datasetinsights

Release 1.0.0

Unity Technologies

Feb 28, 2022

CONTENTS

Unity Dataset Insights is a python package for downloading, parsing and analyzing synthetic datasets generated using the Unity Perception SDK.

CHAPTER

ONE

INSTALLATION

Dataset Insights maintains a pip package for easy installation. It can work in any standard Python environment using `pip install datasetinsights` command. We support Python 3 (3.7 and 3.8).

CHAPTER

TWO

GETTING STARTED

2.1 Dataset Statistics

We provide a sample notebook to help you load synthetic datasets generated using Perception package and visualize dataset statistics. We plan to support other sample Unity projects in the future.

2.2 Dataset Download

You can download the datasets from HTTP(s), GCS, and Unity simulation projects using the download command from *CLI* or *API*.

CLI

```
datasetinsights download \
   --source-uri=<xxx> \
   --output=$HOME/data
```

API

GCSDatasetDownloader downloads a dataset from GCS location.

```
from datasetinsights.io.downloader import GCSDatasetDownloader

source_uri=gs://url/to/file.zip or gs://url/to/folder
dest = "~/data"
downloader = GCSDatasetDownloader()
downloader.download(source_uri=source_uri, output=data_root)
```

HTTPDatasetDownloader downloads a dataset from any HTTP(S) location.

```
from datasetinsights.io.downloader import HTTPDatasetDownloader

source_uri=http://url.to.file.zip
dest = "~/data"
downloader = HTTPDatasetDownloader()
downloader.download(source_uri=source_uri, output=data_root)
```

2.3 Dataset Explore

You can explore the dataset schema by using following API:

Unity Perception

AnnotationDefinitions and MetricDefinitions loads synthetic dataset definition tables and return a dictionary containing the definitions.

```
from datasetinsights.datasets.unity_perception import AnnotationDefinitions,
MetricDefinitions
annotation_def = AnnotationDefinitions(data_root=dest, version="my_schema_version")
definition_dict = annotation_def.get_definition(def_id="my_definition_id")

metric_def = MetricDefinitions(data_root=dest, version="my_schema_version")
definition_dict = metric_def.get_definition(def_id="my_definition_id")
```

Captures loads synthetic dataset captures tables and return a pandas dataframe with captures and annotations columns.

```
from datasetinsights.datasets.unity_perception import Captures
captures = Captures(data_root=dest, version="my_schema_version")
captures_df = captures.filter(def_id="my_definition_id")
```

Metrics loads synthetic dataset metrics table which holds extra metadata that can be used to describe a particular sequence, capture or annotation and return a pandas dataframe with captures and metrics columns.

```
from datasetinsights.datasets.unity_perception import Metrics
metrics = Metrics(data_root=dest, version="my_schema_version")
metrics_df = metrics.filter_metrics(def_id="my_definition_id")
```

CHAPTER THREE

CONTENTS

3.1 datasetinsights

3.1.1 datasetinsights

datasetinsights.datasets

datasetinsights.datasets.unity_perception

datasetinsights.datasets.unity_perception.captures

Load Synthetic dataset captures and annotations tables

class datasetinsights.datasets.unity_perception.captures.**Captures**(*data_root='/data'*, *version='0.0.1'*)

Bases: object

Load captures table

A capture record stores the relationship between a captured file, a collection of annotations, and extra metadata that describes this capture. For more detail, see schema design here:

captures

Examples:

```
>>> captures = Captures(data_root="/data")
#captures class automatically loads the captures (e.g. lidar scan,
image, depth map) and the annotations (e.g semantic segmentation
labels, bounding boxes, etc.)
>>> data = captures.filter(def_id="6716c783-1c0e-44ae-b1b5-7f068454b66e") # noqa␣
↪E501 table command not be broken down into multiple lines
#return the captures and annotations filtered by the annotation
definition id
```

captures

a collection of captures without annotations

Type pd.DataFrame

annotations

a collection of annotations

Type pd.DataFrame

FILE_PATTERN = '/captures_*.json'**

TABLE_NAME = 'captures'

filter (*def_id*)

Get captures and annotations filtered by annotation definition id *captures*

Parameters **def_id** (`int`) – annotation definition id used to filter results

Returns

A pandas dataframe with captures and annotations Columns: ‘id’ (capture id), ‘sequence_id’, ‘step’, ‘timestamp’, ‘sensor’, ‘ego’,

’filename’, ‘format’, ‘annotation.id’, ‘annotation.annotation_definition’,’annotation.values’

Raises ***DefinitionIDError*** – Raised if none of the annotation records in the combined annotation and captures dataframe match the def_id specified as a parameter.

Example Returned Dataframe (first row):

label_id(int)	sequence_id(int)	step(int)	timestamp (float)	sensor (dict)	ego (dict)	filename(str)	format (str)	annotation.id (str)	annotation.annotation_definition(str)	annotation.values
2	None	50	4.9	{‘sensor_id’: ‘dDa873b…’, ‘ego_id’: ‘44ca9…’, ‘modality’: ‘camera’,’translation’: [0.0, 0.0, 0.0], ‘rotation’: [0.0, 0.0, 0.0, 1.0],’scale’: 0.344577253}	…	RGB3/asd.png	PNG	ace0	6716c	[{‘label_id’: 34, ‘label_name’: ‘snack_chips_pringles’,…’height’: 118.0}, {‘label_id’: 35, ‘… ‘height’: 91.0}…]

datasetinsights.datasets.unity_perception.exceptions

exception `datasetinsights.datasets.unity_perception.exceptions.`**DefinitionIDError**

Bases: `Exception`

Raise when a given definition id can’t be found.

datasetinsights.datasets.unity_perception.metrics

Load Synthetic dataset Metrics

class `datasetinsights.datasets.unity_perception.metrics.`**Metrics** (*data_root='/data', version='0.0.1'*)

Bases: `object`

Load metrics table

Metrics store extra metadata that can be used to describe a particular sequence, capture or annotation. Metric records are stored as arbitrary number (M) of key-value pairs. For more detail, see schema design doc: *metrics*

metrics
a collection of metrics records

Type dask.bag.core.Bag

Examples

```
>>> metrics = Metrics(data_root="/data")
>>> metrics_df = metrics.filter_metrics(def_id="my_definition_id")
#metrics_df now contains all the metrics data corresponding to
"my_definition_id"
```

One example of metrics_df (first row shown below):

label_id(int)	instance_id(int)	visible_pixels(int)
2	2	2231

FILE_PATTERN = '/metrics_*.json'**

TABLE_NAME = 'metrics'

filter_metrics(*def_id*)
Get all metrics filtered by a given metric definition id

Parameters **def_id** (*str*) – metric definition id used to filter results

Raises

- ***DefinitionIDError*** – raised if no metrics records match the given
- **def_id** –

Returns (pd.DataFrame): Columns: "label_id", "capture_id", "annotation_id", "sequence_id", "step"

datasetinsights.datasets.unity_perception.references

Load Synthetic dataset references tables

class datasetinsights.datasets.unity_perception.references.**AnnotationDefinitions**(*data_root*, *version='0.0.1'*)

Bases: object

Load annotation_definitions table

For more detail, see schema design here: *annotation_definitions.json*

table
a collection of annotation_definitions records

Type pd

FILE_PATTERN = '/annotation_definitions.json'**

TABLE_NAME = 'annotation_definitions'

find_by_name(*pattern*)
Get the annotation definition by matching patterns

This method will try to match the pattern of the annotation definition by name to determine

Parameters **pattern** (*srt*) – the regex pattern

Returns a dictionary containing the annotation definition

Return type dict

Raises

- *NoRecordError* – If no annotation are found for a given definition id
- *DuplicateRecordError* – If more than one record is found by the given pattern

get_definition (*def_id*)
Get the annotation definition for a given definition id

Parameters **def_id** (*int*) – annotation definition id used to filter results

Returns a dictionary containing the annotation definition

Return type dict

Raises *NoRecordError* – If no annotation are found for a given definition id

load_annotation_definitions (*data_root*, *version*)
Load annotation definition files.

For more detail, see schema design here: *annotation_definitions.json*

Parameters **data_root** (*str*) – the root directory of the dataset containing

tables version (str): desired schema version

Returns A Pandas dataframe with annotation definition records. Columns: 'id' (annotation id), 'name' (annotation name), 'description' (string description), 'format' (string describing format), 'spec' (Format-specific specification for the annotation values)

class datasetinsights.datasets.unity_perception.references.**Egos** (*data_root*, *version='0.0.1'*)

Bases: object

Load egos table

For more detail, see schema design here: *egos.json*

table
a collection of egos records

Type pd

FILE_PATTERN = '/egos.json'**

TABLE_NAME = 'egos'

load_egos (*data_root*, *version*)
Load egos files. For more detail, see schema design here:

egos.json

Parameters

- **data_root** (*str*) – the root directory of the dataset containing
- **tables** (*ego*) –
- **version** (*str*) – desired schema version

Returns id (ego id) and description

Return type A pandas dataframe with all ego records with two columns

class datasetinsights.datasets.unity_perception.references.**MetricDefinitions**(*data_root*, *version='0.0.1'*)

Bases: object

Load metric_definitions table

For more detail, see schema design here:

metric_definitions.json

table

a collection of metric_definitions records with columns: id

Type pd

(id for metric definition), name, description, spec (definition specific spec)

FILE_PATTERN = '/metric_definitions.json'**

TABLE_NAME = 'metric_definitions'

get_definition(*def_id*)

Get the metric definition for a given definition id

Parameters **def_id** (*int*) – metric definition id used to filter results

Returns a dictionary containing metric definition

load_metric_definitions(*data_root*, *version*)

Load metric definition files.

metric_definitions.json

Args: data_root (str): the root directory of the dataset containing tables version (str): desired schema version

Returns: A Pandas dataframe with metric definition records.

a collection of metric_definitions records with columns: id

(id for metric definition), name, description, spec (definition specific spec)

class datasetinsights.datasets.unity_perception.references.**Sensors**(*data_root*, *version='0.0.1'*)

Bases: object

Load sensors table

For more detail, see schema design here:

sensors.json

table

a collection of sensors records with columns:

Type pd

'id'

Type sensor id

({camera, lidar, radar, sonar,...} -- Sensor modality), 'description'

FILE_PATTERN = '/sensors.json'**

TABLE_NAME = 'sensors'

load_sensors (*data_root*, *version*)

Load sensors files.

