lidario

Joffrey Bienvenu

Dec 12, 2020

GETTING STARTED

Lidario is, a high-level python toolbox to manipulate LIDAR raster and point cloud.

CHAPTER

ONE

INSTALLATION

1.1 Dependencies

Lidario depends on Rasterio, which depend on many other Python and C libraries. In case of problem, please refer to the Rasterio installation instructions.

1.2 Install from Pypi

Install and update using pip:

```
pip install lidario
```

CHAPTER

TWO

QUICKSTART

lidario.Translator translate a given data structure (ie: *a raster*), to a point cloud (ie: *a numpy array*).

```
import lidario as lio

# Translate a raster to a numpy point cloud.
translator = lio.Translator("geotiff", "np")
point_cloud = translator.translate("/path/to/file.tif")

# point_cloud: np.array([...])
```

In this example, we initialize a **Translator** object to convert a geotiff file into a numpy array cloud point. Then, we use this object to effectively convert a tif file.

CHAPTER

THREE

ABOUT THE AUTHOR

Joffrey Bienvenu, Machine Learning student @ Becode.

- Website: https://joffreybvn.be
- Twitter: joffreybvn
- Github: https://github.com/Joffreybvn

CHAPTER

FOUR

METADATAREADER

These examples illustrate how to use lidario.MetadataReader to get the metadata of a raster.

4.1 Get metadata from tif file

Retrieve the metadata from a raster (**.tif**) file.

```
import lidario as lio

# Instantiate a MetadataReader object which will
# take a tif file
reader = lio.MetadataReader("tif")

# Get the metadata of a given tif file
metadata = reader.get_metadata("./tests/assets/1.tif")
```

4.2 Get metadata from Rasterio.mask

Retrieve the metadata from a rasterio.mask.

```
import rasterio
from rasterio.mask import mask
import lidario as lio

# Instantiate a MetadataReader object which will
# take a rasterio.mask
reader = lio.MetadataReader("mask")

# Load a raster and create a polygon shape
reader = rasterio.open("/path/to/file.tif")
shape = [{'type': 'Polygon', 'coordinates': [[(0, 0),  (0, 10), (10, 10), (0, 0)]]}]

# Crop the tif file with the shape
mask_values = rasterio.mask.mask(reader, shapes=shape, crop=True)

# Translate the mask_values and get the np.array
metadata = reader.get_metadata(mask_values)
```

CHAPTER

FIVE

TRANSLATOR

These examples illustrate how to use lidario.Translator to translate point cloud data.

5.1 Tif file to Pandas dataframe

Transform raster (**.tif**) file into a pandas.Dataframe.

```
import lidario as lio

# Instantiate a Translator object which take a tif file
# and return a dataframe.
translator = lio.Translator("geotiff", "dataframe")

# Translate the tif file and get the pandas.Dataframe
point_cloud = translator.translate("/path/to/file.tif")
```

5.2 Rasterio.mask to Numpy array

Transform a rasterio.mask into a Numpy array.

```
import rasterio
from rasterio.mask import mask
import lidario as lio

# Instantiate a Translator object which take rasterio.mask
# and return a numpy array.
translator = lio.Translator("mask", "numpy")

# Load a raster and create a polygon shape
reader = rasterio.open("/path/to/file.tif")
shape = [{'type': 'Polygon', 'coordinates': [[(0, 0),  (0, 10), (10, 10), (0, 0)]]}]

# Crop the tif file with the shape
mask_values = rasterio.mask.mask(reader, shapes=shape, crop=True)

# Translate the mask_values and get the np.array
point_cloud = translator.translate(mask_values)
```

5.3 Translate to CSV and get metadata

Transform a raster (**.tif**) file into a CSV without applying the affine geo-transformation, and get the metadata.

```
import lidario as lio

# Instantiate a Translator object which take a tif file,
# save the point cloud to a CSV and return the metadata.
translator = lio.Translator("geotiff", "csv", affine_transform=False, metadata=True)

# With metadata=True, translator return a tuple with
# the point cloud and the metadata.
point_cloud, metadata = translator.translate("/path/to/file.tif")
```

In this case, the point_cloud is None, because we save the values to a CSV file.

