Ouster Project

Release 0.0.1

Ayman Magdy

Jul 01, 2024

CONTENTS:

CHAPTER

ONE

PROJECT DOCUMENTATION

1.1 introduction

The whole idea of this project is to make the best use of the Ouster Gemini, A new kind of core spatial intelligence software that accurately detects, classifies, and tracks people, objects, and vehicles. It is purpose built to address the critical need for highly accurate and dependable data, analytics, and live time tracking in the ITS, security, and crowd analytics industry. Through out this project the valubale data taken from the Gemini perception is received for furthur processing to make the most out of it that helps in creating more robust and convenient applications. The project can be said to be divided into to parts first the part which can deals with the *configuration* of the ouster Gemini as a software and second that deals with the *received stream data* which will be the core of the application we want to develop.

1.2 Configuration

In this part the given APIs from Ouster are used to set the required settings, as well most of the GUI functions can be made through out scripts. as GUI seemes to be a more convinient way to alter these settings but through the scripts it may be way to help in creating the required application, for more explanation about each set of APIs refere to *code documentation APIs* where the documentation and how to use information are given.

1. **Lidar_hub configuration -*Lidar Hub*-**
   deals with Sensor Management, Recording, Settings, Registration, Execution, Point Zones, Event Zones, Security, Diagnostics, Static.
2. **perception configuration -*perception*-**
   deals with the required settings for ouster detect software that sets the work flow of the perception as detection of objects and the perception id.
3. **Event Zones -*Event Zones*-**
   deals with zones and the events ocurring in each zone.
4. **Gemini data: -*Data Recording*-**
   deals with the output data of the gemini software
5. **gemini_login: -*Login*-**
   creates an authorized session for HTTPS requests, it is shared among the above modules to provide the base *URL* and authentication to send requests.

Note: Not all APIs are implemented some APIs are still under development.

Caution: The implemented APIs must be used with caution and response check is the user responsibility as all implementations returns only status code without error handling.

1.3 Recieved stream data

Here as will be refered to as *src* or *source code* is the main starting point for making use of the Ouster detect software data since the Ouster software stack outputs reliable and filtered data, this data can be furthur furnished to develop a well established application. Relying on TCP servers *data will be streamed*, *saved to a queue* and from the queue *data will be fetched for processing*. All this will be done concurently using threads.

the available JSON Streams.

- object_list
- occupations
- clusters
- diagnostics
- aggregation_realtime
- aggregation_timeseries

refere to user manual for more specifications.

1.4 Code flow and Execution

After the socket connection is established, Three threads are created

1. **put_to_queue_thr:**
   This thread is responsible for **READING** data from the SSL socket connection and putting it into `DATA_QUEUE` . It uses the `put_object_to_queue` function from the `processing.data_processing` module.
2. **read_from_queue_thr:**
   This thread is responsible for **RETRIEVING** data from `DATA_QUEUE` and making it available for processing.It uses the `get_from_queue` function from the `processing.data_processing` module.
3. **process_thr:**
   This thread is responsible for **PROCESSING** the data retrieved from the queue.It uses the `processing_from_queue` function from the `processing.data_processing` module.

This application is an example for the future work it was made to be generic as possible by using the approch of modularity. in this application the time of collision is calculated following the preceeding steps.

1. find the nearest two object to the sensor. (an information retrieved from the object list recieved from the stream)
2. find the relative velocity magnitude between the same two objects. (the same time stamp is taken in consideratin to make sure the specific required object is found)
3. find the relative position magnitude between the same two objects.
4. estimate the time of collision between the objects.

Note: assumptions are made that both objects are heading towards the sensor and the written code is just a test for what may come later.

1.5 Data shape

Data streamed refered to as `object_list` from gemini is received in JSON format which may be difficult to make furthur processing, so it was better to reorder the data depending on the timestamp in a python dictionary with the time stamp as the key and the value will be the object it self. refere below to see what the JSON format looks like.

