": false,
  "Description": "",
  "Id": 1755226334,
  "Index": 3,
  "Name": "New application"
}
]

```

Now, let's set the application at index 3 to be the current active application.

```

$ ifm3d dump | jq '.ifm3d.Device.ActiveApplication="3"' | ifm3d config
$ $ ifm3d ls
[
{
  "Active": false,
  "Description": "",
  "Id": 476707713,
  "Index": 1,
  "Name": "Sample Application"
},
{
  "Active": false,
  "Description": "",
  "Id": 476707714,
  "Index": 2,
  "Name": "Sample Application"
},
{
  "Active": true,
  "Description": "",
  "Id": 1755226334,
  "Index": 3,
  "Name": "New application"
}
]

```

Now, let's delete applications 2 and 3.

```

$ ifm3d rm --index=2
$ ifm3d rm --index=3
$ ifm3d ls
[
{
  "Active": false,
  "Description": "",
  "Id": 476707713,
  "Index": 1,
  "Name": "Sample Application"
}
]

```

We note that based on the sequence of steps we took in this example, we are currently left with a camera with only a single application but it is not marked as active. So, let's set that

application as “active” and validate it.

```
$ ifm3d dump | jq '.ifm3d.Device.ActiveApplication="1"' | ifm3d config
$ ifm3d ls
[
  {
    "Active": true,
    "Description": "",
    "Id": 476707713,
    "Index": 1,
    "Name": "Sample Application"
  }
]
```

### Setting NTP-Server connection on the camera



Using jq, you can set easily the NTP-Server on a camera. You just need to provide the right IP address. In this case, the IP: 192.168.0.100 is the NTP server.

```
ifm3d dump | jq '.ifm3d.Time.NTPServers="192.168.0.100"' | ifm3d config
```

After that, we need to activate the usage of the NTP server too.

```
ifm3d dump | jq '.ifm3d.Time.SynchronizationActivated="True"' | ifm3d config
```

### Setting the time on the camera



To set the time on the camera we use the `time` subcommand. Let’s look at its usage.

```
$ ifm3d time --help
Usage:
  ifm3d [<global options>] time [<time options>]

  global options:
    -h, --help          Produce this help message and exit
    --ip arg            IP address of the sensor (default: 192.168.0.69)
    --xmlrpc-port arg  XMLRPC port of the sensor (default: 80)
    --password arg     Password for establishing an edit-session with the
                      sensor (default: )

  time options:
    --epoch arg         Secs since Unix epoch encoding time to be set on camera
                      (-1 == now)
```

To simply see the current time on the camera, we can issue the `time` subcommand with no arguments.

```
$ ifm3d time
Local time on camera is: Tue Mar 13 18:22:16 2018
```

Let's now look at our local Unix time:

```
$ date  
Mon May  7 16:20:49 EDT 2018
```

To synchronize the camera to our local time we can issue the following command.

```
$ ifm3d time --epoch=-1  
Local time on camera is: Mon May  7 16:21:44 2018
```

And, double checking...

