vipertools

Release 0.0.1

Sophia Maedler

May 16, 2023

MODULES:

This python module contains useful wrapper functions to perform stitching with the *Ashlar API <https://labsyspharm.github.io/ashlar/>* directly in python. In addition it contains useful data parsing functions to make imaging data aquired with the *Perkinelmer Opera Phenix Microscope <https://www.perkinelmer.com/uk/product/opera-phenix-plus-system-hh14001000>* accessible to the Ashlar API to perform stitching or also to other downstream applications.

A stitching workflow which stitches imaging data aquired from one well on the OperaPhenix would e.g. look like this:

```
from vipertools.parse import parse_phenix

#after exporting data from Harmony perform image parsing
phenix_dir = "path/to/exported/data"
parse_phenix(phenix_dir, WGAbackground = "Alexa488", export_as_symlink = True)

#the parsed results are written to "path/to/exported/data/parsed_images"
# after this has completed stitching can be performed

input_stitching = os.path.join(phenix_dir, "parsed_images")
outdir_sample = os.path.join(outdir, slidename)
outdir_merged = os.path.join(outdir_sample, 'merged_files')

#create output directory if it does not already exist
if not os.path.exists(outdir_sample):
   os.makedirs(outdir_sample)
if not os.path.exists(outdir_merged):
   os.makedirs(outdir_merged)

#define parameters specific to experiment
RowID = 1
WellID = 1
zstack_value = 1 #since we only took 1 zstack!
overlap = 0.1 #fraction indicating with how much overlap the image tiles were acquired

#define file pattern for reading
pattern = "Timepoint001_Row"+ str(RowID).zfill(2) + "_" + "Well" + str(WellID).
↪zfill(2) + "_{channel}_"+"zstack"+str(zstack_value).zfill(3)+"_r{row:03}_c{col:03}.
↪tif"

generate_stitched(input_stitching,
                  slidename,
                  pattern,
                  outdir_merged,
                  overlap,
                  stitching_channel = "Alexa488",
                  crop = eval(crop),
                  plot_QC = True,
                  filetype = [".tif", "ome.zarr"],
                  WGAchannel = "Alexa488",
                  do_intensity_rescale = True,
                  export_XML = True)
```

The generated stitched images can then be used for downstream processing for example using the SPARCSpy pipeline or also using *BIAS <https://single-cell-technologies.com/bias-2/>*.

CHAPTER

ONE

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*, *WGAbackground=False*, *export_meta=True*, *export_as_symlink=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** (*bool*) – boolean indicating if the data was exported from harmony with or without flatfield correction.
- **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.
- **export_meta** – boolean value indicating if a metadata file containing, tile positions, exact time of measurement etc. should be written out.
- **export_as_symlink** – boolean value indicating if the parsed files should be copied or symlinked. If set to true can lead to issues when accessing remote filesystems from differentoperating systems

vipertools.parse.**sort_timepoints**(*parsed_dir*, *use_symlink=False*)

Additionally sort generated timecourse images according to well and tile position. Function generates a new folder called timecourse_sorted which contains a unqiue 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.
- **use_symlonks** (*bool*) – boolean value indicating if the images should be copied as symlinks or as regular files. Symlinks can potentially cause issues if using the data on different OS but is signficiantly faster and does not produce as much data overhead.

vipertools.parse.**sort_wells**(*parsed_dir*, *use_symlink=False*, *assign_random_id=False*)

Sort acquired phenix images into unique folders for each well.

Parameters

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

- **use_symlink** – boolean value indicating if the images should be copied as symlinks to their new destination
- **assign_random_id** – boolean value indicating if the images in the sorted wells folder should be prepended with a random id.

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'*, *crop={'bottom': 0, 'left': 0, 'right': 0, 'top': 0}*, *plot_QC=True*, *filetype=['.tif']*, *WGAchannel=None*, *do_intensity_rescale=True*, *no_rescale_channel=None*, *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** (*str*) – Path to the folder containing exported TIF files named with the following nameing convention: "Row{#}_Well{#}_{channel}_zstack{#}_r{#}_c{#}.tif". These images can be generated for example by running the vipertools.parse.parse_phenix() function.
- **slidename** (*str*) – string indicating the slidename that is added to the stitched images generated
- **pattern** (*str*) – 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** (*str*) – path indicating where the stitched images should be written out
- **overlap** (*float between 0 and 1*) – value between 0 and 1 indicating the degree of overlap that was used while recording data at the microscope.
- **max_shift** (*int*) – value indicating the maximum threshold for tile shifts. Default value in ashlar is 15. In general this parameter does not need to be adjusted but it is provided to give more control.
- **stitching_channel** (*str*) – 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** (*bool*) – boolean value indicating if QC plots should be generated
- **filetype** (*[str]*) – list containing any of [".tif", ".ome.zarr", ".ome.tif"] defining to which type of file the stiched results should be written. If more than one element is present in the list all export types will be generated in the same output directory.
- **WGAchannel** (*str*) – string indicating the name of the WGA channel in case an illumination correction should be performed on this channel
- **do_intensity_rescale** (*bool*) – boolean value indicating if the rescale_p1_P99 function should be applied before stitching or not. Alternatively partial then those channels listed in no_rescale_channel will not be rescaled.

- **no_rescale_channel** (*None | [str]*) – either None or a list of channel strings on which no rescaling before stitching should be performed.
- **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** (*str*) – Path to the folder containing exported TIF files named with the following nameing convention: "Row{#}_Well{#}_{channel}_zstack{#}_r{#}_c{#}.tif". These images can be generated for example by running the vipertools.parse.parse_phenix() function.
- **pattern** (*str*) – 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')*, *N=2*)

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** (*str*) – string indicating the folder containing all the image files that should be downsampled
- **num_threads** (*int*) – number of threads for multithreading
- **file_ending** (*str | (str, str)*) – string or tuple of strings indicating which file ending the script should filter for in the indicated folder
- **N** (*int*) – number of pixels that should be binned together

vipertools.image_processing.**downsample_img**(*img_path*, *N=2*)

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

Parameters

- **img_path** (*str*) – string indicating the file path to the .tif file which should be downsampled.
- **N** (*int, default = 2*) – number of pixels that should be binned together using mean between pixels

CHAPTER

TWO

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 parses 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, export_as_symlink = True) #export as␣
↪symlink true enabled for better speed and to not duplicate data, set to False if␣
↪you want to work with hardcopies or plan on accessing the data from multiple OS

#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 when running on a 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 aquired 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␣
```

(continues on next page)

(continued from previous page)

```
↪entire PPS slide with cell culture samples imaged with the SPARCSpy protocol

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

#define pattern 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.zfill(3) +"_Row"+ str(row).zfill(2) + "_" + "Well
↪" + str(well).zfill(2) + "_{channel}_"+"zstack"+str(zstack_value).zfill(3)+"_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

INDEX