Phenix Data Analysis in Python

Created	@Apr 12, 2021 2:23 PM
Date	
Files	
Modified	@Apr 12, 2021 2:52 PM
Project	computational-cell-design
Starred	☐
Type	protocol

Overview & Scope

The Opera Phenix system is capable of imaging large FOVs using a tiled approach. The structure of a typical experiment might look like:

- For each timepoint—
  - For each well in a plate—
    - For each tile in a N-by-M rectangle of tiles—
      - For each brightfield and fluorescence channel selected—
        - Acquire an image.

The goal of this guide is to demonstrate how to bring data of this format into a Python data structure of the following format:

- For each well and fluorescence channel, an array of matrices for each timepoint where each matrix consists of all the corresponding tiles stitched together.

Exporting Data from the Phenix

Export raw images from the Phenix (either main or analysis computer):

- Settings → Data Management → Export → Measurements

The result will be a folder containing every single tile collected in TIFF format encoding the raw intensities from the imaging.

Caveat emptor: the export process is extremely slow and may need to run overnight. Also, the number of images exported into one folder is often enough to break Finder if you try to open the folder.

Understanding the Phenix Export Data Structure

The Images folder within the export contains tens of thousands (depending on experiment) fo TIFF tiles with names like:

`r01c01f01p01-ch1sk1fk1fl1.tiff`

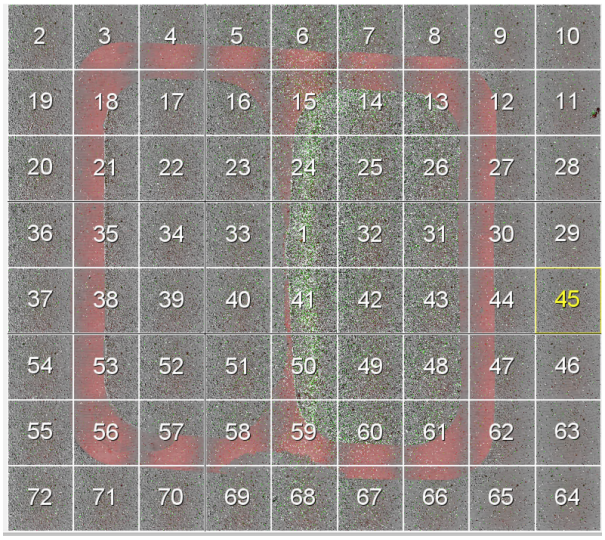
The structure of these names is: row - column - fov - plane - channel - sk - fk - fl

Figuring out which channel is which (e.g. BF, GFP, RFP, etc.) seems to need to be done manually.

I don't remember what sk, fk, and fl are but at least one of them likely addresses the Z plane.

Pretty annoying issue: The tiles (FOVs) are in a weird and kind of arbitary sequence (see how 1 is in the middle).

We will need to hard code this mapping in order to assemble the tiles correctly.



Setting up the analysis Jupyter Notebook

I have created a Jupyter notebook with some of the core functions needed to stitch together global images, perform flatfield correction, make movies, etc. If you need more advanced functionality feel free to reach out as I have a lot more code that I am omittting for simplicity.

If you have never used a Jupyter Notebook, you should Google a tutorial or ask me about the basics. It's easy but you have to install Jupyter/IPython and use the command line to start the notebook.

Basic Setup

1. Open the notebook and try to run the first cell. You will likely need to install some dependencies. You can do this easily using the following Bash (Terminal) command (example is for matplotlib, replace with the dependency of interest):
   - `python3 -m pip install matplotlib`
2. Change 'data_path' in the second cell to address your exported "Images" folder. I highly recommend keeping these files on an SSD as it will significantly speed up your analysis.

Core function: retrieve_and_stitch_image

The main function you need to understand and tweak based on your experiment is called `retrieve_and_stitch_image`. Let's take a look at the stub:

```
def retrieve_and_stitch_image(path, row, col, channel, timepoint, tophat=None, blur=None):
    """
    Retrieve all the tiles for a timepoint and channel and stitch into global image for a measurement.

    Parameters
```

```
    ----------
    directory : path to files
    row, col : position in multi-well plate (1-indexed)
    channel : index of fluorescence/brightfield channel. Probably on a case by case basis.
    timepoint : index of the timepoint to retrieve (1-indexed)
    tophat : kernel size to use for tophat flatfield correction (None for no correction)
    blur : gaussian width to use for blur flatfield correction (None for no correction)

    Returns
    -------
    Pillow Image
    """
```

Within this function, you need to set up some parameters based on your experiment:

1. Change n_rows and n_cols to correspond with the width and height of your grid of tiles (9 by 8 in the screenshot above).
    - If your FOV is not rectangular, you need to use the largest dimension (widest and tallest strips).
2. Change tile_size to correspond to the pixel size of each tile. I haven't imaged with anything other than 10X air and 2×2 binning (which makes 1080×1080 squares) but you can figure this out by opening a tile in Photoshop or something.
3. Change the tiling_map matrix to correspond with the mapping from Harmony when you go to global image and turn on field_map (see screenshot above).
    - If your FOV is not rectangular, this matrix still needs to be rectangular. Put in '-1' anywhere that there will not be an image.

That's it. Run all the cells (shift enter until you get to the bottom. I've left some code in at the bottom that will attempt to retrieve and stitch well A1 for each channel and display it in the notebook.

Quick overview of cool functions

`render_overlay_movie` is probably of interest to people in the lab. The usage is as follows:

```
# Get the timecourse for the channel and well of interest:
# This will get well A1, channel 1, timepoints 1-72
timecourse = retrieve_timecourse(data_path, 1, 1, 1, 1, 72)

# Now render a movie for the timecourse:
# This will save the movie as 'movie.mp4' in the notebook directory
# You can tweak the movie size and frame rate; ask me if you need help with this.
render_movie_for_timecourse(timecourse, 'movie')
```