```
$ ifm3d time  
Local time on camera is: Mon May  7 16:22:09 2018
```

## 2.5 Python API Reference

### 2.5.1 ifm3dpy Module

Bindings for the ifm3d Camera Library

## Variables

<code>__version__</code>	The ifm3d version.
<code>__package__</code>	The ifm3d package.
<code>DEFAULT_IP</code>	The default IP to connect to.
<code>DEFAULT_XMLRPC_PORT</code>	The default XMLRPC port.
<code>DEFAULT_PASSWORD</code>	The default password.
<code>DEFAULT_SCHEMA_MASK</code>	The default pcic schema mask.
<code>IMG_RDIS</code>	Pcic schema constant for the radial distance image.
<code>IMG_AMP</code>	Pcic schema constant for the normalized amplitude image.
<code>IMG_RAMP</code>	Pcic schema constant for the raw amplitude image.
<code>IMG_CART</code>	Pcic schema constant for the cartesian image.
<code>IMG_UVEC</code>	Pcic schema constant for the wrapped unit vectors.
<code>EXP_TIME</code>	Pcic schema constant for the amplitude image.
<code>IMG_GRAY</code>	Pcic schema constant for the wrapped ambient light image.
<code>ILLU_TEMP</code>	Pcic schema constant for the illu temp.
<code>INTR_CAL</code>	Pcic schema constant for the intrinsic calibration.
<code>INV_INTR_CAL</code>	Pcic schema constant for the inverse intrinsic calibration.
<code>JSON_MODEL</code>	Pcic schema constant for the json model.
<code>O3D_TIME_SUPPORT_MAJOR</code>	Constant for querying for O3D time support.
<code>O3D_TIME_SUPPORT_MINOR</code>	Constant for querying for O3D time support.
<code>O3D_TIME_SUPPORT_PATCH</code>	Constant for querying for O3D time support.
<code>O3D_TMP_PARAMS_SUPPORT_MAJOR</code>	Constant for querying for O3D temporary parameter support.
<code>O3D_TMP_PARAMS_SUPPORT_MINOR</code>	Constant for querying for O3D temporary parameter support.
<code>O3D_TMP_PARAMS_SUPPORT_PATCH</code>	Constant for querying for O3D temporary parameter support.
<code>O3D_INTRINSIC_PARAM_SUPPORT_MAJOR</code>	Constant for querying for O3D intrinsic parameter support.
<code>O3D_INTRINSIC_PARAM_SUPPORT_MINOR</code>	Constant for querying for O3D intrinsic parameter support.
<code>O3D_INTRINSIC_PARAM_SUPPORT_PATCH</code>	Constant for querying for O3D intrinsic parameter support.
<code>O3D_INVERSE_INTRINSIC_PARAM_SUPPORT_MAJOR</code>	Constant for querying for O3D inverse intrinsic parameter support.
<code>O3D_INVERSE_INTRINSIC_PARAM_SUPPORT_MINOR</code>	Constant for querying for O3D inverse intrinsic parameter support.
<code>O3D_INVERSE_INTRINSIC_PARAM_SUPPORT_PATCH</code>	Constant for querying for O3D inverse intrinsic parameter support.

## Classes

<a href="#">Camera</a>	Class for managing an instance of an O3D/O3X Camera
<a href="#">CameraBase</a>	Base class for managing an instance of an all cameras
<a href="#">FrameGrabber</a>	Implements a TCP FrameGrabber connected to a provided Camera
<a href="#">ImageBuffer</a>	Class which holds a validated image buffer from the sensor that represents a single time-synchronized set of images based on the current schema mask set on the active framegrabber.
<a href="#">O3RCamera</a>	Class for managing an instance of an O3R Camera

## Camera

**class ifm3dpy.Camera**  
Bases: [ifm3dpy.CameraBase](#)

Class for managing an instance of an O3D/O3X Camera

### Attributes Summary

<a href="#">password</a>	The password associated with this Camera instance
<a href="#">session_id</a>	Retrieves the active session ID

### Methods Summary

<a href="#">active_application(self)</a>	Returns the index of the active application.
<a href="#">application_list(self)</a>	Delivers basic information about all applications stored on the device.
<a href="#">application_types(self)</a>	Lists the valid application types supported by the sensor.
<a href="#">cancel_session(*args, **kwargs)</a>	Overloaded function.
<a href="#">copy_application(self, idx)</a>	Creates a new application by copying the configuration of another application.
<a href="#">create_application(self[, type])</a>	Creates a new application on the camera of the given type.
<a href="#">delete_application(self, idx)</a>	Deletes the application at the specified index from the sensor.
<a href="#">export_ifm_app(self, idx)</a>	Export the application at the specified index into a byte array suitable for writing to a file.

continues on next page

Table 3 - continued from previous page

<code>export_ifm_config(self)</code>	Exports the entire camera configuration in a format compatible with Vision Assistant.
<code>factory_reset(self)</code>	Sets the camera configuration back to the state in which it shipped from the ifm factory.
<code>heartbeat(self, hb)</code>	Sends a heartbeat message and sets the next heartbeat interval
<code>imager_types(self)</code>	Lists the valid imager types supported by the sensor.
<code>import_ifm_app(self, bytes)</code>	Import the IFM-encoded application.
<code>import_ifm_config(self, bytes[, flags])</code>	Imports the entire camera configuration in a format compatible with Vision Assistant.
<code>request_session(self)</code>	Requests an edit-mode session with the camera.
<code>set_current_time(self[, epoch_secs])</code>	Sets the current time on the camera
<code>set_temporary_application_parameters(self, ...)</code>	Sets temporary application parameters in run mode.
<code>unit_vectors(self)</code>	For cameras that support fetching the Unit Vectors over XML-RPC, this function will return those data as a binary blob.

## Attributes Documentation

### `password`

The password associated with this Camera instance

### `session_id`

Retrieves the active session ID

## Methods Documentation

### `active_application(self: ifm3dpy.Camera) → int`

Returns the index of the active application.

A negative number indicates no application is marked as active on the sensor.

### `application_list(self: ifm3dpy.Camera) → object`

Delivers basic information about all applications stored on the device. A call to this function does not require establishing a session with the camera.

The returned information is encoded as an array of JSON objects. Each object in the array is basically a dictionary with the following keys: 'index', 'id', 'name', 'description', 'active'

`dict` A JSON encoding of the application information

`RuntimeError`

### `application_types(self: ifm3dpy.Camera) → List[str]`

Lists the valid application types supported by the sensor.

**list[str]** List of strings of the available types of applications supported by the sensor.  
Each element in the list is a string suitable to passing to 'CreateApplication'.

RuntimeError

**cancel\_session**(\*args, \*\*kwargs)

Overloaded function.

1. `cancel_session(self: ifm3dpy.Camera) -> bool`

Explicitly stops the current session with the sensor.

**bool** Indicates success or failure. On failure, check the ifm3d system log for details.

2. `cancel_session(self: ifm3dpy.Camera, sid: str) -> bool`

Attempts to cancel a session with a particular session id.

**sid** [str] Session ID to cancel.

**bool** Indicates success or failure. On failure, check the ifm3d system log for details.

**copy\_application**(self: [ifm3dpy.Camera](#), idx: int) → int

Creates a new application by copying the configuration of another application. The device will generate an ID for the new application and put it on a free index.

**idx** [int] The index of the application to copy

**int** Index of the new application

RuntimeError

**create\_application**(self: [ifm3dpy.Camera](#), type: str = 'Camera') → int

Creates a new application on the camera of the given type.

To figure out valid type's, you should call the AvailableApplicationTypes() method.

Upon creation of the application, the embedded device will initialize all parameters as necessary based on the type. However, based on the type, the application may not be in an \_activatable\_ state. That is, it can be created and saved on the device, but it cannot be marked as active.

**type** [str, optional] The (optional) application type to create. By default, it will create a new "Camera" application.

**int** The index of the new application.

**delete\_application**(self: [ifm3dpy.Camera](#), idx: int) → None

Deletes the application at the specified index from the sensor.

**idx** [int] The index of the application to delete

RuntimeError

**export\_ifm\_app**(self: [ifm3dpy.Camera](#), idx: int) → List[int]

Export the application at the specified index into a byte array suitable for writing to a file. The exported bytes represent the ifm serialization of an application.

This function provides compatibility with tools like IFM's Vision Assistant.

**idx** [int] The index of the application to export.

**list[int]** A list of bytes representing the IFM serialization of the exported application.

RuntimeError

**export\_ifm\_config**(self: ifm3dpy.Camera) → List[int]

Exports the entire camera configuration in a format compatible with Vision Assistant.

list[int]

**factory\_reset**(self: ifm3dpy.Camera) → None

Sets the camera configuration back to the state in which it shipped from the ifm factory.

**heartbeat**(self: ifm3dpy.Camera, hb: int) → int

Sends a heartbeat message and sets the next heartbeat interval

Heartbeat messages are used to keep a session with the sensor alive. This function sends a heartbeat message to the sensor and sets when the next heartbeat message is required.

**hb** [int] The time (seconds) of when the next heartbeat message will be required.

**int** The current timeout interval in seconds for heartbeat messages

RuntimeError

**imager\_types**(self: ifm3dpy.Camera) → List[str]

Lists the valid imager types supported by the sensor.

**list[str]** List of strings of the available imager types supported by the sensor

RuntimeError

**import\_ifm\_app**(self: ifm3dpy.Camera, bytes: List[int]) → int

Import the IFM-encoded application.

