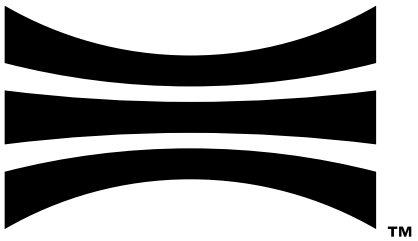


Ouster Detect User Manual

CONFIDENTIAL, preliminary draft v0.2

Ouster

Mar 22, 2023

Contents

1 Important Safety Information

1.1 Safety & Legal Notices

All Ouster sensors have been evaluated to be **Class 1 laser products** per **60825-1: 2014 (Ed. 3)** and operate in the 865nm band.

Tous les capteurs Ouster répondent aux critères des **produits laser de classe 1**, selon la norme **IEC 60825-1: 2014 (3ème édition)** et émettent dans le domaine de l'infrarouge, à une longueur d'onde de 865nm environ.

FDA 21CFR1040 Notice: All Ouster sensors comply with FDA performance standards for laser products except for deviations pursuant to Laser Notice No. 56, dated January 19, 2018.

Notice FDA 21CFR1040: Tous les capteurs Ouster sont conformes aux exigences de performances établies par la FDA pour les produits laser, à l'exception des écarts en application de l'avis nº56, daté du 19 janvier 2018.

Figure 1.1: Class 1 Laser Product

Figure 1.2: Caution "Sharp Edges"

The following symbols appear on the product label and in the user manual have the following meaning.

CAUTIONS

Figure 1.3: This symbol indicates that the sensor emits laser radiation.

Figure 1.4: This symbol indicates the presence of a hot surface that may cause skin burn.

- All Ouster sensors are hermetically sealed unit, and are non user-serviceable.
- Use of controls, or adjustments, or performance of procedures other than those specified herein, may result in hazardous radiation exposure.
- Use of any Ouster sensor is subject to the Terms of Sale that you agreed and signed with Ouster or your distributor/integrator. Included in these terms are the prohibitions of:
    - Removing or otherwise opening the sensor housing
    - Inspecting the internals of the sensor
    - Reverse-engineering any part of the sensor
    - Permitting any third party to do any of the foregoing
- Operating the sensor without the attached mount that is shipped with the sensor, or attaching the sensor to a surface of inappropriate thermal capacity runs the risk of having the sensor overheat under certain circumstances.
- The Ouster sensor features a modular cap design to enable more flexible mounting and integration solutions for the sensor.
- The modular cap design increases design flexibility but it does not remove the need for thermal management on top of the sensor. The attached radial cap serves an important thermal management purpose and the sensor will not operate properly without a cap.
- Operation for extended periods of time without the cap will result in system errors and the sensor overheating. The cap can be replaced with alternative solutions but it cannot be left off altogether.
- If you wish to operate the sensor with a custom mounting solution, please contact our Field Application Team and we can answer your questions and provide guidance for achieving proper operations.
- This product emits Class 1 invisible laser radiation. The entire window is considered to be the laser aperture. While Class 1 lasers are considered to be "eye safe", avoid prolonged direct view-

ing of the laser and do not use optical instruments to view the laser.

- When operated in an ambient temperature >40 °C, the metallic surfaces of the sensor may be hot enough to potentially cause skin burn. Avoid skin contact with the sensor's base, lid and the heatsink when the sensor is operated under these conditions. The sensor should not be used in an ambient temperature above 60°C. The maximum safety certified ambient operating temperature is 60°C.

PRECAUTIONS:

- Tous les capteurs Ouster sont des unité hermétiquement scellée, qui ne peut être entretenue ou modifiée par l'utilisateur.
- L'utilisation de commandes, de réglages, ou l'exécution de procédures autres que celles spécifiées dans le présent document peuvent entraîner des rayonnements laser dangereux.
- L'utilisation d'un capteur Ouster est soumise aux conditions de vente signées avec Ouster ou le distributeur/intégrateur, incluant l'interdiction de:
    - Retirer ou ouvrir de quelque façon le boîtier du capteur
    - Analyser les composants internes du capteur
    - Pratiquer la rétro-ingénierie de toute ou partie du capteur
    - Autoriser une tierce personne à mener les actions listées ci-dessus
- L'utilisation du capteur sans le support (fourni avec les capteur) ou sans contact avec une surface ayant des capacités thermiques adéquates peut entraîner une surchauffe du capteur dans certaines conditions.
- Ce capteur présente une conception avec un dissipateur thermique supérieur modulaire, ceci pour apporter plus de flexibilité de montage et d'intégration au capteur.
- Cette conception modulaire augmente la flexibilité de conception mais ne supprime pas le besoin de dissipation thermique au-dessus du capteur. Le dissipateur thermique radial fourni est essentiel à une bonne gestion thermique. Le capteur ne fonctionnera pas correctement sans cette pièce.
- Une utilisation prolongée du capteur sans le dissipateur thermique supérieur peut résulter à des erreurs système ainsi qu'à une surchauffe du capteur pouvant aller jusqu'à son extinction. Le dissipateur thermique fourni peut être remplacé par une autre solution de dissipation thermique adéquate, mais ne doit pas être simplement retiré.
- Si vous souhaitez utiliser votre capteur avec une dissipation thermique personnalisée, merci de contacter notre Équipe Support qui pourra répondre à vos questions et vous apporter le support et le conseil nécessaire.
- Ce produit émet un rayonnement laser invisible de classe 1. L'ouverture de sortie du laser est constituée par la fenêtre du capteur dans sa totalité. Même si les lasers de classe 1 ne sont pas considérés comme dangereux pour les yeux, ne regardez pas directement le rayonnement laser de façon prolongée et n'utilisez pas d'instruments optiques pour observer le rayonnement laser.
- Lors d'une utilisation à température ambiante supérieure à 40°C, la surface métallique du cap-

teur peut présenter des risques de brûlures pour la peau. Dans ces conditions, il est important d'éviter tout contact avec la partie supérieure, la base ou le dissipateur thermique du capteur. Le capteur ne doit pas être utilisé à une température ambiante supérieure à 60˚C. 60˚C est la température maximale certifiée d'opération sûre du capteur.

Equipment Label: Includes model and serial number and a notice that states the unit is a Class 1 Laser Product, is affixed to the underside of the Sensor Enclosure Base. It is only visible after the attached mount with which the Sensor is shipped, is removed. For location details please refer to the hardware user manual.

L'étiquette de l'équipement, comprenant le modèle, le numéro de série, et la classification du produit laser (ici, classe 1), est apposée au-dessous de la base du boîtier du capteur. Il n'est visible qu'après avoir retiré le diffuseur de chaleur avec lequel le capteur est expédié. L'emplacement est décrit précisément dans le manuel d'utilisation du matériel.

Electromagnetic Compatibility: The Ouster sensors are an FCC 47 CfR 15 Subpart B device. This device complies with part 15 of the FCC Rules. Operation is subject to the following conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

"Ouster" and Ouster sensors are both registered trademarks of Ouster, Inc. They may not be used without express permission from Ouster, Inc.

If you have any questions about the above points, contact us at legal@ouster.io.

1.2 Proper Assembly, Maintenance and Safe Use

All Ouster sensors can be easily set up by following the instructions outlined in the hardware user manual. Any mounting orientation is acceptable. Each sensor is shipped with an attached mount that can be used for test or normal use within the specified operating conditions. The sensor may also be affixed to any other user specific mount of appropriate thermal capacity. Please contact Ouster for assistance with approving the use of user specific mounting arrangements.

Any attempt to utilize the sensor outside the environmental parameters delineated in the relevant data sheet for your Ouster sensor may result in voiding the warranty.

When power is applied, the sensor powers up and commences boot-up with the laser disabled. The boot-up sequence is approximately 60s in duration, after which the internal sensor optics subassembly commences spinning, the laser is activated, and the unit operates in the default 1024 x 10 Hz mode. When the sensor is running, and the laser is operating, a faint red flickering light may be seen behind the optical window.

Note: All Ouster sensors utilize an 865nm infrared laser that is only dimly discernible to the naked eye. The sensor is fully Class 1 eye safe, though Ouster strongly recommends against peering into the optical window at close range while the sensor is operating. Ouster sensors are equipped with a multi-layer series of internal safety interlocks to ensure compliance to Class 1 Laser Eye Safe limits.

All Ouster sensors are hermetically sealed units, and are not user-serviceable. Any attempt to unseal the enclosure has the potential to expose the operator to hazardous laser radiation.

The sensor user interface may be used to configure the sensor to a number of combinations of scan rates and resolutions other than the default values of 1024 x 10 Hz resolution. In all available combinations, the unit has been evaluated by an NRTL to remain within the classification of a Class 1 Laser Device as per IEC 60825-1:2014 (Ed. 3).

1.2.1 Assemblage correct et utilisation sûre

Tous les capteurs Ouster s'installe facilement en fixant la base sur un support percé de trous concordants, et en suivant les instructions d'interconnexion décrites dans le manuel d'utilisation du matériel. Toute orientation de montage est acceptable. Chaque capteur est expédié équipé d'un dissipateur de chaleur, utilisable en phase de test et en conditions normales. Néanmoins tout autre support présentant une capacité thermique appropriée pour l'application de l'utilisateur peut être utilisé. Veuillez contacter Ouster dans le cas où un montage spécifique à votre application serait nécessaire.

Toute tentative d'utilisation du capteur en dehors des paramètres environnementaux définis dans la fiche technique de votre capteur Ouster peut entraîner l'annulation de la garantie.

Lorsque le capteur est sous tension, celui-ci démarre et commence son initialisation avec le laser désactivé. Le temps de démarrage est d'environ 60s, après quoi le sous-système optique entre en rotation et le laser est activé, le capteur opère alors dans son mode par défaut de 1024 x 10 Hz. Lorsque le capteur est en marche et que le laser est activé, on peut apercevoir une faible lumière rouge vacillante derrière la vitre teintée. Tous les capteurs Ouster utilisent une longueur d'ondes infra-rouge de 865nm à peine perceptible pour l'œil humain, et le rayonnement laser IR émis est sans danger pour les yeux. Cependant, bien que les rayonnements laser de classe 1 soient sans danger dans des conditions raisonnablement prévisibles, Ouster recommande fortement de ne pas regard er fixement la vitre teintée pendant que le capteur est en marche. Tous les capteurs Ouster sont des unités hermétiquement scellées, qui ne peuvent pas être entretenues, modifiées ou réparées par l'utilisateur. Toute tentative d'ouverture du boîtier a pour risque d'exposer l'opérateur à un rayon-nement laser dangereux.

Les capteurs Ouster sont équipés d'une série de dispositifs de sécurité à plusieurs niveaux, de façon à assurer en toutes circonstances le respect des limites d'irradiance correspondant aux rayonnements lasers de classe 1, sans danger pour les yeux.

L'interface utilisateur du logiciel du capteur peut être utilisée pour configurer le capteur selon un certain nombre de combinaisons de vitesses de balayage et de résolutions autres que les valeurs utilisées par défaut, respectivement de 1024 x 10 Hz.

2 Ouster Detect Introduction

Ouster Detect is a perception software suite that `detects`, `tracks`, and `classifies` objects from the Ouster digital lidar sensor point cloud streams. Ouster Detect is the foundational building block of Ouster Gemini Smart Infra Solutions platform.

Ouster Detect detects objects, their current geo-position, speed, and direction of movement with a high degree of accuracy. Customers can ingest this data that are commonly referred to as "object list", as JSON stream over either MQTT or TCP protocols into their custom business logic pipelines converting the 3D lidar detection data into actionable business outcomes. Ouster Detect reduces integration time for customers by providing a highly configurable set of options for filtering, down-sampling, and basic aggregation of object data by zone or class-type. Ouster Detect is a containerized software product that can be deployed on customer-provided or Ouster-provided edge compute nodes. It supports all Ouster digital lidar sensor models (Rev6 and later), channel configurations (128 & 64), and beam configurations (below/above/uniform).

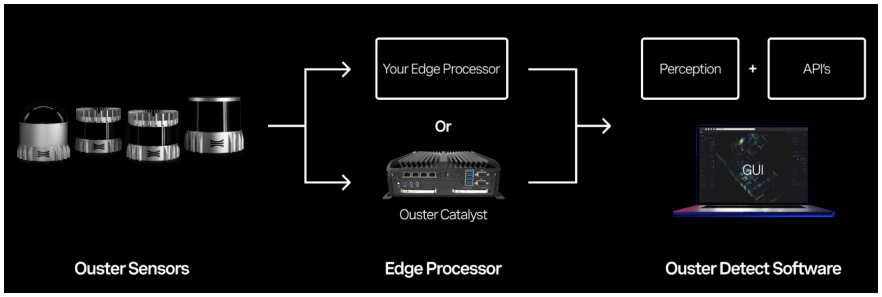


Figure 2.1: Ouster Gemini Overview

This Ouster Detect User Manual provides in-depth information on how to install, setup, and configure Ouster Detect for various customer uses.

2.1 Additional Technical Documentation

The Ouster sensor documentation is meant to allow the users to take advantage of all the features that are available with Ouster Sensors.

- Ouster Python SDK
- Connecting to Sensor
- Firmware User Manual
- HTTP API and TCP-API.
- Hardware User Manual

For more information, please visit Ouster Detect or contact our Field Application Team.

3 Using an Edge Processor

3.1 Requirements

Ouster Detect requires optimized AI Compute engines for multi-sensors spatial intelligence, Ouster recommends the use of one of two types mentioned below:

Note: These edge computers can also be purchased from Ouster, please reach out to Ouster Sales.

3.1.1 Catalyst Pro

Highlights

- Intel® Core™ i7-9700TE 8 Core processor onboard
- 16GB RAM
- 256GB SSD Storage
- 6x GbE LAN RJ45 Ports
- 8 in / 8 out (Isolated) GPIO
- Power switch, remote switch, AT/ATX select
- Status LEDs (HDD, Power, Network)
- Operating temperature: -25 ºC ~ 70 ºC
- Dimensions: 240 (W) x 261 (D) x 79.2 (H) mm

Figure 3.1: Catalyst Pro

3.1.2 Catalyst Lite

Highlights

- Intel® Celeron® J1900 4 Core processor onboard
- 8GB RAM
- 128GB SSD Storage
- 2x GbE LAN RJ45 Ports
- 4 in / 4 out (Isolated) GPIO
- Power switch, remote switch, AT/ATX select
- Status LEDs (HDD, Power, Network)
- Operating temperature: -25 ºC ~ 70 ºC
- Dimensions: 150 (W) x 105 (D) x 49 (H) mm

Figure 3.2: Catalyst lite

The Catalyst devices have a number of built-in Ethernet interfaces on both the front and the back of the unit. Several of these interfaces come pre-configured and are intended for specific network connections.

3.2 Connecting Sensors to the Edge Processor

Sensors can be connected directly to edge processors or indirectly via other networking hardware such as a network switch. If other networking hardware is used to connect multiple Ouster sensors to a single Catalyst interface; the bandwidth of the interface and hardware must be carefully considered.

The Catalyst-Pro has 4 available ethernet ports for sensor connections (LAN4-7) while the Catalyst-lite has one (LAN2). These designated ports will automatically distribute DHCP network addresses if a sensor is configured for DHCP use (default for Ouster sensors).

Ouster sensors will always broadcast an mDNS hostname when directly connected to a Catalyst de-

vice. If other networking devices are between the sensor and Catalyst, these devices will need to allow mDNS traffic. The mDNS hostname of Ouster sensors has the form `os-XX.local` where `XX` is the sensor serial number.

It is recommended to use the mDNS hostname to communicate with sensors when available as they will never change.

Table 3.1: RCO-6000

Port	Address Range
LAN 3	10.125.20.2-10.125.20.22
LAN 4	10.125.21.2-10.125.21.22
LAN 5	10.125.22.2-10.125.22.22
LAN 6	10.125.23.2-10.125.23.22

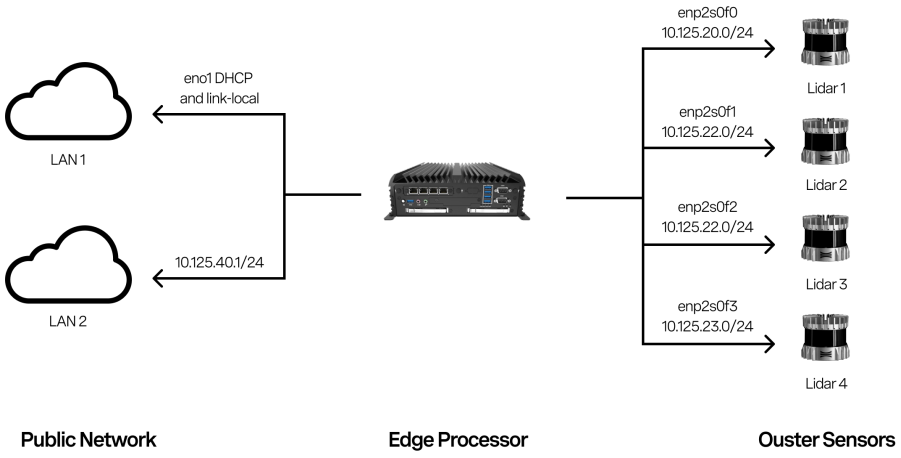


Figure 3.3: RCO-6000

Table 3.2: RCO-1000

	Port	Address Range
RCO-1010A	LAN 2	10.125.20.2-10.125.20.22

Figure 3.4: RCO-1000

3.3 Connecting to the Edge Processor

`LAN1` on both the Catalyst-Lite and Catalyst-Pro is designated for connection to a users network. This interface will accept a DHCP address if connected to a network with a DHCP server. This interface will also broadcast an mDNS hostname if connected to a network which supports mDNS traffic. The hostname will be of the form `edgeproc-XX` where `XX` is a hash unique to the device.

4 Using 3rd Party Hardware

4.1 Computer Requirements

When running **Ouster Detect** on customer-provided hardware, it is important to ensure that the hardware specifications meet the minimum required specifications for the software. The specifications depend on the number of lidars in use with a particular computer. Ouster Detect is built to run on 64-bit Intel or AMD CPUs. The number of required cores scales with the number of lidars that will be used with the system.

Ouster Detect requires *n* + 1 cores, where *n* is the number of lidars in use on a system. For each lidar, Ouster Detect requires 2 GB of RAM. With this in mind, a computer connected to three lidars would require four CPU cores and 6 GB of RAM. Each core must be at least equivalent to a 9th generation Intel Core i7.

Note: Ouster Detect currently is limited to use on a Linux based operating system, Ubuntu 20.04 or later.

If your computer hardware does not meet the requirement, please refer to *Requirements*.

4.2 Installing Docker

Note: When installing Docker, Ouster strongly recommends the instructions provided. Docker packages from `snap package manager` and `Docker compose V1` are not compatible. However, Docker compose v2 is compatible.

Ouster Detect is designed using a microservices architecture. The system is broken up into a number of component services, most of which run inside of docker containers. As such, docker must be installed and running on all computers on which Ouster Detect will be used. In addition to the docker engine, docker compose must also be installed. Compose is used to orchestrate the set of containers that make up Ouster Detect's functionality.

The docker website provides instructions for installing the docker engine and docker compose for a variety of supported platforms. If the customer does not already have a Linux-based operating system installed on their computer, Ouster recommends installing a recent version of Ubuntu Server edition. The docker website provides instructions specific to installing docker on Ubuntu systems.

Once the docker engine is installed, Ouster recommends completing the docker Linux post-installation instructions. This will ensure that the user account used by the customer will be able to interact directly with the docker daemon.

4.3 Installing the Ouster Agent

The Ouster Agent is responsible for handling software licensing and generating unique identifiers for each computer running Ouster Detect. The Agent is distributed as a Debian package. This package may be installed on any modern Debian-based Linux distribution that includes `systemd` version 236 or newer. The docker daemon must be installed on the customer computer prior to installing the Ouster Agent.

The Agent Debian package installs both the Ouster Agent daemon, which is managed by `systemd`, and a software license server daemon, also managed by `systemd`. In addition, the Agent package installs a docker compose file. This file is used to orchestrate the set of docker containers that make up Ouster Detect and is located at `/opt/ouster/compose.yaml`. The customer must use this file to install, update, start, and stop Ouster Detect.

4.4 Installing Ouster Detect Docker Images

Once both docker and the Ouster Solutions Agent have been successfully installed, the user must next retrieve the set of docker images that make up Ouster Detect.

These images are distributed through a docker registry, please contact Ouster Support if you have more questions.

4.4.1 Install Detect from Compressed Archive

- Contact Ouster sales representative or Field Engineer for access to the latest Detect compressed archive.
- Install docker engine on the host machine. See Docker Engine Installation for more information.
- Extract the Detect archive to a directory on the host machine
  - *tar xvf /path/to/Detect-{OS}-{version}.tar.gz*
- Move into the extracted directory and run the Detect script
  - *./setup.sh*
- `Optional` validate the install
  - *docker ps* will show the Detect containers running with a tag matching the version of Detect you installed.
  - *dpkg -l ouster-solutions-agent* will show the version of the Agent you installed.
  - *systemctl status ouster-solutions-agent* will show the the Agent service running.

Note: When updating the images, Ouster Detect must be restarted in order to take advantage of the updated software.

The next section addresses how to start, stop, and restart Ouster Detect.

4.5 Running Docker Images

With the Ouster Detect's docker images retrieved, the system may be started.

All commands in this section assume that the user is in the directory `/opt/ouster`, which is where the `compose.yaml` file is located. To start the Ouster Detect system, the user must execute the following command,

```
$ docker compose up -d
```

This command will create containers from each of the Ouster Detect docker images, create a docker volume where persistent files will be stored, and connect all containers together on a docker bridge

network. All containers make use of docker's built-in logging functionality. To view the container logs, the user must execute the following,

```
$ docker compose logs -f
```

This command will display the history of logs and also continue to display new log entries as they arrive from Ouster Detect. The user may exit viewing the logs by issuing a `CTRL-C`.

Once the Ouster Detect containers are brought up for the first time, the system is configured to restart each time the computer reboots. If the user would like to stop the containers manually, the following command must be issued from the `/opt/ouster` directory,

```
$ docker compose down
```

When a new version of Ouster Detect has been retrieved, the user must restart Ouster Detect in order take advantage of the updated software. Ouster Detect may be restarted by issuing the following command,

```
$ docker compose restart
```

5 Getting Started with Ouster Detect

5.1 Connecting to the Ouster Detect GUI

Once the Catalyst hardware is successfully connected to the customer network, or once the customer-supplied computer is connected to the network and the Ouster Detect software has been installed and started, the user may then connect to the computer using a web browser. This will allow the user to begin setting up Ouster Detect. For the best browser experience, Ouster recommends that the customer use a computer with a discrete GPU and the Google Chrome browser.

Assuming that the computer running Ouster Detect has an IP address of `192.168.1.1`, the user may establish an initial connection to Ouster Detect by pointing their browser at `https://192.168.1.1`.

When accessing the Detect Viewer, user may see a warning that the site is not secure. This is because the Detect Viewer is served using a self-signed certificate. If you are using Chrome, you can bypass this warning by clicking on the `Advanced` link and then clicking on `Proceed to {detect_url} (unsafe)`.

The web interface to Ouster Detect is protected using HTTP authentication. The user will be prompted for credentials when first accessing the GUI. Default credentials (case sensitive) are as follow:

- `Username:` **ouster**
- `Password:` **stone-pass-fill**

The password may be changed using the REST API. All calls made to Ouster Detect's REST API also require authentication with this username and password.

Note: Ouster Detect viewer is accessed through port 443. Please configure your networking accord-

ingly to enable communication through these ports.

5.2 Activating the Software License

Ouster Detect Software package will be sent to the customer along with the software entitlement ID. The entitlement ID is required to activate the Ouster Detect software license.

Example Key

```
f963870f-1c11-46e6-8639-479da2a5732f
```

Note: It is not possible to transfer a license from one machine to another. Your license is tied to your hardware. If you have any trouble activating the license, please reach out to Ouster Support.

6 GUI Overview

6.1 Layout

Ouster Detect has two distinct layouts, one that includes a 3D viewport and one without. These layout accommodate the different modes/sections of the app.

With a 3D viewport is divided into five areas.

- Header
- Left Pane
- Viewport
- Right Pane
- Feedback Line

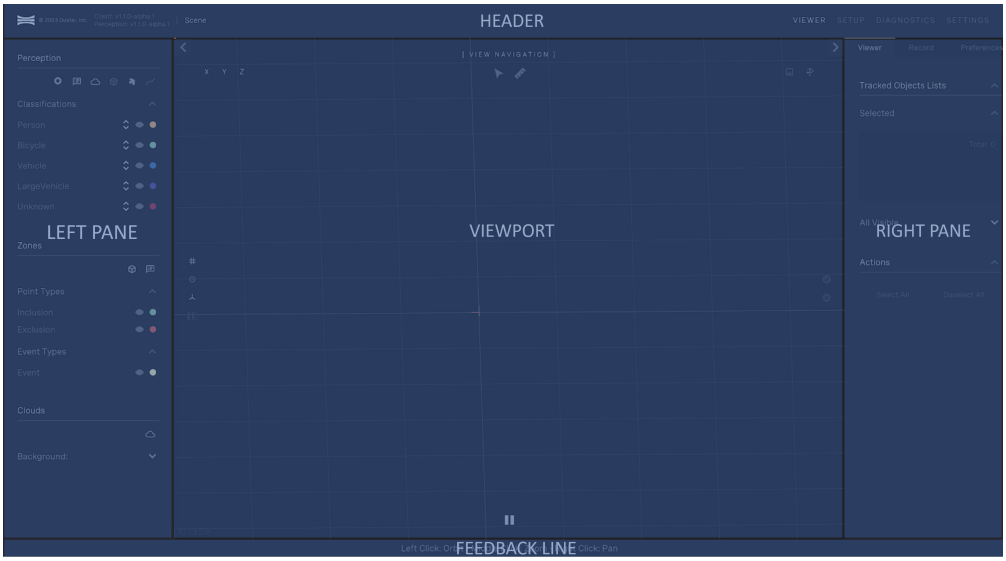


Figure 6.1: With 3D Viewport

Without a 3D viewport is divided into three areas.

- Header
- Content
- Feedback Line

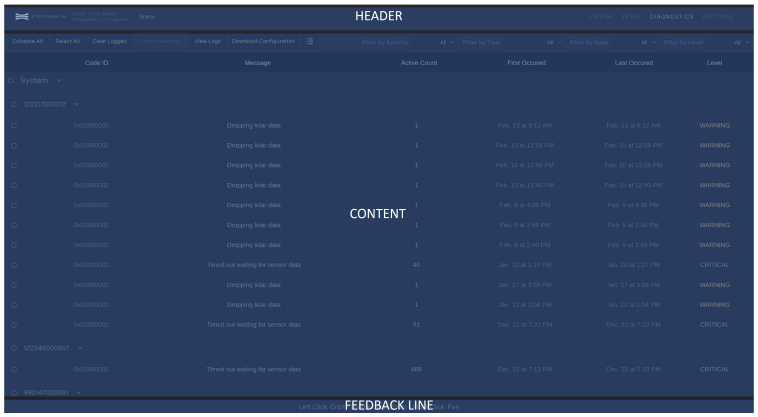


Figure 6.2: Without 3D Viewport

6.1.1 Header

Figure 6.3: Header Overview

The header contains application information, the local scene name and the navigation menu.

