Segmentation of antibiotic-treated cells using the Otsu threshold

Alexandros Papagiannakis, Christine Jacobs-Wagner lab, Sarafan ChEM-H, Stanford University / HHMI, 2024

This notebook provides an example of cell segmentation using the functions in the otsu_segmentation_class which is defined in the Otsu_phase_segmentation_ghv.py script.

1. Run the script and initialize the class

In [ ]:
```
%run "...Otsu_phase_segmentation_ghv.py"

tif_path = "...CJW7323_Cepha_A22_timelapse_TIFF"
save_path = "...results"
experiment = 'CJW7323_Cepha_A22_timelapse'
channels = ['Phase','mCherry','GFP']
frame_interval = 2.5 #min
frame_range = (0,90) #frames
channel_interval =  [1,2,2] #every nth

cell_obj = otsu_segmentation_class(tif_path, experiment, channels, frame_interval,
                                   frame_range, channel_interval, save_path)
```

2. Segment cells

In [ ]:
```
gaussian_level = 0
otsu_fraction = 0.8
dilation_rounds = 1
hole_fill = True
closing = True
cell_obj.segment_phase_images(gaussian_level, otsu_fraction, dilation_rounds,
                              hole_fill, closing)
```

3. Curate bad segmentations

In [ ]:
```
# saves a Pandas dataframe with the sinuosity of the cell, the max width and
# the area. These statistics can be used to curate the segmentation masks.
cell_obj.label_curation()

# Curation based on the..
sinuosity_range = (0.0015,1) # in pixels (draws a bivariate medial axis)
max_distance_range = (7.2,10) #in pixels
cell_area_range = (300,3000) # in pixels
cell_obj.get_good_cells(sinuosity_range, max_distance_range, cell_area_range)

# This function can be used to load the curated cell IDs
cell_obj.load_good_cells()

# This function can be used to load the binary cell masks
cell_obj.load_otsu_masks()
```

4. Cell tracking

In [ ]:
```
# Gets the cell centroids in a Pandas dataframe
cell_obj.get_centroid_dataframe()
# Links the centroids using a search radius and a relative area increase range between timepoints
max_radius = 20 # in pixels
area_increase_ratio = (0.0005,0.15) #x100%
link_dict = cell_obj.link_cells(max_radius, area_increase_ratio)
# Uses the linked centroids to generate cell growth trajectories
cell_obj.connect_cells_into_lineages()
```

5. Draw medial axis and project pixels along the cell length

In [ ]:
```
# draw the bivariate medial axis
cell_obj.draw_medial_axes(min_trajectory_length=18,
                          half_angle_px = 12, curated=True)
# Uses the medial axis statistics (sinuosity and max pixel distance from the medial axis)
# to curate the badly segmented cells or the cells
# with a bad medial axis definition
cell_obj.curate_medial_axes()
# projects each pixel value on the medial axis and the cell length
cell_obj.apply_oned_fluorescence()
```

6. Segment polysome accumulations and nucleoid objects

In [ ]:
```
#An LoG - adaptive filter is used to segment the fluorescent objects
# Filter parameters for RplA-GFP: [2,1000,75,4,7,-2,0]
# Min RplA-GFP area: 25
# Filter parameters for HupA-mCherry: [2,1000,90,5,9,-1,0]
# Min HupA-mCherry area: 40
cell_obj.apply_oned_fluorescence()
cell_obj.apply_fluorescence_segmentation('GFP',  [2,1000,75,4,7,-2,0], 25)
cell_obj.apply_fluorescence_segmentation('mCherry', [2,1000,90,5,9,-1,0], 40)
# The segmentation labels were used to count the polysome accumulations and the nucleoid objects
```