This function provides compatibility with tools like IFM's Vision Assistant. An application configuration exported from VA, can be imported using this function.

**bytes** [list[int]] The raw bytes from the zip'd JSON file. NOTE: This function will base64 encode the data for transmission over XML-RPC.

**int** The index of the imported application.

**import\_ifm\_config**(self: ifm3dpy.Camera, bytes: List[int], flags: int = 0) → None

Imports the entire camera configuration in a format compatible with Vision Assistant.

**bytes** [list[int]] The camera configuration, serialized in the ifm format

flags : int

**request\_session**(self: ifm3dpy.Camera) → str

Requests an edit-mode session with the camera.

In order to (permanently) mutate parameters on the camera, an edit session needs to be established. Only a single edit session may be established at any one time with the camera (think of it as a global mutex on the camera state – except if you ask for the mutex and it is already taken, an exception will be thrown).

Most typical use-cases for end-users will not involve establishing an edit-session with the camera. To mutate camera parameters, the FromJSON family of functions should be used, which, under-the-hood, on the user's behalf, will establish the edit session and gracefully close it. There is an exception. For users who plan to modulate imager parameters (temporary parameters) on the fly while running the framegrabber, managing the session manually is necessary. For this reason, we expose this method in the public Camera interface.

NOTE: The session timeout is implicitly set to ifm3d::MAX\_HEARTBEAT after the session has been successfully established.

**str** The session id issued or accepted by the camera (see IFM3D\_SESSION\_ID environment variable)

RuntimeError

@throws ifm3d::error\_t if an error is encountered.

**set\_current\_time**(self: [ifm3dpy.Camera](#), epoch\_secs: int = -1) → None  
Sets the current time on the camera

**epoch\_secs** [int, optional] Time since the Unix epoch in seconds. A value less than 0 will implicitly set the time to the current system time.

**set\_temporary\_application\_parameters**(self: [ifm3dpy.Camera](#), params: Dict[str, str]) → None  
Sets temporary application parameters in run mode.

The changes are not persistent and are lost when entering edit mode or turning the device off. The parameters "ExposureTime" and "ExposureTimeRatio" of the imager configuration are supported. All additional parameters are ignored (for now). Exposure times are clamped to their allowed range, depending on the exposure mode. The user must provide the complete set of parameters depending on the exposure mode, i.e., "ExposureTime" only for single exposure modes and both "ExposureTime" and "ExposureTimeRatio" for double exposure modes. Otherwise, behavior is undefined.

**params** [dict[str, str]] The parameters to set on the camera.

RuntimeError

**unit\_vectors**(self: [ifm3dpy.Camera](#)) → List[int]  
For cameras that support fetching the Unit Vectors over XML-RPC, this function will return those data as a binary blob.  
list[int]

## CameraBase

**class ifm3dpy.CameraBase**  
Bases: pybind11\_builtins.pybind11\_object  
Base class for managing an instance of an all cameras

## Attributes Summary

<code>ip</code>	The IP address associated with this Camera instance
<code>xmlrpc_port</code>	The XMLRPC port associated with this Camera instance

## Methods Summary

<code>am_i(self, family)</code>	Checking whether a device is one of the specified device family
<code>check_minimum_firmware_version(self, major, ...)</code>	Checks for a minimum ifm camera software version
<code>device_parameter(self, key)</code>	Convenience accessor for extracting a device parameter
<code>device_type(self[, use_cached])</code>	Obtains the device type of the connected camera.
<code>force_trigger(self)</code>	Sends a S/W trigger to the camera over XMLRPC.
<code>from_json(self, json)</code>	Configures the camera based on the parameter values of the passed in JSON.
<code>reboot(self, mode)</code>	Reboot the sensor
<code>to_json(self)</code>	A JSON object containing the state of the camera
<code>trace_logs(self, count)</code>	Delivers the trace log from the camera
<code>who_am_i(self)</code>	Retrieve the device family of the connected device

## Attributes Documentation

### `ip`

The IP address associated with this Camera instance

### `xmlrpc_port`

The XMLRPC port associated with this Camera instance

## Methods Documentation

`am_i(self: ifm3dpy.CameraBase, family: ifm3dpy.CameraBase.device_family) → bool`  
Checking whether a device is one of the specified device family

**family** [CameraBase.device\_family] The family to check for

**bool** True if the device is of the specified family

`check_minimum_firmware_version(self: ifm3dpy.CameraBase, major: int, minor: int, patch: int) → bool`  
Checks for a minimum ifm camera software version

**major** [int] Major version of software

**minor** [int] Minor Version of software

**patch** [int] Patch Number of software

**bool** True if current software version is greater or equal to the value passed

**device\_parameter**(self: [ifm3dpy.CameraBase](#), key: str) → str

Convenience accessor for extracting a device parameter

No edit session is created on the camera

**key** [str] Name of the parameter to extract

**str** Value of the requested parameter

RuntimeError

**device\_type**(self: [ifm3dpy.CameraBase](#), use\_cached: bool = True) → str

Obtains the device type of the connected camera.

This is a convenience function for extracting out the device type of the connected camera. The primary intention of this function is for internal usage (i.e., to trigger conditional logic based on the model hardware we are talking to) however, it will likely be useful in application-level logic as well, so, it is available in the public interface.

**use\_cached** [bool] If set to true, a cached lookup of the device type will be used as the return value. If false, it will make a network call to the camera to get the “real” device type. The only reason for setting this to false would be if you expect over the lifetime of your camera instance that you will swap out (for example) an O3D for an O3X (or vice versa) - literally, swapping out the network cables while an object instance is still alive. If that is not something you are worried about, leaving this set to true should result in a significant performance increase.

**str** Type of device connected

**force\_trigger**(self: [ifm3dpy.CameraBase](#)) → None

Sends a S/W trigger to the camera over XMLRPC.

The O3X does not S/W trigger over PCIC, so, this function has been developed specifically for it. For the O3D, this is a NOOP.

**from\_json**(self: [ifm3dpy.CameraBase](#), json: dict) → None

Configures the camera based on the parameter values of the passed in JSON. This function is \_the\_ way to tune the camera/application/imager/etc. parameters.

**json** [dict] A json object encoding a camera configuration to apply to the hardware.

**RuntimeError** If this raises an exception, you are encouraged to check the log file as a best effort is made to be as descriptive as possible as to the specific error that has occurred.

**reboot**(self: [ifm3dpy.CameraBase](#), mode: [ifm3dpy.CameraBase.boot\\_mode](#) = <boot\_mode.PRODUCTIVE: 0>) → None

Reboot the sensor

**mode** [CameraBase.boot\_mode] The system mode to boot into upon restart of the sensor

RuntimeError

**to\_json**(self: ifm3dpy.CameraBase) → object  
A JSON object containing the state of the camera

**dict** Camera JSON, compatible with python's json module

RuntimeError

**trace\_logs**(self: ifm3dpy.CameraBase, count: int) → List[str]  
Delivers the trace log from the camera  
A session is not required to call this function.

**count** [int] Number of entries to retrieve

**list[str]** List of strings for each entry in the tracelog

**who\_am\_i**(self: ifm3dpy.CameraBase) → ifm3dpy.CameraBase.device\_family  
Retrieve the device family of the connected device

**CameraBase.device\_family** The device family

## FrameGrabber

**class** ifm3dpy.FrameGrabber  
Bases: pybind11\_builtins.pybind11\_object  
Implements a TCP FrameGrabber connected to a provided Camera

### Methods Summary

<b>reset</b> (self, cam, mask, pcic_port)	Resets the FrameGrabber with a new camera/bitmask
<b>sw_trigger</b> (self)	Triggers the camera for image acquisition
<b>wait_for_frame</b> (self, buff[, timeout_millis, ...])	This function is used to grab and parse out time synchronized image data from the camera.

### Methods Documentation

**reset**(self: ifm3dpy.FrameGrabber, cam: ifm3d::CameraBase, mask: int = 10, pcic\_port: int = 0) → None  
Resets the FrameGrabber with a new camera/bitmask

**cam** [ifm3dpy.Camera] The camera instance to grab frames from.

**mask** [uint16] A bitmask encoding the image acquisition schema to stream in from the camera.

**pcic\_port** [uint16] The PCIC port

**sw\_trigger**(self: ifm3dpy.FrameGrabber) → None  
Triggers the camera for image acquisition

You should be sure to set the TriggerMode for your application to SW in order for this to be effective. This function simply does the triggering, data are still received asynchronously via WaitForFrame().

Calling this function when the camera is not in SW trigger mode or on a device that does not support software-trigger should result in a NOOP and no error will be returned (no exceptions thrown). However, we do not recommend calling this function in a tight framegrabbing loop when you know it is not needed. The “cost” of the NOOP is undefined and incurring it is not recommended.

