**Cell Separation Image Analysis Pipeline**

###############################################################################

# Description: #

# This script allows the semi automatic analysis of cell separations in 2D #

# images, and can further perform comparaisons of mean cell separation area #

# between two conditions or genotypes, as well as the analysis of cell #

# separation orientation and anisotropy of multiple conditions or genotypes. #

# This is a "semi-automatic" pipeline, because the analysis requires a crucial#

# manual step that need to be be performed for each image before running the #

# script: an appropriate threshold properly separating the cell signal from #

# the cell separation background has to be define manually. #

# For more details see Verger et al. (2018). The tension-adhesion nexus in #

# plant epidermis. #

# #

# Prerequist: (e.g. in ImageJ) #

# - From a raw confocal Z-stack, make a maximal intensity Z-projection to #

# obtain a 2D image. #

# - If necessary enhance the contrasts #

# - Smooth the image with a median filter to remove noise (Radius ~ 2 pixels).#

# - With the threshold tool in ImageJ, dertermine the appropriate threshold #

# that separates the best the cell separations from the cells. Change the #

# lut to "Grays" for easier visualization. #

# - Depending on the quality of the images it may be difficult or impossible #

# to segment the cell separtations based on a threshold. If some of your #

# images are in this situation, you may exclude them from your analysis. If #

# most of your images are in this situation, this image analysis pipeline #

# is not appropriate for your case. #

# - Then save the images in 8 bit .tif and add "\_xxxthld" at the end of the #

# name (e.g. "sample\_1\_055thld.tif). The value before thld is the threshold #

# value that will be used for the image segmentation(It has to be 3 digits).#

# - Then the file arborescence has to be organised as such: A "main" #

# directory (updir), containing subdirectories for each condition/mutant, #

# each containing all the images corresponding to the given #

# condition/mutant. #

# #

# Settings: #

# - Before running the script, define the paramenters in the section below #

# called "Parameters". #

# #

# Output: #

# - For each image, a .csv file is created containing for each segmented cell #

# separation, the label number, center position, area in pixels and um #

# square, the principal angle (orientation of the separation), the #

# anisotropy, the eigen values and vectors and the standard deviations. #

# - Also for each image, an inverted version of the image, overlaid with the #

# segmented areas as well as a representation of the anisotropy and #

# principal angle for each area, is saved as a vectorial .pdf. #

# - Finally, for each image, a polar histogram representing the distribution #

# of cell separation orientation in the image is created and saved as a #

# vectorial .pdf. #

# - If running the "Global\_Output\_Size", statistical tests will be run on the #

# compared samples, and the summary of these tests will be saved in a .txt #

# file in the "updir", as well as a box plot as a vectorial .pdf. #

# - If running the "Global\_Polarhist\_Output", a global polar histogarm will #

# be created for each condition/mutant and saved as a vectorial .pdf and #

# the circular mean angle, the resultant vector length and the mean #

# anisotropy will be computed and saved in a summary .txt file. #

# #

# This script was developed and run in the tissuelab environment of the #

# OpenAleaLab platform (github.com/VirtualPlants/tissuelab, Cerutti G et al., #

# (2017). Front Plant Sci 8:353. doi:10.3389/fpls.2017.00353). #

###############################################################################

import matplotlib.cm as cm

import matplotlib.colors as colors

import matplotlib.patheffects as patheffects

import matplotlib.pyplot as plt

import numpy as np

from openalea.container import array\_dict

import os

import pandas as pd

import pycircstat

from scipy import stats

from scipy.ndimage.io import imread

import scipy.ndimage as nd

# Parameters==================================================================#

###############################################################################

# Before running the script, define all the parameters. #

# - Define a directory containing all the data to analyse and compare (updir).#

# - Define the pixel size in micrometer (pixel\_size). #

# - Define min and max area of crack to eliminate areas that are too small #

# (min), and/or the background (max). #

# - Define the threshold type "min" or "max". "max" will detect and segment #

# zones with lowest intensity of signal (black gaps between separated cells). #

# - Output global cracks size analysis? True or False. This is if you want to #

# run the analysis of gap size comparing two mutants or conditions in the #

# updir folder. #

# - Output global crack orientation analysis? True or False. This is if you #

# want to run the analysis of gap orientation. #

###############################################################################

updir = "/home/ImageAnalysis/CellSeparationAnalysis/"

pixel\_size = 0.363636 # 0.363636um square is the pixel size

min\_area\_of\_crack = 100 # Size in pixels

max\_area\_of\_crack = 100000

thld\_type = 'max'

Global\_Output\_Size = False

Global\_Polarhist\_Output = False

# Parameters==================================================================#

# Functions===================================================================#

def Segment\_cell\_separation(img, threshold=150, threshold\_type=thld\_type,

min\_area=min\_area\_of\_crack,

max\_area=max\_area\_of\_crack):

if threshold\_type == 'max':

img\_regions = nd.label(img < int(threshold))

elif threshold\_type == 'min':

img\_regions = nd.label(img > int(threshold))

labelled\_regions = img\_regions[0]

labels = np.arange(img\_regions[1])+1

region\_areas = array\_dict(dict(zip(labels,

nd.sum(np.ones\_like(labelled\_regions),

labelled\_regions,

index=labels))))

if max\_area is not None:

regions\_to\_remove = region\_areas.keys()[region\_areas.values() > max\_area]

for l in regions\_to\_remove:

labelled\_regions[labelled\_regions == l] = 0

regions\_to\_remove = region\_areas.keys()[region\_areas.values() < min\_area]

for l in regions\_to\_remove:

labelled\_regions[labelled\_regions == l] = 0

return labelled\_regions

def labelled\_region\_dataframe(labelled\_regions, px\_size=pixel\_size):

region\_labels = np.unique(labelled\_regions)[1:]

region\_data = dict()

for field in ['label', 'center', 'center\_x', 'center\_y', 'area\_px',

'area\_um\_sq', 'principal\_angle', 'anisotropy',

'eigen\_values', 'eigen\_vectors', 'standard\_deviations',

'standard\_deviations0', 'standard\_deviations1']:

region\_data[field] = []

for l in region\_labels:

coords = np.transpose(np.where(labelled\_regions == l))

region\_center = np.mean(coords, axis=0)

region\_covariance = np.cov(coords, rowvar=False)

eigen\_values, eigen\_vectors = np.linalg.eig(region\_covariance)

eigen\_vectors = eigen\_vectors[np.argsort(-np.abs(eigen\_values))]

eigen\_values = eigen\_values[np.argsort(-np.abs(eigen\_values))]

region\_data['label'] += [l]

region\_data['center'] += [region\_center]

region\_data['center\_x'] += [region\_center[0]]

region\_data['center\_y'] += [region\_center[1]]

region\_data['area\_px'] += [len(coords)]

region\_data['area\_um\_sq'] += [len(coords)\*px\_size\*px\_size]

region\_data['eigen\_values'] += [eigen\_values]

region\_data['standard\_deviations'] += [np.sqrt(np.abs(eigen\_values))]

region\_data['standard\_deviations0'] += \

[np.sqrt(np.abs(eigen\_values[0]))]

region\_data['standard\_deviations1'] += \

[np.sqrt(np.abs(eigen\_values[1]))]

region\_data['eigen\_vectors'] += [eigen\_vectors]

region\_data['anisotropy'] += [np.abs(eigen\_values[0]/eigen\_values[1])]

region\_data['principal\_angle'] += \

[((np.sign(eigen\_vectors[0][0]) \*

np.arccos(eigen\_vectors[0][1])\*180./np.pi)) % 180]

return pd.DataFrame.from\_dict(region\_data)

""" Script:

>From the designated directory (updir),

finds subdirectories containing the experiments/conditions to analyse.

"""

for dirname in sorted(os.listdir(updir)):

drpath = updir + dirname

if os.path.isdir(drpath):

# print drpath

print "Folder analyzed: " + dirname

""" For each .tif image in each directory: """

for img\_file in sorted(os.listdir(drpath)):

if "thld.tif" in img\_file:

imgpath = drpath + "/" + img\_file

# print 'image path: ' + imgpath

img = imread(imgpath).astype(float)

print "image analyzed: " + img\_file

# Extracts the threshold value from the title

thld1 = img\_file[:-8]

thld = thld1[-3:]

# Trims the 0 at the begining of the value if it is lower than 100

if thld[:1]is '0':

thld = thld[1:]

""" Runs the segmentation."""

labelled\_regions = Segment\_cell\_separation(img, threshold=thld)

print "segmentation done"

""" Labels of segmented regions, shuffled for visualization."""

labels = np.unique(labelled\_regions)[1:]

np.random.shuffle(labels)

shuffled\_labels = array\_dict(labels,

keys=np.unique(labelled\_regions)[1:])

shuffled\_labels[0] = 0

labelled\_regions = shuffled\_labels.values(labelled\_regions)

""" Makes and saves an inverted (pixel value) .tif image."""

figure = plt.figure(img\_file + "\_inv")

figure.clf()

# figure.patch.set\_facecolor('w')

figure.gca().set\_xlim(0, img.shape[1])

figure.gca().set\_ylim(img.shape[0], 0)

figure.set\_size\_inches(5, 5)

figure.gca().imshow(img, cmap='gray\_r', vmin=0, vmax=255)

figure.savefig(imgpath[:-4]+"\_inv.pdf", dpi=300)

""" Makes and saves an overlay of the inverted image with the detected

labelled regions."""

figure.gca().imshow(np.ma.masked\_where(labelled\_regions == 0, labelled\_regions),

cmap='winter', alpha=1, interpolation='none')

figure.savefig(imgpath[:-4]+"\_inv\_sep\_labels.pdf", dpi=300)

""" Makes the data frame."""

data = labelled\_region\_dataframe(labelled\_regions)

""" Generates and plots the separation orientation axes."""

tensor\_factor = 2

for center, vals, vecs, angle, area in zip(data['center'],

data['standard\_deviations'],

data['eigen\_vectors'],

data['principal\_angle'],

data['area\_px']):

figure.gca().scatter(center[1], center[0], color='k', s=0)

for val, vec in zip(vals, vecs):

figure.gca().plot([center[1]+tensor\_factor\*val\*vec[1],

center[1]-tensor\_factor\*val\*vec[1]],

[center[0]-tensor\_factor\*val\*vec[0],

center[0]+tensor\_factor\*val\*vec[0]],

color='r', linewidth=2,

path\_effects=[patheffects.withStroke(linewidth=4, foreground='w')])

""" Saves an overlay of the inverted image with the detected labelled

regions and the cell separation orientations."""

figure.savefig(imgpath[:-4]+"\_inv\_sep\_orientations.pdf", dpi=300)

""" Makes and saves a polar histogram of cracks orientations."""

figure = plt.figure(img\_file + "\_hist")

figure.clf()

figure.patch.set\_facecolor('w')

figure.set\_size\_inches(5, 5)

ax = figure.add\_subplot(111, polar=True)

colormap = 'plasma'

n\_bins = 36

# weights = data['anisotropy']\*data['area\_px']

# weights = data['anisotropy']

# weights = data['area\_px']

# weights = np.ones\_like(data['area\_px'])

# weights = np.array([np.max(np.abs(v)) for v in data['eigen\_values']])

weights = np.array([np.max(s) for s in data['standard\_deviations']])

for offset in [0, np.pi]:

histo, bins, patches = figure.gca().hist(offset+data['principal\_angle']\*np.pi/180.,

bins=offset+np.linspace(0, 180, n\_bins/2 + 1)\*np.pi/180.,

ec='k', weights=weights)

norm = colors.Normalize(0, histo.max())

for h, p in zip(histo, patches):

p.set\_facecolor(cm.get\_cmap(colormap)(norm(h)))

figure.gca().set\_yticks([])

figure.savefig(imgpath[:-4]+"\_sep\_hist.pdf", dpi=300)

# world.add(data,'crack\_data')

data.to\_csv(imgpath[:-4]+"\_sep\_log.csv")

print "Done analysing each image"

"""Global\_Output\_Size:

Makes and saves a boxplot to compare the sum of crack area per image in the

different conditions

"""

if Global\_Output\_Size is True:

print "Global cell separation size analysis"

fdatasize = open(updir + '/Global cell separation size analysis\_'

'Summary.txt', 'w')

sums = []

plotlabels = []

sum\_ = {}

normpop = []

for dirname in sorted(os.listdir(updir)):

drpath = updir + dirname

if os.path.isdir(drpath):

print dirname

sum\_[dirname] = []

# print drpath

for filename in os.listdir(drpath):

if '\_sep\_log.csv' in filename:

file\_data = pd.read\_csv(drpath + '/' + filename)

# print (drpath+"/"+filename)

sum\_area\_per\_img = np.sum(file\_data['area\_um\_sq'])

sum\_[dirname].append(sum\_area\_per\_img)

print sum\_[dirname]

mean\_area\_ALL\_img = np.mean(sum\_[dirname])

std\_area\_ALL\_img = np.std(sum\_[dirname])

print mean\_area\_ALL\_img

print std\_area\_ALL\_img

sums.append(sum\_[dirname])

print sum\_[dirname]

fdatasize.write("==> " + dirname + "\n" +

"mean area of separation (+-Std) --> " +

str(mean\_area\_ALL\_img) + "+-" +

str(std\_area\_ALL\_img) + "\n")

""" Shapiro's test for population normality."""

w, p = stats.shapiro(sum\_[dirname])

print p

if p > 0.05:

normpop.append(True)

print "population is normal"

fdatasize.write("--> population is normal --> "

"p-value (Shapiro's test) :" + str(p) + "\n\n")

else:

normpop.append(False)

print "population is NOT normal"

fdatasize.write("--> population is NOT normal --> "

"p-value (Shapiro's test) :" + str(p) + "\n\n")

plotlabels.append(dirname)

print normpop

print sums

print plotlabels

if False in normpop:

""" Non parametric Wilcoxon rank sum test."""

print "At least one sample does Not have a normal distibution" \

"--> wilcoxon rank sum test"

fdatasize.write("At least one sample does Not have a normal "

"distibution --> wilcoxon rank sum test" + "\n")

statrank, prank = stats.ranksums(\*sums)

if prank > 0.05:

print "--> populations are NOT statiscically different " \

"--> p-value is " + str(prank)

fdatasize.write("--> populations are NOT statiscically different "

"--> p-value (wilcoxon rank sum test) : " +

str(prank) + "\n\n")

else:

print "--> populations are statiscically different " \

"--> p-value is " + str(prank)

fdatasize.write("--> populations are statiscically different "

"--> p-value (wilcoxon rank sum test) : " +

str(prank) + "\n\n")

else:

""" Bartlett's test for equal variance."""

t, p2 = stats.bartlett(\*sums)

print p2

if p2 > 0.05:

equalvar = True

print "Groups have equal variances"

fdatasize.write("--> The groups have equal variances --> "

"p-value (Bartlett's test) : " + str(p2) + "\n\n")

else:

equalvar = False

print "Groups DO NOT have equal variances"

fdatasize.write("--> The groups Do NOT have equal variances --> "

"p-value (Bartlett's test) : " + str(p2) + "\n\n")

"""t-test (Welch's if variances are unequal)."""

t2, prob = stats.ttest\_ind(\*sums, equal\_var=equalvar)

print prob

if prob > 0.05:

print "populations are NOT statiscically different\np-value is " \

+ str(prob)

fdatasize.write("--> populations are NOT statiscically different "

"--> p-value (t-test, Welch's if variances are "

"unequal) : " + str(p2) + "\n\n")

else:

print "populations are statiscically different\np-value is " + \

str(prob)

fdatasize.write("--> populations are statiscically different "

"--> p-value (t-test, Welch's if variances are "

"unequal) : " + str(p2) + "\n\n")

fdatasize.close()

"""Boxplot."""

figure = plt.figure("Global\_crack\_Size")

figure.clf()

medianprops = {'color': 'red', 'linewidth': 2}

boxprops = {'color': 'black', 'linewidth': 3}

whiskerprops = {'color': 'black', 'linewidth': 3}

capprops = {'color': 'black', 'linewidth': 3}

flierprops = {'color': 'black', 'marker': 'x'}

ax = plt.boxplot(sums, labels=plotlabels, widths=0.7,

medianprops=medianprops, boxprops=boxprops,

whiskerprops=whiskerprops, capprops=capprops,

flierprops=flierprops, patch\_artist=True)

plt.tight\_layout()

plt.show()

figure.savefig(updir + "Boxplot.jpg")

"""Global\_Polarhist\_Output

Makes and saves a global polar histogarm for each experiment and gives the

circ\_mean angle, the resultant vector length and the mean anisotropy.

"""

if Global\_Polarhist\_Output is True:

print "Global orientation analysis"

Crack\_data\_ = {}

for dirname in sorted(os.listdir(updir)):

drpath = updir + dirname

if os.path.isdir(drpath):

print dirname

Crack\_data\_[dirname] = []

# print drpath

fdata = open(drpath + "/" + dirname + "\_Summary.txt", 'w')

fdata.write("Directory used: \n" + updir + dirname +

"\nFiles analysed: \n")

for filename in os.listdir(drpath):

if "\_sep\_log.csv" in filename:

fdata.write(filename + "\n")

file\_data = pd.read\_csv(drpath + "/" + filename)

principal\_angle = file\_data['principal\_angle']

Rad\_angle = np.deg2rad(principal\_angle)

Rad\_angle2 = (np.deg2rad(principal\_angle))\*2

circ\_mean = np.rad2deg(stats.circmean(Rad\_angle,

high=np.pi, low=0))

circ\_std = np.rad2deg(stats.circstd(Rad\_angle,

high=np.pi, low=0))

RVL = pycircstat.resultant\_vector\_length(Rad\_angle2)

ani\_mean = np.mean(file\_data['anisotropy'])

ani\_std = np.std(file\_data['anisotropy'])

fdata.write("Circ\_Mean Angle to x (+-Std) --> " +

str(circ\_mean) + "+-" + str(circ\_std) +

" degrees\nResultant\_vector\_length " +

str(RVL) + "\nMean Anisotropy (+-Std) --> " +

str(ani\_mean) + "+-" + str(ani\_std)+ "\n\n")

Crack\_data\_[dirname].append(file\_data)

data = pd.concat(Crack\_data\_[dirname], ignore\_index=True)

principal\_angles = data['principal\_angle']

Rad\_angles = np.deg2rad(principal\_angles)

Rad\_angles2 = (np.deg2rad(principal\_angles))\*2

circ\_mean\_all = np.rad2deg(stats.circmean(Rad\_angles,

high=np.pi, low=0))

circ\_std\_all = np.rad2deg(stats.circstd(Rad\_angles,

high=np.pi, low=0))

RVL\_all = pycircstat.resultant\_vector\_length(Rad\_angles2)

ani\_mean\_all = np.mean(data['anisotropy'])

ani\_std\_all = np.std(data['anisotropy'])

pval, U, Uc = pycircstat.raospacing(Rad\_angles2)

print pval

print U

print Uc

if pval > 0.05:

print "The populations IS uniformly distributed\np-value is " \

+ str(pval)

fdata.write("Global analysis:\nCirc\_Mean Angle to x (+-Std) "

"--> " + str(circ\_mean\_all) + "+-" +

str(circ\_std\_all) +

" degrees\nResultant\_vector\_length " +

str(RVL\_all) +

"\nMean Anisotropy (+-Std) --> " +

str(ani\_mean\_all) + "+-" + str(ani\_std\_all) +

"\nThe populations IS uniformly distributed\n"

"p-value is " + str(pval) + "\n")

else:

print "The populations is NOT uniformly distributed\np-value" \

"is " + str(pval)

fdata.write("Global analysis:\nCirc\_Mean Angle to x (+-Std) "

"--> " + str(circ\_mean\_all) + "+-" +

str(circ\_std\_all) +

" degrees\nResultant\_vector\_length " +

str(RVL\_all) +

"\nMean Anisotropy (+-Std) --> " +

str(ani\_mean\_all) + "+-" + str(ani\_std\_all) +

"\nThe populations IS NOT uniformly distributed\n"

"p-value is " + str(pval) + "\n")

fdata.close()

figure = plt.figure(0)

figure.clf()

figure.patch.set\_facecolor('w')

ax = figure.add\_subplot(111, polar=True)

colormap = 'plasma'

n\_bins = 36

# weights = data['anisotropy']\*data['area']

# weights = data['anisotropy']

# weights = data['area']

# weights = np.ones\_like(data['area'])

# weights = np.array([np.max(np.abs(v)) for v in data['eigen\_values']])

# weights = np.array([np.max(s) for s in np.array(data['standard\_deviations'])])

weights = data['standard\_deviations0']

for offset in [0, np.pi]:

histo, bins, patches = figure.gca().hist(offset+data['principal\_angle']\*np.pi/180.,

bins=offset+np.linspace(0, 180, n\_bins/2 + 1)\*np.pi/180.,

ec='k', weights=weights)

norm = colors.Normalize(0, histo.max())

for h, p in zip(histo, patches):

p.set\_facecolor(cm.get\_cmap(colormap)(norm(h)))

figure.gca().set\_yticks([])

figure.savefig(drpath + "/" + dirname + "\_cracks\_polarhist.jpg")

data.to\_csv(drpath + "/" + dirname + "\_data\_all.csv")

print "image saved"