- The **Application Information** next to the logo displays the running front and backend end version. The user can copy to their systems pasteboard the version by clicking on the relative links
- The **Scene Name** is a user defined identifier for the viewing setup, its value would be reflected on the browser's tab name and is exposed under preferences.
- The **Navigation Menu** allow the user to navigate to a desired sections sucha as:
  - **Viewer**: Main viewing section of the perception output
  - **Setup**: Section that provides the user with tools and info to position the sensors and create/define zones
  - **Diagnostics**: Section with information for the status of the edge computer and the sensors attached
  - **Settings**: Section exposing all modifiable parameters for the system

6.1.2 Left Pane - Viewer

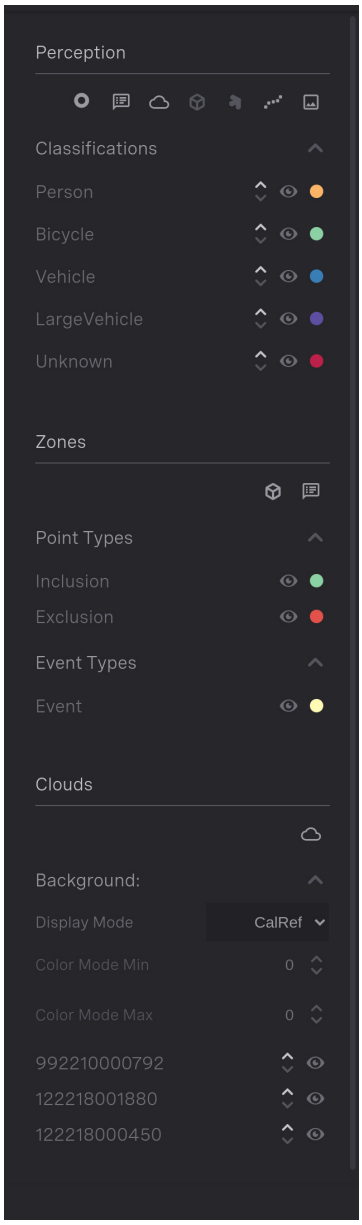


Fig. 6.4: Left Pane Perception

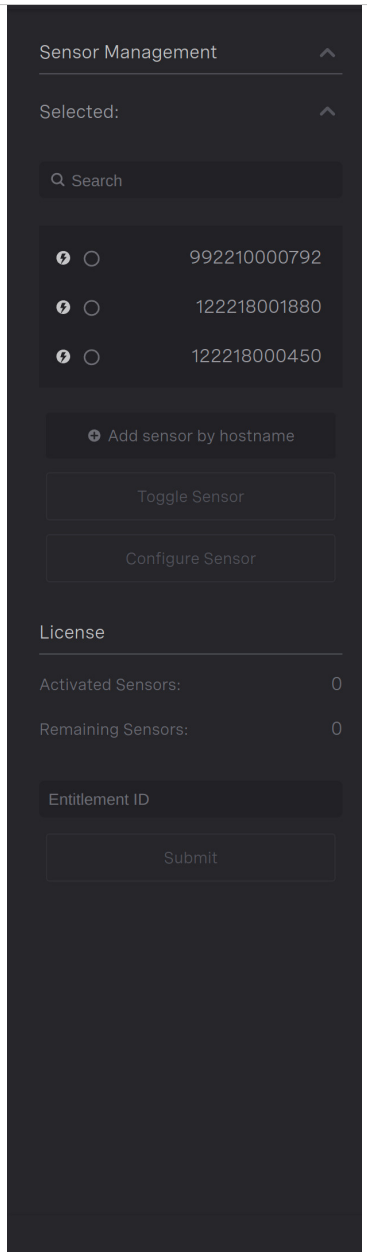


Fig. 6.5: Left Pane Sensor Management

In Viewer section: It will display options for the system's key structures - Perception, Zones and Clouds

- Perception: Hosts an array of icons to control the display of different perception elements and a list of the available classes with options to control their visibility, point size and color.
- Zones: Lists the available types with options to control their visibility and color.

- Clouds: Lists the active sensors with options their visibility, point size and color.

6.1.3 Left Pane - Setup

In Setup section: It will display options for configuring sensors to the system and the license section

6.1.4 Right Pane - Viewer

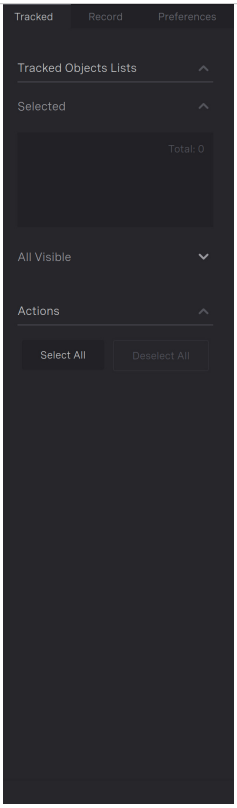


Fig. 6.6: Tracked Objects

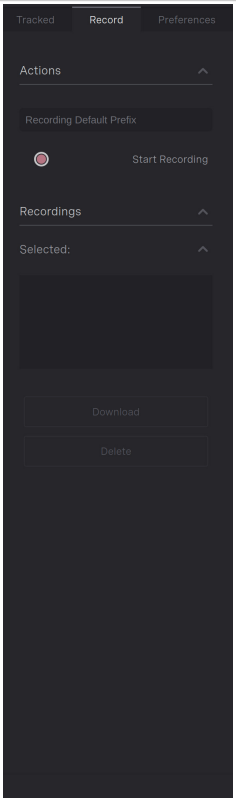


Fig. 6.7: Record

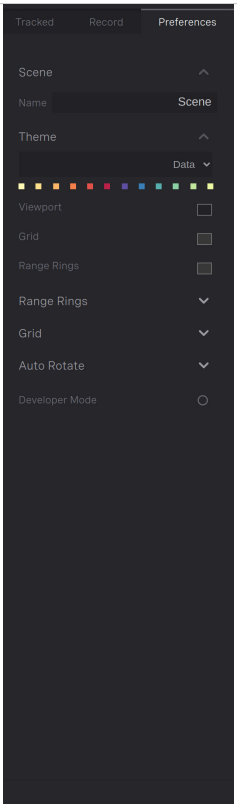


Fig. 6.8: Preferences

In Viewer section there are three tabs

- **Tracked**: Displays lists and actions to highlight extra information in the viewport. Two list are available, the currently selected set and the available objects present in the scene and toggle buttons to assist selecting/deselecting tracked objects.
- **Record**: Hosts lists and actions for recording the raw point clouds from the sensors.
- **Preferences**: User preferences options.

6.1.5 Right Pane - Setup

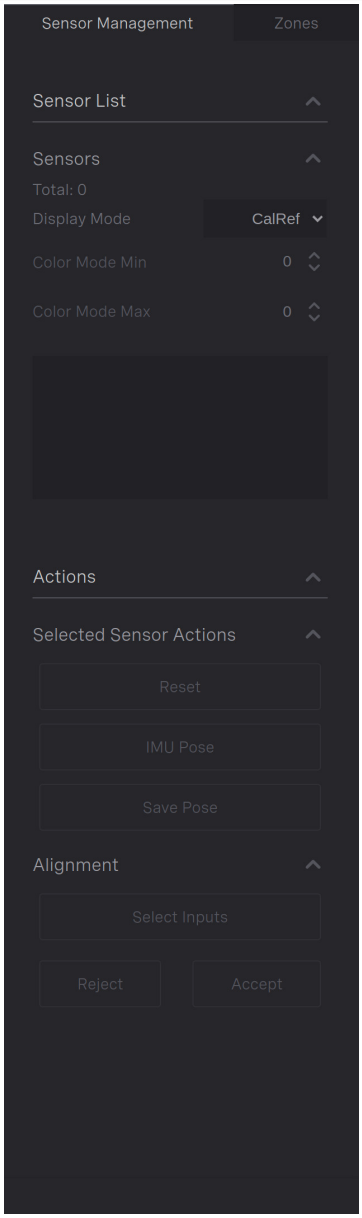


Fig. 6.9: Sensor Management

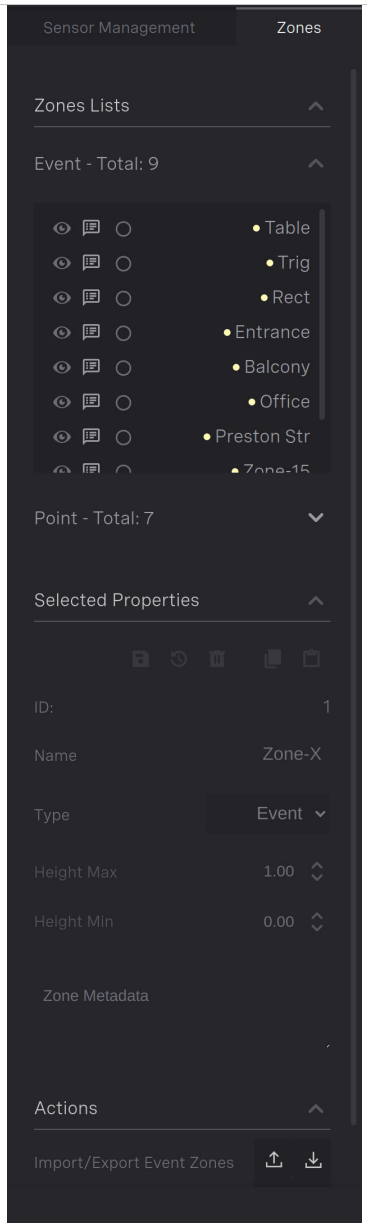


Fig. 6.10: Zones

In Setup section: It will host list and options for the sensor and zones

- **Sensor Management**: Lists connected sensors and actions to store/align them.
- **Zones**: Lists created zones and options to modify their properties.

6.1.6 Feedback Line

The feedback line provides information to the user when interacting with the scene.

Figure 6.11: Feedback

6.1.7 Viewport

The 3D world with an overlay of tools and options the user can interact with.

- **Top Left Corner** has an array of buttons to switch to a predefined direction.
- **Top Middle** has a panel that indicate the selected tool and the available tools for the section.
- **Top Right Corner** has a button to spin the scene around its interest point.
- **Middle Right Side** convenient actions buttons reflecting the actions section in the right panel.
- **Bottom Middle Side** playback controls.
- **Lower Left Corner** displays the current local time.
- **Middle Left Side** toggle buttons controlling the visibility helpful elements, cartesian grid, polar grid and unit axis.

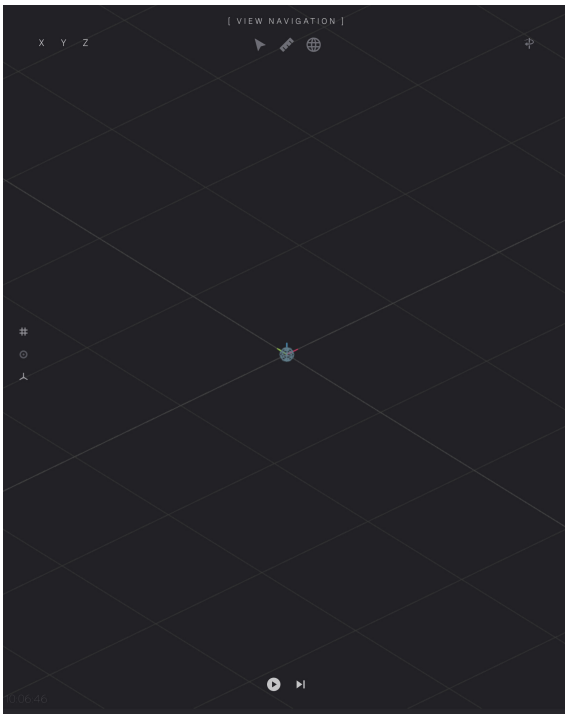


Figure 6.12: Viewport

6.1.8 Content

Sections with a content view are text based views with top row with buttons to control various desired actions.

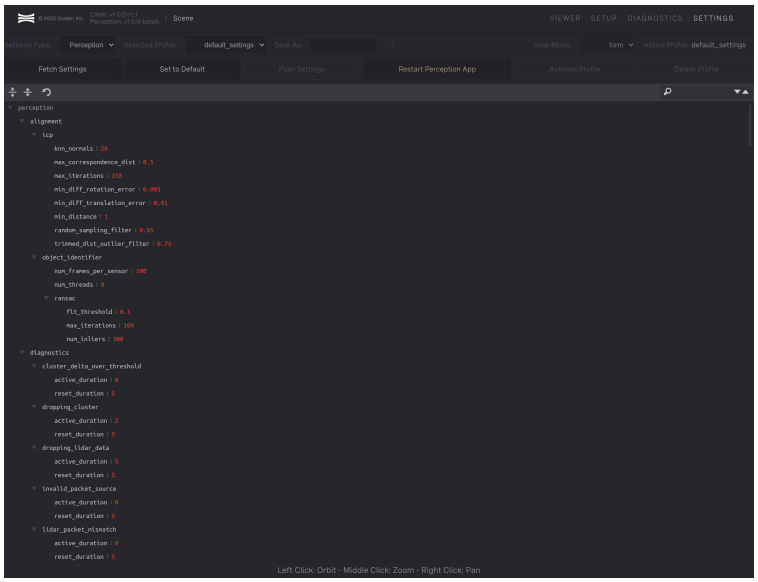


Figure 6.13: Content

6.2 Viewer

The **Viewer** is the section that the user can view the scene and select the output of the perception server.

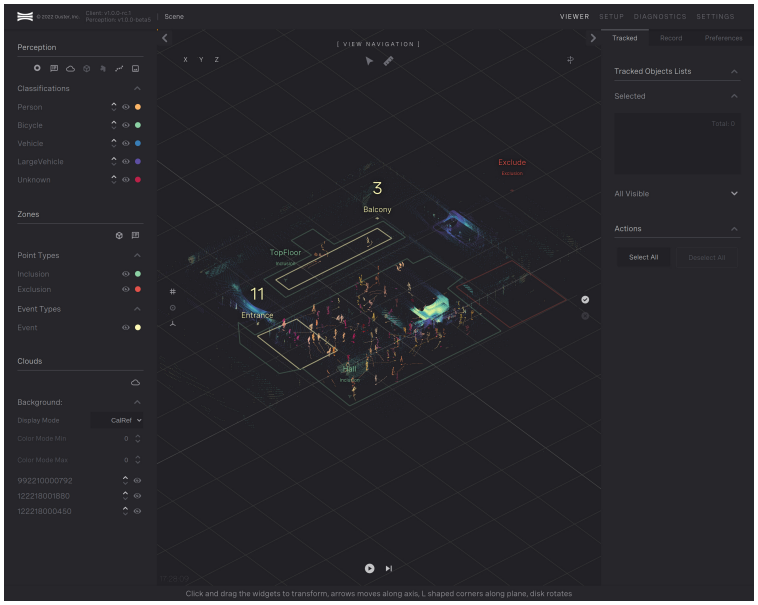


Figure 6.14: Ouster Detect's Viewer section

6.2.1 Tools

Select

Used to select one or more objects being tracked, and when selected, their info bubble is displayed and listed under Tracked Objects List on the right.

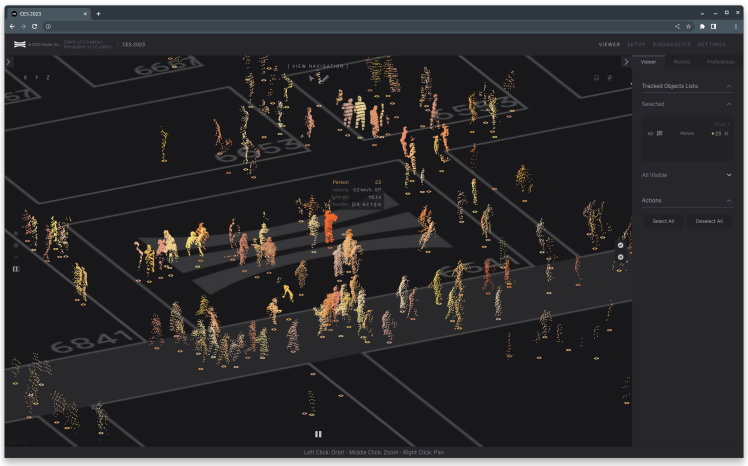


Figure 6.15: Viewer Select

Measure

The measure tool allows to measure distances and display the coordinates of the ground point under the mouse cursor.

Figure 6.16: Ouster Detect's viewer tool options

6.2.2 Left Pane

Dedicated areas for the system's key structures - **Perception**, **Zones** and **Clouds**. Please refer to *Left Pane - Viewer*.

6.2.3 Perception

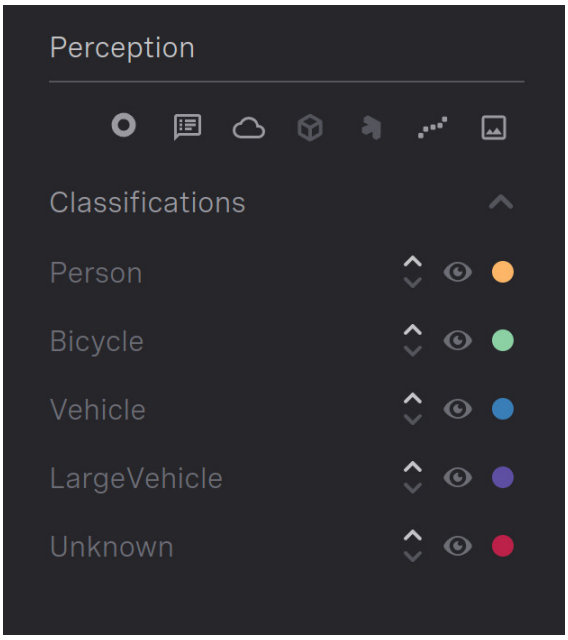


Figure 6.17: Perception

A list of the available classes with options to control their visibility, point size and color.

Perception: Hosts an array of icons to control the display of different perception elements, from left to right:

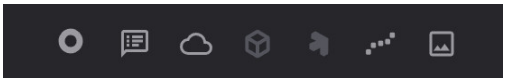


Figure 6.18: Perception Controls

- **Ring icon** - toggles the visibility of the tracked object's location.
- **Bubble icon** - toggles the visibility of the tracked object's informational labels.
- **Cloud icon** - toggles the visibility of the tracked object's point clouds.
- **Cube icon** - toggles the visibility of the tracked object's bounding boxes.
- **Trident icon** - toggles the visibility of the tracked object's top corners.
- **Dotted line icon** - toggles the visibility of the tracked object's trajectory.
- **Image icon** - toggles the visibility of the sensors near IR.

6.2.4 Zones

Lists the available types with options to control their visibility and color.

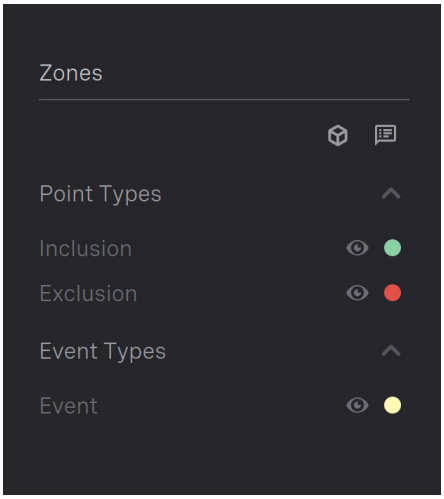


Figure 6.19: Zones

6.2.5 Clouds

Lists the active sensors background point clouds with options to control their visibility, point size, color and colorization method.

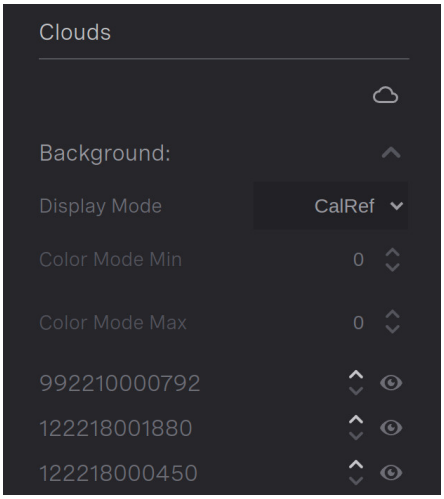


Figure 6.20: Zones

6.2.6 Right Properties Pane

Displays lists and actions to highlight extra information in the viewport. Two list are available, the currently selected set and the available objects present in the scene and toggle buttons to assist selecting/deselecting tracked objects

For more information, please refer to *Right Pane - Viewer*.

6.3 Zones

Zones can be configured and viewed throughout the GUI. To go to the zone editing page choose Setup/Zones.

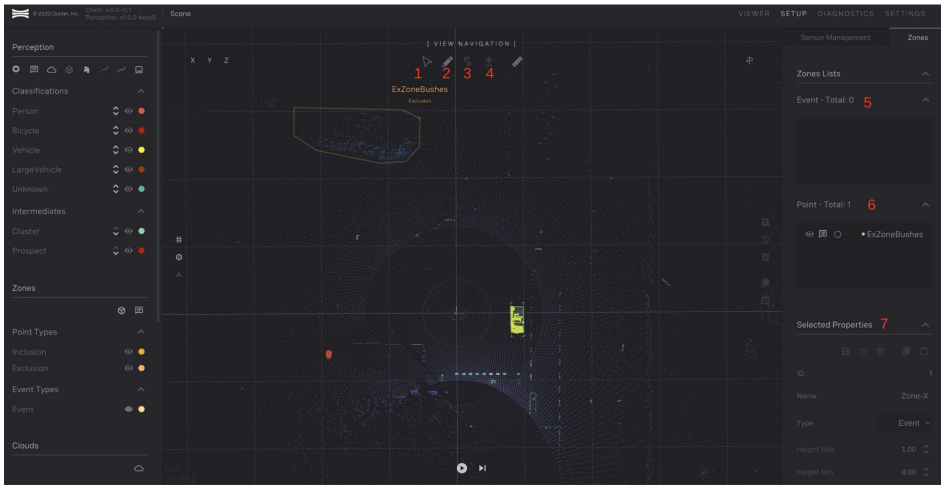


Figure 6.21: Zone configuration page

The tool icons at the top of the 3D viewer provide editing and creation options for zones in the scene. Different tool and panes are numbered in red. They are as follows:

- Tool to select new zones. You can then use the mouse to click on a select a specific zone.
- Tool to create a new Event, Inclusion or Exclusion zone.
- Tool to enable the zone editing tool. Requires a zone to be pre-selected. You can move zone vertices around with the tool.
- Tool to enable the zone the add/remove vertex tool. Requires a zone to be selected.
- Lists all the event zones. Zones can be selected through this pane.
- Lists all point zones. Zones can be selected through this pane.
- Pane for editing properties of specific zones. Also included in this pane is the ability to save, delete and copy zones.

6.3.1 Zone Workflow

A typical workflow for zones creation occurs after the sensors have been setup. Steps are:

- Identify if there are areas that are not required for object tracking. This can include vegetation or area's where there are objects not of interest. Both exclusion and inclusion zones can be used to set this up. Exclusion zones takes priority, and nothing will be detected inside an exclusion zone. Inclusion zones can be used to pick specific area's of interest. Note that if there are any inclusion zones then only points in inclusion zones will be included.
- Draw the zones based on your analysis from a top down view. Edit the zone min and max height in the properties pane. Name them appropriately for situational awareness.
- Create event zones based on area's where you want specific notification of objects. Set a descriptive name.
- Make any modification to the zones as required based on tracked objects.

6.4 Recording

The **Recording** tab of Ouster Detect provides the capability to capture raw point clouds, in `.pcap` format, and metadata for an installation. The dataset can be downloaded along with the current Ouster Detect settings and the sensor metadata for each connected sensor. This tab can be navigated to by clicking `Viewer`, then `Record`.

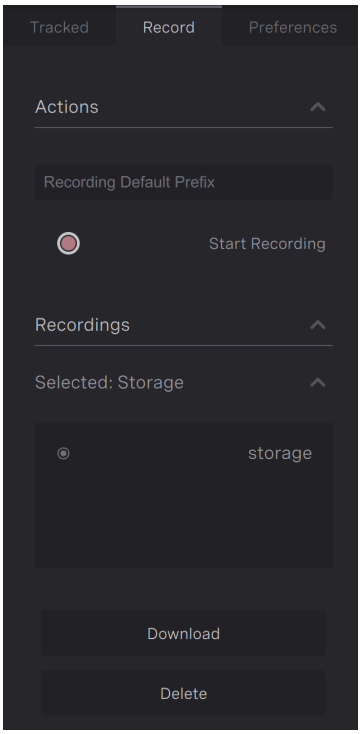


Fig. 6.22: Recording Pane

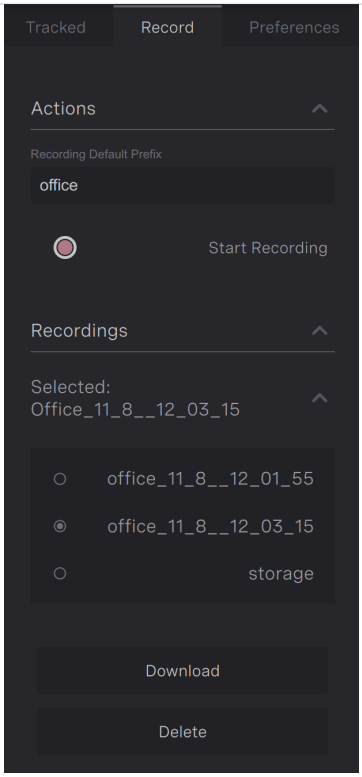


Fig. 6.23: Recording Example

6.4.1 Recording a PCAP

To record a PCAP, click the red icon next to *Start Recording*. This will begin the recording of the raw point cloud from all sensors. To stop the recording, click the *Stop Recording* button that is present when a recording is currently ongoing. The recorded file will be listed in the *Recordings* list.

The prefix for the recorded filename can be set within the *Recording Default Prefix* textbox. The resulting file format is the defined prefix, the date of recording, and the time the recording commenced. If no prefix is supplied, the filename will be just be the date and recording time.

An example of the resulting recorded data is shown in the image below.

- `office` was selected as the prefix, resulting in a file called `office_11_8__12_01_55`.

6.4.2 Downloading a Recording

Once the data has been recorded, the data can be downloaded by selecting the recording in the list and clicking the *Download* button. Three types of files will be downloaded:

- `.pcap` file, storing the point cloud streamed data directly from the sensors,
- `.tar` file, representing Ouster Detect's settings at the time of recording,
- one or more `.json` files, representing the sensor metadata files which are necessary for playback.

6.4.3 Deleting a Recording

To delete a recording from disk, select a recording to discard in the recording list, then click the `Delete` button. The recording will not be recoverable afterwards.

6.4.4 Important Recording Notes

It is important to note that PCAP Recording only records the raw point cloud data. It does not record objects within the scene.

6.5 Sensor Management

The sensor management tab is responsible for the setup of sensors. This section provides an overview for adding and removing sensors, sensor configuration, and sensor alignment. This tab can be navigated to by clicking `Setup`, then `Sensor Management`.

6.5.1 Adding Sensors

To add sensors, the user must have already inputted a valid active license.

Refer to section *Activating the Software License* for these steps.

The user will not be able to add sensors greater than their license allows.

Assuming that the user's license supports additional sensors, the left panel shows a list of the discovered sensors on the local network which can be connected to Ouster Detect.

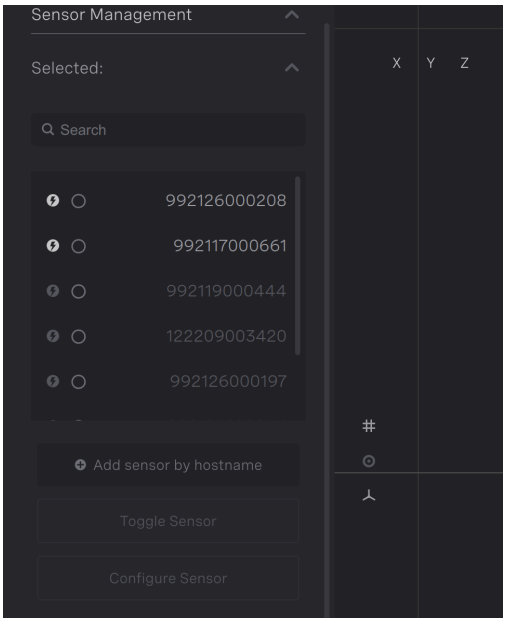


Figure 6.24: Adding Sensors Panel

The user can use the search bar to filter the sensors either by hostname or ip address. To add a sensor, select one of the sensors in the left panel, and then click **Add Sensor**. The sensor will take about ten seconds to be added. Bolded text in the panel indicates that the sensor is already connected to the system. Once the sensor is added, the sensor's point cloud should be viewable in the view pane.