```
wait_for_frame(self: ifm3dpy.FrameGrabber, buff: ifm3dpy.ImageBuffer,  
               timeout_millis: int = 0, copy_buff: bool = False, organize: bool =  
               True) → bool
```

This function is used to grab and parse out time synchronized image data from the camera.

**buff** [ifm3dpy.FrameBuffer] A FrameBuffer object to update with the latest data from the camera.

**timeout\_millis** [int] Timeout in millis to wait for new image data from the FrameGrabber. If timeout\_millis is set to 0, this function will block indefinitely.

**copy\_buff** [bool] Flag indicating whether the framegrabber’s internal buffer should be copied ( $O(n)$ ) or swapped ( $O(1)$ ) with the raw bytes of the passed in buff. You should only flag this as true if you are planning to use multiple clients with a single FrameGrabber – even then, think carefully before copying data around.

**organize** [bool] Flag indicating whether or not Organize should be called on the ByteBuffer before returning, or whether it should be deferred until the data is accessed.

**bool** True if a new buffer was acquired w/in “timeout\_millis”, false otherwise.

## ImageBuffer

```
class ifm3dpy.ImageBuffer  
Bases: pybind11_builtins.pybind11_object
```

Class which holds a validated image buffer from the sensor that represents a single time-synchronized set of images based on the current schema mask set on the active framegrabber.

### Methods Summary

amplitude_image(self)	Retrieves the amplitude image
confidence_image(self)	Retrieves the confidence image
distance_image(self)	Retrieves the radial distance image
exposure_times(self)	Returns the exposure times for the current frame.
extrinsics(self)	Returns a 6-element vector containing the extrinsic calibration of the camera.
gray_image(self)	Retrieves the gray image

continues on next page

Table 7 - continued from previous page

<code>illu_temp(self)</code>	Returns the temperature of the illumination unit.
<code>intrinsics(self)</code>	Retrieves the intrinsic calibration of the camera
<code>inverse_intrinsics(self)</code>	Retrieves the inverse intrinsic calibration of the camera.
<code>jpeg_image(self)</code>	Retrieves the jpeg encoded 2d image
<code>json_model(self)</code>	Returns the JSON model of the output of the active application
<code>organize(self)</code>	This is the interface hook that synchronizes the internally wrapped byte buffer with the semantically meaningful image/cloud data structures.
<code>raw_amplitude_image(self)</code>	Retrieves the raw amplitude image
<code>timestamp(self)</code>	Returns the time stamp of the image data.
<code>unit_vectors(self)</code>	Retrieves the unit vector image
<code>xyz_image(self)</code>	Retrieves the xyz image (cartesian point cloud)

## Methods Documentation

**`amplitude_image(self: ifm3dpy.ImageBuffer) → numpy.ndarray`**

Retrieves the amplitude image

**`numpy.ndarray`** Image organized on the pixel array [rows, cols]

**`confidence_image(self: ifm3dpy.ImageBuffer) → numpy.ndarray`**

Retrieves the confidence image

**`numpy.ndarray`** Image organized on the pixel array [rows, cols]

**`distance_image(self: ifm3dpy.ImageBuffer) → numpy.ndarray`**

Retrieves the radial distance image

**`numpy.ndarray`** Image organized on the pixel array [rows, cols]

**`exposure_times(self: ifm3dpy.ImageBuffer) → List[int]`**

Returns the exposure times for the current frame.

**`list[int]`** A 3-element vector containing the exposure times (usec) for the current frame. Unused exposure times are reported as 0.

If all elements are reported as 0 either the exposure times are not configured to be returned back in the data stream from the camera or an error in parsing them has occurred.

**`extrinsics(self: ifm3dpy.ImageBuffer) → List[float]`**

Returns a 6-element vector containing the extrinsic calibration of the camera.

NOTE: This is the extrinsics WRT to the ifm optical frame.

The elements are: tx, ty, tz, rot\_x, rot\_y, rot\_z

Translation units are mm, rotations are degrees

Users of this library are highly DISCOURAGED from using the extrinsic calibration data stored on the camera itself.

**gray\_image**(self: ifm3dpy.ImageBuffer) → numpy.ndarray

Retrieves the gray image

**numpy.ndarray** Image organized on the pixel array [rows, cols]

**illu\_temp**(self: ifm3dpy.ImageBuffer) → float

Returns the temperature of the illumination unit.

NOTE: To get the temperature of the illumination unit to the frame, you need to make sure your current pcic schema asks for it.

**float** The temperature of the illumination unit

**intrinsics**(self: ifm3dpy.ImageBuffer) → List[float]

Retrieves the intrinsic calibration of the camera

**list[float]** 16-element list containing the intrinsic calibration of the camera The elements are:

Name Unit Description  
fx px Focal length of the camera in the sensor's x axis direction.  
fy px Focal length of the camera in the sensor's y axis direction.  
mx px Main point in the sensor's x direction  
my px Main point in the sensor's y direction  
alpha dimensionless Skew parameter  
k1 dimensionless First radial distortion coefficient  
k2 dimensionless Second radial distortion coefficient  
k5 dimensionless Third radial distortion coefficient  
k3 dimensionless First tangential distortion coefficient  
k4 dimensionless Second tangential distortion coefficient  
transX mm Translation along x-direction in meters.  
transY mm Translation along y-direction in meters.  
transZ mm Translation along z-direction in meters.  
rotX float degree Rotation along x-axis in radians. Positive values indicate clockwise rotation.  
rotY float degree Rotation along y-axis in radians. Positive values indicate clockwise rotation.  
rotZ float degree Rotation along z-axis in radians. Positive values indicate clockwise rotation.

**inverse\_intrinsics**(self: ifm3dpy.ImageBuffer) → List[float]

Retrieves the inverse intrinsic calibration of the camera. See the documentation for ifm3dpy.intrinsics for details on contents.

**jpeg\_image**(self: ifm3dpy.ImageBuffer) → numpy.ndarray

Retrieves the jpeg encoded 2d image

**numpy.ndarray** Jpeg encoded image data

**json\_model**(self: ifm3dpy.ImageBuffer) → object

Returns the JSON model of the output of the active application

NOTE: To get the JSON data for the application running on the device, you need to make sure your current pcic schema asks for it by including ifm3d::JSON\_MODEL in the schema. This will return an empty dict for Camera devices like the O3D303, versus ifm Smart Sensors like the O3D302.

**dict** A JSON encoding of the model

**organize**(self: ifm3dpy.ImageBuffer) → None

This is the interface hook that synchronizes the internally wrapped byte buffer with the semantically meaningful image/cloud data structures. Within the overall ifm3d framework, this function is called by the FrameGrabber when a complete "frame packet" has been received. This then parses the bytes and, in-line, will statically dispatch to the underlying derived class to populate their image/cloud data structures.

Additionally, this function will populate the extrinsics, exposure times, timestamp, and illumination temperature as appropriate and subject to the current pcic schema.

NOTE: This function is called automatically as needed the first time frame data are accessed.

**raw\_amplitude\_image**(self: ifm3dpy.ImageBuffer) → numpy.ndarray

Retrieves the raw amplitude image

**numpy.ndarray** Image organized on the pixel array [rows, cols]

**timestamp**(self: ifm3dpy.ImageBuffer) → datetime.datetime

Returns the time stamp of the image data.

NOTE: To get the timestamp of the confidence data, you need to make sure your current pcic schema mask have enabled confidence data.

datetime.datetime

**unit\_vectors**(self: ifm3dpy.ImageBuffer) → numpy.ndarray

Retrieves the unit vector image

**numpy.ndarray** Image organized on the pixel array [rows, cols]

**xyz\_image**(self: ifm3dpy.ImageBuffer) → numpy.ndarray

Retrieves the xyz image (cartesian point cloud)

**numpy.ndarray nxmx3** Image organized on the pixel array [rows, cols, chans(x,y,z)]

## O3RCamera

**class ifm3dpy.O3RCamera**

Bases: [ifm3dpy.CameraBase](#)

Class for managing an instance of an O3R Camera

### Methods Summary

<code>factory_reset(self, keep_network_settings)</code>	Sets the camera configuration back to the state in which it shipped from the ifm factory.
<code>get(self[, path])</code>	Returns the configuration formatted as JSON based on a path.
<code>get_init(self)</code>	Return the initial JSON configuration.
<code>get_init_status(self)</code>	Returns the init status of the device
<code>get_schema(self)</code>	Returns the current JSON schema configuration
<code>lock(self, password)</code>	Release the lock from the Device
<code>save_init(self)</code>	Save to current temporary JSON configuration as initial JSON configuration
<code>set(self, json)</code>	Overwrites parts of the temporary JSON configuration which is achieved by merging the provided JSON fragment with the current temporary JSON.
<code>unlock(self, password)</code>	Locks the device until it is unlocked.

## Methods Documentation

**factory\_reset**(self: [ifm3dpy.O3RCamera](#), keep\_network\_settings: bool) → None  
Sets the camera configuration back to the state in which it shipped from the ifm factory.

**keep\_network\_settings** [bool] A bool indicating whether to keep the current network settings

**get**(self: [ifm3dpy.O3RCamera](#), path: List[str] = []) → object  
Returns the configuration formatted as JSON based on a path. If the path is empty, returns the whole configuration.

**dict** The JSON configuration for the list of object path fragments

**get\_init**(self: [ifm3dpy.O3RCamera](#)) → object  
Return the initial JSON configuration.

**dict** The initial JSON configuration

**get\_init\_status**(self: [ifm3dpy.O3RCamera](#)) → str  
Returns the init status of the device

**dict** The init status of the device

**get\_schema**(self: [ifm3dpy.O3RCamera](#)) → object  
Returns the current JSON schema configuration

**dict** The current json schema configuration

**lock**(self: [ifm3dpy.O3RCamera](#), password: str) → None  
Release the lock from the Device

**string** The password used to unlock the device

**save\_init**(self: [ifm3dpy.O3RCamera](#)) → None  
Save to current temporary JSON configuration as initial JSON configuration

**set**(self: [ifm3dpy.O3RCamera](#), json: dict) → None  
Overwrites parts of the temporary JSON configuration which is achieved by merging the provided JSON fragment with the current temporary JSON.

**json** [dict] The new temporary JSON configuration of the device.

**unlock**(self: [ifm3dpy.O3RCamera](#), password: str) → None  
Locks the device until it is unlocked. If the device is unlocked and an empty password is provided the password protection is removed.

**string** The password used to lock the device

## 2.6 C++ API Reference

C++ documentation has been disabled

---

**CHAPTER  
THREE**

---

**INDICES AND TABLES**

- genindex
- modindex
- search



## ROS WRAPPERS FOR IFM3D

### 4.1 ifm3d-ros

#### 4.1.1 ifm3d-ros

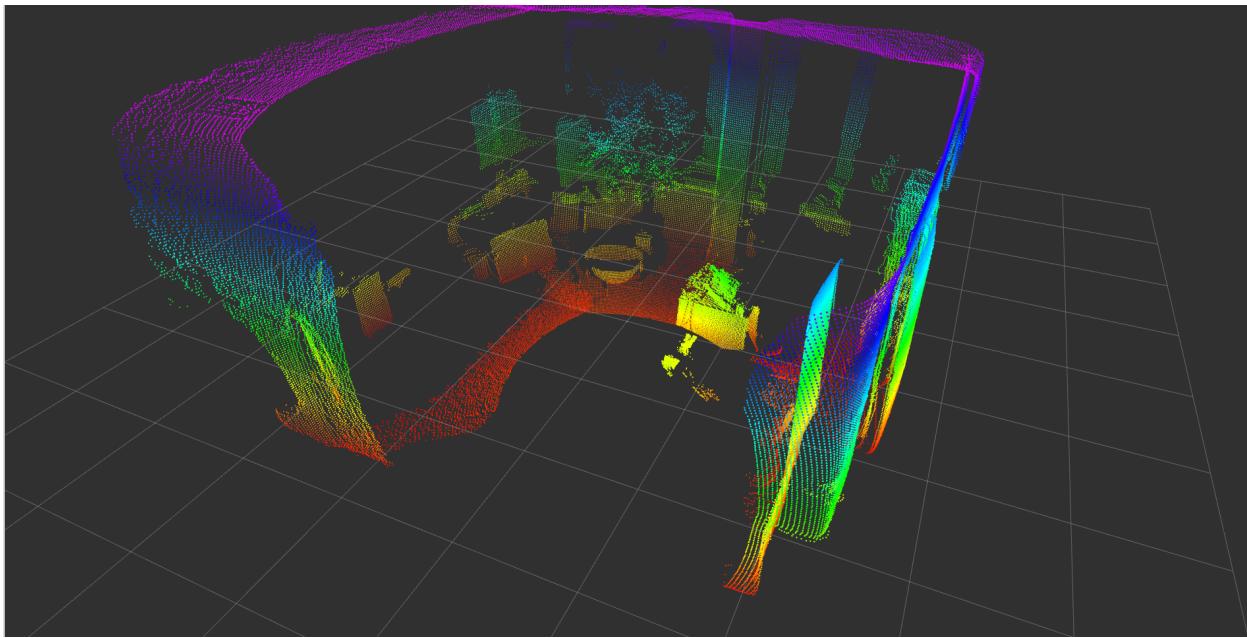
##### Release versions

:warning: Note that the v1.0.x branch is generally in a work in progress state, and you probably want to use a tagged [release version](#) for production.

##### ifm3d-ros for the O3R

**NOTE: The ifm3d-ros package has had major changes recently. Please be aware that this might cause problems on your system for building pipelines based on our old build instructions.** We tried to ensure backward compatibility where ever possible. If you find any major breaks, please let us know.

ifm3d-ros is a wrapper around [ifm3d](#) enabling the usage of the O3R camera platform (ifm ToF cameras) from within [ROS](#) software systems. Please make sure to use the v1.0.x branch for O3R compatibility.



## Software Compatibility Matrix

<b>ifm3d-ros version</b>	<b>ifm3d version</b>	<b>ROS distribution(s)</b>
1.0.0	in development - latest version checked 0.91.0	Noetic

## Internal ifm3d-ros subpackage version structure

Please see the internal subpackage version structure for a known ifm3d-ros version.

<b>ifm3d-ros version</b>	<b>ifm3d_ros_driver</b>	<b>ifm3d_ros_msgs</b>	<b>ifm3d_ros_examples</b>
1.0.0	0.7.0	0.1.0	0.1.0

## Organization of the software

The ifm3d-ros meta package provides three subpackages:

- ifm3d\_ros\_driver provides the core interface for receiving data for ifm 3d (O3R) cameras.
- ifm3d\_ros\_msgs gathers the ifm-specific messages types and the services for configuring and triggering the camera.
- ifm3d\_ros\_examples provides additional helper scripts and examples.

The name ifm3d-ros was kept even tough this is not consistent with ROS package naming conventions. This ROS package has been split into three sub packages in an effort to facilitate dependency handling on distributed systems and simplify deployment on embedded platforms. For instance, the package ifm3d\_ros\_msgs can be installed independently of the other

packages to control the camera from a separate computing platform. The `ifm3d_ros_examples` holds our launch files and examples.

## Building and installing the software

1. Preparing your system: Noetic
2. Installing the ifm3d-ros node

## LICENSE

Please see the file called LICENSE.

### 4.1.2 The `ifm3d_ros_driver` package

#### ROS Interface

The core `ifm3d-ros` sensor interface is implemented as a ROS nodelet. This allows for lower-latency data processing vs. the traditional out-of-process node-based ROS interface for applications that require it. However, we ship a launch file with this package that allows for using the core `ifm3d-ros` driver as a standard node. To launch the node, the following command can be used:

```
$ roslaunch ifm3d_ros_examples camera.launch
```

Note: Please notice the use of the subpackage `ifm3d_ros_examples`.

For further information about the internal ROS nodelet infrastructure and how to apply this to your application, please see our exemplary launch files and a short run down on the nodelet structure in the [ifm3d\\_ros\\_examples/README](#).

## Nodelet - parameters

Name	Data Type	De-fault Value	Description
<code>~assume_sw_trig</code>	bool	false	This provides a hint to the driver that the camera is configured for software triggering (as opposed to free running). In this mode, certain default values are applied to lessen the noise in terms of timeouts from the frame grabber.
<code>~frame_id</code>	string	<code>ifm3d/this</code>	This string provides a prefix into the tf tree for ifm3d_ros coordinate frames.
<code>~frame_tendency_thresh</code>	float	60.0	Time (seconds) used to determine that timestamps from the camera cannot be trusted. When this threshold is exceeded, when compared to system time, we use the reception time of the frame and not the capture time of the frame.
<code>~ip</code>	string	192.168.0.100	IP address of the VPU.
<code>~pass-word</code>	string	""	The password required to establish an edit session on the VPU
<code>~schema</code>	uint32	0xf	The pcic schema mask to apply to the active session with the frame grabber. This determines which images are available for publication from the camera. More about pcic schemas can be gleaned from the <a href="#">ifm3d projects documentation</a> .
<code>~time-out_millis</code>	int	500	The number of milliseconds to wait for the framegrabber to return new frame data before declaring a "timeout" and to stop blocking on new data.
<code>~time-out_tolerance_secs</code>	float	5.0	The wall time to wait with no new data from the camera before trying to establish a new connection to the camera. This helps to provide robustness against camera cables becoming unplugged or other in-field pathologies which would cause the connection between the ROS node and the camera to be broken.
<code>~sync_clocks_DEP-RE-CATED</code>	bool	false	Attempt to sync the camera clock to the system clock at start-up. The side-effect is that timestamps on the image should reflect the capture time as opposed to the receipt time.
<code>~xml-rpc_port</code>	uint32	18000	The TCP port the camera's xmlrpc server is listening on for requests.
<code>~pcic_port</code>	uint32	1500010	The TCP (data) port the camera's pcic server is listening on for requests.

## Nodelet - published Topics

Name	Data Type	Description
amplitude	sensor_msgs/Image	The normalized amplitude image.
confidence	sensor_msgs/Image	The confidence image.
cloud	sensor_msgs/PointCloud2	The point cloud data, i.e. X-, Y-, Z-coordinates.
distance	sensor_msgs/Image	The radial distance image.
raw_amplitude	sensor_msgs/Image	The raw (non normalized) amplitude image.
good_bad_pixels	sensor_msgs/Image	The binary image representation of the confidence image.
xyz_image	sensor_msgs/Image	A 3-channel image encoding of the point cloud. Each of the three image channels represents a spatial data plane encoding the x, y, z Cartesian values respectively.
unit_vectors	sensor_msgs/Image	The rotated unit vectors.
extrinsics	ifm3d/Extrinsics	The extrinsic calibration of the camera with respect to the camera optical frame. This 3D pose is encoded in mm and rad.

Note: Some topics may have empty data fields. We are working on publishing data on all available topics, but have kept all previous topics active for the moment for legacy reasons.

## Nodelet - subscribed Topics

None.

## Nodelet - advertised Services

Note: the services are provided by the `ifm3d_ros_msgs` package.

Name	Service Definition	Description
Dump	<code>ifm3d/Dump</code>	Dumps the state of the camera system as a JSON (formatted as a string)
Config	<code>ifm3d/Config</code>	Provides a means to configure the VPU and Heads (imager settings), declaratively from a JSON (string) encoding of the desired settings.
Soft-Off	<code>ifm3d/Soft-Off</code>	Sets the active application of the camera into software triggered mode which will turn off the active illumination reducing both power and heat.
SoftOn	<code>ifm3d/SoftOn</code>	Sets the active application of the camera into free-running mode. Its intention is to act as the inverse of SoftOff.
Trigger	<code>ifm3d/Trigger</code>	Requests the driver to software trigger the imager for data acquisition.

## Known limitations



## Additional Documentation

- Inspecting and configuring the camera / imager settings
- Troubleshooting

## LICENSE

Please see the file called LICENSE.

### 4.1.3 The `ifm3d_ros_msgs` package

This package provides the messages and services interfaces for the `ifm3d_ros_driver` package. It can be installed independently of the driver package `ifm3d_ros_driver` and examples package `ifm3d_ros_examples`.

#### Standalone Installation of the messages packages

If you plan on installing only one subpackage please see the instructions below.