5.4 Translate to PLY file

Transform a raster (**.tif**) file into a PLY (.ply) file.

```
import lidario as lio

# Instantiate a Translator object which take a tif file
# and the point cloud to a PLY file.
translator = lio.Translator("geotiff", "ply")

# Translate the tif to a binary .ply file
translator.translate("/path/to/file.tif", out_format="binary")

# Translate the tif to a text .ply file (may be slow !)
translator.translate("/path/to/file.tif", out_format="ascii")
```

CHAPTER

SIX

LIDARIO.METADATAREADER

Open a raster (.tif) or a rasterio.mask and return the metadata as a dictionary. See the examples for more details about how to use this class.

class lidario.**MetadataReader**(*input_type*)

Instantiate a MetadataReader object which will handle the metadata retrieval from the given input.

Parameters **input_type** (*str*) – Type of raster data provided: "**geotiff**" or "**mask**".

- "geotiff": a .tif raster file.
- "mask", a *rasterio.mask.mask()* result.

get_metadata(*input_values*)

Retrieve and return the metadata from a given "input_values".

Parameters **input_values** – Data values to translate. Depend on the Translator's "input_type" parameter:

- For a "**geotiff**": Takes the path to your .tif file (string).
- For a "**mask**": Takes the np.array returned by a rasterio.mask.mask() method.

Returns A dictionary of the metadata.

Return type dict

CHAPTER

SEVEN

LIDARIO.TRANSLATOR

Convert a raster (.tif) or a rasterio.mask into a point cloud (x, y, z). See the examples for more details about how to use this class.

class lidario.**Translator**(*input_type*, *output_type*, *affine_transform=True*, *metadata=False*)

Instantiate a Translator object which will handle the translation between given input and desired output type.

Parameters

- **input_type** (*str*) – Type of raster data provided: "**geotiff**" or "**mask**".
  - "geotiff": a .tif raster file.
  - "mask", a *rasterio.mask.mask()* result.
- **output_type** (*str*) – Type of point cloud data to return: "**csv**", "**numpy**", "**pandas**", "**dictionary**", "**list**", "**tuple**".
  - "csv": a CSV file.
  - "ply": a .ply file.
  - "numpy": a Numpy array. Alternatives: "np", "array".
  - "dataframe": A Pandas dataframe: Alternatives: "pandas", "pd", "df".
  - "dictionary": A pure Python dictionary: Alternative: "dict".
  - "list" a pure Python list.
  - "tuple": a pure Python tuple.
- **affine_transform** (*bool, optional*) – If True (default), apply an affine geo-transformation to the translated coordinates.
- **metadata** (*bool, optional*) – If True, the "translate" method will return a tuple with the point cloud and the metadata. If False (default), it will only return the point cloud.

translate(*input_values*, *out_file='output1.csv'*, *out_format='binary'*, *no_data=None*, *decimal=None*, *transpose=False*, *band=1*)

Translate a given "input_values" into a X, Y, Z point cloud.

Parameters

- **input_values** (*str or np.array*) – Data values to translate. Depend on the Translator's "input_type" parameter:
  - For a "**geotiff**": Takes the path to your .tif file (string).
  - For a "**mask**": Takes the np.array returned by a rasterio.mask.mask() method.
- **out_file** (*str, optional*) – Name of the file to save the point cloud. Used only if the Translator's "output_type" is a file type: "csv", "ply". Optional, default: "output.csv".

- **out_format** – Data format to save the file: "**binary**" (default) or "**ascii**" (not recommended, may be slow). Used only when "ply" is specified as "output_type". Optional.
- **no_data** (*int, optional*) – Value to exclude from the translation.
  - For a "**geotiff**": By default, use the nodata value stored in the tif file. If this value is missing, use -9999.
  - For a "**mask**": By default, use -9999.
- **band** (*bool, optional*) – Band of the raster to translate. Used only if Translator's "input_values" is "geotiff". Default: 1.
- **decimal** (*int, optional*) – Round the coordinate numbers to the given decimal. Default: None.
- **transpose** (*bool, optional*) – If True, transpose the coordinates. Default: False.
- **out_format** – str, optional

Returns The translated point cloud, typed as specified. If Translator's "output_type" is set to "csv", return None instead and save the CSV file. If Translator's "metadata" is set to True, return a tuple with the point cloud and the metadata.

INDEX