1.5.1 shape of received data from stream


```
{
  "frame_count": 226599,
  "timestamp": 1663175920089867
"objects": [
{
  "classification": "PERSON",
  "classification_confidence": 0,
  "creation_ts": 1663175901389312,
  "dimensions": {
    "height": 0.6693140268325806,
    "length": 0.44045835733413696,
    "width": 0.24217760562896729
  },
  "distance_to_primary_sensor": 1.8241156339645386,
  "frame_count": 188,
  "heading": 17.792495727539062,
  "id": 1094,
  "initial_position": {
    "x": 0.866864025592804,
    "y": -1.3293535709381104,
    "z": 1.1521248817443848
  },
  "num_points": 1052,
  "num_points_from_primary_sensor": 706,
  "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,
```


(continues on next page)

(continued from previous page)


```
    "y": 0.00872983346207417,
    "z": 0.03328978381826185
  },
  "primary_sensor": "992251000353",
  "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
  }
},
{
  "classification": "UNKNOWN",
  "classification_confidence": 0,
  "creation_ts": 1663175902884161,
  "dimensions": {
    "height": 0.5432571172714233,
    "length": 0.48804545402526855,
    "width": 0.20569449663162231
  },
  "distance_to_primary_sensor": 1.8241156339645386,
  "frame_count": 171,
  "heading": -89.16230010986328,
  "id": 1096,
  "initial_position": {
    "x": 0.9158645868301392,
    "y": -1.8394945859909058,
    "z": 0.7948745489120483
  },
  "num_points": 539,
  "num_points_from_primary_sensor": 706,
  "orientation": {
    "qw": 0.7122570276260376,
    "qx": 0,
    "qy": 0,
    "qz": -0.7019187808036804
  },
  "position": {
    "x": 0.7764403223991394,
    "y": -1.7604234218597412,
    "z": 0.9274996519088745
  },
  "position_uncertainty": {
    "x": 0.00872983346207417,
    "y": 0.00872983346207417,
    "z": 0.03328978381826185
```


(continues on next page)

(continued from previous page)


```
  },
  "primary_sensor": "992251000353",
  "update_ts": 1663175920076580,
  "uuid": "b3caf8eb-8e23-47fe-a839-999f354ae4a0",
  "velocity": {
    "x": -0.10920844900324483,
    "y": 0.31853616628659553,
    "z": 0.10758132742665225
  },
  "velocity_uncertainty": {
    "x": 0.7745966692414834,
    "y": 0.7745966692414834,
    "z": 0.8143665760550121
  }
}
]
}
```


Table 1: object_list description

Field	Format	Definition
frame_count	int	Number of frames since perception started outputting data. This count should always be sequential.
timestamp	int	Timestamp of when the last point cloud arrived which contributed to the object list. Units in microseconds since Jan. 1, 1970.
objects	array	Array of *objects* visible to perception for the current frame

Table 2: object_array description

Field	Format	Definition
id	int	Unique number identifying object in current running instance of perception. If perception restarts, this count will reset.
uuid	string	Unique 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.
primary_senso	string	Serial number of the primary lidar sensor with the most points on object.
distance_to_pri	float	Distance in meters from the primary lidar sensor.
num_points_	int	Number of points on object from the primary lidar sensor.
num_points	int	Number of points belonging to the object.
classification	string	Classification of the object (i.e., PERSON, VEHICLE).
classification_confide	float	Number between 0 and 1 representing the system's confidence in the assigned classification. 0 represents no confidence. 1 represents fully confident.
initial_position	vector	Initial XYZ location of the object in the first frame it was visible in the field of view of the lidars. This position is either in respect to the world reference frame (measured in meters), or the geo-coordinates.
position	vector	Current XYZ location of the object in the world reference frame (measured in meters), or the geo-coordinates.
position_uncerta	vector	Estimated variance of the position measurement. Units in meters^2.
velocity	vector	current rate of change in the XYZ position of the object. Velocity is in the world reference frame. Units are in m/s.
velocity_uncertai	vector	Estimated variance of the velocity measurement. Units in (m/s)^2.
orientation	orientation	Quaternion representing the orientation of the object with respect to the world reference frame.
heading	float	Positive rotation about the z-axis (right-hand rule).
dimension	dimension	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).
frame_count	int	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.
creation_ts	int	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.
update_ts	int	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.