If the sensor's hostname or ip address is known, the user can skip sensor discovery by clicking *Add sensor by hostname*, and then type in the sensor's ip address or hostname.

6.5.2 Removing Sensors

To remove a sensor from Ouster Detect, select a bolded sensor hostname in the left panel, and then click *Remove Sensor*. The point cloud will no longer be visible in the view pane.

6.5.3 Configuring Sensors

To view the sensor's configuration page, select the sensor in the left panel and click *Configure Sensor*. This will open the sensor's dashboard, where it allows the user to update the firmware and edit the sensor's configuration.

Click the **Documentation** panel to see more information.

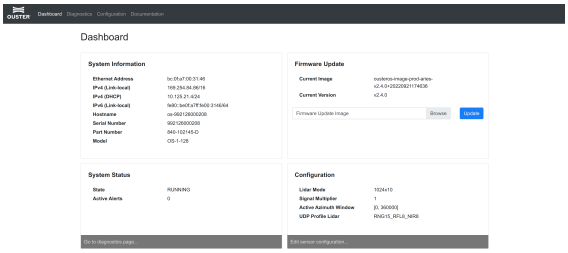


Figure 6.25: Sensor Configuration Page

6.5.4 Sensor Alignment Tools

After sensors have been added to Ouster Detect, the next step is to align the sensors. The right panel shows a list of attached sensors to Ouster Detect. Each sensor has four buttons: *Toggle visibility*, *Adjust Point Size*, *Set as Reference*, and *Adjust Transform*.

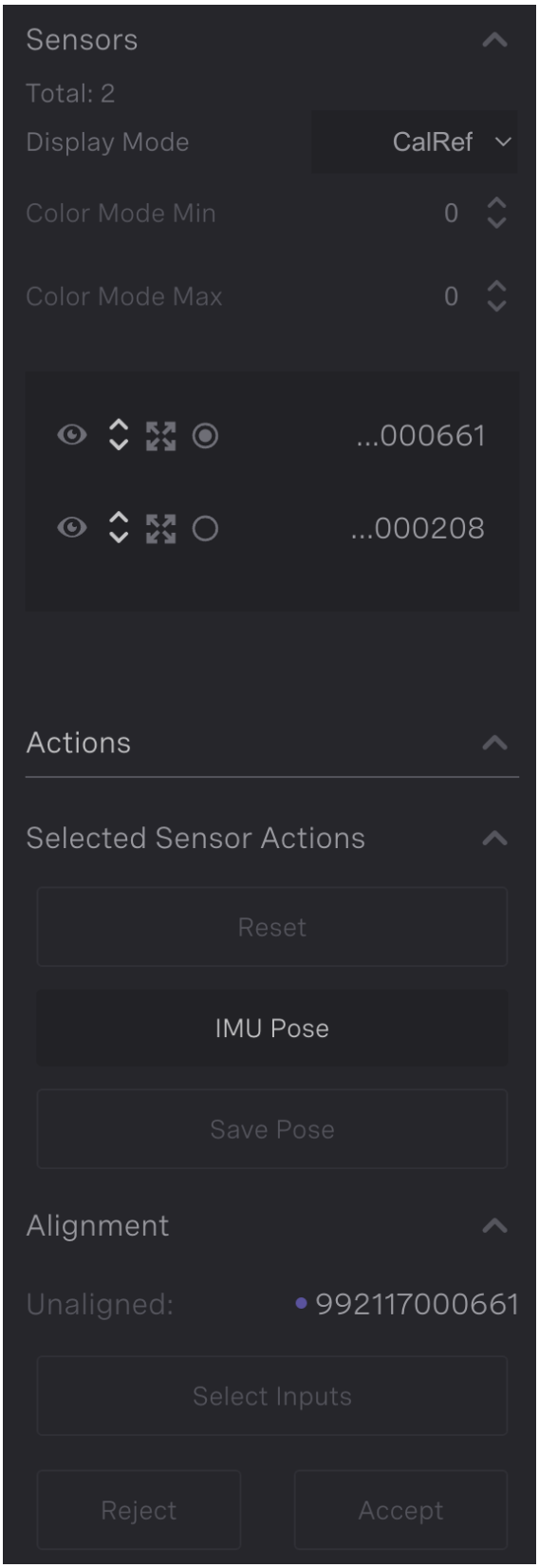


Figure 6.26: Sensor List and Alignment

An image of the alignment tool is shown below.

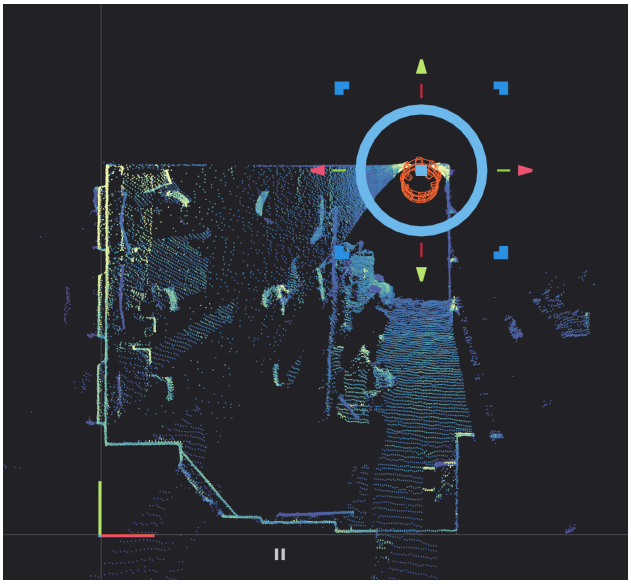


Figure 6.27: Alignment Tool

Using the Alignment Tool

The alignment tool is used to translate or rotate the point cloud. Clicking, holding then sliding the arrows will result in a translation. Clicking, holding then sliding the circle will result in a yaw rotation.

It is possible to also adjust the pitch and roll of the point cloud. The circles are hidden by default, but the hotkey *L* toggles the pitch and roll circle visibility.

IMU Pose and Pitch and Roll Alignment

The **IMU Pose** button aligns the transform with the positive z-axis pointing in the up direction relative to gravity. Click *IMU Pose* to align the sensor up relative to gravity. Then, click *X* to view the side view of the point cloud. If there appears to be some error, click *L* to show the pitch and roll circles, and adjust the roll rotation until precise. Then repeat this for the pitch, by clicking *Y*, and adjusting the pitch using the alignment tool. It is recommended to check the pitch and roll for alignment of the clouds, since the IMU inherently has small measurement error. It also will ensure proper alignment between point clouds from different LiDARs.

Automatic Alignment Tool

This tool automatically aligns two clouds together that have some overlapping field of view. It first requires that the two clouds are roughly aligned to allow the algorithm to converge. Then a cloud needs to be selected as a reference and another cloud as the unaligned cloud. The reference cloud will not move after the automatic alignment, while the unaligned cloud will move to align with the reference cloud. Select one cloud as the reference cloud by clicking *set as reference* for the cloud and then select the unaligned cloud by clicking *Adjust Transform*. Then, in the right panel, click *Align* to perform the automatic alignment. This will take about five seconds. Once the alignment is finished, the unaligned cloud will have moved, and the *Reject* and *Accept* buttons will highlight. If the alignment looks correct, click *Accept* to save the transform to disk. Sometimes, the tool may converge to a incorrect transform. In this circumstance, click *Reject* to reverse the alignment. Retry the automatic alignment again, and if it still fails, then resort to aligning using the manual alignment tool, or try again after better aligning the two clouds.

Saving and Resetting Transforms

To return the cloud to its original position, click *Reset* to return to the cloud to its originally saved orientation and position.

To save the transform, click *Save*. This will save the cloud's position to disk and will be updated in the viewer pane.

6.5.5 General Alignment Procedure

These are the general steps for LiDAR alignment.

- It is recommended to align a single sensor at a time. Toggle the visibility of other clouds to only show a single visible cloud. Select that point cloud in the right panel. The alignment tool will appear.
- Align the the transform with the positive z-axis pointing in the up direction relative to gravity. Refer to section *IMU Pose and Pitch and Roll Alignment* for this procedure.
- Use the alignment tool to adjust the yaw and translation for the point cloud. The position and yaw of a single point cloud can be set arbitrarily based on preference. While not required, it is recommended to set the point cloud's ground/floor inline with the x-y plane. This will allow the user to easily see the pitch and roll error of the point cloud, if the ground plane is flat.
- Click *Save Pose* to save the cloud's transform to disk. This will apply the transform in the viewer page.

Multiple LiDAR Installations

The following steps are for multiple LiDAR installations and are a continuation of the previous steps. The point clouds between LiDARs must be properly aligned together for cohesive object tracking.

- Show the visibility of a single point cloud which overlaps with the originally aligned reference cloud. If the two scenes do not have overlapping field of views, it's difficult to align the two clouds.
- Align the cloud with the positive z-axis pointing in the up direction relative to gravity using the steps outlined in the section *IMU Pose and Pitch and Roll Alignment*.
- Roughly align the cloud to the originally set cloud using the alignment tool. Typically, the x, y and z translation will need to be adjusted, as well as the yaw. Use physical references within the two scenes to align the two clouds together.
- Refer to section *Automatic Alignment Tool* and align a reference cloud to the current unaligned cloud. The reference cloud should be an already aligned cloud that has the most overlapping features with the unaligned cloud.
- Repeat these steps for any additional sensors to be added to the scene.

6.6 Diagnostics

Ouster Detect system and Ouster sensor diagnostics can be seen in the GUI **Diagnostics** tab. Details of the various Ouster sensor alerts can be found in the Ouster Firmware User Manual. Ouster Detect system alerts are documented in the Alert Data (ref to alert data section) section.

The **Diagnostics** alert table view can be filtered by status, alert code and alert level. *System* and *Sensors* sections can be collapsed to focus on specific alerts. Each alert can be expanded with further details.

The **Diagnostics** tab offers access to log files, configuration files and API testing tools (Swagger UI). Logs and configuration files can be downloaded for further investigation. Swagger UI webpages can be used as documentation of Ouster Detect APIs as well as a test platform to issue commands to running Ouster Detect components.

6.7 Settings

Ouster Detect's processing settings can be changed on the settings page. When the settings page is accessed current settings are fetched from the server.

Component on the settings page are numbered in red and are:

1. A selection drop down for either Perception or Lidar Hub settings.
2. A profile drop down to select from preconfigured profiles for different scenarios. Additional changes to profiles can be made but they provide a good starting point.
3. Button to grab the current settings from the the server. This will erase local changes that have been made to the settings

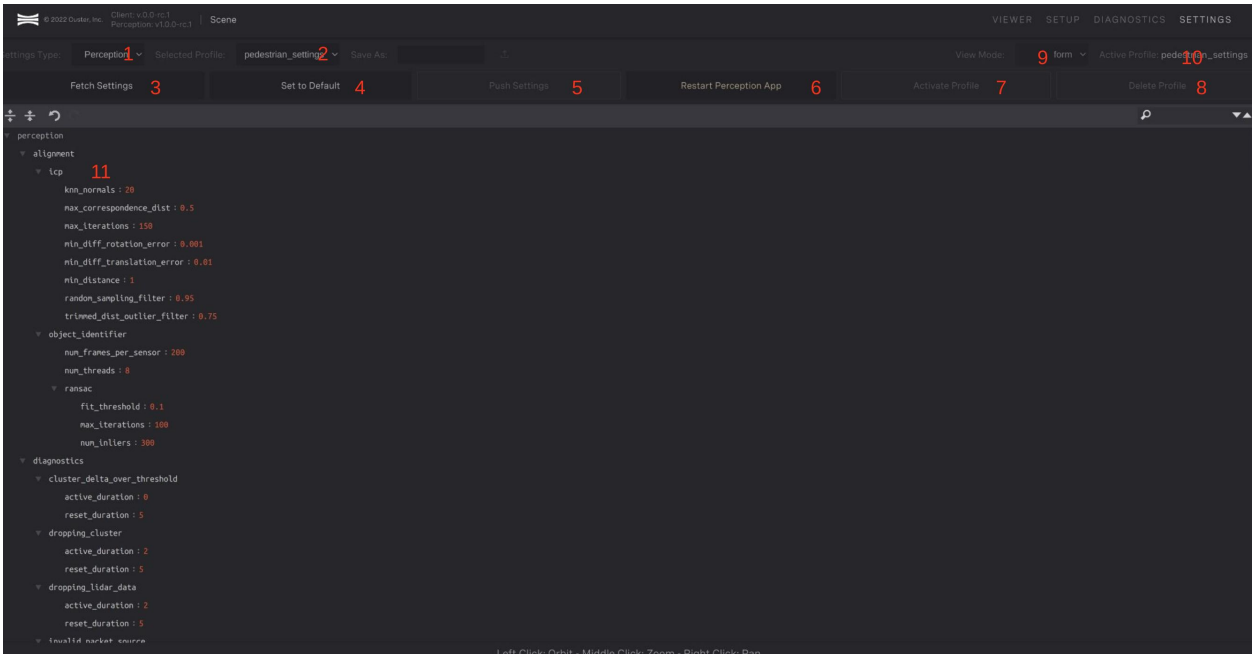


Figure 6.28: Settings Page

4. Button to reset the settings back to the default settings for the chosen profile
5. Button to push settings to the server
6. Button to restart the perception application
7. Button to change the settings on the based on the profile chosen
8. Button to delete the chosen profile
9. Chose between a tree and from view mode. The tree mode gives more control over settings, allowing you to add new ones and move them around to different sections. The form mode is more restrictive but prevents you from adding new settings that will not work. Tree mode is currently only available for LidarHub settings.
10. Indicator of the active profile on the server
11. JSON pane

6.7.1 JSON Pane

The JSON pane shows the the settings that have been fetched from the server. Settings can be changed interactively clicking in invidual values and changing them. Types are restricted to match the previous type. Changes will not affect the server until the Push Settings button is clicked.

To help find and edit settings the settings are shown in a nested view. Hide/Expand arrows can be used for sections. Specific settings can be searched for in the top right of the pane.

For the perception, all settings are listed on the JSON pane. For LidarHub new settings and sections can be added when operating in tree form.

** Should we discuss settings that the custom might typically want to change here? **

6.8 Lidar Hub

The Ouster Lidar Hub is designed to be an interface to the Ouster Perception software stack. It is deployed as a separate **Docker container** in conjunction with the **Ouster Perception solution** and extends many customer related features not inherent to the Ouster Perception software. These features include:

- On-device Aggregation of occupations and object lists
- The gathering and reporting of Diagnostics and Alerts to the Ouster Connect
- Down-sampling, Batching and Filtering of Perception JSON Streams used by:
  - TCP Relay Server(s)
  - MQTT Publishers
- On-device Data Recording of JSON Streams and Point Cloud data

Please refer to *Lidar Hub Overview* for more information.

6.9 Preferences

The Ouster Detect web GUI provides a number of preferences for the user to customize their viewing experience. These preferences are stored in the user's browser. The preferences are accessible from the main Viewer tab of the GUI. On the right side of the screen, below the main application tabs, there are several sub-tabs. Clicking on the Preferences sub-tab will open the preferences panel.

Note: This setting is local to each client and will not be reflected on other browsers connected to the backend.

- **Scene**: This section allows the user to supply a name for the scene, such as `Parking Lot 1`. This name will be displayed at the top of the GUI.
- **Theme:** This section allows the user to select from several pre-defined colour schemes for the GUI.

- **Range Rings**: This section allows the user to configure the properties of range rings in the viewer.
- **Grid**: This section allows the user to configure the properties of the square grid in the viewer.
- **Auto Rotate**: Ouster Detect provides a mode to enable an automatic rotation of the scene in the viewer. This section allows the user to configure the speed of rotation.
- **Developer Mode**: Enabling this mode exposes advanced functionality for developers.

The preferences panel is shown in the following image.

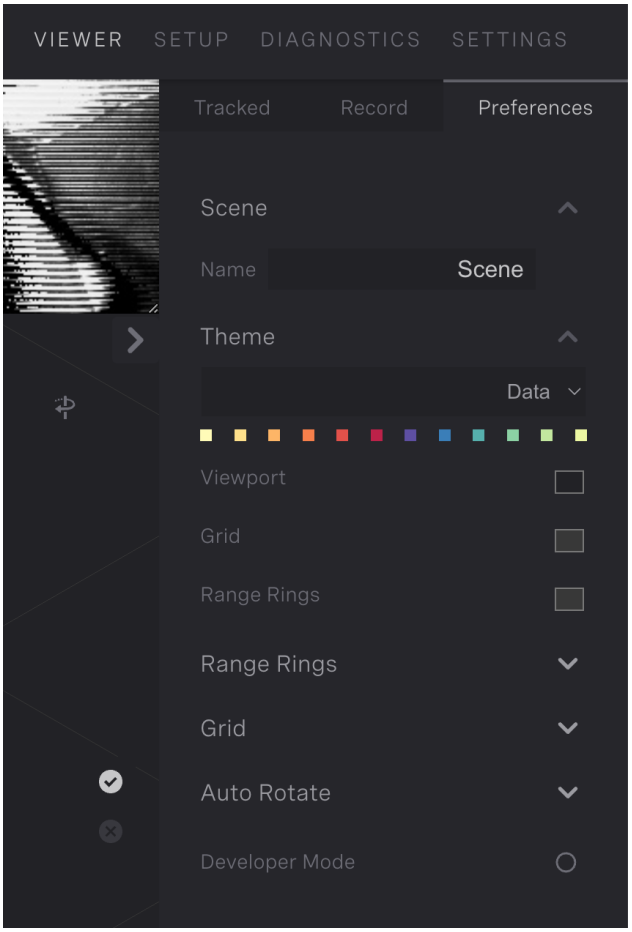


Figure 6.29: Ouster Detect Preferences Panel

7 Lidar Hub Overview

The Ouster Lidar Hub is designed to be an interface to the Ouster Perception software stack. It is deployed as a separate **Docker container** in conjunction with the **Ouster Perception solution** and extends many customer related features not inherent to the Ouster Perception software. These features include:

- On-device Aggregation of occupations and object lists
- The gathering and reporting of Diagnostics and Alerts to the Ouster Connect
- Down-sampling, Batching and Filtering of Perception JSON Streams used by: - TCP Relay Server(s) - MQTT Publishers
- On-device Data Recording of JSON Streams and Point Cloud data

7.1 Architecture

The **Ouster Lidar Hub** is driven by the Ouster Perception software data streams, including object lists and occupations via TCP sockets, configuration and diagnostics via RESTful API requests and point cloud data via web sockets. These streams are distributed to sub-processes via an internal memory-based pub/sub broker for optimal efficiency and minimal latency.

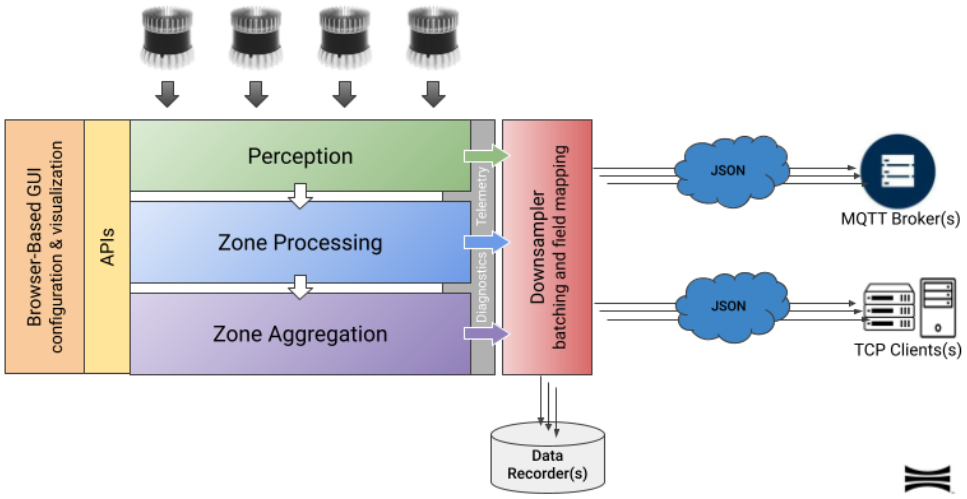


Figure 7.1: Lidar Hub Architecture

7.1.1 Application Configuration

The Ouster Lidar Hub is configured via the lidar_hub_config.json settings file found in /opt/ouster/-conf/settings/. When started for the first time, the Lidar Hub will load its default settings.

The "application" section of the Lidar Hub settings file is used for defining top-level application-specific settings.

Example

```
"application": {
    "id": "FF:FF:FF:FF:FF:FF",
    "name": "LidarHub",
    "version": "0.18.0",
    "logging": {
    },
    "ouster_connect": {
    },
    "world": {
    }
}
```

7.1.2 Primary application Fields

Table7.1: Primary Application Fields

Field	Format	Description
id	string	Unique ID of this installation (typically the unique hardware identifier generated by the Ouster Agent)
name	string	Application name (informational only)
version	string	Application version (should not be altered)
logging	object	Application Logging configuration (*System Logging*)
ouster_connect	object	Ouster Connect configuration (*Ouster Connect*)
world	object	World configuration (*World*)

7.1.3 Optional Application Fields

Table7.2: Optional Application Fields

Field	Format	Description
Debug	bool	Used for application diagnostics
log_health_level	int	Application health logging level (default 10 - DEBUG)
log_health_interval_secs	int	Application health logging interval (default 3600)

continues on next page

Table 7.2 - continued from previous page

Field	Format	Description
concurrency_port	int	System port number to use for ensuring a single instance of the application (default = 4999)
internal_port_range	list	Internal queues bind port range...5 ports in total must be defined (default = [4002, 4003, 4004, 4005, 4006])
web_server_port	int	Web server listening port number (default = 8003)
web_server_ssl	bool	Flag indicating if local web server should implement SSL (default = false)

7.1.4 Perception Streams Configuration

The "perception" section of the Lidar Hub settings file is used for defining Perception Software-specific settings. By default, the Lidar Hub will connect to the Perception software stack running in Docker containers on the local host.

Example

```
"perception": {
    "perception_websocket": {
      "host": "perception",
      "port": 3001
    },
    "event_zone_websocket": {
      "host": "event-zones",
      "port": 3004
    },
    "object_list": {
        "host": "perception",
        "port": 3002
    },
    "occupations": {
        "host": "event-zones",
        "port": 3003
    },
    "aggregation": {
    }
}
```

7.1.5 Primary perception Fields

Table7.3: Primary perception Fields

Field	Format	Description
perception_websocket.host	string	Host name of the Perception Server web socket server
perception_websocket.port	int	Port number of the Perception Server web socket server
event_zone_websocket.host	string	Host name of the Event Zone Server web socket server
event_zone_websocket.port	int	Port number of the Event Zone Server web socket server
object_list.host	string	Host name of the JSON object_list TCP server
object_list.port	int	Port number of the JSON object_list TCP server
occupations.host	string	Host name of the JSON occupations TCP server
occupations.port	int	Port number of the JSON occupations TCP server
aggregation	object	Aggregation configuration

7.1.6 Optional perception Fields

Table7.4: Primary perception Fields

Field	Format	Description
perception_server_api	string	URL of the Perception Server RESTful API endpoint (default = https://perception:8000/perception/api/v1/)
event_zone_server_api	string	URL of the Event Zone Server RESTful API endpoint (default = https://event-zones:8001/event/api/v1/)
agent_api	string	URL of the Ouster Agent RESTful API endpoint (default = http://172.17.0.1:4443/agent/api/v1/)
perception_websocket.tls	bool	Flag indicating if the Perception Server web socket server requires TLS (default = true)

continues on next page

Table 7.4 - continued from previous page

Field	Format	Description
perception_websocket.connect _retry_interval_secs	float	Delay in seconds before retrying a failed connection to the Perception Server web socket server (default = 60.0)
perception_websocket.receive_timeout _secs	float	Seconds to wait before resetting a connection due to no data being received from the Perception Server web socket server (default = 60.0)
event_zone_websocket.tls	bool	Flag indicating if the Event Zone Server web socket server requires TLS (default = true)
event_zone_websocket.connect _retry_interval_secs	float	Delay in seconds before retrying a failed connection to the Event Zone Server web socket server (default = 60.0)
event_zone_websocket.receive_time out_secs	float	Seconds to wait before resetting a connection due to no data being received from the Event Zone Server web socket server (default = 60.0)
object_list.tls	bool	Flag indicating if the JSON object_list TCP server requires TLS (default = true)
object_list.connect_retry_interval _secs	float	Delay in seconds before retrying a failed connection to the JSON object_list TCP server (default = 60.0)
object_list.receive_timeout_secs	float	Seconds to wait before resetting a connection due to no data being received from the JSON object_list TCP server (default = 60.0)
occupations.tls	bool	Flag indicating if the JSON occupations TCP server server requires TLS (default = true)
occupations.connect_retry_interval _secs	float	Delay in seconds before retrying a failed connection to the JSON occupations TCP server (default = 60.0)
occupations.receive_timeout _secs	float	Seconds to wait before resetting a connection due to no data being received from the JSON occupations TCP server (default = 60.0)

7.2 System Logging

The Lidar Hub supports System logging to both the console and to file. When running in a Docker container, console logging is captured by Docker. When installed on an Ouster Catalyst device, console logging is forwarded to the common Ouster log located at /opt/ouster/logs/ouster-docker.log.

7.2.1 Configuration

The application and application.logging sections of the Lidar Hub settings file are used for configuring application logging. By default, file logging is disabled, and console logging is set to INFO.

Example

```
"application": {
    ...
    "logging": {
        "console_log_level": 20
    }
}
```

7.2.2 Primary logging Fields

Table7.5: Optional logging Fields

Field	Format	Description
console_log_level	int	Application logging level (CRITICAL = 50, ERROR = 40, WARNING = 30, INFO = 20, DEBUG = 10)

7.2.3 Optional logging Fields

Table7.6: Optional logging Fields

Field	Format	Description
file_log_level	int	Optional application file logging level (CRITICAL = 50, ERROR = 40, WARNING = 30, INFO = 20, DEBUG = 10)
file_log_path	string	Optional application file logging path (default = /opt/ouster/logs/)

7.3 Ouster Connect

Ouster's cloud-based device management, monitoring and analytics platform is where the end user can gain visibility into the health and performance of their Lidar deployment(s).

7.3.1 Configuration

The application.ouster_connect section of the Lidar Hub settings file is used for configuring Ouster Connect. By default, System Diagnostics are published to Ouster Connect every 60 seconds. When Aggregation is enabled, real-time and timeseries data will also be published to Ouster Connect when generated by default.

Example

```
"application": {
    ...
    "ouster_connect": {
        "host": "connect.ouster.com",
        "state": "provisioned",
        "diagnostics_interval_secs": 60.0
    }
}
```

7.3.2 Primary ouster_connect Fields

Table7.7: Primary ouster_connect Fields

Field	Format	Description
host	string	Host name/IP of Ouster Connect
state	string	Provisioning state of Ouster Connect
diagnostics_interval_secs	float	System Diagnostics interval...set to 0 to disable

7.3.3 Optional ouster_connect Fields

Table7.8: Optional ouster_connect Fields

Field	Format	Description
port	int	Ouster Connect MQTT port number (default = 1883)
tls	bool	Flag indicating if the Ouster Connect requires TLS (default = false)
qos	int	Ouster Connect MQTT QOS (0 = At most once, 1 = At least once, 2 = Exactly once) (default = 1)

continues on next page

Table 7.8 - continued from previous page

Field	Format	Description
provisioning_device_key	string	Provisioning device key for "Ouster Edge Device" profile
provisioning_device_secret	string	Provisioning device secret for "Ouster Edge Device" profile
provisioning_lidar_key	string	Provisioning device key for "Ouster Lidar Sensor Device" profile
provisioning_lidar_secret	string	Provisioning device secret for "Ouster Lidar Sensor" profile
topic_attributes	string	Ouster Connect MQTT topic for publishing attributes
topic_telemetry	string	Ouster Connect MQTT topic for publishing telemetry
topic_alerts	string	Ouster Connect MQTT topic for publishing alerts
topic_provisioning_request	string	Ouster Connect MQTT topic for publishing provisioning requests
topic_provisioning_response	string	Ouster Connect MQTT topic for subscribing to provisioning responses
connecting_timeout_secs	float	Seconds to wait for a connection to be established to the Ouster Connect (default = 5.0)
connect_retry_interval_secs	float	Delay in seconds before retrying a failed connection to the Ouster Connect (default = 60.0)
aggregation_realtime_transmit_hertz	float	Frequency at which Aggregation real-time updates will be published to Ouster Connect (default = 0.0)
aggregation_timeseries_transmit_hertz	float	Frequency at which Aggregation timeseries data will be published to Ouster Connect (default = 0.0)

7.4 World

The Lidar Hub has the ability to optionally convert JSON Data Stream x/y cartesian coordinates to WGS84 longitude/latitude coordinates given a scene's world reference geo-coordinate and azimuth. By default, this option is disabled.

7.4.1 Configuration

The world section of the Lidar Hub settings file is used for configuring the scene's world reference geo-coordinate and azimuth.

Example

```
"application": {
    ...
    "world": {
      "latitude": 37.76466741329934,
      "longitude": -122.4136976526134,
      "azimuth": 57.6
    }
}
```

7.4.2 Primary World Fields

Table7.9: Primary aggregation Fields

Field	Format	Description
latitude	float	Scene's world reference latitude
longitude	float	Scene's world reference longitude
Azimuth	float	Scene's world reference azimuth

7.4.3 Optional World Fields

None

7.5 System Diagnostics

The Lidar Hub has a Systems Diagnostics module that collects and aggregates statistics, health, alerts, and telemetry from the compute device, all active Lidar sensors and all software related to the Perception Solution...including the Lidar Hub itself. The Diagnostics data is broken into three buckets:

- Attributes: Static data that is only reported when changed.
- Telemetry: Dynamic data that is reported every diagnostics_interval_secs
- Alerts: Lidar and software alerts reported with every detection and update

7.5.1 Output: Attributes

Compute device, lidar sensor and software Attributes will be published when any value changes to the internal memory-based broker for consumption by MQTT Publishers, TCP Relay Servers and Data Recorders.

Example


```
{
  "hardware_id": "*1234567890",
  "latitude": 40.7127837,
  "longitude": -74.0059413,
  "compute": {
    "mac_address": "FF:FF:FF:FF:FF:FF",
    "os_version": "",
    "last_upgrade": "",
    "last_upgrade_timestamp": 0,
    "total_cpu_cores": 16,
    "total_memory": 33354051584,
    "total_disk_space": 982141468672
  },
  "lidars": [
    {
      "model": "OS1",
      "serial_number": "992144000616",
      "hostname": "os-992144000616.local.",
      "beam_configuration": "128",
      "firmware": {
        "version": "v2.3.1"
      }
    }
  ],
  "perception_server": {
    "software_version": "0.1.0.284",
    "settings_version": "1.2.0",
    "is_running": true,
    "is_paused": false,
    "license": "none",
    "num_inclusion_zones": 0,
    "num_exclusion_zones": 1,
    "configuration": {
    },
    "timestamp": 1660673730063520
  },
  "event_zone_server": {
    "software_version": "0.1.0.284",
    "settings_version": "0.0.0",
    "is_running": true,
    "license": "none",
    "num_event_zones": 3,
    "configuration": {
    },
    "timestamp": 1660673730110978
  },
  "lidar_hub": {
```


(continues on next page)

(continued from previous page)


```
    "software_version": "0.18.4",
    "settings_version": "0.18.0",
    "license": "none",
    "is_running": true,
    "diagnostics_enabled": true,
    "aggregation_enabled": false,
    "mqtt_publishers": 0,
    "tcp_relay_servers": 2,
    "datarecorders": [],
    "configuration": {
    }
  },
  "timestamp": 1660698930000000,
  "diagnostics_version": 1
}
```

JSON Field Definitions

Table7.10: JSON Field Definitions

Field	Format	Description
hardware_id	string	Hardware ID of the compute device (MAC Address or Agent Locking Code)
latitude	float	Approximate latitude of the compute device as reported by http://ipwho.is/
longitude	float	Approximate longitude of the compute device as reported by http://ipwho.is/
compute	object	Compute Device details
compute.mac_address	string	MAC Address of the compute device
compute.os_version	string	Operating system of the compute device
compute.last_upgrade	string	Description of last OS upgrade performed on the compute device
compute.last_upgrade_timestamp	int	Timestamp of last OS upgrade performed on the compute device (microseconds since Jan. 1, 1970)
compute.total_cpu_cores	int	Number of CPU cores on the compute device
compute.total_memory	int	Total memory installed in compute device (bytes)
compute.total_disk_space	int	Total storage installed in compute device (bytes)
lidars	array	Array of active Lidar sensors
lidars[].model	string	Lidar sensor model
lidars[].serial_number	string	Lidar senor serial number
lidars[].hostname	string	Lidar senor host name
lidars[].beam_configuration	string	Lidar senor beam configuration as reported by the firmware
lidars[].firmware	object	Lidar Firmware details

continues on next page

Table 7.10 - continued from previous page

Field	Format	Description
lidars[].firmware.version	string	Lidar sensor firmware version as reported by the firmware
perception_server	object	Perception Server details
perception_server.software_version	string	Perception Server software version
perception_server.settings_version	string	Perception Server settings version
perception_server.is_running	bool	Indication if the Perception Server is running
perception_server.is_paused	bool	Indication if the Perception Server is paused on a replay
perception_server.license	string	Perception Server license details
perception_server.num_inclusion_zones	int	Number of Perception Server inclusion zones
perception_server.num_exclusion_zones	int	Number of Perception Server exclusion zones
perception_server.configuration	object	Perception Server configuration
perception_server.timestamp	int	Timestamp of these attributes (microseconds since Jan. 1, 1970)
event_zone_server	object	Event Zone Server details
event_zone_server.software_version	string	Event Zone Server software version
event_zone_server.settings_version	string	Event Zone Server settings version
event_zone_server.is_running	bool	Indication if the Event Zone Server is running
event_zone_server.license	string	Event Zone Server license details
event_zone_server.num_event_zones	int	Number of Event Zone Server event zones
event_zone_server.configuration	object	Event Zone Server configuration
event_zone_server.timestamp	int	Timestamp of these attributes (microseconds since Jan. 1, 1970)
lidar_hub	object	Lidar Hub details
lidar_hub.software_version	string	Lidar Hub software version

continues on next page

Table 7.10 - continued from previous page

Field	Format	Description
lidar_hub.settings_version	string	Lidar Hub settings version
lidar_hub.license	string	Lidar Hub license details
lidar_hub.is_running	bool	Indication if the Lidar Hub is running
lidar_hub.diagnostics_enabled	bool	Indication if Diagnostics are enabled
lidar_hub.aggregation_enabled	bool	Indication if Aggregation is enabled
lidar_hub.mqtt_publishers	int	Number of configured MQTT publishers
lidar_hub.tcp_relay_servers	int	Number of configured TCP Relay Servers
lidar_hub.datarecorders	list	List of configured Data Recorders
lidar_hub.configuration	object	Lidar Hub configuration
timestamp	int	Timestamp of this message (microseconds since Jan. 1, 1970)
diagnostics_version	int	Version of this message

7.5.2 Output: Telemetry

Compute device, lidar sensor and software Telemetry will be published at diagnostics_interval_secs to the internal memory-based broker for consumption by MQTT Publishers, TCP Relay Servers and Data Recorders.

Example


```
{
  "hardware_id": "*1234567890",
  "compute": {
    "cpu_utilization": 19.7,
    "cpu_core_utilization": [
      17.2,
      19,
      21.3,
      18.3,
      22.1,
      18.9,
      18.2,
      19.7,
      18.6,
      17.5,
      16.9,
```


(continues on next page)

(continued from previous page)


```
      23,
      25.2,
      16.7,
      20.1,
      21.7
    ],
    "available_memory": 18890395648,
    "available_disk_space": 762672287744
  },
  "lidars": [
    {
      "model": "OS1",
      "serial_number": "992144000616",
      "firmware": {
        "input_current_ma": 704,
        "input_voltage_mv": 24044,
        "internal_temperature_deg_c": 0,
        "phase_lock_status": "DISABLED",
        "timestamp_ns": 1727503038782740
      },
      "num_background_points": 83010,
      "num_foreground_points": 416,
      "num_failed_points": 865,
      "num_no_return_points": 46781,
      "num_zone_filtered_points": 4,
      "num_ground_points": 0,
      "num_clusters": 6,
      "num_filtered_clusters": 0
    }
  ],
  "perception_server": {
    "up_time": 0,
    "num_objects": 0,
    "num_websocket_clients": 3,
    "num_tcp_clients": 3,
    "timestamp": 1660673730063520
  },
  "event_zone_server": {
    "up_time": 0,
    "num_occupations": 0,
    "num_websocket_clients": 2,
    "num_tcp_clients": 2,
    "timestamp": 1660673730110978
  },
  "lidar_hub": {
    "mqtt_publishers_connected": 0,
    "tcp_relay_server_connections": 0
  },
  "timestamp": 1660698930000000,
  "diagnostics_version": 1
}
```

JSON Field Definitions

Table7.11: JSON Field Definitions

Field	Format	Description
hardware_id	string	MAC Address of the compute device (MAC Address or Agent Locking Code)
compute	object	Compute Device details
compute.cpu_utilization	float	Total CPU utilization across all cores
compute.cpu_core_utilization	list	CPU utilization by cores
compute.available_memory	int	Available memory on the compute device (bytes)
compute.available_disk_space	int	Available storage on the compute device (bytes)
lidars	array	Array of active Lidar sensors
lidars[].model	string	Lidar sensor model
lidars[].serial_number	string	Lidar senor serial number
lidars[].firmware	object	Lidar Firmware details
lidars[].firmware.input_current_ma	int	Lidar sensor input current as reported by the firmware (milliamps)
lidars[].firmware.input_voltage_mv	int	Lidar sensor input voltage as reported by the firmware (millivolts)
lidars[].firmware.internal_temperature_deg_c	int	Lidar sensor internal temperature as reported by the firmware (degrees celsius)
lidars[].firmware.phase_lock_status	string	Lidar sensor phase lock status as reported by the firmware
lidars[].firmware.timestamp_ns	int	Timestamp of these values as reported by the firmware (nanoseconds since Jan. 1, 1970)
lidars[].num_background_points	int	Number of background points for the last frame
lidars[].num_foreground_points	int	Number of foreground points for the last frame

continues on next page

Table 7.11 - continued from previous page

Field	Format	Description
lidars[].num_failed_points	int	Number of failed points for the last frame
lidars[].num_no_return_points	int	Number of points not returned for the last frame
lidars[].num_zone_filtered_points	int	Number of points filted by inclusion and exclusion zones for the last frame
lidars[].num_ground_points	int	Number of points filtered as ground points for the last frame
lidars[].num_clusters	int	Number of clusters detected for the last frame
lidars[].num_filtered_clusters	int	Number of clusters remaining after filtering and merging for the last frame
perception_server	object	Perception Server details
perception_server.up_time	int	Perception Server run time since last started
perception_server.num_objects	int	Number of objects currently being tracked by the Perception Server
perception_server.num_websocket_clients	int	Number of web socket clients connected to the Perception Server
perception_server.num_tcp_clients	int	Number of TCP clients connected to the Perception Server
perception_server.timestamp	int	Timestamp of these values (microseconds since Jan. 1, 1970)
event_zone_server	object	Event Zone Server details
event_zone_server.up_time	int	Perception Server run time since last started
event_zone_server.num_occupations	int	Number of objects currently being positioned inside event zone(s)
event_zone_server.num_websocket_clients	int	Number of web socket clients connected to the Event Zone Server
event_zone_server.num_tcp_clients	int	Number of TCP clients connected to the Event Zone Server

continues on next page

Table 7.11 - continued from previous page

Field	Format	Description
event_zone_server.timestamp	int	Timestamp of these values (microseconds since Jan. 1, 1970)
lidar_hub	object	Lidar Hub details
lidar_hub.mqtt_publishers_connected	int	Number of configured MQTT publishers connected to their endpoint
lidar_hub.tcp_relay_server_connections	int	Number of TCP clients connected to Lidar Hub TCP Relay Servers
timestamp	int	Timestamp of this message (microseconds since Jan. 1, 1970)
diagnostics_version	int	Version of this message

7.5.3 Output: Alerts

Lidar sensor and software Alerts will be published when first detected and when any update is reported to the internal memory-based broker for consumption by MQTT Publishers, TCP Relay Servers and Data Recorders.

Example


```
{
  "alerts": [
    {
      "active": true,
      "category": "SHOT_LIMITING",
      "cursor": 2833,
      "alert_code": "0x0100000f",
      "level": "WARNING",
      "source_info": "OS1-992144000616",
      "id": 13962805723535194000,
      "msg": "Shot limiting mode is active. Laser power is partially attenuated;
             please see user guide for heat sinking requirements.",
      "msg_verbose": "Shot limiting has started.",
      "first_occurred": 0,
      "last_occurred": 0,
      "active_count": 0
    },
    {
      "active": true,
      "category": "UDP_TRANSMISSION",
      "cursor": 2822,
      "alert_code": "0x01000016",
      "level": "WARNING",
      "source_info": "OS1-992144000616",
      "id": 11061546793406626000,
```


(continues on next page)

(continued from previous page)


```
      "msg": "Could not send lidar data UDP packet to host; check that network is up.",
      "msg_verbose": "Failed to send lidar UDP data to destination host 10.0.0.39:34642",
      "first_occurred": 0,
      "last_occurred": 0,
      "active_count": 0
    }
  ]
}
```


JSON Field Definitions

Table7.12: JSON Field Definitions

Field	Format	Description
alerts	array	Array of active alerts
alerts[].active	bool	Indication if the alert is still active
alerts[].category	string	Alert category (application specific)
alerts[].cursor	int	Alert cursor (lidar only)
alerts[].alert_code	string	Alert code (application specific - hex)
alerts[].level	string	Alert level (application specific)
alerts[].source_info	string	Alert source information (application specific)
alerts[].id	int	Unique alert ID (application specific)
alerts[].msg	int	Alert message
alerts[].msg_verbose	int	Additional alert details (if any)
alerts[].first_occurred	int	First occurrence of the alert code (micro/-nano seconds since Jan. 1, 1970)
alerts[].last_occurred	int	Last occurrence of the alert code (micro/-nano seconds since Jan. 1, 1970)
alerts[].active_count	int	Total occurrences of the alert code

7.6 Aggregation

The Lidar Hub has an on-device Aggregation module that aggregates zone occupations by timeseries by object classification. A system-defined classification of "ALL" will also be aggregated for each zone representing an aggregate of all objects. The Aggregation module also aggregates all tracked objects by timeseries by object classification into a system-defined site-wide zone with the id of 0.

In addition to timeseries data, the Aggregation module will generate real-time events whenever the occupancy of a zone changes at the transmit_realtime_hertz interval specified.

7.6.1 Configuration

The perception.aggregation section of the Lidar Hub settings file is used for configuring the Aggregation module. By default, Aggregation is enabled with 60-second timeseries intervals, and will generate real-time events at 1Hz. Setting a value to zero will disable the feature.

Example

```
"perception": {
    ...
    "aggregation": {
        "timeseries_secs": 60.0,
        "realtime_hertz": 1.0,
        "storage_path": "/opt/ouster/conf/data/storage"
    }
}
```

7.6.2 Primary aggregation Fields

Table7.13: Primary aggregation Fields

Field	Format	Description
timeseries_secs	float	Aggregation interval in seconds...a timeseries record is emitted every interval (default = 60.0)
realtime_hertz	float	Frequency at which Aggregation will emit real-time updates (default = 1.0)
storage_path	str	Path for the non-volatile storage of aggregation metrics...clear to disable (default = /opt/ouster/conf/-data/storage)

7.6.3 Optional aggregation Fields

Table7.14: Optional aggregation Fields

Field	Format	Description
source_hertz	float	Frequency at which Aggregation will process Perception data (default = 2.0)
departure_debounce_ms	int	Timeout before departing track from a given zone in milliseconds (default = 2500)
excluded_classifications	list	List of classifications to be excluded from Aggregation (default = [“PROSPECT”, “UNKNOWN”])

7.6.4 Output: Real-Time Events

Zone real-time occupancy events will be generated in real-time and published at transmit_realtime_hertz to the internal memory-based broker for consumption by MQTT Publishers, TCP Relay Servers and Data Recorders.

Example


```
{
  "aggregation_realtime": [
    {
      "id": 1659467677889,
      "name": "Office",
      "timestamp": 1660664443771926,
      "classification_metrics": [
        {
          "description": "ALL",
          "active_occupants": 1,
          "total_visits": 1,
          "last_update": 1660664443771926
        },
        {
          "description": "PERSON",
          "active_occupants": 1,
          "total_visits": 1,
          "last_update": 1660664443771926
        }
      ]
    },
    {
      "id": 1658947733821,
      "name": "Desk",
      "timestamp": 1660664443771926,
      "classification_metrics": [
        {
          "description": "ALL",
          "active_occupants": 1,
          "total_visits": 1,
          "last_update": 1660664443771926
        },
        {
          "description": "PERSON",
          "active_occupants": 1,
          "total_visits": 1,
          "last_update": 1660664443771926
        }
      ]
    }
  ]
}
```

JSON Field Definitions

Table7.15: JSON Field Definitions

Field	Format	Description
id	int	Zone ID
name	string	Zone name
timestamp	int	Timestamp of the real-time event (microseconds since Jan. 1, 1970)
classification_metrics	array	Array of object classifications occupying the zone
classification_metrics.description	string	Object classification name ("ALL" is an aggregate of all classifications)
classification_metrics.active_occupants	int	Number of objects currently occupying the zone
classification_metrics.total_visits	int	Total visits to the zone since the application last started
classification_metrics.last_update	int	Last zone update (microseconds since Jan. 1, 1970)

7.6.5 Output: Timeseries Aggregates

Zone timeseries aggregates will be generated every timeseries_secs and published at transmit_timeseries_hertz to the internal memory-based broker for consumption by MQTT Publishers, TCP Relay Servers and Data Recorders.

Example


```
{
  "aggregation_timeseries": [
      {
      "id": 1659467677889,
      "name": "Office",
      "timestamp": 1660664440000000,
      "classification_metrics": [
        {
          "description": "ALL",
          "active_occupants": 1,
          "max_active_occupants": 1,
          "avg_active_occupants": 1,
          "total_visitors": 5,
          "total_visits": 5,
          "new_visitors": 1,
          "new_visits": 1,
          "departed_visitors": 0,
          "current_max_speed": 0.227,
          "current_avg_speed": 0.122,
          "overall_max_speed": 0.22669327197460112,
```


(continues on next page)

(continued from previous page)


```
        "overall_avg_speed": 0.122,
        "current_avg_dwell": 0,
        "overall_avg_dwell": 0,
        "last_update": 1660664449970829
      },
      {
        "description": "PERSON",
        "active_occupants": 1,
        "max_active_occupants": 1,
        "avg_active_occupants": 1,
        "total_visitors": 5,
        "total_visits": 5,
        "new_visitors": 1,
        "new_visits": 1,
        "departed_visitors": 0,
        "current_max_speed": 0.227,
        "current_avg_speed": 0.122,
        "overall_max_speed": 0.22669327197460112,
        "overall_avg_speed": 0.122,
        "current_avg_dwell": 0,
        "overall_avg_dwell": 0,
        "last_update": 1660664449970829
      }
    ]
  },
  {
    "id": 1658947733821,
    "name": "Desk",
    "timestamp": 1660664440000000,
    "classification_metrics": [
      {
        "description": "ALL",
        "active_occupants": 1,
        "max_active_occupants": 1,
        "avg_active_occupants": 1,
        "total_visitors": 5,
        "total_visits": 5,
        "new_visitors": 1,
        "new_visits": 1,
        "departed_visitors": 0,
        "current_max_speed": 0.227,
        "current_avg_speed": 0.122,
        "overall_max_speed": 0.22669327197460112,
        "overall_avg_speed": 0.122,
        "current_avg_dwell": 0,
        "overall_avg_dwell": 0,
        "last_update": 1660664449970829
      },
      {
        "description": "PERSON",
        "active_occupants": 1,
        "max_active_occupants": 1,
        "avg_active_occupants": 1,
        "total_visitors": 5,
```


(continues on next page)

(continued from previous page)


```
        "total_visits": 5,
        "new_visitors": 1,
        "new_visits": 1,
        "departed_visitors": 0,
        "current_max_speed": 0.227,
        "current_avg_speed": 0.122,
        "overall_max_speed": 0.22669327197460112,
        "overall_avg_speed": 0.122,
        "current_avg_dwell": 0,
        "overall_avg_dwell": 0,
        "last_update": 1660664449970829
      }
    ]
  },
  {
    "id": 0,
    "name": "Edge-FF:FF:FF:FF:FF:FF",
    "timestamp": 1660664440000000,
    "classification_metrics": [
      {
        "description": "ALL",
        "active_occupants": 1,
        "max_active_occupants": 1,
        "avg_active_occupants": 1,
        "total_visitors": 5,
        "total_visits": 5,
        "new_visitors": 1,
        "new_visits": 1,
        "departed_visitors": 0,
        "current_max_speed": 0.227,
        "current_avg_speed": 0.124,
        "overall_max_speed": 0.22669327197460112,
        "overall_avg_speed": 0.124,
        "current_avg_dwell": 0,
        "overall_avg_dwell": 0,
        "last_update": 1660664449970829
      },
      {
        "description": "PERSON",
        "active_occupants": 1,
        "max_active_occupants": 1,
        "avg_active_occupants": 1,
        "total_visitors": 1,
        "total_visits": 1,
        "new_visitors": 1,
        "new_visits": 1,
        "departed_visitors": 0,
        "current_max_speed": 0.227,
        "current_avg_speed": 0.124,
        "overall_max_speed": 0.22669327197460112,
        "overall_avg_speed": 0.124,
        "current_avg_dwell": 0,
        "overall_avg_dwell": 0,
        "last_update": 1660664449970829
```


(continues on next page)

(continued from previous page)

```
        }
      ]
    }
  ]
}
```

JSON Field Definitions

Table7.16: JSON Field Definitions

Field	Format	Description
id	int	Zone ID
name	string	Zone name
timestamp	int	Timestamp of the timeseries aggregate (microseconds since Jan. 1, 1970)
classification_metrics	array	Array of object classifications occupying the zone
classification_metrics.description	string	Object classification name ("ALL" is an aggregate of all classifications)
classification_metrics.active_occupants	int	Number of objects occupying the zone at this moment
classification_metrics.max_active_occupants	int	Maximum number of objects occupying the zone in this timeseries
classification_metrics.avg_active_occupants	int	Average number of objects occupying the zone in this timeseries
classification_metrics.total_visitors	int	Total unique visitors to the zone
classification_metrics.total_visits	int	Total visits to the zone
classification_metrics.new_visitors	int	New unique visitors to the zone in this timeseries
classification_metrics.new_visits	int	New visits to the zone in this timeseries
classification_metrics.departed_visitors	int	Visitors that left the zone in this timeseries
classification_metrics.current_max_speed	float	Maximum speed observed in the zone in this timeseries
classification_metrics.current_avg_speed	float	Average speed observed in the zone in this timeseries
classification_metrics.overall_max_speed	float	Maximum speed observed in the zone since the application last started

continues on next page

Table 7.16 - continued from previous page

Field	Format	Description
classification_metrics.overall_avg_speed	float	Average speed observed in the zone since the application last started
classification_metrics.current_avg_dwell	float	Average dwell observed in the zone in this timeseries
classification_metrics.overall_avg_dwell	float	Average dwell observed in the zone since the application last started
classification_metrics.last_update	int	Last zone update (microseconds since Jan. 1, 1970)

7.7 JSON Data Streams w/Down-sampling, Batching & Field Mapping

One or more MQTT Publishers and/or TCP Relay Servers may be configured for supported JSON data streams (object_list | occupations | aggregation_realtime | aggregation_timeseries | diagnostics). MQTT Publishers and TCP Relay Servers enable the down-sampling and batching of JSON output, as well as custom field mapping, filtering and decimal precision of floating-point values. When running in a Docker container, all TCP Relay Server port(s) must be exposed in the compose.yml.

Since batching is inherent to this feature, transmissions are wrapped in a JSON array with the data stream name as the field name. This wrapping is enabled by default, but can be overridden by setting return_as_array = false to the specific MQTT Publisher/TCP Relay Server configuration. Please note that a malformed JSON message will be received if return_as_array is disabled and batching returns more than one frame.

Example Batching

```
{"object_list": [{...}, {...}, {...}]}
```

object_list, occupations and diagnostics JSON fields can be filtered and (optionally) renamed by setting field_mappings to a dictionary of key/value pairs, where the key is the actual field name and the value is the new name (leaving blank will result in no renaming). Once field_mappings are applied, all desired fields must be included in the field_mappings dictionary. If the field is an object, list or dictionary, the entire object, list or dictionary will be returned...unless one of its fields are added to the field_mappings.

Example Field_mapping

```
{"timestamp": "ts", "objects": "objs", "classification": "typ", "position": "pos",
 "velocity": "velo", "dimensions": "dim", "x": "x", "y": "y","length": "l", "width": "w", "height": "h"}
```

Once field_mappings are in use, decimal precision for floating-point values can be defined by setting decimal_precision. decimal_precision will only apply to fields defined in field_mappings and does NOT automatically propagate to nested fields in objects, lists or dictionaries. A value of -1 disables decimal_precision.

7.8 MQTT Publisher Configuration

The mqtt_publishers section of the Lidar Hub settings file is used for configuring one or more MQTT Publisher(s). By default, no active MQTT Publishers are pre-configured, but an invalid example is included for reference.

Multiple publishing threads can be configured for a given MQTT Publisher. If a publisher falls behind, additional publishing threads will be launched until max_publishers is reached. A high_water_mark controls how far behind publishing can fall before messages are dropped. Once publishing is caught back up, the additional publishing threads will gradually be terminated in attempts to minimize resources while maintaining pace with the velocity of data.

${device_name} is a special variable that can be used in-line when defining the user_name and/or topic of an MQTT Publisher. The variable is replaced with its equivalent value at runtime.

