MAPS XML Parser

Release 0.1.0

Joel Luethi

Jun 20, 2019

CONTENTS:

1 Indices and tables 11

Python Module Index 13

Index 15

class sites_of_interest_parser.**MapsXmlParser**(*project_folder: str*, *name_of_highmag_layer: str = 'highmag'*, *use_unregistered_pos: bool = True*, *stitch_radius: int = 1*)

XML Parser class that processes MAPS XMLs and reveals the image tiles belonging to each annotation

This class takes XML files from the Thermo MAPS Application, extracts the location of all its annotations and compares it to the location of all the image tiles. This gets a bit more complicated, because tiles only have a relative position in pixels within their layers and those layers are (potentially) turned at arbitrary angles relative to the global coordinate system (which is in meters). It is important that the functions are run in the correct order to extract & process this information. Therefore, use it by initializing the class and the calling parser.parse_xml().

Parameters

- **project_folder** (*str*) – The path to the project folder of the MAPS project, containing the XML file, as a string
- **name_of_highmag_layer** (*str*) – Name of the image layer in MAPS for which tiles containing annotations should be found. Defaults to 'highmag'
- **use_unregistered_pos** (*bool*) – Whether to use the unregistered position of the tiles (where MAPS thinks it acquired them => True, default) or a calculated position (e.g. through stitching in MAPS) should be used for the tiles (False)
- **stitch_radius** (*int*) – The number of images in each direction from the tile containing the annotation should be stitched. Parser doesn't do the stitching, just extracts relevant information about neighboring tiles. Defaults to 1 => 3x3 images would be stitched.

project_folder_path

The path to the project folder of the MAPS project, containing the XML file, as a string

Type str

layers

Contains information about the layers (squares/acquisitions). The keys are local paths to the metadata about the layer and the values are dictionaries again. Each layer contains the information about the stage position of the center of the layer (in meters, StagePosition_center_x & StagePosition_center_y), about the rotation of the layer relative to the global stage positions (in degrees, rotation), the number of columns of tiles (images) in the layer (columns), the vertical & horizontal overlap between the tiles (overlapVertical, overlapHorizontal), the number of rows of tiles (images) in the layer (rows), the horizontal field width of each tile (its width in meters, tileHfw), the horizontal field width of the whole layer (its width in meters, totalHfw), the name of the layer (layer_name), the vertical field width of each tile (its width in meters, tileVfw), and the global stage position of the corner of the layer (in meters, StagePosition_corner_x & StagePosition_corner_y)

Type dict

tiles

Contains information about individual tiles. The keys are the combined layer name & filename of the tile and the values are a dictionary again. Each tile contains the information about what layer it belongs to (layer, key to the layer dict), the path to the image as a Path variable (img_path), its filename, the name of the layer (layer_name) and its relative position x & y within that layer (RelativeTilePosition_x & RelativeTilePosition_y)

Type dict

annotations

Contains the information about all the annotations. The keys are the names of the annotations (MAPS enforces uniqueness), its values are a dictionary containing the StagePosition_x & StagePosition_y positions of the annotation (in m => global coordinate system for the experiment)

Type dict

annotation_tiles
Contains the relevant output of the XML parsing. The keys are the names of the annotation. The values are the key to the corresponding layer in the layer dict (layer), the path to the image as a Path variable (img_path), its filename, the name of the layer (layer_name) and the relative position x & y of the tile within that layer (RelativeTilePosition_x & RelativeTilePosition_y), the absolute stage position of the annotation (Annotation_StagePosition_x and Annotation_StagePosition_y), the position of the annotation within the tile image (in pixels, Annotation_tile_img_position_x & Annotation_tile_img_position_y), a list of the surrounding tile names (surrounding_tile_names) and a list of booleans of whether each of the surrounding tiles exist, including the tile itself (surrounding_tile_exists)

Type dict

stitch_radius
The number of images in each direction from the tile containing the annotation should be stitched.

Type int

pixel_size
The pixel size of the acquisition in the name_of_highmag_layer [in meters]

Type float

img_height
The height of the highmag image in pixels

Type int

img_width
The width of the highmag image in pixels

Type int

calculate_absolute_tile_coordinates()
Calculate the absolute stage positions of all tiles based on their relative positions

Calculate the absolute stage position of the center of each tile based on the relative tile positions, the rotation of the layer and the absolute stage position of the center of the layer. The resulting position is saved to the _tile_center_stage_positions and the corresponding tile name to the _tile_names list.

convert_img_path_to_local_path(*img_path*)
Converts a local path of the microscope computer to a path of the image in the project folder

MAPS saves the image paths of the local storage of the image files. In our setup, this path starts with 'D:ProjectName', even though the files aren't actually on the D drive anymore but were copied to a share, into the project_folder_path. This function strips 'D:ProjectName' away from the path and returns a Path object for the location of the images on the share.

Parameters **img_path** (*str*) – Original path to the images on the microscope computer, starting with 'D:ProjectName'

Returns Path object of the corrected path on the share

Return type path

static convert_windows_pathstring_to_path_object(*string_path*)
Converts a windows path string to a path object

Some paths are provided in the XML file and the metadata as Windows paths. This function creates pathlib Path objects out of them.

Parameters **string_path** (*str*) – String of a Windows path containing double backslashes

Returns Path object of the string_path

Return type path

static create_logger(*log_file_path*, *multiprocessing_logger: bool = False*)
Returns a logger and creates it if it doesn't yet exist

Gets the correct logger for normal or multiprocessing. If it already exists, just returns this logger. If it doesn't exist yet, sets it up correctly.

Parameters

- **log_file_path** (*str*) – Path to the log file
- **multiprocessing_logger** (*bool*) – Whether a multiprocessing logger is needed. Defaults to False.

Returns Logger object that has the correct handlers

Return type logger

determine_surrounding_tiles()
Checks whether each annotation tile has surrounding tiles to be stitched with it

For each annotation tile, it checks whether the surrounding tiles in a given stitch_radius exist. It saves a boolean list of their existence (including its own existence) to surrounding_tile_exists and the a list of names of the surrounding filenames to surrounding_tile_names of the annotation_tiles dictionary

extract_layers_and_annotations(*check_for_annotations: bool = True*)
Extract the information about all the layers in the high magnification acquisition layers and the annotations

Go through the XML file and submit all LayerGroups for processing. The LayerGroups contain both the Layer with the _name_of_highmag_layer (=> the actual images) as well as the Annotation Layers containing the annotations. At the end of processing, check whether the self.layers and the self.annotations dictionaries were filled. If one of them was not filled, the parser could not find the highmag images or the annotations and raises an exception.

Parameters **check_for_annotations** (*bool*) – Whether the parsing should check for the existence of annotations and throw an error if none are found. Defaults to True, thus checking for annotations.

find_annotation_tile()
Find the image tile in which each annotation is

Based on the absolute stage position of the annotations and the calculated stage positions of the center of the tiles, this function calculates the tiles within which all annotation are and saves this information to the annotation_tiles dictionary. The keys are the names of the annotation. The values are the key to the corresponding layer in the layer dict (layer), the path to the image as a Path variable (img_path), its filename, the name of the layer (layer_name) and the relative position x & y of the tile within that layer (RelativeTilePosition_x & RelativeTilePosition_y), the absolute stage position of the annotation (Annotation_StagePosition_x and Annotation_StagePosition_y), the position of the annotation within the tile image (in pixels, Annotation_tile_img_position_x & Annotation_tile_img_position_y). The surrounding_tile_names and surrounding_tile_exists are added in determine_surrounding_tiles

get_relative_tile_locations()
Read in all the metadata files for the different layers to get the relative tile positions

Each layer has its own metadata XML file that contains the relative positions of all the tiles in the layer. This function goes through all of them, extracts the information and saves it to the tiles dictionary. The keys are the combined layer name & filename of the tile and the values are a dictionary again. Each tile contains the information about what layer it belongs to (layer, key to the layer dict), the path to the image

as a Path variable (img_path), its filename, the name of the layer (layer_name) and its relative position x & y within that layer (RelativeTilePosition_x & RelativeTilePosition_y)

static load_annotations_from_csv(*base_header*, *csv_path*)
Load the annotations saved by sites_of_interest_parser.save_annotation_tiles_to_csv

Recreates a dictionary for all the contents except the columns of the base_header Parses the string of a list for surrounding_tile_exists & surrounding_tile_names back to a list

Parameters

- **base_header** (*list*) – list of column headers that will be ignored when loading the data
- **csv_path** (*Path*) – pathlib Path to the csv file that will be created. Must end in .csv

Returns annotation_tiles

Return type dict

static load_xml(*xml_file_path*)
Loads and returns the MAPS XML File

Parameters **xml_file_path** (*Path*) – Path to the XML file as a pathlib path

Returns The root of the XML file parsed with xml.etree.ElementTree

Return type root

parse_classifier_output(*file_path*, *annotation_shift: int = 128*)
Parses the output of the classifier and returns the annotation_tiles

Goes through the csv file produced by the classifier, parses the corresponding MAPS XML File and builds the necessary MapsXmlParser dictionaries so that they can be saved to csv afterwards. Excpects the following structure in the classifier csv: Dictionary with TileSets as keys and a dictionary as value. The value dictionary contains dictionaries for each tile with annotations. The name of the tile is the key and the top left corner of the annotations are the values, saved as a list of tuples (x,y)

Parameters

- **file_path** (*str*) – Path to the classifier output csv
- **annotation_shift** (*int*) – How much the center of the annotation is shifted from the top left corner. Defaults to 128 (for 256x256 images from the classifier)

Returns Dictionary of all annotation tiles

Return type self.annotation_tiles

parse_xml()
Run function for the class

parse_xml calls all the necessary functions of the class in the correct order to parse the XML file. Call this function after initializing a class object, then access the results via the annotation_tiles variable that contains the relevant output of the XML parsing

Returns annotation_tiles

Return type dict

static save_annotation_tiles_to_csv(*annotation_tiles*, *base_header*, *csv_path*, *batch_size=0*)
Saves the information about all annotations to a csv file

Goes through the annotation_tiles dictionary and saves it to a csv file. Overwrites any existing file in the same location. Can write everything into one csv file (default) or into multiple batches of a given size

Parameters

- **annotation_tiles** (*dict*) – annotation tiles dictionary with a structure like self.annotation_tiles
- **base_header** (*list*) – list of strings that will be headers but will not contain any content
- **csv_path** (*Path*) – pathlib Path to the csv file that will be created. Must end in .csv
- **batch_size** (*int*) – The size of each batch. Defaults to 0. If it's 0, everything is saved into one csv file. Otherwise, the dataframe is divided into batches of batch_size and saved to separate csv files

Returns list of all the paths to the saved csv files

Return type list

exception sites_of_interest_parser.**XmlParsingFailed**(*message*)

class stitch_MAPS_annotations.**Stitcher**(*base_path: str*, *project_name: str*, *csv_folder: str = 'annotation_csv'*, *output_folder: str = 'stitched-Forks'*, *stitch_radius: int = 1*)

Stitches Talos images based on csv files containing the necessary information

This class handles the stitching of Talos images based on a csv file like the one created by the MapsXmlParser. It creates the necessary folders, calls MapsXmlParser for the parsing and saving of the necessary metadata and then manages the stitching of all annotations. Current implementation of stitching is hard-coded to stitch_radius = 1, the size of the Talos images and an overlap of 10% in its stitching parameters in the stitch_annotated_tiles function

Parameters

- **base_path** (*str*) – Path (as a string) to the directory containing the project_name folder.
- **project_name** (*str*) – Name of the directory containing the MAPSProject.xml file and the LayersData folder of the MAPS project. Will be used as the location for the output folders. Must be in base_path folder
- **csv_folder** (*str*) – Name of the folders where the csv files are saved to
- **output_folder** (*str*) – Name of the folder where the stitched forks are saved to
- **stitch_radius** (*int*) – The number of images in each direction from the tile containing the annotation should be stitched.

stitch_radius
The number of images in each direction from the tile containing the annotation should be stitched.

Type int

project_name
Name of the current project being processed

Type str

project_folder_path
Full path to the project folder, containing the MAPSProject.xml file and the LayersData folder

Type Path

output_path
Full path to the output folder where the stitched images are stored

Type Path

csv_base_path
Full path to the folder where the csv files with all annotation metadata are stored

Type Path

base_header
List of all the column headers that should be added to the csv file (for classification of annotations and for measurements made on them)

Type list

combine_csvs(*delete_batches: bool = False*, *to_excel: bool = True*)
Combines batch csv output files into the final csv file and optionally an excel file

Parameters

- **delete_batches** (*bool*) – Whether the batch csv files should be deleted after stitching them to the combined csv file. Defaults to False (not deleting the batch csv files)
- **to_excel** (*bool*) – Whether the csv file should also be saved as an excel file. Defaults to True (creating the Excel file)

static local_contrast_enhancement(*img_path*, *output_path*, *logger=None*, *save_img: bool = False*, *eight_bit: bool = True*, *use_norm_local_contrast: bool = False*, *use_CLAHE: bool = False*, *return_java_img: bool = False*, *return_numpy_img: bool = False*, ***kwargs*)
Loads an image and performs local contrast enhancement

Loads the specified image, performs either NormalizeLocalContrast (default), CLAHE or no local contrast enhancement (in that order, thus performs NormalizeLocalContrast if both it and CLAHE are True). Optionally converts the image to 8bit and saves it. The parameters for NormalizeLocalContrast and CLAHE have default values that can be overwritten using the **kwargs by providing an argument with the specific name of the parameter to be changed

Parameters

- **img_path** (*Path or str*) – Full path to the image to be processed
- **output_path** (*Path or str*) – Full path to where the output image should be saved (if save_img is True)
- **logger** (*Logging*) – Logging object that is configured for the logging either in multiprocessing or normal processing. Defaults to None, thus loading the logger
- **save_img** (*bool*) – Whether the processed image should be saved. Defaults to False (not saving the image)
- **eight_bit** (*bool*) – Whether the image should be converted to 8 bit . Defaults to True (converting to 8 bit)
- **use_norm_local_contrast** (*bool*) – Whether NormalizeLocalContrast Fiji Plugin should be run on the image. Defaults to False (not running NormalizeLocalContrast on the image)
- **use_CLAHE** (*bool*) – Whether CLAHE Fiji Plugin should be run on the image. Defaults to False (not running CLAHE on the image)
- **return_java_img** (*bool*) – Whether the function should return the java image. Defaults to False (not returning the image)
- **return_numpy_img** (*bool*) – Whether the function should return a numpy version of the image. Defaults to False (not returning the image)

Returns The processed image as an ImageJ1 ImagePlus image (if return_image is True)

Return type ImagePlus

manage_batches(*stitch_threshold: int = 1000*, *eight_bit: bool = True*, *show_arrow: bool = True*, *max_processes: int = 4*, *enhance_contrast: bool = True*)

Manages the parallelization of the stitching of batches

As multiprocessing can make some issues, if max_processes is set to 1, it does not use multiprocessing calls.

Parameters

- **stitch_threshold** (*int*) – Threshold to judge stitching quality by. If images would be moved more than this threshold, the stitching is not performed
- **eight_bit** (*bool*) – Whether the stitched image should be saved as an 8bit image. Defaults to True, thus saving images as 8bit
- **show_arrow** (*bool*) – Whether an arrow should be added to the image overlay that points to the annotation in the stitched image. Defaults to True, thus adding an arrow to the overlay
- **max_processes** (*int*) – The number of parallel processes that should be used to process the batches. Be careful, each batch needs a lot of memory
- **enhance_contrast** (*bool*) – Whether contrast enhancement should be performed on the images before stitching. Defaults to True (thus enhancing contrast in the images)

parse_create_classifier_csv_batches(*batch_size: int*, *classifier_csv_path: str*, *highmag_layer: str = 'highmag'*)

Creates the batch csv files of annotation_tiles based on classifier output

Calls the MapsXmlParser to parse the XML file of the acquisition and save the annotation_tiles as a csv in batches

Parameters

- **batch_size** (*int*) – Batch size of the csv files
- **classifier_csv_path** (*str*) – Path to the classifier output csv file
- **highmag_layer** (*str*) – Name of the image layer in MAPS for which tiles containing annotations should be found. Defaults to 'highmag'

Returns

First value: The annotation_tiles dictionary. Second value: A list of the filenames of the csv files that were created

Return type list

parse_create_csv_batches(*batch_size: int*, *highmag_layer: str = 'highmag'*)

Creates the batch csv files of annotation_tiles

Calls the MapsXmlParser to parse the XML file of the acquisition and save the annotation_tiles as a csv in batches

Parameters

- **batch_size** (*int*) – Batch size of the csv files
- **highmag_layer** (*str*) – Name of the image layer in MAPS for which tiles containing annotations should be found. Defaults to 'highmag'

Returns

First value: The annotation_tiles dictionary. Second value: A list of the filenames of the csv files that were created

Return type list

static process_stitching_params(*stitch_params*, *annotation_coordinates*, *stitch_threshold*, *original_positions*, *center_index: int*, *logger*)

Calculates the position of the annotation in the stitched image and decides if stitching worked well

Based on the stitch_threshold, this function decides whether the stitching has worked well. If any image was moved by more than the threshold, it returns False.

Parameters

- **stitch_params** (*np.array*) – Array of the stitching parameters calculated by imageJ stitching
- **annotation_coordinates** (*np.array*) – Array of the coordinates of the annotation in the image before stitching
- **stitch_threshold** (*int*) – Threshold to decide whether stitching should be performed
- **original_positions** (*np.Array*) – Array of the initial positions of the images before stitching, used to calculate the shift by stitching
- **center_index** (*int*) – Index of which tile in the stitched image was the original center. Used to calculate the position of the annotation in the stitched image
- **logger** (*Logging*) – Logging object that is configured for the logging either in multiprocessing or normal processing

Returns

First value is a bool that informs whether the stitching should be done. Second value is an array with the coordinates of the annotations in the stitched image

Return type list

stitch_annotated_tiles(*annotation_tiles: dict*, *logger*, *stitch_threshold: int = 1000*, *eight_bit: bool = True*, *show_arrow: bool = True*, *enhance_contrast: bool = True*)

Stitches 3x3 images for all annotations in annotation_tiles

Goes through all annotations in annotation_tiles dict, load the center file and the existing surrounding files. Sets up the stitching configuration according to the neighboring tiles and calculates the stitching parameters. If the calculated stitching moves all images by less than the threshold in any direction, it performs the stitching. Otherwise, a log message is made and the center image is copied to the results folder. Finally, it sets the pixel size and converts the stitched image to 8bit before saving it to disk. Everything is performed using pyimagej api to use imageJ Java APIs. Can deal with 3x3, 3x2, 2x3 and 2x2 squares with 10% overlap.

Parameters

- **annotation_tiles** (*dict*) – annotation_tiles dictionary, e.g. from MapsXmlParser. See MapsXmlParser for content details. Needs to contain img_path, pixel_size, Annotation_tile_img_position_x, Annotation_tile_img_position_y, surrounding_tile_names & surrounding_tile_exists for this function to work
- **logger** (*Logging*) – Logging object that is configured for the logging either in multiprocessing or normal processing
- **stitch_threshold** (*int*) – Threshold to judge stitching quality by. If images would be moved more than this threshold, the stitching is not performed

- **eight_bit** (*bool*) – Whether the stitched image should be saved as an 8bit image. Defaults to True, thus saving images as 8bit
- **show_arrow** (*bool*) – Whether an arrow should be added to the image overlay that points to the annotation in the stitched image. Defaults to True, thus adding an arrow to the overlay
- **enhance_contrast** (*bool*) – Whether contrast enhancement should be performed on the images before stitching. Defaults to True (thus enhancing contrast in the images)

Returns annotation_tiles, now includes information about the position of the annotation in the stitched image

Return type dict

stitch_batch(*annotation_csv_path*, *stitch_threshold: int = 1000*, *eight_bit: bool = True*, *show_arrow: bool = True*, *enhance_contrast: bool = True*, *multiprocessing_logger: bool = False*)

Submits the stitching of a batch, the writing of an updated csv file and the deletion of the old csv file

Parameters

- **annotation_csv_path** (*Path*) – The path to the folder containing the annotation_tiles csvs.
- **stitch_threshold** (*int*) – Threshold to judge stitching quality by. If images would be moved more than this threshold, the stitching is not performed
- **eight_bit** (*bool*) – Whether the stitched image should be saved as an 8bit image. Defaults to True, thus saving images as 8bit
- **show_arrow** (*bool*) – Whether an arrow should be added to the image overlay that points to the annotation in the stitched image. Defaults to True, thus adding an arrow to the overlay
- **enhance_contrast** (*bool*) – Whether contrast enhancement should be performed on the images before stitching. Defaults to True (thus enhancing contrast in the images)
- **multiprocessing_logger** (*bool*) – Whether a multiprocessing logger or a normal logger should be used. Defaults to False, thus using a normal logger

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

sites_of_interest_parser, ??
stitch_MAPS_annotations, 5

INDEX

A

annotation_tiles (*sites_of_interest_parser.MapsXmlParser attribute*), 2
annotations (*sites_of_interest_parser.MapsXmlParser attribute*), 1

B

base_header (*stitch_MAPS_annotations.Stitcher attribute*), 6

C

calculate_absolute_tile_coordinates() (*sites_of_interest_parser.MapsXmlParser method*), 2
combine_csvs() (*stitch_MAPS_annotations.Stitcher method*), 6
convert_img_path_to_local_path() (*sites_of_interest_parser.MapsXmlParser method*), 2
convert_windows_pathstring_to_path_object() (*sites_of_interest_parser.MapsXmlParser static method*), 2
create_logger() (*sites_of_interest_parser.MapsXmlParser static method*), 3
csv_base_path (*stitch_MAPS_annotations.Stitcher attribute*), 5

D

determine_surrounding_tiles() (*sites_of_interest_parser.MapsXmlParser method*), 3

E

extract_layers_and_annotations() (*sites_of_interest_parser.MapsXmlParser method*), 3

F

find_annotation_tile() (*sites_of_interest_parser.MapsXmlParser method*), 3

G

get_relative_tile_locations() (*sites_of_interest_parser.MapsXmlParser method*), 3

I

img_height (*sites_of_interest_parser.MapsXmlParser attribute*), 2
img_width (*sites_of_interest_parser.MapsXmlParser attribute*), 2

L

layers (*sites_of_interest_parser.MapsXmlParser attribute*), 1
load_annotations_from_csv() (*sites_of_interest_parser.MapsXmlParser static method*), 4
load_xml() (*sites_of_interest_parser.MapsXmlParser static method*), 4
local_contrast_enhancement() (*stitch_MAPS_annotations.Stitcher static method*), 6

M

manage_batches() (*stitch_MAPS_annotations.Stitcher method*), 7
MapsXmlParser (*class in sites_of_interest_parser*), 1

O

output_path (*stitch_MAPS_annotations.Stitcher attribute*), 5

P

parse_classifier_output() (*sites_of_interest_parser.MapsXmlParser method*), 4
parse_create_classifier_csv_batches() (*stitch_MAPS_annotations.Stitcher method*), 7
parse_create_csv_batches() (*stitch_MAPS_annotations.Stitcher method*), 7
parse_xml() (*sites_of_interest_parser.MapsXmlParser method*), 4

pixel_size (*sites_of_interest_parser.MapsXmlParser attribute*), 2
process_stitching_params() (*stitch_MAPS_annotations.Stitcher static method*), 8
project_folder_path (*sites_of_interest_parser.MapsXmlParser attribute*), 1
project_folder_path (*stitch_MAPS_annotations.Stitcher attribute*), 5
project_name (*stitch_MAPS_annotations.Stitcher attribute*), 5

S

save_annotation_tiles_to_csv() (*sites_of_interest_parser.MapsXmlParser static method*), 4
sites_of_interest_parser (*module*), 1
stitch_annotated_tiles() (*stitch_MAPS_annotations.Stitcher method*), 8
stitch_batch() (*stitch_MAPS_annotations.Stitcher method*), 9
stitch_MAPS_annotations (*module*), 5
stitch_radius (*sites_of_interest_parser.MapsXmlParser attribute*), 2
stitch_radius (*stitch_MAPS_annotations.Stitcher attribute*), 5
Stitcher (*class in stitch_MAPS_annotations*), 5

T

tiles (*sites_of_interest_parser.MapsXmlParser attribute*), 1

X

XmlParsingFailed, 5