The previous *code snippet* shows the shape of the streamed data followed by its explanation in the tables. the code refines the data streamed and puts it in a python dictionary form as mentioned above. the global variable `TIMESTAMP` defined in `processing.data_processing` *module* is the variable that will hold the refined data. it consists of *key* as the *timestamp* and *dictionary values* where their *key* are the *name of the attribute* and the value *are lists of the data*. refere below for explanation

```
{
1701174658528965: {
'obj_id': [11111, 22222, 33333, 44444],
'creation_time': [1701174621516041, 1701174621516041, 1701174621516041,␣
↪1701174621516041],
'frame_count': [343, 343, 343, 343],
'classification': ['PERSON', 'PERSON', 'PERSON', 'PERSON'],
'velocity': [{'x': 11, 'y': 11, 'z': 11}, {'x': 22, 'y': 22, 'z': 22}, {'x': 33, 'y': 33,
↪ 'z': 33}, {'x': 44, 'y': 44, 'z': 44}],
'position': [{'x': 11, 'y': 11, 'z': 11}, {'x': -22, 'y': 22, 'z': 228}, {'x': -33, 'y':␣
↪33, 'z': 33}, {'x': -44, 'y': 44, 'z': 44}],
'heading':  [11, -37.560306549072266, -37.560306549072266, -37.44]
}
}
```

Tip: every object is known by its obj_id and to get the related information of that specific obj_id use indexing. for example in the above data shape. `obj_id` 11111 will have index 0 then to locate its velocity for example index `velocity` with index 0.

```
print(timestamp[1701174658528965]['velocity'][0])

>>> {'x': 11, 'y': 11, 'z': 11}
```

this made the data more refined and every time stamp is known what objects are captured within it with each information related to that object.

Note: for calculations this data is converted to pandas DataFrame for the ease of its calculations. refere to the helper methods at *processing.utils module*

1.6 Data integrity

To make sure that each object has a unique ID we have to make sure that the sensors installed are all in the same perception. The ouster detect software stack gives each object a unique ID `uuid` once it is detected in the perception if the object lost from the perception and enters again the ouster detect will give it a new `uuid` so its our responsibility to make sure that our interested area is fully covered and the sensors area of coverage are well installed to make sure that if an object never leaves our area of interest. another consideration while receiveing through the stream to make sure that we don't loose any data or we get an interrupted message ouster software stack sends a 32-bit unsigned integer with big endian byte order which represents the total length of the message by checking on these 4-bytes we know the size of the expected data to receive. see `put_object_to_queue` method at *processing.data_processing module*

CHAPTER

TWO

REFRENCES

The source code can be found at the github repository .

The HTML page of this documentation can be found at readthedocs link.

CHAPTER

THREE

CODE DOCUMENTATION

3.1 sorce code

3.1.1 main

main.**main**()

the starting function of the project. this function defines two types of loggers to be used in dignosing system healt first logger for information where it loggs the normal operation of the system creation of queue size of queue and makes the system more friendly. second logger for warnings such as no data present, connection failed and other messages which needs action.

3.1.2 connection package

SSL/TLS Socket Conection

The method *connect_to_ssl_socket* is called to establish an SSL socket connection. This function atemps to create a secure connection using the TLSv1.2 protocol. If the connection is successful, an informational message is logged stating that the connection has been establishe

connection.socket_connection module

connection.socket_connection.**ADDRESS = ('127.0.1.1', 3302)**

tuple:
A global constant to show the host and port to creat socket connection to it

connection.socket_connection.**HOST = '127.0.1.1'**

Defines the host requiresd to connect which is local host by default.

connection.socket_connection.**PORT_OBJECT_LIST = 3302**

Defines the port to listen to during receiving 3302 by default which is the object_list on tcp_server.

connection.socket_connection.**connect_to_ssl_socket**(*address: tuple[str | None, int] = ('127.0.1.1', 3302)*) → SSLSocket

Description

this function creats a socket connection and gives it a SSL wrap.

Args

address: tuple
the host and the port needed to connect to. ('local.host,3302' default).

Returns

ssl.SSLSocket
a socket connection with SSL/TLS wrap.

3.1.3 utilities package

Submodules

utilities.utilities module

utilities.utilities.**get_object_list_arr**(*data: dict*, *stamp: int*, *key: Literal['id', 'uuid', 'classification', 'classification_confidence', 'creation_ts', 'update_ts', 'dimensions', 'frame_count', 'heading', 'initial_position', 'num_points', 'position', 'position_uncertainty', 'velocity', 'velocity_uncertainty', 'distance_to_primary_sensor']*) → list

Description:
This function is used to fetch in the object data of the recieved object list.

