Stereographic Cloud Base Reconstruction Documentation

Release 1.0.0

Florian M. Savoy, Soumyabrata Dev, Yee Hui Lee and Stefan Winkler

May 09, 2017
Contents

1 Code
   1.1 Classes .............................................................. 3
   1.2 Sample usage .......................................................... 8
   1.3 Dependencies .......................................................... 9

2 License ............................................................................. 11

3 Indices and tables .......................................................... 13

Python Module Index ......................................................... 15
This is the documentation for the algorithm presented in the publication “Stereographic Cloud Base Reconstruction using High-Resolution Whole Sky Imagers”. Reference to the paper will be provided here upon publication.

This algorithm estimates 3D models of the base of clouds in real world coordinates. It uses two images captured at the same time by two whole sky imagers positioned about 100 meters away. The resulting models have applications in meteorology, solar energy prediction or satellite communication signal attenuation.

The figure below illustrates the algorithm. Both top images are the input images captured at the same time by two sky imagers about 100 meters apart. The left image has been segmented, and displays the feature points used for the reconstruction in green as well as the triangulation in red. The three figures at the bottom show the reconstructed model, from 3D, side and top views. The axis refer to real world coordinates in meters. Please refer to the publication for further details.
The code is available on github at https://github.com/FSavoy/cloud-base-reconstruction. It consists of three classes encapsulating functions regarding sky imagers, segmentation and the reconstruction itself. A sample file which generates the figure shown above is provided too.

The documentation is organized as follows:

## Classes

Three classes encapsulate the code logic:

### Imager

```python
class imager.Imager(name, center, radius, position, rot, trans, longitude=None, latitude=None, elevation=None)
```

Describes an imager and all the parameters related to it. Contains the methods dealing with lens distortions and mis-alignment correction.

- **name** `str` – Name of the imager
- **center** `list[int]` – List of two ints describing the center of the fish-eye lens in the captured pictures
- **radius** `int` – The radius of the circle created by the fish-eye lens
- **position** `numpy.array` – The coordinates of the relative position of the imager in real world coordinates (meters)
- **rot** `numpy.array` – A rotation matrix (3x3) to cope with the misalignment of the image
**trans**

`numpy.array` – A translation matrix (3x1) to cope with the misalignement of the image

**longitude**

`str` – The longitude coordinate of the imager (optional, only needed for segmentation)

**latitude**

`str` – The latitude coordinate of the imager (optional, only needed for segmentation)

**elevation**

`int` – The elevation of the integer in meters (optional, only needed for segmentation)

**cam2world**

Returns the 3D coordinates with unit length corresponding to the pixels locations on the image.

Parameters

- `m` (`numpy.array`) – 2 x `m` matrix containing `m` pixel coordinates

Returns

- 3 x `m` matrix of the corresponding unit sphere vectors

Return type

`numpy.array`

**undistort_image**

Undistort a fish-eye image to a standard camera image by a ray-tracing approach.

Parameters

- `image` (`numpy.array`) – the fish-eye image to undistort
- `sizeResult` (`list[int]`) – 2D size of the output undistorted image [pixels]
- `altitude` (`int`) – the altitude of the virtual image plane to form the undistorted image [meters]
- `bearing` (`float`) – the bearing angle of the undistorted plane (in degrees)
- `elevation` (`float`) – the elevation angle of the undistorted plane (in degrees)
- `pixelWidth` (`int`) – the width in real world coordinates of one pixel of the undistorted image [meters]

**world2cam**

Returns the pixel locations corresponding to the given 3D points.

Parameters

- `M` (`numpy.array`) – 3 x `m` matrix containing `M` point coordinates

Returns

- 2 x `m` matrix of the corresponding pixel coordinates

Return type

`numpy.array`

**Segmentation**

**class** `segmentation.Segmentation` (`image`, `imager`, `time`)

Contains all the methods needed for the segmentation a sky/cloud image

**image**

`np.array` – The image to segment

**imager**

`Imager` – The imager used to take the image

**time**

`datetime.datetime` – The time at which the image was captured (UTC+8)
**cmask** \((\text{index}, \text{radius}, \text{array})\)

Generates the mask for a given input image. The generated mask is needed to remove occlusions during post-processing steps.

**Parameters**

- **\text{index}** (numpy array) – Array containing the x- and y- co-ordinate of the center of the circular mask.
- **\text{radius}** (float) – Radius of the circular mask.
- **\text{array}** (numpy array) – Input sky/cloud image for which the mask is generated.

**Returns**  Generated mask image.

**Return type**  numpy array

**getBRchannel** \((\text{input\_image}, \text{mask\_image})\)

Extracts the ratio of red and blue blue channel from an input sky/cloud image. It is used in the clustering step to generate the binary sky/cloud image.

**Parameters**

- **\text{input\_image}** (numpy array) – Input sky/cloud image.
- **\text{mask\_image}** (numpy array) – Mask to remove occlusions from the input image. This mask contains boolean values indicating the allowable pixels from an image.

**Returns**  Ratio image using red and blue color channels, normalized to \([0,255]\).

**Return type**  numpy array

**make_cluster_mask** \((\text{input\_matrix}, \text{mask\_image})\)

Clusters an input sky/cloud image to generate the binary image and the coverage ratio value.

**Parameters**

- **\text{input\_matrix}** (numpy array) – Input matrix that needs to be normalized.
- **\text{mask\_image}** (numpy array) – Mask to remove occlusions from the input image. This mask contains boolean values indicating the allowable pixels from an image.

**Returns**  Binary output image, where white denotes cloud pixels and black denotes sky pixels. float: Cloud coverage ratio in the input sky/cloud image. float: The first (out of two) cluster center. float: The second (out of two) cluster center.

**Return type**  numpy array

**removeSunIfNonCloudy** \((\text{im\_mask})\)

Removes the circumsolar area from the mask if it is detected as non cloudy.

**Parameters**  **\text{im\_mask}** (numpy array) – Segmentation mask

**Returns**  Modified segmentation mask.

**Return type**  numpy array

**segment** ()

Main function for sky/cloud segmentation. Perform all the necessary steps.

**showasImage** \((\text{input\_matrix})\)

Normalizes an input matrix to the range \([0,255]\). It is useful in displaying the matrix as an image.

**Parameters**  **\text{input\_matrix}** (numpy array) – Input matrix that needs to be normalized.

**Returns**  Returns the normalized matrix.
Return type  numpy array

Reconstruction

class reconstruction.Reconstruction(imager1, imager2, time, sizeUndistorted=[1000, 1000, 3], altitudes=[500], pixelWidths=[1], tilts=[[0, 0], [35, 0], [35, 90], [35, 180], [35, 270]])

Contains all the methods for creating a 3D reconstruction from the base height from two sky pictures taken at the same time. Most functions need to be applied in the correct order to complete the workflow. See main.py for an example.

imager1
  Imager – Imager used to take the first image

imager2
  Imager – Imager used to take the second image

time
datetime.datetime – Time at which the images were captured

sizeUndistorted
  list[int] – Size of the undistorted planes

altitudes
  list[int] – Altitudes of the undistorted planes

pixelWidths
  list[int] – Widths in meters of one pixel of the undistorted plane

tilts
  list[list[int]] – List of angles [elevation, azimuth] of the undistorted planes

images
  dict[list[numpy.array]] – Dictionary (over each imager) of arrays of LDR images captured by that imager

undistorted
  dict[list[numpy.array]] – Dictionary of list of LDR undistorted planes for each combination of undistortion parameters for the first imager

tonemapped
  dict[dict[numpy.array]] – Dictionary (over each imager) of dictionaries of tonemapped images for each combination of undistortion parameters for that imager

pts
  dict[dict[numpy.array]] – Dictionary (over each imager) of dictionaries of n*2 feature points locations in the undistorted planes of the that imager

rays
  dict[dict[numpy.array]] – Dictionary (over each imager) of dictionary of 3*n vectors from the origin at the center of the imager towards each feature point in the planes for the that imager

ptsWorld
  dict[numpy.array] – Dictionary of 3*n’ coordinates of the reconstructed points, at the midpoint of the shortest segment between both rays from both imagers

ptsWorldMerged
  numpy.array – ptsWorld concatenated in a single array independently of undistortion planes

ptsWorldRays
  dict[dict[numpy.array]] – Dictionary (over each imager) of dictionary of 3*n’ coordinates of the reconstructed points, on the rays originating from the first imager
**ptsWorldRaysMerged**

