```
//Image processing macro for ImageJ (NIH) to analyze images of paper-based channel invasion
/////////////////////////////////////////////////////////////////////////////////////////////////////
//Definitions:
// 1) Distance markers: wax-printed distance markers on the top and bottom of the
//        channel at 3.0, 4.0, 5.5, 8.0 and 9.0 mm (Figure 2A). These markers are fluorescent.
// 2) Orientation marker: wax-printed marker on the right end of the channel. This marker spans the
//        width of the channel (Figure 2A) and is fluorescent.
//
//This macro will:
// 1) Allow the user to select a root folder with images corresponding to multiple time points of the
//        same channel.
// 2) Open the first image in the root folder (time point = 0 h), and convert it into a 16-bit image if its bit
//        depth is greater than 16-bit.
//3) Prompt the user to make an initial line selection. This line should underline the upper distance
//        markers running along the length of the channel. The image is then rotated based on the slope
//        of this line.
// 3) Find the channel orientation marker, and use it to build a "bounding box" within the channel,
//        spanning from the 3.0 mm distance markers to the 9.0 mm distance markers. The upper y-
//        coordinate border is defined by the user-drawn line (half the vertical distance between the line's
//        endpoints) and spans the width of channel, 1300 pixels down. The x-coordinate of
//        the left border is a constant 5190 pixels less than the orientation marker. The x-coordinate of
//        the right border is a constant 1810 pixels less than the orientation marker.
//        (Note: This macro is made for 7392 x 1946 pixel images; constants will need to be adjusted if
//        the image resolution changes.)
// 4) Average all y-pixels, for each x-pixel, across the length of the bounding box, and construct an array
//        with these values.
// 5) Find the maximum value in the array and use 70% of this value to serve as a cutoff to determine
//        the location of the bulk cell population. To do this, the array values at the leftmost and
//        rightmost x-pixel are compared to 70% max value, and if these values are smaller, the macro will
//        increment one pixel towards the center of the bounding box until the x-pixel exceeds 70% of the
//        max value.
//        (Note: We chose 70% of the max array value, because cells are seeded in a square-shaped
//        distribution with spatial noise across the plateau. To ensure that that we capture the bulk of the
//        seeded cells but do not pick an artificially high cutoff due to noise, we allotted a
//        30% signal variation between max averaged value across the channel and the bulk of seeded
//        cells.)
// 6) Use the left and right 70% cutoff pixels to find the average intensity within the two newly defined
//        cutoff boundaries.
// 7) Use the average intensity value found in the seeded region to perform another incremental
//        comparison within the original bounding box, comparing the average seeded region intensity to
//        the averaged y-pixels per x-pixel. This will determine the cell front on either side of the seeded
//        region. The comparison will increment towards the center of the bounding box, until the
//        average seeded signal is exceeded, establishing two new cutoff coordinates (L-cutoff, and R-
//        cutoff) which will serve as boundaries for building rectangular selections for center of mass
//        measurements.
// 8) Create one rectangular selection using the bounding box's left boundary and the L-cutoff x-
//        coordinates, and another selection using the R-cutoff and bounding box's right boundary x-
//        coordinates. The upper y-pixel boundary is determined by the original line selection, and
```

```
//	extends 1300 pixels down, corresponding to the width of the channel.
//	(Note: The pixel difference between boundary x-coordinates and the orientation marker is
//	recorded, and will be used to construct rectangular selections for center of mass measurements
//	in the subsequent images.)
// 9) Measure the center of mass in each rectangular selection, and save these values for an end-
//	readout.
// 10) Close the original image, and open the subsequent image in the root folder.
// 11) Prompt the user to create a line selection, rotate the image, and find the orientation marker.
// 12) Create two rectangular selections of the same dimensions as the first image analyzed. The
//	location of these selections is determined by the difference in pixels between the first image's
//	rectangular selection boundaries and orientation marker.
// 13) Close the image, and sequentially open the remaining images, repeating steps 10-12 for the
//	remaining files.
//////////////////////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////////////////////
/////Select root file directory
// 1) User defines root folder.
// 2) Acquires list of files to be processed in root folder.
// 3) Opens first file.
//4) Converts file to 16-bit if file is greater than 16-bit.
// 5) Acquires image height and width in pixels.
output = getDirectory("Select root file with images to process...")
fileList = getFileList(output);
open(fileList[0]);
if(bitDepth>16)
	run("16-bit");
imageHeight = getHeight();
imageWidth = getWidth();

//////////////////////////////////////////////////////////////////////////////////////////////////
/////Boundary arrays
//All arrays to be used for boundary creation and end-readout.
BoundaryValue = newArray(fileList.length);
LeftBoundary = newArray(fileList.length);
RightBoundary = newArray(fileList.length);
LeftBoundaryXM = newArray(fileList.length);
LeftBoundaryYM = newArray(fileList.length);
RightBoundaryXM = newArray(fileList.length);
RightBoundaryYM = newArray(fileList.length);
YValues = newArray(1300);
//1300 represents the distance, in pixels, across the width of the channel
XValueAvg = newArray(3380);
//3380 represents the distance, in pixels, across the length of the channel between left- and rightmost
//	channel distance markers.

//////////////////////////////////////////////////////////////////////////////////////////////////
/////Orientation Arrays
```

```
// All arrays to be used for finding the orientation marker
OrientationXValue = newArray(fileList.length);
OrientationCheck = newArray(1300);
OrientationCheckXValue = newArray(1000);
OrientationPixel = newArray(fileList.length);

///////////////////////////////////////////////////////////////////////////////////////////////
/////Rotate image
// 1) Prompts user to draw a line selection under the top channel length markers.
// 2) Uses this line selection to rotate the image.
waitForUser("Draw line selection", "Draw line under top-most channel distance markers to correct for image rotation:\nOnce done, click 'OK'");
getLine(Lx1, Ly1, Lx2, Ly2, Lw);
if (Ly2 > Ly1) {ay = Ly2;}
	else {ay = Ly1;};
dLx = Lx2 - Lx1;
dLy = Ly2 - Ly1;
hyp = dLx;
Angle = tan(dLy/dLx);
radAngle = -Angle*(180/3.145);
run("Rotate...", "angle=" + radAngle + " grid=1 interpolation=Bilinear");

///////////////////////////////////////////////////////////////////////////////////////////////
/////Find rightmost orientation marker
// 1) Calculates the average intensity of 1300 y-pixels per x-pixel of the 1000 rightmost x-pixels.
// 2) Finds the average intensity value of the 100 right most pixels, which serves as a background
//	intensity.
// 3) Compares the background intensity to the average intensities across the 1000 rightmost x-pixels.
//	The orientation marker is found when the average intensity is greater than 1.5*background
//	intensity.
// 4) From this orientation marker, a bounding box is made with a left border at orientation marker -
//	5190 pixels, and a right border of orientation marker - 1810.
// 5) The upper y-boundary is the equidistant y-pixel between the original line selection's end points.
for(i=0; i<1000; i++) {
	for(j=0; j < 1300; j++){
		if(j==0) {
			OrientationSum = 0;
		};
		OrientationCheck[j] = getPixel(i+imageWidth-1000, j+Ly1+(dLy/2));
		OrientationSum = OrientationSum + OrientationCheck[j];
		OrientationAvg = OrientationSum/1300;
	};
	OrientationCheckXValue[i] = OrientationAvg;
};
for(i=999; i>899; i--) {
	if(i==999) {
		OrientationCheckXSum = 0;
	};
```

```
	OrientationCheckXSum = OrientationCheckXSum + OrientationCheckXValue[i];
};
AvgReferenceValue = 1.5*OrientationCheckXSum/(100);
Check = 1000;
while(OrientationCheckXValue[Check-1] < AvgReferenceValue) {
	Check--;
};
LeftLimitPixel = imageWidth - (5190+(1000-Check));
RightLimitPixel = imageWidth - (1810+(1000-Check));
makeRectangle(LeftLimitPixel, Ly1+(dLy/2), RightLimitPixel-LeftLimitPixel, 1300);
OrientationXValue[0] = Check;

//////////////////////////////////////////////////////////////////////////////////////////////////////
////Set boundary limitations across channel
// 1) Using the x-pixel boundaries outlined in the previous section, the average intensity across 1300 y-
//	pixels are measured per x-pixel, and sorted in an array.
// 2) Starting at the left border, the averaged y-pixels per x-pixel are compared to 70% of the maximum
//	value in the newly created array. The x-pixel is incremented one unit until it exceeds 70% of the
//	maximum value.
// 3) This process is repeated for the right boundary.
// 4) The right and left boundaries are saved in an array for the final readout, and the average of the
//	averaged y-pixels is found within the 70% cutoff region.
// 5) This average value is then checked against the averaged y-pixels in one pixel increments from left
//	to right and from right to left, establishing the L-cutoff and R-cutoff respectively. These edges
//	are used to define new rectangle selections which enclose the cell front on either side of the
//	seeded cell population.


for(i=0; i<RightLimitPixel-LeftLimitPixel; i++) {
	for(j=0; j < 1300; j++) {
		if(j==0) {
			Sum = 0;
		};
		YValues[j] = getPixel(i+LeftLimitPixel,j+Ly1+dLy/2);
		Sum = Sum + YValues[j];
	};
	XValueAvg[i] = Sum/(1300);
};
Array.getStatistics(XValueAvg, min, max, mean, stdDev);
k = 0;
while(XValueAvg[k] < max*0.7) {
	k++;
};
l = RightLimitPixel-LeftLimitPixel;
while(XValueAvg[l-1] < max*0.7) {
	l--;
};
AvgCutOff = 0;
```

```
for( i=k; i<l; i++) {
	AvgCutOff = (AvgCutOff + XValueAvg[i]);
};
AvgCutOff = AvgCutOff/(l-k);
m = 0;
while(XValueAvg[m] < AvgCutOff) {
	m++;
};
n = RightLimitPixel-LeftLimitPixel;
while(XValueAvg[n-1] < AvgCutOff) {
	n--;
};
LeftBoundaryPixel = m + LeftLimitPixel;
RightBoundaryPixel = n + LeftLimitPixel;
LeftBoundaryWidth = LeftBoundaryPixel-LeftLimitPixel;
RightBoundaryWidth = RightLimitPixel - RightBoundaryPixel;
OrientationPixel[0] = imageWidth - (1000-Check);
LeftBoundary[0] = LeftBoundaryPixel;
RightBoundary[0] = RightBoundaryPixel;

////////////////////////////////////////////////////////////////////////////////////////////////////
////Measure center of mass
// 1) To remove the impact of background signal on the center of mass measurement, a rolling ball
//	background subtraction is done.
// 2) A selection rectangle is made using the left most bounding box x-pixel, stopping at the L-cutoff. The
//	center of mass is then measured.
// 3) Another selection rectangle is made using the right most border x-pixel, stopping at the R-cutoff.
//	The center of mass is then measured.
// 4) The center of mass values are saved in an array for post-analysis readout.

run("Subtract Background...", "rolling=100");
makeRectangle(LeftLimitPixel, Ly1+(dLy/2), m, 1300);
run("Measure");
LeftBoundaryXM[0] = getResult("XM", 0); LeftBoundaryYM[0] = getResult("YM", 0);
makeRectangle(RightBoundaryPixel, Ly1+(dLy/2),RightBoundaryWidth, 1300);
run("Measure");
RightBoundaryXM[0] = getResult("XM", 1); RightBoundaryYM[0] = getResult("YM", 1);
close();

////////////////////////////////////////////////////////////////////////////////////////////////////
//Process remaining images
// 1) Opens second file in the root folder to be processed.
// 2) User will be prompted to draw a line selection to orientate the image for further analysis.
// 3) After analysis, the image will be closed, and the proceeding image will be opened.

for(file=1; file < fileList.length; file++) {
	open(fileList[file]);
	if(bitDepth==24)
```

```
            run("16-bit");

////////////////////////////////////////////////////////////////////////////////////////////////////
/////Rotate image
//Same function as in previous section
waitForUser("Draw line selection", "Draw line under top-most channel distance markers to correct for
image rotation:\nOnce done, click 'OK'");

        getLine(Lx1, Ly1, Lx2, Ly2, Lw);
        if (Ly2 > Ly1) {ay = Ly2;}
            else {ay = Ly1;};
        h = 1100;
        dLx = Lx2 - Lx1;
        dLy = Ly2 - Ly1;
        hyp = dLx;
        Angle = tan(dLy/dLx);
        radAngle = -Angle*(180/3.145);
        run("Rotate...", "angle=" + radAngle + " grid=1 interpolation=Bilinear");

////////////////////////////////////////////////////////////////////////////////////////////////////
/////Orientate image
//Same function as previous orientation section, but uses distances from the orientation marker
//      from the first image to reproduce those center of mass bounding boxes.
        Check = 1000;
        for(i=0; i<1000; i++) {
                    for(j=0; j < 1300; j++){
                    if(j==0)  {
                        OrientationSum = 0;
                    };
                    OrientationCheck[j] = getPixel(i+imageWidth-1000,j+Ly1+(dLy/2));
                    OrientationSum = OrientationSum + OrientationCheck[j];
                    OrientationAvg = OrientationSum/1300;
            };
            OrientationCheckXValue[i] = OrientationAvg;
        };
        for(i=999; i>899; i--) {
            if(i==999)  {
                        OrientationCheckXSum = 0;
                    };
            OrientationCheckXSum = OrientationCheckXSum + OrientationCheckXValue[i];
        };
        AvgReferenceValue = 1.5*OrientationCheckXSum/(100);
        while(OrientationCheckXValue[Check-1] < AvgReferenceValue) {
            Check--;
        };
OrientationPixel[file] = imageWidth - (1000-Check) ;
LeftLimitPixel = imageWidth - (5190+(1000-Check));
RightLimitPixel = imageWidth - (1860+(1000-Check));
```

```
LeftBoundary[file] = LeftLimitPixel + LeftBoundaryWidth;
RightBoundary[file] = RightLimitPixel - RightBoundaryWidth;

/////////////////////////////////////////////////////////////////////////////////////////////////////
////Measure center of mass
//Same function as previous section
	run("Subtract Background...", "rolling=100");
	makeRectangle(LeftLimitPixel, Ly1+(dLy/2), LeftBoundaryWidth, 1300);
	run("Measure");
	LeftBoundaryXM[file] = getResult("XM", file*2); LeftBoundaryYM[file] = getResult("YM", file*2);
	makeRectangle(RightLimitPixel-RightBoundaryWidth, Ly1+(dLy/2), RightBoundaryWidth, 1300);
	run("Measure");
	RightBoundaryXM[file] = getResult("XM", file*2+1); RightBoundaryYM[file] = getResult("YM",
file*2+1);
close();
};

/////////////////////////////////////////////////////////////////////////////////////////////////////
/////Results readout
//Produces a readout of the x-pixels for the orientation marker, left and right cutoff pixels (averaged
//	seeding cutoff), and center of mass pixels.
run("Clear Results");
for (i=0; i<fileList.length; i++) {
	setResult("Orientation Pixel", i, OrientationPixel[i]);
	setResult("Left Boundary pixel", i, LeftBoundary[i]);
	setResult("Right Boundary pixel", i, RightBoundary[i]);
	setResult("Left XM", i, LeftBoundaryXM[i]);
	setResult("Left YM",i, LeftBoundaryYM[i]);
	setResult("Right XM", i, RightBoundaryXM[i]);
	setResult("Right YM",i, RightBoundaryYM[i]);
};
updateResults;

/////////////////////////////////////////////////////////////////////////////////////////////////////
////Data analysis
//The final readout provides the x-coordinates for the orientation marker, left and right boundary, and
//	left and right center of mass values (Left XM and Right XM, respectively).
//To analyze this data, the x-coordinates must first be corrected for horizontal shifts between images.
//	This is done by taking the difference between the orientation marker coordinate at time = 0 h
//	and the other time points.
//	Ex: Shift_24h = Orientation_0h – Orientation_24h
//This shift is then added to the XM values. These corrected XM values are then compared to one
//	another to derive change in center of mass as a function of time.
```