Args:
data: which is the recieved object list. key: which is the desired information or data we want refere to "object_key: TypeAlias"

Returns:
list of the fetched data ex: id, velocity, position, distance_to_primary_sensor

utilities.utilities.**get_root_info**(*data: dict*, *key: Literal['frame_count', 'timestamp', 'objects']*) → list

Description:

This function is used to fetch for the root data of the recieved object.

Args:
data: which is the recieved object. key: which is the desired information or data (rootinfo_key: TypeAlias)

Returns:
list of the fetched data ex: frame_count, timestamp

we use the TypeAlisa (rootinfo_key: TypeAlias)

utilities.utilities.**object_key**

Description:
this is a type alias used to refere to the object to be selected or the object of interest from the recieved object list created by the ouster detect software.

Usage:
it is used as a parameters sent to the function in this module to search in the array of the recieved object list.

alias of `Literal`['id', 'uuid', 'classification', 'classification_confidence', 'creation_ts', 'update_ts', 'dimensions', 'frame_count', 'heading', 'initial_position', 'num_points', 'position', 'position_uncertainty', 'velocity', 'velocity_uncertainty', 'distance_to_primary_sensor']

utilities.utilities.**recv_stream**(*ssl_socket_client: SSLSocket*, *num_bytes: int*) → bytearray

Description:
recieves directly form the socket listens to the server and recieve.

Args:
ssl_socket_client for the connection of the server num_byte number of bytes to be recieved

Returns:
bytearray

utilities.utilities.**rootinfo_key**

Description:
this is a type alias used to refere to the object to be selected or the object of interest from the recieved object list created by the ouster detect software.

Usage:
it is used as a parameters sent to the function in this module to search in the array of the recieved object list.

alias of `Literal`['frame_count', 'timestamp', 'objects']

3.1.4 processing package

Submodules

processing.data_processing processing.utils

processing.data_processing module

description

contains methods related to dealing with the queues.

processing.data_processing.**TIMESTAMP = {}**

global dictionary variable used to hold the data after being refined from the queue key is the *timestamp* values are

processing.data_processing.**get_from_queue**(*data_queue: Queue*, *lock: allocate_lock*, *q_ready_event: Event*, *d_ready_event: Event*) → None

Description:
this function checks if the queue is not empty then reads the time stamp of the received frame from queue time stamp is saved in TIMESTAMP global dictionary as the key the values are dictionaries of parameter name as key and values of the data values.

Example:
{1701174658528965: { 'obj_id': [11111, 22222, 33333, 44444]},'position': [111, 222, 333, 4444]}.

Args:
queue to get timestamp and other data from

Returns:
None

processing.data_processing.**processing_from_queue**(*lock: allocate_lock*, *d_ready_event: Event*) → None

Description:
This function is meant to calculate furthur processes after refining the data from stream. first convert the TIMESTAMP to `pd.DataFrame` of the related attribute ex: posdf, veldf position data frame and velcoity data frame respectevily. then call helper functions for calculations.

Args:
lock to protect shared resources d_read_event waits for the signal when data is ready to be processed then thread starts.

Returns:
None

processing.data_processing.**put_object_to_queue**(*data_queue: Queue*, *ssl_socket_client: SSLSocket*, *q_ready_event: Event*) → None

Description:
read data from stream and put in queue this function doesn't end until queue is full or there is not data recieved from the stream (byte array is empty).

Args:
socket connection with ssl wrap. data_queue to put received data in.

Returns:
None.

processing.utils module

description

contains helper methods used in data_processing module.

processing.utils.**calc_pos_mag_diff**(*dataframe: DataFrame*, *stamp: int*, *index1: int*, *index2: int*) → floating

Description:
a helper function to be called to calculate the position difference between two rows from the passed dataframe.

Args:
dataframe: positon dataframe to get the difference. stamp: the required time stamp. index1: the first index/row in the dataframe required. index2: the second index/row in the dataframe required.

Returns:
pandas series of the calculated position norm between index1 and index2.

processing.utils.**calc_vel_mag_diff**(*dataframe: DataFrame*, *stamp: int*, *index1: int*, *index2: int*) → floating

Description:
a helper function to be called to calculate the velocity norm between two objects at a specific time stamp.

Args:
dataframe: velocity dataframe to get the difference. stamp: the required time stamp. index1: the first index/row in the dataframe required. index2: the second index/row in the dataframe required.

