Flamingo Documentation

Release 1.0.1

Bas Hoonhout

December 15, 2014

The Flamingo toolbox is an open-source toolbox for image segmentation, classification and rectification. It is developed by the Department of Hydraulic Engineering of Delft University of Technology for coastal image analysis. The toolbox is built around the *scikit-image*, *scikit-learn*, *OpenCV* and *pystruct* toolboxes.

Flamingo is developed and maintained by:

Bas Hoonhout <b.m.hoonhout@tudelft.nl>
Max Radermacher <m.radermacher@tudelft.nl>

CHAPTER

ONE

CONTENTS

1.1 Rectification

This module provides functions to project an image onto a real-world coordinate system using ground control points. The module is largely based on the *OpenCV Camera Calibration and 3D Reconstruction* workflow and works nicely together with the *argus2* toolbox for coastal image analysis.

A typical workflow consists of determining ground control points by measuring the real-world coordinates of object visible in the image and the image coordinates of these very same objects. Also the camera matrix and lens distortion parameters should be determined.

Subsequently, a homography can be determined using the `flamingo.rectification.rectification.find_homography` function and a projection of the image can be plotted using the accompanying `flamingo.rectification.plot` module.

See also:

http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

1.1.1 Rectification

`flamingo.rectification.rectification.`**`find_homography`**(*UV*, *XYZ*, *K*, *distortion=array([[0., 0., 0., 0.]])*, *z=0*)

Find homography based on ground control points

Parameters

- **UV** (*np.ndarray*) – Nx2 array of image coordinates of gcp's
- **XYZ** (*np.ndarray*) – Nx3 array of real-world coordinates of gcp's
- **K** (*np.ndarray*) – 3x3 array containing camera matrix
- **distortion** (*np.ndarray, optional*) – 1xP array with distortion coefficients with P = 4, 5 or 8
- **z** (*float, optional*) – Real-world elevation on which the image should be projected

Returns 3x3 homography matrix

Return type np.ndarray

Notes

Function uses the OpenCV image rectification workflow as described in http://docs.opencv.org/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html starting with solvePnP.

Examples

```
>>> camera_id = 4
>>> r = argus2.rest.get_rectification_data('kijkduin')
>>> H = flamingo.rectification.find_homography(r[camera_id]['UV'],
                                               r[camera_id]['XYZ'],
                                               r[camera_id]['K'])
```

flamingo.rectification.rectification.**get_pixel_coordinates**(*img*)

Get pixel coordinates given an image

Parameters **img** (*np.ndarray*) – NxMx1 or NxMx3 image matrix

Returns

- *np.ndarray* – NxM matrix containing u-coordinates
- *np.ndarray* – NxM matrix containing v-coordinates

flamingo.rectification.rectification.**rectify_coordinates**(*U*, *V*, *H*)

Get projection of image pixels in real-world coordinates given image coordinate matrices and homography

Parameters

- **U** (*np.ndarray*) – NxM matrix containing u-coordinates
- **V** (*np.ndarray*) – NxM matrix containing v-coordinates
- **H** (*np.ndarray*) – 3x3 homography matrix

Returns

- *np.ndarray* – NxM matrix containing real-world x-coordinates
- *np.ndarray* – NxM matrix containing real-world y-coordinates

flamingo.rectification.rectification.**rectify_image**(*img*, *H*)

Get projection of image pixels in real-world coordinates given an image and homography

Parameters

- **img** (*np.ndarray*) – NxMx1 or NxMx3 image matrix
- **H** (*np.ndarray*) – 3x3 homography matrix

Returns

- *np.ndarray* – NxM matrix containing real-world x-coordinates
- *np.ndarray* – NxM matrix containing real-world y-coordinates

1.1.2 Visualization

flamingo.rectification.plot.**find_horizon_offset**(*x*, *y*, *max_distance=10000.0*)

Find minimum number of pixels to crop to guarantee all pixels are within specified distance

Parameters

- **x** (*np.ndarray*) – NxM matrix containing real-world x-coordinates
- **y** (*np.ndarray*) – NxM matrix containing real-world y-coordinates
- **max_distance** (*float, optional*) – Maximum distance from origin to be included in the plot. Larger numbers are considered to be beyond the horizon.

Returns Minimum crop distance in pixels (from the top of the image)

Return type float

flamingo.rectification.plot.**plot_coverage**(*X*, *Y*, *rotation=None*, *translation=None*, *max_distance=10000.0*, *ax=None*, *figsize=(30, 20)*, *cmap=<matplotlib.colors.LinearSegmentedColormap object at 0x10d7571d0>*, *alpha=0.4*)

Plot the coverage of the projection of multiple images in a single axis.

Plot the outline of lists of real-world x and y coordinate matrices. The resulting composition can be rotated and translated seperately.

Points projected at infinite distance can be ignored by specifying a maximum distance.

Parameters

- **X** (*list of np.ndarrays*) – List of NxM matrix containing real-world x-coordinates
- **Y** (*list of np.ndarrays*) – List of NxM matrix containing real-world y-coordinates
- **rotation** (*float, optional*) – Rotation angle in degrees
- **translation** (*list or tuple, optional*) – 2-tuple or list with x and y translation distances
- **max_distance** (*float, optional*) – Maximum distance from origin to be included in the plot. Larger numbers are considered to be beyond the horizon.
- **ax** (*matplotlib.axes.AxesSubplot, optional*) – Axis object used for plotting
- **figsize** (*tuple, optional*) – 2-tuple or list containing figure dimensions
- **cmap** (*matplotlib.colors.Colormap, optional*) – Colormap to determine colors for individual patches
- **alpha** (*float, optional*) – Alpha value for patches

Returns

- *matplotlib.figure.Figure* – Figure object containing axis object
- *matplotlib.axes.AxesSubplot* – Axis object containing plot

flamingo.rectification.plot.**plot_rectified**(*X*, *Y*, *imgs*, *rotation=None*, *translation=None*, *max_distance=10000.0*, *ax=None*, *figsize=(30, 20)*, *color=True*)

Plot the projection of multiple RGB images in a single axis.

Plot a list of images using corresponding lists of real-world x and y coordinate matrices. The resulting composition can be rotated and translated seperately.

Points projected at infinite distance can be ignored by specifying a maximum distance.

Parameters

- **X** (*list of np.ndarrays*) – List of NxM matrix containing real-world x-coordinates
- **Y** (*list of np.ndarrays*) – List of NxM matrix containing real-world y-coordinates
- **imgs** (*list of np.ndarrays*) – List of NxMx1 or NxMx3 image matrices
- **rotation** (*float, optional*) – Rotation angle in degrees
- **translation** (*list or tuple, optional*) – 2-tuple or list with x and y translation distances
- **max_distance** (*float, optional*) – Maximum distance from origin to be included in the plot. Larger numbers are considered to be beyond the horizon.
- **ax** (*matplotlib.axes.AxesSubplot, optional*) – Axis object used for plotting
- **figsize** (*tuple, optional*) – 2-tuple or list containing figure dimensions
- **color** (*bool, optional*) – Whether color image should be plotted or grayscale

Returns

- *matplotlib.figure.Figure* – Figure object containing axis object
- *matplotlib.axes.AxesSubplot* – Axis object containing plot

`flamingo.rectification.plot.`**`rotate_translate`**(*x*, *y*, *rotation=None*, *translation=None*)

Rotate and/or translate coordinate system

Parameters

- **x** (*np.ndarray*) – NxM matrix containing x-coordinates
- **y** (*np.ndarray*) – NxM matrix containing y-coordinates
- **rotation** (*float, optional*) – Rotation angle in degrees
- **translation** (*list or tuple, optional*) – 2-tuple or list with x and y translation distances

Returns

- *np.ndarrays* – NxM matrix containing rotated/translated x-coordinates
- *np.ndarrays* – NxM matrix containing rotated/translated y-coordinates

1.2 Segmentation

This module provides functions to segmentate an image into superpixels. It is largely based on the *scikit-image* toolbox. Apart from the regular segmentation functions it provides postprocessing functions to ensure connected segments in a regular grid. It also provides various visualization tools for segmented images.

See also:

http://scikit-image.org/docs/0.10.x/api/skimage.segmentation.html

1.2.1 Superpixels

`flamingo.segmentation.superpixels.`**`average_colors`**(*img*, *segments*)

Average colors per superpixels

Returns an image where each pixel has the average color of the superpixel that it belongs to.

Parameters

- **img** (*np.ndarray*) – NxM or NxMx3 array with greyscale or colored image information respectively
- **segments** (*np.ndarray*) – NxM matrix with segment numbering

Returns NxM or NxMx3 matrix with averaged image

Return type np.ndarray

flamingo.segmentation.superpixels.**check_segmentation**(*segments*, *nx*, *ny*)

Checks if segmentation data is complete

Checks if the segmentation data indeed contains nx*ny segments and if the set of segment numbers is continuous.