Example

```
"mqtt_publishers": [
    {
        "source": "object_list",
        "host": "mqttbroker.mycompany.com",
        "port": 1883,
        "user_name": "",
        "password": "",
        "topic": "/v1/${device_name}/object_list",
        "qos": 1,
        "data_hertz": 1,
        "transmit_hertz": 1
    },
    {
      ...
    }
]
```

7.8.1 Primary mqtt_publishers Fields

Table7.17: Primary mqtt_publishers Fields

Field	Format	Description
mqtt_publishers	array	Array of MQTT Publishers
mqtt_publishers[].source	string	JSON data stream source (object_list/occupations/aggregation_realtime /aggregation_timeseries/diagnostics)
mqtt_publishers[].host	string	MQTT Broker host name/IP address
mqtt_publishers[].port	int	MQTT Broker port number (default = 1883)
mqtt_publishers[].user_name	string	MQTT Broker username to authenticate with [MQTT-3.1.3-11] (default = “”)

continues on next page

Table 7.17 - continued from previous page

Field	Format	Description
mqtt_publishers[].password	string	MQTT Broker password to authenticate with. Optional, set to "" if not required. (default = "")
mqtt_publishers[].topic	string	MQTT Broker topic to publish to
mqtt_publishers[].qos	int	Quality of Service (0 = At most once, 1 = At least once (default), 2 = Exactly once)
mqtt_publishers[].data_hertz	float	Down-sampling Hz (a value of 0 disables down-sampling)
mqtt_publishers[].transmit_hertz	float	Transmission Hz (a value of 0 sends immediately while a value greater than data_hertz invokes batching)

7.8.2 Optional mqtt_publishers Fields

Table7.18: Optional mqtt_publishers Fields

Field	Format	Description
mqtt_publishers[].connecting_timeout_secs	float	Seconds to wait for a connection to be established to the MQTT Broker (default = 5.0)
mqtt_publishers[].connect_retry_interval_secs	float	Delay in seconds before retrying a failed connection to the MQTT Broker (default = 60.0)
mqtt_publishers[].tls	bool	Indicator if MQTT Broker requires TLS (default = false (CURRENTLY NOT TESTED))
mqtt_publishers[].return_as_array	bool	Controls the wrapping of transmissions into a well-formed JSON array (default = true)
mqtt_publishers[].field_mappings	dict	JSON field filtering and renaming (default = {})
mqtt_publishers[].decimal_precision	int	Decimal precision for floating-point values (default = -1 (disabled))
mqtt_publishers[].convert_to_geo_coordinates	bool	Converts cartesian coordinates to WGS84 longitude/latitude coordinates
mqtt_publishers[].min_publishers	int	Minimum number of publishing threads (default = 1)

continues on next page

Table 7.18 - continued from previous page

Field	Format	Description
mqtt_publishers[].max_publishers	int	Maximum number of publishing threads (default = 3)
mqtt_publishers[].max_retries	int	Maximum retry attempts to publish a given message (default = 2)
mqtt_publishers[].high_water_mark	int	Maximum number of messages pending to be published (default = 100)
mqtt_publishers[].reenqueue_failures	bool	Add messages back to data_queue if max_retries is exceeded. If the high_water_mark is exceeded, the message will be dropped. (default = true)
mqtt_publishers[].heartbeat_interval_secs	float	Interval to send heartbeat_message when source stream is inactive (default = 0 (disabled))
mqtt_publishers[].heartbeat_message	string	Message to transmit when source stream is inactive (default = "{}")

7.9 TCP Relay Server Configuration

The tcp_servers section of the Lidar Hub settings file is used for configuring one or more TCP Relay Server(s). By default, TCP Relay Servers are pre-configured for object_list and occupations JSON data streams at 1Hz.

Example

```
"tcp_servers": [
    {
      "source": "object_list",
      "port": 3302,
      "data_hertz": 1,
      "transmit_hertz": 1
    },
    {
      "source": "occupations",
      "port": 3303,
      "data_hertz": 1,
      "transmit_hertz": 1
    },
    {
      ...
    }
]
```

7.9.1 Primary tcp_servers Fields

Table7.19: Primary tcp_servers Fields

Field	Format	Description
tcp_servers	array	Array of TCP Relay Servers
tcp_servers[].source	string	JSON data stream source object_list/occupations/aggregation_realtime/aggregation_timeseries/diagnostics
tcp_servers[].port	int	TCP Relay Server listening port
tcp_servers[].data_hertz	float	Down-sampling Hz (a value of 0 disables down-sampling)
tcp_servers[].transmit_hertz	float	Transmission Hz (a value of 0 sends immediately while a value greater than data_hertz invokes batching)

7.9.2 Optional tcp_servers Fields

Table7.20: Optional tcp_servers Fields

Field	Format	Description
tcp_servers[].tls	bool	Enable TLS (default = true)
tcp_servers[].return_as_array	bool	Controls the wrapping of transmissions into a well-formed JSON array (default = true)
tcp_servers[].field_mappings	dict	JSON field filtering and renaming (default = {})
tcp_servers[].decimal_precision	int	Decimal precision for floating-point values (default = -1 (disabled))
tcp_servers[].convert_to_geo_coordinates	bool	Converts cartesian coordinates to WGS84 longitude/latitude coordinates
tcp_servers[].heartbeat_interval_secs	float	Interval to send heartbeat_message when source stream is inactive (default = 0 (disabled))
tcp_servers[].heartbeat_message	string	Message to transmit when source stream is inactive (default = “{}”)

7.10 Event Data Recorder

The Lidar Hub has an on-device Event Data Recorder with a rolling buffer of perception output for recording activity before an event occurs. Completed recordings are uploaded to Ouster Connect (if configured) for playback. The module also supports a retention period, compression and purge strategies for handling recordings when the device is offline/not configured to upload to Ouster Connect. By default, recordings are saved to /opt/ouster/conf/data/events/ before they are uploaded to Ouster Connect.

7.10.1 Configuration

The event_recorder section of the Lidar Hub settings file is used for configuring the Event Data Recorder module. By default, Data Recording is disabled for all features.

When both json_data.object_list_enabled and binary_data.trackedobjects_enabled are set, binary_data.trackedobjects_enabled will take precedence.

When both json_data.occupations_enabled and binary_data.zone_events_enabled are set, binary_data.zone_events_enabled will take precedence.

When used in combination with the Ouster Connect Player, the Event Recorder configuration must be set at least as follows:

- binary_data.pointclouds_enabled: true
- binary_data.clusters_enabled: true
- binary_data.trackedobjects_enabled: true
- binary_data.images_enabled: true
- binary_data.zone_events_enabled: true
- retention_count: >= 1
- use_compression: true [either true(default) or false]

Example

```
"event_recorder": {
    "json_data": {
      "object_list_enabled": false,
      "occupations_enabled": false,
      "aggregation_timeseries_enabled": false,
      "aggregation_realtime_enabled": false,
      "diagnostics_enabled": false
    },
    "binary_data": {
      "pointclouds_enabled": true,
      "clusters_enabled": true,
      "trackedobjects_enabled": true,
      "images_enabled": true,
      "zone_events_enabled": true
```

(continues on next page)

(continued from previous page)

```
    },
    "max_rolling_buffer_secs": 60,
    "retention_count": 5,
    "min_available_disk": 0.35,
    "purge_strategy": "oldest",
    "use_compression": false,
    "log_path": "/opt/ouster/conf/data/events"
}
```

7.10.2 Primary data_recorder Fields

Table7.21: Primary data_recorder Fields

Field	Format	Description
json_data.object_list_enabled	bool	Flag indicating if JSON object lists should be captured in recordings (default = False)
json_data.occupations_enabled	bool	Flag indicating if JSON occupations should be captured in recordings (default = False)
json_data.aggregation_enabled	bool	Flag indicating if JSON aggregation timeseries output should be captured in recordings (default = False)
json_data.diagnostics_enabled	bool	Flag indicating if JSON diagnostics (attributes and telemetry combined) and alerts output should be captured in recordings (default = False)
binary_data.pointclouds_enabled	bool	Flag indicating if FlatBuf point clouds should be recaptured in recordings (default = False)
binary_data.clusters_enabled	bool	Flag indicating if FlatBuf clusters should be captured in recordings (default = False)
binary_data.trackedobjects_enabled	bool	Flag indicating if FlatBuf tracked objects should be captured in recordings (default = False)
binary_data.images_enabled	bool	Flag indicating if NearIR/CalRef images should be captured in recordings (default = False)
binary_data.zone_events_enabled	bool	Flag indicating if JSON zone events should be captured in recordings (default = False)
max_rolling_buffer_secs	float	Length of perception output rolling buffer in seconds (default = 60.0)
retention_count	int	Number of recordings to retain (default = unlimited)

continues on next page

Table 7.21 - continued from previous page

Field	Format	Description
min_available_disk	float	Minimal percentage of available storage before implementing the purge_strategy (default = 0.25 (25%))
purge_strategy	string	Recording purge strategy to invoke when minimum available storage is reached oldest/newest/disable (default = oldest)
use_compression	bool	Flag indicating if completed recordings should be compressed w/GZip (default = false)
log_path	string	Recordings path (default = /opt/ouster/conf/-data/recordings/)

7.10.3 Optional data_recorder Fields

none

7.10.4 Accessing Event Data

Event recordings can be accessed/downloaded by the user at the following URL:

```
https://<<host>>/data/events/
```

7.11 JSON Data Recorder

The Lidar Hub has an on-device JSON Data Recorder module with timed-rotation, retention period, compression and purge strategies. Object Lists, Occupations, Aggregation and Diagnostics are all recorded into a combined log file. By default, recordings are saved to /opt/ouster/conf/data/recordings/.

7.11.1 Configuration

The json_data_recorder section of the Lidar Hub settings file is used for configuring the JSON Data Recorder module. By default, Data Recording is disabled for all features. Data Recording can also be disabled by setting rotation_minutes = 0.

Example

```
"json_data_recorder": {
    "object_list_enabled": false,
    "occupations_enabled": false,
    "aggregation_enabled": false,
    "diagnostics_enabled": false,
    "rotation_minutes": 5,
```

(continues on next page)

(continued from previous page)

```
    "retention_count": 288,
    "min_available_disk": 0.35,
    "purge_strategy": "oldest",
    "use_compression": true,
    "log_path": "/opt/ouster/conf/data/recordings"
}
```

7.11.2 Primary data_recorder Fields

Table7.22: Primary data_recorder Fields

Field	Format	Description
object_list_enabled	bool	Flag indicating if JSON object lists should be recorded to disk (default = False)
occupations_enabled	bool	Flag indicating if JSON occupations should be recorded to disk (default = False)
aggregation_enabled	bool	Flag indicating if JSON aggregation timeseries output should be recorded to disk (default = False)
diagnostics_enabled	bool	Flag indicating if JSON diagnostics (attributes and telemetry combined) and alerts output should be recorded to disk (default = False)
rotation_minutes	float	Data recording file rotation interval in fractional minutes...set to 0 to disable data recording (default = 5.0)
retention_count	int	Number of recordings to retain (default = 288 (1 day))
min_available_disk	float	Minimal percentage of available storage before implementing the purge_strategy (default = 0.35 (35%))
purge_strategy	string	Recording purge strategy to invoke when minimum available storage is reached oldest/newest/disable (default = oldest)
use_compression	bool	Flag indicating if recordings should be compressed w/GZip after rotation (default = true)
log_path	string	Recordings path (default = /opt/ouster/conf/-data/recordings/)

7.11.3 Optional data_recorder Fields

None

7.11.4 Accessing JSON Data

JSON data recordings can be accessed/downloaded by the user at the following URL:

```
https://<<host>>/data/recordings/
```

7.12 Binary Data Recorder

The Lidar Hub has an on-device Binary Data Recorder module with timed-rotation, retention period, compression and purge strategies. Point Clouds, Clusters and Tracked Objects are recorded as flatbuf messages, while Event Zone Server Zone Events are stored as JSON messages. All messages are merged into a combined file in binary format, each prepended with a 4-byte unsigned length (big-endian). By default, recordings are saved to /opt/ouster/conf/data/recordings/.

7.12.1 Configuration

The binary_data_recorder section of the Lidar Hub settings file is used for configuring the Binary Data Recorder module. By default, Data Recording is disabled for all features. Data Recording can also be disabled by setting rotation_minutes = 0.

Example

```
"data_recorder": {
    "pointclouds_enabled": false,
    "clusters_enabled": false,
    "trackedobjects_enabled": false,
    "images_enabled": false,
    "zone_events_enabled": false,
    "rotation_minutes": 5,
    "retention_count": 288,
    "min_available_disk": 0.35,
    "purge_strategy": "oldest",
    "use_compression": true,
    "log_path": "/opt/ouster/conf/data/recordings"
}
```

7.12.2 Primary data_recorder Fields

Table7.23: Primary data_recorder Fields

Field	Format	Description
pointclouds_enabled	bool	Flag indicating if FlatBuf point clouds should be recorded to disk (default = False)
clusters_enabled	bool	Flag indicating if FlatBuf clusters should be recorded to disk (default = False)
trackedobjects_enabled	bool	Flag indicating if FlatBuf tracked objects should be recorded to disk (default = False)
images_enabled	bool	Flag indicating if NearIR/CalRef images should be captured in recordings (default = False)
zone_events_enabled	bool	Flag indicating if JSON zone events should be recorded to disk (default = False)
rotation_minutes	float	Data recording file rotation interval in fractional minutes...set to 0 to disable data recording (default = 5.0)
retention_count	int	Number of recordings to retain (default = 288 (1 day))
min_available_disk	float	Minimal percentage of available storage before implementing the purge_strategy (default = 0.35 (35%))
purge_strategy	string	Recording purge strategy to invoke when minimum available storage is reached oldest/newest/disable (default = oldest)
use_compression	bool	Flag indicating if recordings should be compressed w/GZip after rotation (default = true)
log_path	string	Recordings path (default = /opt/ouster/conf/-data/recordings/)

7.12.3 Optional data_recorder Fields

None

7.12.4 Accessing Binary Data

Binary data recordings can be accessed/downloaded by the user at the following URL:

```
https://<<host>>/data/recordings/
```

8 Connecting to Output

This section describes how to integrate with Ouster Detect to get object list, occupation data as well as any diagnostic information to monitor the health of the system.

The entrypoint for data generated from Ouster Detect is the *Lidar Hub Overview*. The Lidar Hub runs in it's own container and subscribes to the object list and occupation data generated by the perception and event zone containers. The role of Lidar Hub is to configure the output to accommodate the consumer. Through the Lidar Hub the user can select a subset of the available data, change the frequency at which they receive data, alter the field names in the messages, and/or the configure number of messages they receive at a time.

This section introduces some terms needed to understand the contents of each message.

- The term **lidar frame** refers to the point cloud data for one revolution for a single lidar sensor.
- The term **frame** will refer to the output from multiple unique lidar frames (i.e., one lidar frame from Lidar A, one lidar frame from Lidar B). The perception system calculates the object list and occupations on a single frame.
- An **object** is a moving entity in the scene which Ouster Detect is tracking over time. An object consists of a position, orientation, velocity, dimensions, and classification but also contains historic data such as how long we've been tracking it or what the initial position was when we saw it. An object is associated over time by it's *object id* or *UUID*. More details about what's contained in an object is listed in *Object Information*.
- An **event zone** refers to a user-defined area where object information is desired.
- An **occupation** is an object within an event zone at a specific instant in time.
- **Telemetry** is defined as time-based snapshots of the internal state of perception.
- **Alerts** represent error conditions where Ouster Detect is operating outside the desired state.
- **Attributes** refers to static information about the edge processor.

In this section, we go over how object lists, occupations, aggregation data, and diagnostic information is serialized into messages. We'll refer to these messages as *sources*. In *Publishing Configuration*, we go over how to configure publishing messages from these sources through a TCP stream and to an MQTT broker.

8.1 Object List Data

The object list data contains information about all the moving objects in the scene for a single frame. The object list is serialized in JSON format. The table below shows all the root-level information in the object list message for a single frame.

Table *Object list information* below describes the root-level information for an object list

Table 8.1: Object list definition

Field	Type	Description	Example
frame_count	integer	Number of frames since the system started outputting object lists. This value should be sequential	*1*
timestamp	integer	Timestamp of when the last point cloud arrived which contributed to the object list. Units in microseconds since Jan. 1, 1970.	*1667785112323234*
object_list	JSON array	Object list array. See Table *Object information* below	-

The *object_list* field contains an array of all moving objects in the scene. Table *Object information* below shows all the fields available for the objects.

Table 8.2: Object Information

Field	Type	Description	Example
id	integer	Unique number identifying object in current running instance of perception. If perception restarts, this count will reset.	*100*
uuid	string	Unique universal identifier (UUID) for objects over all running instances. If perception restarts, objects in the new running instance will have unique UUID's relative to all other running instances.	"74b4e42e-1989-40d0-91d6-ae498b173001"
classification	string	Classification of the object. For Ouster Detect 1.0, this set can be "PERSON", "TWO_WHEELER", "VEHICLE", "LARGE_VEHICLE".	"PERSON"
classification_confidence	float	Number between 0 and 1 representing the system's confidence in the assigned classification. 0 represents no confidence. 1 represents fully confident.	0.95
creation_ts	integer	Timestamp when the object was first visible in the system's field of view. This point in time will be before the object was tracked and classified. This number and frame_count both represent the duration the object has been in the sytem's field of view. Units in microseconds since Jan. 1, 1970.	1663175901389312
update_ts	integer	Timestamp the object was last updated. For objects the system has measured in the current frame, this timestamp will be the same as the timestamp at the root level. For objects the system has not measured in the current frame, this timestamp will stay at the timestamp when the object was last measured and lag behind the timestamp at the root level. An object will be considered measured when the lidars have captured a minimum number of points on the target. Units in microseconds since Jan. 1, 1970.	1663175920076580

Table 8.3: Object Information Cntd.

Field	Type	Description	Example
Dimensions	JSON container	Length, width, height of the bounding box enclosing all points on the object. Length is the extents along the x-axis, width the extents along the y-axis, height along the z-axis. Axis referenced are the axis of the object with x pointing in the direction of motion, y pointing perpendicular to the left, and z pointing up (right-hand rule).	*{"length": 0.4404, "width": 0.24217, "height": 0.6693}*
frame_count	integer	Number of frames an object has been visible for. This number and the creation_ts number both represent the duration the object has been in the sytem's field of view.	188
heading	float	Positive rotation about the z-axis (right-hand rule). Measured off of the positive x-axis.	17.7924
initial_position	JSON container	Initial XYZ location of the object in the first frame it was visible in the field of view of the lidars. This position is in the world reference frame.	*{"x": 0.8668, "y": -1.3293, "z": 1.1521}*
num_points	integer	Number of points belonging to the object	*101*
position	JSON container	Current XYZ location of the object in the world reference frame.	*{"x": 0.8668, "y": -1.3293, "z": 1.1521}*
position_uncertainty	JSON container	Estimated variance of the position measurement. Units in meters^2	*{"x": 0.03, "y": 0.19, "z": 0.38}*
orientation	JSON container	Quaternion representing the orientation of the object with respect to the world reference frame.	*{"w": 1.0000, "x": 0.0000, "y": 0.0000, "z": 0.0000}*
velocity	JSON container	Current rate of change in the XYZ position of the object. Velocity is in the world reference frame. Units are in m/s.	*{"x": 3.8668, "y": -10.3293, "z": 0.1521}*
velocity_uncertainty	JSON container	Estimated variance of the velocity measurement. Units in (m/s)^2.	*{"x": 0.4323, "y": 1.9123, "z": 0.1921}*

The sample below shows example object list data in JSON format


```
"object_list": [
  {
    "frame_count": 226599,
    "objects": [
      {
        "classification": "PERSON",
        "classification_confidence": 0,
        "creation_ts": 1663175901389312,
        "dimensions": {
          "height": 0.6693140268325806,
          "length": 0.44045835733413696,
          "width": 0.24217760562896729
        },
        "frame_count": 188,
        "heading": 17.792495727539062,
        "id": 1094,
        "initial_position": {
          "x": 0.866864025592804,
          "y": -1.3293535709381104,
          "z": 1.1521248817443848
        },
        "num_points": 1052,
        "orientation": {
          "qw": 0.9879699945449829,
          "qx": 0,
          "qy": 0,
          "qz": 0.15464559197425842
        },
        "position": {
          "x": 0.9670451879501343,
          "y": -1.5673401355743408,
          "z": 1.0692270994186401
        },
        "position_uncertainty": {
          "x": 0.00872983346207417,
          "y": 0.00872983346207417,
          "z": 0.03328978381826185
        },
        "update_ts": 1663175920076580,
        "uuid": "74b4e42e-1989-40d0-91d6-ae498b173001",
        "velocity": {
          "x": -0.03508763682949244,
          "y": 0.01674633024355329,
          "z": 0.029468615562023993
        },
        "velocity_uncertainty": {
          "x": 0.7745966692414834,
          "y": 0.7745966692414834,
          "z": 0.8143665760550121
        }
      }
    ],
    "timestamp": 1663175920089867
```


(continues on next page)

(continued from previous page)

```
  }
]
```

8.2 Occupation Data

The occupation data contains information about all objects which intersect any event zones for a single frame. The event zones are serialized in JSON format. Table *Occupation information* below shows all root-level information in the occupation message.

Table 8.4: Occupation Information

Field	Type	Description	Example
timestamp	integer	Timestamp of the when last point cloud arrived which contributed to the object list. Units in microseconds since Jan. 1, 1970.	*1667785112323234*
occupations	JSON array	Array of occupations for the current frame. See Table *Zone occupation information*.	*{[{"id": ..., "name": ..., "objects": ...}]}*

The *occupations* field contains an array of all zones which are being occupied at the current timestamp. Table *Zone occupation information* below shows the information available for each zone:

Table 8.5: Zone occupation information

Field	Type	Description	Example
id	integer	Zone unique identifier	*10*
name	string	Zone human-readable name	"South entrance"
num_objects	integer	Number of objects occupying the zone for the current frame	*101*
objects	JSON array	Array of objects occupying the zone for the current frame. See Table *Object information*.	-

The sample below shows example occupation data in JSON format

```
"occupations": [
  {
    "id": 1658947733821,
    "name": "Desk",
    "num_objects": 1,
    "objects": [
      {
        "classification": "PERSON",
```

(continues on next page)

(continued from previous page)


```
        "classification_confidence": 0,
        "creation_ts": 1663175901389312,
        "dimensions": {
          "height": 1.0555479526519775,
          "length": 0.713068962097168,
          "width": 0.6797903776168823
        },
        "frame_count": 6169,
        "heading": 239.63951110839844,
        "id": 1094,
        "initial_position": {
          "x": 0.866864025592804,
          "y": -1.3293535709381104,
          "z": 1.1521248817443848
        },
        "num_points": 2353,
        "orientation": {
          "qw": -0.4972730875015259,
          "qx": 0,
          "qy": 0,
          "qz": 0.8675940632820129
        },
        "position": {
          "x": 1.423563003540039,
          "y": -1.1258434057235718,
          "z": 1.4065864086151123
        },
        "position_uncertainty": {
          "x": 0.0087298334620742,
          "y": 0.0087298334620742,
          "z": 0.03328978381826278
        },
        "update_ts": 1663176519190541,
        "uuid": "74b4e42e-1989-40d0-91d6-ae498b173001",
        "velocity": {
          "x": -0.07373628162375315,
          "y": 0.17994346251739213,
          "z": 0.05531235350852935
        },
        "velocity_uncertainty": {
          "x": 0.7745966692415875,
          "y": 0.7745966692415875,
          "z": 0.8143665760552492
        }
      }
    ]
  }
],
"timestamp": 1663176519190541
```

8.3 Aggregation

Ouster Detect allows users to configure output to be grouped by zone and by time-interval in a process called *aggregation*. This allows a user to only receive condensed occupation data separated by classification. For example, imagine you're interested in knowing the number of pedestrians in the crosswalk at an intersection at 30 second intervals. The user could draw a zone around the intersection and setup aggregation for the desired zone and only receive this data from Ouster Detect. The user could then parse the received data and only iterate over the metrics reported for the objects with classification "PERSON". Along with the user-defined zones, a system-defined zone exists which includes all classifications. This zone is named "ALL" and has zone id *0*. Aggregation sends data calculated in two ways: by *time-series* and in *real-time*.

Time-series aggregation will report maximum, cumlative and average statitics about each classification that occupied the zone during the last interval. The table below shows the root-level information reported with each time-series aggregation message.

Table 8.6: Time-series aggregation output

Field	**Type**	**Description**
id	integer	Zone identifier as configured in UI
name	string	Human readable zone name
timestamp	integer	Timestamp at the end? of the timeseries interval. Units in microseconds since Jan. 1, 1970
classification_metrics	JSON array	Array of JSON objects where each object contains the classification metrics for a certain class (See *Time-series classification metrics* below)

The table below shows the time-series classification metrics for each class

Table 8.7: Time-series aggregation output for classification metrics

Field	Type	Description
description	string	Object classification name ("ALL" is an aggregate of all classifications)
active_occupants	integer	Number of objects occupying the zone in this time interval
avg_active_occupants	float	Average number of objects occupying the zone
total_visits	integer	Total visits to the zone
new_visitors	integer	New unique visitors to the zone in this interval
new_visits	integer	Unique visitors that left the zone in this interval
current_max_speed	float	Maximum speed observed in the zone in this interval
current_avg_speed	float	Average speed observed in the zone in this interval
overall_max_speed	float	Maximum speed observed in the zone since the application last started
overall_avg_speed	float	Average speed observed in the zone since the application last started
current_avg_dwell	float	Average dwell observed in the zone in this interval
overall_avg_dwell	float	Average dwell time observed in the zone since the application last started
last_update	integer	Timestamp when the zone was last updated. Units in microseconds since Jan. 1, 1970

The following code sample shows time-series aggregation data in JSON format


```
"aggregation_timeseries": [
    {
    "id": 1659467677889,
    "name": "Office",
    "timestamp": 1660664440000000,
    "classification_metrics": [
      {
        "description": "ALL",
        "active_occupants": 1,
        "avg_active_occupants": 1,
        "total_visits": 1,
        "new_visitors": 1,
        "departed_visitors": 0,
        "current_max_speed": 0.227,
        "current_avg_speed": 0.122,
        "overall_max_speed": 0.22669327197460112,
        "overall_avg_speed": 0.122,
        "current_avg_dwell": 0,
```


(continues on next page)

(continued from previous page)


```
        "overall_avg_dwell": 0,
        "last_update": 1660664449970829
      },
      {
        "description": "PERSON",
        "active_occupants": 1,
        "avg_active_occupants": 1,
        "total_visits": 1,
        "new_visitors": 1,
        "departed_visitors": 0,
        "current_max_speed": 0.227,
        "current_avg_speed": 0.122,
        "overall_max_speed": 0.22669327197460112,
        "overall_avg_speed": 0.122,
        "current_avg_dwell": 0,
        "overall_avg_dwell": 0,
        "last_update": 1660664449970829
      }
    ]
  }
]
```


Real-time aggregation will report the instantaneous changes to any zones at a user-defined frequency. By only reporting the changes, real-time aggregation helps reduce the bandwidth required to communicate relevant changes in the scene. The table below shows information that will be reported for any zones whose occupations counts change since the last update. The table below shows the root-level information reported for each real-time aggregation message

Table 8.8: Real-time aggregation output

Field	Type	Description
id	integer	Zone identifier
name	string	Human readable zone name
timestamp	integer	Timestamp when the classification metrics were measured, in microseconds since Jan. 1, 1970.
classification_metrics	JSON array	Array of JSON objects where each object contains the classification metrics for a certain class (See *Real-time classification metrics* below)

The table below shows the classification metrics reported for each class

Table 8.9: Real-time aggregation output for classification metrics

Field	Type	Description
description	string	Object classification name (“ALL” is an aggregate of all classifications)
active_occupants	integer	Number of objects currently occupying the zone
total_visits	integer	Total visits to the zone since the application last started
last_update	integer	Timestamp when the zone was last updated. Units in microseconds since Jan. 1, 1970

Time-series and real-time aggregation are configured by navigating to the “Settings” tab in the UI. From the “Settings Type” drop-down, select “Lidar Hub”. Navigate to the “lidarHub.perception.aggregation” subsection. The table *Aggregation configuration* below shows required an optional fields needed to configure aggregation

Table 8.10: Aggregation configuration

Field	Description
timeseries_secs	Duration of one interval, in seconds. Time-series aggregation messages will be emitted every interval (default = 60.0)
realtime_hertz	Frequency of updates for real-time aggregation (default = 1.0)
storage_path	Path for the non-volatile storage of aggregation metrics. Leaving this field blank will disable saving aggregation data to disk. (default = /opt/ouster/conf/data/storage).
source_hertz	Frequency perception data is processed. This value should be *more frequent* than the *timeseries_secs* and *realtime_hertz* (default = 2.0).
departure_debounce_ms	Duration an object needs to be outside of the zone boundaries before aggregation considers the object to have left the zone, in milliseconds (default = 2500). This is useful to remove noise when objects are situated at the zone boundaries.
excluded_classifications	List of classifications to be excluded from Aggregation (default = [“PROSPECT”, “UNKNOWN”]).

Once aggregation is configured, a publisher can be configured to send the aggregation data. (See *Publishing Configuration*)

The following code sample shows real-time aggregation data in JSON format

```
"aggregation_realtime": [
  {
    "id": 1659467677889,
    "name": "Office",
    "timestamp": 1660664443771926,
```

(continues on next page)

(continued from previous page)


```
    "classification_metrics": [
      {
        "description": "ALL",
        "active_occupants": 1,
        "total_visits": 1,
        "last_update": 1660664443771926
      },
      {
        "description": "PERSON",
        "active_occupants": 1,
        "total_visits": 1,
        "last_update": 1660664443771926
      }
    ]
  },
  {
    "id": 1658947733821,
    "name": "Desk",
    "timestamp": 1660664443771926,
    "classification_metrics": [
      {
        "description": "ALL",
        "active_occupants": 1,
        "total_visits": 1,
        "last_update": 1660664443771926
      },
      {
        "description": "PERSON",
        "active_occupants": 1,
        "total_visits": 1,
        "last_update": 1660664443771926
      }
    ]
  }
]
```


8.4 Telemetry Data

The **Telemetry data** provides time-based information about the state of the Ouster Detect system. Telemetry is forwarded from the following components:

- perception container (divided into lidar-specific telemetry and general perception telemetry)
- event zones container
- lidar hub container
- computation resources and other information regarding edge processor

The **Perception Telemetry** is divided into information related to each active lidar and data related to the overall perception system. The information for specific lidars includes the same telemetry information which can be retrieved from the lidar directly.

Table *Lidar telemetry information* below shows the information available from the perception telemetry for each lidar.

Table 8.11: Lidar telemetry information

Name	Description
serial_number	Identifier for the lidar
firmware	Container of values from firmware telemetry. See firmware manual for details
num_background_points	Number of points the system detected from static objects from the latest scan
num_no_return_points	Number of firings without a range return from the latest scan
num_failed_points	Number of no return points which the system has identified as likely from an object within range (i.e., not shooting off into space)
num_foreground_points	Number of points the system detected from moving objects from the latest scan
num_filtered_points	Number of points the system has filtered as noise
num_zone_filtered_points	Number of points the system has filtered from user-defined point zones
num_ground_points	Number of points identified as part of the ground from the latest scan
num_clusters	Number of moving objects in the latest scan

Table *Perception telemetry information* below shows information available from the perception telemetry for the whole system.

Table 8.12: Perception telemetry information

Name	Description
timestamp	Timestamp of the system, in microseconds
num_objects	Number of moving objects detected in the last frame
num_tcp_clients	Number of clients connected to the TCP server
num_websocket_clients	Number of clients connected to the websocket server
recording	Container of values about the current recording session. If not recording session is active, this value is *null*. See *Recording telemetry*.
largest_arrival_delta	Measures the difference between the earliest and latest frames which contributed to the last object list. Units in microseconds
lidars	Container of JSON objects where each object contains the information in Table *Lidar telemetry information*

Table *Recording telemetry* shows details about the *recording* value above when a recording session is active.

Table 8.13: Recording telemetry

Name	Description
timestamp	Timestamp of the system, in microseconds

The table below shows the information available from the event zones container

Table 8.14: Event zones telemetry information

Name	Description
up_time	Time since the event zone server was started, in microseconds?
num_occupations	Number of objects currently being positioned inside event zone(s)
num_websocket_clients	Number of web socket clients connected to the Event Zone Server
num_tcp_clients	Number of TCP clients connected to the Event Zone Server
timestamp	Timestamp of these values (microseconds since Jan. 1, 1970)

The table below shows information available from the lidar hub container

Table 8.15: Lidar hub telemetry information

Name	Description
mqtt_publishers_connected	Number of MQTT publishers configured
tcp_relay_server_connections	Number of TCP clients connected to Lidar Hub TCP Relay Servers

The table below shows the information available for computation resources

Table 8.16: Other telemetry information

Name	Description
cpu_utilization	Total CPU utilization across all cores
cpu_core_utilization	CPU utilization by cores
available_memory	Available memory on the compute device (bytes)
available_disk_space	Available storage on the compute device (bytes)
cpu_temperature_deg_c	Temperature measurement from the CPU of the edge processor, in degrees Celcius
cpu_core_temperature_deg_c	JSON list of CPU core temperatures, in degrees Celcius

Table *Other telemetry information* shows other telemetry information

Table 8.17: Other telemetry information

Name	Description
hardware_id	Unique identifier for the edge processor
timetamp	Timestamp of the edge processor, units in microseconds since Jan. 1, 1970
diagnostics_version	Single-digit version of the diagnostics information

Consumers can receive this data through web requests to /perception/api/v1/telemetry or by configuring a publisher with a "diagnostics" source (See *Publishing Configuration*)

The following code sample shows example telemetery data in JSON format

```
"hardware_id": "*1234567890",
"compute": {
  "cpu_utilization": 19.7,
  "cpu_core_utilization": [
    17.2,
    19,
    21.3,
    18.3
  ],
  "available_memory": 18890395648,
  "available_disk_space": 762672287744
},
"lidars": [
  {
    "model": "OS1",
    "serial_number": "992144000616",
    "firmware": {
      "input_current_ma": 704,
```

(continues on next page)

(continued from previous page)


```
      "input_voltage_mv": 24044,
      "internal_temperature_deg_c": 0,
      "phase_lock_status": "DISABLED",
      "timestamp_ns": 1727503038782740
    },
    "num_background_points": 83010,
    "num_foreground_points": 416,
    "num_failed_points": 865,
    "num_no_return_points": 46781,
    "num_zone_filtered_points": 4,
    "num_ground_points": 0,
    "num_clusters": 6,
    "num_filtered_clusters": 0
  }
],
"perception_server": {
  "up_time": 0,
  "num_objects": 0,
  "num_websocket_clients": 3,
  "num_tcp_clients": 3,
  "timestamp": 1660673730063520
},
"event_zone_server": {
  "up_time": 0,
  "num_occupations": 0,
  "num_websocket_clients": 2,
  "num_tcp_clients": 2,
  "timestamp": 1660673730110978
},
"lidar_hub": {
  "mqtt_publishers_connected": 0,
  "tcp_relay_server_connections": 0
},
"timestamp": 1660698930000000,
"diagnostics_version": 1
```

8.5 Alert Data

Ouster Detect exposes alerts to the user to communicate when the system is in an error state or when an aspect of the system should be investigated. The user can query the alerts through the RESTful interface as well as view them through the web page. The underlying format of the alerts is in JSON and is very similar to the format defined in the Ouster Firmware User Manual.

The table below lists the alerts in Ouster Detect along with their alert code, description, and guidance on what could be happening to cause it.

Table 8.18: Alert list

Alert Code	Name	Severity	Description/Possible causes	Recommended action
0x2000001	Sensor timeout	CRITICAL	The system is not receiving data from a lidar that's been configured. Possible causes are the UDP destination has been changed, the network infrastructure has been severed, or the lidar hardware has failed.	Investigate the state of the lidar by navigating to the IP address directly (If the lidar is directly connected to the catalyst, use the sensor proxy feature in the UI). Check the UDP destination address and confirm it's pointing to the Catalyst. The lidar's serial number will be in this alert's source information.
0x2000002	Dropping lidar data	WARNING	The system is not keeping up with lidar data it's ingesting	Check to make sure your compute device is within the recommended specifications for the number of lidars in your setup.
0x2000003	Dropping cluster	WARNING	The system is not able to process clusters fast enough	This is most likely an issue with the number of moving objects the system is trying to track in the scene. Check to see if this number is correlated with a high volume of moving objects

Table 8.19: Alert list Cntd.

Alert Code	Name	Severity	Description/Possible causes	Recommended action
0x2000004	Cluster delta over threshold	WARNING	The timestamps of data from all lidar sources are over the configurable threshold. This will likely cause issues with tracking since the system is attempting to process data from multiple lidars which are unsynchronized.	Make sure all sensors have a stable fast connection to the Catalyst. If this does not resolve the issue, contact customer support about putting the connected sensors in phase lock.
0x2000005	Websocket client behind	WARNING	A client connected to the websocket server is not keeping up with the data behind sent. The system is dropping data as a result	The source information field of this alert will contain the IP address of the client which is not keeping up. Check the network bandwidth between the Catalyst and this client
0x2000006	TCP client behind	WARNING	A client connected to the TCP server is not keeping up with the data behind sent. The system is dropping data as a result	The source information field of this alert will contain the IP address of the client which is not keeping up. Check the network bandwidth between the Catalyst and this client
0x2000007	Invalid packet source	ERROR	The system is receiving data from an an expected IP address on a network port configured to receive lidar data	The cause of this alert is most likely a configuration issue with one of the lidars. This occurs when a lidar was previously configured to send it's data to a Catalyst unit but the Catalyst unit is expecting data from another lidar now on that port. Navigate to the IP addresses of all lidars on the network and confirm any unused lidars are not sending their data to the Catalyst

Table 8.20: Alert list Cntd.

Alert Code	Name	Severity	Description/Possible causes	Recommended action
0x2000008	Lidar packet mismatch	WARNING	The system is receiving data of unexpected size on a port it's expecting lidar data on. This is usually configuration issue related to UDP profiles. There is likely a difference between the UDP profile Ouster Detect is expecting and the UDP profile configured on the lidar.	The cause of this alert is most likely a configuration issue with one of the lidars. Navigate to the IP address of the browser of the lidar in source info and confirm the UDP profile is the one Ouster Detect is configured for (Default is RNG15_RFL8_NIR8)

Ouster Detect also forwards alerts from the lidar. We distinguish between lidar alerts and Ouster Detect's alert with the 4th byte (0x01XXXXXX indicates a lidar alert, 0x02XXXXXX indicates a Ouster Detect alert). See the Ouster Firmware User Manual for a description of the lidar alerts.

Ouster Detect alerts go from RESET to ACTIVE when the condition for the alert is satisfied. Once the condition becomes false, the alert state goes from ACTIVE to LOGGED. Users can query the ACTIVE, LOGGED, or all alerts with the endpoints defined in **insert reference to diagnostic section**.

The table below describes the information returned when querying alerts from Ouster Detect.

Table 8.21: Alert fields

Field	Type	Description
active	boolean	Whether the condition enabling this alert is current true
active_count	integer	How many times this alert has been true since it first occurred
alert_code	string	Hexidecimal identifier for the alert type. See table above
category	string	Module where the alert exists
first_occurred	integer	Timestamp when alert first occurred, in microseconds since Jan. 1, 1970
last_occurred	integer	Timestamp when alert last occurred, in microseconds since Jan. 1, 1970
source_info	string	String identifier for alert instance. For example, in a multi-lidar setup we can have alerts for different instances processing two different lidars. This field distinguishes between those instances such that we can have more than one active alert with the same *alert_code*. In the case where we want to distinguish between lidars, this field would be the serial number of the lidar
id	integer	Unique identifier for all alerts generated from the system over all time. Two alerts occurring at different times with the same alert code will have different id's.
level	string	Severity of the alert. Can be *WARNING*, *ERROR*, or *CRITICAL*
msg	string	Short description
msg_verbose	string	Long description

Consumers can receive this data through web requests to /perception/api/v1/alerts or by configuring a publisher with a "diagnostics" source (See *Publishing Configuration*).

The following code sample shows alert data in JSON format


```
"alerts": [
  {
    "active": true,
    "category": "SHOT_LIMITING",
    "cursor": 2833,
    "alert_code": "0x0100000f",
    "level": "WARNING",
    "source_info": "OS1-992144000616",
    "id": 13962805723535194000,
    "msg": "Shot limiting mode is active. Laser power is partially attenuated; please see user guide for
↪heat sinking requirements.",
    "msg_verbose": "Shot limiting has started.",
    "first_occurred": 0,
    "last_occurred": 0,
    "active_count": 0
  },
  {
    "active": true,
    "category": "UDP_TRANSMISSION",
    "cursor": 2822,
    "alert_code": "0x01000016",
    "level": "WARNING",
    "source_info": "OS1-992144000616",
    "id": 11061546793406626000,
    "msg": "Could not send lidar data UDP packet to host; check that network is up.",
    "msg_verbose": "Failed to send lidar UDP data to destination host 10.0.0.39:34642",
    "first_occurred": 0,
    "last_occurred": 0,
    "active_count": 0
  }
]
```


8.6 Publishing Configuration

This section goes over how to configure Ouster Detect publishers. Ouster Detect allows the user to configure the rate at which messages are sent, the number of messages to put in a packet, and the field names for each piece of information. It supports sending information through a TCP stream as well as through MQTT.

8.7 Downsampling and Batching

Ouster Detect calculates the object list and occupations at the same rate which it's receiving data from the lidars. Sometimes this frequency and volume of data can be too much for the consumer. To remedy this, Ouster Detect allows the user to configure a reduction in the data sent through *downsampling*, *batching*, and *mapping*. *Downsampling* reduces the frequency of the object list and occupation messages being sent from Ouster Detect, giving the user snapshots of the state of each object without having to consume it at the rate it's being calculated. *Batching* collects messages desired by the consumer and sends them in bursts at a desired frequency.

For example, say a consumer wants to receive the data at 1 Hz intervals but only wants to receive a

message every 10 seconds with all this information (i.e., receive 10 object lists every second representing the 10 second interval). The user can configure downsampling to 1 Hz and batching to 0.1 Hz. The values can be configured for the protocols below with the *data_hertz* configuring the downsampling and the *transmit_hertz* configuring the batching. Setting either of these values to *0* disables the functionality.

8.8 Mapping

The *mapping* functionality allows the user to change the the fields being published, enabling translation to another protocol using JSON. This is convenient if the consumer wants to send the Ouster Detect data directly to another service using different field names. Mapping also has the ability to remove fields, reducing the bandwidth required by not sending extra information. Mapping is achieved by setting the *field_mapping* value when configuring a publisher. If the field mapping value is empty, the full message will be sent with the fields defined by Ouster Detect.

Once a value is specified in *field_mapping*, only the specified values will be sent. Omitting a field will stop it from being sent.

8.8.1 Publishing Protocols

Ouster Detect supports sending messages containing object lists, occupations, aggregation (time-series and/or real-time), and diagnostics (telemetry and alert information). These messages are referred to as *sources*. Publshers can be configured to send messages from sources through a TCP stream or through MQTT. As described in Section *Publishing Configuration*, Ouster Detect can be configured to downsample, batch and map the data as desired.

8.9 TCP stream

To send distinct messages over TCP, we need a way to signal the beginning and end of a message. Ouster Detect does this by first sending a **32-bit unsigned integer with big-endian byte-ordering** representing the total length of the message (including the 32-bit length). The consumer can parse this length and discern how much more there is to read for the current message (i.e., length - 4 bytes). The consumer can then read the remaind bytes in the message and interpret them as JSON. See *Code Samples* for a code sample reading a TCP stream in python.

A picture here would be useful showing an example of the first 4 bytes and the remainder of the message

A TCP stream requires a source (i.e., object_list, occupations, aggregation_realtime, aggregation_timeseries, diagnostics), a server port, a data frequency (for downsampling), and a transmit frequency (for batching). The table below shows the required parameters for each TCP stream

Table 8.22: Required TCP Relay server configuration

Field	Type	Description
source	string	JSON data stream source (either object_list, occupations, aggregation_realtime, aggregation_timeseries, or diagnostics)
port	TCP Relay Server listening port	Port for server to listen on
data_hertz	Hertz	Downsampling frequency to keep data. Setting to *0* will disable downsampling and send data as fast as it's available.
transmit_hertz	Hertz	Frequency to send data at. If this value is greater than *data_hertz*, batching will be enabled keeping data at *data_hertz* and sending it at this frequency.

Table 8.23: Optional TCP Relay server configuration

Field	Type	Description
tls	bool	Enable TLS (default = true)
return_as_array	bool	Controls the wrapping of transmissions into a well-formed JSON array (default = true)
field_mappings	JSON dict	JSON field filtering and renaming (default = {})
decimal_precision	integer	Decimal precision for floating-point values (default = -1 (disabled))
heartbeat_interval_secs	float	Interval to send heartbeat_message when source stream is inactive (default = 0 (disabled))
heartbeat_message	string	Message to transmit when source stream is inactive (default = "{}")

To configure a TCP replay server, navigate to the "Settings" tab in the UI and in the "Settings Type" drop-down, select "LidarHub". Scroll down to the "tcp_servers" section and configure one of the server entries as desired. By default, Ouster Detect has two TCP servers configured; one for the object list on port 3302 and one for the occupations on port 3303. Both default configurations down sample data to 1 Hz with no batching. Ouster Detect configures the incoming ports through a *compose.yaml* file used by *docker*. Users with a catalyst units are restricted to use 3302 and 3303. Users with their own hardware can change the *compose.yaml* file to expose more ports if desired.

8.10 MQTT

Ouster Detect supports publishing to MQTT brokers enabling IoT applications using this protocol. The table below displays the information required to configure an MQTT publisher.

Table 8.24: Optional TCP Relay server configuration

Field	Type	Description
source	string	JSON data stream source (either object_list, occupations, aggregation_realtime, aggregation_timeseries, or diagnostics)
host	string	MQTT Broker host name/IP address
port	integer	MQTT Broker port number (default = 1883)
user_name	string	MQTT Broker username to authenticate with [MQTT-3.1.3-11] (default = "")
password	string	MQTT Broker password to authenticate with. Optional, set to "" if not required. (default = "")
topic	string	MQTT Broker topic to publish to
qos	integer	Quality of Service (0 = At most once, 1 = At least once (default), 2 = Exactly once)
data_hertz	integer	Downsampling frequency to keep data. Setting to *0* will disable downsampling and send data as fast as it's available.
transmit_hertz	integer	Frequency to send data at. If this value is greater than *data_hertz*, batching will be enabled keeping data at *data_hertz* and sending it at this frequency.

To configure a MQTT publisher, navigate to the "Settings" tab in the UI and in the "Settings Type" dropdown, select "LidarHub". Scroll down. Scroll down to the "mqtt_publishers" section and configure one of the publisher entries as desired. There are no MQTT publishers configured by default.

9 Networking Guide - Ouster sensors

Warning: Gemini software does not support link-local connections. Please use instructions in the following guide to configure your sensor to a static IP or to use DHCP to configure the sensor IP. If you need further assistance please contact Ouster support.

This guide will help you understand how to quickly get connected to your sensor to start doing great things with it. When trying to connect to the sensor for the first time there are some basics that need to be achieved for successful communication between the host machine and the sensor.

We need to ensure that the sensor receives an IP address from the host machine so that we can talk to it. This can be achieved with a few different methods such as DHCP, link-local, static IP. We also need to ensure that the sensor and the host machine are talking on the same subnet.

Once the sensor receives an IP address and is on the correct subnet we can talk to it using its hostname, `os-991234567890.local`, where `991234567890` is the sensor serial number. The sensor serial number can be found on a sticker affixed to the top of the sensor.

Based on the platform being used the user can refer to the following:

- *Windows*
- *macOS*
- *Linux*

Note: For support on the edge processor refer to *Connecting Sensors to the Edge Processor*.

9.1 Networking Terminology

If some of this terminology is new to you don't fret, we have defined some of it for you. Here is some basic terminology that will help you digest the steps and be more familiar with networking in general.

IPv4 Address This is the address that can be used to communicate with devices on a network. The format of an IPv4 address is a set of four octets, `xxx.xxx.xxx.xxx` with `xxx` being in the range `0-255`. For example, your host machine Ethernet port may have an address of `192.0.2.1` and your sensor may have an address of `192.0.2.130`.

DHCP (Dynamic Host Configuration Protocol) Server This is a server that may run on your host machine, switch, or router which will serve an IPv4 address to a device that is connected to it. It will ensure that each device connected will have a unique IPv4 address on the network.

Link-local IPv4 Address These are the addresses that are self-assigned between the host machine and a device connected to it in the absence of a DHCP server. They are only valid within the network segment that the host is connected to. The addresses lie within the block `169.254.0.0/16 (169.254.0.0 - 169.254.255.255)`.

Subnet Mask This defines which bits of the IPv4 address are the network prefix and which are the host identifiers. See the table below for an example.

	Binary Form	Decimal-dot notation
IP address	`11000000.00000000.00000010.10000010`	`192.0.2.130`
Subnet mask	`11111111.11111111.11111111.00000000`	`255.255.255.0`
Network prefix	`11000000.00000000.00000010.00000000`	`192.0.2.0`
Host identifier	`00000000.00000000.00000000.10000010`	`0.0.0.130`

Note: Subnet mask can be abbreviated with the number of bits that apply to the network prefix. E.g. /24 for 255.255.255.0 or /16 for 255.255.0.0.

Static IPv4 Address This is when you specify the addresses for the host machine and/or connected device rather than letting the host machine self-assign or using a DHCP server. For example, you may want to specify the host machine IPv4 address to be `192.0.2.100/24` and the sensor to be `192.0.2.200`.

Hostname This is the more human readable name that comes with your sensor. The sensor's hostname is `os-991234567890.local`, where `991234567890` is the sensor serial number.

Note: The `.local` portion of the hostname denotes the local domain used in combination with multicast DNS (mDNS). It is employed when using the sensor in a local network environment with supporting operating system services. This means when the sensor is directly connected to the host machine or if the host machine and sensor are on the same network connected through a router or switch. If you are trying to connect to the sensor on another domain with a supporting DHCP and DNS server configuration you should replace the .local with the domain the sensor is on. For example, if the sensor is connected to a network with domain `ouster-domain.com` the sensor will be reachable on `os-991234567890.ouster-domain.com`.

9.2 Windows

The following steps have been tested on Windows 10. The sensor's hostname is `os-991234567890.local`, where `991234567890` is the sensor serial number. The sensor serial number can be found on a sticker affixed to the top of the sensor.

9.2.1 Connecting the Sensor

1. Connect the sensor to an available Ethernet port on your host machine or router.
2. The sensor will automatically obtain an IP address either through link-local or DHCP (if preconfigured) depending on your network configuration.

Note: It can take up to 60 seconds to obtain an IP address from the initial power-up of the sensor.

9.2.2 The Sensor Homepage

1. Type `os-991234567890.local`/ in the address bar of your browser to view the sensor homepage

---

Note: If you are unable to load the sensor homepage, follow the steps in *Determining the IPv4 Address of the Sensor* to verify your sensor is on the network and has a valid IPv4 address.

9.2.3 Determining the IPv4 Address of the Sensor

Warning: Gemini software does not support link-local connections. Please use instructions in the following guide to configure your sensor to a static IP or to use DHCP to configure the sensor IP. If you need further assistance please contact Ouster support.

1. Open a command prompt on the host machine by pressing **Win+X** and then **A**
2. Use the ping command to determine the IPv4 address of the sensor

Command

```
ping -4 [sensor_hostname]
```

Example

```
C:\\WINDOWS\\system32>ping -4 |os-sn|
```

Note: If this command hangs you may need to go back and configure your interface to link-local in the section *Connecting the Sensor*

Response

```
Pinging |os-sn| [|sensor-ip|] with 32 bytes of data:
Reply from |sensor-ip|: bytes=32 time<1ms TTL=64
Reply from |sensor-ip|: bytes=32 time<1ms TTL=64
Reply from |sensor-ip|: bytes=32 time<1ms TTL=64
Reply from |sensor-ip|: bytes=32 time<1ms TTL=64
```

(continues on next page)

(continued from previous page)

```
Ping statistics for |sensor-ip|:
  Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

3. You can also browse for the sensor IPv4 address using dns-sd and the sensor hostname. Learn more about this in *Finding a Sensor with mDNS Service Discovery*

Command

```
dns-sd -G v4 [sensor_hostname]
```

Example

```
C:\\WINDOW\\system32>dns-sd -G v4 |os-sn|
```

Response

```
Timestamp     A/R  Flags   if   Hostname                Address         TTL
14:22:46.897  Add  2       6    |os-sn|  |sensor-ip|   120
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

9.2.4 Determining the IPv4 Address of the Interface

1. Open a command prompt by pressing **Win+X** and then **A**
2. View the IPv4 address of your interfaces

Command

```
netsh interface ip show config
```

Example

```
C:\\WINDOWS\\system32>netsh interface ip show config
```

Response

```
Configuration for interface "Local Area Connection"
    DHCP enabled:                         Yes
```

(continues on next page)

(continued from previous page)

```
    IP Address:                           |interface-ip|
    Subnet Prefix:                        169.254.0.0/16 (mask 255.255.0.0)
    InterfaceMetric:                      25
    DNS servers configured through DHCP:  None
    Register with which suffix:           Primary only
    WINS servers configured through DHCP: None

Configuration for interface "Loopback Pseudo-Interface 1"
    DHCP enabled:                         No
    IP Address:                           127.0.0.1
    Subnet Prefix:                        127.0.0.0/8 (mask 255.0.0.0)
    InterfaceMetric:                      75
    Statically Configured DNS Servers:    None
    Register with which suffix:           Primary only
    Statically Configured WINS Servers:   None
```

- In this example, your sensor is plugged into interface "Local Area Connection"
- Your host IPv4 address will be on the line that starts with IP Address: In this case it is `169.254.0.1`

Note: If your interface IPv4 address is of the form `169.254.x.x`, it is connected via link-local to the sensor. This means that Windows self-assigned an IP address in the absence of a DHCP server.

9.2.5 Setting the Host Interface to DHCP

Use this to set your interface to automatically obtain an IP address via DHCP. This is useful for architectures that need to be more plug and play.

Set your interface to DHCP.

Command

```
netsh interface ip set address ["Network Interface Name"] dhcp
```

Example

with interface name `"Local Area Connection"`

```
C:\\WINDOWS\\system32>netsh interface ip set address "Local Area Connection" dhcp
```

Response

```
blank
```

9.2.6 Setting the Host Interface to Static IP

Use this to set your interface to be assigned a static IPv4 address. This is useful for controlling the IP address that the sensor will be sending data to.

Set your interface to static.

Command

```
netsh interface ip set address name="Network Interface Name" static [IP address] [Subnet Mask]
[Gateway]
```

Example

with interface name "Local Area Connection" and IPv4 address 192.0.2.1/24.

```
C:\\WINDOWS\\system32>netsh interface ip set address name="Local Area Connection"
 static 192.0.2.1/24
```

Note: The `/24` is shorthand for Subnet Mask = `255.255.255.0`

Response

```
blank
```

9.2.7 Finding a Sensor with mDNS Service Discovery

The sensor announces its presence on the network using Multicast Domain Name Service (mDNS) with a service type named `_roger._tcp`. You can use service discovery tools such as Bonjour browser (Windows) to find all sensors connected to the network.

Note: Click Bonjour to install Bonjour Browser.

Example using Bonjour Browser:

Step 1: User can download the Bonjour Browser

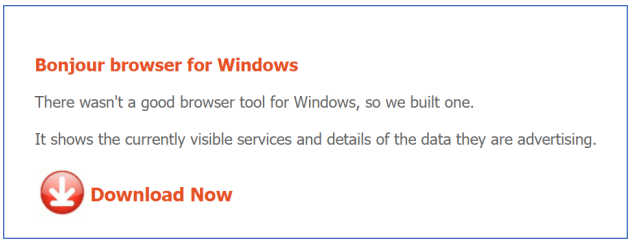


Figure 9.1: Downloading Application

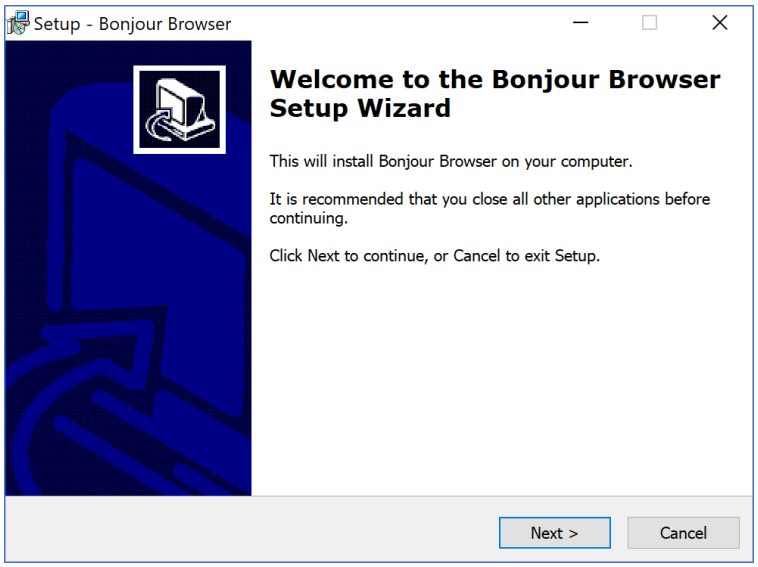


Figure 9.2: Software Setup and Installation

Step 2: Sensor announces its presence on the network using Multicast Domain Name Service (mDNS) with a service type named `_roger._tcp`. Click on this to get all the information required.

