```
// This macro allows the user to straighten and average across the channel
// culture to generate plot profiles of each fluorescence image to Excel.
// These can be further processed in Excel to form profiles.
// The plugin used to write to the Excel files can be found here:
// https://github.com/bkromhout/Read_and_Write_Excel_Modified
////////////////////////////////////////////////////////////////////////////////////////////////
// At a high level, this macro will:
// 1) Straighten every image in the directory according to a user-drawn line
// 2) Average every y-pixel for every x-pixel across the length of the channel
// 3) Output the data into an excel spreadsheet, organized by holder and channel

////////////////////////////////////////////////////////////////////////////////////////////////
//// Make sure everything is clear for the macro run.
//
run("Close All");
run("Collect Garbage");

////////////////////////////////////////////////////////////////////////////////////////////////
//// Define the following global variables.
//
// These are the names of the fluorescence cubes whose images we want data for
fcNames = newArray("Sedat Red", "Sedat DPA", "Sedat FITC");
// We use these values to help draw the ROI on the straightened images
boxHeight = 700;
boxWidth = 3350;
// How far to displace the ROI in the y-direction relative to the original y-value
// of the line's left point.
yDisplacement = 250;

////////////////////////////////////////////////////////////////////////////////////////////////
//// Preparation steps.
// 1) Have the user choose the folder with the images to process.
// 2) Make folder for the Excel output files inside of the input folder.
// 3) Parse the experiment name from an image in the input folder.
//
inDir = getDirectory("Choose directory to process.");
resultsDir = inDir + File.separator + "Results";
if (!File.exists(resultsDir)) File.makeDirectory(resultsDir);
experimentName = getExperimentName();

////////////////////////////////////////////////////////////////////////////////////////////////
//// Rotation and Data Collection.
// The macro will loop over all of the channels in all of the holders, and:
// 1) Open the channel image.
// 2) Ask the user to draw a line along the top edge of the channel.
// 3) Automatically straighten the image based on the angle of the line.
// 4) Draw a rectangular ROI based on the location of the line.
// 5) Collect the plot profile data for each slice in the image stack
```

```
//	(each slice represents a single timepoint).
// 6) Write the data to the channel's page in the holder's Excel sheet.
// 7) Close the image (without saving it).
// 8) Repeat 3-7 for the other images taken of the same channel using
//	the other fluorescence cubes (the line doesn't have to be drawn
//	again until the next channel is processed).
//
holderNum = 1;
while (holderNum <= 6) {
	// For this holder...
	// Figure out the path to the results file we'll use for this holder.
	resultsFilePath = resultsDir + File.separator + experimentName + "_Holder_" +
		holderNum + ".xlsx";
	// If the file already exists, remove it first.
	if (File.exists(resultsFilePath)) ignored = File.delete(resultsFilePath);

	channelNum = 1;
	while (channelNum <=3) {
		// For this channel...
		// Figure out the sheet name to put results for this channel in in the spreadsheet.
		sheetName = "Channel " + channelNum;

		// We'll store the line coordinates for this channel here.
		// The user will have to draw it the first time.
		lx = -1;
		ly = -1;
		rx = -1;
		ry = -1;

		// Loop over fluorescence cube names...
		for (i = 0; i < fcNames.length; i++) {
			// Build the name of the image we want to open.
			imageName = experimentName + "_" + fcNames[i] + "_Holder_" + holderNum +
				"-" + channelNum + ".tif";
			imagePath = inDir + File.separator + imageName;

			if (File.exists(imagePath)) {
				// Open the image.
				open(imagePath);
				imageTitle = getTitle();

				// Rotate image
				// 1) Check to see if we already have line coordinates stored for this.
				// 2) If we don't, prompts user to draw a line selection under the top
				//	channel length markers. If we do, we draw it ourselves.
				// 3) Uses this line selection to rotate the image.
				if (lx == -1) {
					// We don't already have line coordinates for this channel,
```

```
		// so the user needs to draw one. Note that we expect the
		// user to draw the line from left to right.
		waitForUser("Draw line selection", "Draw line under the " +
			"top-most channel markers to correct for image " +
			"rotation:\nOnce done, click 'OK'");
		getLine(lx, ly, rx, ry, w);
}

// Calculate angle and rotate image.
dx = rx - lx;
dy = ry - ly;
angle = tan(dy/dx);
radAngle = -angle * (180 / PI);

if (nSlices > 1) run("Rotate...", "angle=" + radAngle +
		" grid=1 interpolation=Bilinear stack");
else run("Rotate...", "angle=" + radAngle +
		" grid=1 interpolation=Bilinear");

// Draw ROI based on where the line was drawn.
makeRectangle(lx, (ly + yDisplacement), boxWidth, boxHeight);

// Get the data for each slice.
for (slice = 1; slice <= nSlices; slice++) {
		selectWindow(imageTitle);
		setSlice(slice);

		// Get the plot profile. The returned array contains the y-values
		// of a plot profile at each index.
		profile = getProfile();

		// Put the profile plot values into the results table. (This is what
		// the Excel plugin reads from.)
		for (j = 0; j < profile.length; j++) {
			setResult("X", j, j);
			setResult("Y Avg", j, profile[j]);
		}

		// Run the "Read and Write Excel" plugin to write the results
		// data to a specified sheet in a specified Excel file. We specify
		// the label that we write above the dataset columns (and we
		// specify that we don't want the plugin to add a "Count"
		// column for us, since we have the X column already).
		datasetLabel = fcNames[i] + " " + slice;
		run("Read and Write Excel", "no_count_column file=[" +
			resultsFilePath + "] sheet=[" + sheetName +
			"] dataset_label=[" + datasetLabel + "]");
```

```
                                        // Wait, just to make sure that the Excel file has been closed.
                                        wait(150);

                                        // Now clear the results.
                                        run("Clear Results");
                                }

                                // Close the image.
                                close();
                                run("Collect Garbage");
                        }
                }
                // Increment channel number.
                channelNum++;
        }
        // Increment holder number.
        holderNum++;
}

////////////////////////////////////////////////////////////////////////////////////////////////
//// Results
// After the macro finishes, the output folder will contain one Excel file per holder,
// each Excel file will contain one page per channel in that holder, and each page will
// contain the plot profile data for all of the fluorescence cube-timepoint pairs.
//
showMessage("All done!");


////////////////////////////////////////////////////////////////////////////////////////////////
//// Helper Function: Parse the experiment name from an image filename.
//
function getExperimentName() {
        inFiles = getFileList(inDir);
        for (i = 0; i < inFiles.length; i++) {
                currFile = inDir + inFiles[i];
                if (!File.isDirectory(currFile)) {
                        filenameParts = split(inFiles[i], "_.");
                        return filenameParts[0];
                }
        }
}
```