Parameters

- **segments** (*np.ndarray*) – NxM matrix with segment numbering
- **ny** (*nx,*) – Size of supposed segmentation grid

Returns Returns true if segmentation is valid and false otherwise

Return type bool

flamingo.segmentation.superpixels.**get_contours**(*segments*)

Return contours of superpixels

Parameters **segments** (*np.ndarray*) – NxM matrix with segment numbering

Returns list of lists for each segment in *segments*. Each segment list contains one or more contours. Each contour is defined by a list of 2-tuples with an x and y coordinate.

Return type list

Examples

```
>>> contours = get_contours(segments)
>>> plot(contours[0][0][0][0], contours[0][0][0][1]) # plot first contour of first segment
```

flamingo.segmentation.superpixels.**get_segmentation**(*img*, *method='slic'*, *method_params={}*, *extract_contours=False*, *remove_disjoint=True*)

Return segmentation of image

Parameters

- **img** (*np.ndarray*) – NxM or NxMx3 array with greyscale or colored image information respectively
- **method** (*str, optional*) – Segmentation method to use, supported by scikit-image toolbox
- **method_params** (*dict, optional*) – Extra parameters supplied to segmentation method
- **extract_contours** (*bool, optional*) – Also extract contours of segments
- **remove_disjoint** (*bool, optional*) – Ensure that the output contains connected segments only **and** that the superpixels form a more or less regular grid. In case the segmentation method does not provide both constraints, the constraint is ensured in a postprocessing step.

Returns NxM matrix with segment numbering

Return type np.ndarray

Examples

```
>>> img = argus2.rest.get_image(station='kijkduin')[0]
>>> segments = get_segmentation(img)
```

flamingo.segmentation.superpixels.**get_superpixel_grid**(*segments*, *img_shape*)
Return shape of superpixels grid

Parameters

- **segments** (*np.ndarray*) – NxM matrix with segment numbering
- **img_shape** (*2-tuple or list*) – Dimensions of image

Returns tuple containing M and N dimension of regular superpixel grid

Return type 2-tuple

flamingo.segmentation.superpixels.**shuffle_pixels**(*img*)
Shuffle class identifiers

Parameters **img** (*np.ndarray*) – NxM matrix with segment numbering

Returns NxM matrix with shuffled segment numbering

Return type np.ndarray

Examples

```
>>> seg = get_segmentation(img)
>>> fig, axs = plt.subplots(1, 2)
>>> axs[0].imshow(seg)
>>> axs[1].imshow(shuffle_pixels(seg))
```

1.2.2 Postprocessing

flamingo.segmentation.postprocess.**region_growing**(*mask*, *connectivity=8*)
Simple region growing algorithm

Parameters

- **mask** (*np.ndarray*) – Binary matrix indicating what pixels are within a region and what are not
- **connectivity** (*int, 4 or 8*) – Number of neighbouring pixels taken into account

Returns

- *list* – List of 2-tuples with coordinates within a region
- *list* – List of 2-tuples with coordinates at the edge of the region

flamingo.segmentation.postprocess.**regularize**(*segments*, *nx*, *ny*)
Create a regular grid from a collection of image segments

The number of segments supplied is supposed to be larger than the number of segments in the target grid (nx*ny). A regular grid of size nx*ny over the image grid NxM is constructed. Subsequently, the segments are ordered based on size. the nx*ny largest segments are preserved and assigned to a single grid cell in the regular grid based on least squares fit. The smaller segments are added to the preserved segment that is closest based on their centroids.

Parameters

- **segments** (*np.ndarray*) – NxM matrix with segment numbering
- **ny** (*nx,*) – Dimensions of the target superpixel grid

Returns NxM matrix with alternative segment numbering with segments in a regular grid

Return type np.ndarray

flamingo.segmentation.postprocess.**remove_disjoint**(*segments*)

Remove disjoint regions in segmentation

Remove disjoint regions in segmentation by running a region growing algorihtm for each segment. Any segment that appears to consist out of multiple disconnected parts is splitted. The biggest part remains as placeholder of the existing superpixel. The smaller parts are joined with the neighbouring superpixels. If multiple neighbouring superpixels exist, the one that shares the largest border is chosen.