`dict[dict[numpy.array]]` – Dictionary (over each imager) of the reconstructed points in ptsWorldRays (all undistortion planes merged in common arrays)

**ptsPictureImager**

`dict[dict[numpy.array]]` – Dictionary (over each imager) of dictionaries of `2*n'` vectors of the image plane locations corresponding to each reconstructed points for that imager

**ptsPictureImagerMerged**

`dict[numpy.array]` – Dictionary (over each imager) of the image locations in ptsPictureImager (all undistortion planes merged in common arrays)

**distances**

`dict[numpy.array]` – Dictionary of `n'` vectors of the lengths of the shortest segments between the rays of the first and the second imager

**distancesMerged**

`numpy.array` – distances concatenated in a single array independently of undistortion planes

**triangulationError**

`float` – Mean length of the shortest segments between rays for each pair of matching features

**meanHeight**

`float` – Mean z coordinate of the reconstructed points

**stdHeight**

`float` – Standard deviation of the z coordinates of the reconstructed points

**nbPts**

`int` – Number of reconstructed points

**cloudBaseHeight**

`float` – 10th percentile of the z coordinates of the reconstructed points, assumed to be the cloud base height

**segmentationMask**

`np.array` – Sky/cloud segmentation mask for the first imager

**triangulation**

`matplotlib.tri.Triangulation` – Delaunay triangulation between the reconstructed points

**computeMetrics()**

Computes the light rays from the pairs of matching points. `concatenateValues()` must be called before.

**computeRays()**

Computes the light rays from the pairs of matching points. `match()` must be called before.

**computeTriangulation()**

Creates the triangulation between reconstructed points for visualization. `concatenateValues()` must be called before.

**concatenateValues()**

Concatenates all the computed values from all the different planes into a single vectors. `intersectRays()` must be called before.

**intersectRays (threshold=100)**

Computes the intersection the light rays from the matching pairs. `computeRays()` must be called before.

- **Parameters**
  - `threshold (int)` – the minimum distance between two rays to discard a match.

**iter_planes()**

Helper function to iterate over all undistortion plane parameters (altitudes, pixel width and tilts).
Returns over all (altitude, pixel width, tilts) tuples

Return type iterator

load_images (images1, images2)

Loads the input LDR images captured at the same instant by both imagers.

Parameters

- images1 (list[numpy.array]) – LDR images captured by imager1
- images2 (list[numpy.array]) – LDR images captured by imager2

match()

Creates sift matching reconstruction between a pair of undistorted images. tonemap() must be called before.

process()

Executes the entire reconstruction process. Alternatively, every function can be called individually. load_images() must be called before.

removeOutliers()

Removes outliers in the set of reconstructed points intersectRays() must be called before.

tonemap()

Tonemaps the undistorted planes. undistort() must be called before.

undistort()

Generates the undistortion planes. load_images() must be called before.

writeFigure (filename)

Creates a visualization of the reconstructed elements. createTriangulation() must be called before.

- Imager: describes an imager and all the parameters related to it. Contains the methods dealing with lens distortions and mis-alignment correction.
- Segmentation: contains all the methods needed for the segmentation a sky/cloud image.
- Reconstruction: contains all the methods for creating a 3D reconstruction from the base height from two sky pictures taken at the same time.