```
catkin_make --only-pkg-withdeps <target_package>
```

Please replace the tag `<target_package>` with the name of the package you want to compile:

- `ifm3d_ros_driver`
- `ifm3d_ros_msgs`

- `ifm3d_ros_examples`

Some of our subpackages have dependencies to other packages and therefore will trigger a compiling of more subpackages, namely the packages `ifm3d_ros_examples` and `ifm3d_ros_driver`. These subpackages can not be used standalone.

Don't forget to switch back to building all packages afterwards:

```
catkin_make -DCATKIN_WHITELIST_PACKAGES=""
```

## LICENSE

Please see the file called LICENSE.

### 4.1.4 The `ifm3d_ros_examples` package

This package provides examples and helper scripts for using the ifm O3R camera platform.

#### Launchfiles

Please see the list below for launch files shipped with the examples package:

Name	Description
<code>six_camera.launch</code>	Launches six nodes, reading data streams on ports 0, 1, 2, 3, 4 and 5. Provide coordinate frame transforms for each node. Note: you can use this example for less than six heads, but you will get a timeout error where no heads are connected. This does not disrupt the proper functioning of the connected heads.
<code>nodelet.launch</code>	This is handling the nodelet manager which makes it possible to launch a nodelet similarly as you would a simple node.
<code>head.launch</code>	Launches two data streams for both the 2D RGB imager and 3D TOF imager on a camera head. Default ports are 0 (pcic_port:=50010) and 2 (pcic_port:=50012). For different port numbers input port as a parameter when launching.
<code>camera.launch</code>	Launches a single camera stream - so only 3D data or 2D RGB data. This launch file is comparable to a single camera setup (O3Ds and O3Xs)

#### Nodelet launch structure

Note: The O3R platform can handle multiple data streams.\*The `camera.launch` file only launches a node for one data stream, on the default pcic port 50010. To launch a node for a different port, use:

```
$ rosrun ifm3d_ros_driver camera.launch pcic_port:=<PORT_NUMBER>
```

The launch file(s) encapsulate several features:

1. It (partially) exposes the `camera_nodelet` parameters as command-line arguments for ease of runtime configuration.
2. It instantiates a nodelet manager which the `camera_nodelet` will be loaded into.
3. It launches the camera nodelet itself.

4. It publishes the static transform from the camera's optical frame to a traditional ROS sensor frame as a tf2 static\_transform\_publisher.

You can either use this launch file directly, or, use it as a basis for integrating ifm3d\_ros into your own robot software system.

Note: the O3R camera heads carry two imagers, a 3D time-of-flight and a RGB imager.

We provide the head.launch launchfile to handle a whole O3R camera head, that starts two nodes, one for the RGB image (we assume it is plugged in port 0), and one for the 3D imager (we assume it is plugged in port 2).

### **Building launch files distributed systems**

Note: This is WIR. We are currently working on Docker images which will allow you an easy deployment of our ROS node to the VPU.

### **LICENSE**

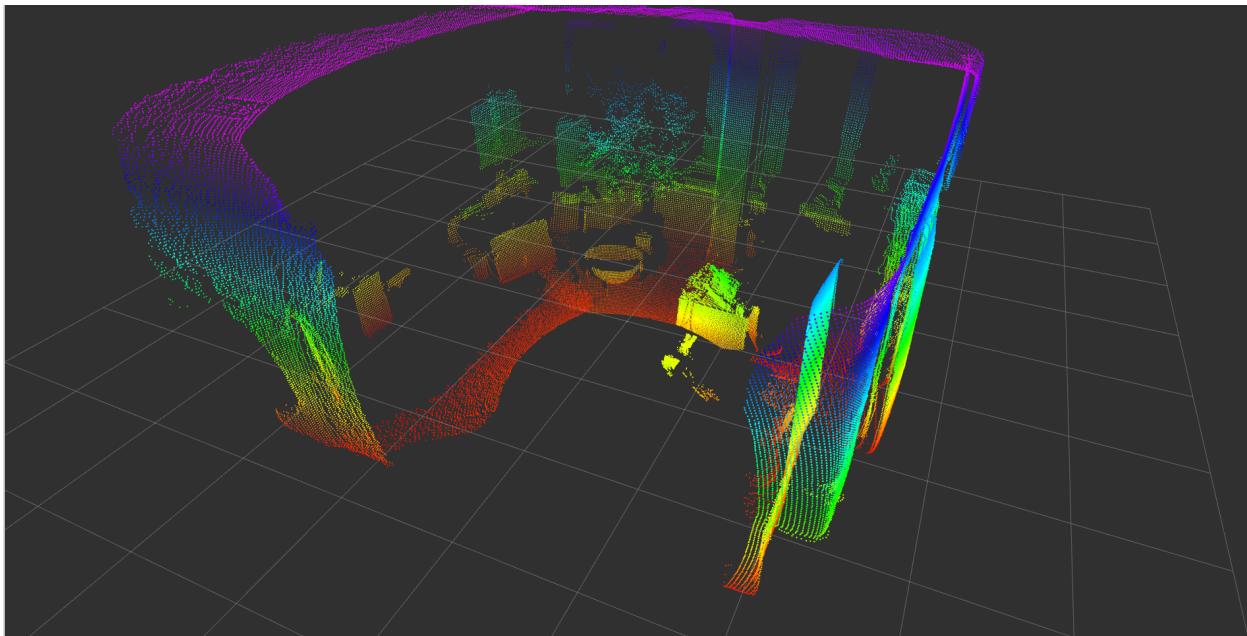
Please see the file called LICENSE.

## **4.2 ifm3d-ros2**

**This release is intended to be used with the O3R camera platform ONLY. For other ifm cameras please see release tag v0.3.0.**

**NOTE: The ifm3d-ros2 package has had major changes recently. Please be aware that this might cause problems on your system for building pipelines based on our old build instructions.** We tried to ensure backward compatibility where ever possible. If you find any major breaks, please let us know.

ifm3d-ros2 is a wrapper around ifm3d enabling the usage of ifm O3R ToF camera platform from within ROS 2 software systems.



#### 4.2.1 Software Compatibility Matrix

ifm3d_ros2 version	ifm3d version	ROS 2 distribution
1.0.0	0.92.0	Galactic
0.3.0 DEPRECATED	0.17.0	Dashing, Eloquent
0.2.0 DEPRECATED	0.12.0	Dashing
0.1.1 DEPRECATED	0.12.0	Dashing
0.1.0 DEPRECATED	0.12.0	Dashing

Note: ifm3d\_ros2 version 1.0.0 is released as an early developer release for the O3R camera.

#### 4.2.2 Building and Installing the Software

##### Pre-requisites

1. ROS2
2. ifm3d - be sure to build the IMAGE module (using PCL and OPENCV).

In addition to the base packages found in ros-\*desktop-full you will need the following ROS packages:

- cv\_bridge
- vision\_opencv
- pcl-conversions

These two packages are only required for testing but not at runtime:

- launch\_testing
- launch\_testing\_ament\_cmake

On debian based systems they may be installed as follows (replacing `galactic`with your target ROS2 distribution).

```
$ sudo apt install ros-galactic-cv-bridge ros-galactic-vision-opencv ros-galactic-pcl-  
↳ conversions
```

```
$ sudo apt install ros-galactic-launch-testing ros-galactic-launch-testing-ament-cmake
```

## Building from source

Please see the separate building instruction for building from source: [here](#)

## Launch the node

Launch the camera node (assuming you are in `~/colcon_ws/`):

```
$ . install/setup.bash  
$ ros2 launch ifm3d_ros2 camera_managed.launch.py
```

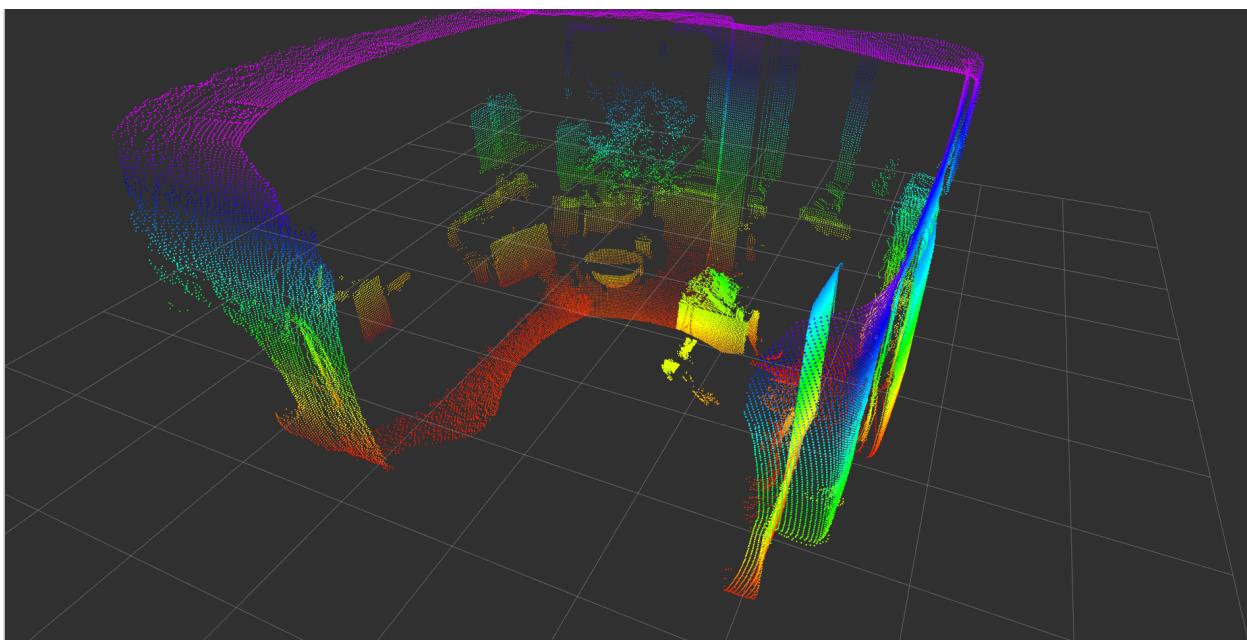
Note: we also provide a helper launch file to start multiple camera nodes. See the documentation [here](#).

Open another shell and start the RVIZ node to visualize the data coming from the camera:

```
$ ros2 launch ifm3d_ros2 rviz.launch.py
```

Note: `rviz.launch.py` does not include the camera node itself, but subscribes to published topics (distance, amplitude, etc). A camera node need to be running in parallel to `rviz` (you can use `camera_managed.launch`). Note also that the `rviz.launch.py` launchfile assumes one data stream publishes at `/ifm3d/camera/<topic_name>`.

At this point, you should see an `rviz` window that looks something like the image below (note that this is the view from 3 camera heads):



Congratulations! You can now have complete control over the O3R perception platform from inside ROS.

#### 4.2.3 ROS Interface

**Parameters**

Name	Data Type	De-fault Value	Description
<code>~/frame_latency_thresh</code>	float	1.0	Time (seconds) used to determine that timestamps from the camera cannot be trusted. When this threshold is exceeded, when compared to system time, we use the reception time of the frame and not the capture time of the frame.
<code>~/ip</code>	string	192.168.0.109	IP address of the camera.
<code>~/password</code>	string		The password required to establish an edit session with the camera.
<code>~/schema_mask</code>	int	0	The schema mask to apply to the active session with the frame grabber. This determines which images are available for publication from the camera. More about schemas can be gleaned from the ifm3d project
<code>~/timeout_millis</code>	int	500	The number of milliseconds to wait for the framegrabber to return new frame data before declaring a “timeout” and to stop blocking on new data.
<code>~/timeout_tolerance_secs</code>	float	5.0	The wall time to wait with no new data from the camera before trying to establish a new connection to the camera. This helps to provide robustness against camera cables becoming unplugged or other in-field pathologies which would cause the connection between the ROS node and the camera to be broken.
<code>~/sync_DEP_RE-CATED</code>	bool	false	Attempt to sync the camera clock to the system clock at start-up. The side-effect is that timestamps on the image should reflect the capture time as opposed to the receipt time. Please note: resolution of this sync is only granular to 1 second. If fine-grained image acquisition times are needed, consider using the on-camera NTP server (available on select camera models).
<code>~/xmlrpc_port</code>	uint16	680	The TCP port the camera’s xmlrpc server is listening on for requests.
<code>~/pcic_port</code>	uint16	65001	The TCP (data) port the camera’s pcic server is listening on for requests.

## Published Topics

Name	Data Type	Quality of Service (QoS)	Description
amplitude	sensor_msgs/msg/Image	ifm3d_ros::LowLatencyQoS	The normalized amplitude image
cloud	sensor_msgs/msg/PointCloud2	ifm3d_ros::LowLatencyQoS	The point cloud data
confidence	sensor_msgs/msg/Image	ifm3d_ros::LowLatencyQoS	The confidence image
distance	sensor_msgs/msg/Image	ifm3d_ros::LowLatencyQoS	The radial distance image
raw_amplitude	sensor_msgs/msg/Image	ifm3d_ros::LowLatencyQoS	The raw amplitude image (currently not available for the O3R)
rgb	sensor_msgs/msg/Image	ifm3d_ros::LowLatencyQoS	The RGB 2D image of the 2D imager

## Subscribed Topics

None.

## Advertised Services

Name	Service Definition	Description
Dump	ifm3d/Dump	Dumps the state of the camera parameters to JSON
Config	ifm3d/Config	Provides a means to configure the camera and imager settings, declaratively from a JSON encoding of the desired settings.

### 4.2.4 Additional Documentation

- Inspecting and configuring the camera/imager settings
- Building the ROS node from source
- Visualization
- Running the ROS node on a distributed system
- ifm3d API RPC error codes

#### **4.2.5 ToDo**

We are currently working on rounding out the feature set of our ROS2 interface. Our current objectives are to get the feature set to an equivalent level to that of our ROS1 interface and to tune the ROS2/DDS performance to optimize the usage of our cameras from within ROS2 system (for different DDS implementations). Thanks for your patience as we continue to ensure our ROS2 interface is feature-rich, robust, and performant. Your feedback is greatly appreciated.

#### **4.2.6 Known limitations**

This ROS 2 node build on top of the ifm3d API which handles the data communication between the camera platform and the outside world. This is based on ASIO which conflicts with the DDS middleware fastRTPS implementation in ROS (until ROS foxy). We are currently working on a solution. Until then we suggest to use cyclone DDS for older ROS 2 distributions.

#### **4.2.7 LICENSE**

Please see the file called LICENSE.

## **RESOURCES AVAILABLE FOR DOWNLOAD**

### **5.1 Previous versions of the documentation**

Document	firmware version	ifm3d version	ifm3d-ros1 version	ifm3d-ros2 version
2022-02-22	13.13	0.92.x	1.0.0	1.0.0



## PYTHON MODULE INDEX

i

ifm3dpy, 102



# INDEX

## A

active\_application() (ifm3dpy.Camera method), 105  
am\_i() (ifm3dpy.CameraBase method), 109  
amplitude\_image() (ifm3dpy.ImageBuffer method), 113  
application\_list() (ifm3dpy.Camera method), 105  
application\_types() (ifm3dpy.Camera method), 105

## C

Camera (class in ifm3dpy), 104  
CameraBase (class in ifm3dpy), 108  
cancel\_session() (ifm3dpy.Camera method), 106  
check\_minimum\_firmware\_version() (ifm3dpy.CameraBase method), 109  
confidence\_image() (ifm3dpy.ImageBuffer method), 113  
copy\_application() (ifm3dpy.Camera method), 106  
create\_application() (ifm3dpy.Camera method), 106

## D

delete\_application() (ifm3dpy.Camera method), 106  
device\_parameter() (ifm3dpy.CameraBase method), 110  
device\_type() (ifm3dpy.CameraBase method), 110  
distance\_image() (ifm3dpy.ImageBuffer method), 113

## E

export\_ifm\_app() (ifm3dpy.Camera method), 106  
export\_ifm\_config() (ifm3dpy.Camera method), 107

exposure\_times() (ifm3dpy.ImageBuffer method), 113  
extrinsics() (ifm3dpy.ImageBuffer method), 113

## F

factory\_reset() (ifm3dpy.Camera method), 107  
factory\_reset() (ifm3dpy.O3RCamera method), 116  
force\_trigger() (ifm3dpy.CameraBase method), 110  
FrameGrabber (class in ifm3dpy), 111  
from\_json() (ifm3dpy.CameraBase method), 110

## G

get() (ifm3dpy.O3RCamera method), 116  
get\_init() (ifm3dpy.O3RCamera method), 116  
get\_init\_status() (ifm3dpy.O3RCamera method), 116  
get\_schema() (ifm3dpy.O3RCamera method), 116  
gray\_image() (ifm3dpy.ImageBuffer method), 113

## H

heartbeat() (ifm3dpy.Camera method), 107

## I

ifm3dpy module, 102  
illu\_temp() (ifm3dpy.ImageBuffer method), 114  
ImageBuffer (class in ifm3dpy), 112  
imager\_types() (ifm3dpy.Camera method), 107  
import\_ifm\_app() (ifm3dpy.Camera method), 107  
import\_ifm\_config() (ifm3dpy.Camera method), 107

**i** `intrinsics()` (ifm3dpy.ImageBuffer method), 114  
`inverse_intrinsics()` (ifm3dpy.ImageBuffer method), 114  
`ip` (ifm3dpy.CameraBase attribute), 109

**J** `jpeg_image()` (ifm3dpy.ImageBuffer method), 114  
`json_model()` (ifm3dpy.ImageBuffer method), 114

**L** `lock()` (ifm3dpy.O3RCamera method), 116

**M** `module ifm3dpy`, 102

**O** `O3RCamera` (class in ifm3dpy), 115  
`organize()` (ifm3dpy.ImageBuffer method), 114

**P** `password` (ifm3dpy.Camera attribute), 105

**R** `raw_amplitude_image()` (ifm3dpy.ImageBuffer method), 115  
`reboot()` (ifm3dpy.CameraBase method), 110  
`request_session()` (ifm3dpy.Camera method), 107  
`reset()` (ifm3dpy.FrameGrabber method), 111

**S** `save_init()` (ifm3dpy.O3RCamera method), 116  
`session_id` (ifm3dpy.Camera attribute), 105  
`set()` (ifm3dpy.O3RCamera method), 116  
`set_current_time()` (ifm3dpy.Camera method), 108  
`set_temporary_application_parameters()` (ifm3dpy.Camera method), 108  
`sw_trigger()` (ifm3dpy.FrameGrabber method), 111

**T** `timestamp()` (ifm3dpy.ImageBuffer method), 115

**U** `to_json()` (ifm3dpy.CameraBase method), 111  
`trace_logs()` (ifm3dpy.CameraBase method), 111

**W** `unit_vectors()` (ifm3dpy.Camera method), 108  
`unit_vectors()` (ifm3dpy.ImageBuffer method), 115  
`unlock()` (ifm3dpy.O3RCamera method), 116

**X** `xmlrpc_port` (ifm3dpy.CameraBase attribute), 109  
`xyz_image()` (ifm3dpy.ImageBuffer method), 115