Returns:
pandas series of the calculated velocity norm between index1 and index2

processing.utils.**get_dis_from_sensor**(*data_dict: dict*) → DataFrame

Description:
a helper funtion to iterate over the the time stamps being recieved from stream and converts the specified element (distance from sensor) to a dataframe

Args:

data_dict
the recieved object list after time stamped

Returns:
pandas DataFrame

processing.utils.**get_hed_df**(*data_dict: dict*) → DataFrame

Description:
a helper funtion to iterate over the the time stamps being recieved from stream and converts the specified element (heading) to a dataframe

Args:

data_dict
the recieved object list after time stamped

Returns:
pandas DataFrame

processing.utils.**get_nearest_from_sensor**(*dataframe: DataFrame*, *col_stamp: str*) → Series

Description:
this function find the nearest two objects from the senosr.

Args:

dataframe
the dataframe related to the distance from the sensor.

stamp
the time stamp or the column in the data frame to search in it.

Returns:
series with the values of the distances

processing.utils.**get_pos_df**(*data_dict: dict*) → DataFrame

Description:
a helper funtion to iterate over the the time stamps being recieved from stream and converts the specified element (position) to a dataframe

Args:
data_dict: the recieved object list after time stamped

Returns:
pd.DataFrame

processing.utils.**get_vel_df**(*data_dict: dict*) → DataFrame

Description:
a helper funtion to iterate over the the time stamps being recieved from stream and converts the specified element (velocity) to a dataframe

Args:

data_dict
the recieved object list after time stamped

Returns:
pandas DataFrame

processing.utils.**time_to_col**(*vel: floating*, *pos: floating*) → floating

Description: calculates the time expected for two objects to collide.

3.2 APIs

3.2.1 Login

Description

The module is used to set up an authorized session for HTTPS request. The module uses `HOST` located at *connection package* which is by **default** the *local host* as the base `URL` for the connection. The module as well defines **the AUTHENTICATION credentials** which may be required to be changed if the user changes them after his/her first login.

Caution: the default values for user name and password are:

- `USERNAME` : ouster
- `PASSWORD` : stone-pass-fill

gemini_login.**URL = 'https://127.0.1.1/'**

It is the base url used through all the application.

gemini_login.**login_ouster**() → Session

Description

creates a session between host and server through HTTPS.

ARGs

None.

Return

requests.Session
authorized session for HTTPS requests.

3.2.2 Lidar Hub

Description

The module is used to set/get the setting of the Lidar Hub . Lidar Hub works as the interface between the ouster detect software and the user. it integrates other features as well.

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

The module uses URL located at *Login* as the base url.

Note: for more information on the module and for the information about the fields that can be altered in the JSON files being sent as request body please refere to the user manual.

---

enum gemini_lidar_hub_API.**LidarHubEndPoint**(*value*)

Bases: Enum

defines the endpoints of the url for more information refer to ouster swagger api documentation

Valid values are as follows:

STATIC_INFO = <LidarHubEndPoint.STATIC_INFO: 'lidar-hub/api/v1/about'>

SYSTEM_DIAGNOSTICS = <LidarHubEndPoint.SYSTEM_DIAGNOSTICS:
'lidar-hub/api/v1/diagnostics'>

SYSTEM_HEALTH = <LidarHubEndPoint.SYSTEM_HEALTH: 'lidar-hub/api/v1/system_health'>

LIDAR_TELEMETRY = <LidarHubEndPoint.LIDAR_TELEMETRY: 'lidar-hub/api/v1/telemetry'>

LIDAR_HUB_DEFAULTS = <LidarHubEndPoint.LIDAR_HUB_DEFAULTS:
'lidar-hub/api/v1/default_settings'>

LIDAR_HUB_SETTINGS = <LidarHubEndPoint.LIDAR_HUB_SETTINGS:
'lidar-hub/api/v1/settings'>

LIDAR_HUB_WORLD = <LidarHubEndPoint.LIDAR_HUB_WORLD: 'lidar-hub/api/v1/world'>

LIDAR_HUB_EVENT_ZONES = <LidarHubEndPoint.LIDAR_HUB_EVENT_ZONES:
'lidar-hub/api/v1/event_zones'>

LIDAR_HUB_EVENT_ZONES_ACTIVE = <LidarHubEndPoint.LIDAR_HUB_EVENT_ZONES_ACTIVE:
'lidar-hub/api/v1/event_zones/active'>

LIDAR_HUB_EVENT_ZONES_ALERTS = <LidarHubEndPoint.LIDAR_HUB_EVENT_ZONES_ALERTS:
'lidar-hub/api/v1/event_zones/alerts'>

LIDAR_HUB_EVENT_ZONES_REALTME = <LidarHubEndPoint.LIDAR_HUB_EVENT_ZONES_REALTME:
'lidar-hub/api/v1/event_zones/realtime'>

LIDAR_HUB_EVENT_ZONES_TIMESERIES =
<LidarHubEndPoint.LIDAR_HUB_EVENT_ZONES_TIMESERIES:
'lidar-hub/api/v1/event_zones/timeseries'>

LIDAR_HUB_RESET = <LidarHubEndPoint.LIDAR_HUB_RESET: 'lidar-hub/api/v1/reset'>

LIDAR_HUB_ACTIVE_REC = <LidarHubEndPoint.LIDAR_HUB_ACTIVE_REC:
'lidar-hub/api/v1/user_recording/active'>

LIDAR_HUB_START_REC = <LidarHubEndPoint.LIDAR_HUB_START_REC:
'lidar-hub/api/v1/user_recording/start'>

LIDAR_HUB_STOP_REC = <LidarHubEndPoint.LIDAR_HUB_STOP_REC:
'lidar-hub/api/v1/user_recording/stop'>

gemini_lidar_hub_API.**get_lidar_hub_active_rec**(*session: Session*) → Response

Description

Get active user recordings.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_all_settings**(*session: Session*) → Response

Description

Get all application settings.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_default_settings**(*session: Session*) → Response

Description

Get the application default values

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_diagnostics**(*session: Session*) → Response

Description

gets the diagnostics hub.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_event_active**(*session: Session*, *ids: str = '0'*, *classification: str = '0'*, *min_dwell_secs: float = 0*) → Response

Description

Query Active Occupancy by ID, Classification and Dwell

Args

session: requests.Session
a request session for HTTP requests.

ids: str
even zode ID (default = 0: ALL)

classification: str
classification class required (default = 0: ALL)

min_dwell_secs: float
minimum dwell in seconds (default = 0.0)

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_event_alerts**(*session: Session*) → Response

Description

Get the alerts for all evnet zones.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_event_realtime**(*session: Session*) → Response

Description

Get real time occupancy of all event zones.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_event_timeseries**(*session: Session*) → Response

Description

Get time series for all event zones.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_event_zones**(*session: Session*) → Response

Description

Get the evnet zones in the perception.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_health**(*session: Session*) → Response

Description

gets the system health.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_info**(*session: Session*) → Response

Description

gets the static information about the lidar hub.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_telemetry**(*session: Session*) → Response

Description

Get Lidar Hub 'telemetry' information.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**get_lidar_hub_world_coordinates**(*session: Session*) → Response

Description

Get the Geo-coordinates.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**reset_lidar_hub**(*session: Session*) → Response

Description

restarts the lidar hub.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**reset_lidar_hub_default_settings**(*session: Session*) → Response

Description

restores the default values.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**set_lidar_hub_all_settings**(*session: Session*, *data_file: str*) → Response

Description

setts all application to the desired settings.

Args

session: requests.Session
a request session for HTTP requests.

data_file: str
the path to the settings file in **JSON** format.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**set_lidar_hub_world_coordinates**(*session: Session*, *data_file: str*) → Response

Description

sets the world Geo-coordinate

Args

session: requests.Session
a request session for HTTP requests.

data_file: str
a file required to read the new settings in **JSON** format.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**start_lidar_hub_rec**(*session: Session*, *body_path: str*) → Response

Description

starts a user reconrding.

Args

session: requests.Session
a request session for HTTP requests.

body_path: str
the path to the request body file in **json** format.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_lidar_hub_API.**stop_lidar_hub_rec**(*session: Session*, *id: str*) → Response

Description

stops a user recording.

Args

session: requests.Session
a request session for HTTP requests.

id: str
query for the id for the required recording to be stopped.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

3.2.3 perception

Description

The module is used to set/get the setting of the perception. The module uses URL located at *Login* as the base url.

Note: for more information on the module and for the information about the fields that can be altered in the JSON files being sent as request body please refere to the user manual.