9.3 macOS

The following steps have been tested on macOS 10.15.4. The sensor's hostname is `os-991234567890.local`, where `991234567890` is the sensor serial number. The sensor serial number can be found on a sticker affixed to the top of the sensor.

9.3.1 Connecting the Sensor

1. Connect the sensor to an available Ethernet port on your host machine or router.
2. The sensor will automatically obtain an IP address either through link-local or DHCP (if preconfigured) depending on your network configuration.

Note: It can take up to 60 seconds to obtain an IP address from the initial power-up of the sensor.

9.3.2 The Sensor Homepage

1. Type `os-991234567890.local` in the address bar of your browser to view the sensor homepage

---

Note: If you are unable to load the sensor homepage, follow the steps in *Determining the IPv4 Address of the Sensor* to verify your sensor is on the network and has a valid IPv4 address.

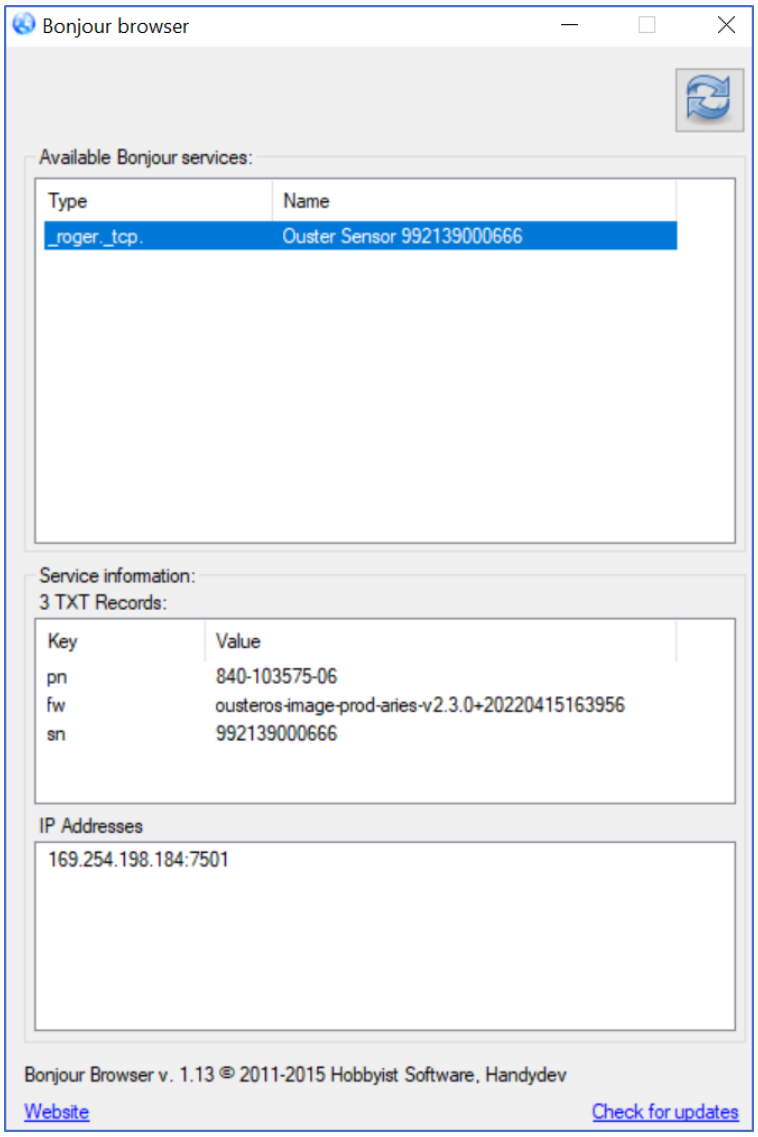


Figure 9.3: `_roger._tcp`

9.3.3 Determining the IPv4 Address of the Sensor

Warning: Gemini software does not support link-local connections. Please use instructions in the following guide to configure your sensor to a static IP or to use DHCP to configure the sensor IP. If you need further assistance please contact Ouster support.

1. Open a Terminal window on the host machine by pressing **CMD+SPACE** and typing **Terminal** in the search bar, then press enter.
2. Use the ping command to determine the IPv4 address of the sensor

Command

```
ping -c3 [sensor_hostname]
```

Example

```
Mac-Computer:~ username$ ping -c3 |os-sn|
```

Note: If this command hangs you may need to go back and configure your interface to link-local in the section *Connecting the Sensor*

Response

```
PING |os-sn| (|sensor-ip|): 56 data bytes
64 bytes from |sensor-ip|: icmp_seq=0 ttl=64 time=0.644 ms
64 bytes from |sensor-ip|: icmp_seq=1 ttl=64 time=0.617 ms
64 bytes from |sensor-ip|: icmp_seq=2 ttl=64 time=0.299 ms

--- |os-sn| ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.299/0.520/0.644/0.157 ms
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

3. You can also browse for the sensor IPv4 address using dns-sd and the sensor hostname. Learn more about this in *Finding a Sensor*

Command

```
dns-sd -G v4 [sensor_hostname]
```

Example

```
Mac-Computer:~ username$ dns-sd -G v4 |os-sn|
```

Response

```
DATE: ---Tue 28 Apr 2020---
11:40:43.228  ...STARTING...
Timestamp     A/R  Flags  if  Hostname               Address          TTL
11:40:43.414  Add  2      18  |os-sn|.  |sensor-ip|     120
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

9.3.4 Determining the IPv4 Address of the Interface

This will help you find the IPv4 address of the interface that you have plugged the sensor into. It is helpful to know which interface you have plugged into, e.g. `en1` in the example below.

1. Open a Terminal window on the host machine by pressing **CMD+SPACE** and typing **Terminal** in the search bar, then press enter.
2. View the IPv4 address of your interfaces

Command

```
ifconfig
```

Example

```
Mac-Computer:~ username$ ifconfig
```

Response

```
   lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 16384
      options=1203<RXCSUM,TXCSUM,TXSTATUS,SW_TIMESTAMP>
      inet 127.0.0.1 netmask 0xff000000
      inet6 ::1 prefixlen 128
      inet6 fe80::1%lo0 prefixlen 64 scopeid 0x1
      nd6 options=201<PERFORMNUD,DAD>
   en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      options=400<CHANNEL_IO>
      ether 38:f9:d3:d6:33:8a
      inet6 fe80::1c30:1246:93a2:9f68%en0 prefixlen 64 secured scopeid 0x7
      inet 192.0.2.7 netmask 0xffffff00 broadcast 192.0.2.255
      nd6 options=201<PERFORMNUD,DAD>
      media: autoselect
      status: active
   en1: flags=8963<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
      options=400<CHANNEL_IO>
      ether 48:65:ee:1d:22:35
      inet6 fe80::c27:1917:47ed:bcfe%en1 prefixlen 64 secured scopeid 0x12
      inet |interface-ip| netmask 0xffff0000 broadcast 169.254.255.255
      nd6 options=201<PERFORMNUD,DAD>
      media: autoselect (1000baseT <full-duplex>)
      status: active


* In this example, your sensor is plugged into interface ``en1``
* Your host IPv4 address will be on the line that starts with ``inet``: In this case it is |interface-ip|
```

Note: If your interface IPv4 address is of the form `169.254.x.x`, it is connected via link-local to the sensor. This means that the macOS self-assigned an IP address in the absence of a DHCP server.

9.3.5 Setting the Host Interface to DHCP

Use this to set your interface to automatically obtain an IP address via DHCP. This is useful for architectures that need to be more plug and play.

Set your interface to DHCP

Command

```
sudo ipconfig set [interface_name] DHCP
```

Example

with interface name `en1`

```
Mac-Computer:~ username$ sudo ipconfig set en1 DHCP
```

Response

```
blank

Note: However you can verify the change has been made with the ``ifconfig`` command.
      The ``inet`` line will be blank if nothing is plugged in or shows the DHCP or
      link-local self-assigned IPv4 address. E.g. |interface-ip|


en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500

options=6407<RXCSUM,TXCSUM,VLAN_MTU,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
   ether 48:65:ee:1d:22:35
   inet6 fe80::1c24:5e0a:2ea8:12e9%en1 prefixlen 64 secured scopeid 0x7
   inet |interface-ip| netmask 0xffff0000 broadcast 169.254.255.255
   nd6 options=201<PERFORMNUD,DAD>
   media: autoselect (1000baseT <full-duplex>)
   status: active
```

9.3.6 Setting the Host Interface to Static IP

Use this to set your interface to be assigned a static IPv4 address. This is useful for controlling the IP address that the sensor will be sending data to.

Set your interface to static

Command

```
sudo ipconfig set [interface_name] MANUAL [ip_address] [subnet_mask]
```

Example

with interface name `en1` and IPv4 address `192.0.2.1` and subnet mask `255.255.255.0`.

```
Mac-Computer:~ username$ sudo ipconfig set en1 MANUAL 192.0.2.1 255.255.255.0
```

Note: The `/24` is shorthand for Subnet Mask = `255.255.255.0`

Response

```
blank

Note: However you can verify the change has been made with the ``ifconfig`` command.
      The ``inet`` line will show the static IPv4 address. e.g. ``192.0.2.1``.
```

(continues on next page)

(continued from previous page)

```
en1: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500

options=6407<RXCSUM,TXCSUM,VLAN_MTU,CHANNEL_IO,PARTIAL_CSUM,ZEROINVERT_CSUM>
 ether 48:65:ee:1d:22:35
 inet6 fe80::1c24:5e0a:2ea8:12e9%en1 prefixlen 64 secured scopeid 0x7
 inet 192.0.2.1 netmask 0xffffff00 broadcast 192.0.2.255
 nd6 options=201<PERFORMNUD,DAD>
 media: autoselect (1000baseT <full-duplex>)
 status: active
```

9.3.7 Finding a Sensor

With mDNS Service Discovery:

The sensor announces its presence on the network using Multicast Domain Name Service (mDNS) with a service type named `_roger._tcp`. You can use service discovery tools such as dns-sd (Windows/macOS) to find all sensors connected to the network.

1. Find all sensors and their associated service text on a network.

Command

```
dns-sd -Z [service type]
```

Example

```
Mac-Computer:~ username$ dns-sd -Z _roger._tcp
```

Response

```
Browsing for _roger._tcp
DATE: ---Thu 30 Apr 2020---
17:27:52.242  ...STARTING...

; To direct clients to browse a different domain, substitute that domain in
 place of '@'
lb._dns-sd._udp                                 PTR     @

; In the list of services below, the SRV records will typically reference dot-local
 Multicast DNS names.
; When transferring this zone file data to your unicast DNS server, you'll need to
 replace those dot-local
; names with the correct fully-qualified (unicast) domain name of the target host
 offering the service.

_roger._tcp                                     PTR
Ouster Sensor |sn|._roger._tcp
Ouster Sensor |sn|._roger._tcp    SRV     0 0 7501 |os-sn|. ;
Replace with unicast FQDN of target host
Ouster Sensor |sn|._roger._tcp    TXT     "pn=840-102145-B" "sn= |sn|"
"fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn= |sn|"
```

2. Browse for the sensor IPv4 address using dns-sd and the sensor hostname.

Command

```
dns-sd -G v4 [sensor_hostname]
```

Example

```
Mac-Computer:~ username$ dns-sd -G v4 |os-sn|
```

Response

```
DATE: ---Thu 30 Apr 2020---
17:37:33.155  ...STARTING...
Timestamp     A/R  Flags  if  Hostname               Address           TTL
17:37:33.379  Add  2      7   |os-sn|.  |sensor-ip|     120
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`

With Discovery App:

Step 1: User can download the Discovery DNS-SD

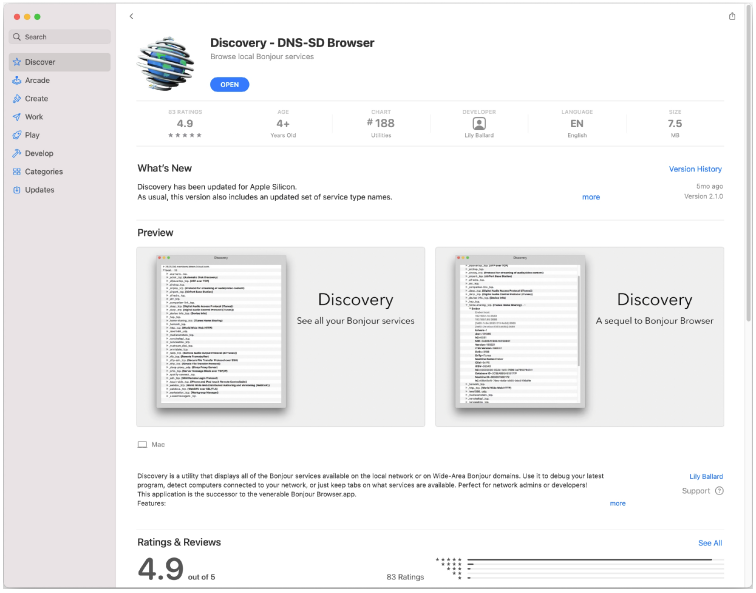


Figure 9.4: Downloading Application

Step 2: Using finder, the user can search for *Discovery*

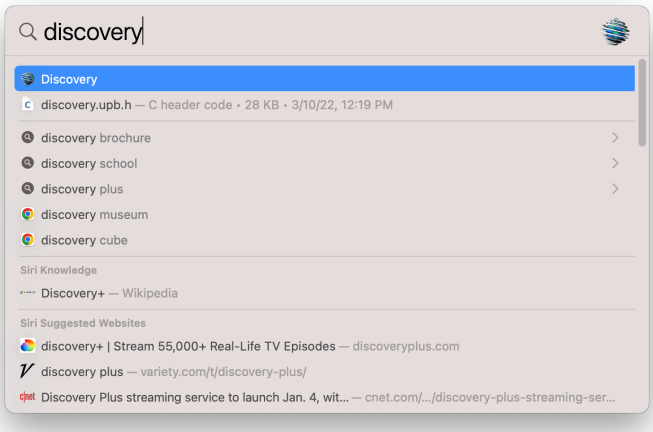


Figure 9.5: Finding the Application

Step 3: Sensor announces its presence on the network using Multicast Domain Name Service (mDNS) with a service type named `_roger._tcp`. Click on this to get all the information required.

9.4 Linux

The following steps have been tested on Ubuntu 18.04 & 20.04.4 LTS. The sensor's hostname is `os-991234567890.local`, where `991234567890` is the sensor serial number. The sensor serial number can be found on a sticker affixed to the top of the sensor.

9.4.1 Connecting the Sensor

1. Connect the sensor to an available Ethernet port on your host machine or router.
2. The sensor will automatically obtain an IP address either through link-local or DHCP (if preconfigured) depending on your network configuration.
3. If directly connecting to the host machine you may need to set your Ethernet interface to `Link-Local Only` mode. This can be done via the command line or GUI. See instructions in *Setting the Interface to Link-Local Only*

Note: It can take up to 60 seconds to obtain an IP address from the initial power-up of the sensor.

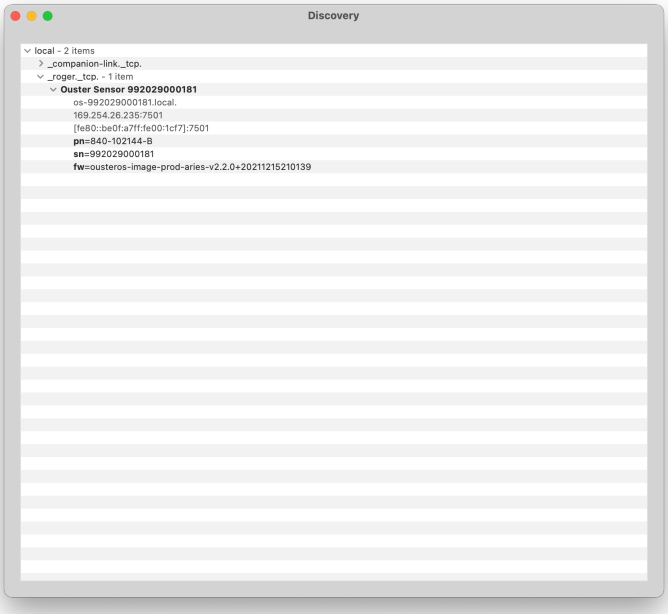


Figure 9.6: `_roger._tcp`

9.4.2 Setting the Interface to Link-Local Only

Via Command Line

Command

```
nmcli con modify [interface_name] ipv4.method link-local ipv4.addresses ""
```

Note: To identify the name of your connection, please use the command: `nmcli connection show`.

Example

with interface name `eth0` and IPv4 address `""`.

```
username@ubuntu:~$ nmcli con modify eth0 ipv4.method link-local ipv4.addresses ""
```

Response

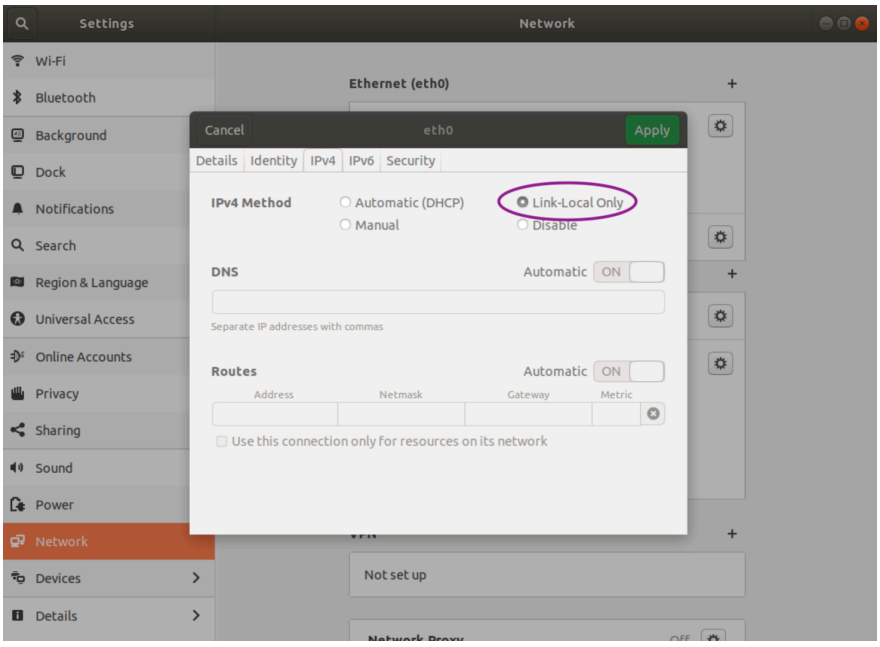
```
blank

Note: However you can verify the change has been made with the ``ip addr`` command.
      The ``inet`` line for the interface ``eth0`` will show the link-local IPv4 address automatically
      negotiated once the sensor is reconnected to the interface. e.g. |interface-ip|.
```

(continues on next page)

(continued from previous page)

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
       default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
      valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
      valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
       default qlen 1000
   link/ether 00:0c:29:2b:cc:48 brd ff:ff:ff:ff:ff:ff
   inet |interface-ip|/16 brd 169.254.255.255 scope link noprefixroute eth0
      valid_lft forever preferred_lft forever
   inet6 fe80::be9f:d2a4:4451:3dfe/64 scope link noprefixroute
      valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
   default qlen 1000
   link/ether 00:50:56:28:7a:8a brd ff:ff:ff:ff:ff:ff
   inet 172.16.79.232/24 brd 172.16.79.255 scope global wlan0
      valid_lft forever preferred_lft forever
   inet6 fe80::250:56ff:fe28:7a8a/64 scope link
      valid_lft forever preferred_lft forever
```

Via GUI: The image below illustrates how to set the interface to `Link-Local Only` mode using the graphical user interface.



Note: It can take up to 60 seconds to obtain an IP address from the initial power-up of the sensor.

9.4.3 The Sensor Homepage

1. Type `os-991234567890.local/` in the address bar of your browser to view the sensor homepage

---

Note: If you are unable to load the sensor homepage, follow the steps in *Determining the IPv4 Address of the Sensor* to verify your sensor is on the network and has a valid IPv4 address.

9.4.4 Determining the IPv4 Address of the Sensor

> **Warning:** Gemini software does not support link-local connections. Please use instructions in the following guide to configure your sensor to a static IP or to use DHCP to configure the sensor IP. If you need further assistance please contact Ouster support.

1. Open a Terminal window on the host machine by pressing **Ctrl+Alt+T**.
2. Use the `ping` command to determine the IPv4 address of the sensor

Command

```
ping -4 -c3 [sensor_hostname]
```

Example

```
username@ubuntu:~$ ping -4 -c3 |os-sn|
```

Note: If this command hangs you may need to go back and configure your interface to link-local in the section *Setting the Interface to Link-Local Only*

Response

```
PING |os-sn| (|sensor-ip|) 56(84) bytes of data.
64 bytes from |os-sn| (|sensor-ip|): icmp_seq=1 ttl=64 time=1.56 ms
64 bytes from |os-sn| (|sensor-ip|): icmp_seq=2 ttl=64 time=0.893 ms
64 bytes from |os-sn| (|sensor-ip|): icmp_seq=3 ttl=64
time=0.568 ms

--- |os-sn| ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2025ms
rtt min/avg/max/mdev = 0.568/1.008/1.565/0.416 ms
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

---

3. You can also browse for the sensor IPv4 address using `avahi-browse` and the sensor service type, which is `_roger._tcp`. Learn more about this in *Finding a Sensor with mDNS Service Discovery*

Command

```
avahi-browse -lrt [service type]
```

Example

```
username@ubuntu:~$ avahi-browse -lrt _roger._tcp
```

Response

```
+   eth0 IPv6 Ouster Sensor |sn|                    _roger._tcp          local
+   eth0 IPv4 Ouster Sensor |sn|                    _roger._tcp          local
=   eth0 IPv6 Ouster Sensor |sn|                    _roger._tcp          local
   hostname = [|os-sn|]
   address = [fe80::be0f:a7ff:fe00:1852]
   port = [7501]
   txt = ["fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn=99201000067
8" "pn=840-102145-B"]
=   eth0 IPv4 Ouster Sensor |sn|                    _roger._tcp          local
   hostname = [|os-sn|]
   address = [|sensor-ip|]
   port = [7501]
   txt = ["fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn= |sn|"
         "pn=840-102145-B"]
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`. If your sensor IPv4 address is of the form `169.254.x.x` it is connected via link-local.

9.4.5 Determining the IPv4 Address of the Interface

This will help you find the IPv4 address of the interface that you have plugged the sensor into. It is helpful to know which interface you have plugged into, e.g. `eth0` in the example below.

1. Open a Terminal window on the host machine by pressing **Ctrl+Alt+T**.
2. View the IPv4 address of your interfaces

Command

```
ip addr
```

Example

```
username@ubuntu:~$ ip addr
```

Response

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
       default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
```

(continues on next page)

(continued from previous page)

```
       default qlen 1000
    link/ether 00:0c:29:2b:cc:48 brd ff:ff:ff:ff:ff:ff
    inet |interface-ip|/16 brd 169.254.255.255 scope link noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::be9f:d2a4:4451:3dfe/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
       default qlen 1000
    link/ether 00:50:56:28:7a:8a brd ff:ff:ff:ff:ff:ff
    inet 192.0.2.232/24 brd 192.0.2.255 scope global wlan0
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:fe28:7a8a/64 scope link
       valid_lft forever preferred_lft forever
4: gpd0: <POINTOPOINT,MULTICAST,NOARP> mtu 1500 qdisc noop state DOWN group
       default qlen 500
    link/none
```

- In this example, your sensor is plugged into interface `eth0`.
- Your host IPv4 address will be on the line that starts with `inet`: In this case it is `169.254.0.1`.

Note: If your interface IPv4 address is of the form `169.254.x.x`, it is connected via link-local to the sensor. This means that the Linux self-assigned an IP address in the absence of a DHCP server.

9.4.6 Setting the Host Interface to DHCP

Use this to set your interface to automatically obtain an IP address via DHCP. This is useful for architectures that need to be more plug and play.

Note: It is recommended that you unplug the cable from the interface prior to making changes to the interface.

Via Command Line

Command

```
nmcli con modify [interface_name] ipv4.method auto ipv4.addresses ""
```

Example

with interface name `eth0`

```
username@ubuntu:~$ nmcli con modify eth0 ipv4.method auto ipv4.addresses ""
```

Response

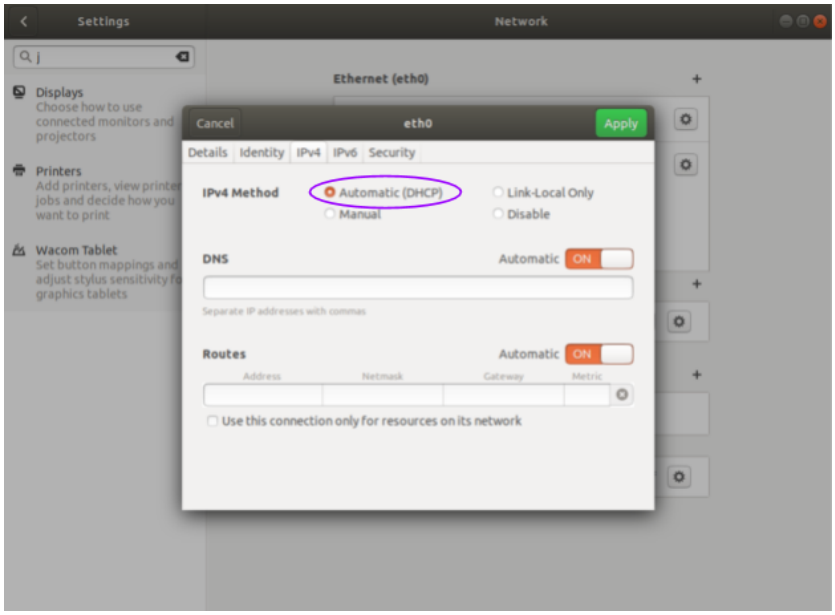
```
 blank
```

(continues on next page)

(continued from previous page)

```
 Note: However you can verify the change has been made with the ``ip addr`` command.
       There will be no ``inet`` line for the interface ``eth0`` until you plug in a cable
       to a device that has a DHCP server to provide an IPv4 address the interface

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
   default qlen 1000
   link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
   inet 127.0.0.1/8 scope host lo
   valid_lft forever preferred_lft forever
   inet6 ::1/128 scope host
   valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
   default qlen 1000
   link/ether 00:0c:29:2b:cc:48 brd ff:ff:ff:ff:ff:ff
   inet6 fe80::be9f:d2a4:4451:3dfe/64 scope link noprefixroute
   valid_lft forever preferred_lft forever
3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
   default qlen 1000
   link/ether 00:50:56:28:7a:8a brd ff:ff:ff:ff:ff:ff
   inet 172.16.79.232/24 brd 172.16.79.255 scope global wlan0
   valid_lft forever preferred_lft forever
   inet6 fe80::250:56ff:fe28:7a8a/64 scope link
   valid_lft forever preferred_lft forever
```

Via GUI The image below illustrates how to set the interface to `Automatic (DHCP)` mode using the graphical user interface.



9.4.7 Setting the Host Interface to Static IP

Use this to set your interface to be assigned a static IPv4 address. This is useful for controlling the IP address that the sensor will be sending data to.

Note: It is recommended that you unplug the cable from the interface prior to making changes to the interface.

Via Command Line

Command

```
nmcli con modify [interface_name] ipv4.method manual ipv4.addresses [ip_address]
```

Example

with interface name `eth0` and IPv4 address `192.0.2.1/24`.

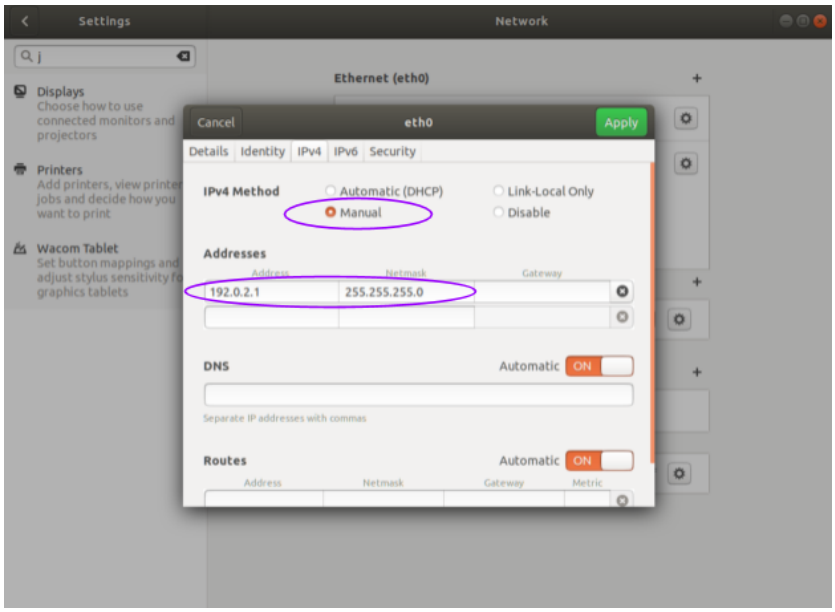
```
username@ubuntu:~$ nmcli con modify eth0 ipv4.method manual ipv4.addresses 192.0.2.1/24
```

Note: The `/24` is shorthand for Subnet Mask = `255.255.255.0`

Response

```
blank

Note: However you can verify the change has been made with the ``ip addr`` command.
      The ``inet`` line for the interface ``eth0`` will show the static IPv4 address. e.g. ``192.0.2.1``

   1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
          default qlen 1000
       link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
       inet 127.0.0.1/8 scope host lo
          valid_lft forever preferred_lft forever
       inet6 ::1/128 scope host
          valid_lft forever preferred_lft forever
   2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
          default qlen 1000
       link/ether 00:0c:29:2b:cc:48 brd ff:ff:ff:ff:ff:ff
       inet 192.0.2.1/24 brd 192.0.2.255 scope global noprefixroute eth0
          valid_lft forever preferred_lft forever
       inet6 fe80::be9f:d2a4:4451:3dfe/64 scope link noprefixroute
          valid_lft forever preferred_lft forever
   3: wlan0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
          default qlen 1000
       link/ether 00:50:56:28:7a:8a brd ff:ff:ff:ff:ff:ff
       inet 172.16.79.232/24 brd 172.16.79.255 scope global wlan0
          valid_lft forever preferred_lft forever
       inet6 fe80::250:56ff:fe28:7a8a/64 scope link
          valid_lft forever preferred_lft forever
```

Via GUI The image below illustrates how to set the interface to `Manual (static)` mode using the graphical user interface.



9.4.8 Finding a Sensor with mDNS Service Discovery

The sensor announces its presence on the network using Multicast Domain Name Service (mDNS) with a service type named `_roger._tcp`. You can use service discovery tools such as `avahi-browse` (Linux) to find all sensors connected to the network.

1. Find all sensors and their associated service text which includes the sensor IPv4 address using `avahi-browse` and the sensor service type `_roger._tcp`.

Command

```
avahi-browse -lrt [service type]
```

Example

```
username@ubuntu:~$ avahi-browse -lrt _roger._tcp
```

Response

```
+   eth0 IPv6 Ouster Sensor |sn|                  _roger._tcp          local
+   eth0 IPv4 Ouster Sensor |sn|                  _roger._tcp          local
=   eth0 IPv6 Ouster Sensor |sn|                  _roger._tcp          local
   hostname = [|os-sn|]
   address = [fe80::be0f:a7ff:fe00:1852]
   port = [7501]
   txt = ["fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn=99201000067
8" "pn=840-102145-B"]
=   eth0 IPv4 Ouster Sensor |sn|                  _roger._tcp          local
   hostname = [|os-sn|]
```

(continues on next page)

(continued from previous page)

```
  address = []
  port = [7501]
  txt = ["fw=ousteros-image-prod-aries-v2.0.0-20200417193957" "sn= |sn|"
         "pn=840-102145-B"]
```

Note: In this example, your sensor IPv4 address is determined to be `169.254.0.123`.

10 Perception API

Perception HTTPS API specification.

10.1 default

`GET /perception/api/v1/`

Root api endpoint

Status Codes

- 200 OK - OK

`GET /`

Root endpoint

Status Codes

- 200 OK - OK

`GET /perception/api/v1`

Root endpoint

Status Codes

- 200 OK - OK

10.2 Sensor Management

`GET /perception/api/v1/sensor`

Get list of Sensors

Get full list of sensors connected to perception pipeline

Status Codes

- 200 OK - OK

`DELETE /perception/api/v1/sensor`

Clears all lidars

Removes all live and pcap sensors from perception pipelines and sources. Effectively puts system back into original state with respect to added sensors. Note this does not affect extrinsics, that should be cleared separately

Status Codes

- 200 OK - OK

`GET /perception/api/v1/sensor/{status}`

Get list of active/inactive Sensors

Get list of sensors with given status ({active, inactive}) connected to perception pipeline.

Parameters

- `status` (`string`) -

Status Codes

- 200 OK - OK

`PUT /perception/api/v1/sensor/{hostname}`

Add a sensor

Add sensor to perception server

Parameters

- `hostname` (`string`) -

Status Codes

- 200 OK - OK

`DELETE /perception/api/v1/sensor/{sensor_id}`

Remove Sensor

Remove sensor by serial number

Parameters

- `sensor_id` (`string`) -

Status Codes

- 200 OK - OK

`PUT /perception/api/v1/pcap`

Add PCAP and sensor metadata for replay

Adds a PCAP and the associated lidar sensor metadata to the perception server for replay

Status Codes

- 200 OK - OK

10.3 Settings

`GET /perception/api/v1/settings`

Get all settings

Get all current settings

Status Codes

- 200 OK - OK

`POST /perception/api/v1/settings`

Set all settings.

Set entirely new settings file

Status Codes

- 200 OK - OK

`PUT /perception/api/v1/set_profile/{profile}`

Set the current profile

Set the current profile

Parameters

- `profile` (`string`) -

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`GET /perception/api/v1/profile/{profile}`

Get profile by name

Get profile by name

Parameters

- `profile` (`string`) -

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`PUT /perception/api/v1/profile/{profile}`

Add/Update settings profile

Add/Update settings profile

Parameters

- `profile` (`string`) -

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`DELETE /perception/api/v1/profile/{profile}`

Remove profile by name

Remove profile by name

Parameters

- `profile` (`string`) –

Status Codes

- 400 Bad Request – Bad Request
- 200 OK – OK

`GET /perception/api/v1/profiles`

Get list of profiles

Get list of profiles

Status Codes

- 400 Bad Request – Bad Request
- 200 OK – OK

`PUT /perception/api/v1/restore_profile/{profile}`

Restore profile to default values

Restore profile to default values

Parameters

- `profile` (`string`) –

Status Codes

- 400 Bad Request – Bad Request
- 200 OK – OK

10.4 Registration

`GET /perception/api/v1/extrinsics`

Access all extrinsics data.

The REST endpoint/path used to get extrinsics data.

Status Codes

- 200 OK – OK

`PUT /perception/api/v1/extrinsics`

Upload all extrinsics data.

The REST endpoint/path used to set all sensors extrinsic data.

Status Codes

- 200 OK – OK

`DELETE /perception/api/v1/extrinsics`

Clear all extrinsics data.

The REST endpoint/path used to clear all sensors extrinsic data.

Status Codes

- 200 OK - OK

`GET /perception/api/v1/extrinsics/{sensor_id}`

Access sensor extrinsics data.

The REST endpoint/path used to get extrinsics data for a given sensor_id

Parameters

- `sensor_id` (`integer`) -

Status Codes

- 200 OK - OK

`PUT /perception/api/v1/extrinsics/{sensor_id}`

Upload sensor extrinsics data.

The REST endpoint/path used to set extrinsics for a given sensor_id.

Parameters

- `sensor_id` (`integer`) -

Status Codes

- 200 OK - OK

`DELETE /perception/api/v1/extrinsics/{sensor_id}`

Remove sensor extrinsics data.

The REST endpoint/path used to remove extrinsics for a given sensor_id.

Parameters

- `sensor_id` (`integer`) -

Status Codes

- 200 OK - OK

`POST /perception/api/v1/extrinsics/icp`

Run ICP algorithm on point clouds

Status Codes

- 200 OK - OK

10.5 Execution

`PUT /perception/api/v1/execution/reset`

Stop perception application. Operating system will restart the application

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`POST /perception/api/v1/execution/play`

Play streaming of perception data

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`POST /perception/api/v1/execution/pause`

Pause perception data stream

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`POST /perception/api/v1/execution/step`

Step data stream

Step forward a single frame in the data stream.

Status Codes

- 200 OK - OK

`PUT /perception/api/v1/execution/start_recording/{filename}`

Start recording PCAP of all connected sensors with given filename

Parameters

- `filename` (`string`) -

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`GET /perception/api/v1/execution/stop_recording`

Stop recording PCAP

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`DELETE /perception/api/v1/execution/delete_recording/{filename}`

Delete PCAP with given filename

Parameters

- `filename` (`string`) –

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`GET /perception/api/v1/execution/list_recordings`

Get list of of all recordings in a directory

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

10.6 Point Zones

`GET /perception/api/v1/point_zones`

Get all zones

Get all zones

Status Codes

- 400 Bad Request - Bad Request
- 200 OK - OK

`PUT /perception/api/v1/point_zones`

Add zones

Replace the current point zones

Status Codes

- 200 OK - OK

`PUT /perception/api/v1/point_zones/{point_zone_id}`

Add zone

Add/update point zone with given point_zone_id

Parameters

- `point_zone_id` (`integer`) –

Status Codes

- 200 OK - OK

`DELETE /perception/api/v1/point_zones/{point_zone_id}`

Remove zone

Remove zone with given point_zone_id

Parameters

- `point_zone_id` (`integer`) –

Status Codes

- 200 OK – OK

10.7 Access

`PUT /perception/api/v1/password`

Set the login password for ouster user

Replace the current password for ouster user

Status Codes

- 200 OK – OK

10.8 Diagnostics

`GET /perception/api/v1/alerts`

Get alert data

Get alert data from the server.

Status Codes

- 200 OK – OK

`GET /perception/api/v1/alerts/active`

Get active alert data

Get active alert data from the server.

Status Codes

- 200 OK – OK

`GET /perception/api/v1/alerts/logged`

Get logged alert data

Get logged alert data from the server.

Status Codes

- 200 OK – OK

`GET /perception/api/v1/telemetry`

Get telemetry information

Get telemetry information from the server

Status Codes

- 200 OK – OK

`GET /perception/api/v1/configuration`

Get configuration information

Get all the configuration files from the server

Status Codes

- 200 OK - OK

10.9 Static

`GET /perception/api/v1/about`

Gets perception server's static 'about' information

Get static system information from the server

Status Codes

- 200 OK - OK

11 Appendix

11.1 Code Samples

The LidarHub has open TCP ports which stream the occupancy and object data. The user can listen to this data and create custom logic to fit their needs.

11.1.1 TCP Server Heartbeat Setup

The current TCP stream stops outputting data when there are no objects in the scene. When there are no objects in the scene for an extended period of time, the `recv()` call may timeout. It is recommended to configure a heartbeat message for the tcp_servers.

The heartbeat message only outputs when there are no objects present in the scene (when listening to the `object_list`), or when there are no objects present in any event zone (when listening to the `occupations`). The below LidarHub setting was used for the examples. Note that the `heartbeat_message` can be modified to a different message.

```
"tcp_servers": [
  {
    "source": "object_list",
    "port": 3302,
    "data_hertz": 1,
    "transmit_hertz": 1,
    "heartbeat_interval_secs": 1,
    "heartbeat_message": "{\"heartbeat\": [{}]}"
  },
```

11.1.2 Simple Example

The following examples assume that the heartbeat message is configured for the LidarHub TCP stream. The example assumes that a heartbeat message contains a key called `heartbeat`, and does not process those frames.

The following example demonstrates how to connect to a TCP stream, which are open on ports 3302 and 3303. By default, the `object_list` data is streamed on port 3302, and the `occupations` data is streamed on port 3303. The `object_list` output is the object count data for the whole scene, while the `occupations` data is the zone occupation data.

For this example, note that the listening port can be changed with the `PORT` variable. The `read_frames()` function takes in a `socket_client`, as well as a `Callable` function. The function is expected to take in the JSON data as the first parameter, and then additional parameters can be passed in the same call to `read_frames()`. The example below passes the python print function, which will print the JSON data from the TCP stream.

```
#!/usr/bin/env python
"""
Example how to connect to a Ouster Detect TCP stream. Assumes that the object_list
```

(continues on next page)

(continued from previous page)

```
is streaming on port 3302, and that the heartbeat is set to
"heartbeat_message": "{\"heartbeat\": [{}]}".
"""

import json
import socket
import ssl

from typing import Callable

# User defined variables. This is currently configured to listen to port
# 3302, which is by default is the `object_list` data.
HOST = "localhost"
PORT = 3302

# Ouster Detect defined variables
ENDIAN_TYPE = "big"
FRAME_SIZE_B = 4
ADDRESS = (HOST, PORT)


def recv(socket_client: ssl.SSLContext, num_bytes: int) -> bytearray:
    """
    Helper Function to recv n bytes or return an empty byte array if EOF is
    hit.

    Args:
        socket_client (ssl.SSLSocket): The socket connected to the TCP
        stream.
        num_bytes (int): The number of bytes to receive

    Returns:
        bytearray: The read bytes from the socket. Empty bytearray on
        timeout or connection reset.
    """
    data = bytearray()

    # It is possible only part of the message is read. We loop until we
    # received the whole message
    while len(data) < num_bytes:
        remaining_bytes = num_bytes - len(data)
        try:
            packet = socket_client.recv(remaining_bytes)

        # If the socket times out or is reset, no data could be received.
        except (socket.timeout, ConnectionResetError):
            return bytearray()

        # Append the data
        data.extend(packet)

    return data
```

(continues on next page)

(continued from previous page)

```
def read_frames(
    socket_client: ssl.SSLContext, callback_function: Callable, *args: tuple
) -> None:
    """
    Indefinitely reads in frames of data. The first 4 bytes of the message is
    expected to be the size of the message, and then that size will be read
    immediately afterwards. Repeats until connection is lost.

    Args:
        socket_client (ssl.SSLSocket): The socket connected to the TCP
        stream.
        callback_function (Callable): The callback function to call when
        receiving a valid set of data. The first parameter must be the
        JSON from the TCP stream, and the remaining arguments will be passed
        through args.
        args (tuple): The remaining arguments of the callback_function.
    """
    while True:
        # Gets the size of the frame
        frame_size_b = recv(socket_client, FRAME_SIZE_B)

        # If the size is different than expected, we didn't receive a response.
        # Return None, signalling either a failure to read the message, or that
        # there were no present objects
        if len(frame_size_b) == 0:
            return

        # Convert the byte data to an integer, representing the number of bytes
        # of the message. Then read that size of data from the stream
        frame_size = int.from_bytes(frame_size_b, ENDIAN_TYPE)
        data = recv(socket_client, frame_size)

        # Received no data, return None
        if len(data) == 0:
            return

        data = json.loads(data.decode("utf-8"))

        # If the dictionary contains "heartbeat" as a key, the message was a
        # heartbeat. Continue to the next message. Note that this is
        # configurable using the LidarHub settings.
        if "heartbeat" in data.keys():
            continue

        callback_function(data, *args)


# Create the ssl context
ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
ssl_context.verify_mode = ssl.CERT_NONE
# Create the socket client
socket_client = ssl_context.wrap_socket(socket.create_connection(ADDRESS))

# This reads the frame data indefinitely. The first parameter is the socket.
```

(continues on next page)

(continued from previous page)

```
# The next parameter is the callback function to pass the JSON data to.
# Additional parameters can be passed to be passed onto the function. The
# following line currently passes the JSON data to the print function. The
# following line can be updated with a custom callback function
read_frames(socket_client, print)

# Close the socket connection
socket_client.close()
```

11.1.3 Receiving objects from object_list

This is a continuation of the *Simple Example*. To connect to the `object_list`, the following code block can be appended to the beginning of the script, proceeding the imports. This parses the json data, and outputs the object's position if it matches the correct classification.

```
def print_objects_by_class(data: dict, classification: str) -> None:
    """
    Callback function to print out the timestamp and position of any objects
    that match the classification type. Must be listening to the object_list.

    Args:
        data (dict): Dictionary containing the batched object data.
        classification (str): Classification to print out for
    """
    # The data["object_list"] is a list containing the batched object_list data
    # for the frames. The number of batched object_lists is configurable in the
    # LidarHub settings using the `data_hertz` setting. See user manual for
    # more details of the structure. The following checks in read_frames() will
    # only enter this function if there is valid data. We use the first element
    # in the list.
    timestamp = data["object_list"][0]["timestamp"]

    for object_dict in data["object_list"][0]["objects"]:

        # Print out the object's position and its timestamp if the
        # classification is correct
        if object_dict["classification"] == classification:
            print(f"time: {timestamp}\tPosition: {object_dict['position']}")
```

The `read_frames()` line in the original *Simple Example* should be replaced with the following line, resulting in a print on any PERSON classified objects. Notice that `print_objects_by_class()` takes in two parameters, the first which must be the JSON data from the TCP stream, and the second being the classification. The classification parameter is passed to the `read_frames()` call, which forwards it to the `print_objects_by_class()` call. The following line prints for any objects that meet the "PERSON" classification.

```
read_frames(socket_client, print_objects_by_class, "PERSON")
```

11.1.4 Reading zone data from occupations

This is continuation of the *Simple Example*. The `occupations` by default outputs to port 3303. Change the `PORT` variable to 3303 (`PORT = 3303`) within the *Simple Example*.

To connect to the `occupations`, the following code block can be appended to the beginning of the script, and after the imports. This parses the json data, and prints the object count within a specified zone. The zone is identified using the `zone_name` parameter.

```
def print_occupation_count(data: dict, zone_name: str) -> None:
    """
    Callback function to print out the object count within a specific zone. The
    zone will be referenced by its name.

    Args:
        data (dict): Dictionary containing the batched object data.
        classification (str): Classification to print out for
    """
    queried_zone_data = {}

    # The data["occupations"] is a list containing the batched occupations data
    # for the frames. The number of batched occupations is configurable in the
    # LidarHub settings using the `data_hertz` setting. See user manual for
    # more details of the structure. The following checks in read_frames() will
    # only enter this function if there is valid data. We use the first element
    # in the list.

    # This finds the queried zone's data
    for zone_data in data["occupations"][0]["occupations"]:
        if zone_data["name"] == zone_name:
            queried_zone_data = zone_data

    # If the dictionary is empty, return
    if not queried_zone_data:
        return

    number_objects = len(queried_zone_data["objects"])
    print(f"Number of objects in {zone_name}: {number_objects}")
```

The `read_frames()` in the original *Simple Example* should be replaced with the following line, resulting in a print when there are any objects within the specified zone. Notice that `print_occupation_count()` takes in two parameters, the first which must be the JSON data from the TCP stream, and the second being the zone name. The zone name parameter is passed to the `read_frames()` call, which forwards it to the `print_occupation_count()` call. The below call uses `Zone-0` as the zone's identifying name.

```
read_frames(socket_client, print_occupation_count, "Zone-0")
```

11.2 Sensor Placement

The goal of sensor placement can vary depending on usage but typically involves maximizing the number of returns on objects of interest. This should take into account static obstacles, for example buildings, traffic light or furniture as well as dynamics obstacles such as vehicles or pedestrians.

For some use cases single sensors are appropriate but for others multiple sensors are required. Sensors should be placed to maximize the number of returns on objects of interest.

11.2.1 Tips for individual sensor placement

- Sensor field of view and maximum range should be taken into account. The maximum tracking range is less than the maximum range of the sensor due to a minimum number of point on targets in order to indentify it.
- Suggested tracking range for Rev6 sensors are. * OS-O: 25 meters * OS-1: 45 meters * OS-2: 60 meters
- Sensor must mounted on static position. Sensor movement will result in poor tracking. Typically mounting on poles or building works, but be carefull with traffic pole supporting arms because they may move in the wind.
- Ouside sensors should be placed higher that typical maximum vehicle heights (3m as a baseline). Higher sensor mounting height does result in wider blind spots around non dome sensor so multiple sensors may be required if a buffer zone around the sensor is unacceptable.
- Inside sensors should be place on ceilings or above obstacles if possible.
- Sensors may be angled or placed flat. Angled can get better coverage in one direction, but results in larger blind spots behind the sensor.
- Take into account the sensor FOV's. The FOV's of sensors are: * OS-0 += 45 degrees from horizontal * OS-1 +- 22.5 degrees from horizontal * OS-2 += 11.25 degrees from horizontal

11.2.2 Multi-Sensor Usage

Multiple sensors can help improve coverage in different monitoring situations:

- They can avoid blindspot, for example there is typically a blindspot under sensors and on multi-lane roads large vehicles may shield smaller vehicles from view.
- They also increase coverage area, either by being seperated geographicaly or angularily.
- Data from multiple sensors is fused so that if an object is seen by multiple sensors it will have improved accuracy.

11.2.3 Procedure for planning multi sensor locations

- Start with a map of floorplan of the area. A satellite view of the area is ideal if it is up to date because it includes locations of most obstacles.
- Figure out the priority areas of interest on the map. That may be a single area or the whole map. Draw those areas on the map.
- It is easiest to start by assuming the sensor are placed flat. Depending on the sensor height a X by X donut can be drawn on the map. Look for ideal placement locations with good views in many directions. These may vary depending on already placed poles, walls or other features.
    - For an OS-0, the circle should have an outer radius of ~25 m (max tracking range) and an inside radius of the sensor height.
    - For an OS-1, the circle should have an outer radius of ~45 M (max tracking range) and an inside radius of 2.4 times the sensor height.
    - For an OS-2, the circle should have an outer radius of ~60 M (max tracking range) and an inside radius of 5 times the sensor height.

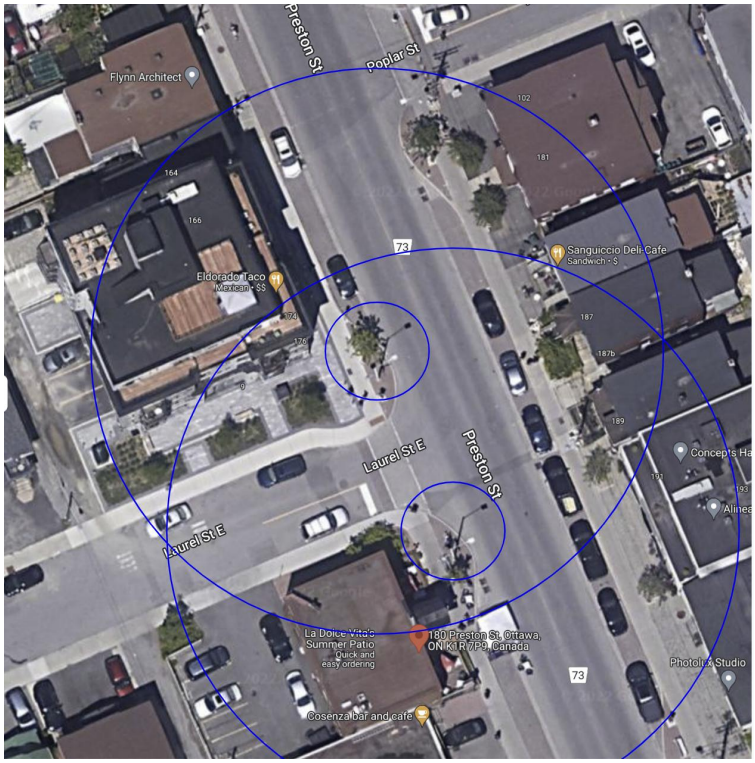


Figure 11.1: Example of sensor coverage on a map

This will give an initial ideal of what can be covered. Move the sensor around to try to figure out good placement. If you need continuous coverage of the whole area sensors will have to cover the blind spots of each other. If good coverage cannot be obtained with flat sensors they can be tilted. This will reduce the inner radius on the ground in one direction but increase it in the other.

11.3 Enabling PTP & Phase Lock

PTP is network time synchronization protocol. Enabling PTP on a Catalyst device will enable Ouster sensors to use Phase Lock configuration. Phase Lock will synchronize the phase of multiple Ouster sensors to improve tracking performance in high speed environments. To find more information regarding PTP, PMC and System Clock synchronization refer to the Ouster Firmware User Manual.

11.3.1 Enabling PTP on Catalyst as the Master Clock

1. SSH into the edgeproc as a user with superuser access.
2. update:

```
sudo apt update
```

3. Install dependencies:

```
sudo apt install linuxptp chrony ethtool
```

4. Identify the interfaces in which sensors are connected. PTP will need to be enabled on all these interfaces. (enp2s0f0 - enp2s0f3 on the Catalyst-Pro)
5. Open `/etc/linuxptp/ptp4l.conf` as sudo a. Insert each interface enclosed in [] at the END of the file b. If enabling more than one interface, the interfaces need to share 1 hardware clock. To do this, enable the boundary_clock_jbod c. Configure ptp4l of this edgeproc as the the Master Clock by changing clockClass to lower value

```
…
#clockClass 248 #comment out this line and insert below
clockClass 128
…
Boundary_clock_jbod 1 #leave at 0 if only 1 interface is required
…
[interface_1] #end of file
[interface_x]
```

6. Create override for ptp4l override file:

```
sudo mkdir -p /etc/systemd/system/ptp4l.service.d
```

7. Create the override file `/etc/systemd/system/ptp4l.service.d/override.conf`. This file ensures your modified ptp4l.conf is used by the systemd service.

```
[Service]
ExecStart=/usr/sbin/ptp4l -f /etc/linuxptp/ptp4l.conf
```

8. Reload and restart the systemd process with the changes. When querying the status there will be a reference to the override created in step 7.

```
sudo systemctl daemon-reload
sudo systemctl restart ptp4l
sudo systemctl status ptp4l
```

11.4 Enabling PTP & Phase Lock on Ouster Sensor

1. On each sensor configuration page change the *Timestamp Mode* under the *Timing* section to `time_from_ptp_1588`.
2. Also in the *Timing* section, select the `Phase Lock` radio button.
3. *Apply Config* and *Persist Active Config*.
4. Query the sensor's HTTP API to ensure PTP is operational. This command will return a JSON response.

```
curl -i http://{sensor-hostname}/api/v1/time/ptp
```

   - `parent_data_set.grandmaster_identity` should list the identity of the local grandmaster
   - `port_data_set.port_state` should be SLAVE
   - `time_status_np.gm_present` should be true
   - `time_status_np.master_offset` which is given in nanoseconds, should be less than 250000. This equates to 250 microseconds.

5. Query the sensor's HTTP API to ensure Phase Lock is operational. This command will return a JSON response.

```
curl -i http://{sensor-hostname}/api/vi/telemetry
```

   - `phase_lock_status` should be ENABLED

HTTP Routing Table

/

/perception