EVTech Documentation

Release 1.1.0

David Nilosek

Jul 27, 2020

CONTENTS:

CHAPTER

ONE

EVTECH

Simple tools for working with data provided by Eagleview for the RIT Hack.tiff 2020 Hackathon

- Free software: MIT license
- Documentation: https://evtech.readthedocs.io.

1.1 Installation

1.1.1 Stable release

To install EVTech, run this command in your terminal:

```
$ pip3 install evtech
```

This is the preferred method to install EVTech, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

1.1.2 From sources

The sources for EVTech can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/dnilosek/evtech
```

Or download the tarball:

```
$ curl -OJL https://github.com/dnilosek/evtech/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

1.2 Usage

To use EVTech in a project:

```
import evtech
```

In order to load a dataset (a single collection of images):

```
# Load the cameras
nadirs, obliques = evtech.load_dataset('/path/to/dataset')

# Get the image data for the first nadir camera as a numpy array
nadir_cam = nadirs[0]
img = nadir_cam.load_image()
```

The camera object also stores the average elevation of the image and the geographical bounds of the image, which can be retreived as a Shapely polgon:

```
# Get average elevation for image
elev = nadir_cam.elevation

# Get bounding polygon
bounds = nadir_cam.get_bounds()

# Create geojson from bounds using shapely, json
from shapely.geometry import mapping
import json

geo_json = json.dumps(mapping(bounds))
```

From here you can also look at operations with the camera, such as projecting a ray from the camera at a given pixel. Also projecting a latitude, longitude, elevation point into the camera to get the pixel location:

```
image_pt = nadir_cam.project_to_camera(lon, lat, elevation)
ray = nadir_cam.project_from_camera(col, row)
```

Rays can be used with ground elevation to find the intersection of the pixel and the ground:

```
ground_point_at_pxiel = ray.intersect_at_elevation(nadir_cam.elevation)
```

We have provided a simple single-image height measurement function that uses the camera and two points to compute the height of an object at a given elevation:

```
height = nadir_cam.height_between_points(base_img_pt, peak_image_pt, nadir_cam.
↪elevation)
```

Lastly, we have proivded a function to take multiple cameras and associated image points, and triangulate a three dimensional point:

```
# Load cameras from dataset
cam1 = nadirs[0]
cam2 = obliques[1]
```

(continues on next page)

(continued from previous page)

```
cam3 = obliques[2]
cams = [cam1, cam2, cam3]

# Points from images associated with cam1, cam2, cam3
pt1 = [605,171]
pt2 = [304,536]
pt3 = [879,441]
pts = [pt1, pt2, pt3]

world_pt = evtech.triangulate_point_from_cameras(cams, pts)
```

OpenCV has a number of tools that can be used for image manipulation and display, refer to the imgproc and highgui packages. Note that these are c++ bindings, you can find many examples that may be helpful on how to use the OpenCV Python bindings here:

```
import cv2

# Get image
img = nadir_cam.load_image()

# Display an image
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)
k = cv2.waitKey(0)

# wait for ESC key to exit
if k == 27:
    cv2.destroyAllWindows()
```

1.3 API

1.3.1 Dataset

evtech.dataset.**load_dataset**(*dir_path*, *loader=<function camera_from_json>*)

Loads a dataset into two arrays of cameras

Parameters

- **dir_path** (*string*) – Path to the dataset
- **loader** (*function*) – function(str, str), optional, defaults to camera_from_json

Returns A tuple with list of nadir cams and list of oblique cams

Return type tuple: list,list

1.3.2 Camera

Camera class for evtech.

class evtech.camera.**Camera**(*proj*, *bounds*, *cen*, *geo_bounds*, *elev*, *crs*, *image_path*)

This class represents camera information for a given image and allows for world<->camera interactions

Parameters

- **proj** (class: *numpy.array*) – The 3x4 projection matrix for the camera
- **bounds** (*list*) – The bounds of the image chip within the larger image [x_min, y_min, x_max, y_max]
- **cen** (*list*) – The center of the camera [x, y, z]
- **geo_bounds** (*list*) – The geographic bounds of the image in lat/lon [x_min, y_min, x_max, y_max]
- **elev** (*float*) – The average elevation of the image
- **crs** (class: *pyproj.CRS*) – The coordinate system for the projection matrix (Must be linear such as UTM)
- **image_path** – The filepath to the image data

get_bounds()

Get the bounds of the camera

Returns The bounds of the image on the ground

Return type class: *shapely.Polygon*

get_elevation()

Get the average ground elevation of the image

Returns The average elevation of the ground in the frame

Return type float

height_between_points(*base_point*, *peak_point*, *elev=None*)

Compute the height between two image points, given the elevation of the base point. If no elevation is passed the stored elevation will be used.