---

enum gemini_perception_API.**PerceptionEndPoint**(*value*)

Bases: Enum

defines the endpoints of the url for more information refer to ouster swagger api documentation

Valid values are as follows:

LIST_OF_ALL_SENSORS = <PerceptionEndPoint.LIST_OF_ALL_SENSORS: 'perception/api/v1/sensor/'>

LOAD_BACKGROUND = <PerceptionEndPoint.LOAD_BACKGROUND: 'perception/api/v1/background'>

SETTINGS = <PerceptionEndPoint.SETTINGS: 'perception/api/v1/settings/'>

UNDERLAY_MAP = <PerceptionEndPoint.UNDERLAY_MAP: 'perception/api/v1/underlay/'>

UNDERLAY_CONFIG = <PerceptionEndPoint.UNDERLAY_CONFIG: 'perception/api/v1/underlay_config/'>

SET_PROFILE = <PerceptionEndPoint.SET_PROFILE: 'perception/api/v1/set_profile/'>

GET_PROFILE = <PerceptionEndPoint.GET_PROFILE: 'perception/api/v1/profile/'>

DEFAULT_PROFILE = <PerceptionEndPoint.DEFAULT_PROFILE: 'perception/api/v1/profile_defaults/'>

LIST_PROFILES = <PerceptionEndPoint.LIST_PROFILES: 'perception/api/v1/profiles/'>

RESTORE_PROFILE = <PerceptionEndPoint.RESTORE_PROFILE: 'perception/api/v1/restore_profile/'>

IMU_POSE = <PerceptionEndPoint.IMU_POSE: 'Perception/api/v1/imu_pose/'>

ALL_EXTRINSICS = <PerceptionEndPoint.ALL_EXTRINSICS: 'perception/api/v1/extrinsics/'>

ICP_ALGORITHM = <PerceptionEndPoint.ICP_ALGORITHM: 'perception/api/v1/extrinsics/icp/'>

gemini_perception_API.**add_perception_profile**(*session: Session*, *body_path: str*, *profile_name: str*) → Response

Description

add/update settings profile.

Args

session: requests.Session
a request session for HTTP requests.

body_path: str
the path for the file with the requited settings.

profile_name: str
the name of the added profile.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**add_senor**(*session: Session*, *host_name: str*) → Response

Description

adds a new sensor to perception.

Args

session: requests.Session
a request session for HTTP requests.

host_name: str
the address of the sensor to be added.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**clear_all_lidars**(*session: Session*) → Response

Description

removes all lice sensors from perception.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**get_all_sensors**(*session: Session*) → Response

Description

gets full list of sensors connected to perception pipeline.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**get_perception_profile_by_name**(*session: Session*, *profile_name: str*) → Response

Description

get the settings for the profile_name.

Args

session: requests.Session
a request session for HTTP requests.

profile_name: str
profile name.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**get_perception_profile_defaults**(*session: Session*, *profile_name: str*)

gemini_perception_API.**get_perception_profile_list**(*session: Session*) → Response

Description

get all list of profiles.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**get_perception_settings**(*session: Session*) → Response

Description

get all current settings.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**get_sensor_by_status**(*session: Session*, *status: str*) → Response

Description

gets list of sensors with the passed status query.

Args

session: requests.Session
a request session for HTTP requests.

status: str
active or inactive

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**get_underlay_config**(*session: Session*) → Response

gemini_perception_API.**get_underlay_map**(*session: Session*) → Response

gemini_perception_API.**load_backgrounds**(*session: Session*) → Response

Description

Load all current background available for all sensors.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**remove_perception_profile**(*session: Session*, *profile_name: str*) → Response

Description

delete settings profile.

Args

session: requests.Session
a request session for HTTP requests.

profile_name: str
the name of the needed profile.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**remove_sensor**(*session: Session*, *sensor_id: str*) → Response

Description

removes the sensor with the passed query sensor_id.

Args

session: requests.Session
a request session for HTTP requests.

sensor_id: str
required senosr serial number to be removed.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**restore_to_defaults**(*session: Session*, *profile_name: str*)

Description

restores the profile to default values.

Args

session: requests.Session
a request session for HTTP requests.

profile_name: str
the name of the added profile.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**save_backgrounds**(*session: Session*) → Response

Description

Save background for all sensors.

Args

session: requests.Session
a request session for HTTP requests.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**set_perception_current_profile**(*session: Session*, *profile_name: str*) → Response

Description

sets the current prfile by name

Args

session: requests.Session
a request session for HTTP requests.

profile_name: str
the required profile name to be set.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**set_perception_setting**(*session: Session*, *body_path: str*) → Response

Description

sets all perception settings.

Args

session: requests.Session
a request session for HTTP requests.

body_path: str
path to the file with the required settings in **JSON** format.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_perception_API.**set_underlay_config**(*session: Session*, *body_path: str*) → Response

gemini_perception_API.**set_underlay_map**(*session: Session*, *body_path*) → Response

3.2.4 Event Zones

Description

The module is used to set/get the setting of event zones and add/delete zones.

The module uses URL located at *Login* as the base url.

Note: for more information on the module and for the information about the fields that can be altered in the JSON files being sent as request body please refere to the user manual.

---

enum gemini_event_zone_API.**EventEndPoint**(*value*)

Bases: Enum

defines the endpoints of the url for more information refer to ouster swagger api documentation

Valid values are as follows:

GET_ALL_EVENTS = <EventEndPoint.GET_ALL_EVENTS: 'event/api/v1/event_zones/'>

REMOVE_ZONE_ID = <EventEndPoint.REMOVE_ZONE_ID: 'event/api/v1/event_zones'>

DIAGNOSTICS = <EventEndPoint.DIAGNOSTICS: 'event/api/v1/telemetry'>

STATIC_INFO = <EventEndPoint.STATIC_INFO: '/event/api/v1/about'>

gemini_event_zone_API.**add_event_zone_zones**(*session: Session*, *data: str*) → Response

Description

Add multiple zones at once

Args

session: requests.Session
a request session for HTTP requests.

data: str
the path to the file containing information of the zones to be created.

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_event_zone_API.**add_zone_id**(*session: Session*, *zone_id: str*, *data: str*)

Description

Adds a unique zone with an ID.

Args

session: requests.Session
a request session for HTTP requests.

zone_id: str
the ID of the zone to be added

data: str
the file path of the information of the added zone in *json* format

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_event_zone_API.**get_event_zone_all_zones**(*session: Session*, *file_type: str = 'txt'*) → Response

Description

gets all event zones in the perception.

Args

session: requests.Session
a request session for HTTP requests.

file_type: str
select the required file type to write the response to .txt, .json. (default .txt)

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

gemini_event_zone_API.**get_event_zone_telemetry**(*session: Session*, *file_type: str = 'txt'*) → Response

Description

writes telemetry information from the server to afile.

Args

session: requests.Session
a request session for HTTP requests.

file_type: str
select the required file type to write the response to .txt, .json. (default .txt)

Returns

requests.Response
Response object, which contains a server's response to an HTTP request.

3.2.5 Data Recording

Description

used to retreive the recorded data from the ouster perception.

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 will be recorded into a combined file. By default, recordings are saved to /opt/ouster/conf/data/recordings/.

The Lidar Hub includes 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 an id of 0

The module uses URL located at *Login* as the base url.

Note: for more information on the module and for the information about the fields that can be altered in the JSON files being sent as request body please refere to the user manual.

refere to the module *Lidar Hub* to know how to set the fields to start and define the desired data for recording.

---

enum gemini_data_recording.**DataEndPoint**(*value*)

Bases: Enum

defines the endpoints of the url for more information refer to ouster swagger api documentation

Valid values are as follows:

JSON_BINARY_DATA_ENDPOINT = <DataEndPoint.JSON_BINARY_DATA_ENDPOINT: 'data/recordings/'>

AGGREGATION_DATA_ENDPOINT = <DataEndPoint.AGGREGATION_DATA_ENDPOINT: 'data/storage/'>

gemini_data_recording.**aggregation_data**(*session: Session*) → None

Description

Function to get aggregation data Fetch '/data/storage' save file to aggregation.txt. Download files and write it to 'aggregation_total_visitors.json' 'aggregation_total_visits.json

ARGs

session: requests.Session
an authorized HTTPS connection.

Return

None

gemini_data_recording.**json_binary_data**(*session: Session*) → None

Description

Function to get json data Writes saved files to data_rec_links.txt Open data_rec_links.txt and fetch for the required link to download. Write objects to data_rec_objects.json file.

ARGs

session: requests.Session
an authorized HTTPS connection.

Return

None

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

O

P

R

S

T

U