
vipertools

Release 0.0.1

Sophia Maedler

Aug 31, 2022

MODULES:

1	Modules	2
1.1	parse	2
1.2	stitch	2
1.3	image processing	4
2	Tutorials	5
2.1	Parsing and Stitching Data from Opera Phenix	5
	Python Module Index	7
	Index	8

Welcome the documentation of VIPERTOOLS!

MODULES

1.1 parse

Contains functions to parse imaging data into a usable formats for downstream pipelines.

`vipertools.parse.parse_phenix(phenix_dir, flatfield_exported=True, parallel=False, WGAbackground=False)`

Function to automatically rename TIFS exported from Harmony into a format where row and well ID as well as Tile position are indicated in the file name. Example of an exported file name: “Timepoint{#}_Row{#}_Well{#}_{channel}_zstack{#}_r{#}_c{#}.tif”

Parameters

- **phenix_dir** – Path indicating the exported harmony files to parse.
- **flatfield_exported** – boolean indicating if the data was exported from harmony with or without flatfield correction.
- **parallel** – boolean value indicating if the data parsing should be performed with parallelization or without (CURRENTLY NOT FUNCTIONAL ONLY USE AS FALSE)
- **WGAbackground** – export second copy of WGA stains for background correction to improve segmentation. If set to False not performed. Else enter value of the channel that should be copied and contains the WGA stain.

`vipertools.parse.sort_timepoints(parsed_dir)`

Additionally sort generated timecourse images according to well and tile position. Function generates a new folder called `timecourse_sorted` which contains a unique folder for each unique tile position containing all imaging data (i.e. zstacks, timepoints, channels) of that tile. This function is meant for quick sorting of generated images for simple import of e.g. timecourse experiments into FIJI.

Parameters *parsed_dir* – filepath to parsed images folder generated with the function `parse_phenix`.

1.2 stitch

Collection of functions to perform stitching of parsed image Tiffs.

`vipertools.stitch.generate_stitched(input_dir, slidename, pattern, outdir, overlap=0.1, max_shift=30, stitching_channel='Alexa488', no_rescale_channel=None, crop={'bottom': 0, 'left': 0, 'right': 0, 'top': 0}, plot_QC=True, filetype=['.tif'], WGAchannel=None, do_intensity_rescale=True, export_XML=True)`

Function to generate a scaled down thumbnail of stitched image. Can be used for example to get a low resolution overview of the scanned region to select areas for exporting high resolution stitched images.

Parameters

- **input_dir** – Path to the folder containing exported TIF files named with the following naming convention: “Row{#}_Well{#}_{channel}_zstack{#}_r{#}_c{#}.tif”. These images can be generated for example by running the `vipertools.parse.parse_phenix()` function.
- **pattern** – Regex string to identify the naming pattern of the TIFs that should be stitched together. For example: “Row1_Well2_{channel}_zstack3_r{row:03}_c{col:03}.tif”. All values in { } indicate those which are matched by regex to find all matching tifs.
- **outdir** – path indicating where the stitched images should be written out
- **overlap** – value between 0 and 1 indicating the degree of overlap that was used while recording data at the microscope.
- **max_shift** – value indicating the maximum threshold for tile shifts. Default value in ashlar is 15.
- **name** – string indicating the slidename that is added to the stitched images generated
- **stitching_channel** – string indicating the channel name on which the stitching should be calculated. the positions for each tile calculated in this channel will be passed to the other channels.
- **crop** – dictionary of the form {‘top’:0, ‘bottom’:0, ‘left’:0, ‘right’:0} indicating how many pixels (based on a generated thumbnail, see `vipertools.stitch.generate_thumbnail`) should be cropped from the final image in each indicated dimension. Leave this set to default if no cropping should be performed.
- **plot_QC** – boolean value indicating if QC plots should be generated
- **filetype** – list containing any of [“.tif”, “.ome.zarr”, “.ome.tif”] defining to which type of file the stitched results should be written. If more than one element all export types will be generated in the same output directory.
- **WGChannel** – string indicating the name of the WGA channel in case an illumination correction should be performed on this channel
- **do_intensity_rescale** – boolean value indicating if the `rescale_p1_P99` function should be applied before stitching or not. Alternatively partial then it will only rescale those channels provided as a list in `rescale_channels`
- **export_XML** – boolean value. If true then an xml is exported when writing to .tif which allows for the import into BIAS.

```
vipertools.stitch.generate_thumbnail(input_dir, pattern, outdir, overlap, name,
                                   stitching_channel='DAPI', export_examples=False,
                                   do_intensity_rescale=True)
```

Function to generate a scaled down thumbnail of stitched image. Can be used for example to get a low resolution overview of the scanned region to select areas for exporting high resolution stitched images.

Parameters

- **input_dir** – Path to the folder containing exported TIF files named with the following naming convention: “Row{#}_Well{#}_{channel}_zstack{#}_r{#}_c{#}.tif”. These images can be generated for example by running the `vipertools.parse.parse_phenix()` function.
- **pattern** – Regex string to identify the naming pattern of the TIFs that should be stitched together. For example: “Row1_Well2_{channel}_zstack3_r{row:03}_c{col:03}.tif”. All values in { } indicate those which are matched by regex to find all matching tifs.
- **outdir** – path indicating where the stitched images should be written out
- **overlap** – value between 0 and 1 indicating the degree of overlap that was used while recording data at the microscope.
- **name** – string indicating the slidename that is added to the stitched images generated

- **export_examples** – boolean value indicating if individual example tiles should be exported in addition to performing thumbnail generation.
- **do_intensity_rescale** – boolean value indicating if the `rescale_p1_P99` function should be applied before stitching or not.

1.3 image processing

Contains functions to perform standard image processing steps, e.g. downsampling.

`vipertools.image_processing.downsample_folder(folder_path, num_threads=20, file_ending=('tif', 'tiff'))`

Multi-Threaded Function to downsample image equivalent to 2x2 binning. Overwrites original images! Do not run multiple times. Output is saved as uint16.

Parameters

- **folder_path** – string indicating the folder containing all the image files that should be downsampled
- **num_threads** – int, number of threads for multithreading
- **file_ending** – string or tuple of strings indicating which file ending the script should filter for in the indicated folder

`vipertools.image_processing.downsample_img(img_path)`

Function to downsample a single image equivalent to 2x2 binning using the mean between pixels. Overwrites the original image(!), do not run multiple times on the same image.

Parameters **img_path** – string indicating the file path to the .tif file which should be downsampled.

TUTORIALS

2.1 Parsing and Stitching Data from Opera Phenix

First you need to export your data from harmony and rename the path to eliminate any spaces in the name. Then you can run the following script to parse and stitch your data.

Listing 1: example script for parsing and stitching phenix data

```
#import relevant libraries
import os
from vipertools.parse import parse_phenix
from vipertools.stitch import generate_stitched

#parse image data
path = "path to exported harmony project without any spaces"
parse_phenix(path, flatfield_exported = True, parallel = False)

#define important information for your slide that you want to stitch

# the code below needs to be run for each slide contained in the imaging experiment!
# Can be put into a loop for example to automate this or also can be subset to
→seperate
# jobs to run on for example the hpc

input_dir = os.path.join(path, "parsed_images")
slidename = "Slide1"
outdir = os.path.join(path, "stitched", slidename)
overlap = 0.1 #adjust in case your data was acquired with another overlap

#define parameters to find correct slide in experiment folder
row = 1
well = 1
zstack_value = 1
timepoint = 1

#define on which channel should be stitched
stitching_channel = "Alexa647"
output_filetype = [".tif", ".ome.zarr"] #one of .tif, .ome.tif, .ome.zarr (can pass
→several if you want to generate all filetypes)

#adjust cropping parameter
crop = {'top':0, 'bottom':0, 'left':0, 'right':0} #this does no cropping
#crop = {'top':72, 'bottom':52, 'left':48, 'right':58} #this is good default values for an
→entire PPS slide with cell culture samples imaged with my protocol
```

(continues on next page)

(continued from previous page)

```
#create output directory if it does not exist
if not os.path.exists(outdir):
    os.makedirs(outdir)

#define patter to recognize which slide should be stitched
#remember to adjust the zstack value if you aquired zstacks and want to stitch a
↪speciifc one in the parameters above

pattern = "Timepoint"+str(timepoint)+"_Row"+ str(row) + "_" + "Well" + str(well) + "_
↪{channel}_"+"zstack"+str(zstack_value)+"_r{row:03}_c{col:03}.tif"
generate_stitched(input_dir,
                  slidename,
                  pattern,
                  outdir,
                  overlap,
                  crop = crop ,
                  stitching_channel = stitching_channel,
                  filetype = output_filetype)
```


PYTHON MODULE INDEX

V

`vipertools.image_processing`, [4](#)
`vipertools.parse`, [2](#)
`vipertools.stitch`, [2](#)

INDEX

D

`downsample_folder()` (*in module vipertools.image_processing*), 4
`downsample_img()` (*in module vipertools.image_processing*), 4

G

`generate_stitched()` (*in module vipertools.stitch*), 2
`generate_thumbnail()` (*in module vipertools.stitch*), 3

M

module
 vipertools.image_processing, 4
 vipertools.parse, 2
 vipertools.stitch, 2

P

`parse_phenix()` (*in module vipertools.parse*), 2

S

`sort_timepoints()` (*in module vipertools.parse*), 2

V

vipertools.image_processing
 module, 4
vipertools.parse
 module, 2
vipertools.stitch
 module, 2