Parameters **segments** (*np.ndarray*) – NxM matrix with segment numbering

Returns NxM matrix with alternative segment numbering with connected segments

Return type np.ndarray

1.2.3 Visualization

flamingo.segmentation.plot.**get_image_data**(*fig*, *dpi=96*, *axis_only=True*, *transparent=True*)

Get binary image data

Parameters

- **fig** (*matplotlib.figure.Figure*) – Figure object containing axis object
- **dpi** (*int, optional*) – Image resolution
- **axis_only** (*bool, optional*) – Only include contents of axis
- **transparent** (*bool, optional*) – Plot background transparent

Returns Binary image data

Return type str

flamingo.segmentation.plot.**plot**(*img*, *segments*, *mark_boundaries=True*, *shuffle=False*, *average=False*, *slice=1*, *raw=False*)

Plot segmentation result

Parameters

- **img** (*np.ndarray*) – NxM or NxMx3 array with greyscale or colored image information respectively
- **segments** (*np.ndarray*) – NxM matrix with segment numbering
- **mark_boundaries** (*bool, optional*) – Draw boundaries in image
- **shuffle** (*bool, optional*) – Shuffle segment numbering for more scattered coloring (ignored when *average* is used)
- **average** (*bool, optional*) – Average colors per segment
- **slice** (*int, optional*) – Use slice to reduce the image size
- **raw** (*bool, optional*) – Return raw binary output

Returns Binary image data or 2-tuple with matplotlib.figure.Figure and matplotlib.axes.AxesSubplot objects

Return type str or 2-tuple

`flamingo.segmentation.plot.`**`plot_image`**(*img*, *cmap='Set2'*, *dpi=96*, *slice=0*, *transparent=True*, *raw=False*)

Get binary image data

Parameters

- **img** (*np.ndarray*) – NxM or NxMx3 array with greyscale or colored image information respectively
- **cmap** (*matplotlib.colors.Colormap, optional*) – Colormap to determine colors for individual patches
- **dpi** (*int, optional*) – Image resolution
- **slice** (*int, optional*) – Use slice to reduce the image size
- **transparent** (*bool, optional*) – Plot background transparent
- **raw** (*bool, optional*) – Return raw binary output

Returns Binary image data or 2-tuple with matplotlib.figure.Figure and matplotlib.axes.AxesSubplot objects

Return type str or 2-tuple

1.3 Classification

This module provides functions to train image classification models, like Logistic Regressors and Conditional Random Fields. It provides functions for feature extraction that are largely based on the *scikit-image* toolbox and it provides functions for model training and optimization that are largely based on the *pystruct* and *scikit-learn* toolbox.

See also:

http://scikit-image.org/docs/0.10.x/api/skimage.feature.html

See also:

http://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model

See also:

https://pystruct.github.io/references.html

1.3.1 Models

`flamingo.classification.models.`**`get_model`**(*model_type='LR'*, *n_states=None*, *n_features=None*, *rlp_maps=None*, *rlp_stats=None*, *C=1.0*)

Returns a bare model object

Parameters **model_type** (*string, optional*) – String indicating the type of model to be constructed. LR = Logistic Regressor (default), LR_RLP = Logistic Regressor with Relative Location Prior, SVM = Support Vector Machine, CRF = Conditional Random Field

Returns Bare model object

Return type object

Other Parameters

- **n_states** (*integer*) – Number of classes (CRF only)
- **n_features** (*integer*) – Number of features (CRF only)

flamingo.classification.models.**score_model**(*model*, *X_train*, *Y_train*, *X_test*, *Y_test*, *maps=None*, *stats=None*, *features=None*)

Scores a single model using a train and test set

Parameters

- **model** (*object*) – Trained model object. Model object should have a score() method.
- **X_train** (*list or numpy.ndarray*) – 2D array containing training data. Each row is a training instance, while each column is a feature.
- **Y_train** (*list or numpy.ndarray*) – Array containing class annotations for each training instance.
- **X_test** (*Similar to X_train, but with test data.*) –

Returns

- **score_train** (*float*) – Training score
- **score_test** (*float*) – Test score

flamingo.classification.models.**score_models**(*models*, *train_sets*, *test_sets*, ***kwargs*)

Compute train/test scores for a set of trained models

Parameters

- **models** (*list*) – List of lists with each item a trained instance of a model.
- **train_sets** (*list*) – List of tuples containing training data corresponding to the model list.
- **test_sets** (*list*) – List of tuples containing test data corresponding to the model list.

Returns MultiIndex DataFrame containing training and test scores. Indices "model" and "set" indicate the model and training set number used. Columns "train" and "test" contain the train and test scores respectively.

Return type pandas.DataFrame

Notes

Models should be trained. Model and set lists should be of equal length. In case of N models and M training sets the models should be organized in a

N-length list of M-length lists.

The train and test sets should both be M-length lists.

Examples

```
>>> models = [models.get_model(model_type='LR'),
              models.get_model(model_type='CRF', n_states=5, n_features=10)]

>>> models_trained = models.train_models(models, [(X_train, Y_train)])
```

```
>>> scores = test.score_models(models, [(X_train, Y_train)], [(X_test, Y_test)])
```

flamingo.classification.models.**train_model**(*model*, *X_train*, *Y_train*, *X_train_prior=None*)

Trains a single model against a single training set

Parameters

- **model** (*object*) – Bare model object. Model object should hava a fit() method.
- **X_train** (*list or numpy.ndarray*) – 2D array containing training data. Each row is a training instance, while each column is a feature.
- **Y_train** (*list or numpy.ndarray*) – Array containing class annotations for each training instance.

Notes

Models are passed by reference and trained without copying.

flamingo.classification.models.**train_models**(*models*, *train_sets*, *prior_sets=None*, *callback=None*)

Trains a set of model against a series of training sets

Parameters

- **models** (*list*) – List of model objects. Model objects should have a fit() method.
- **train_sets** (*list*) – List of tuples containing training data. The first item in a tuple is a 2D array. Each row is a training instance,

  while each column is a feature.

  The second item in a tuple is an array containing class annotations for each training instance.

- **prior_sets** (*list*) – List of 2D arrays containing prior data. Similar to first tuple item in train_sets Each item is a 2D array. Each row is a training instance,

  while each column is a feature.

- **callback** (*function*) – Callback function that is called after training of a model finished. Function accepts two parameters: the model object and a tuple with location

  indices in the resulting model matrix.

Returns List of lists with each item a trained instance of one of the models.

Return type list

1.3.2 Features

flamingo.classification.features.features.**linearize**(*features*)

convert all items in each matrix feature into individual features

flamingo.classification.features.blocks.**extract_blocks**(*data*, *segments*, *colorspace='rgb'*, *blocks=None*, *blocks_params={}*)

Extract all blocks in right order

flamingo.classification.features.blocks.**list_blocks**()
List all block extraction functions in module

flamingo.classification.features.relativelocation.**compute_prior**(*annotations*, *centroids*, *image_size*, *superpixel_grid*, *n=100*)
Compute relative location prior according to Gould et al. (2008)

Parameters **ds** (*string*) – String indicating the dataset to be used.

Returns **maps** – 4D panel containing the relative location prior maps: maps[<other class>][<given class>] gives a n*n dataframe representing the dimensionless image map

Return type pandas.Panel4D

Other Parameters **n** (*integer*) – Half the size of the dimensionless image map

flamingo.classification.features.relativelocation.**smooth_maps**(*maps*, *sigma=2*)
Convolve relative location prior maps with a gaussian filter for smoothing purposes

Parameters

- **ds** (*string*) – String indicating the dataset to be used.
- **maps** (*pandas.Panel4D*) – 4D panel containing the relative location prior maps: maps[<other class>][<given class>] gives a n*n dataframe representing the dimensionless image map

Returns **maps** – 4D panel containing the smoothed relative location prior maps.

Return type pandas.Panel4D

Other Parameters **sigma** (*integer*) – Size of the gaussian kernel that is to be convolved with the relative location prior maps

flamingo.classification.features.relativelocation.**vote_image**(*Y*, *maps*, *centroids=None*, *img_size=None*, *winner_takes_all_mode=False*)
Class voting based on 1st order prediction and relative location prior maps

Parameters

- **ds** (*string*) – String indicating the dataset to be used.
- **Ipred** (*list of lists of tuple of lists of arrays with size [n_models][n_partitions](training,testing)[n_images]*) – Arrays contain the 1st order prediction of the labelled images.

Returns

- **votes** (*pandas.Panel*) – Panel containing the votes for all classes and superpixels: maps[<class>] gives a nx*ny dataframe representing the probability of every superpixel to be <class>
- **Ivote** (*np.array*) – Labelled image based on classes in votes with maximum probability for every superpixel

1.3.3 Channels

flamingo.classification.channels.**add_channels**(*img, colorspace='rgb', methods=['gabor', 'gaussian', 'sobel'], methods_params=None*)

add channels to an image. Channels are: - 0: greyscale - 1,2,3: colorspace - extra channels

1.3.4 Test

flamingo.classification.test.**aggregate_scores**(*scores*)

Aggregate model scores over training and test sets

Parameters **scores** (*pandas.DataFrame*) – DataFrame with test scores for different models and training sets. Should have at least one level index names "model".

Returns DataFrame averaged over all indices except "model".

Return type pandas.DataFrame

flamingo.classification.test.**compute_learning_curve**(*models, train_sets, test_sets, step=10, **kwargs*)

Computes learning curves for combinations of models and training/test sets

Parameters

- **models** (*list*) – List of model objects. Model objects should have a fit() and score() method.
- **train_sets** (*list*) – List of tuples containing training data corresponding to the model list.
- **test_sets** (*list*) – List of tuples containing test data corresponding to the model list.
- **step** (*integer, optional*) – Step size of learning curve (default: 10)
- ****kwargs** – All other named arguments are redirected to the function models.train_models()

Returns

- **all_scores** (*pandas.DataFrame*) – MultiIndex DataFrame containing training and test scores. Indices "model" and "set" indicate the model and training set number used. Index "n" indicates the number of samples used during training. Columns "train" and "test" contain the train and test scores respectively.
- **all_models** (*list*) – List with trained models. Each item corresponds to a single point on the learning curve and

  can consist of several models organized in a NxM matrix where N is the original number of models trained and M is the number of training sets used.

flamingo.classification.test.**plot_learning_curve**(*scores, ylim=(0.75, 1), filename=None*)

Plots learning curves

Parameters

- **scores** (*pandas.DataFrame*) – DataFrame containing all scores used to plot one or more learning curves. Should at least have the index "n" indicating the number of training samples used.
- **ylim** (*2-tuple, optional*) – Vertical axis limit for learning curve plots.
- **filename** (*string, optional*) – If given, plots are saved to indicated file path.

Returns

- *list* – List with figure handles for all plots
- *list* – List with axes handles for all plots

1.3.5 Visualization

flamingo.classification.plot.**save_figure**(*fig*, *filename*, *ext=''*, *figsize=None*, *dpi=30*, ***kwargs*)

Save figure to file

Parameters

- **fig** (*object*) – Figure object
- **filename** (*string*) – Path to output file
- **ext** (*string, optional*) – String to be added to the filename before the file extension
- **a Matplotlib figure as an image without borders or frames.** (*Save*) –

  Args: fileName (str): String that ends in .png etc.

  fig (Matplotlib figure instance): figure you want to save as the image

  Keyword Args: orig_size (tuple): width, height of the original image used to maintain aspect ratio.

1.3.6 Utils

flamingo.classification.utils.**check_sets**(*train_sets*, *test_sets*, *models=None*)

Checks if train sets, test sets and models have matching dimensions

Parameters

- **train_sets** (*list*) – List of tuples containing training data corresponding to the model list.
- **test_sets** (*list*) – List of tuples containing test data corresponding to the model list.
- **models** (*list*) – List of lists with each item a trained instance of a model.

1.4 Calibration

In development.

CHAPTER

TWO

COMMAND-LINE TOOLS

Several command-line functions are supplied with the toolbox for batch processing of large datasets. Each command-line function serves a specific part of the image analysis. See for more information the *–help* option of each command.

2.1 rectify-images

Ortho-rectify images based on ground control points (GCP)

Usage: rectify-image <image> <gcpfile> [–dist-model=NAME] [–dist-coefs=VALUES] [–verbose]

Positional arguments: image image to be rectified gcpfile file containing GCP's in image (UVXYZ)

Options:

-h, --help	show this help message and exit
--dist-model=NAME	name of distortion model to use [default: OPENCV]
--dist-coefs=VALUES	coefficients used for distortion model [default: 0,0,0,0]
--size=SIZE	size of output figure [default: 30,20]
--rotation=ANGLE	rotate resulting image [default: 0]
--translation=DIST	translate resulting image [default: 0,0]
--maxdistance=DIST	maximum distance from origin included in plot [default: 1e4]
--verbose	print logging messages

2.2 classify-images

Train, score and use classification models on image datasets.

Usage: classify-images preprocess <dataset> [–segmentate] [–channels] [–features] [–extract] [–update] [–normalize] [–relloc] [–relloc_maps] [–images=FILE] [–config=FILE] [–overwrite] [–verbose]

classify-images partition <dataset> [–n=N] [–frac=FRAC] [–images=FILE] [–config=FILE] [–verbose]

classify-images train <dataset> [–type=NAME] [–partitions=N] [–images=FILE] [–config=FILE] [–verbose]

classify-images score <dataset> [–model=NAME] [–images=FILE] [–config=FILE] [–verbose]

classify-images predict <dataset> [–model=NAME] [–images=FILE] [–config=FILE] [–overwrite] [–verbose]

classify-images regularization <dataset> [–type=NAME] [–images=FILE] [–config=FILE] [–verbose]

Positional arguments: dataset dataset containing the images image image file to be classified

Options:

-h, --help	show this help message and exit
--segmentate	create segmentation of images
--channels	include channel extraction
--features	include feature extraction
--extract	extract channels/features
--update	update channels/features
--normalize	normalize channels/features
--relloc	include relative location features
--relloc_maps	compute new relative location maps
--n=N	number of partitions [default: 5]
--frac=FRAC	fraction of images used for testing [default: 0.25]
--type=NAME	model type to train [default: LR]
--partitions=N	only train these partitions
--model=NAME	name of model to be scored, uses last trained if omitted
--images=FILE	images to include in process
--config=FILE	configuration file to use instead of command line options
--overwrite	overwrite existing files
--verbose	print logging messages

2.3 calibrate-camera

CHAPTER

THREE

FILE SYSTEM

The toolbox uses a file system structure for the analysis of datasets. The `flamingo.filesys` module takes care of any reading and writing of files in this file structure. Each dataset is stored in a single directory and can consist out of the following file types:

Image files Any image file recognized by the system

Cropped image files Names start with *cropped_*. A non-cropped version of the image file should exist.

Export files Pickle files with data concerning an image. Each export file name has the following format: *<image_name>.<key>.pkl*. A special type of export file is the feature file. Not all features are written to a single export file, but they are subdivided into multiple export files depending on the feature block they belong to. The block name is added to the export file, just before the file extension.

Log files Pickle files with data concerning the entire dataset. Log file names can have any name.

Model files Pickle files with a trained model. Each model file is accompanied by a meta file. Each model file name has the following format: *model_<model_type>_<dataset>_I<nr_of_images>_B<nr_of_blocks>_<timestamp>.pkl*. The corresponding meta file has *meta* added to the name, just before the file extension.

`flamingo.filesys.`**`check_export_file`**(*ds*, *im*, *ext*)
Check if export file exists

`flamingo.filesys.`**`get_dataset_list`**()
Get list of available datasets

`flamingo.filesys.`**`get_dataset_path`**()
Get path to datasets

`flamingo.filesys.`**`get_export_file`**(*ds*, *im=None*, *ext=None*)
Get path to export file

`flamingo.filesys.`**`get_image_list`**(*ds*)
Get list with all images in dataset

`flamingo.filesys.`**`get_image_location`**(*ds*, *im*)
Get absolute path to specific image in dataset

`flamingo.filesys.`**`get_image_path`**(*ds*)
Get absolute path to images within dataset

`flamingo.filesys.`**`get_model_list`**(*ds*)
Get list of model files in dataset

`flamingo.filesys.`**`has_features`**(*ds*, *im*)
Determine if features for image are extracted

flamingo.filesys.**is_classified**(*ds*, *im*)
Determine if image is annotated

flamingo.filesys.**is_segmented**(*ds*, *im*)
Determine if image is segmented

flamingo.filesys.**read_default_categories**(*ds*)
Read list of uniquely defined classes in dataset

flamingo.filesys.**read_export_file**(*ds*, *im*, *ext*)
Read contents of export file

flamingo.filesys.**read_feature_files**(*ds, im, blocks=['extract_blocks_grey', 'extract_blocks_intensity', 'extract_blocks_mask', 'extract_blocks_pixel', 'extract_blocks_shape', 'extract_blocks_intensitystatistics'], ext=None*)
Read features from a collection of export files including only selected feature blocks

flamingo.filesys.**read_image_file**(*ds*, *im*, *crop=True*)
Read image file

flamingo.filesys.**read_log_file**(*ds*, *keys=None*)
Read contents of log file

flamingo.filesys.**read_model_file**(*ds*, *model_name*)
Read a model from export file including meta data

flamingo.filesys.**set_dataset_path**(*fpath*)
Set path to datasets

flamingo.filesys.**write_export_file**(*ds*, *im*, *ext*, *contents*, *append=False*)
Write contents to export file

flamingo.filesys.**write_feature_files**(*ds*, *im*, *features*, *features_in_block*, *ext=None*)
Write features to a collection of export files depending on their feature block

flamingo.filesys.**write_log_file**(*ds*, *contents*)
Write contents to log file

flamingo.filesys.**write_model_file**(*ds*, *model*, *meta*, *ext=''*)
Write a single model to export file including meta data

flamingo.filesys.**write_model_files**(*ds*, *models*, *meta*, *ext=''*)
Write a series of models to export files including meta data

CHAPTER

FOUR

CONFIGURATION

Only the very basic options of the toolbox are exposed through the command-line functions. For the full extent of options a configuration file is used. This configuration file is parsed by the `flamingo.config` module. The module also supplies wrappers for the automated updating of a function call based on the configuration file used.

flamingo.config.**CLASSIFICATION_DEFAULTS = {'channels': {'enabled': True, 'methods': ['gabor', 'gaussian', 'sobel']**

Configuration constants for classification toolbox

flamingo.config.**get_function_args**(*fcn*, *cfg*, *sections=[]*)

Get relevant function arguments given a configuration file

flamingo.config.**parse_config**(*sections=[]*)

Wrapper for parsing config file for specific function call

flamingo.config.**read_config**(*cfgfile, defaults={'channels': {'enabled': True, 'methods': ['gabor', 'gaussian', 'sobel'], 'methods_params': {'frequencies': [0.05, 0.15, 0.25], 'sigmas': [1, 8, 15], 'thetas': [0.0, 0.785, 1.571, 2.356]}}, 'segmentation': {'remove_disjoint': True, 'extract_contours': False, 'enabled': True, 'method': 'slic', 'method_params': {}}, 'relative_location': {'sigma': 2, 'enabled': False, 'n': 100}, 'features': {'feature_blocks': 'all', 'enabled': True, 'blocks_params': {}}, 'score': {}, 'train': {'partitions': 'all'}, 'partition': {'n_partitions': 5, 'force_split': False, 'enabled': True, 'frac_test': 0.25, 'frac_validation': 0.0}, 'regularization': {'C': [0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0], 'partition': 0}, 'general': {'model_type': 'LR', 'colorspace': 'rgb', 'class_aggregation': '', 'model_dataset': ''}}*)

Read configuration file and update default settings

flamingo.config.**write_config**(*cfgfile, defaults={'channels': {'enabled': True, 'methods': ['gabor', 'gaussian', 'sobel'], 'methods_params': {'frequencies': [0.05, 0.15, 0.25], 'sigmas': [1, 8, 15], 'thetas': [0.0, 0.785, 1.571, 2.356]}}, 'segmentation': {'remove_disjoint': True, 'extract_contours': False, 'enabled': True, 'method': 'slic', 'method_params': {}}, 'relative_location': {'sigma': 2, 'enabled': False, 'n': 100}, 'features': {'feature_blocks': 'all', 'enabled': True, 'blocks_params': {}}, 'score': {}, 'train': {'partitions': 'all'}, 'partition': {'n_partitions': 5, 'force_split': False, 'enabled': True, 'frac_test': 0.25, 'frac_validation': 0.0}, 'regularization': {'C': [0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0], 'partition': 0}, 'general': {'model_type': 'LR', 'colorspace': 'rgb', 'class_aggregation': '', 'model_dataset': ''}}*)

Write configuration file

4.1 Example configuration

```
[channels]
enabled = True
methods = ["gabor", "gaussian", "sobel"]
methods_params = {"frequencies": [0.05, 0.15, 0.25], "sigmas": [1, 8, 15], "thetas": [0.0, 0.785, 1.5

[segmentation]
remove_disjoint = True
extract_contours = False
enabled = True
method = slic
method_params = {}

[relative_location]
sigma = 2
enabled = False
n = 100

[features]
feature_blocks = all
enabled = True
blocks_params = {}

[score]

[train]
partitions = all

[partition]
n_partitions = 5
force_split = False
enabled = True
frac_test = 0.25
frac_validation = 0.0

[regularization]
c = [0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
partition = 0

[general]
model_type = LR
colorspace = rgb
class_aggregation =
model_dataset =
```

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

f

P

R

S

T

V

W