For more detail, see schema design here:

sensors.json

Parameters **data_root** (*str*) – the root directory of the dataset containing

tables version (str): desired schema version

Returns

Return type A pandas dataframe with all sensors records with columns

'id' (sensor id), 'ego_id', 'modality' ({camera, lidar, radar, sonar,. . . } – Sensor modality), 'description'

datasetinsights.datasets.unity_perception.tables

class datasetinsights.datasets.unity_perception.tables.**FileType** (*value*)

Bases: enum.Enum

An enumeration.

BINARY = 'binary'

CAPTURE = 'capture'

METRIC = 'metric'

REFERENCE = 'reference'

class datasetinsights.datasets.unity_perception.tables.**Table** (*file*, *pattern*, *filetype*)

Bases: tuple

property file

Alias for field number 0

property filetype

Alias for field number 2

property pattern

Alias for field number 1

datasetinsights.datasets.unity_perception.tables.**glob** (*data_root*, *pattern*)

Find all matching files in a directory.

Parameters

- **data_root** (*str*) – directory containing capture files
- **pattern** (*str*) – Unix file pattern

Yields *str* – matched filenames in a directory

datasetinsights.datasets.unity_perception.tables.**load_table** (*json_file*, *table_name*, *version*, ***kwargs*)

Load records from json files into a pandas table

Parameters

- **json_file** (*str*) – filename to json.
- **table_name** (*str*) – table name in the json file to be loaded
- **version** (*str*) – requested version of this table
- ****kwargs** – arbitrary keyword arguments to be passed to pandas' json_normalize method.

Returns a pandas dataframe of the loaded table.

Raises

- ***VersionError*** – If the version in json file does not match the requested
- **version.** –

datasetinsights.datasets.unity_perception.validation

Validate Simulation Data

exception datasetinsights.datasets.unity_perception.validation.**DuplicateRecordError**
Bases: `Exception`

Raise when the definition file has duplicate definition id

exception datasetinsights.datasets.unity_perception.validation.**NoRecordError**
Bases: `Exception`

Raise when no record is found matching a given definition id

exception datasetinsights.datasets.unity_perception.validation.**VersionError**
Bases: `Exception`

Raise when the data file version does not match

datasetinsights.datasets.unity_perception.validation.**check_duplicate_records**(*table*, *column*, *table_name*)

Check if table has duplicate records for a given column

Parameters

- **table** (*pd.DataFrame*) – a pandas dataframe
- **column** (*str*) – the column where no duplication is allowed
- **table_name** (*str*) – table name

Raises ***DuplicateRecordError*** – If duplicate records are found in a column

datasetinsights.datasets.unity_perception.validation.**verify_version**(*json_data*, *version*)

Verify json schema version

Parameters

- **json_data** (*json*) – a json object loaded from file.
- **version** (*str*) – string of the requested version.

Raises ***VersionError*** – If the version in json file does not match the requested

version.

class datasetinsights.datasets.unity_perception.**AnnotationDefinitions**(*data_root*, *version='0.0.1'*)

Bases: `object`

Load annotation_definitions table

For more detail, see schema design here: *annotation_definitions.json*

table
a collection of annotation_definitions records

Type pd

FILE_PATTERN = '/annotation_definitions.json'**

TABLE_NAME = 'annotation_definitions'

find_by_name(*pattern*)
Get the annotation definition by matching patterns

This method will try to match the pattern of the annotation definition by name to determine

Parameters **pattern** (*srt*) – the regex pattern

Returns a dictionary containing the annotation definition

Return type dict

Raises

- ***NoRecordError*** – If no annotation are found for a given definition id
- ***DuplicateRecordError*** – If more than one record is found by the given pattern

get_definition(*def_id*)
Get the annotation definition for a given definition id

Parameters **def_id** (*int*) – annotation definition id used to filter results

Returns a dictionary containing the annotation definition

Return type dict

Raises ***NoRecordError*** – If no annotation are found for a given definition id

load_annotation_definitions(*data_root*, *version*)
Load annotation definition files.

For more detail, see schema design here: *annotation_definitions.json*

Parameters **data_root** (*str*) – the root directory of the dataset containing

tables version (str): desired schema version

Returns A Pandas dataframe with annotation definition records. Columns: 'id' (annotation id), 'name' (annotation name), 'description' (string description), 'format' (string describing format), 'spec' (Format-specific specification for the annotation values)

class datasetinsights.datasets.unity_perception.**Captures**(*data_root='/data'*, *version='0.0.1'*)

Bases: `object`

Load captures table

A capture record stores the relationship between a captured file, a collection of annotations, and extra metadata that describes this capture. For more detail, see schema design here:

captures

Examples:

```
>>> captures = Captures(data_root="/data")
#captures class automatically loads the captures (e.g. lidar scan,
image, depth map) and the annotations (e.g semantic segmentation
labels, bounding boxes, etc.)
>>> data = captures.filter(def_id="6716c783-1c0e-44ae-b1b5-7f068454b66e") # noqa␣
↪E501 table command not be broken down into multiple lines
#return the captures and annotations filtered by the annotation
definition id
```

captures

a collection of captures without annotations

Type pd.DataFrame

annotations

a collection of annotations

Type pd.DataFrame

FILE_PATTERN = '/captures_*.json'**

TABLE_NAME = 'captures'

filter(*def_id*)

Get captures and annotations filtered by annotation definition id *captures*

Parameters def_id (*int*) – annotation definition id used to filter results

Returns

A pandas dataframe with captures and annotations Columns: 'id' (capture id), 'sequence_id', 'step', 'timestamp', 'sensor', 'ego',

'filename', 'format', 'annotation.id', 'annotation.annotation_definition','annotation.values'

Raises DefinitionIDError – Raised if none of the annotation records in the combined annotation and captures dataframe match the def_id specified as a parameter.

Example Returned Dataframe (first row):

label_id(int)	sequence_id(int)	step(int)	timestamp (float)	sensor (dict)	ego (dict)	filename(str)	format (str)	annotation.id (str)	annotation.annotation_definition(str)	annotation.values
2	None	50	4.9	{'sensor_id': 'dDa873b…', 'ego_id': '44ca9…', 'modality': 'camera','translation': [0.0, 0.0, 0.0], 'rotation': [0.0, 0.0, 0.0, 1.0],'scale': 0.344577253}	…	RGB3/asd.png	PNG	ace0	6716c	[{'label_id': 34, 'label_name': 'snack_chips_pringles',…'height': 118.0}, {'label_id': 35, '… 'height': 91.0}…]

class datasetinsights.datasets.unity_perception.**Egos**(*data_root*, *version='0.0.1'*)

Bases: object

Load egos table

For more detail, see schema design here: *egos.json*

table

a collection of egos records

Type pd

FILE_PATTERN = '/egos.json'**

TABLE_NAME = 'egos'

load_egos(*data_root*, *version*)

Load egos files. For more detail, see schema design here:

egos.json

Parameters

- **data_root** (*str*) – the root directory of the dataset containing
- **tables** (*ego*) –
- **version** (*str*) – desired schema version

Returns id (ego id) and description

Return type A pandas dataframe with all ego records with two columns

class datasetinsights.datasets.unity_perception.**MetricDefinitions**(*data_root*, *version='0.0.1'*)

Bases: object

Load metric_definitions table

For more detail, see schema design here:

metric_definitions.json

table

a collection of metric_definitions records with columns: id

Type pd

(id for metric definition), name, description, spec (definition specific spec)

FILE_PATTERN = '/metric_definitions.json'**

TABLE_NAME = 'metric_definitions'

get_definition(*def_id*)

Get the metric definition for a given definition id

Parameters def_id (*int*) – metric definition id used to filter results

Returns a dictionary containing metric definition

load_metric_definitions(*data_root*, *version*)

Load metric definition files.

metric_definitions.json

Args: data_root (str): the root directory of the dataset containing tables version (str): desired schema version

Returns: A Pandas dataframe with metric definition records.

a collection of metric_definitions records with columns: id

(id for metric definition), name, description, spec (definition specific spec)

class `datasetinsights.datasets.unity_perception.`**`Metrics`**(*data_root='/data'*, *version='0.0.1'*)

Bases: `object`

Load metrics table

Metrics store extra metadata that can be used to describe a particular sequence, capture or annotation. Metric records are stored as arbitrary number (M) of key-value pairs. For more detail, see schema design doc: *metrics*

metrics

a collection of metrics records

Type dask.bag.core.Bag

Examples

```
>>> metrics = Metrics(data_root="/data")
>>> metrics_df = metrics.filter_metrics(def_id="my_definition_id")
#metrics_df now contains all the metrics data corresponding to
"my_definition_id"
```

One example of metrics_df (first row shown below):

label_id(int)	instance_id(int)	visible_pixels(int)
2	2	2231

FILE_PATTERN = '/metrics_*.json'**

TABLE_NAME = 'metrics'

filter_metrics(*def_id*)

Get all metrics filtered by a given metric definition id

Parameters **def_id** (*str*) – metric definition id used to filter results

Raises

- ***DefinitionIDError*** – raised if no metrics records match the given
- **def_id** –

Returns (pd.DataFrame): Columns: “label_id”, “capture_id”, “annotation_id”, “sequence_id”, “step”

class `datasetinsights.datasets.unity_perception.`**`Sensors`**(*data_root*, *version='0.0.1'*)

Bases: `object`

Load sensors table

For more detail, see schema design here:

sensors.json

table

a collection of sensors records with columns:

Type pd

'id'

Type sensor id

({camera, lidar, radar, sonar,...} -- Sensor modality), 'description'

FILE_PATTERN = '/sensors.json'**

TABLE_NAME = 'sensors'

load_sensors (*data_root*, *version*)

Load sensors files.

For more detail, see schema design here:

sensors.json

Parameters **data_root** (*str*) – the root directory of the dataset containing

tables version (str): desired schema version

Returns

Return type A pandas dataframe with all sensors records with columns

'id' (sensor id), 'ego_id', 'modality' ({camera, lidar, radar, sonar,. . . } – Sensor modality), 'description'

datasetinsights.datasets.exceptions

exception datasetinsights.datasets.exceptions.**DatasetNotFoundError**

Bases: Exception

Raise when a dataset file can't be found.

datasetinsights.datasets.synthetic

Simulation Dataset Catalog

datasetinsights.datasets.synthetic.**read_bounding_box_2d** (*annotation*, *label_mappings=None*)

Convert dictionary representations of 2d bounding boxes into objects of the BBox2D class

Parameters

- **annotation** (*List[dict]*) – 2D bounding box annotation
- **label_mappings** (*dict*) – a dict of {label_id: label_name} mapping

Returns A list of 2D bounding box objects

datasetinsights.datasets.synthetic.**read_bounding_box_3d** (*annotation*, *label_mappings=None*)

Convert dictionary representations of 3d bounding boxes into objects of the BBox3d class

Parameters

- **annotation** (*List[dict]*) – 3D bounding box annotation
- **label_mappings** (*dict*) – a dict of {label_id: label_name} mapping

Returns A list of 3d bounding box objects

datasetinsights.io

datasetinsights.io.downloader

datasetinsights.io.downloader.base

class datasetinsights.io.downloader.base.**DatasetDownloader**(***kwargs*)

Bases: abc.ABC

This is the base class for all dataset downloaders The DatasetDownloader can be subclasses in the following way

class NewDatasetDownloader(DatasetDownloader, protocol="protocol://")

Here the 'protocol://' should match the prefix that the method download source_uri supports. Example http:// gs://

abstract download(*source_uri*, *output*, ***kwargs*)

This method downloads a dataset stored at the source_uri and stores it in the output directory

Parameters

- **source_uri** – URI that points to the dataset that should be downloaded
- **output** – path to local folder where the dataset should be stored

datasetinsights.io.downloader.base.**create_dataset_downloader**(*source_uri*, ***kwargs*)

This function instantiates the dataset downloader after finding it with the source-uri provided

Parameters

- **source_uri** – URI used to look up the correct dataset downloader
- ****kwargs** –

Returns: The dataset downloader instance matching the source-uri.

datasetinsights.io.downloader.gcs_downloader

class datasetinsights.io.downloader.gcs_downloader.**GCSDatasetDownloader**(***kwargs*)

Bases: *datasetinsights.io.downloader.base.DatasetDownloader*

This class is used to download data from GCS

download(*source_uri=None*, *output=None*, ***kwargs*)

Parameters

- **source_uri** – This is the downloader-uri that indicates where on GCS the dataset should be downloaded from. The expected source-uri follows these patterns gs://bucket/folder or gs://bucket/folder/data.zip
- **output** – This is the path to the directory where the download will store the dataset.

datasetinsights.io.downloader.http_downloader

class datasetinsights.io.downloader.http_downloader.**HTTPDatasetDownloader**(***kwargs*)

Bases: *datasetinsights.io.downloader.base.DatasetDownloader*

This class is used to download data from any HTTP or HTTPS public url and perform function such as downloading the dataset and checksum validation if checksum file path is provided.

download(*source_uri*, *output*, *checksum_file=None*, ***kwargs*)

This method is used to download the dataset from HTTP or HTTPS url.

Parameters

- **source_uri** (*str*) – This is the downloader-uri that indicates where the dataset should be downloaded from.
- **output** (*str*) – This is the path to the directory where the download will store the dataset.
- **checksum_file** (*str*) – This is path of the txt file that contains checksum of the dataset to be downloaded. It can be HTTP or HTTPS url or local path.

Raises ***ChecksumError*** – This will raise this error if checksum doesn't matches

class datasetinsights.io.downloader.**GCSDatasetDownloader**(***kwargs*)

Bases: *datasetinsights.io.downloader.base.DatasetDownloader*

This class is used to download data from GCS

download(*source_uri=None*, *output=None*, ***kwargs*)

Parameters

- **source_uri** – This is the downloader-uri that indicates where on GCS the dataset should be downloaded from. The expected source-uri follows these patterns gs://bucket/folder or gs://bucket/folder/data.zip
- **output** – This is the path to the directory where the download will store the dataset.

class datasetinsights.io.downloader.**HTTPDatasetDownloader**(***kwargs*)

Bases: *datasetinsights.io.downloader.base.DatasetDownloader*

This class is used to download data from any HTTP or HTTPS public url and perform function such as downloading the dataset and checksum validation if checksum file path is provided.

download(*source_uri*, *output*, *checksum_file=None*, ***kwargs*)

This method is used to download the dataset from HTTP or HTTPS url.

Parameters

- **source_uri** (*str*) – This is the downloader-uri that indicates where the dataset should be downloaded from.
- **output** (*str*) – This is the path to the directory where the download will store the dataset.
- **checksum_file** (*str*) – This is path of the txt file that contains checksum of the dataset to be downloaded. It can be HTTP or HTTPS url or local path.

Raises ***ChecksumError*** – This will raise this error if checksum doesn't matches

datasetinsights.io.downloader.**create_dataset_downloader**(*source_uri*, ***kwargs*)

This function instantiates the dataset downloader after finding it with the source-uri provided

Parameters

- **source_uri** – URI used to look up the correct dataset downloader
- ****kwargs** –

Returns: The dataset downloader instance matching the source-uri.

datasetinsights.io.bbox

class datasetinsights.io.bbox.**BBox2D**(*label*, *x*, *y*, *w*, *h*, *score=1.0*)

Bases: `object`

Canonical Representation of a 2D bounding box.

label

string representation of the label.

Type str

x

x pixel coordinate of the upper left corner.

Type float

y

y pixel coordinate of the upper left corner.

Type float

w

width (number of pixels)of the bounding box.

Type float

h

height (number of pixels) of the bounding box.

Type float

score

detection confidence score. Default is set to score=1. if this is a ground truth bounding box.

Type float

Examples

Here is an example about how to use this class.

```
>>> gt_bbox = BBox2D(label='car', x=2, y=6, w=2, h=4)
>>> gt_bbox
"label='car'|score=1.0|x=2.0|y=6.0|w=2.0|h=4.0"
>>> pred_bbox = BBox2D(label='car', x=2, y=5, w=2, h=4, score=0.79)
>>> pred_bbox.area
8
>>> pred_bbox.intersect_with(gt_bbox)
True
>>> pred_bbox.intersection(gt_bbox)
6
>>> pred_bbox.union(gt_bbox)
```

(continues on next page)

(continued from previous page)

```
10
>>> pred_bbox.iou(gt_bbox)
0.6
```

property area
Calculate area of this bounding box

Returns width x height of the bound box

intersect_with(*other*)
Check whether this box intersects with other bounding box

Parameters **other** (BBox2D) – other bounding box object to check intersection

Returns True if two bounding boxes intersect, False otherwise

intersection(*other*)
Calculate the intersection area with other bounding box

Parameters **other** (BBox2D) – other bounding box object to calculate intersection

Returns float of the intersection area for two bounding boxes

iou(*other*)
Calculate intersection over union area with other bounding box

$$IOU = \frac{intersection}{union}$$

Parameters **other** (BBox2D) – other bounding box object to calculate iou

Returns float of the union area for two bounding boxes

union(*other*, *intersection_area=None*)
Calculate union area with other bounding box

Parameters

- **other** (BBox2D) – other bounding box object to calculate union
- **intersection_area** (`float`) – pre-calculated area of intersection

Returns float of the union area for two bounding boxes

class datasetinsights.io.bbox.**BBox3D**(*translation*, *size*, *label*, *sample_token*, *score=1*, *rotation: pyquaternion.quaternion.Quaternion = Quaternion(1.0, 0.0, 0.0, 0.0)*, *velocity=(nan, nan, nan)*)

Bases: `object`

Class for 3d bounding boxes which can either be predictions or ground-truths. This class is the primary representation in this repo of 3d bounding boxes and is based off of the Nuscenes style dataset.

property back_left_bottom_pt
Back-left-bottom point.

Type Returns

Type `float`

property back_left_top_pt
Back-left-top point.

Type `float`

property back_right_bottom_pt
Back-right-bottom point.

Type `float`

property back_right_top_pt
Back-right-top point.

Type `float`

property front_left_bottom_pt
Front-left-bottom point.

Type `float`

property front_left_top_pt
Front-left-top point.

Type `float`

property front_right_bottom_pt
Front-right-bottom point.

Type `float`

property front_right_top_pt
Front-right-top point.

Type `float`

property p

list of all 8 corners of the box beginning with the the bottom four corners and then the top

four corners, both in counterclockwise order (from birds eye view) beginning with the back-left corner

Type Returns

`datasetinsights.io.bbox.`**group_bbox2d_per_label**(*bboxes*)
Group 2D bounding boxes with same label.

Parameters **bboxes** (*list[BBox2D]*) – a list of 2D bounding boxes

Returns a dictionary of 2d boundign box group. {label1: [bbox1, bboxes2, ...], label2: [bbox1, ...]}

Return type dict

datasetinsights.io.download

class `datasetinsights.io.download.`**TimeoutHTTPAdapter**(*timeout*, **args*, ***kwargs*)
Bases: `requests.adapters.HTTPAdapter`

send(*request*, ***kwargs*)
Sends PreparedRequest object. Returns Response object.

Parameters

- **request** – The `PreparedRequest` being sent.
- **stream** – (optional) Whether to stream the request content.
- **timeout** (*float or tuple or urllib3 Timeout object*) – (optional) How long to wait for the server to send data before giving up, as a float, or a (connect timeout, read timeout) tuple.

- **verify** – (optional) Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** – (optional) Any user-provided SSL certificate to be trusted.
- **proxies** – (optional) The proxies dictionary to apply to the request.

Return type requests.Response

datasetinsights.io.download.**checksum_matches**(*filepath*, *expected_checksum*, *algorithm='CRC32'*)

Check if the checksum matches

Parameters

- **filepath** (*str*) – the doaloaded file path
- **expected_checksum** (*int*) – expected checksum of the file
- **algorithm** (*str*) – checksum algorithm. Defaults to CRC32

Returns True if the file checksum matches.

datasetinsights.io.download.**compute_checksum**(*filepath*, *algorithm='CRC32'*)

Compute the checksum of a file.

Parameters

- **filepath** (*str*) – the doaloaded file path
- **algorithm** (*str*) – checksum algorithm. Defaults to CRC32

Returns the checksum value

Return type int

datasetinsights.io.download.**download_file**(*source_uri: str*, *dest_path: str*, *file_name: Optional[str] = None*)

Download a file specified from a source uri

Parameters

- **source_uri** (*str*) – source url where the file should be downloaded
- **dest_path** (*str*) – destination path of the file
- **file_name** (*str*) – file name of the file to be downloaded

Returns String of destination path.

datasetinsights.io.download.**get_checksum_from_file**(*filepath*)

This method return checksum of the file whose filepath is given.

Parameters **filepath** (*str*) – Path of the checksum file. Path can be HTTP(s) url or local path.

Raises **ValueError** – Raises this error if filepath is not local or not HTTP or HTTPS url.

datasetinsights.io.download.**validate_checksum**(*filepath*, *expected_checksum*, *algorithm='CRC32'*)

Validate checksum of the downloaded file.

Parameters

- **filepath** (*str*) – the doaloaded file path
- **expected_checksum** (*int*) – expected checksum of the file
- **algorithm** (*str*) – checksum algorithm. Defaults to CRC32

Raises **ChecksumError if the file checksum does not match.** –

datasetinsights.io.exceptions

exception datasetinsights.io.exceptions.**ChecksumError**
Bases: Exception

Raises when the downloaded file checksum is not correct.

exception datasetinsights.io.exceptions.**DownloadError**
Bases: Exception

Raise when download file failed.

exception datasetinsights.io.exceptions.**InvalidTrackerError**
Bases: Exception

Raises when unknown tracker requested .

datasetinsights.io.gcs

class datasetinsights.io.gcs.**GCSClient**(***kwargs*)
Bases: object

This class is used to download data from GCS location and perform function such as downloading the dataset and checksum validation.

GCS_PREFIX = '^gs://'

KEY_SEPARATOR = '/'

download(**, url=None, local_path=None, bucket=None, key=None*)
This method is used to download the dataset from GCS.

Parameters

- **url** (str) – This is the downloader-uri that indicates where the dataset should be downloaded from.
- **local_path** (str) – This is the path to the directory where the download will store the dataset.
- **bucket** (str) – gcs bucket name
- **key** (str) – object key path
- **Examples** –

```
>>> url = "gs://bucket/folder or gs://bucket/folder/data.zip"
>>> local_path = "/tmp/folder"
>>> bucket ="bucket"
>>> key ="folder/data.zip" or "folder"
```

get_most_recent_blob(*url=None, bucket_name=None, key=None*)
Get the last updated blob in a given bucket under given prefix

Parameters

- **bucket_name** (str) – gcs bucket name
- **key** (str) – object key path

upload(**, local_path=None, bucket=None, key=None, url=None, pattern='*'*)
Upload a file or list of files from directory to GCS

Parameters

- **url** (*str*) – This is the gcs location that indicates where
- **dataset should be uploaded.** (*the*) –
- **local_path** (*str*) – This is the path to the directory or file
- **the data is stored.** (*where*) –
- **bucket** (*str*) – gcs bucket name
- **key** (*str*) – object key path
- **pattern** – Unix glob patterns. Use **/* for recursive glob.
- **Examples** –

  For file upload:

  ```
  >>> url = "gs://bucket/folder/data.zip"
  >>> local_path = "/tmp/folder/data.zip"
  >>> bucket ="bucket"
  >>> key ="folder/data.zip"
  ```

  For directory upload:

  ```
  >>> url = "gs://bucket/folder"
  >>> local_path = "/tmp/folder"
  >>> bucket ="bucket"
  >>> key ="folder"
  >>> key ="**/*"
  ```

class datasetinsights.io.**BBox2D**(*label*, *x*, *y*, *w*, *h*, *score=1.0*)

Bases: object

Canonical Representation of a 2D bounding box.

label

string representation of the label.

Type str

x

x pixel coordinate of the upper left corner.

Type float

y

y pixel coordinate of the upper left corner.

Type float

w

width (number of pixels)of the bounding box.

Type float

h

height (number of pixels) of the bounding box.

Type float

score

detection confidence score. Default is set to score=1. if this is a ground truth bounding box.

Type float

Examples

Here is an example about how to use this class.

```
>>> gt_bbox = BBox2D(label='car', x=2, y=6, w=2, h=4)
>>> gt_bbox
"label='car'|score=1.0|x=2.0|y=6.0|w=2.0|h=4.0"
>>> pred_bbox = BBox2D(label='car', x=2, y=5, w=2, h=4, score=0.79)
>>> pred_bbox.area
8
>>> pred_bbox.intersect_with(gt_bbox)
True
>>> pred_bbox.intersection(gt_bbox)
6
>>> pred_bbox.union(gt_bbox)
10
>>> pred_bbox.iou(gt_bbox)
0.6
```

property area
Calculate area of this bounding box

Returns width x height of the bound box

intersect_with(*other*)
Check whether this box intersects with other bounding box

Parameters **other** (BBox2D) – other bounding box object to check intersection

Returns True if two bounding boxes intersect, False otherwise

intersection(*other*)
Calculate the intersection area with other bounding box

Parameters **other** (BBox2D) – other bounding box object to calculate intersection

Returns float of the intersection area for two bounding boxes

iou(*other*)
Calculate intersection over union area with other bounding box

$$IOU = \frac{intersection}{union}$$

Parameters **other** (BBox2D) – other bounding box object to calculate iou

Returns float of the union area for two bounding boxes

union(*other*, *intersection_area=None*)
Calculate union area with other bounding box

Parameters

- **other** (BBox2D) – other bounding box object to calculate union
- **intersection_area** (*float*) – pre-calculated area of intersection

Returns float of the union area for two bounding boxes

datasetinsights.io.**create_dataset_downloader**(*source_uri*, ***kwargs*)

This function instantiates the dataset downloader after finding it with the source-uri provided

Parameters

- **source_uri** – URI used to look up the correct dataset downloader
- ****kwargs** –

Returns: The dataset downloader instance matching the source-uri.

datasetinsights.stats

datasetinsights.stats.visualization

datasetinsights.stats.visualization.app

`datasetinsights.stats.visualization.app.get_app()`

datasetinsights.stats.visualization.bbox2d_plot

Use a bounding box library to plot pretty bounding boxes with a simple Python API. This library helps to display pretty bounding boxes with a chosen set of colors. Reference: https://github.com/nalepae/bounding-box

`datasetinsights.stats.visualization.bbox2d_plot.add_single_bbox_on_image(image, bbox, label, color, font_size=100, box_line_width=15)`

Add single bounding box with label on a given image.

Parameters

- **image** (*numpy array*) – a numpy array for an image.
- **bbox** (BBox2D) – a canonical bounding box.
- **color** (*str*) – a color name for one bounding box.
- **color = None** (*If*) –
- **will randomly assign a color for each box.** (*it*) –
- **font_size** (*int*) – font size for each label. Defaults to 100.
- **box_line_width** (*int*) – line width of the bounding boxes. Defaults to 15.

datasetinsights.stats.visualization.bbox3d_plot

Helper bounding box 3d library to plot pretty 3D boundign boxes with a simple Python API.

`datasetinsights.stats.visualization.bbox3d_plot.`**`add_single_bbox3d_on_image`**(*image*, *box*, *proj*, *color=None*, *box_line_width=2*, *orthographic=False*)

” Add single 3D bounding box on a given image.

Parameters

- **image** (*numpy array*) – a numpy array for an image
- **box** (BBox3D) – a 3D bounding box in camera’s coordinate system
- **proj** (*numpy 2D array*) – camera’s 3x3 projection matrix
- **color** (*tuple*) – RGBA color of the bounding box. Defaults to None. If
- **= None the the tuple of [0** (*color*) –
- **255** (*Green*) –
- **0** (*Green*) –
- **255]** (*Green*) –
- **box_line_width** (*int*) – line width of the bounding boxes. Defaults to 2.
- **orthographic** (*bool*) – true if proj is orthographic, else perspective

datasetinsights.stats.visualization.constants

datasetinsights.stats.visualization.keypoints_plot

Helper keypoints library to plot keypoint joints and skeletons with a simple Python API.

`datasetinsights.stats.visualization.keypoints_plot.`**`draw_keypoints_for_figure`**(*image*, *figure*, *draw*, *templates*, *visual_width=6*)

Draws keypoints for a figure on an image.

keypoints { label_id: <int> Integer identifier of the label. instance_id: <str> UUID of the instance. template_guid: <str> UUID of the keypoint template. pose: <str> String label for current pose. keypoints [

{ index: <int> Index of keypoint in template. x: <float> X subpixel coordinate of keypoint. y: <float> Y subpixel coordinate of keypoint state: <int> 0: keypoint does not exist,

1: keypoint exists but is not visible, 2: keypoint exists and is visible.

}, …

]

}

Parameters

- **image** (*PIL Image*) – a PIL image.
- **figure** – The figure to draw.
- **draw** (*PIL ImageDraw*) – PIL image draw interface.
- **templates** (*list*) – a list of keypoint templates.
- **visual_width** (*int*) – the visual width of the joints.

Returns: a PIL image with keypoints for a figure drawn on it.

datasetinsights.stats.visualization.object_detection

class datasetinsights.stats.visualization.object_detection.**Lighting**(*data_root*)

Bases: object

This class contains methods for object lighting statistics visualization.

metrics

a collection of metrics records

Type sim.Metrics

lighting

contains information about per-frame light color and orientation information.

Type pandas.DataFrame

COLOR_COLUMNS = ['color.r', 'color.g', 'color.b', 'color.a']

X_Y_COLUMNS = ['x_rotation', 'y_rotation']

html()

Method for generating html layout for the lighting statistics.

Returns displays lighting graphs.

Return type html layout

class datasetinsights.stats.visualization.object_detection.**ObjectPlacement**(*data_root*)

Bases: object

This class contains methods for object orientation statistics visualization.

metrics

a collection of metrics records

Type sim.Metrics

lighting

contains information about per-frame light color and orientation information.

Type pandas.DataFrame

OBJECT_ORIENTATION = ('x_rot', 'y_rot', 'z_rot')

html()

Method for generating html layout for the object orientation statistics.

Returns displays object orientation graphs.

Return type html layout

class datasetinsights.stats.visualization.object_detection.**ScaleFactor**(*data_root*)

Bases: object

Generate scale factor distribution.

Scale Factor describes the size of the rendered object in a capture. Higher the scale factor, higher would be the visible pixels.

Atrributes: captures(sim.Captures): a collection of capture records.

static generate_scale_data(*captures*)

Method to extract scale parameter from sensor data.

Parameters **captures** (*sim.Captures*) – a collection of capture records.

Returns contains 'scale' parameter from the sensor data.

Return type pandas.DataFrame

html()

Method for generating plots for scale factor distribution.

Returns displays scale factor distribution.

Return type html layout

class datasetinsights.stats.visualization.object_detection.**UserParameter**(*data_root*)

Bases: object

Generate User Parameter

Generate User Parameter table to be displayed on the Dashboard. Users parameters, such as ScaleFactors, MaxFrames, MaxForegroundObjectsPerFrame are used to control the domain randomization parameter used in the simulation.

Atrributes: metrics(sim.Metrics): a collection of metrics records user_parameter_table (pandas.DataFrame): dataframe containing user

parameters.

Parameters **data_root** (*str*) – path to the dataset.

html()

Method for generating html layout for the user input parameter table.

Returns displays user input parameter table.

Return type html layout

datasetinsights.stats.visualization.object_detection.**render_object_detection_layout**(*data_root*)

Method for displaying object detection statistics.

Parameters **data_root** (*str*) – path to the dataset.

Returns

displays graphs for rotation and lighting statistics for the object.

Return type html layout

datasetinsights.stats.visualization.overview

datasetinsights.stats.visualization.overview.**generate_per_capture_count_figure**(*max_samples*, *roinfo*)

Method for generating object count per capture histogram using ploty.

Parameters

- **max_samples** (*int*) – maximum number of samples that will be included in the plot.
- **roinfo** (*datasetinsights.data.datasets.statistics.RenderedObjectInfo*) – Rendered Object Info in Captures.

Returns chart to display object counts per capture

Return type plotly.graph_objects.Figure

datasetinsights.stats.visualization.overview.**generate_pixels_visible_per_object_figure**(*max_*
roinf

Method for generating pixels visible per object histogram using ploty.

Parameters

- **max_samples** (*int*) – maximum number of samples that will be included in the plot.
- **roinfo** (*datasetinsights.data.datasets.statistics.RenderedObjectInfo*) – Rendered Object Info in Captures.

Returns chart to display visible pixels per object

Return type plotly.graph_objects.Figure

datasetinsights.stats.visualization.overview.**generate_total_counts_figure**(*max_samples*, *roinfo*)

Method for generating total object count bar plot using ploty.

Parameters

- **max_samples** (*int*) – maximum number of samples that will be included in the plot.
- **roinfo** (*datasetinsights.data.datasets.statistics.RenderedObjectInfo*) – Rendered Object Info in Captures.

Returns chart to display total object count

Return type plotly.graph_objects.Figure

datasetinsights.stats.visualization.overview.**html_overview**(*data_root*)

Method for displaying overview statistics.

Parameters **data_root** (*str*) – path to the dataset.

Returns displays graphs for overview statistics.

Return type html layout

datasetinsights.stats.visualization.overview.**update_object_counts_capture_figure**(*label_value*, *json_data_roo*

Method for generating object count per capture histogram for selected object.

Parameters **label_value** (*str*) – value selected by user using drop-down

Returns displays object count distribution.

Return type plotly.graph_objects.Figure datasetinsights.stats.visualization.overview.**update_visible_pixels_figure**(*label_value*, *json_data_root*)

Method for generating pixels visible histogram for selected object. :param label_value: value selected by user using drop-down :type label_value: str

Returns displays visible pixels distribution.

Return type plotly.graph_objects.Figure

datasetinsights.stats.visualization.plots

datasetinsights.stats.visualization.plots.**bar_plot**(*df*, *x*, *y*, *title=None*, *x_title=None*, *y_title=None*, *x_tickangle=0*, ***kwargs*)

Create plotly bar plot

Parameters

- **df** (`pd.DataFrame`) – A pandas dataframe that contain bar plot data.
- **x** (`str`) – The column name of the data in x-axis.
- **y** (`str`) – The column name of the data in y-axis.
- **title** (`str, optional`) – The title of this plot.
- **x_title** (`str, optional`) – The x-axis title.
- **y_title** (`str, optional`) – The y-axis title.
- **x_tickangle** (`int, optional`) – X-axis text tickangle (default: 0)

This method can also take addition keyword arguments that can be passed to [plotly.express.bar](https://plotly.com/python-api-reference/generated/plotly.express.bar.html#plotly.express.bar) method.

Examples

```
>>> import pandas as pd
>>> df = pd.DataFrame({"id": [0, 1, 2], "name": ["a", "b", "c"],
...                    "count": [10, 20, 30]})
>>> bar_plot(df, x="id", y="count", hover_name="name")
```

datasetinsights.stats.visualization.plots.**grid_plot**(*images*, *figsize=(3, 5)*, *img_type='rgb'*, *titles=None*)

Plot 2D array of images in grid. :param images: 2D array of images. :type images: list :param figsize: target figure size of each image in the grid. :type figsize: tuple :param Defaults to: :type Defaults to: 3, 5 :param img_type: image plot type (“rgb”, “gray”). Defaults to “rgb”. :type img_type: string :param titles: a list of titles. Defaults to None. :type titles: list[str]

Returns matplotlib figure the combined grid plot.

datasetinsights.stats.visualization.plots.**histogram_plot**(*df*, *x*, *max_samples=None*, *title=None*, *x_title=None*, *y_title=None*, ***kwargs*)

Create plotly histogram plot

Parameters

- **df** (`pd.DataFrame`) – A pandas dataframe that contain raw data.

- **x** (*str*) – The column name of the raw data for histogram plot.
- **title** (*str, optional*) – The title of this plot.
- **x_title** (*str, optional*) – The x-axis title.
- **y_title** (*str, optional*) – The y-axis title.

This method can also take addition keyword arguments that can be passed to [plotly.express.histogram](https://plotly.com/python-api-reference/generated/plotly.express.histogram.html) method.

Examples

```
>>> import pandas as pd
>>> df = pd.DataFrame({"id": [0, 1, 2], "count": [10, 20, 30]})
>>> histogram_plot(df, x="count")
```

Histnorm plot using probability density:

```
>>> histogram_plot(df, x="count", histnorm="probability density")
```

datasetinsights.stats.visualization.plots.**model_performance_box_plot**(*title=None, mean_ap=None, mean_ap_50=None, mean_ar=None, range=[0, 1.0], **kwargs*)

Create a box plot for one model performance :param title: title of the plot :type title: str :param mean_ap: a list of base mAP :type mean_ap: list :param mean_ap_50: a list of base mAP :type mean_ap_50: list :param mean_ar: a list of base mAP :type mean_ar: list :param range: the range of y axis. Defaults to [0, 1.0] :type range: list

Returns A plotly.graph_objects.Figure containing the box plot

datasetinsights.stats.visualization.plots.**model_performance_comparison_box_plot**(*title=None, mean_ap_base=None, mean_ap_50_base=None, mean_ar_base=None, mean_ap_new=None, mean_ap_50_new=None, mean_ar_new=None, range=[0, 1.0], **kwargs*)

Create a box plot for a base and new model performance :param title: title of the plot :type title: str :param mean_ap_base: a list of base mAP :type mean_ap_base: list :param mean_ap_50_base: a list of base mAP :type mean_ap_50_base: list :param mean_ar_base: a list of base mAP :type mean_ar_base: list :param mean_ap_new: a list of base mAP :type mean_ap_new: list :param mean_ap_50_new: a list of base mAP :type mean_ap_50_new: list :param mean_ar_new: a list of base mAP :type mean_ar_new: list :param range: the range of y axis. Defaults to [0, 1.0] :type range: list

Returns A plotly.graph_objects.Figure containing the box plot

datasetinsights.stats.visualization.plots.**plot_bboxes**(*image, bboxes, label_mappings=None, colors=None*)

Plot an image with bounding boxes.

For ground truth image, a color is randomly selected for each bounding box. For prediction, the color of a boundnig box is coded based on IOU value between prediction and ground truth bounding boxes. It is considered true positive if IOU >= 0.5. We only visualize prediction bounding box with score >= 0.5. For prediction, it's a green box if the predicted bounding box can be matched to a ground truth bounding boxes. It's a red box if the predicted bounding box can't be matched to a ground truth bounding boxes.

Parameters

- **image** (*PIL Image*) – a PIL image.
- **bboxes** (*list*) – a list of BBox2D objects.
- **label_mappings** (*dict*) – a dict of {label_id: label_name} mapping
- **to None.** (*Defaults*) –
- **colors** (*list*) – a color list for boxes. Defaults to None.
- **colors = None** (*If*) –
- **will randomly assign PIL.COLORS for each box.** (*it*) –

Returns a PIL image with bounding boxes drawn.

Return type PIL Image

`datasetinsights.stats.visualization.plots.`**`plot_bboxes3d`**(*image*, *bboxes*, *projection*, *colors=None*, *orthographic=False*)

Plot an image with 3D bounding boxes

Currently this method should only be used for ground truth images, and doesn't support predictions. If a list of colors is not provided as an argument to this routine, the default color of green will be used.

Parameters

- **image** (*PIL Image*) – a PIL image.
- **bboxes** (*list*) – a list of BBox3D objects
- **projection** – The perspective projection of the camera which
- **the ground truth.** (*captured*) –
- **colors** (*list*) – a color list for boxes. Defaults to none. If
- **= None** (*colors*) –
- **will default to coloring all boxes green.** (*it*) –
- **orthographic** (*bool*) – true if proj is orthographic, else perspective

Returns a PIL image with bounding boxes drawn on it.

Return type PIL image

`datasetinsights.stats.visualization.plots.`**`plot_keypoints`**(*image*, *annotations*, *templates*, *visual_width=6*)

Plot an image with keypoint data.

Currently only used for ground truth info. Keypoints and colors are defined in templates.

Parameters

- **image** (*PIL Image*) – a PIL image.
- **annotations** (*list*) – a list of keypoint annotation data.
- **templates** (*list*) – a list of keypoint templates.

- **visual_width** (*int*) – the width of the visual elements

Returns a PIL image with keypoints drawn.

Return type PIL Image

`datasetinsights.stats.visualization.plots.`**`rotation_plot`**(*df*, *x*, *y*, *z=None*, *max_samples=None*, *title=None*, ***kwargs*)

Create a plotly 3d rotation plot :param df: A pandas dataframe that contains the raw data. :type df: pd.DataFrame :param x: The column name containing x rotations. :type x: str :param y: The column name containing y rotations. :type y: str :param z: The column name containing z rotations. :type z: str, optional :param title: The title of this plot. :type title: str, optional

This method can also take addition keyword arguments that can be passed to [plotly.graph_objects.Scatter3d](https://plotly.com/python-api-reference/generated/plotly.graph_objects.Scatter3d.html) method.

Returns A plotly.graph_objects.Figure containing the scatter plot

`datasetinsights.stats.visualization.`**`grid_plot`**(*images*, *figsize=(3, 5)*, *img_type='rgb'*, *titles=None*)

Plot 2D array of images in grid. :param images: 2D array of images. :type images: list :param figsize: target figure size of each image in the grid. :type figsize: tuple :param Defaults to: :type Defaults to: 3, 5 :param img_type: image plot type ("rgb", "gray"). Defaults to "rgb". :type img_type: string :param titles: a list of titles. Defaults to None. :type titles: list[str]

Returns matplotlib figure the combined grid plot.

`datasetinsights.stats.visualization.`**`plot_bboxes`**(*image*, *bboxes*, *label_mappings=None*, *colors=None*)

Plot an image with bounding boxes.

For ground truth image, a color is randomly selected for each bounding box. For prediction, the color of a boundnig box is coded based on IOU value between prediction and ground truth bounding boxes. It is considered true positive if IOU >= 0.5. We only visualize prediction bounding box with score >= 0.5. For prediction, it's a green box if the predicted bounding box can be matched to a ground truth bounding boxes. It's a red box if the predicted bounding box can't be matched to a ground truth bounding boxes.

Parameters

- **image** (*PIL Image*) – a PIL image.
- **bboxes** (*list*) – a list of BBox2D objects.
- **label_mappings** (*dict*) – a dict of {label_id: label_name} mapping
- **to None.** (*Defaults*) –
- **colors** (*list*) – a color list for boxes. Defaults to None.
- **colors = None** (*If*) –
- **will randomly assign PIL.COLORS for each box.** (*it*) –

Returns a PIL image with bounding boxes drawn.

Return type PIL Image

datasetinsights.stats.statistics

class datasetinsights.stats.statistics.**RenderedObjectInfo**(*data_root='/data'*, *version='0.0.1'*, *def_id=None*)

Bases: object

Rendered Object Info in Captures

This metric stores common object info captured by a sensor in the simulation environment. It can be used to calculate object statistics such as object count, object rotation and visible pixels.

raw_table

rendered object info stored with a tidy

Type pd.DataFrame

pandas dataframe. Columns "label_id", "instance_id", "visible_pixels",

"capture_id, "label_name".

Examples:

```
>>> # set the data root path to where data was stored
>>> data_root = "$HOME/data"
>>> # use rendered object info definition id
>>> definition_id = "659c6e36-f9f8-4dd6-9651-4a80e51eabc4"
>>> roinfo = RenderedObjectInfo(data_root, definition_id)
#total object count per label dataframe
>>> roinfo.total_counts()
label_id label_name count
       1    object1    10
       2    object2    21
#object count per capture dataframe
>>> roinfo.per_capture_counts()
capture_id  count
    qwerty     10
    asdfgh     21
```

COUNT_COLUMN = 'count'

INDEX_COLUMN = 'capture_id'

LABEL = 'label_id'

LABEL_READABLE = 'label_name'

VALUE_COLUMN = 'values'

num_captures()

Total number of captures

Returns Total number of captures

Return type integer

per_capture_counts()

Aggregate Object Counts Per Label

Returns

Total object counts table. Columns “capture_id”, “count”

Return type pd.DataFrame

total_counts()

Aggregate Total Object Counts Per Label

Returns

Total object counts table. Columns "label_id", "label_name", "count"

Return type pd.DataFrame

class datasetinsights.stats.**RenderedObjectInfo**(*data_root='/data'*, *version='0.0.1'*, *def_id=None*)

Bases: object

Rendered Object Info in Captures

This metric stores common object info captured by a sensor in the simulation environment. It can be used to calculate object statistics such as object count, object rotation and visible pixels.

raw_table

rendered object info stored with a tidy

Type pd.DataFrame

pandas dataframe. Columns "label_id", "instance_id", "visible_pixels",

"capture_id, "label_name".

Examples:

```
>>> # set the data root path to where data was stored
>>> data_root = "$HOME/data"
>>> # use rendered object info definition id
>>> definition_id = "659c6e36-f9f8-4dd6-9651-4a80e51eabc4"
>>> roinfo = RenderedObjectInfo(data_root, definition_id)
#total object count per label dataframe
>>> roinfo.total_counts()
label_id label_name count
       1     object1    10
       2     object2    21
#object count per capture dataframe
>>> roinfo.per_capture_counts()
capture_id  count
    qwerty     10
    asdfgh     21
```

COUNT_COLUMN = 'count'

INDEX_COLUMN = 'capture_id'

LABEL = 'label_id'

LABEL_READABLE = 'label_name'

VALUE_COLUMN = 'values'

num_captures()

Total number of captures

Returns Total number of captures

Return type integer

per_capture_counts()

Aggregate Object Counts Per Label

Returns

Total object counts table. Columns "capture_id", "count"

Return type pd.DataFrame

total_counts()

Aggregate Total Object Counts Per Label

Returns

Total object counts table. Columns "label_id", "label_name", "count"

Return type pd.DataFrame

datasetinsights.stats.**bar_plot**(*df*, *x*, *y*, *title=None*, *x_title=None*, *y_title=None*, *x_tickangle=0*, ***kwargs*)

Create plotly bar plot

Parameters

- **df** (*pd.DataFrame*) – A pandas dataframe that contain bar plot data.
- **x** (*str*) – The column name of the data in x-axis.
- **y** (*str*) – The column name of the data in y-axis.
- **title** (*str, optional*) – The title of this plot.
- **x_title** (*str, optional*) – The x-axis title.
- **y_title** (*str, optional*) – The y-axis title.
- **x_tickangle** (*int, optional*) – X-axis text tickangle (default: 0)

This method can also take addition keyword arguments that can be passed to [plotly.express.bar](https://plotly.com/python-api-reference/generated/plotly.express.bar.html#plotly.express.bar) method.

Examples

```
>>> import pandas as pd
>>> df = pd.DataFrame({"id": [0, 1, 2], "name": ["a", "b", "c"],
...                     "count": [10, 20, 30]})
>>> bar_plot(df, x="id", y="count", hover_name="name")
```

datasetinsights.stats.**grid_plot**(*images*, *figsize=(3, 5)*, *img_type='rgb'*, *titles=None*)

Plot 2D array of images in grid. :param images: 2D array of images. :type images: list :param figsize: target figure size of each image in the grid. :type figsize: tuple :param Defaults to: :type Defaults to: 3, 5 :param img_type: image plot type ("rgb", "gray"). Defaults to "rgb". :type img_type: string :param titles: a list of titles. Defaults to None. :type titles: list[str]

Returns matplotlib figure the combined grid plot.

datasetinsights.stats.**histogram_plot**(*df*, *x*, *max_samples=None*, *title=None*, *x_title=None*, *y_title=None*, ***kwargs*)

Create plotly histogram plot

Parameters

- **df** (*pd.DataFrame*) – A pandas dataframe that contain raw data.
- **x** (*str*) – The column name of the raw data for histogram plot.
- **title** (*str, optional*) – The title of this plot.
- **x_title** (*str, optional*) – The x-axis title.

- **y_title** (*str, optional*) – The y-axis title.

This method can also take addition keyword arguments that can be passed to [plotly.express.histogram](https://plotly.com/python-api-reference/generated/plotly.express.histogram.html) method.

Examples

```
>>> import pandas as pd
>>> df = pd.DataFrame({"id": [0, 1, 2], "count": [10, 20, 30]})
>>> histogram_plot(df, x="count")
```

Histnorm plot using probability density:

```
>>> histogram_plot(df, x="count", histnorm="probability density")
```

`datasetinsights.stats.`**`model_performance_box_plot`**(*title=None*, *mean_ap=None*, *mean_ap_50=None*, *mean_ar=None*, *range=[0, 1.0]*, ***kwargs*)

Create a box plot for one model performance :param title: title of the plot :type title: str :param mean_ap: a list of base mAP :type mean_ap: list :param mean_ap_50: a list of base mAP :type mean_ap_50: list :param mean_ar: a list of base mAP :type mean_ar: list :param range: the range of y axis. Defaults to [0, 1.0] :type range: list

Returns A plotly.graph_objects.Figure containing the box plot

`datasetinsights.stats.`**`model_performance_comparison_box_plot`**(*title=None*, *mean_ap_base=None*, *mean_ap_50_base=None*, *mean_ar_base=None*, *mean_ap_new=None*, *mean_ap_50_new=None*, *mean_ar_new=None*, *range=[0, 1.0]*, ***kwargs*)

Create a box plot for a base and new model performance :param title: title of the plot :type title: str :param mean_ap_base: a list of base mAP :type mean_ap_base: list :param mean_ap_50_base: a list of base mAP :type mean_ap_50_base: list :param mean_ar_base: a list of base mAP :type mean_ar_base: list :param mean_ap_new: a list of base mAP :type mean_ap_new: list :param mean_ap_50_new: a list of base mAP :type mean_ap_50_new: list :param mean_ar_new: a list of base mAP :type mean_ar_new: list :param range: the range of y axis. Defaults to [0, 1.0] :type range: list

Returns A plotly.graph_objects.Figure containing the box plot

`datasetinsights.stats.`**`plot_bboxes`**(*image*, *bboxes*, *label_mappings=None*, *colors=None*)

Plot an image with bounding boxes.

For ground truth image, a color is randomly selected for each bounding box. For prediction, the color of a boundnig box is coded based on IOU value between prediction and ground truth bounding boxes. It is considered true positive if IOU >= 0.5. We only visualize prediction bounding box with score >= 0.5. For prediction, it's a green box if the predicted bounding box can be matched to a ground truth bounding boxes. It's a red box if the predicted bounding box can't be matched to a ground truth bounding boxes.

Parameters

- **image** (*PIL Image*) – a PIL image.
- **bboxes** (*list*) – a list of BBox2D objects.
- **label_mappings** (*dict*) – a dict of {label_id: label_name} mapping

- **to None.** (*Defaults*) –
- **colors** (*list*) – a color list for boxes. Defaults to None.
- **colors = None** (*If*) –
- **will randomly assign PIL.COLORS for each box.** (*it*) –

Returns a PIL image with bounding boxes drawn.

Return type PIL Image

`datasetinsights.stats.`**`plot_keypoints`**(*image*, *annotations*, *templates*, *visual_width=6*)

Plot an image with keypoint data.

Currently only used for ground truth info. Keypoints and colors are defined in templates.

Parameters

- **image** (*PIL Image*) – a PIL image.
- **annotations** (*list*) – a list of keypoint annotation data.
- **templates** (*list*) – a list of keypoint templates.
- **visual_width** (*int*) – the width of the visual elements

Returns a PIL image with keypoints drawn.

Return type PIL Image

`datasetinsights.stats.`**`rotation_plot`**(*df*, *x*, *y*, *z=None*, *max_samples=None*, *title=None*, ***kwargs*)

Create a plotly 3d rotation plot :param df: A pandas dataframe that contains the raw data. :type df: pd.DataFrame :param x: The column name containing x rotations. :type x: str :param y: The column name containing y rotations. :type y: str :param z: The column name containing z rotations. :type z: str, optional :param title: The title of this plot. :type title: str, optional

This method can also take addition keyword arguments that can be passed to [plotly.graph_objects.Scatter3d](https://plotly.com/python-api-reference/generated/plotly.graph_objects.Scatter3d.html) method.

Returns A plotly.graph_objects.Figure containing the scatter plot

3.2 Synthetic Dataset Schema

Synthetic datasets generated by the Perception package are captured in JSON. This document describes the schema used to store the data. This schema provides a generic structure for simulation output which can be easily consumed to show statistics or train machine learning models. Synthetic datasets are composed of sensor captures, annotations, and metrics e.g. images and 2d bounding box labels. This data comes in various forms and might be captured by different sensors and annotation mechanisms. Multiple sensors may be producing captures at different frequencies. The dataset organizes all of the data into a single set of JSON files.

3.2.1 Goals

- It should include captured sensor data and annotations in one well-defined format. This allows us to maintain a contract between the Perception package and the dataset consumers (e.g. Statistics and ML Modeling…)
- It should maintain relationships between captured data and annotations that were taken by the same sensor at the same time. It should also maintain relationships between consecutive captures for time-related perception tasks (e.g. object tracking).
- It should support streaming data, since the data will be created on the fly during the simulation from multiple processes or cloud instances.
- It should be able to easily support new types of sensors and annotations.
- It assumes the synthetic dataset are captured in a directory structure, but does not make assumptions about transmission of storage of the dataset on a particular database management system.

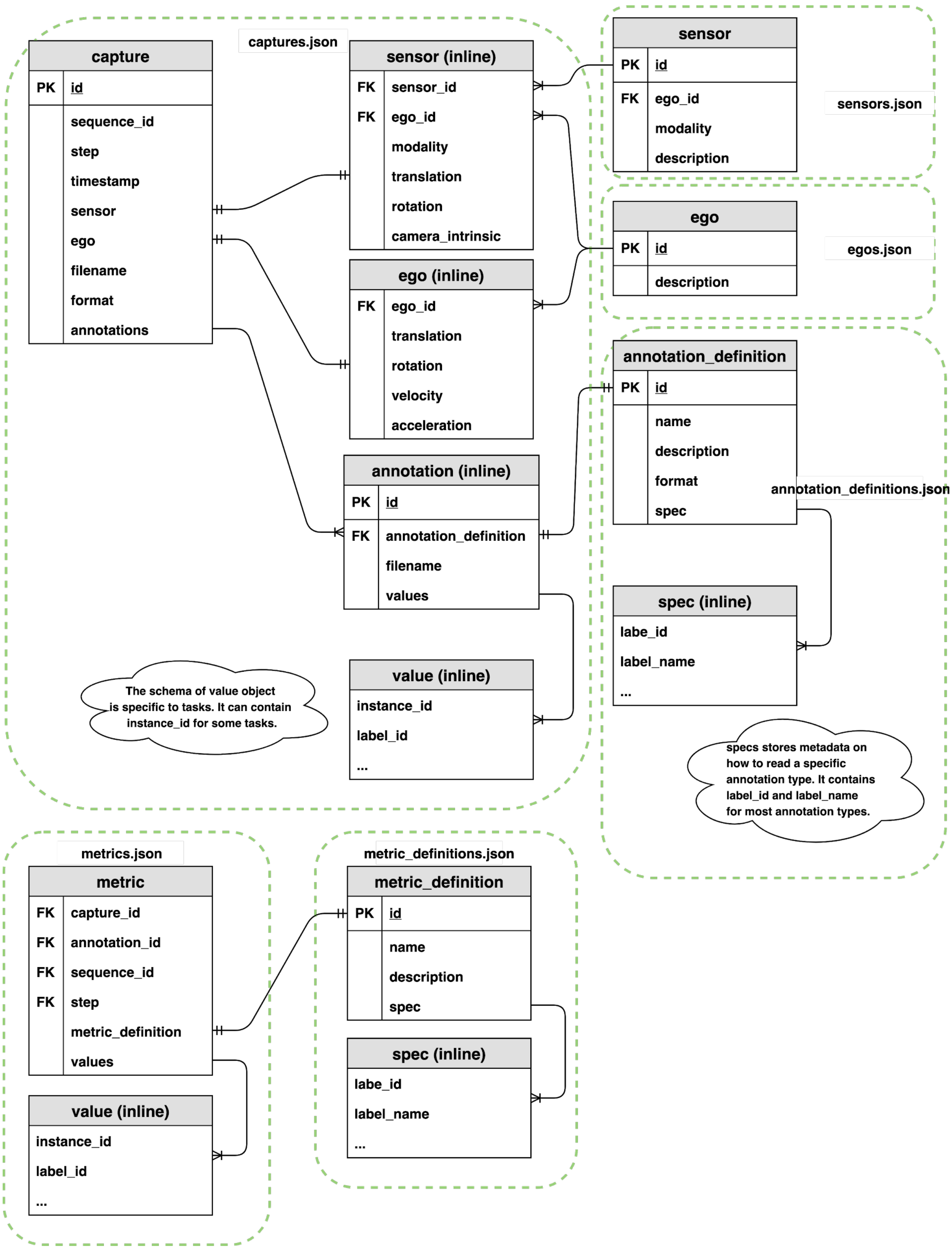
3.2.2 Terminology

- **simulation**: one or more executions of a Unity player build, potentially with different parameterization.
- **capture**: a full rendering process of a Unity sensor which saved the rendered result to data files e.g. (PNG, pcd, etc).
- **sequence**: a time-ordered series of captures generated by a simulation.
- **annotation**: data (e.g. bounding boxes or semantic segmentation) recorded that is used to describe a particular capture at the same timestamp. A capture might include multiple types of annotations.
- **step**: id for data-producing frames in the simulation.
- **ego**: a frame of reference for a collection of sensors (camera/LIDAR/radar) attached to it. For example, for a robot with two cameras attached, the robot would be the ego containing the two sensors.
- **label**: a string token (e.g. car, human.adult, etc.) that represents a semantic type, or class. One GameObject might have multiple labels used for different annotation purposes. For more on adding labels to GameObjects, see labeling.
- **coordinate systems**: there are three coordinate systems used in the schema
  - **global coordinate system**: coordinate with respect to the global origin in Unity.
  - **ego coordinate system**: coordinate with respect to an ego object. Typically, this refers to an object moving in the Unity scene.
  - **sensor coordinate system**: coordinate with respect to a sensor. This is useful for ML model training for a single sensor, which can be transformed from a global coordinate system and ego coordinate system. Raw value of object pose using the sensor coordinate system is rarely recorded in simulation.

3.2.3 Design

The schema is based on the nuScenes data format. The main difference between this schema and nuScenes is that we use **document based schema design** instead of **relational database schema design**. This means that instead of requiring multiple id-based "joins" to explore the data, data is nested and sometimes duplicated for ease of consumption.

3.2.4 Components



captures

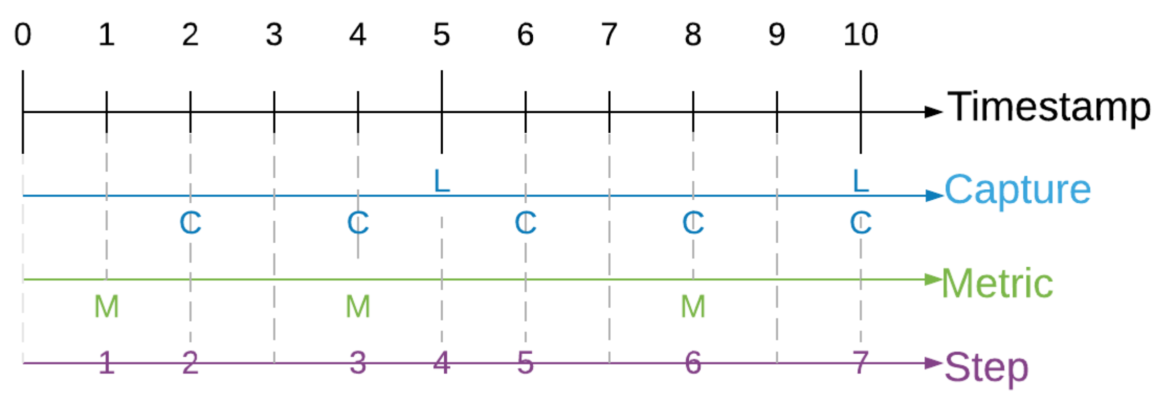
A capture record contains the relationship between a captured file, a collection of annotations, and extra metadata that describes the state of the sensor.

```
capture {
  id:                  <str> -- UUID of the capture.
  sequence_id:         <str> -- UUID of the sequence.
  step:                <int> -- The index of capture in the sequence. This field is␣
↪used to order of captures within a sequence.
  timestamp:           <int> -- Timestamp in milliseconds since the sequence started.
  sensor:              <obj> -- Attributes of the sensor. see below.
  ego:                 <obj> -- Ego pose of this sample. See below.
  filename:            <str> -- A single file that stores sensor captured data. (e.g.␣
↪camera_000.png, points_123.pcd, etc.)
  format:              <str> -- The format of the sensor captured file. (e.g. png, pcd,␣
↪etc.)
  annotations:         [<obj>,...] [optional] -- List of the annotations in this␣
↪capture. See below.
}
```

sequence, step and timestamp

In some use cases, two consecutive captures might not be related in time during simulation. For example, if we generate randomly placed objects in a scene for X steps of simulation. In this case, sequence, step and timestamp are irrelevant for the captured data. We can add a default value to the sequence, step and timestamp for these types of captures.

In cases where we need to maintain time order relationship between captures (e.g. a sequence of camera capture in a 10 second video) and *metrics*, we need to add a sequence, step and timestamp to maintain the time ordered relationship of captures. Sequence represents the collection of any time ordered captures and annotations. Timestamps refer to the simulation wall clock in milliseconds since the sequence started. Steps are integer values which increase when a capture or metric event is triggered. We cannot use timestamps to synchronize between two different events because timestamps are floats and therefore make poor indices. Instead, we use a "step" counter which make it easy to associate metrics and captures that occur at the same time. Below is an illustration of how captures, metrics, timestamps and steps are synchronized.



Since each sensor might trigger captures at different frequencies, at the same timestamp we might contain 0 to N captures, where N is the total number of sensors included in this scene. If two sensors are captured at the same timestamp, they should share the same sequence, step and timestamp value.

Physical camera sensors require some time to finish exposure. Physical LIDAR sensor requires some time to finish one 360 scan. How do we define the timestamp of the sample in simulation? Following the nuScene sensor synchronization strategy, we define a reference line from ego origin to the ego's "forward" traveling direction. The timestamp of the LIDAR scan is the time when the full rotation of the current LIDAR frame is achieved. A full rotation is defined as the 360 sweep between two consecutive times passing the reference line. The timestamp of the camera is the exposure trigger time in simulation.

capture.ego

An ego record stores the ego status data when a sample is created. It includes translation, rotation, velocity and acceleration (optional) of the ego. The pose is with respect to the **global coordinate system** of a Unity scene.

```
ego {
  ego_id:       <str> -- Foreign key pointing to ego.id.
  translation:  <float, float, float> -- Position in meters: (x, y, z) with respect␣
↪to the global coordinate system.
  rotation:     <float, float, float, float> -- Orientation as quaternion: w, x, y, z.
  velocity:     <float, float, float> -- Velocity in meters per second as v_x, v_y, v_
↪z.
  acceleration: <float, float, float> [optional] -- Acceleration in meters per second^
↪2 as a_x, a_y, a_z.
}
```

capture.sensor

A sensor record contains attributes of the sensor at the time of the capture. Different sensor modalities may contain additional keys (e.g. field of view FOV for camera, beam density for LIDAR).

```
sensor {
  sensor_id:        <str> -- Foreign key pointing to sensor.id.
  ego_id:           <str> -- Foreign key pointing to ego.id.
  modality:         <str> {camera, lidar, radar, sonar,...} -- Sensor modality.
  translation:      <float, float, float> -- Position in meters: (x, y, z) with␣
↪respect to the ego coordinate system. This is typically fixed during the simulation,
↪ but can allow small variation for domain randomization.
  rotation:         <float, float, float, float> -- Orientation as quaternion: (w, x,␣
↪y, z) with respect to ego coordinate system. This is typically fixed during the␣
↪simulation, but we can allow small variation for domain randomization.
  camera_intrinsic: <3x3 float matrix> [optional] -- Intrinsic camera calibration.␣
↪Empty for sensors that are not cameras.

  # add arbitrary optional key-value pairs for sensor attributes
}
```

reference: camera_intrinsic

capture.annotation

An annotation record contains the ground truth for a sensor either inline or in a separate file. A single capture may contain many annotations.

```
annotation {
  id:                     <str> -- UUID of the annotation.
  annotation_definition: <int> -- Foreign key which points to an annotation_
↪definition.id. see below
  filename:              <str> [optional] -- Path to a single file that stores␣
↪annotations. (e.g. sementic_000.png etc.)
  values:                [<obj>,...] [optional] -- List of objects that store␣
↪annotation data (e.g. polygon, 2d bounding box, 3d bounding box, etc). The data␣
↪should be processed according to a given annotation_definition.id.
}
```

Example annotation files

semantic segmentation - grayscale image

A grayscale PNG file that stores integer values (label pixel_value in *annotation spec* reference table, semantic segmentation) of the labeled object at each pixel.

capture.annotation.values

2D bounding box

Each bounding box record maps a tuple of (instance, label) to a set of 4 variables (x, y, width, height) that draws a bounding box. We follow the OpenCV 2D coordinate system where the origin (0,0), (x=0, y=0) is at the top left of the image.

```
bounding_box_2d {
  label_id:     <int> -- Integer identifier of the label
  label_name:   <str> -- String identifier of the label
  instance_id:  <str> -- UUID of the instance.
  x:            <float> -- x coordinate of the upper left corner.
  y:            <float> -- y coordinate of the upper left corner.
  width:        <float> -- number of pixels in the x direction
  height:       <float> -- number of pixels in the y direction
}
```

metrics

Metrics store extra metadata that can be used to describe a particular sequence, capture or annotation. Metric records are stored as an arbitrary number (M) of key-value pairs. For a sequence metric, capture_id, annotation_id and step should be null. For a capture metric, annotation_id can be null. For an annotation metric, all four columns of sequence_id, capture_id, annotation_id and step are not null.

Metrics files might be generated in parallel from different simulation instances.

```
metric {
  capture_id:        <str> -- Foreign key which points to capture.id.
  annotation_id:     <str> -- Foreign key which points to annotation.id.
  sequence_id:       <str> -- Foreign key which points to capture.sequence_id.
  step:              <int> -- Foreign key which points to capture.step.
  metric_definition: <int> -- Foreign key which points to metric_definition.id
  values:            [<obj>,...] -- List of all metric records stored as json objects.
}
```

definitions

Ego, sensor, annotation, and metric definition tables are static during the simulation. This typically comes from the definition of the simulation and are generated during the simulation.

egos.json

A json file containing a collection of egos. This file is an enumeration of all egos in this simulation. A specific object with sensors attached to it is a commonly used ego in a driving simulation.

```
ego {
  id:           <str> -- UUID of the ego.
  description:  <str> [optional] -- Ego instance description.
}
```

sensors.json

A json file containing a collection of all sensors present in the simulation. Each sensor is assigned a unique UUID. Each is associated with an ego and stores the UUID of the ego as a foreign key.

```
sensor {
  id:                 <str> -- UUID of the sensor.
  ego_id:             <int> -- Foreign key pointing to ego.id.
  modality:           <str> {camera, lidar, radar, sonar,...} -- Sensor modality.
  description:        <str> [optional] -- Sensor instance description.
}
```

annotation_definitions.json

A json file containing a collection of annotation specifications (annotation_definition). Each record describes a particular type of annotation and contains an annotation-specific specification describing how annotation data should be mapped back to labels or objects in the scene.

Typically, the `spec` key describes all labels_id and label_name used by the annotation. Some special cases like semantic segmentation might assign additional values (e.g. pixel value) to record the mapping between label_id/label_name and pixel color in the annotated PNG files.

```
annotation_definition {
  id:           <int> -- Integer identifier of the annotation definition.
  name:         <str> -- Human readable annotation spec name (e.g. sementic_
↪segmentation, instance_segmentation, etc.)
  description:  <str, optional> -- Description of this annotation specifications.
  format:       <str> -- The format of the annotation files. (e.g. png, json, etc.)
  spec:         [<obj>...] -- Format-specific specification for the annotation values␣
↪(ex. label-value mappings for semantic segmentation images)
}

# semantic segmentation
annotation_definition.spec {
  label_id:          <int> -- Integer identifier of the label
  label_name:        <str> -- String identifier of the label
  pixel_value:       <int> -- Grayscale pixel value
  color_pixel_value: <int, int, int> [optional] -- Color pixel value
}

# label enumeration spec, used for annotations like bounding box 2d. This might be a␣
↪subset of all labels used in simulation.
annotation_definition.spec {
  label_id:    <int> -- Integer identifier of the label
  label_name:  <str> -- String identifier of the label
}
```

metric_definitions.json

A json file that stores collections of metric specifications records (metric_definition). Each specification record describes a particular metric stored in *metrics* values. Each metric_definition record is assigned a unique identifier to a collection of specification records, which is stored as a list of key-value pairs. The design is very similar to *annotation_definitions*.

```
metric_definition {
  id:           <int> -- Integer identifier of the metric definition.
  name:         <str> -- Human readable metric spec name (e.g. object_count, average␣
↪distance, etc.)
  description:  <str, optional> -- Description of this metric specifications.
  spec:         [<obj>...] -- Format-specific specification for the metric values
}

# label enumeration spec, used to enumerate all labels. For example, this can be used␣
↪for object count metrics.
metric_definition.spec {
  label_id:    <int> -- Integer identifier of the label
  label_name:  <str> -- String identifier of the label
}
```

schema versioning

- The schema uses semantic versioning.
- Version info is placed at root level of the json file that holds a collection of objects (e.g. captures.json, metrics.json, annotation_definitions.json,...). All json files in a dataset will share the same version. It should change atomically across files if the version of the schema is updated.
- The version should only change when the Perception package changes (and even then, rarely). Defining new metric_definitions or annotation_definitions will not change the schema version since it does not involve any schema changes.

3.2.5 example

A mockup of synthetic dataset according to this schema can be found here.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER FIVE

CITATION

If you find this package useful, consider citing it using:

```
@misc{datasetinsights2020,
    title={Unity {D}ataset {I}nsights Package},
    author={{Unity Technologies}},
    howpublished={\url{https://github.com/Unity-Technologies/datasetinsights}},
    year={2020}
}
```

PYTHON MODULE INDEX

d

INDEX

E

F

G

H

I

K

L

M

N

O

P

R

S

T