function compositeBinary(RGB,FILENAME)

%COMPOSITEBINARY Thresholds and saves the largest object of an image to

%a new image file.

% COMPOSITEBINARY(RGB,FILENAME), where RGB is the input image. FILENAME

% is the desired output file name.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% RGB to 8-bit

I = rgb2gray(RGB);

% threshold

level = graythresh(I);

adj = 0.2;

BW = im2bw(I,level\*adj);

% morphological operations

BW = imfill(BW,'holes');

lineLength = 10;

se = strel('line',lineLength,0); BW = imclose(BW,se);

se = strel('line',lineLength,45); BW = imclose(BW,se);

se = strel('line',lineLength,90); BW = imclose(BW,se);

BW = imfill(BW,'holes');

% largest object

CC = bwconncomp(BW,8);

numPixels = cellfun(@numel,CC.PixelIdxList);

[~,idx] = max(numPixels);

mask = zeros(size(BW));

mask(CC.PixelIdxList{idx}) = 1;

figure; imshow(mask);

% save image to file

imwrite(mask,FILENAME,'png');

end

function migrationDistance(FULL\_PATH)

%MIGRATIONDISTANCE Calculates the average radial difference between two

%circular paths.

% MIGRATIONDISTANCE(FULL\_PATH), where FULL\_PATH is the directory

% containing input images, relative to the location of this M-file.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% import segmented images

tic; fprintf('Importing the images...');

EXECUTING\_PATH = pwd;

cd(FULL\_PATH);

currentDir = dir();

files = {currentDir.name};

index = regexp(files,'\.tif$'); % tif files only

files = files(~cellfun(@isempty,index));

stem = files{1}(1:end-7);

innerFile = fullfile(FULL\_PATH,[stem '-origin' '.png']);

outerFile = fullfile(FULL\_PATH,[stem '-silhouette.png']);

fprintf('done. [%0.3g s]\r',toc);

cd (EXECUTING\_PATH);

% boundaries

inner = outline(imread(innerFile));

outer = outline(imread(outerFile));

% origin centroid

s = regionprops(inner,'Centroid');

origin = cat(1,s.Centroid);

% distance

[r,c] = find(outer); % return white pixels coordinates

dist = sqrt((r-origin(1)).^2+(c-origin(2)).^2); % Euclidean distance of boundary pixel to origin

avgDist = mean(dist);

[r,c] = find(inner);

innerRadius = sqrt((r-origin(1)).^2+(c-origin(2)).^2);

innerRadius = mean(innerRadius);

diff = avgDist - innerRadius;

diff = diff\*3.683; % at 4x, pixels to um

% reconstruct circle

[x,y] = meshgrid(-(origin(1)-1):(size(outer,2)-origin(1)),-(origin(2)-1):(size(outer,1)-origin(2)));

circle = ((x.^2+y.^2)<=avgDist^2);

circle = im2uint8(circle);

% display

outerThick = outline(imread(outerFile),5);

innerThick = outline(imread(innerFile),5);

averageCircle = outline(circle,5);

imwrite(outerThick,fullfile(FULL\_PATH,[stem '-outlinedIntermediate' '.png']),'png');

% merge/overlay

idx = find(outerThick);

idx2 = find(innerThick);

idx3 = find(averageCircle);

combo = uint8(zeros(size(outerThick))); % blank mask

combo = combo(:); % vectorize

combo(idx) = 100; % outer

combo(idx2) = 180; % inner

combo(idx3) = 255; % average

combo = reshape(combo,size(outerThick,1),size(outerThick,2));

% re-construct image

colors = [0 0 0; 1 1 1; 1 1 0; 0 0 1]; % black, white, yellow, blue

h = figure; iptsetpref('ImshowBorder','tight'); imshow(combo,'colormap',colors); hold on;

plot(origin(:,1),origin(:,2),'ro');

title(sprintf('Average Migration Distance: %0.5g um',diff),'FontSize',20);

hold off;

export\_fig(h,fullfile(FULL\_PATH,[stem sprintf('-outlined-%0.5gum',diff) '.png']));

fprintf('\n\nAverage Migration Distance: %0.5g um\r\n\n',diff);

end

function createBins(FULL\_PATH)

%CREATEBINS Creates fixed-width bins to be used in subsequent cellular

%analyses.

% CREATEBINS(FULL\_PATH), where FULL\_PATH is the directory containing

% input images, relative to the location of this M-file.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

processing = false; % true if manual adjustment for image specified below

processingImage = 1;

% checkpoint controls

% check which images you'd like to see, only if processing

boolAdjust = false; boolAdjustReturn = false;

% intercellular gaps & rim

boolThresholding = true; boolThresholdingReturn = true;

% rings, nuclear orientation

boolLineAngle = true; boolLineAngleReturn = true;

% rings

boolRing0 = false; boolRing0Return = false;

% rings

boolRingOverlay = true; boolRingOverlayReturn = true;

% IMAGE FILE INFORMATION

t = tic; fprintf('Examining the folder contents...');

EXECUTING\_PATH = pwd;

cd(FULL\_PATH);

importFileList = dir();

filenames = {importFileList.name};

index = regexp(filenames,'\.tif$'); % get all .tif & .tiff files

filenames = filenames(~cellfun(@isempty,index));

files = fullfile(FULL\_PATH,filenames);

num = numel(files);

inputFolder = 'input-parameters';

inputFile1 = fullfile(inputFolder,'adjustments.txt');

inputFile2 = fullfile(inputFolder,'lineAngle.txt');

inputFile3 = fullfile(inputFolder,'skipImage.txt');

if ( ~exist(inputFile1,'file') || ~exist(inputFile2,'file') )

fprintf('!\r\r');

error('manualAdjust:noInput','Missing input files or improperly labeled.');

end

outputFolder1 = 'output-adjustedOriginals';

outputFolder2 = 'output-rings';

outputFolder3 = 'output-intermediates';

outputFolder4 = 'output-results';

if ( ~exist(outputFolder1,'dir') && ~processing )

mkdir(outputFolder1);

end

if ( ~exist(outputFolder2,'dir') && ~processing )

mkdir(outputFolder2);

end

if ( ~exist(outputFolder3,'dir') && ~processing )

mkdir(outputFolder3);

end

if ( ~exist(outputFolder4,'dir') && ~processing )

mkdir(outputFolder4);

elseif ( exist(outputFolder4,'dir') && ~processing )

cd(outputFolder4);

if ( exist('output-intercellularGaps-total.xlsx','file') )

delete('output-intercellularGaps-total.xlsx');

end

cd('..');

end

cd(EXECUTING\_PATH);

inputFile1 = fullfile(FULL\_PATH,inputFile1);

inputFile2 = fullfile(FULL\_PATH,inputFile2);

inputFile3 = fullfile(FULL\_PATH,inputFile3);

outputFolder1 = fullfile(FULL\_PATH,outputFolder1);

outputFolder2 = fullfile(FULL\_PATH,outputFolder2);

outputFolder3 = fullfile(FULL\_PATH,outputFolder3);

outputFile = fullfile(FULL\_PATH,inputFolder,'migrationDirections.txt'); % final migration directions

fprintf('complete. [%0.3g s]\r',toc(t));

% READ INPUT PARAMETERS

tic; fprintf('Reading contrast and thresholding adjustments...');

fid = fopen(inputFile1,'r');

A = fscanf(fid,'%g\t %g\t %g\t %g\t %g');

fclose(fid);

counter = 1;

numElements = 5;

for c = 1:numElements:size(A,1)-(numElements-1)

Bin(counter,1) = A(c); % n (image number)

Bin(counter,2) = A(c+1); % n (file number)

Bin(counter,3) = A(c+2); % LI

Bin(counter,4) = A(c+3); % HI

Bin(counter,5) = A(c+4); % fudge

counter = counter + 1; % move to next row in matrix

end

clearvars A fid counter;

fprintf('complete. [%0.3g s]\r',toc);

tic; fprintf('Reading line angle adjustments...');

fid = fopen(inputFile2,'r');

A = fscanf(fid,'%g\t %g\t %g');

fclose(fid);

counter = 1;

numElements = 3;

for c = 1:numElements:size(A,1)-(numElements-1)

Cin(counter,1) = A(c); % n (image number)

Cin(counter,2) = A(c+1); % n (file number)

Cin(counter,3) = A(c+2); % line angle adjustment

counter = counter + 1; % move to next row in matrix

end

clearvars A fid counter;

fprintf('complete. [%0.3g s]\r\r',toc);

% import skip images, if any

skipImage = [];

if ( exist(inputFile3,'file') )

fid = fopen(inputFile3,'r');

skipImage = fscanf(fid,'%i');

fclose(fid);

end

% covert skipped images to 3-channel system

skipImage = skipImage + 2\*(skipImage-1);

% ADJUST IMAGES

processingImage = processingImage + 2\*(processingImage-1);

if ( ~processing )

boolAdjust = false;

boolThresholding = false;

boolLineAngle = false;

boolRing0 = false;

boolRingOverlay = false;

end

nCount = 1;

for n = 1:3:num % for each image with three separate channels

if ( ( n == processingImage && ~ismember(n,skipImage) ) || ( ~processing && ~ismember(n,skipImage) ) )

% import & adjust image

t0 = tic;

green = imread(files{n});

red = imread(files{n+1});

blue = imread(files{n+2});

name = filenames{n}(1:end-4);

% get image size

IMAGE\_DIM = size(green);

if ( processing )

fprintf('IMAGE #%i:\r',processingImage);

else

fprintf('IMAGE #%i of %i:\r',nCount,num/3);

end

% adjust red

tic; fprintf('\tAdjusting red channel...');

red = imadjust(red,[Bin(n,3); Bin(n,4)],[],1);

red = medianFilter(red);

fprintf('complete. [%0.3g s]\r',toc);

% adjust green

tic; fprintf('\tAdjusting green channel...');

green = imadjust(green,[Bin(n+1,3); Bin(n+1,4)],[],1);

green = medianFilter(green);

fprintf('complete. [%0.3g s]\r',toc);

% adjust blue

tic; fprintf('\tAdjusting blue channel...');

blue = imadjust(blue,[Bin(n+2,3); Bin(n+2,4)],[],1);

blue = medianFilter(blue);

fprintf('complete. [%0.3g s]\r',toc);

% put back into RGB

clearvars origCombo;

origCombo = cat(3,red-blue,green,blue);

if ( boolAdjust )

figure('Name','Adjusted Image'); imshow(origCombo);

if ( boolAdjustReturn )

return;

end

end

% find cells vs. not cells to know direction of lines

tic; fprintf('\tThresholding & extracting rim...');

clearvars indivThresh cellMass totalMass rim;

indivThresh(:,:,1) = silhouette(origCombo(:,:,1),Bin(n,5));

indivThresh(:,:,2) = silhouette(origCombo(:,:,2),Bin(n+1,5));

indivThresh(:,:,3) = silhouette(origCombo(:,:,3),Bin(n+2,5));

indivThresh = im2uint8(indivThresh);

cellMass = indivThresh(:,:,1) + indivThresh(:,:,2) + indivThresh(:,:,3);

smallSize = 100000; % for a 1024x1024 image, 100K is 10% of the total

cellMass = bwareaopen(cellMass,smallSize);

totalMass = imfill(cellMass,'holes');

totalMass = bwareaopen(totalMass,smallSize);

totalMass = im2uint8(totalMass);

% create rim mask from actin

rim = outline(totalMass);

rim(1,:) = 0; rim(IMAGE\_DIM(1),:) = 0;

rim(:,1) = 0; rim(:,IMAGE\_DIM(2)) = 0;

rim = im2uint8(rim);

CC1 = bwconncomp(rim,8);

if ( boolThresholding )

% approxLine is white, ring0 is yellow

debug = im2uint8(origCombo);

debug(:,:,1) = debug(:,:,1) + rim;

debug(:,:,2) = debug(:,:,2);

debug(:,:,3) = debug(:,:,3);

figure('Name','red = rim'); imshow(debug);

figure('Name','totalMass'); imshow(totalMass);

clearvars debug;

if ( boolThresholdingReturn )

return;

end

end

% count number of cells in view based on # nuclei

numCells = bweuler(indivThresh(:,:,3),8);

% calculate intercellular area, normalizing to number of cells

area1 = bwarea(totalMass); % pixels

area2 = bwarea(cellMass); % pixels

area1 = area1\*1.5^2; % um^2 (10x = 1.5 um/pix; 20x = 0.73 um/pix)

area2 = area2\*1.5^2; % um^2 (10x = 1.5 um/pix; 20x = 0.73 um/pix)

intercellular = 100\*(area1-area2)/area1;

intercellularNormToNumCells = (area1-area2)/numCells; % um^2 of intercellular space per cell

fprintf('complete. [%0.3g s]\r',toc);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% get line approximation

tic; fprintf('\tApproximating migratory front line...');

CC = bwconncomp(rim,8);

s = regionprops(CC,'MajorAxisLength');

[Y,X] = find(rim); % X = column, Y = row (reverse order)

M = (Y(2)-Y(end-1))/(X(end-1)-X(2));

lineAngle = rad2deg(atan(M)); % always between [-90,90], in degrees

lineAngleAdjust = Cin(n-2\*(n-1)/3,3);

lineAngle = lineAngle + lineAngleAdjust; % manually adjust slope, in degrees

if ( lineAngle > 90 ) % keep line angle [-90,90]

lineAngle = lineAngle - 180;

elseif ( lineAngle < -90 ) % keep line angle [-90,90]

lineAngle = lineAngle + 180;

end

M = tan(deg2rad(lineAngle)); % convert angle back to slope

tempY = Y(end-1) - 2\*Y(end-1); % need to do this since MATLAB matrices have inverted the normal x-y coordinate map

B = tempY - M\*X(end-1);

B0 = B;

% make continuous line image

vertSlope = 15; horizSlope = 1/vertSlope;

if ( abs(M) >= vertSlope || abs(M) <= horizSlope ) % essentially veritcal or horizontal line

clear('B');

B = size(rim,1)/2;

B0 = round(B);

end

[approxLine,~,~] = imageLine(rim,M,B0);

if ( boolLineAngle )

debug = im2double(origCombo);

debug(:,:,1) = debug(:,:,1) + approxLine;

debug(:,:,2) = debug(:,:,2) + approxLine;

debug(:,:,3) = debug(:,:,3) + approxLine;

figure('Name','Line Angle'); imshow(debug);

clearvars debug;

if ( boolLineAngleReturn )

return;

end

end

fprintf('complete. [%0.3g s]\r',toc);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% determine angle of migration

tic; fprintf('\tCalculating migration direction...');

s = regionprops(approxLine,'Centroid');

centroids = cat(1,s.Centroid);

Mperp = -1/M; % perpendicular is negative reciprocal

tempY = centroids(1) - 2\*centroids(1); % need to do this since MATLAB matrices have inverted the normal x-y coordinate map

Bperp = tempY - Mperp\*centroids(2);

if ( abs(Mperp) >= vertSlope || abs(Mperp) <= horizSlope ) % essentially vertical or horizontal line

Bperp = size(rim,1)/2;

Bperp = round(Bperp);

end

[perpLine,~,~] = imageLine(rim,Mperp,Bperp);

if ( abs(Mperp) >= vertSlope || abs(Mperp) <= horizSlope ) % essentially vertical or horizontal line

perpLine = bwmorph(perpLine,'skel',Inf);

perpLine = im2double(perpLine);

end

perpAngle = lineAngle - 90;

if ( perpAngle > 90 ) % keep line angle [-90,90]

perpAngle = perpAngle - 180;

elseif ( perpAngle < -90 ) % keep line angle [-90,90]

perpAngle = perpAngle + 180;

end

[migrationDirection,~] = migDirection(perpLine,totalMass,perpAngle,Mperp);

fprintf('complete. [%0.3g s]\r',toc);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% adjust y-intercept, B, so that the line creats a valid ring

% first, move the line until it hits the tip of the rim, then subtract from there

tic; fprintf('\tFinding ring0:\r\t\t'); iter = 1;

tip = false; longEnough = false; symLim = 75;

while ( (~tip && ~longEnough) )

fprintf('\*');

if mod(iter,symLim) == 0

fprintf('%5.0i\r\t\t',iter);

end

% shift line to move TOWARDS migrationDirection

if ( abs(M) < vertSlope && abs(M) > horizSlope )

if ( migrationDirection >= 0 && migrationDirection <= 180 ) % Quadrants I & II, move line to the right by increasing B

B = B + 1;

elseif ( migrationDirection < 0 && migrationDirection >= -180 ) % Quadrants III & IV, move line to the left by decreasing B

B = B - 1;

end

end

if ( abs(M) >= vertSlope ) % vertical line

if ( migrationDirection > 90 || migrationDirection < -90 ) % moving left, so add lines to the left to find tip

B = B - 1;

elseif ( (migrationDirection>=0 && migrationDirection<90) || (migrationDirection<0 && migrationDirection>-90) ) % moving right, so add lines to the right to find tip

B = B + 1;

end

end

if ( abs(M) <= horizSlope ) % horizontal line

if ( migrationDirection == 90 ) % moving up, so add lines to the left to find tip

B = B - 1;

elseif ( migrationDirection == -90 ) % moving down, so add lines to the right to find tip

B = B + 1;

end

end

% make continuous line image

B = round(B);

[ring0,~,~] = imageLine(approxLine,M,B);

% check if at rim

ring0Vec = ring0(:);

lineIdx = find(ring0Vec==1); % indices of the line in the image

if sum(totalMass(lineIdx)>0) == 0 % if there are no white pixels from rim along the line, then we found the tip

tip = true;

end

iter = iter + 1;

if ( tip && iter<10 ) % premature tip since line started in black pixels, so jump back and start over

if ( migrationDirection >= 0 && migrationDirection <= 180 )

B = B - 50;

elseif ( migrationDirection < 0 && migrationDirection >= -180 )

B = B + 50;

end

tip = false; iter = 1;

elseif ( tip && iter>10 )

longEnough = true; % legit tip

end

end

if ( boolRing0 )

% approxLine is white, ring0 is yellow

debug = im2uint8(origCombo);

debug(:,:,1) = debug(:,:,1) + im2uint8(approxLine) + im2uint8(ring0) + rim;

debug(:,:,2) = debug(:,:,2) + im2uint8(approxLine) + im2uint8(ring0);

debug(:,:,3) = debug(:,:,3) + im2uint8(approxLine);

figure('Name','white = approxLine, yellow = ring0, red = rim'); imshow(debug);

clearvars debug;

if ( boolRing0Return )

return;

end

end

fprintf('%5.0i\r',iter);

fprintf('\t\t...complete. [%0.3g s]\r',toc);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% add parallel lines with same slope, but just change y-intercept, B

tic; fprintf('\tConstructing ring mask...');

% 100 pixels at 20x (0.73 um/pixel) is 200um (274 originally)

% 68 pixels at 10x (1.4691 um/pix) is 100um

numRings = 20; deltaD = 68;

if ( abs(M) >= vertSlope || abs(M) <= horizSlope ) % essentially zero from a vertical line, angle 88-90

deltaB = deltaD;

else

deltaB = deltaD/cos(deg2rad(lineAngle)); % calculation for y-intercept change given distance between parallel lines

end

ringMask = zeros(size(ring0));

for i=1:numRings-1 % subtract one to account for ring0

% adjust B to move line AWAY from migrationDirection

if ( abs(M) < vertSlope && abs(M) > horizSlope )

if ( M >= 0 && migrationDirection >= 0 && migrationDirection <= 180 ) % Quadrants I & II, move line to the right by increasing B

B = B - deltaB;

elseif ( M >= 0 && migrationDirection < 0 && migrationDirection >= -180 ) % Quadrants III & IV, move line to the left by decreasing B

B = B + deltaB;

elseif ( M < 0 && migrationDirection >= 0 && migrationDirection <= 180 ) % Quadrants I & II, move line to the right by decreasing B

B = B - deltaB;

elseif ( M < 0 && migrationDirection < 0 && migrationDirection >= -180 ) % Quadrants III & IV, move line to the left by increasing B

B = B + deltaB;

end

end

if ( abs(M) >= vertSlope ) % vertical

if ( migrationDirection > 90 || migrationDirection < -90 ) % moving left, so add lines to the right

B = B + deltaB;

elseif ( (migrationDirection>=0 && migrationDirection<90) || (migrationDirection<0 && migrationDirection>-90) ) % moving right, so add lines to the left

B = B - deltaB;

end

end

if ( abs(M) <= horizSlope ) % horizontal

if ( migrationDirection == 90 ) % moving up, so add lines to the bottom

B = B + deltaB;

elseif ( migrationDirection == -90 ) % moving down, so add lines to the top

B = B - deltaB;

end

end

% make sure it is not off the image if horiz or vert

if ( (abs(M) >= vertSlope && B > IMAGE\_DIM(1)) || ...

(abs(M) <= horizSlope && B > IMAGE\_DIM(1)) )

B = IMAGE\_DIM(1);

end

if ( (abs(M) >= vertSlope && B < 1) || ...

(abs(M) <= horizSlope && B < 1) )

B = 1;

end

% make new line

[rings,~,~] = imageLine(ring0,M,B);

% combine with others

ringMask = ringMask + rings;

end

% overlay rings onto original image, with ring0 being yellow

ringOverlay = im2double(origCombo);

ringOverlay(:,:,1) = ringOverlay(:,:,1) + ringMask + ring0;

ringOverlay(:,:,2) = ringOverlay(:,:,2) + ringMask + ring0;

ringOverlay(:,:,3) = ringOverlay(:,:,3) + ringMask;

ringOverlay = im2uint8(ringOverlay);

% adjust ring mask so ring0 can be identified

% ring0 is red, other rings are blue

clearvars ringMaskFinal;

ringMaskFinal(:,:,1) = ring0;

ringMaskFinal(:,:,2) = zeros(size(ring0));

ringMaskFinal(:,:,3) = ringMask;

if ( boolRingOverlay )

figure('Name','Ring Overlay'); imshow(ringOverlay);

if ( boolRingOverlayReturn )

return;

end

end

fprintf('complete. [%0.3g s]\r',toc);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% write images to file once ready to do so

if ( ~processing )

tic; fprintf('\tSaving images...');

imwrite(origCombo, fullfile(outputFolder1,[name sprintf('-output-adjustedOriginal') '.bmp']), 'bmp');

imwrite(ringOverlay, fullfile(outputFolder2,[name sprintf('-output-rings') '.bmp']), 'bmp'); % of tif

imwrite(cellMass, fullfile(outputFolder3,[name sprintf('-output-intermed1-thresholded') '.bmp']), 'bmp');

imwrite(ringMaskFinal, fullfile(outputFolder3,[name sprintf('-output-intermed2-ringMask') '.bmp']), 'bmp');

fprintf('complete. [%0.3g s]\r',toc);

end

if ( processing )

fprintf('IMAGE #%i complete: %0.3g s\r\r',processingImage,toc(t0));

else

fprintf('IMAGE #%i of %i complete: %0.3g s\r\r',nCount,num/3,toc(t0));

end

migrationDirectionAll(nCount) = migrationDirection;

slopeAll(nCount) = M;

nameAll{nCount} = name;

nCount = nCount + 1;

else

fprintf('Skipped image due to bad quality.\r\r');

nCount = nCount + 1;

end

end

% save migration directions to file for other analyses

if ( ~processing )

tic; fprintf('Writing migration directions to output file...');

for i = 1:numel(migrationDirectionAll);

A(i,1) = i; % image number

A(i,2) = migrationDirectionAll(i);

A(i,3) = slopeAll(i);

end

fid = fopen(outputFile,'w');

for i = 1:size(A,1)

fprintf(fid,'%i\t%.2f\t%.4f',A(i,:)); % 2 or 4 numerals on right of number

fprintf(fid,'\r\n');

end

fclose(fid);

fprintf('complete. [%0.3g s]\r',toc);

end

min = floor(toc(t)/60); sec = mod(toc(t),60);

fprintf('\rThis entire operation took %0.3g min., %0.3g s\r',min,sec);

end

function [imLine,r,c] = imageLine(IMAGE,M,B)

%IMAGELINE Returns the image and pixel coordinates of a line drawn

%according to the input slope and y-intercept.

% IMAGELINE(IMAGE,M,B), where IMAGE is a 2-dimensional matrix used to

% determine the dimensions of the output image. M and B are the slope and

% y-intercept, respectively, of the line to be drawn.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

total1 = size(IMAGE,1); % 1024 usually

total2 = size(IMAGE,2); % usually the same as total1

vertM = 15; horizM = 1/vertM;

if ( abs(M) >= vertM ) % essentially veritcal line

r = repmat(1:total1,2); % create column vector (x-axis) with repeats for creating a continuous line of thickness 2

c = ones(size(r));

c(1,:) = B; % here, B is just how much to move the line either left or right

c(2,:) = B+1; % make it thicker to see

elseif ( abs(M) <= horizM ) % horizontal line

c = repmat(1:total2,2); % create column vector (x-axis) with repeats for creating a continuous line of thickness 2

r = ones(size(c));

r(1,:) = B; % here, B is just how much to move the line either up or down

r(2,:) = B+1; % make it thicker to see, BUT THIS SCREWS UP perpLine (fixed when you return the IMAGE from this function)

else

if ( abs(M) >= 1 ) % add repeats in multiple rows of the same column

repeats = ceil(abs(M));

toggle = false;

else

repeats = ceil(abs(1/M));

toggle = true;

end

end

% construct disjoint line

rep = 1:repeats;

c = repmat(1:total2,[repeats 1]); % create column vector (x-axis) with repeats for creating a continuous line

c = c(:)';

r = M\*c + B;

r = r-2\*r; % need to do this since MATLAB matrices have inverted the normal x-y coordinate map

r = round(r); % disjoint points on line

% 8-connect line

% for each point, add top and bottom pixels until they are 8-connected to another pixel

if ( ~toggle )

for j = 1:repeats:numel(r)-repeats % iterate through each disjoint point on line

r(j+1:j+repeats) = r(j+1:j+repeats) + rep; % add more pixels to hopefully make it at least 8-conn

end

else

for j = 1:repeats:numel(c)-repeats % iterate through each disjoint point on line

c(j+1:j+repeats) = c(j+1:j+repeats) + rep;

end

end

% remove out-of-bounds values

negIdx = find(r<=0|r>total1|c<=0|c>total2);

c(negIdx) = []; r(negIdx) = [];

% order list if not vertical or horizontal

if ( abs(M) < vertM && abs(M) > horizM )

[c,sIdx] = sort(c,'ascend');

r = r(sIdx);

end

% convert to image

idx = sub2ind(size(IMAGE),r,c);

imLine = zeros(size(IMAGE));

imLine(idx) = 1;

% remove little pieces

cutoff = 10;

imLine = bwareaopen(imLine,cutoff);

[r,c] = find(imLine==1);

% make sure line touches two edges

rEdges = numel(r(r==1|r==total1));

cEdges = numel(c(c==1|c==total1));

if ( (rEdges+cEdges) < 2 )

% if first or last index are close to an edge, but not on an edge, then...

% but list is unordered, and have to check both ends near either type of edge

if ( numel(find(r==1)) < 1 && ~isempty(r(r==2)) ) % not touching edge, but should be!

idx = find(r==2);

imLine(1,c(idx(1))) = 1;

end

if ( numel(find(r==total1)) < 1 && ~isempty(r(r==(total1-1))) ) % not touching edge, but should be!

idx = find(r==(total1-1));

imLine(total1,c(idx(1))) = 1;

end

if ( numel(find(c==1)) < 1 && ~isempty(c(c==2)) ) % not touching edge, but should be!

idx = find(c==2);

imLine(r(idx(1)),1) = 1;

end

if ( numel(find(c==total1)) < 1 && ~isempty(c(c==(total1-1))) ) % not touching edge, but should be!

idx = find(c==(total1-1));

imLine(r(idx(1)),total1) = 1;

end

[r,c] = find(imLine==1);

end

% make sure final output is a double

imLine = im2double(imLine);

end

function [migDir,migAng] = migDirection(PERP\_IM,TOTAL,PERP\_ANG,PERP\_SLOPE)

%MIGDIRECTION Measures and returns the migration direction of migrating

%cells based on the approximate migratory front line.

% MIGDIRECTION(PERP\_IM,TOTAL,PERP\_ANG,PERP\_SLOPE), where PERP\_IM is the

% image of the approximate migratory front line image, PERP\_ANG is the

% angle of that line, and PERP\_SLOPE is the slope of that line.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

line = TOTAL(PERP\_IM==1);

% determine directionality of trace, only necessary if vertical

topToBottom = false;

if ( abs(PERP\_SLOPE) >= 15 )

topToBottom = true;

end

% migDir always between [-180,180]

% migAng always between [-90,90]

migAng = PERP\_ANG;

migDir = migAng;

left = sum(line(1:floor(numel(line)/2)));

right = sum(line(ceil(numel(line)/2):end));

if ( abs(PERP\_SLOPE) >= 15 ) % vertical line

top = left;

bottom = right;

if ( topToBottom ) % goes from top to bottom

if ( top < bottom ) % top < bottom, so moving up

migAng = 90;

migDir = 90;

else % top > bottom, so moving down

migAng = -90;

migDir = -90;

end

else % goes from bottom to top

top = right;

bottom = left;

if ( top < bottom ) % top < bottom, so moving up

migAng = 90;

migDir = 90;

else % top > bottom, so moving down

migAng = -90;

migDir = -90;

end

end

else % non-vertical line

if ( migDir > 0 )

if ( left < right )

migDir = migDir - 180;

end

else

if ( left < right )

migDir = migDir + 180;

end

end

end

end

function nuclearMeasures(FULL\_PATH)

%NUCLEARMEASURES Analyzes nuclei and outputs data about their

%orientation as well as intercellular distribution of beta-catenin.

% NUCLEARMEASURES(FULL\_PATH), where FULL\_PATH is the directory containing

% input images, relative to the location of this M-file.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% input & output paths

inputFolder1 = fullfile(FULL\_PATH,'input-parameters');

inputFolder2 = fullfile(FULL\_PATH,'output-adjustedOriginals');

inputFolder3 = fullfile(FULL\_PATH,'output-intermediates');

outputFolder1 = fullfile(FULL\_PATH,'output-intermediates');

outputFolder2 = fullfile(FULL\_PATH,'output-results');

% import ring masks and thresholds

t = tic; fprintf('Examining the folder contents...\r');

EXECUTING\_PATH = pwd;

cd(inputFolder3);

importFileList = dir();

filenames = {importFileList.name};

index = regexp(filenames,'output-intermed2-ringMask\.bmp$'); % get only the ringMask files

filenames = filenames(~cellfun(@isempty,index));

filesRing = fullfile(inputFolder3,filenames);

num = numel(filesRing);

filenames = {importFileList.name};

index = regexp(filenames,'output-intermed1-thresholded\.bmp$');

filenames = filenames(~cellfun(@isempty,index));

filesCellMass = fullfile(inputFolder3,filenames);

cd(EXECUTING\_PATH);

% import nuclei from adjusted images

cd(inputFolder2);

importFileList = dir();

filenames = {importFileList.name};

index = regexp(filenames,'output-adjustedOriginal\.bmp$');

filenames = filenames(~cellfun(@isempty,index));

filesAdjusted = fullfile(inputFolder2,filenames);

cd(EXECUTING\_PATH);

% import migration directions

cd(inputFolder1);

inputFile = 'migrationDirections.txt';

fid = fopen(inputFile,'r');

migDir = fscanf(fid,'%g\t%g\t%g');

slope = migDir;

migDir(1:3:end) = []; % remove every other value to keep only the migration directions

migDir(2:2:end) = [];

slope(1:3:end) = [];

slope(1:2:end) = [];

fclose(fid);

% import skip images, if any

inputFile = 'skipImage.txt';

skipImage = [];

if ( exist(inputFile,'file') )

fid = fopen(inputFile,'r');

skipImage = fscanf(fid,'%i');

fclose(fid);

end

cd(EXECUTING\_PATH);

% make results folder

if ( ~processing )

if ( ~exist(outputFolder2,'dir') && ~processing )

mkdir(outputFolder2);

end

ExcelFile = fullfile(outputFolder2,['output-' importDir '.xlsx']);

cd(EXECUTING\_PATH);

end

cd(EXECUTING\_PATH);

fprintf('complete. [%0.3g s]\r',toc(t));

% image processing

nCount = 1; % actual number of images processed, excluding skips (for Excel writing mostly)

for n=1:num % for each image

if ( ( n == processingImage && ~ismember(n,skipImage) ) || ( ~processing && ~ismember(n,skipImage) ) ) % debugging only, eventually remove!

tic; fprintf('\rIMAGE #%i of %i...\r',n,num);

name{nCount} = filenames{n}(1:end-28);

% read and parse image channels

clear('image','red','green','blue');

image = imread(filesAdjusted{n});

red = image(:,:,1);

green = image(:,:,2);

blue = image(:,:,3);

% get image size

IMAGE\_DIM = size(image(:,:,1));

% import total mass to make sure nuclei aren't outside rim

tic; fprintf('\tConstructing nuclei band masks...');

clear('cellMass','totalMass');

cellMass = imread(filesCellMass{n});

totalMass = imfill(cellMass,'holes');

totalMass = im2uint8(totalMass);

% create rim mask from actin

clear('rim');

rim = outline(totalMass); % assume already properly thresholded

% remove back end to just keep the line

rim(1,:) = 0; rim(IMAGE\_DIM(1),:) = 0;

rim(:,1) = 0; rim(:,IMAGE\_DIM(2)) = 0;

limit = IMAGE\_DIM(1)-1;

rim = bwareaopen(rim,limit);

rim = im2double(rim);

% remove large and small blobs

blue = im2bw(blue,graythresh(blue));

LB = 5;

UB = 1000;

blue = xor(bwareaopen(blue,LB,4),bwareaopen(blue,UB,4));

blue = imclearborder(blue);

% remove nuclear outside of cell mass

blue(~totalMass) = 0;

% nuclear beta-catenin (NBC)

clear('nbc','nnbc');

nbc0 = gaussFilter(green);

nbc = nbc0;

nbc(find(~blue)) = 0; % select just nuclear pixels from green by erasing non-nuclear pixels

nnbc = nbc0;

nnbc(~totalMass) = 0;

nnbc(blue) = 0;

% read ring masks

clear('mask','ring0','ringRest');

mask = imread(filesRing{n});

ring0 = mask(:,:,1); % red line is ring0

ring0 = im2bw(ring0,graythresh(ring0));

ringRest = mask(:,:,3); % blue lines are other rings

ringRest = im2bw(ringRest,graythresh(ringRest));

ring0orig = ring0;

ringRestOrig = ringRest;

% add constant line features

clearvars combo I;

combo(:,:,1) = rim + ring0 + ringRest;

combo(:,:,2) = ring0 + ringRest;

combo(:,:,3) = ringRest;

I(:,:,1) = rim + ring0 + ringRest;

I(:,:,2) = ring0 + ringRest;

I(:,:,3) = ringRest;

I = im2double(I);

nucleiForLabeling = zeros(size(I(:,:,1)));

nucleiForLabeling = im2double(nucleiForLabeling);

fprintf('complete. [%0.3g s]\r',toc);

fprintf('Creating bins...');

% find centroid of ring0

s = regionprops(ring0,'Centroid');

centroids = cat(1,s.Centroid);

Mperp = -1/slope(n); % perpendicular is negative reciprocal

tempY = centroids(1) - 2\*centroids(1); % need to do this since MATLAB matrices have inverted the normal x-y coordinate map

Bperp = tempY - Mperp\*centroids(2);

% draw perpendicular line to new image

% if horizontal, make everything 1 pixel thick

vertSlope = 15; horizSlope = 1/vertSlope;

horiz = false;

if ( abs(Mperp) >= vertSlope || abs(Mperp) <= horizSlope ) % essentially veritcal line

Bperp = size(rim,1)/2;

horiz = true;

ring0 = bwmorph(ring0,'skel',Inf);

ringRest = bwmorph(ringRest,'skel',Inf);

end

if ( ~horiz )

% determine number of lines that should be crossed

CCLines = bwconncomp(ring0+ringRest,8);

numLines = CCLines.NumObjects;

adjusted = false;

Bperp = round(Bperp);

BperpOptions = Bperp-2000:50:Bperp+2000;

best(1) = 0; best(2) = 0;

for i = 1:numel(BperpOptions)

[perpLine,~,~] = imageLine(ring0,Mperp,BperpOptions(i));

% does it intersect ring0?

idx0 = find(perpLine==1 & ring0==1);

% find neighboring line

% do this by finding the closest intersection on perpLine that hasn't already been seen

[r,c] = find(perpLine==1&(ringRest==1|ring0==1));

X = cat(2,r,c);

% find Euclidean distance between each intersetion

distances = pdist(X,'euclidean')';

if ( numel(distances) > 5 )

% remove redundant points

cutoff = 10;

for j = 1:size(X,1)

if ( distances(j) < cutoff )

X(j,1) = -50;

end

end

for j = 2:size(X,1)

if ( abs(distances(j-1)-distances(j)) < cutoff ) % pixel is so close and coordinate should be ignored

X(j,1) = -50;

end

end

idx = X(:,1)<0;

X(idx,:) = [];

end

% save if intersects the most number of lines AND also intersects ring0

if ( numel(X) > best(1) && ~isempty(idx0) )

best(1) = numel(X);

best(2) = BperpOptions(i);

end

end

% final perpLine choice

[perpLine,~,~] = imageLine(ring0,Mperp,best(2));

else % horizontal line (skip earlier stuff to save time)

Bperp = round(Bperp);

[perpLine,~,~] = imageLine(ring0,Mperp,Bperp);

if ( abs(Mperp) >= vertSlope )

perpLine(:,IMAGE\_DIM(1)/2+1) = 0; % make it just one line, not a thick one

end

if ( abs(Mperp) <= horizSlope )

perpLine(round(IMAGE\_DIM(1)/2+1),:) = 0; % make it just one line, not a thick one

end

end

s1 = regionprops(perpLine,'PixelList');

% find neighboring line

% do this by finding the closest intersection on perpLine that hasn't already been seen

[r,c] = find(perpLine==1&(ringRest==1|ring0==1));

X = cat(2,r,c);

% find Euclidean distance between each intersetion

distances = pdist(X,'euclidean')';

% remove redundant points

cutoff = 10;

for j = 1:size(X,1)

if ( distances(j) < cutoff )

X(j,1) = -50;

end

end

for j = 2:size(X,1)

if ( abs(distances(j-1)-distances(j)) < cutoff ) % pixel is so close and coordinate should be ignored

X(j,1) = -50;

end

end

idx = X(:,1)<0;

X(idx,:) = [];

% determine where ring0 is

[r,c] = find(perpLine==1&ring0==1);

X0 = cat(2,r,c);

X0 = intersect(X,X0,'rows'); % selects just one pixel that we found earlier when removing redundancies

% if horizontal, make rings thick again (2 pixels)

ring0 = ring0orig;

ringRest = ringRestOrig;

% find intersection pixel closest to this one (should be about 68 pixels away for 10x and 100um bins

clear('distances');

temp = pdist(X,'euclidean')';

% reconstruct distances into 2D array

% each column is one pixel with the distances to all over pixels

% the closest pixel is the one that is around 68 (for 10x and 100um bins)

distances = diag(-ones(size(X,1),1));

startPoint = 1;

endPoint = size(distances,2)-2;

for i = 1:size(distances,2) % go through columns

distances(i+1:size(distances,1),i) = temp(startPoint:startPoint+endPoint);

startPoint = startPoint + endPoint + 1;

endPoint = endPoint - 1;

end

% determine where to start

clear('coord1A');

coord1A = X0;

idx1 = find(ismember(X,coord1A,'rows'));

if ( idx1 == 1 ) % then look down columns for next line

colLook = true;

else % then look down rows for next line

colLook = false;

end

fprintf('complete. [%0.3g s]\r',toc);

clear('s','CC');

CC = bwconncomp(ringRest+ring0); % all rings

s = regionprops(CC,'PixelList'); % unordered list of objects

bandsFinal = zeros(IMAGE\_DIM);

bandsFinal = im2uint8(bandsFinal);

for i = 1:size(X,1)-1 % number of perpLine intersections (may be less than number of bin lines)

tic; fprintf('Measurements for each bin...');

% find closest intersection

clear('idx1','idx2','temp');

idx1 = find(ismember(X,coord1A,'rows'));

if ( colLook )

temp = distances(:,idx1);

temp(temp<=0) = IMAGE\_DIM(1)\*2;

idx2 = find(temp==min(temp));

else

temp = distances(idx1,:);

temp(temp<=0) = IMAGE\_DIM(1)\*2;

idx2 = find(temp==min(temp));

end

coord2A = X(idx2,:);

% find corresponding line

clear('lineNum1','lineNum2');

coord1B = [coord1A(2) coord1A(1)];

coord2B = [coord2A(2) coord2A(1)];

found1 = false; found2 = false;

for j = 1:numel(s)

if ( ismember(coord1B,s(j).PixelList,'rows') )

lineNum1 = j;

found1 = true;

end

if ( ismember(coord2B,s(j).PixelList,'rows') )

lineNum2 = j;

found2 = true;

end

if ( found1 && found2 )

break;

end

end

% reconstruct line into image

line1 = zeros(IMAGE\_DIM);

line2 = zeros(IMAGE\_DIM);

% replace pixels defined in coordinate array

line1(sub2ind(IMAGE\_DIM,s(lineNum1).PixelList(:,2),s(lineNum1).PixelList(:,1))) = 1;

line2(sub2ind(IMAGE\_DIM,s(lineNum2).PixelList(:,2),s(lineNum2).PixelList(:,1))) = 1;

% find a point between the lines

clear('idx','inside','x1','x2');

inside = s1.PixelList;

x1 = find(ismember(inside,coord1B,'rows'));

x2 = find(ismember(inside,coord2B,'rows'));

inside(1:min(x1,x2),:) = -50;

inside(max(x1,x2):end,:) = -50;

idx = inside(:,1)<0; % efficient logical indexing

inside(idx,:) = [];

% find starting points so imfill will color in background

clear('midCoord');

midCoord = inside(round(size(inside,1)/2),:);

% isolate band

clear('band','bandFocus');

band = logical(line1+line2);

bandFocus = imfill(band,[midCoord(2) midCoord(1)],4); % backwards coordinates!!!!

bandFocus = im2uint8(bandFocus);

bandFocus = bandFocus/255;

% adjust color & add to cumulative image

bandsFinal = bandsFinal + bandFocus\*(255-15\*i); % create any number of bands

% take measurements for each band

clear('idx','CCBand','sBand');

tic; fprintf('\tExtracting ring info...');

CCBand = bwconncomp(bandFocus,8);

sBand = regionprops(CCBand,'PixelIdxList');

% isolate beta-catenin in band

clear('nbcBand','nnbcBand');

nbcBand = nbc;

nbcBand(bandFocus==0) = 0;

nnbcBand = nnbc;

nnbcBand(bandFocus==0) = 0;

% isolate nuclei in band

clear('nBand','sNuclear');

CCBlue = bwconncomp(blue,8);

sNuclear = regionprops(CCBlue,'PixelIdxList');

nBand = zeros(size(blue));

nBand = nBand(:);

for j = 1:numel(sNuclear) % for each nucleus

if ( ismember(sNuclear(j).PixelIdxList,sBand.PixelIdxList) )

nBand(sNuclear(j).PixelIdxList) = 1;

end

end

nBand = reshape(nBand,IMAGE\_DIM(1),IMAGE\_DIM(2));

% find cell area within band, important if only part of bin has cells

clear('cellBand','s0','a');

cellBand = cellMass;

cellBand(bandFocus==0) = 0;

s0 = regionprops(cellBand,'Area');

[a,~] = max(cat(1,s0.Area));

% set up and take measurements

cell = ['B' num2str(nCount+6)];

storedCategories = {'Band','# Nuclei', ...

'Band Area (um^2)', ...

'Cell Density (cells per um^2)', ...

'Nuclear Beta-Catenin Intensity', ...

'Nuclear Beta-Catenin Intensity Normalized to # Cells', ...

'Non-Nuclear Beta-Catenin Intensity', ...

'Non-Nuclear Beta-Catenin Intensity Normalized to # Cells', ...

'Nuclear:Non-Nuclear Beta-Catenin Ratio', ...

'Non-Nuclear:Nuclear Beta-Catenin Ratio'};

stored(i,1,nCount) = i;

stored(i,2,nCount) = bweuler(nBand,8); % # cells

stored(i,3,nCount) = a\*1.4691^2; % area in um^2 (0.73^2 = 0.5329, 1.4691^2 = 2.1583)

stored(i,4,nCount) = stored(i,2,nCount)/stored(i,3,nCount); % cell density (cells per um^2)

stored(i,5,nCount) = sum(sum(nbcBand)); % nbc intensity

stored(i,6,nCount) = stored(i,5,nCount)/stored(i,2,nCount); % nbc intensity normalized to cell number

stored(i,7,nCount) = sum(sum(nnbcBand)); % nnbc intensity

stored(i,8,nCount) = stored(i,7,nCount)/stored(i,2,nCount); % nnbc intensity normalized to cell number

stored(i,9,nCount) = stored(i,5,nCount)/stored(i,7,nCount); % nbc:nnbc ratio

stored(i,10,nCount) = stored(i,7,nCount)/stored(i,5,nCount); % nnbc:nbc ratio

% combine sections with variable elements

if ( mod(i,3) == 1 ) % blue

combo(:,:,3) = combo(:,:,3) + nBand;

elseif ( mod(i,3) == 2 ) % cyan

combo(:,:,2) = combo(:,:,2) + nBand;

combo(:,:,3) = combo(:,:,3) + nBand;

elseif ( mod(i,3) == 0 ) % green

combo(:,:,2) = combo(:,:,2) + nBand;

end

% add nuclei to cumulative image

I(:,:,3) = I(:,:,3) + nBand;

nucleiForLabeling = nucleiForLabeling + nBand;

% establish an order for going through the nuclei

clear('L','s3','extrema','left\_most\_top','sort\_order','s2');

L = bwlabel(nBand')';

s3 = regionprops(L,'BoundingBox','Extrema','Centroid','Orientation','MajorAxisLength');

extrema = cat(1, s3.Extrema);

left\_most\_top = extrema(1:8:end, :);

[~, sort\_order] = sortrows(fliplr(left\_most\_top));

s2 = s3(sort\_order);

% calculate nuclear orientation

clear('angles','normAngles');

storedAnglesCatetories = {'Normalized Angles [-90,90]'};

angles = cat(1,s2.Orientation);

if ( isempty(angles) ) % no nuclei in this bin

normAngles = 0;

bins = 1;

storeAngles(1,i,n) = -1;

else % normal steps if there is at least one nucleus in bin

if ( migDir(n) > 0 )

normAngles = angles - abs(migDir(n)); % shift angles

normAngles(normAngles<-90) = normAngles(normAngles<-90)+180;

else

normAngles = angles + abs(migDir(n)); % shift angles

normAngles(normAngles>90) = normAngles(normAngles>90)-180;

end

storeAngles(1:numel(normAngles),i,n) = abs(normAngles);

bins = 40; % default is 20

end

% draw rose plot (radial histogram)

clear('roseAngles');

roseAngles = normAngles;

roseAngles(roseAngles<0) = roseAngles(roseAngles<0)+360;

roseAngles = deg2rad(roseAngles);

h = figure; set(h,'color','w');

g = rose(roseAngles,bins); hold on;

x = get(g,'XData');

y = get(g,'YData');

p = patch(x,y,'r'); % change color

axis tight;

set(gcf,'Position',[600 300 225 410]);

hold off;

if ( ~processing )

saveas(h,fullfile(outputFolder2,[name{nCount} '-output-results2-' num2str(i) '-nuclear\_orientation' '.bmp']),'bmp');

close(h);

end

% move on to next band

coord1A = coord2A;

fprintf('complete. [%0.3g s]\r',toc);

end

tic; fprintf('Saving images...');

% make inter-bin lines black by subtracting them out

bandsFinal = bandsFinal - im2uint8(ringRest+ring0)\*255;

% save images

imwrite(bandsFinal,fullfile(outputFolder1,[name{nCount} '-output-intermed3-bands' '.bmp']),'bmp');

imwrite(combo,fullfile(outputFolder1,[name{nCount} '-output-intermed4-separated\_nuclei' '.bmp']),'bmp');

imwrite(nbc,fullfile(outputFolder1,[name{nCount} '-output-intermed6-nbc' '.bmp']),'bmp');

imwrite(nnbc,fullfile(outputFolder1,[name{nCount} '-output-intermed7-nnbc' '.bmp']),'bmp');

% establish an order for going through the nuclei

clear('L','s3','extrema','left\_most\_top','sort\_order','s2','imgTemp');

L = bwlabel(nucleiForLabeling')';

s3 = regionprops(L,'BoundingBox','Extrema','Centroid','Orientation','MajorAxisLength');

extrema = cat(1, s3.Extrema);

left\_most\_top = extrema(1:8:end, :);

[~, sort\_order] = sortrows(fliplr(left\_most\_top));

s2 = s3(sort\_order);

% label each nuclei with orientation line

h = figure; iptsetpref('ImshowBorder','tight');

I = im2uint8(I);

imshow(I); hold on;

% draw lines over nuclei

angles = cat(1,s2.Orientation);

mag = cat(1,s2.MajorAxisLength);

theta = angles;

centroids = cat(1,s2.Centroid);

hypo = mag;

opp = abs(sin(degtorad(theta)).\*hypo)/2;

adj = abs(cos(degtorad(theta)).\*hypo)/2;

for j = 1:size(angles,1) % for each nucleus

if ( angles(j) > 0 && angles(j) <= 90 )

c1(j) = centroids(j,1) - adj(j); % x1

r1(j) = centroids(j,2) + opp(j); % y1

c2(j) = centroids(j,1) + adj(j); % x2

r2(j) = centroids(j,2) - opp(j); % y2

else

c1(j) = centroids(j,1) - adj(j); % x1

r1(j) = centroids(j,2) - opp(j); % y1

c2(j) = centroids(j,1) + adj(j); % x2

r2(j) = centroids(j,2) + opp(j); % y2

end

end

if ( size(angles,1) ~= 0 )

x1 = c1; x2 = c2; y1 = r1; y2 = r2;

presentableAngles = round(angles);

presentableX1 = round(x1); presentableY1 = round(y1);

presentableOpp = round(opp); presentableAdj = round(adj);

for k = 1:numel(s2)

centroid = s2(k).Centroid;

line( [x1(k) x2(k)], [y1(k) y2(k)], 'LineWidth',2, 'Color','w');

end

end

hold off;

if ( ~processing )

export\_fig(fullfile(outputFolder2,[name{nCount} '-output-results1-labeled' '.bmp']),h);

close(h);

end

fprintf('complete. [%0.3g s]\r',toc);

% next image

nCount = nCount + 1;

else

fprintf('\rSkipped image due to bad quality.\r');

nCount = nCount + 1;

% fixes Excel writing problem

if ( n == num )

nCount = nCount - 1;

end

end

fprintf('IMAGE #%i of %i COMPLETE: %0.3g s\r',n,num,toc);

end

if ( ~processing )

% Excel writing

tExcel = tic; fprintf('\rWriting to Excel...\r');

for j = 1:nCount-1 % for each image that was actually analyzed

tic; fprintf('\tSheet %i of %i...',j,nCount-1);

sheet = j; % one sheet per image for normalized angles

nums = 1:size(stored(:,:,1),1); % number of bins

xlswrite(ExcelFile,name(j),sheet,'A1');

xlswrite(ExcelFile,nums,sheet,'A2');

xlswrite(ExcelFile,storedAnglesCatetories,sheet,'B1');

col = ['A','B','C','D','E','F','G','H','I', ...

'J','K','L','M','N','O','P','Q','R','S', ...

'T','U','V','W','X','Y','Z'];

xlswrite(ExcelFile,storedCategories,sheet,'AA1');

xlswrite(ExcelFile,stored(:,:,1),sheet,'AA2');

for k = 1:size(storeAngles(:,:,1),2)

tempAngles = storeAngles(:,k,1);

tempAngles(tempAngles==0) = [];

cell = [col(k) num2str(6)];

if ( ~isempty(tempAngles) )

xlswrite(ExcelFile,tempAngles,sheet,cell);

else

errorAngles = -1;

xlswrite(ExcelFile,errorAngles,sheet,cell);

end

end

stored(:,:,1) = [];

storeAngles(:,:,1) = [];

fprintf('complete. [%0.3g s]\r',toc);

end

fprintf('Complete. [%0.3g s]\r',toc(tExcel));

end

tot = toc(tTotal); min = floor(tot/60); sec = rem(tot,60);

fprintf('\rThis entire operation took %0.3g min, %0.3g s\r',min,sec);

end

function [mask] = outline(RGB,THICKNESS,FUDGE)

%OUTLINE Returns the outline image of the foreground of an image.

% OUTLINE(FULL\_PATH), where FULL\_PATH is the directory containing

% input images, relative to the location of this M-file.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

thicken = true;

if nargin < 2

thicken = false;

FUDGE = 1;

elseif nargin < 3

FUDGE = 1;

end

% RGB TO 8-bit

if ~strcmp(class(RGB),'logical') && size(RGB,3) > 1

I = RGB2gray(RGB);

else

I = RGB;

end

% threshold

level = graythresh(I);

BW = im2bw(I,level\*FUDGE);

% morphological operations

BW = imfill(BW,'holes');

% largest object

CC = bwconncomp(BW,8);

numPixels = cellfun(@numel,CC.PixelIdxList);

[~,idx] = max(numPixels);

mask = zeros(size(BW));

mask(CC.PixelIdxList{idx}) = 1;

% origin outline

mask = bwperim(mask,8);

if ( thicken )

mask = imdilate(mask,strel('disk',THICKNESS));

end

end

function interfaceRoughness(FULL\_PATH)

%INTERFACEROUGHNESS Calculates the roughness of an interface between

%foreground and background objects.

% INTERFACEROUGHNESS(FULL\_PATH), where FULL\_PATH is the directory

% containing input images, relative to the location of this M-file.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% get image info

fprintf('Examining the folder contents...');

cd(FULL\_PATH);

importFileList = dir();

filenames = {importFileList.name};

index = regexp(filenames,'\.tif$'); % get all .tif & .tiff files

filenames = filenames(~cellfun(@isempty,index));

files = fullfile(importDir,filenames);

num = numel(files);

cd(EXECUTING\_PATH); % go back to execute scripts

for i = 1:num

% read image

I = imread(files{i});

name = files{i}(1:numel(files{i}-4));

% grayscale image with 4 shades of gray

[grayOrig,boundaryOrig] = inOutBoundary(I);

gray = grayOrig;

boundary = cat(3,boundaryOrig,boundaryOrig,boundaryOrig);

% find starting pixel on boundary

START = findStart(gray,1);

% trace

count = 1; finished = false;

while ( ~finished )

if ( count == 1 ) % first one

curr = START;

prev = [-1 -1]; % empty values

next = [0 0]; % dummy values to be completed later

map = getNeighborMap(gray,curr);

C(count) = struct('Current', curr, ...

'Previous', prev, ...

'Next', next, ...

'Map',map);

else

curr = C(count-1).Next;

prev = C(count-1).Current;

if ( nearPixelOfInterest(curr,START) && count > 5 )

finished = true;

break; % breaks out of while

end

end

% Algorithm for next pixel:

% 1. classify neighbors 3x3 pixel blocks as:

% C = current pixel: val = 0.2 (51/255)

% P = previous pixel: val = 0.2 (51/255)

% N = a possible next pixel: val = 1 (255/255)

% I = inside pixel (not on boundary): val = 0.5 (127.5/255)

% O = outside pixel (not on boundary): val = 0 (0/255)

% B = blacklist pixel that was already seen: val = 0.2 (51/255)

% 2. if just one N, choose it

% 3. else choose the N closest to the a O pixel and lowest on the

% clock numbering system (ERROR devlops, may miss some isolated

% pixels on sharp corners)

% 4. update blacklist by changing value of already-seen boundary

% pixels on the image (avoid costly search through growing array)

% find all neighbor values

map = getNeighborMap(gray,curr);

% choose next pixel

fprintf('Iteration #%i at pixel (%i,%i)...\n',count,curr(1),curr(2));

[next,gray] = chooseNext(map,curr,gray,1,false);

% update image pixels (blacklist pixels)

gray(curr(1),curr(2)) = 0.2 + 0.00001\*count;

% show images path

boundary(curr(1),curr(2),1) = 1;

boundary(curr(1),curr(2),2) = 0;

boundary(curr(1),curr(2),3) = 0;

% create new structure of boundary pixel

C(count) = struct('Current', curr, ...

'Previous', prev, ...

'Next', next, ...

'Map',map);

count = count + 1;

end

% get curvature

[roughness,~] = curvature(C);

% output images

fprintf('Writing images to file...');

imwrite(I,fullfile(outPath1,[name sprintf('-1-input-%0.5g',roughness) '.bmp']),'bmp');

imwrite(grayOrig,fullfile(outPath1,[name '-2-grayOrig' '.bmp']),'bmp');

imwrite(gray,fullfile(outPath1,[name '-3-gray' '.bmp']),'bmp');

imwrite(boundary,fullfile(outPath1,[name '-4-tracing' '.bmp']),'bmp');

fprintf('complete.');

end

fprintf('\nThis entire operation took %0.3g seconds',toc(tTotal));

end

function [gray,boundary] = inOutBoundary(IMAGE)

%INOUTBOUNDARY Re-classifies a binary image to a tri-color grayscale

%system. The background becomes black, the foreground becomes gray, and the

%interface becomes white.

% INOUTBOUNDARY(IMAGE), where IMAGE is the original image.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

if ( numel(size(IMAGE)) > 2 ) % multiple channels, presumably RGB

BW = im2bw(IMAGE(:,:,1),graythresh(IMAGE(:,:,1))) + ...

im2bw(IMAGE(:,:,2),graythresh(IMAGE(:,:,2))) + ...

im2bw(IMAGE(:,:,3),graythresh(IMAGE(:,:,3)));

else

BW = im2bw(IMAGE,graythresh(IMAGE));

end

BW = imfill(BW,'holes');

BW = bwareaopen(BW,size(IMAGE,1)^2/4); % eliminte is area is smaller than one-quarter the image

boundary = bwperim(BW,4);

% remove holes within boundary

pix1 = find1DInterface(boundary(1,:));

pix2 = find1DInterface(boundary(size(boundary,1),:));

% temporarily remove these

boundTemp = boundary;

numDelete1 = pix1\*0.1; % 10% so it won't go off the map

for i=1:numDelete1

boundTemp(1,pix1-i) = 0;

end

numDelete2 = pix2\*0.1; % 10% so it won't go off the map

for i=1:numDelete2

boundTemp(size(boundTemp,1),pix2-i) = 0;

end

% fill holes

boundTemp = imfill(boundTemp,'holes');

% add pixels back in

for i=1:numDelete1

boundTemp(1,pix1-i) = 1;

end

for i=1:numDelete2

boundTemp(size(boundTemp,1),pix2-i) = 1;

end

% reassign

boundary = boundTemp;

% determine inner area

inside = BW - boundary;

gray = zeros(size(BW));

gray = gray(:);

gray(boundary==1) = 1;

gray(inside==1) = 0.5;

gray = reshape(gray,size(BW,1),size(BW,2));

boundary = im2double(boundary);

end

function out = getNeighborMap(IMAGE,CURR)

%GETNEIGHBORMAP Returns the linear map of neighboring pixel values on an

%image for a given pixel.

% GETNEIGHBORMAP(IMAGE,CURR), where IMAGE is the 2-dimensional matrix

% containing the image and CURR is the pixel of interest.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% position 1

coords(1,1) = CURR(1)-1;

coords(1,2) = CURR(2)-1;

% position 2

coords(2,1) = CURR(1)-1;

coords(2,2) = CURR(2);

% position 3

coords(3,1) = CURR(1)-1;

coords(3,2) = CURR(2)+1;

% position 4

coords(4,1) = CURR(1);

coords(4,2) = CURR(2)+1;

% position 5

coords(5,1) = CURR(1)+1;

coords(5,2) = CURR(2)+1;

% position 6

coords(6,1) = CURR(1)+1;

coords(6,2) = CURR(2);

% position 7

coords(7,1) = CURR(1)+1;

coords(7,2) = CURR(2)-1;

% position 8

coords(8,1) = CURR(1);

coords(8,2) = CURR(2)-1;

out = zeros(size(coords,1),1);

for i = 1:size(coords,1)

row = coords(i,1); col = coords(i,2);

if ( row < 1 || row > size(I,1) || col < 1 || col > size(I,1) )

out(i) = -1; % off the map

else

out(i) = IMAGE(row,col);

end

end

end

function start = findStart(IMAGE,TYPE)

%FINDSTART Returns the linear map of neighboring pixel values on an

%image for a given pixel.

% FINDSTART(IMAGE,TYPE), where IMAGE is the 2-dimensional matrix

% containing the image and TYPE is the method for finding the starting

% pixel.

% TYPE 1 is used for tracing a large blob contained within the IMAGE.

% TYPE 2 is used for tracing a line within the IMAGE.

% TYPE 3 is used for tracing a segment that extends between two edges of

% the IMAGE.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

switch TYPE

case 1 % for global roughness tracing around a blob

% start from middle outward

I = im2bw(IMAGE,graythresh(IMAGE));

I = imfill(I,'holes');

boundary = bwperim(I,4);

boundary = bwmorph(boundary,'skel',Inf);

% find centroid

s = regionprops(I,'Centroid');

col = round(s(1).Centroid(2));

% find pixel on boundary directly above centroid (at 12 o'clock)

temp = boundary(:,col); % return column vector that goes through centroid

row = min(find(temp==1));

start = [row,col];

case 2 % for tracing segment of image (just a line) that touches edges (e.g. typical 20x images)

start = [1,find1DInterface(IMAGE(1,:))];

case 3 % for tracing a line that is guaranteed to touch edges

[x,y] = find(IMAGE==1);

idx = find(x==1|x==size(IMAGE,1)|y==1|y==size(IMAGE,1));

start = [x(idx(1)) y(idx(1))];

otherwise

% unsophisticated method: problem is that it may find a dead end pixel

% find boundary crudely with incorrectly-ordered list of pixels

[x,y] = find(IMAGE==1);

% find starting pixel on boundary

start = [x(1) y(1)];

end

end

function out = find1DInterface(ARRAY)

%FIND1DINTERFACE Finds and returns the white pixel that interfaces with

%many black pixels in a 1D array.

% FIND1DINTERFACE(ARRAY), where ARRAY is the 1-dimensional binary

% structure that will be analyzed.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

out = 1;

zeroSize = 0;

for i = 1:numel(ARRAY)-1;

leftSide = sum(ARRAY(1:i));

rightSide = sum(ARRAY(i+1:end));

if ( leftSide == 0 && i > zeroSize )

zeroSize = i;

out = i+1;

end

if ( rightSide == 0 && (numel(ARRAY)-i) > zeroSize )

zeroSize = numel(ARRAY) - i;

out = i;

end

end

end

function [nextPixel,GRAY] = chooseNext(MAP,CURR,GRAY,DEBUG)

%CHOOSENEXT Determines which pixel to choose next when tracing an

%interface.

% CHOOSENEXT(MAP,CURR,GRAY,DEBUG), where MAP is the output from

% getNeighborMap(), CURR is the pixel of interest, GRAY is the imaged

% being traced, and DEBUG toggles debugging code.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% choose N adjacent to most number of O pixels

nextPixel = 0; outCount = 0;

for i = 1:size(MAP,1)

outCountL = 0; outCountR = 0;

% find adjacent 0s

if ( MAP(i) == 1 ) % found a possible next pixel

if ( DEBUG )

fprintf('Position %i is a 1 with this MAP: %0.1f %0.1f %0.1f %0.1f %0.1f %0.1f %0.1f %0.1f\n', ...

i,MAP(1),MAP(2),MAP(3),MAP(4),MAP(5),MAP(6),MAP(7),MAP(8));

end

% check if not a dead end first

map2 = getNeighborMap(GRAY,getCoords(CURR,i));

% change map2 so that it shows where we came from and does not

% include it in the deadEnd assessment i = where we are moving

% to.

switch i

case 1

map2(5) = 0.2;

case 2

map2(6) = 0.2;

case 3

map2(7) = 0.2;

case 4

map2(8) = 0.2;

case 5

map2(1) = 0.2;

case 6

map2(2) = 0.2;

case 7

map2(3) = 0.2;

case 8

map2(4) = 0.2;

otherwise

end

if ( DEBUG )

fprintf('Position %i is a 1 with this map2: %0.1f %0.1f %0.1f %0.1f %0.1f %0.1f %0.1f %0.1 -->', ...

i,map2(1),map2(2),map2(3),map2(4),map2(5),map2(6),map2(7),map2(8));

deadEnd(map2);

if ( deadEnd(map2) )

fprintf('DEAD END\n');

else

fprintf('Not dead end...\n');

end

end

if ( ~deadEnd(map2) )

% look left

for j = i-1:-1:1 % look left from next pixel

if ( j == 1 && MAP(j) == 0 )

outCountL = outCountL + 1;

for k = size(MAP,1):-1:i+1 % wrap around looking left

if ( MAP(k) == 0 )

outCountL = outCountL + 1;

else

break; % get out for-loop if hit a non-zero

end

end

elseif ( MAP(j) == 0 ) % wrap around needed

outCountL = outCountL + 1;

else

break; % get out for-loop if hit a non-zero

end

end

% look right

for j = i+1:size(MAP,1) % look right from next pixel

if ( j == size(MAP,1) && MAP(j) == 0 ) % wrap around needed

for k = 1:i-1 % wrap around looking right

if ( MAP(k) == 0 )

outCountR = outCountR + 1;

else

break; % get out of for-loop if hit a non-zero

end

end

elseif ( MAP(j) == 0 )

outCountR = outCountR + 1;

else

break; % get out for-loop if hit a non-zero

end

end

if ( DEBUG )

fprintf('%i: final outCountL+outCountR = %i; ',i,(outCountL+outCountR));

fprintf('final outCount = %i; ',outCount);

end

if ( (outCountL+outCountR) >= outCount )

% Any pixel can overwrite a currently zero next pixel,

% but only a face pixel can overwrite another face

% pixel and corner pixels can't overwrite face pixels.

% This biases the choice toward face pixels.

if ( nextPixel == 0 || nextPixel == 1 || ...

nextPixel == 3 || nextPixel == 5 || ...

nextPixel == 7 )

outCount = outCountL + outCountR;

nextPixel = i;

end

end

if ( DEBUG )

fprintf('nextPixel = %i\n',nextPixel);

end

else

GRAY(getCoords(CURR,i)) = 0.05; % if dead end, mark as seen to avoid

end

end

end

% update image with next pixel

if ( nextPixel == 0 )

nextPixel = min(find(MAP==1));

end

nextPixel = getCoords(CURR,nextPixel);

end

function out = deadEnd(MAP)

%DEADEND Returns a boolean, true if the pixel of interest is a dead end

%or false if the pixel of interest is not a dead end.

% DEADEND(MAP), where MAP is map of pixels surrounding a pixel of

% interest.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

err1 = false; err2 = false; err3 = false; out = false;

if ( numel(MAP(MAP==0.5)) < 1 ) % not near inner part

out = true;

err1 = true;

end

if ( intoSpace(MAP) ) % surrounded by black pixels

out = true;

err3 = true;

end

if ( err1 && err2 ) % if isolated pixel (seemingly), but only option, then take it

out = false;

end

if ( numel(MAP(MAP==-1)) > 0 && numel(MAP(MAP==1)) < 1 ) % only say true if at edge ands there is no other option

out = true;

end

end

function out = intoSpace(MAP)

%INTOSPACE Returns a Boolean, true if the pixel of interest is surrounded

%by black pixels and false otherwise.

% INTOSPACE(MAP), where MAP is map of pixels surrounding a pixel of

% interest.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% looking at 4 directions only

if ( MAP(2) == 0 && MAP(6) == 0 && numel(MAP(MAP==0.5)) < 1 )

out = true;

elseif ( MAP(4) == 0 && MAP(8) == 0 && numel(MAP(MAP==0.5)) < 1 )

out = true;

else

out = false;

end

% if you are not next to any inside (0.5) pixels, then you're probably

% a dead end

if ( numel(MAP(MAP==0.5)) < 1 )

out = true;

end

end

function [out,curvatureArray] = curvature(C)

%CURVATURE Determines the curvature of an interface. This function is an

%implementation of the mathematical model used by Lacayo, et al. (citation

%below).

% CURVATURE(C), C is the ordered list of pixels traced along

% an interface.

%

% Implementation of method from:

% Lacayo, et al. Emergence of large-scale cell morphology and movement

% from local actin filament growth dynamics. PLoS Biology, 2007.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% discrete 1st derivative (central approximation)

counter = 1; SAMPLE\_FREQ = 1;

for i = 1:SAMPLE\_FREQ:numel(C)-SAMPLE\_FREQ

x\_prime(counter) = (C(i+SAMPLE\_FREQ).Current(1) - C(i).Current(1))/(SAMPLE\_FREQ);

y\_prime(counter) = (C(i+SAMPLE\_FREQ).Current(2) - C(i).Current(2))/(SAMPLE\_FREQ);

counter = counter + 1;

end

for i = 1:numel(x\_prime)-1

x\_dubPrime(i) = (x\_prime(i+1) - x\_prime(i))/(1);

y\_dubPrime(i) = (y\_prime(i+1) - y\_prime(i))/(1);

end

% for calculation, make dimensions agree

x\_prime(end) = [];

y\_prime(end) = [];

% curvature (Lacayo & Theriot, PLoS Biology, 2007)

curvatureArray = (x\_prime.\*y\_dubPrime - y\_prime.\*x\_dubPrime)./ ...

(x\_prime.^2 + y\_prime.^2).^1.5;

% "To determine the values of 'local leading-edge curvature,' we summed

% the absolute values of the curvatures along the leading edge, and

% multiplied this by the length of the leading edge to account for the

% fact that smaller keratinocytes will have higher total curvature due to

% their size alone."

% Lacayo, et al. Emergence of large-scale cell morphology and movement

% from local actin filament growth dynamics. PLoS Biology, 2007.

% C = sum(abs(curvatureArray)).\*perimeter;

% out = mean(abs(curvatureArray));

out = sum(abs(curvatureArray))/numel(C);

end

function voronoiApprox(FULL\_PATH)

%VORONOIAPPROX Analyzes and outputs approximate cell area data and images

%based on Voronoi tessellation.

% VORONOIAPPROX(FULL\_PATH), where FULL\_PATH is the directory containing

% input images, relative to the location of this M-file.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% import nuclei from adjusted images

inputFolder2 = fullfile(FULL\_PATH,'output-adjustedOriginals');

EXECUTING\_PATH = pwd;

cd(inputFolder2);

importFileList = dir();

filenames = {importFileList.name};

index = regexp(filenames,'output-adjustedOriginal\.bmp$');

filenames = filenames(~cellfun(@isempty,index));

filesAdjusted = fullfile(inputFolder2,filenames);

cd(EXECUTING\_PATH);

% import ring masks and thresholds

inputFolder3 = fullfile(FULL\_PATH,'output-intermediates');

cd(inputFolder3);

importFileList = dir();

filenames = {importFileList.name};

index = regexp(filenames,'output-intermed2-ringMask\.bmp$'); % get only the ringMask files

filenames = filenames(~cellfun(@isempty,index));

filesRing = fullfile(inputFolder3,filenames);

num = numel(filesRing);

filenames = {importFileList.name};

index = regexp(filenames,'output-intermed1-thresholded\.bmp$');

filenames = filenames(~cellfun(@isempty,index));

filesCellMass = fullfile(inputFolder3,filenames);

cd(EXECUTING\_PATH);

% output folders

outputFolder2 = fullfile(FULL\_PATH,'output-results');

for n = 1:numel(filesAdjusted) % for each image

% read nuclei and actin

name = filenames{n}(1:end-33);

image = imread(filesAdjusted{n});

green = image(:,:,2);

blue = image(:,:,3);

% get image size

IMAGE\_DIM = size(image(:,:,1));

% import total mass to make sure nuclei aren't outside rim

tic; fprintf('\tConstructing nuclei band masks...');

cellMass = imread(filesCellMass{n});

totalMass = imfill(cellMass,'holes');

totalMass = im2uint8(totalMass);

% create rim mask from actin

rim = outline(totalMass); % assume already properly thresholded

rim(1,:) = 0; rim(IMAGE\_DIM(1),:) = 0;

rim(:,1) = 0; rim(:,IMAGE\_DIM(2)) = 0;

rim = im2double(rim);

% remove nuclei that aren't part of cell mass

blue = im2bw(blue,graythresh(blue));

blue = excludeBlobs(blue,totalMass);

% nuclei thresholding, excluding small and large blobs

LL = 5; UL = 1000;

CC = bwconncomp(blue,8);

s = regionprops(CC,'Area');

idx = find([s.Area] > LL & [s.Area] < UL);

mask = ismember(labelmatrix(CC),idx);

mask = imclearborder(mask);

mask = mask + rim;

fprintf('complete. [%0.3g s]\r',toc);

% tessellation

CC = bwconncomp(mask,8);

s = regionprops(CC,'PixelList','PixelIdxList','Centroid');

centroids = cat(1,s.Centroid);

[V,C] = voronoin(centroids);

maskBlack = zeros(size(rim));

h = figure; imshow(maskBlack); hold on;

for i = 1:length(C)

if ( all(C{i}~=1) ) % If at least one of the indices is 1,

% then it is an open region and we can't

% patch that.

x = V(C{i},1);

y = V(C{i},2);

% remove off-the-map vertices

clearvars idx1 idx2 idx3;

idx1 = find(x<1|x>size(maskBlack,1));

idx2 = find(y<1|y>size(maskBlack,1));

x(idx1) = 5000; y(idx1) = 5000;

x(idx2) = 5000; y(idx2) = 5000;

x(x==5000) = []; y(y==5000) = [];

% remove outside-of-mask vertices by comparing to largest negative blob

plot(x,y,'w-');

end

end

hold off;

X = frame2im(getframe(h));

close(h);

X = imresize(X,[512,512]);

X = X(:,:,1);

X = im2double(X);

X = im2bw(X,graythresh(X));

X = bwmorph(X,'skel',Inf);

% fix up total mass

negativeMass = im2double(~totalMass);

negativeMass = imerode(negativeMass,strel('disk',7));

negativeMass = imresize(negativeMass,[512,512]);

negativeMass = im2bw(negativeMass,graythresh(negativeMass));

rimSmall = imresize(rim,[512,512]);

rimSmall = im2bw(rimSmall,graythresh(rimSmall));

rimSmall = bwmorph(rimSmall,'skel',Inf);

voronoiFinal = X + rimSmall - negativeMass;

voronoiFinal2 = X - negativeMass;

voronoiFinal = im2bw(voronoiFinal,graythresh(voronoiFinal));

voronoiFinal2 = im2bw(voronoiFinal2,graythresh(voronoiFinal2));

temp = imresize(green,[512,512]);

finalMask(:,:,1) = im2double(temp) + rimSmall;

finalMask(:,:,2) = im2double(temp) + voronoiFinal2;

finalMask(:,:,3) = im2double(temp);

temp2 = imresize(blue,[512,512]);

CCtemp2 = bwconncomp(temp2,8);

s = regionprops(CCtemp2,'PixelList','PixelIdxList','Centroid');

centroids2 = cat(1,s.Centroid);

h = figure; imshow(finalMask); hold on;

plot(centroids2(:,1),centroids2(:,2),'bo','MarkerFaceColor','b','MarkerSize',2);

hold off;

saveas(h,fullfile(outputFolder2,[name '-output-results4-tessellationOverlay' '.bmp']),'bmp');

close(h);

cells = ~voronoiFinal;

% remove large and small blobs

LB = 20;

UB = 10000;

cellsOut = xor(bwareaopen(cells,LB,4), bwareaopen(cells,UB,4));

imwrite(cellsOut,fullfile(outputFolder2,[name '-output-results3-tessellation' '.bmp']),'bmp');

end

end

function elongation(FULL\_PATH)

%ELONGATION Analyzes and outputs data and images about cellular area and

%aspect ratio.

% ELONGATION(FULL\_PATH), where FULL\_PATH is the directory containing

% input images, relative to the location of this M-file.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% input variables

saveImages = true;

NUM\_CHANNELS = 3;

% make output folder to store new images

outFolder = 'output-elongation';

outDir = fullfile(FULL\_PATH,outFolder);

if ( ~exist(outDir,'dir') )

mkdir(outDir);

end

outputExcel = fullfile(FULL\_PATH,outFolder,'output-elongation.xlsx');

% image file information

fprintf('Examining the folder contents...');

EXECUTING\_PATH = pwd;

cd(FULL\_PATH);

importFileList = dir();

filenames = {importFileList.name};

index = regexp(filenames,'\.tif$'); % get all .tif & .tiff files

filenames = filenames(~cellfun(@isempty,index));

files = fullfile(FULL\_PATH,filenames);

num = numel(files);

cd(EXECUTING\_PATH);

fprintf('done.\r\n');

% analysis for each image

count=1;

if ( saveImages )

h1 = figure('units','normalized','outerposition',[0 0 1 1]); hold on;

h2 = figure('units','normalized','outerposition',[0 0 1 1]); hold on;

h3 = figure('units','normalized','outerposition',[0 0 1 1]); hold on;

h4 = figure('units','normalized','outerposition',[0 0 1 1]); hold on;

h5 = figure('units','normalized','outerposition',[0 0 1 1]); hold on;

h6 = figure('units','normalized','outerposition',[0 0 1 1]); hold on;

end

for n=1:NUM\_CHANNELS:num

fprintf('Analyzing image %i of %i...',count,num/3);

% new filename

stem{count,1} = filenames{n}(1:end-4);

% import images

clearvars green0 red0 blue0

green0 = imread(files{n});

red0 = imread(files{n+1});

blue0 = imread(files{n+2});

% fill in and threshold nuclei

clearvars nuclei blue

blue = blue0;

blue = im2bw(blue,graythresh(blue));

blue = imclose(blue,strel('disk',3));

blue = imfill(blue,'holes');

blue = xor(bwareaopen(blue,100),bwareaopen(blue,5000));

blue = im2double(blue);

blue = watershedImage(blue,0.8,100,false);

nuclei = imdilate(blue,strel('disk',5));

% process red channel

clearvars rEdge red

red = red0;

red = im2double(red);

actinArea = red;

actinArea = im2bw(actinArea,graythresh(actinArea)\*0.5);

actinArea = bwareaopen(actinArea,5000);

actinArea = im2double(actinArea);

% process green channel

clearvars gEdge green

gEdge = green0;

gEdge = edge(gEdge,'canny');

gEdge(nuclei==1) = 0;

gEdge(actinArea==0) = 0;

gEdge2 = gEdge;

for i = -90:5:90

gEdge = imclose(gEdge,strel('line',5,i));

end

gEdge = bwareaopen(gEdge,100);

gEdge2 = bwareaopen(gEdge2,100);

gEdge2 = imdilate(gEdge2,strel('disk',3));

gTemp = gEdge;

gTemp(gEdge2==1) = 0;

ggg = cat(3,gTemp,gEdge2,zeros(size(gEdge)));

imwrite(gEdge2,fullfile(outDir,[stem{count} '-12-GwithoutConnections.bmp']),'bmp');

imwrite(ggg,fullfile(outDir,[stem{count} '-13-Goverlap.bmp']),'bmp');

% edge detection function

clearvars G Img\_smooth cells

G = fspecial('gaussian',15,1.5);

Img\_smooth = conv2(im2double(gEdge),G,'same'); % smooth image by Gaussian convolution

Img\_smooth = bwmorph(Img\_smooth,'thin','Inf');

cells = imcomplement(imdilate(Img\_smooth,strel('disk',3)));

cells(actinArea==0) = 0;

cells = imfill(cells,'holes');

cells = imclearborder(cells);

% if a blob has two nuclei, use voronoi to approximate membrane

clearvars ccCells ccNuc

ccCells = bwconncomp(cells);

cellsFinal = zeros(size(cells));

dubFinal = zeros(size(cells));

clusterFinal = zeros(size(cells));

emptyFinal = zeros(size(cells));

fprintf(' %i objects\n',ccCells.NumObjects);

for i = 1:ccCells.NumObjects

if ( mod(i,100) == 0 )

fprintf('\n');

end

clearvars cIndiv nuc numNuc

cIndiv = zeros(size(blue));

cIndiv = cIndiv(:);

cIndiv(ccCells.PixelIdxList{i}) = 1;

cIndiv = reshape(cIndiv,size(blue,1),size(blue,2));

nuc = blue;

nuc(cIndiv==0) = 0;

numNuc = bweuler(nuc);

switch numNuc

case 0,

% do nothing

emptyFinal(cIndiv==1) = 1;

fprintf('0');

case 1,

cellsFinal(cIndiv==1) = 1;

fprintf('.');

case 2,

fprintf('2');

% draw line between two nuclei centroids perpendicular to

% the line between nuclei

clearvars ccNuc sNuc centroids mid ang angPerp

ccNuc = bwconncomp(nuc);

sNuc = regionprops(ccNuc,'Centroid');

centroids = round(cat(1,sNuc.Centroid));

mid = round([(centroids(1,1)+centroids(2,1))/2,(centroids(1,2)+centroids(2,2))/2]);

ang = asind((centroids(1,2)-centroids(2,2))/pdist(centroids));

angPerp = ang + 90;

added = zeros(size(cIndiv));

added(mid(2),mid(1)) = 1;

added = imdilate(added,strel('line',250,angPerp));

added = imdilate(added,strel('disk',3));

cIndiv(added==1) = 0;

dubFinal(cIndiv==1) = 1;

otherwise,

fprintf('M');

% Voronoi 3+ nuclei

clusterFinal(cIndiv==1) = 1;

end

end

fprintf(' DONE!\n');

clearvars classified nucLoc0 nucLoc tempIm

nucLoc0 = bwmorph(blue,'shrink',Inf);

tempIm = cat(3,clusterFinal+dubFinal+cellsFinal,zeros(size(cells)),nucLoc0);

tempIm = sum(tempIm,3);

tempIm(nucLoc0==0) = 0;

tempIm(tempIm<2) = 0; % remove nuclei that aren't in any classification

nucLoc = imdilate(tempIm,strel('disk',5));

classified = cat(3,clusterFinal+emptyFinal-nucLoc,cellsFinal+dubFinal+emptyFinal-nucLoc,dubFinal+nucLoc);

cellsFinal = cellsFinal + dubFinal;

outlined1 = bwperim(cells);

outlined2 = bwperim(cellsFinal);

outlined1 = imdilate(outlined1,strel('disk',1));

outlined2 = imdilate(outlined2,strel('disk',1));

% generate RGB image

clearvars combo orig

orig = cat(3,red0,green0,blue0);

combo = cat(3,red,gEdge,blue);

olay1 = imoverlay(orig,outlined1,[0 1 0]);

olay2 = imoverlay(orig,outlined2,[0 1 0]);

% calculations

clearvars ccAll sAll ccMonoNuc sMonoNuc

cells = bwareaopen(cells,1000);

ccAll = bwconncomp(cells);

ccMonoNuc = bwconncomp(cellsFinal);

sAll = regionprops(ccAll,'MajorAxisLength','MinorAxisLength','Orientation','Eccentricity','Area');

sMonoNuc = regionprops(ccMonoNuc,'MajorAxisLength','MinorAxisLength','Orientation','Eccentricity','Area');

allAR = cat(1,sAll.MajorAxisLength)./cat(1,sAll.MinorAxisLength);

mnAR = cat(1,sMonoNuc.MajorAxisLength)./cat(1,sMonoNuc.MinorAxisLength);

allEcc = cat(1,sAll.Eccentricity);

mnEcc = cat(1,sMonoNuc.Eccentricity);

allArea = cat(1,sAll.Area)\*(0.7325^2); % 20x conversion to microns

mnArea = cat(1,sMonoNuc.Area)\*(0.7325^2); % 20x conversion to microns

calc{count,1} = allAR;

calc{count,2} = allEcc;

calc{count,3} = allArea;

calc{count,4} = mnAR;

calc{count,5} = mnEcc;

calc{count,6} = mnArea;

calcStats(count,1) = mean(mnAR);

calcStats(count,2) = mean(mnEcc);

calcStats(count,3) = mean(mnArea);

if ( saveImages && count <= RR\*CC )

% histograms

clearvars X

X = 0:0.5:10;

figure(h1);

calcStat{count,1} = histc(calc{count,1},X);

sh = subplot(RR,CC,count);

bar(X,calcStat{count,1},1,'LineWidth',2);

xlim([0,10]);

set(sh,'FontSize',12);

ylabel('Frequency'); xlabel('Aspect Ratio');

hold off;

clearvars X

X = 0:0.1:1;

figure(h2);

calcStat{count,2} = histc(calc{count,2},X);

sh = subplot(RR,CC,count);

bar(X,calcStat{count,2},1,'LineWidth',2);

xlim([0,1]);

ylabel('Frequency'); xlabel('Eccentricity');

set(sh,'FontSize',12);

hold off;

clearvars X

X = 0:100:5000;

figure(h3);

calcStat{count,3} = histc(calc{count,3},X);

sh = subplot(RR,CC,count);

bar(X,calcStat{count,3},1,'LineWidth',2);

xlim([0,5000]);

ylabel('Frequency'); xlabel('Area');

set(sh,'FontSize',12);

hold off;

clearvars X

X = 0:0.5:10;

figure(h4);

calcStat{count,4} = histc(calc{count,4},X);

sh = subplot(RR,CC,count);

xlim([0,10]);

ylabel('Frequency'); xlabel('Aspect Ratio');

set(sh,'FontSize',12);

hold off;

clearvars X

X = 0:0.1:1;

figure(h5);

calcStat{count,5} = histc(calc{count,5},X);

sh = subplot(RR,CC,count);

bar(X,calcStat{count,5},1,'LineWidth',2);

xlim([0,1]);

ylabel('Frequency'); xlabel('Eccentricity');

set(sh,'FontSize',12);

hold off;

clearvars X

X = 0:100:5000;

figure(h6);

calcStat{count,6} = histc(calc{count,6},X);

sh = subplot(RR,CC,count);

bar(X,calcStat{count,6},1,'LineWidth',2);

xlim([0,5000]);

ylabel('Frequency'); xlabel('Area');

set(sh,'FontSize',12);

hold off;

% write new images to file

imwrite(orig,fullfile(outDir,[stem{count} '-1-combo.bmp']),'bmp');

imwrite(classified,fullfile(outDir,[stem{count} '-2-classified.bmp']),'bmp');

imwrite(olay1,fullfile(outDir,[stem{count} '-3-overlayAll.bmp']),'bmp');

imwrite(olay2,fullfile(outDir,[stem{count} '-4-overlayMonoNuc.bmp']),'bmp');

imwrite(cat(3,zeros(size(outlined1)),outlined1,zeros(size(outlined1))),fullfile(outDir,[stem{count} '-5-outlineAll.bmp']),'bmp');

imwrite(cat(3,zeros(size(outlined2)),outlined2,zeros(size(outlined2))),fullfile(outDir,[stem{count} '-6-outlineMonoNuc.bmp']),'bmp');

imwrite(cells,fullfile(outDir,[stem{count} '-7-cellsAll.bmp']),'bmp');

imwrite(cellsFinal,fullfile(outDir,[stem{count} '-8-cellsMonoNuc.bmp']),'bmp');

imwrite(red,fullfile(outDir,[stem{count} '-9-R.bmp']),'bmp');

imwrite(gEdge,fullfile(outDir,[stem{count} '-10-G.bmp']),'bmp');

imwrite(blue,fullfile(outDir,[stem{count} '-11-B.bmp']),'bmp');

end

fprintf('done.\n\n');

count = count + 1;

end

if ( saveImages )

export\_fig(fullfile(outDir,'stats1-ALL-AR.bmp' ),h1);

delete(h1);

export\_fig(fullfile(outDir,'stats2-ALL-Eccentricity.bmp' ),h2);

delete(h2);

export\_fig(fullfile(outDir,'stats3-ALL-Area.bmp' ),h3);

delete(h3);

export\_fig(fullfile(outDir,'stats4-MonoNuc-AR.bmp' ),h4);

delete(h4);

export\_fig(fullfile(outDir,'stats5-MonoNuc-Eccentricity.bmp' ),h5);

delete(h5);

export\_fig(fullfile(outDir,'stats6-MonoNuc-Area.bmp' ),h6);

delete(h6);

end

warning('off','MATLAB:xlswrite:AddSheet');

xlswrite(outputExcel,stem,1,'A2');

head = {'Mean AR (Mono-Nuclear Only)','Mean Eccentricity (Mono-Nuclear Only','Area (Mono-Nuclear Only'};

xlswrite(outputExcel,head,1,'B1');

xlswrite(outputExcel,calcStats,1,'B2');

end

function ecm(FULL\_PATH)

%ECM Analyzes and outputs data about fiber length and orientation.

% ECM(FULL\_PATH), where FULL\_PATH is the directory containing

% input images, relative to the location of this M-file.

%

% Copyright 2015, written by Adam C. Canver & Alisa Morss Clyne

% input

interface = false;

saveIms = true; saveExcel = true; statSave = true; rosePlots = true;

LENGTHCUT = 20; % 20 ~= 4.5 um

groups = [3,2,3]; % number of image files per experimental group (assumes they are grouped files)

RR = 3; CC = 3;

gridOrd = [1,2,3,4,5,7,8,9];

names0 = {'4 kPa','4 kPa','4 kPa','14 kPa','14 kPa','50 kPa','50 kPa','50 kPa'}; % change based on files

% image file info

t0 = tic; fprintf('Examining the folder contents...');

EXECUTING\_PATH = pwd;

cd(FULL\_PATH);

importFileList = dir();

filenames = {importFileList.name};

index = regexp(filenames,'\.tif$'); % get all .tif & .tiff files

filenames = filenames(~cellfun(@isempty,index));

files = fullfile(FULL\_PATH,filenames);

num = numel(files);

outputFolder1 = 'outputAnalysis-FN';

if ( ~exist(outputFolder1,'dir') )

mkdir(outputFolder1);

end

cd(outputFolder1);

outputFolder2 = 'statistics';

if ( ~exist(outputFolder2,'dir') )

mkdir(outputFolder2);

end

outputFolder3 = 'rose';

if ( ~exist(outputFolder3,'dir') )

mkdir(outputFolder3);

end

if ( saveIms || statSave )

outputFolder1 = fullfile(importDir,outputFolder1);

outputFolder2 = fullfile(outputFolder1,outputFolder2);

outputFolder3 = fullfile(outputFolder1,outputFolder3);

msg = 'IMAGES AND EXCEL FILES WILL BE GENERATED/SAVED.';

else

outputFolder1 = '';

outputFolder2 = '';

outputFolder3 = '';

msg = 'NO IMAGES OR EXCEL FILES WILL BE GENERATED/SAVED.';

end

cd(EXECUTING\_PATH);

fprintf('complete. [%0.2f s]\r',toc(t0));

fprintf(['\r' msg '\r\r']);

% ADJUST IMAGES

nCount = 1; calc0im = cell(num/3,4); calc0rawIm = cell(num/3,2);

for n = 1:3:num % for each image with three separate channels

t = tic; fprintf('IMAGE #%i of %i:\r',nCount,num/3);

% import image

clearvars blue0 green0 red0 blue green red

blue0 = imread(files{n});

green0 = imread(files{n+1});

red0 = imread(files{n+2});

blue = blue0;

green = green0;

red = red0;

name = filenames{n}(1:end-6);

names{nCount} = name;

scaleB = [0.05, 0.5];

blue = imadjust(blue,scaleB,[],1);

% get nuclear mask

clearvars bwB

bwB = im2bw(blue,graythresh(blue)\*0.5);

bwB = imfill(bwB,'holes');

bwB = bwareaopen(bwB,200);

% cell area mask, assuming it is moving to the right

clearvars bwR r1 r2

bwR = im2bw(red,graythresh(red)\*0.5);

bwR = imfill(bwR,'holes');

bwR = bwareaopen(bwR,5000);

% morphology for approach 2

clearvars filt bw2 br brBig fib

filt = medianFilter(green);

filt = gaussFilter(filt,50,[5 5],1);

bw = im2bw(filt,graythresh(filt));

bw(bwR==0) = 0;

bw2 = bw;

bw2 = bwmorph(bw2,'thin',Inf);

br = bwmorph(bw2,'branchpoints'); % branch points

brBig = imdilate(br,strel('square',2));

fib = bw2;

fib(brBig==1) = 0; % fibers only without branch points

clearvars fibBelow3 fib3to10 fibAbove10 fibClassified fibClassifiedOverlay

fibBelow3 = xor(bwareaopen(fib,0),bwareaopen(fib,3));

fib3to10 = xor(bwareaopen(fib,3),bwareaopen(fib,10));

fibAbove10 = bwareaopen(fib,10);

fibClassified = cat(3,fibBelow3+fib3to10,fib3to10+fibAbove10,zeros(size(fib)));

fibClassifiedThick = cat(3,imdilate(fibBelow3+fib3to10,strel('disk',1)),imdilate(fib3to10+fibAbove10,strel('disk',1)),zeros(size(fib)));

fibClassifiedOverlay = imoverlay(green,imdilate(fibBelow3,strel('disk',1)),[1 0 0]);

fibClassifiedOverlay = imoverlay(fibClassifiedOverlay,imdilate(fib3to10,strel('disk',1)),[1 1 0]);

fibClassifiedOverlay = imoverlay(fibClassifiedOverlay,imdilate(fibAbove10,strel('disk',1)),[0 1 0]);

fibClassifiedOverlayR = imoverlay(green,imdilate(fibBelow3,strel('disk',1)),[1 0 0]);

fibClassifiedOverlayY = imoverlay(green,imdilate(fib3to10,strel('disk',1)),[1 1 0]);

fibClassifiedOverlayG = imoverlay(green,imdilate(fibAbove10,strel('disk',1)),[0 1 0]);

% morphology for approach 2

clearvars filtA bw2A brA brAbig fibA

filtA = imadjust(red,[0,0.5],[],1);

filtA = medianFilter(filtA);

filtA = gaussFilter(filtA,50,[5 5],1);

filtA(bwB==1) = 0;

bwA = im2bw(filtA,graythresh(filtA));

bwA(bwR==0) = 0;

bw2A = bwA;

bw2A = bwmorph(bw2A,'thin',Inf);

brA = bwmorph(bw2A,'branchpoints'); % branch points

brAbig = imdilate(brA,strel('square',2));

fibA = bw2A;

fibA(brAbig==1) = 0; % fibers only without branch points

fibA = bwareaopen(fibA,LENGTHCUT);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% binning

clearvars rim M B0 B approxLine

rim = outline(im2uint8(bwR));

M = 15;

B = size(rim,1)/4;

[approxLine,~,~] = imageLine(rim,M,B);

migrationDirection = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% adjust y-intercept, B, so that the line creats a valid ring

% first, move the line until it hits the tip of the rim, then subtract from there

tic; fprintf('\tFinding ring0:\r\t\t'); iter = 1;

if ( ~interface )

[ring0,~,~] = imageLine(approxLine,M,size(approxLine,2));

else

tip = false; longEnough = false; symLim = 75;

while ( (~tip && ~longEnough) )

fprintf('\*');

if ( mod(iter,symLim) == 0 )

fprintf('%5.0i\r\t\t',iter);

end

% shift line to move TOWARDS migrationDirection

B = B + 1;

% make continuous line image

B = round(B);

[ring0,~,~] = imageLine(approxLine,M,B);

% check if at rim

ring0Vec = ring0(:);

lineIdx = find(ring0Vec==1); % indices of the line in the image

if sum(bwR(lineIdx)>0) == 0 % if there are no white pixels from rim along the line, then we found the tip

tip = true;

end

iter = iter + 1;

if ( tip && iter<10 ) % premature tip since line started in black pixels, so jump back and start over

if ( migrationDirection >= 0 && migrationDirection <= 180 )

B = B - 50;

elseif ( migrationDirection < 0 && migrationDirection >= -180 )

B = B + 50;

end

tip = false; iter = 1;

elseif ( tip && iter>10 )

longEnough = true; % legit tip

end

end

end

fprintf('DONE!\r');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% add parallel lines with same slope, but just change y-intercept, B

tic; fprintf('\tConstructing ring mask...');

% 100 pixels at 20x (0.73 um/pixel) is 200um (274 originally)

% 68 pixels at 10x (1.4691 um/pix) is 100um

% 437 pixels at 60x (0.2287 um/pix) is 100um

width = 50;

resolution = 0.2287;

numRings = 20; deltaB = round(width/resolution);

ringRest = zeros(size(ring0));

if (~interface)

B = size(ring0,2);

end

for i=1:numRings-1 % subtract one to account for ring0

% adjust B to move line AWAY from migrationDirection

B = B - deltaB;

% make sure it is not off the image if horiz or vert

if ( B >= 1 )

% make new line

[rings,~,~] = imageLine(ring0,M,B);

% combine with others

ringRest = ringRest + rings;

end

end

% adjust ring mask so ring0 can be identified

% ring0 is red, other rings are blue

clearvars ringMaskFinal;

ringMaskFinal = cat(3,ring0,zeros(size(ring0)),ringRest);

ringMaskFinal2 = cat(3,ring0+ringRest,ring0+ringRest,ringRest);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% prepare to break up into bands

[perpLine,~,~] = imageLine(ring0,0,size(ring0,1)/2);

% find neighboring line

% do this by finding the closest intersection on perpLine that hasn't already been seen

clearvars r c distList

[r,c] = find(perpLine==1&(ringRest==1|ring0==1));

X = cat(2,r,c);

% find Euclidean distance between each intersection

distList = pdist(X,'euclidean')';

% reconstitute to 2D matrix

clearvars count i distM

distM = zeros(size(X,1)-1);

count = 1; i = 1;

while ( i <= numel(distList) )

distM(count:size(X,1)-1,count) = distList(i:i+size(X,1)-1-count)';

count = count + 1;

i = i + size(X,1)-count + 1;

end

% make simple distance matrix

clearvars distListS

distListS = [1; distList(1:size(X,1)-1)]; % now equal length to size(X,1)

% remove redundant points assuming they have similar distances +/- a

% couple pixels

clearvars cutoff distListF

cutoff = 10;

distListF = zeros(size(distListS));

for i = 2:numel(distListS)

if ( distListS(i) - distListS(i-1) < cutoff )

distListF(i) = -50;

else

distListF(i) = distListS(i);

end

end

X(distListF==-50,:) = [];

X(end+1,:) = [X(1,1) 1];

X = sort(X,1,'descend');

X = round(X);

fprintf('DONE!\r');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% for each bin

bandsFinal = zeros(size(ring0));

bandsFinal = im2uint8(bandsFinal);

for i = 1:size(X,1)-1 % number of perpLine intersections (may be less than number of bin lines)

tic; fprintf('\tMeasurements for each bin %i of %i...',i,size(X,1)-1);

% find two lines

clearvars L1 L2

L1 = zeros(size(ring0));

L2 = zeros(size(ring0));

L1(:,X(i,2)\*ones(size(L1,1),1)) = 1;

L2(:,X(i+1,2)\*ones(size(L1,1),1)) = 1;

% find a point between the lines

clearvars midPix

midPix = [size(L1,1)/2 round(X(i+1,2)+(X(i,2)-X(i+1,2))/2)];

% isolate band

clearvars band bandFocus

band = logical(L1+L2);

bandFocus = imfill(band,[midPix(1) midPix(2)],4); % backwards coordinates!!!!

band = logical(bandFocus); % logical for masking

% adjust color & add to cumulative image

bandFocus = im2uint8(bandFocus); % for imaging

bandFocus2 = bandFocus/255;

bandsFinal = bandsFinal + bandFocus2\*(255-15\*i); % create any number of bands

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% MEASUREMENTS

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% FN alignment to the right (0 degrees) and length

clearvars branchBand fibBand cellBand actinBand ...

CCf sf c of oof lf bwBand bwEmpty

branchBand = br;

branchBand(band==0) = 0;

fibBand = fib;

fibBand(band==0) = 0;

cellBand = bwB;

cellBand(band==0) = 0;

actinBand = bwR;

actinBand(band==0) = 0;

bwEmpty = imcomplement(bw);

bwEmpty(bwR==0) = 0;

bwEmpty(band==0) = 0;

bwBand = bw;

bwBand(band==0) = 0;

CCf = bwconncomp(fibBand);

sf = regionprops(CCf,'Orientation','Area');

c = bweuler(cellBand);

of = abs(cat(1,sf.Orientation));

oof = cat(1,sf.Orientation);

lf = cat(1,sf.Area);

% FN alignment to the right (0 degrees) and length

clearvars CCa sa oa la bwABand

actinBand = bwR;

actinBand(band==0) = 0;

fibABand = fibA;

fibABand(band==0) = 0;

bwABand = bwA+bwB;

bwABand(band==0) = 0;

CCa = bwconncomp(fibABand);

sa = regionprops(CCa,'Orientation','Area');

oa = abs(cat(1,sa.Orientation));

la = cat(1,sa.Area);

% store length and orientation of each fiber for distribution

% statistics

if ( numel(of) == 0 )

of = 0;

end

if ( numel(oof) == 0 )

oof = 0;

end

if ( numel(lf) == 0 )

lf = 0;

end

calc0{i,1,nCount} = abs(sort(oof,'ascend')); % absolute value angle of [-90,90] in order

calc0{i,2,nCount} = sort(oof,'ascend'); % original angle of [-90,90] in order

calc0{i,3,nCount} = sort(of,'ascend'); % absolute value angle of [0,90] in order

calc0{i,4,nCount} = sort(lf,'ascend'); % length

calc0im{nCount,1} = cat(1,calc0im{nCount,1},sort(oof,'ascend'));

calc0im{nCount,2} = cat(1,calc0im{nCount,2},sort(oof,'ascend'));

calc0im{nCount,3} = cat(1,calc0im{nCount,3},sort(of,'ascend'));

calc0im{nCount,4} = cat(1,calc0im{nCount,4},sort(lf,'ascend'));

calc0rawIm{nCount,1} = cat(1,calc0rawIm{nCount,1},oof);

calc0rawIm{nCount,2} = cat(1,calc0rawIm{nCount,2},lf);

% general stats

calc(i,1,nCount) = c; % number of cells

calc(i,2,nCount) = sum(sum(actinBand)); % cell area

calc(i,3,nCount) = numel(of); % number of FN fibers

calc(i,4,nCount) = mean(of); % average FN fiber orientation

calc(i,5,nCount) = mean(lf); % average FN fiber length

calc(i,6,nCount) = numel(oa); % number of Actin fibers

calc(i,7,nCount) = mean(oa); % average Actin fiber orientation

calc(i,8,nCount) = mean(la); % average Actin fiber length

calc(i,9,nCount) = sum(sum(bwBand))/sum(sum(bwABand)); % ratio coverage of actin

calc(i,10,nCount) = sum(sum(bwEmpty))/sum(sum(bwABand)); % ratio of empty space in Fn among actin

% length bins

calc2(i,1,nCount) = numel(lf(lf<=5));

calc2(i,2,nCount) = numel(lf(lf>5&lf<=10));

calc2(i,3,nCount) = numel(lf(lf>10&lf<=15));

calc2(i,4,nCount) = numel(lf(lf>15&lf<=20));

calc2(i,5,nCount) = numel(lf(lf>20&lf<=25));

calc2(i,6,nCount) = numel(lf(lf>25));

calc2(i,7,nCount) = numel(lf(lf<=5))\*mean(lf(lf<=5));

calc2(i,8,nCount) = numel(lf(lf>5&lf<=10))\*mean(lf(lf>5&lf<=10));

calc2(i,9,nCount) = numel(lf(lf>10&lf<=15))\*mean(lf(lf>10&lf<=15));

calc2(i,10,nCount) = numel(lf(lf>15&lf<=20))\*mean(lf(lf>15&lf<=20));

calc2(i,11,nCount) = numel(lf(lf>20&lf<=25))\*mean(lf(lf>20&lf<=25));

calc2(i,12,nCount) = numel(lf(lf>25))\*mean(lf(lf>25));

calc2(i,13,nCount) = sum(sum(branchBand)); % number of branch points

% orientation bins

calc3(i,1,nCount) = numel(of(of<=20));

calc3(i,2,nCount) = numel(of(of>20&of<=45));

calc3(i,3,nCount) = numel(of(of>45&of<=60));

% long orientation bins

calc4(i,1,nCount) = mean(of(lf>5));

calc4(i,2,nCount) = mean(of(lf>10));

calc4(i,3,nCount) = mean(of(lf>20));

calc4(i,4,nCount) = numel(of(lf>5&of<=20));

calc4(i,5,nCount) = numel(of(lf>5&of>20&of<=45));

calc4(i,6,nCount) = numel(of(lf>5&of>45&of<=60));

calc4(i,7,nCount) = numel(of(lf>5&of<=20))/numel(of);

calc4(i,8,nCount) = numel(of(lf>5&of>20&of<=45))/numel(of);

calc4(i,9,nCount) = numel(of(lf>5&of>45&of<=60))/numel(of);

calc4(i,10,nCount) = numel(of(lf>5&of<=45))/numel(of);

fprintf('DONEZO!\r');

% final stored parameters

storedNames{nCount} = name;

end

if ( rosePlots )

clearvars bins

bins = 40;

clearvars roseAngles x y g h p

roseAngles = calc0rawIm{nCount,1}(calc0rawIm{nCount,2}>=10);

roseAngles(roseAngles<0) = roseAngles(roseAngles<0)+360;

roseAngles = deg2rad(roseAngles);

h = figure; set(h,'color','w');

g = rose(roseAngles,bins); hold on;

x = get(g,'XData');

y = get(g,'YData');

% convert 1 row vector into columns (# bins), each with 4 points bounding the polygon

polygonPts = 4;

x = reshape(x,polygonPts,numel(x)/polygonPts);

y = reshape(y,polygonPts,numel(y)/polygonPts);

p = patch(x,y,[0 1 0]); % change color

axis tight;

set(gcf,'Position',[600 300 225 410]);

hold off;

if ( saveIms )

saveas(h,fullfile(outputFolder3,[name '-rose' '.bmp']),'bmp');

end

delete(h);

end

% output images

clearvars ovBinFib ovBinBW ovFib ovFibA ovFibAFib

ovBinFib = cat(3,fib+ring0,fib,fib+ringRest);

ovBinBW= cat(3,bw+ring0,bw,bw+ringRest);

ovFib = cat(3,im2double(green),im2double(green)+fib,im2double(green));

ovFibA = cat(3,im2double(red)+fibA,im2double(red),im2double(red));

ovFibAFib = cat(3,fibA,fib,zeros(size(fib)));

clearvars rr gg bb orig

bb = imadjust(blue,[0,0.5],[],1);

rr = imadjust(red,[0,0.5],[],1);

gg = imadjust(green,[0,0.7],[],1);

if ( interface )

rr(bwB==1) = 0;

gg(bwB==1) = 0;

end

orig = cat(3,im2double(rr),im2double(gg),im2double(bb));

chanFN = green;

if ( interface )

chanFN(bwR==0) = 0;

end

% save images

if ( saveIms )

imwrite(orig,fullfile(outputFolder1,[name sprintf('-output-1-combo') '.bmp']), 'bmp');

imwrite(chanFN,fullfile(outputFolder1,[name sprintf('-output-2-FN') '.bmp']), 'bmp');

imwrite(ovBinBW,fullfile(outputFolder1,[name sprintf('-output-3-FnBin') '.bmp']), 'bmp');

imwrite(ovBinFib,fullfile(outputFolder1,[name sprintf('-output-4-FnFibBin') '.bmp']), 'bmp');

imwrite(ovFib,fullfile(outputFolder1,[name sprintf('-output-5-FnFibOverlay') '.bmp']), 'bmp');

imwrite(ovFibA,fullfile(outputFolder1,[name sprintf('-output-6-ActinFibOverlay') '.bmp']), 'bmp');

imwrite(ovFibAFib,fullfile(outputFolder1,[name sprintf('-output-7-FnActinOverlap') '.bmp']), 'bmp');

imwrite(fibClassifiedThick,fullfile(outputFolder1,[name sprintf('-output-8-fibers') '.bmp']), 'bmp');

imwrite(fibClassifiedOverlay,fullfile(outputFolder1,[name sprintf('-output-9-fibersOverlay') '.bmp']), 'bmp');

imwrite(fibClassifiedOverlayR,fullfile(outputFolder1,[name sprintf('-output-10-fibersOverlayR') '.bmp']), 'bmp');

imwrite(fibClassifiedOverlayY,fullfile(outputFolder1,[name sprintf('-output-11-fibersOverlayY') '.bmp']), 'bmp');

imwrite(fibClassifiedOverlayG,fullfile(outputFolder1,[name sprintf('-output-12-fibersOverlayG') '.bmp']), 'bmp');

else

fprintf('NO EXCEL FILE GENERATED.\r');

end

minut = floor(toc(t)/60); sec = mod(toc(t),60);

fprintf('IMAGE #%i of %i COMPLETE. [%0.2f min., %0.2f s]\r\r',nCount,num/3,minut,sec);

nCount = nCount + 1;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% PLOTS

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if ( statSave )

% general settings

set(0,'DefaultAxesFontSize',14,'DefaultTextFontSize',14);

% plot histograms

shape = ['o','+','s','^'];

color = ['k','b','r','g','m','c','y'];

type = {'-',':','--','-.','-',':','--','-.'};

style = [];

for i = 1:numel(groups)

style = [style;repmat(i,1,groups(i))',repmat(1:groups(i),1,1)'];

end

clearvars normality kurt stdev variance avg skew h

normality = zeros(size(calc0im,1),2);

kurt = zeros(size(calc0im,1),1);

stdev = zeros(size(calc0im,1),1);

numCell = zeros(size(calc0im,1),1);

sem = zeros(size(calc0im,1),1);

variance = zeros(size(calc0im,1),1);

avg = zeros(size(calc0im,1),1);

skew = zeros(size(calc0im,1),1);

for i = 1:size(calc0im,1)

clearvars hc

hc = histc(calc0im{i,2},-90:10:90);

[normality(i,1),normality(i,2)] = kstest(calc0im{i,1});

kurt(i) = kurtosis(calc0im{i,2});

stdev(i) = std(calc0im{i,2});

numCell(i) = numel(calc0im{i,1});

sem(i) = stdev(i)/numCell(i)^0.5;

variance(i) = var(calc0im{i,1});

avg(i) = mean(calc0im{i,1});

skew(i) = skewness(calc0im{i,2}); % neg = longer left tail, pos = longer right tail

end

c = 1;

cc = [1 0 0; 0 1 0; 0 0 1;1 0 0; 0 1 0; 0 0 1];

for ii = 1:numel(groups)

for jj = 1:groups(ii)

colmap(c,:) = cc(ii,:);

c = c+1;

end

end

maxHist = 0;

for i = 1:size(calc0im,1)

clearvars hc

hc = histc(calc0im{i,2},-90:10:90);

if ( max(hc) > maxHist )

maxHist = max(hc);

end

end

maxHist = round(maxHist,2);

if ( maxHist == 0 ) % therefore, no frequency higher than 50

maxHist = 50;

end

% individual

h2sub = figure('units','normalized','outerposition',[0 0 1 1]);

ha2sub = axes;

h2subNorm = figure('units','normalized','outerposition',[0 0 1 1]);

ha2subNorm = axes;

h2subLin = figure('units','normalized','outerposition',[0 0 1 1]);

ha2subLin = axes;

hLengthHist = figure('units','normalized','outerposition',[0 0 1 1]);

haLengthHist = axes;

hAngleHistShort10 = figure('units','normalized','outerposition',[0 0 1 1]);

haAngleHistShort10 = axes;

hAngleHistLong10 = figure('units','normalized','outerposition',[0 0 1 1]);

haAngleHistLong10 = axes;

hAngleHistShort3 = figure('units','normalized','outerposition',[0 0 1 1]);

haAngleHistShort3 = axes;

hAngleHistLong3 = figure('units','normalized','outerposition',[0 0 1 1]);

haAngleHistLong3 = axes;

hAngleHistAll = figure('units','normalized','outerposition',[0 0 1 1]);

haAngleHistAll = axes;

hLengthHist10 = figure('units','normalized','outerposition',[0 0 1 1]);

haLengthHist10 = axes;

for i = 1:size(calc0im,1)

clearvars hc

hc = histc(calc0im{i,2},-90:10:90);

figure(h2sub); hold on;

sh2 = subplot(RR,CC,gridOrd(i));

bar(-90:10:90,hc,1,'LineWidth',2);

ylim([0,maxHist]);

set(sh2,'FontSize',12);

set(sh2,'XTick',-90:30:90);

title(names0(i),'FontSize',12);

hold off;

clearvars hcNorm

hcNorm = hc/sum(hc);

figure(h2subNorm); hold on;

sh3 = subplot(RR,CC,gridOrd(i));

bar(-90:10:90,hcNorm,1,'LineWidth',2);

ylim([0,0.4]);

set(sh3,'FontSize',12);

set(sh3,'XTick',-90:30:90);

title(names0(i),'FontSize',12);

hold off;

clearvars hc2 xinterp yinterp

hc2x = 0:10:90;

hc2 = histc(calc0im{i,1},hc2x);

hc2 = hc2/sum(hc2);

xinterp = 0:0.1:90;

yinterp = interp1(hc2x,hc2,xinterp);

figure(h2subLin); hold on;

sh4 = subplot(RR,CC,gridOrd(i));

plot(0:10:90,hc2,'ko','LineWidth',2); hold on;

plot(xinterp,yinterp,'r');

clearvars F0 G

[F0,G] = fit(hc2x',hc2,'exp1');

plot(F0,'b');

ylim([0,0.6]); set(sh4,'XTick',0:30:90);

set(sh4,'FontSize',12);

title(names0(i),'FontSize',12);

hold off;

decays(i) = F0.b;

rSqFit(i) = G.rsquare;

% lengths ALL

clearvars hc4x hc4

hc4x = 0:5:100;

hc4 = histc(calc0im{i,4},hc4x);

figure(hLengthHist); hold on;

sh4 = subplot(RR,CC,gridOrd(i));

bar(hc4x,hc4,1,'LineWidth',2);

ylim([0,5500]);

xlim([-5,105]);

set(sh4,'FontSize',12);

set(sh4,'XTick',0:20:100);

title(names0(i),'FontSize',12);

hold off;

% Orientation of lengths < 10 pixels

clearvars hc9x hc9

hc9x = -90:10:90;

hc9 = histc(calc0rawIm{i,1}(calc0rawIm{i,2}<10),hc9x);

figure(hAngleHistShort10); hold on;

sh9 = subplot(RR,CC,gridOrd(i));

bar(hc9x,hc9,1,'LineWidth',2);

ylim([0,4000]);

set(sh9,'FontSize',12);

set(sh9,'XTick',[-90 -45 0 45 90]);

title(names0(i),'FontSize',12);

hold off;

% Orientation of lengths >= 10 pixels

clearvars hc6x hc6

hc8x = -90:10:90;

hc8 = histc(calc0rawIm{i,1}(calc0rawIm{i,2}>=10),hc8x);

figure(hAngleHistLong10); hold on;

sh8 = subplot(RR,CC,gridOrd(i));

bar(hc8x,hc8,1,'LineWidth',2);

ylim([0,500]);

set(sh8,'FontSize',12);

set(sh8,'XTick',[-90 -45 0 45 90]);

title(names0(i),'FontSize',12);

hold off;

% Orientation of lengths < 3 pixels

clearvars hc12x hc12

hc12x = -90:10:90;

hc12 = histc(calc0rawIm{i,1}(calc0rawIm{i,2}<3),hc12x);

figure(hAngleHistShort3); hold on;

sh12 = subplot(RR,CC,gridOrd(i));

bar(hc12x,hc12,1,'LineWidth',2);

ylim([0,2500]);

set(sh12,'FontSize',12);

set(sh12,'XTick',[-90 -45 0 45 90]);

title(names0(i),'FontSize',12);

hold off;

% Orientation of lengths >= 3 pixels

clearvars hc13x hc13

hc13x = -90:10:90;

hc13 = histc(calc0rawIm{i,1}(calc0rawIm{i,2}>=3),hc13x);

figure(hAngleHistLong3); hold on;

sh13 = subplot(RR,CC,gridOrd(i));

bar(hc13x,hc13,1,'LineWidth',2);

ylim([0,1550]);

set(sh13,'FontSize',12);

set(sh13,'XTick',[-90 -45 0 45 90]);

title(names0(i),'FontSize',12);

hold off;

% Orientation of all lengths (raw)

clearvars hc7x hc7

hc7x = -90:10:90;

hc7 = histc(calc0rawIm{i,1},hc7x);

figure(hAngleHistAll); hold on;

sh7 = subplot(RR,CC,gridOrd(i));

bar(hc7x,hc7,1,'LineWidth',2);

ylim([0,4000]);

set(sh7,'FontSize',12);

set(sh7,'XTick',[-90 -45 0 45 90]);

title(names0(i),'FontSize',12);

hold off;

% lengths >= 10

clearvars hc4c5

hc4c10 = histc(calc0im{i,4}(calc0im{i,4}>=10),hc4x);

figure(hLengthHist10); hold on;

sh4c10 = subplot(RR,CC,gridOrd(i));

bar(hc4x,hc4c10,1,'LineWidth',2);

ylim([0,900]);

xlim([-5,105]);

set(sh4c10,'FontSize',12);

set(sh4c10,'XTick',0:20:100);

title(names0(i),'FontSize',12);

hold off;

clearvars len10 ang10

len10 = calc0im{i,4}(calc0im{i,4}>=10); %\*0.2287;

ang10 = calc0rawIm{i,1}(calc0rawIm{i,2}>=10);

len3 = calc0im{i,4}(calc0im{i,4}>=3);

% length

calcLen{i,1} = mean(len10); % avg

calcLen{i,2} = std(len10)/numel(len10)^0.5; % sem

calcLen{i,3} = std(len10); % std

calcLen{i,4} = kurtosis(len10); % kurtosis

calcLen{i,5} = histc(len10,0:1:100); % bins of every 2 length

calcLen{i,6} = [sum(len10(len10>10)),sum(len10(len10>20)),sum(len10(len10>30)), ...

sum(len10(len10>40)),sum(len10(len10>50)),sum(len10(len10>60)), ...

sum(len10(len10>70)),sum(len10(len10>80)),sum(len10(len10>90))];

calcLen{i,7} = numel(len10)/numel(len3);

% angles

calcAng{i,1} = mean(abs(ang10)); % avg

calcAng{i,2} = std(ang10)/numel(ang10)^0.5; % sem

calcAng{i,3} = std(ang10); % std

calcAng{i,4} = kurtosis(ang10); % kurtosis

calcAng{i,5} = numel(ang10(abs(ang10)<=20))/numel(ang10); % percent of total less than 20 degrees

calcAng{i,6} = numel(ang10);

end

fprintf('WRITING EXCEL...');

head = {'File','Mean','SEM','STD','Kurtosis','% < 20 degrees','Num Fibers'};

xlswrite(fullfile(outputFolder2,'output-AngLen.xlsx'),head,1,'A1');

xlswrite(fullfile(outputFolder2,'output-AngLen.xlsx'),names',1,'A2');

xlswrite(fullfile(outputFolder2,'output-AngLen.xlsx'),calcAng,1,'B2');

head = {'File','Length (% of 10+ out of all above 3'};

xlswrite(fullfile(outputFolder2,'output-AngLen.xlsx'),head,2,'A1');

xlswrite(fullfile(outputFolder2,'output-AngLen.xlsx'),names',2,'A2');

xlswrite(fullfile(outputFolder2,'output-AngLen.xlsx'),calcLen(:,7),2,'B2');

fprintf('DONE!\n');

export\_fig(fullfile(outputFolder2,'stats-TILE-histogram.bmp' ),h2sub);

close(h2sub);

export\_fig(fullfile(outputFolder2,'stats-TILE-histogram\_NORM.bmp' ),h2subNorm);

close(h2subNorm);

export\_fig(fullfile(outputFolder2,'stats-TILE-histogram\_DEVIATIONS.bmp' ),h2subLin);

close(h2subLin);

export\_fig(fullfile(outputFolder2,'stats-TILE-length\_histogram.bmp' ),hLengthHist);

close(hLengthHist);

export\_fig(fullfile(outputFolder2,'stats-TILE-angle\_histogram\_short10.bmp' ),hAngleHistShort10);

close(hAngleHistShort10);

export\_fig(fullfile(outputFolder2,'stats-TILE-angle\_histogram\_long10.bmp' ),hAngleHistLong10);

close(hAngleHistLong10);

export\_fig(fullfile(outputFolder2,'stats-TILE-angle\_histogram\_short3.bmp' ),hAngleHistShort3);

close(hAngleHistShort3);

export\_fig(fullfile(outputFolder2,'stats-TILE-angle\_histogram\_long3.bmp' ),hAngleHistLong3);

close(hAngleHistLong3);

export\_fig(fullfile(outputFolder2,'stats-TILE-angle\_histogram\_all.bmp' ),hAngleHistAll);

close(hAngleHistAll);

export\_fig(fullfile(outputFolder2,'stats-TILE-length\_histogram\_10.bmp' ),hLengthHist10);

close(hLengthHist10);

b6 = figure('units','normalized','outerposition',[0 0 1 1]); hold on;

ylabel('Lambda (Exponential Fit Parameter)');

legend(names0);

for ii = 1:numel(decays)

bb = bar(ii,decays(ii),1,'LineWidth',2);

set(bb,'FaceColor',colmap(ii,:));

end

set(gca,'XTick',1:numel(decays));

xlim([0 numel(decays)+1]);

set(gca,'XTickLabel',names0);

export\_fig(fullfile(outputFolder2,'stats-ALL-Lambda.bmp'),b6);

close(b6);

b7 = figure('units','normalized','outerposition',[0 0 1 1]); hold on;

ylabel('R^2 (Exponential Fit)');

legend(names0);

for ii = 1:numel(rSqFit)

bb = bar(ii,rSqFit(ii),1,'LineWidth',2);

set(bb,'FaceColor',colmap(ii,:));

end

set(gca,'XTick',1:numel(rSqFit));

xlim([0 numel(rSqFit)+1]);

set(gca,'XTickLabel',names0);

export\_fig(fullfile(outputFolder2,'stats-ALL-R2.bmp'),b7);

close(b7);

end

minut = floor(toc(t0)/60); sec = mod(toc(t0),60);

fprintf('Done with directory. [%0.2f min., %0.2f s]\r',minut,sec);

end