Sample usage

The file main.py shows a sample usage for the code to generate the reconstruction of cloud base height. Here is a walkthrough the file:

First import the dependencies:

```python
from imager import Imager
from reconstruction import Reconstruction
import numpy as np
import cv2
import datetime
```

name3 = 'wahrsis3'
center3 = [1724, 2592]
radius3 = 1470
relativePosition3 = np.array([0,0,0])
calibRot3 = np.array([[0.99555536, 0.09404159, 0.00506982], [-0.09393761, 0.99541774, -0.01786745], [0.00672686, 0.01731178, 0.99982751]])
calibTrans3 = np.array([[0.00552915, [0.00141732], [0.00553584]])
longi2d3 = '103:40:49.9'
alatitude3 = '1:20:35'
altitude3 = 59
wahrsis3 = Imager(name3, center3, radius3, relativePosition3, calibRot3, calibTrans3, longitude3, latitude3, altitude3)

The second imager does not need the longitude, latitude and altitude information.

name4 = 'wahrsis4'
center4 = [2000, 2975]
radius4 = 1665
relativePosition4 = np.array([-2.334, 101.3731, -8.04])
calibRot4 = np.array([[0.9710936, -0.23401871, 0.04703662], [0.234924, 0.97190314, -0.01466276], [-0.04228367, 0.02528894, 0.99878553]])
calibTrans4 = np.array([[-0.00274625], [-0.00316865], [0.00516088]])
wahrsis4 = Imager(name4, center4, radius4, relativePosition4, calibRot4, calibTrans4)

Then load the LDR images in two arrays:
images3 = [cv2.imread('wahrsis3/2015-10-29-12-58-01-wahrsis3-low.jpg'),
           cv2.imread('wahrsis3/2015-10-29-12-58-01-wahrsis3-med.jpg'),
           cv2.imread('wahrsis3/2015-10-29-12-58-01-wahrsis3-high.jpg')]
images4 = [cv2.imread('wahrsis4/2015-10-29-12-58-01-wahrsis4-low.jpg'),
           cv2.imread('wahrsis4/2015-10-29-12-58-01-wahrsis4-med.jpg'),
           cv2.imread('wahrsis4/2015-10-29-12-58-01-wahrsis4-high.jpg')]

Create the Reconstruction object, load the images, process and write the result to out.png.
reconst = Reconstruction(wahrsis3, wahrsis4, datetime.datetime(2015,10,29,12,58,1))
reconst.load_images(images3, images4)
reconst.process()
reconst.writeFigure("out.png")

Dependencies

This project was built with python 2.7.12

Furthermore, the following packages were used:

<table>
<thead>
<tr>
<th>Package</th>
<th>Version</th>
<th>Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>openCV</td>
<td>3.1.0</td>
<td><a href="http://opencv.org/">http://opencv.org/</a></td>
</tr>
<tr>
<td>numpy</td>
<td>1.12.0</td>
<td><a href="http://www.numpy.org/">http://www.numpy.org/</a></td>
</tr>
<tr>
<td>scipy</td>
<td>0.18.1</td>
<td><a href="https://www.scipy.org/">https://www.scipy.org/</a></td>
</tr>
<tr>
<td>scikit-image</td>
<td>0.12.3</td>
<td><a href="http://scikit-image.org/">http://scikit-image.org/</a></td>
</tr>
<tr>
<td>scikit-learn</td>
<td>0.18.1</td>
<td><a href="http://scikit-learn.org/">http://scikit-learn.org/</a></td>
</tr>
<tr>
<td>matplotlib</td>
<td>2.0.0</td>
<td><a href="http://matplotlib.org/">http://matplotlib.org/</a></td>
</tr>
<tr>
<td>PyEphem</td>
<td>3.7.6.0</td>
<td><a href="http://rhodesmill.org/py">http://rhodesmill.org/py</a> ephem/</td>
</tr>
<tr>
<td>pycuda (optional)</td>
<td>2016.1.2</td>
<td><a href="https://documen.tician.de/">https://documen.tician.de/</a> pycuda/</td>
</tr>
</tbody>
</table>
License

This software is released under a 3-Clause BSD License. It is provided “as is” without any warranty. Users are required to obtain the licenses for the required dependencies.
CHAPTER 3

Indices and tables

• genindex
• modindex
• search
Python Module Index

i
imager, 3

r
reconstruction, 6

s
segmentation, 4
Index

A
altitudes (reconstruction.Reconstruction attribute), 6

C
cam2world() (imager.Imager method), 4
center (imager.Imager attribute), 3
cloudBaseHeight (reconstruction.Reconstruction attribute), 7
cmask() (segmentation.Segmentation method), 4
computeMetrics() (reconstruction.Reconstruction method), 7
computeRays() (reconstruction.Reconstruction method), 7
computeTriangulation() (reconstruction.Reconstruction method), 7
concatenateValues() (reconstruction.Reconstruction method), 7

distances (reconstruction.Reconstruction attribute), 7
distancesMerged (reconstruction.Reconstruction attribute), 7

elevation (imager.Imager attribute), 4

G
getBRchannel() (segmentation.Segmentation method), 5

I
image (segmentation.Segmentation attribute), 4
Imager (class in imager), 3
imager (module), 3
imager (segmentation.Segmentation attribute), 4
imager1 (reconstruction.Reconstruction attribute), 6
imager2 (reconstruction.Reconstruction attribute), 6
images (reconstruction.Reconstruction attribute), 6
intersectRays() (reconstruction.Reconstruction method), 7
inter Plaintes() (reconstruction.Reconstruction method), 7

L
latitude (imager.Imager attribute), 4
load_images() (reconstruction.Reconstruction method), 8
longitude (imager.Imager attribute), 4

M
make_cluster_mask() (segmentation.Segmentation method), 5
match() (reconstruction.Reconstruction method), 8
meanHeight (reconstruction.Reconstruction attribute), 7

N
name (imager.Imager attribute), 3
nbPts (reconstruction.Reconstruction attribute), 7

P
pixelWidths (reconstruction.Reconstruction attribute), 6
position (imager.Imager attribute), 3
process() (reconstruction.Reconstruction method), 8
pts (reconstruction.Reconstruction attribute), 6
ptsPictureImager (reconstruction.Reconstruction attribute), 7
ptsPictureImagerMerged (reconstruction.Reconstruction attribute), 7
ptsWorld (reconstruction.Reconstruction attribute), 6
ptsWorldMerged (reconstruction.Reconstruction attribute), 6
ptsWorldRays (reconstruction.Reconstruction attribute), 6
ptsWorldRaysMerged (reconstruction.Reconstruction attribute), 6

R
radius (imager.Imager attribute), 3
rays (reconstruction.Reconstruction attribute), 6
Reconstruction (class in reconstruction), 6
reconstruction (module), 6
removeOutliers() (reconstruction.Reconstruction method), 8
removeSunIfNonCloudy() (segmentation.Segmentation method), 5
rot (imager.Imager attribute), 3

S
segment() (segmentation.Segmentation method), 5
Segmentation (class in segmentation), 4
segmentation (module), 4
segmentationMask (reconstruction.Reconstruction attribute), 7
showasImage() (segmentation.Segmentation method), 5
sizeUndistorted (reconstruction.Reconstruction attribute), 6
stdHeight (reconstruction.Reconstruction attribute), 7

T
tilts (reconstruction.Reconstruction attribute), 6
time (reconstruction.Reconstruction attribute), 6
time (segmentation.Segmentation attribute), 4
tonemap() (reconstruction.Reconstruction method), 8
tonemapped (reconstruction.Reconstruction attribute), 6
trans (imager.Imager attribute), 3
triangulation (reconstruction.Reconstruction attribute), 7
triangulationError (reconstruction.Reconstruction attribute), 7

U
undistort() (reconstruction.Reconstruction method), 8
undistort_image() (imager.Imager method), 4
undistorted (reconstruction.Reconstruction attribute), 6

W
world2cam() (imager.Imager method), 4
writeFigure() (reconstruction.Reconstruction method), 8