Parameters

- **base_point** (*list*) – The image point at the given elevation
- **peak_point** (*list*) – The image point to compute the height at
- **elev** (*float, optional*) – The associated elevation of the base point, defaults to None

load_image(*loader=<built-in function imread>*)

Load the image for this camera

Parameters **loader** (*function, optional*) – A function to load the image, defaults to cv2.imread

Returns image data

Return type numpy.array

project_from_camera(*col*, *row*)

Project a ray from the camera

Parameters

- **col** (*float*) – The column index of the pixel to project
- **row** (*float*) – The row index of the pixel to project

Returns A ray from the camera

Return type class: *evtech.Ray*

project_to_camera(*lon*, *lat*, *elevation*)
Project a lat/lon/elevation point into the image

Parameters

- **lat** (*float*) – The latitiude
- **lon** (*float*) – The longitiude
- **elevation** (*float*) – The elevation

Returns The row, col value of the pixel

Return type class: *np.Array*

set_path(*image_path*)
Mutator to set path data member

Parameters **path** (*string*) – Path to image data

to_full_image(*col*, *row*)
Convert an image point from the subset image to the full image

Parameters

- **col** (*float*) – The column to offset
- **row** (*float*) – The row to offset

Returns The offset row,col

Return type tuple

evtech.camera.**camera_from_json**(*json_data*, *image_path=''*)
Generate a camera from the seralized JSON data

Parameters

- **json_data** (*dict*) – The json data loaded from the serlized JSON
- **image_path** (*str, optional*) – The path to the associated image data

Returns A camera object

Return type evtech.Camera

1.3.3 Ray

Ray class for evtech.

class evtech.ray.**Ray**(*origin*, *direction*, *crs*)
A class to represent rays in three dimensional space

Parameters

- **origin** (*class: list*) – A 3 elemnt list representing the origin of the ray
- **direction** (*list*) – A 3 element list representing the direction of the ray
- **crs** (class: *pyproj.CRS*) – The CRS for the coordinates of the ray

depth_at_elevation(*elevation*)
Return the depth at a given elevation

Parameters **elevation** (*float*) – The elevation to get the depth of

intersect_at_elevation(*elevation*, *latlng=True*)
Return a three dimensional point intersected at a given elevation

Parameters

- **elevation** (*float*) – The elevation to intersect
- **latlng** (*bool, optional*) – Return the point as lat,lng, elevation, defaults to True

Returns A 3d point

Return type numpy.array

point_at_depth(*depth*)
Return a 3D point at a given depth along the ray

Parameters **depth** (*float*) – The depth along the ray

Returns A 3d point

Return type numpy.array

1.3.4 Geodesy

Functions for converting coordinates

evtech.geodesy.**utm_crs_from_latlon**(*lat*, *lon*)
Determines the UTM CRS from a given lat lon point

Parameters

- **lat** (*float*) – The latitude
- **lon** (*float*) – The longitude

Returns A coordinate system for the associated UTM zone

Return type class:*pyroj.CRS*

1.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

1.4.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/dnilosek/evtech/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

EVTech could always use more documentation, whether as part of the official EVTech docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/dnilosek/evtech/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

1.4.2 Get Started!

Ready to contribute? Here's how to set up *evtech* for local development.

1. Fork the *evtech* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/evtech.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv evtech
$ cd evtech/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 evtech tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

1.4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.org/dnilosek/evtech/pull_requests and make sure that the tests pass for all supported Python versions.

1.4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_evtech
```

1.4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

1.5 Credits

1.5.1 Development Lead

- David Nilosek <david.nilosek@eagleview.com>

1.5.2 Contributors

None yet. Why not be the first?

1.6 History

1.6.1 1.1.0 (2020-01-24)

- Documentation fixes
- Added access to camera elevation

1.6.2 1.0.1 (2020-01-22)

- Documentation build fixes

1.6.3 1.0.0 (2020-01-22)

- Added ability to get shapely geometry of image bounds on ground
- Included example application for measuring height from an image

1.6.4 0.7.0 (2020-01-20)

- Added triangulation function
- Updated usage docs
- Fixed bug in project from camera

1.6.5 0.6.0 (2020-01-20)

- Added height from two points to camera class
- Added ray interesection with elevation

1.6.6 0.5.0 (2020-01-19)

- Added ray class
- Added project from camera to camera class

1.6.7 0.4.0 (2020-01-13)

- Added coverage checking.
- Added functions for reading a full dataset with nadirs and obliques

1.6.8 0.3.0 (2020-01-08)

- First release on PyPI.

1.7 Example

A simple tool that uses the height measurement tool to allow a user to measure height on the first oblique image in a dataset

This app uses the following dependecies:

- EVTech
- OpenCV

1.7.1 Demo

1.7.2 Source

```
import argparse
import evtech
import cv2

# Load arguments
parser = argparse.ArgumentParser()
parser.add_argument("-d",
                    "--dataset",
                    help="Location of dataset",
                    type=str)
args = parser.parse_args()

# Only care about obliques for this app
_, obliques = evtech.load_dataset(args.dataset)

# Load first oblique image
img = obliques[0].load_image()

# mouse callback function
```

(continues on next page)

(continued from previous page)

```
# For drawing
drawing = False # true if mouse is pressed
ix,iy = -1,-1
def draw_line(event,x,y,flags,param):
    global ix,iy,drawing,mode
    if event == cv2.EVENT_LBUTTONDOWN:
        if drawing:
            drawing = False
            # Draw line
            cv2.line(img,pt1=(ix,iy),pt2=(x,y),color=(0,0,255),thickness=3)

            # Compute height
            height = obliques[0].height_between_points([ix,iy],[x,y])

            # Computed in meters, convert to feet
            height *= 3.28084

            # Label line
            lbl = "{:.2f}".format(height) + " feet"
            cv2.putText(img, lbl, (x+10,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,255,
↪0), 1)
        else:
            drawing = True
            ix,iy = x,y

# Making Window For The Image
cv2.namedWindow("Image")
# Adding Mouse CallBack Event
cv2.setMouseCallback("Image",draw_line)

# Starting The Loop So Image Can Be Shown
while(True):
    cv2.imshow("Image",img)

    if cv2.waitKey(20) & 0xFF == ord('q'):
        break

cv2.destroyAllWindows()
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX