Documentation new TSI software Documentation

Release 0.0.1

Job Mos

May 25, 2018

CONTENTS:

CHAPTER

ONE

INTRODUCTION

This is the introduction.

CHAPTER TWO

CODE EXAMPLE

Here is a Python function.

```python
def greet(name):
    print("Hello {}".format(name))
```

Here is a C function.

```c
int add(int a, int b) {
    return a + b;
}
```

```python
"""This is an example script."""
import sys

def greet(name):
    """Return greeting."""
    return "Hello {}!".format(name)

if __name__ == "__main__":
    name = sys.argv[1]
    print(greet(name))
```

CHAPTER

THREE

CLOUDDETECTION

3.1 color_bands module

`color_bands.`**`extract`**`(`*scaler*, *masked_img*`)`

Extract the red, green and blue bands from the masked image

Parameters

- **`scaler`** (*`int`*) – Maximum number of color levels, used in GLCM matrix
- **`masked_img`** (*`int`*) – The masked image

Returns Red, green and blue bands

Return type tuple

3.2 createmask module

`createmask.`**`calculate_band_position`**`(`*theta*`)`

Calculate the inner and outer position of the shadow band, required for drawing the shadow band mask line.

The formula of a circle is used: $x = r \cos \theta \wedge y = r \sin \theta$.

Parameters **`theta`** (*`float`*) – Azimuth of the sun with respect to the East. Normally, azimuth is measured from the North. However, to simplify calculations, the east was used.

Returns X and y locations of the inner and outer points of the shadow band

Return type tuple

`createmask.`**`create`**`(`*img*, *azimuth*`)`

Create the mask using the original image and the azimuth.

The mask consists of three parts:

- The circle bordering the hemispherical mirror.
- The shadow band.
- The camera and camera arm.

Parameters

- **`img`** (*`int`*) – Image in NumPy format
- **`azimuth`** (*`float`*) – Azimuth of the sun, taken from the properties file

Returns The masked image of shape (x_resolution,y_resolution,3) for an RGB image

Return type int

3.3 createregions module

createregions.**create**(*img*, *azimuth*, *altitude*)

Create the empty arrays and call drawing and masking functions

Parameters

- **img** (*int*) – image in NumPy format
- **azimuth** (*float*) – azimuth of the sun, taken from the properties file
- **altitude** (*float*) – altitude of the sun, taken from the properties file

Returns regions, outlines, labels, stencil, image_with_outlines

Return type tuple

createregions.**create_stencil**(*stencil*, *stencil_labels*)

Create the stencil which is used to mask the outside of the large circle

Parameters

- **stencil** (*int*) – empty stencil array in RGB format
- **stencil_labels** (*int*) – empty stencil array in scalar format

Returns stencil in both RGB and scalar format

Return type tuple

createregions.**draw_arm**(*regions*, *labels*, *image_with_outlines*)

Draw the camera arm mask

Parameters

- **regions** (*int*) – RGB representation of the segmented image
- **labels** (*int*) – Scalar (1,2,3,4) representation of the segmented image
- **image_with_outlines** (*int*) – image with outlines as overlay

Returns regions, labels, image_with_outlines

Return type tuple

createregions.**draw_band**(*regions*, *labels*, *image_with_outlines*, *theta*)

Draw the shadow band mask

Parameters

- **regions** (*int*) – RGB representation of the segmented image
- **labels** (*int*) – Scalar (1,2,3,4) representation of the segmented image
- **image_with_outlines** (*int*) – image with outlines as overlay
- **theta** (*float*) – azimuth measured from the East

Returns regions, labels, image_with_outlines

createregions.**draw_horizon_area**(*azimuth*, *regions*, *labels*, *outlines*)

Draw polygon which (when combined with other segments) makes up the horizon area

Parameters

- **azimuth** (*float*) – solar azimuth in degrees from North
- **regions** (*int*) – RGB representation of the segmented image
- **labels** (*int*) – Scalar (1,2,3,4) representation of the segmented image
- **outlines** (*int*) – RGB array of the segment outlines

Returns Regions, labels, outlines and angle θ describing the azimuth measured from the East

Return type tuple

createregions.**inner_circle**(*regions*, *labels*, *outlines*)

Draw the inner circle of the segmented image

The radius is smaller than the radius used in *createregions.large_circle()*

Parameters

- **regions** (*int*) – RGB representation of the segmented image
- **labels** (*int*) – Scalar (1,2,3,4) representation of the segmented image
- **outlines** (*int*) – RGB array of the segment outlines

Returns regions, labels, outlines

Return type tuple

createregions.**large_circle**(*regions*, *labels*, *outlines*)

Draw a circle centered in the middle of the image.

This circle has a radius slightly smaller than that of the mirror.

Parameters

- **regions** (*int*) – RGB representation of the segmented image
- **labels** (*int*) – Scalar (1,2,3,4) representation of the segmented image
- **outlines** (*int*) – RGB array of the segment outlines

Returns regions, labels, outlines

Return type tuple

createregions.**outer_circle**(*regions*, *labels*, *outlines*, *stencil*, *stencil_labels*)

Mask the outside of the large circle

Parameters

- **regions** (*int*) – RGB representation of the segmented image
- **labels** (*int*) – Scalar (1,2,3,4) representation of the segmented image
- **outlines** (*int*) – RGB array of the segment outlines
- **stencil** (*int*) – stencil array in RGB format
- **stencil_labels** (*int*) – stencil array in scalar format

Returns regions, labels, outlines, stencil (RGB), stencil (scalar)

Return type tuple

createregions.**overlay_outlines_on_image**(*img*, *outlines*, *stencil*)

Overlay outlines on image by converting to BW and performing several other operations

Parameters

- **img** (*int*) – image in NumPy format
- **outlines** (*int*) – RGB array of the segment outlines
- **stencil** (*int*) – stencil array in RGB format

Returns image with outlines as overlay

Return type int

createregions.**sun_circle**(*altitude*, *regions*, *labels*, *outlines*, *theta*)

Draw the sun cirlce segment

The position of the sun in the image plane is calculated using an approximation of the mirror. The function that is used to estimate the mirror geometry is $y = -0.23x + 1.25$.

The radial distance from the center of the image to the center of the sun can subsequently be calculated using the quadratic equation (abc formula).

Using the description of a circle (*createmask.calculate_band_position()*), the solar position is calculated.

Parameters

- **altitude** (*float*) – altitude of the sun, taken from the properties file
- **regions** (*int*) – RGB representation of the segmented image
- **labels** (*int*) – Scalar (1,2,3,4) representation of the segmented image
- **outlines** (*int*) – RGB array of the segment outlines
- **theta** (*float*) – azimuth measured from the East

Returns regions, labels, outlines

Return type tuple

3.4 labelled_image module

labelled_image.**calculate_pixels**(*labels*, *red_blue_ratio*, *threshold*)

Get amount of pixels in the four different areas to be used in postprocessing corrections

Parameters

- **labels** (*int*) – Scalar representation of the segmented image
- **red_blue_ratio** (*float*) – ratio of red/blue bands
- **threshold** (*float*) – fixed threshold of sunny/cloudy

Returns amount of sunny and cloudy pixels in each of the four regions

Return type tuple

3.5 main module

main.**main**()

Call processing functions and write output to file

3.6 myimports module

3.7 overlay module

overlay.**fixed**(*img*, *outlines*, *stencil*, *fixed_sunny_threshold*, *fixed_thin_threshold*)

Preprocess image to be compatible with *overlay.outlines_over_image()* using fixed thresholding

Parameters

- **img** (*int*) – image in NumPy format
- **outlines** (*int*) – RGB array of the segment outlines
- **stencil** (*int*) – stencil array in RGB format
- **fixed_sunny_threshold** (*float*) – threshold for sun/cloud
- **fixed_thin_threshold** (*float*) – threshold for thin/opaque cloud

Returns image with outlines

Return type int

overlay.**hybrid**(*img*, *outlines*, *stencil*, *threshold*)

Preprocess image to be compatible with *overlay.outlines_over_image()* using the hybrid threshold

Parameters

- **img** (*int*) – image in NumPy format
- **outlines** (*int*) – RGB array of the segment outlines
- **stencil** (*int*) – stencil array in RGB format
- **threshold** (*float*) – threshold for sun/cloud determined by HYbrid Thresholding Algorithm (HYTA)

Returns image with outlines

Return type int

overlay.**outlines_over_image**(*img*, *outlines*, *stencil*)

Overlay outlines on image by converting to BW and performing several other operations

Parameters

- **img** (*int*) – image in NumPy format
- **outlines** (*int*) – RGB array of the segment outlines
- **stencil** (*int*) – stencil array in RGB format

Returns image with outlines as overlay

Return type int

3.8 postprocessor module

postprocessor.**aerosol_correction**()

Perform the horizon area/sun circle correction.

The data from the main processing loop is used which is then subjected to a few steps. The approach by Long 2010 is used. Several statistical features of the segmetns are tested against a set of thresholds defined in `settings()`. Subsequently, the corrected sky cover percentages are written to a file.

3.9 processor module

`processor.`**`processor`**(*img*, *img_tsi*, *azimuth*, *altitude*, *filename*)

Call underlying processing routines and return the information to `main()`.

Parameters

- **`img`** (*`int`*) – Original image
- **`img_tsi`** (*`int`*) – Processed image from the tsi software
- **`azimuth`** (*`float`*) – azimuth of the sun, taken from the properties file
- **`altitude`** (*`float`*) – altitude of the sun, taken from the properties file
- **`filename`** (*`str`*) – Name of the file currently in use

Returns sky cover percentages, masked image, and pixel counts

Return type tuple

3.10 ratio module

`ratio.`**`red_blue`**(*maskedImg*)

Calculate the red/blue ratio per image pixel

Parameters **`maskedImg`** (*`int`*) – masked image

Returns red/blue ratio per image pixel

Return type float

3.11 read_properties_file module

`read_properties_file.`**`get_altitude`**(*lines*)

Get the solar altitude from the TSI properties file

Parameters **`lines`** – line by line reading of the properties file

Returns altitude of the sun

`read_properties_file.`**`get_azimuth`**(*lines*)

Get the solar azimuth from the TSI properties file

Parameters **`lines`** – line by line reading of the properties file

Returns azimuth of the sun

`read_properties_file.`**`get_fractional_sky_cover_tsi`**(*lines*)

Get the fractional sky cover from the TSI properties file

Parameters **`lines`** – line by line reading of the properties file

Returns fractional sky cover

3.12 resolution module

resolution.**get_resolution**(*img*)

Get the resolution of the image

Order (x,y) is swapped/reversed because the image format of TSI jpg files is reversed (for some reason). Resolution in both directions is then set as global so that it can be called like:

```
print(resolution.x)
print(resolution.y)
```

Parameters **img** (*int*) – Original unprocessed image

Returns:

3.13 settings module

Set the global variables for:

- Directories
- Aerosol corrections
- Colors
- Masking
- Image segments
- Sun features
- GLCM calculations
- Thresholding
- Plotting
- Toggling functions

3.14 skycover module

skycover.**fixed**(*red_blue_ratio*, *fixed_sunny_threshold*, *fixed_thin_threshold*)

Calculate the fractional sky cover based on fixed thresholding.

Parameters

- **red_blue_ratio** (*float*) – Pixel per pixel representation of the red/blue ratio
- **fixed_sunny_threshold** (*float*) – clear sky/cloudy fixed threshold
- **fixed_thin_threshold** (*float*) – thin/opaque fixed threshold

Returns thin sky cover, opaque sky cover and fractional sky cover

Return type tuple

skycover.**hybrid**(*ratioBR_norm_1d_nz*, *hybrid_threshold*)

Calculate the fractional sky cover based on hybrid thresholding.

Parameters

- **ratioBR_norm_1d_nz** (*float*) – normalized, masked, flattened red/blue ratio
- **hybrid_threshold** (*float*) – clear sky/cloud threshold determined by the hybrid algorithm

Returns fractional sky cover as determined by the hybrid thresholding algorithm

Return type float

3.15 statistical_analysis module

`statistical_analysis.`**`work`**`(`*maskedImg*`)`

Calculate the Grey Level Co-occurence Matrix (GLCM) and determine statistical features from it

This strategy is proposed by Heinle et al 2010

The statistical features can be used in machine learning algorithms such as k-nearest neighbor

Parameters **maskedImg** (*int*) – masked RGB image

Returns energy, entropy, contrast, homogeneity

Return type tuple

3.16 thresholds module

`thresholds.`**`fixed`**`()`

Get the fixed thresholds from the settings file

Returns the fixed thresholds

Return type tuple

`thresholds.`**`flatten_clean_array`**`(`*img*`)`

Convert 2D masked image to 1D flattened array to be used in MCE algorithm

Parameters **img** (*int*) – masked image

Returns normalized, 1D, flattened masked red/blue ratio array

Return type float

`thresholds.`**`hybrid`**`(`*img*`)`

Decide between fixed or MCE thresholding as part of hybrid thresholding algorithm

Parameters **img** (*int*) – masked image

Returns normalized 1D flattened masked red/blue ratio array, standard deviation of the image and hybrid threshold

Return type tuple

`thresholds.`**`min_cross_entropy`**`(`*data, nbins*`)`

Minimum cross entropy algorithm to determine the minimum of a histogram

Parameters

- **data** (*foat*) – the image data (e.g. blue/red ratio) to be used in the histogram
- **nbins** (*int*) – number of histogram bins

Returns the MCE threshold

Return type float

CHAPTER FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX