

```

1 // import statements
2
3 import ij.IJ;
4 import ij.ImagePlus;
5 import ij.ImageStack;
6 import ij.WindowManager;
7 import ij.gui.*;
8 import ij.measure.Calibration;
9 import ij.measure.ResultsTable;
10 import ij.plugin.Duplicator;
11 import ij.plugin.PlugIn;
12 import ij.plugin.ZProjector;
13 import ij.plugin.frame.RoiManager;
14 import ij.process.ImageProcessor;
15
16 import java.awt.*;
17 import java.io.File;
18
19
20 // InsertFancyNameHere (IFNH) plugin for Fiji (tested with Fiji version 1.52p)
21
22 public class InsertFancyNameHere implements PlugIn {
23
24 // main plugin function, secondary functions called within can be found below
25
26
27 public void run(String arg) {
28
29
30 // check for the number of open images: plugin continues only if exactly one image is open
31
32 if (checkExit()) {
33     return;
34 }
35
36
37 // primary parameters for subsequent analysis: user input, plugin aborts if dialog is cancelled
38
39 boolean[] PrimaryParameters = PrimaryParameters_Dialog();
40 boolean one_channel = PrimaryParameters[0];
41 boolean channel1_main = PrimaryParameters[1];
42 boolean Z_scan_into_tissue = PrimaryParameters[2];
43 boolean horizontal = PrimaryParameters[3];
44 boolean automatic_centroid = PrimaryParameters[4];
45 boolean advancedOptions = PrimaryParameters[5];
46 boolean PrimaryParameters_Dialog_cancelled = PrimaryParameters[6];
47
48 if (PrimaryParameters_Dialog_cancelled) {
49     return;
50 }
51
52
53 // secondary parameters for subsequent analysis: user input, plugin aborts if dialog is cancelled
54
55 double[] SecondaryParameters = SecondaryParameters_Dialog(automatic_centroid,
advancedOptions);
56     double upper_threshold = SecondaryParameters[0];
57     double lower_threshold = SecondaryParameters[1] * upper_threshold / 100;
58     double filter_radius = SecondaryParameters[2];
59
60     double domain_diameter = SecondaryParameters[3];
61
62     boolean peripheral_analysis = false;
63     double peripheral_analysis_double = SecondaryParameters[4];
64
65     if (peripheral_analysis_double == 1) {
66         peripheral_analysis = true;
67     }
68
69     double periphery_width = SecondaryParameters[5];
70
71     double double_CirclePointSelectionDelay = SecondaryParameters[6];
72     int CirclePointSelectionDelay = (int) Math.round(double_CirclePointSelectionDelay);

```

```

73    double SecondaryParameters_Dialog_cancelled = SecondaryParameters[7];
74
75    if (SecondaryParameters_Dialog_cancelled == 1) {
76        return;
77    }
78
79
80 // selection of save-directory, automatic naming to avoid overwriting of existing data
81
82    ImagePlus activeImage = WindowManager.getCurrentImage();
83    String title = activeImage.getShortTitle();
84    String[] createDirectory = createDirectory(title);
85    String directory = createDirectory[0];
86    title = createDirectory[1];
87    String folder_status = createDirectory[2];
88
89    if (folder_status.equals("not created")) {
90        activeImage.close();
91        return;
92    }
93
94
95 // ROI manager reset, Results reset
96
97    RoiManager ROI = RoiManager.getRoiManager();
98    ROI.reset();
99    IJ.run("Clear Results");
100   activeImage.setSlice(1);
101
102
103 // pre-processing (LUT, naming) and splitting of channels depending on previous user selection of
104 // primary parameters
105
106    if (one_channel) {
107        IJ.run(activeImage, "royal", "");
108        IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel");
109    }
110
111    else { // != one_channel
112        activeImage.setTitle(title + ".tif");
113        IJ.run(activeImage, "Split Channels", "");
114
115        if (channel1_main) {
116            activeImage = WindowManager.getImage("C2-" + title + ".tif");
117            IJ.run(activeImage, "royal", "");
118            IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel");
119            activeImage.changes = false;
120            activeImage.close();
121
122            activeImage = WindowManager.getImage("C1-" + title + ".tif");
123            IJ.run(activeImage, "royal", "");
124            IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel");
125        }
126
127        else { // != channel1_main
128            activeImage = WindowManager.getImage("C1-" + title + ".tif");
129            IJ.run(activeImage, "royal", "");
130            IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel");
131            activeImage.changes = false;
132            activeImage.close();
133
134            activeImage = WindowManager.getImage("C2-" + title + ".tif");
135            IJ.run(activeImage, "royal", "");
136            IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel");
137        }
138
139
140 // rotation of images to create a top-view image (if not already defined as top-view, based on previous
141 // user selection of primary parameters)
142
143    if (horizontal) {

```

```

144     activeImage = rotate_horizontal2vertical(activeImage, Z_scan_into_tissue);
145
146     IJ.saveAs(activeImage, "Tiff", directory + title + "_temporary1");
147
148     if (!one_channel) {
149         activeImage = IJ.openImage(directory + title + "_support-channel.tif");
150         activeImage.show();
151
152         activeImage = rotate_horizontal2vertical(activeImage, Z_scan_into_tissue);
153
154         IJ.saveAs(activeImage, "Tiff", directory + title + "_temporary2");
155         activeImage.changes = false;
156         activeImage.close();
157
158         activeImage = WindowManager.getCurrentImage();
159     }
160 }
161
162
163 // definition of variables for analysis domain center point definition (for automated and manual
164 // selection, based on previous user selection of primary parameters)
165 Calibration activeImageCalibration;
166
167 double double_CenterPointX_microns;
168 double double_CenterPointY_microns;
169
170 double voxel_width;
171 double voxel_height;
172
173 double double_CenterPointX_pixels;
174 double double_CenterPointY_pixels;
175
176 int int_CenterPointX_pixels = 0;
177 int int_CenterPointY_pixels = 0;
178
179
180 // (semi-)automated definition of the analysis domain center point via thresholding (if previously
181 // selected by the user)
182 if (automatic_centroid) {
183     IJ.setThreshold(activeImage, lower_threshold, upper_threshold);
184
185     for (int subcounter = 1; subcounter <= activeImage.getStackSize(); subcounter++) {
186         IJ.setSlice(subcounter);
187         IJ.run(activeImage, "Create Selection", "");
188         Roi selection_type = activeImage.getRoi();
189
190         if (selection_type != null) {
191             IJ.run(activeImage, "Clear Outside", "slice");
192         }
193
194         if (selection_type == null) {
195             IJ.run(activeImage, "Select All", "");
196             IJ.run(activeImage, "Clear", "slice");
197         }
198     }
199
200     IJ.resetThreshold(activeImage);
201     IJ.run(activeImage, "Select None", "");
202
203     IJ.run(activeImage, "Median 3D...", "x=" + filter_radius + " y=" + filter_radius + " z=" +
204     filter_radius);
205
206     activeImage = maxProjection(activeImage);
207     IJ.run(activeImage, "royal", "");
208
209     IJ.setThreshold(activeImage, lower_threshold, upper_threshold);
210     IJ.run(activeImage, "Analyze Particles...", "size=10-Infinity show=Masks include clear
211     include add");
212
213     if (horizontal) {
214         WindowManager.getImage("Mask of MAX_" + title + "_temporary1.tif").close();

```

```

213
214
215     }
216
217     else { // = (!horizontal)
218         WindowManager.getImage("Mask of MAX_" + title + "_main-channel.tif").close();
219     }
220
221     WindowManager.getCurrentImage().setTitle("ROI selection window - " + (int)Math.round(
222     SecondaryParameters[1]) + " %");
223
224     ROI.deselect();
225     int number_ROIs = ROI.getCount();
226
227     int columns = 3;
228     int rows_remainder = number_ROIs % columns;
229     int rows = 0;
230
231     if (rows_remainder == 1 || rows_remainder == 2) {
232         rows = number_ROIs / columns + 1;
233     }
234
235     if (rows_remainder == 0) {
236         rows = number_ROIs / columns;
237     }
238
239     String[] CheckboxLabels = new String[number_ROIs];
240     boolean[] CheckboxDefaults = new boolean[number_ROIs];
241
242     for (int ROI_index = 0; ROI_index < number_ROIs; ROI_index++) {
243         int ROI_number = ROI_index + 1;
244         CheckboxLabels[ROI_index] = "ROI " + ROI_number;
245         CheckboxDefaults[ROI_index] = false;
246     }
247
248     GenericDialog ROI_Dialog = new GenericDialog("ROI selection");
249
250     ROI_Dialog.addCheckboxGroup(rows, 3, CheckboxLabels, CheckboxDefaults);
251
252     ROI_Dialog.setCancelLabel("Threshold");
253     ROI_Dialog.showDialog();
254
255 // repeats thresholding based ROI definition (based on the users decision that the ROIs found do not
256 // sufficiently describe the domain of interest, can be repeated indefinitely)
257
258     boolean ThresholdReset;
259     boolean ThresholdReset_repeat = false;
260
261     if (ROI_Dialog.wasCanceled()) {
262         ThresholdReset = true;
263
264         double percentage = SecondaryParameters[1];
265         double new_percentage = percentage;
266
267         do {
268             if (ThresholdReset_repeat) {
269                 percentage = new_percentage;
270             }
271
272             GenericDialog ThresholdReset_Dialog = new GenericDialog("Reset lower threshold");
273
274             ThresholdReset_Dialog.setInsets(10, 10, 0);
275             ThresholdReset_Dialog.addMessage("Lower threshold: ");
276             ThresholdReset_Dialog.setInsets(5, 10, 7);
277             ThresholdReset_Dialog.addNumericField("", percentage, 0, 3, "% of upper threshold");
278         });
279
280         ThresholdReset_Dialog.showDialog();
281
282         if (ThresholdReset_Dialog.wasCanceled()) {
283             return;
284         }

```

```

283     new_percentage = ThresholdReset_Dialog.getNextNumber();
284     lower_threshold = upper_threshold / 100 * new_percentage;
285
286     if (horizontal) {
287         activeImage = IJ.openImage(directory + title + "_temporary1.tif");
288     }
289
290     else { // = (!horizontal)
291         activeImage = IJ.openImage(directory + title + "_main-channel.tif");
292     }
293
294     activeImage.show();
295
296     IJ.setThreshold(activeImage, lower_threshold, upper_threshold);
297
298     for (int subcounter = 1; subcounter <= activeImage.getStackSize(); subcounter++) {
299         IJ.setSlice(subcounter);
300         IJ.run(activeImage, "Create Selection", "");
301         Roi selection_type = activeImage.getRoi();
302
303         if (selection_type != null) {
304             IJ.run(activeImage, "Clear Outside", "slice");
305         }
306
307         if (selection_type == null) {
308             IJ.run(activeImage, "Select All", "");
309             IJ.run(activeImage, "Clear", "slice");
310         }
311     }
312
313     IJ.resetThreshold(activeImage);
314     IJ.run(activeImage, "Select None", "");
315
316     IJ.run(activeImage, "Median 3D...", "x=" + filter_radius + " y=" + filter_radius + "
317     z=" + filter_radius);
318
319     activeImage = maxProjection(activeImage);
320     IJ.run(activeImage, "royal", "");
321
322     IJ.setThreshold(activeImage, lower_threshold, upper_threshold);
323     IJ.run(activeImage, "Analyze Particles...", "size=10-Infinity show=Masks include
324     clear include add");
325
326     if (horizontal) {
327         WindowManager.getImage("Mask of MAX_" + title + "_temporary1.tif").close();
328     }
329
330     else { // = (!horizontal)
331         WindowManager.getImage("Mask of MAX_" + title + "_main-channel.tif").close(
332     );
333     }
334
335     WindowManager.getCurrentImage().setTitle("ROI selection window - " + (int)Math.
336     round(new_percentage) + " %");
337
338     ROI.deselect();
339     number_ROIs = ROI.getCount();
340
341     columns = 3;
342     rows_remainder = number_ROIs % columns;
343     rows = 0;
344
345     if (rows_remainder == 1 || rows_remainder == 2) {
346         rows = number_ROIs / columns + 1;
347     }
348
349     if (rows_remainder == 0) {
350         rows = number_ROIs / columns;
351     }
352
353     CheckboxLabels = new String[number_ROIs];
354     CheckboxDefaults = new boolean[number_ROIs];

```

```

352
353     for (int ROI_index = 0; ROI_index < number_ROIs; ROI_index++) {
354         int ROI_number = ROI_index + 1;
355         CheckboxLabels[ROI_index] = "ROI " + ROI_number;
356         CheckboxDefaults[ROI_index] = false;
357     }
358
359     ROI_Dialog = new GenericDialog("ROI selection");
360
361     ROI_Dialog.addCheckboxGroup(rows, 3, CheckboxLabels, CheckboxDefaults);
362
363     ROI_Dialog.setCancelLabel("Threshold");
364     ROI_Dialog.showDialog();
365
366     ThresholdReset_repeat = true;
367
368     if (ROI_Dialog.wasOKed()) {
369
370         int[] IDlist = WindowManager.getIDList();
371
372         for (int subcounter = 0; subcounter < IDlist.length - 1; subcounter++) {
373             WindowManager.getImage(IDlist[subcounter]).close();
374         }
375
376         ThresholdReset = false;
377     }
378
379     } while (ThresholdReset);
380 }
381
382
383 // combination of multiple ROIs (if selected) to a single ROI of interest
384
385 for (int ROI_index = 0; ROI_index < number_ROIs; ROI_index++) {
386
387     boolean ROI_selected = ROI_Dialog.getNextBoolean();
388
389     if (!ROI_selected) {
390         ROI.select(ROI_index);
391         ROI.runCommand("Delete");
392         ROI.deselect();
393         ROI_index--;
394         number_ROIs--;
395     }
396 }
397
398 int number_ROIs_chosen = ROI.getCount();
399
400 if (number_ROIs_chosen != 1) {
401     ROI.deselect();
402     ROI.runCommand("Combine");
403     ROI.deselect();
404     ROI.runCommand("Delete");
405     ROI.addRoi(activeImage.getRoi());
406 }
407
408 WindowManager.getCurrentImage().close();
409
410
411 // definition of the centroid of the ROI of interest, to be used as the center point for the circular
412 // analysis domain
413
414     if (horizontal) {
415         activeImage = IJ.openImage(directory + title + "_temporary1.tif");
416     }
417
418     else { // = (!horizontal)
419         activeImage = IJ.openImage(directory + title + "_main-channel.tif");
420     }
421
422     activeImage.show();
423
424     activeImage = maxProjection(activeImage);

```

```

424
425     ROI.select(0);
426     IJ.run("Set Measurements...", "centroid redirect=None decimal=3");
427     IJ.run("Measure", "");
428     ROI.reset();
429     IJ.run("Select None", "");
430
431     ResultsTable centroid_ResultsTable = ResultsTable.getResultsTable();
432
433     double_CenterPointX_microns = centroid_ResultsTable.getValue("X", 0);
434     double_CenterPointY_microns = centroid_ResultsTable.getValue("Y", 0);
435
436     IJ.run("Clear Results");
437
438     activeImageCalibration = activeImage.getCalibration();
439     voxel_width = activeImageCalibration.pixelWidth;
440     voxel_height = activeImageCalibration.pixelHeight;
441
442     double_CenterPointX_pixels = double_CenterPointX_microns / voxel_width;
443     double_CenterPointY_pixels = double_CenterPointY_microns / voxel_height;
444
445     int_CenterPointX_pixels = (int) Math.round(double_CenterPointX_pixels);
446     int_CenterPointY_pixels = (int) Math.round(double_CenterPointY_pixels);
447
448     ImageProcessor activeImageProcessor = activeImage.getProcessor();
449     IJ.setForegroundColor(255, 255, 255);
450     activeImageProcessor.fillOval(int_CenterPointX_pixels - 2, int_CenterPointY_pixels - 2, 5, 5)
451 ;
452 }
453
454 // manual definition of the analysis domain center point via clicking into the image (if previously
455 // selected by the user)
456 else { // = (!automated_centroid)
457
458     activeImage = maxProjection(activeImage);
459
460     IJ.setTool("hand");
461     ImageCanvas activeImageCanvas = activeImage.getCanvas();
462     ImageProcessor activeImageProcessor = activeImage.getProcessor();
463
464     WaitForUserDialog CentroidSelection_Dialog = new WaitForUserDialog("Center Point
465 Definition", " \nClick to define the center point of the analysis domain. \n ");
466     CentroidSelection_Dialog.show();
467
468     boolean CenterPoint_selected = false;
469
470     while (!CenterPoint_selected) {
471
472         int[] CursorValues = getCursorLocation(activeImageCanvas);
473         int CursorX = CursorValues[0];
474         int CursorY = CursorValues[1];
475         int CursorModifiers = CursorValues[2];
476
477         if (CursorModifiers == 16) {
478             IJ.setForegroundColor(255, 255, 255);
479             activeImageProcessor.fillOval(CursorX - 2, CursorY - 2, 5, 5);
480             int_CenterPointX_pixels = CursorX;
481             int_CenterPointY_pixels = CursorY;
482             CenterPoint_selected = true;
483         }
484
485         activeImage.updateAndDraw();
486     }
487
488     double_CenterPointX_pixels = int_CenterPointX_pixels;
489     double_CenterPointY_pixels = int_CenterPointY_pixels;
490
491     activeImageCalibration = activeImage.getCalibration();
492     voxel_width = activeImageCalibration.pixelWidth;
493     voxel_height = activeImageCalibration.pixelHeight;

```

```

494     double_CenterPointX_microns = double_CenterPointX_pixels * voxel_width;
495     double_CenterPointY_microns = double_CenterPointY_pixels * voxel_height;
496 }
497
498
499 // specification of the circular analysis domain based on the previously selected or semi-automatically
500 // defined center point
501 IJ.run(activeImage, "Specify...", "width=" + domain_diameter + " height=" +
502 domain_diameter + " x=" + double_CenterPointX_microns + " y=" + double_CenterPointY_microns
503 + " slice=1 oval centered scaled");
504 ROI.addRoi(activeImage.getRoi());
505 ROI.select(0);
506 IJ.setForegroundColor(255, 255, 255);
507 IJ.run(activeImage, "Draw", "stack");
508
509 IJ.run("Select None", " ");
510
511
512 // specification of a second circular analysis domain if analysis of the periphery has been selected by
513 // the user
514 if(peripheral_analysis){
515
516     double peripheral_domain_diameter = domain_diameter + 2 * periphery_width;
517
518     IJ.run(activeImage, "Specify...", "width=" + peripheral_domain_diameter + " height=" +
519 peripheral_domain_diameter + " x=" + double_CenterPointX_microns + " y=" +
520 double_CenterPointY_microns + " slice=1 oval centered scaled");
521 ROI.addRoi(activeImage.getRoi());
522 ROI.select(1);
523 IJ.setForegroundColor(255, 255, 255);
524 IJ.run(activeImage, "Draw", "stack");
525
526 IJ.run("Select None", " ");
527
528 IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel-analysis-domain(s)");
529 activeImage.changes = false;
530 activeImage.close();
531 }
532
533 else {
534     IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_analysis-domain");
535     activeImage.changes = false;
536     activeImage.close();
537 }
538
539
540 if (horizontal) {
541     activeImage = IJ.openImage(directory + title + "_temporary1.tif");
542 }
543
544 else { // = (!horizontal)
545     activeImage = IJ.openImage(directory + title + "_main-channel.tif");
546 }
547
548 activeImage.show();
549
550 if (peripheral_analysis){
551     ROI.select(1);
552 }
553
554 else { // = (!peripheral_analysis)
555     ROI.select(0);
556 }
557
558
559 // removal of all signal intensity that is not included within the analysis domain (or the peripheral
560 // analysis domain), processing of all images needed for later polar transformation

```

```

560
561 IJ.run(activeImage, "Clear Outside", "stack");
562 IJ.run("Select None", "");
563
564 activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);
565
566 int stack_size = activeImage.getStackSize();
567 int startslice = 1;
568 int endslice = stack_size;
569
570 boolean startslice_determined = false;
571 boolean endslice_determined = false;
572
573 for (int subcounter = 1; subcounter <= stack_size; subcounter++) {
574     IJ.setSlice(subcounter);
575     IJ.run("Clear Results");
576     IJ.run("Set Measurements...", "integrated redirect=None decimal=3");
577     IJ.run("Measure", "");
578
579     ResultsTable startslice_endslice_results_table = ResultsTable.getResultsTable();
580     double total_pixel_intensity = startslice_endslice_results_table.getValue("RawIntDen", 0);
581     IJ.run("Clear Results");
582
583     if (total_pixel_intensity != 0 && !startslice_determined) {
584         startslice = activeImage.getCurrentSlice();
585         startslice_determined = true;
586     }
587
588     if (total_pixel_intensity == 0 && startslice_determined && !endslice_determined) {
589         endslice = activeImage.getCurrentSlice() - 1;
590         endslice_determined = true;
591     }
592 }
593
594 activeImage.close();
595
596
597 if (horizontal) {
598     activeImage = IJ.openImage(directory + title + "_temporary1.tif");
599 }
600
601 else { // = (!horizontal)
602     activeImage = IJ.openImage(directory + title + "_main-channel.tif");
603 }
604
605 activeImage.show();
606
607 ROI.select(0);
608 IJ.run(activeImage, "Clear Outside", "stack");
609 IJ.run("Select None", "");
610
611 activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);
612
613 ImagePlus duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
614 duplicatedImage.show();
615
616 activeImage.changes = false;
617 activeImage.close();
618
619 IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_main-channel_not-unrolled");
620 duplicatedImage.changes = false;
621 duplicatedImage.close();
622
623
624 if(peripheral_analysis){
625
626     if (horizontal) {
627         activeImage = IJ.openImage(directory + title + "_temporary1.tif");
628     }
629
630     else {
631         activeImage = IJ.openImage(directory + title + "_main-channel.tif");
632     }

```

```

633     activeImage.show();
634
635     ROI.select(1);
636     IJ.run(activeImage, "Clear Outside", "stack");
637     IJ.run("Select None", "");
638
639     ROI.select(0);
640     IJ.run(activeImage, "Clear", "stack");
641
642     IJ.run("Select None", "");
643
644     activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);
645
646     duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
647     duplicatedImage.show();
648
649     activeImage.changes = false;
650     activeImage.close();
651
652     IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_main-channel_not-
unrolled_periphery");
653     duplicatedImage.changes = false;
654     duplicatedImage.close();
655
656 }
657
658 if (one_channel) {
659     ROI.reset();
660
661     if (horizontal) {
662         activeImage = IJ.openImage(directory + title + "_temporary1.tif");
663     } else {
664         activeImage = IJ.openImage(directory + title + "_main-channel.tif");
665     }
666     activeImage.show();
667
668     activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);
669
670     duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
671     duplicatedImage.show();
672     activeImage.changes = false;
673     activeImage.close();
674 }
675
676 if(horizontal) {
677     File file = new File(directory + title + "_temporary1.tif");
678
679     if (!file.delete()) {
680         IJ.error("Error Message", "File could not be deleted");
681     }
682 }
683
684 if(one_channel) {
685     IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_main-channel_circle-points");
686     duplicatedImage.changes = false;
687     duplicatedImage.close();
688 }
689
690 else { // = (!one_channel)
691
692     if (horizontal) {
693         activeImage = IJ.openImage(directory + title + "_temporary2.tif");
694     }
695
696     else {
697         activeImage = IJ.openImage(directory + title + "_support-channel.tif");
698     }
699     activeImage.show();
700
701     ROI.select(0);
702
703     IJ.run(activeImage, "Clear Outside", "stack");
704

```

```

705 IJ.run("Select None", "");
706
707 activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);
708
709 duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
710 duplicatedImage.show();
711 activeImage.changes = false;
712 activeImage.close();
713 IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_support-channel_not-unrolled");
714 duplicatedImage.changes = false;
715 duplicatedImage.close();
716
717 if (peripheral_analysis) {
718
719     if (horizontal) {
720         activeImage = IJ.openImage(directory + title + "_temporary2.tif");
721     }
722
723     else {
724         activeImage = IJ.openImage(directory + title + "_support-channel.tif");
725     }
726     activeImage.show();
727
728     ROI.select(1);
729     IJ.run(activeImage, "Clear Outside", "stack");
730     IJ.run("Select None", "");
731
732
733     ROI.select(0);
734     IJ.run(activeImage, "Clear", "stack");
735     IJ.run("Select None", "");
736
737     activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);
738
739     duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
740     duplicatedImage.show();
741     activeImage.changes = false;
742     activeImage.close();
743     IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_support-channel_not-
744     unrolled_periphery");
744     duplicatedImage.changes = false;
745     duplicatedImage.close();
746
747 }
748
749 ROI.reset();
750
751
752 if (horizontal) {
753     activeImage = IJ.openImage(directory + title + "_temporary2.tif");
754 }
755
756 else {
757     activeImage = IJ.openImage(directory + title + "_support-channel.tif");
758 }
759
760 activeImage.show();
761
762 activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);
763
764 duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
765 duplicatedImage.show();
766 activeImage.changes = false;
767 activeImage.close();
768
769
770 if (horizontal) {
771     File file = new File(directory + title + "_temporary2.tif");
772
773     if (!file.delete()) {
774         IJ.error("Error Message", "File could not be deleted");
775     }
776 }

```

```

777     IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_support-channel_circle-points");
779     duplicatedImage.changes = false;
780     duplicatedImage.close();
781 }
782
783
784 // selection of circle points for polar transformation
785
786 if (one_channel) {
787     activeImage = IJ.openImage(directory + title + "_main-channel_circle-points.tif");
788     activeImage.show();
789 }
790
791 else { // != (!one_channel)
792     activeImage = IJ.openImage(directory + title + "_support-channel_circle-points.tif");
793     activeImage.show();
794 }
795
796 stack_size = activeImage.getStackSize();
797
798 int[][] CirclePointsX = new int[3][stack_size];
799 int[][] CirclePointsY = new int[3][stack_size];
800
801 ImageCanvas activeImageCanvas = activeImage.getCanvas();
802 ImageProcessor activeImageProcessor = activeImage.getProcessor();
803
804 IJ.setTool("hand");
805
806 WaitForUserDialog PointSelection_Dialog1 = new WaitForUserDialog("Circle Point Definition
", " \nDefine first point-array for meristem unrolling. \n \nClick 'OK' to start with circle point
definition.\n ");
807 PointSelection_Dialog1.show();
808
809 for (int subcounter = 0; subcounter <= 2; subcounter++) {
810
811     int currentSlice = 1;
812
813     IJ.setForegroundColor(255, 255, 255);
814
815     while (currentSlice <= stack_size) {
816         IJ.setSlice(currentSlice);
817         int[] CursorValues = getCursorLocation(activeImageCanvas);
818         int CursorX = CursorValues[0];
819         int CursorY = CursorValues[1];
820         int CursorModifiers = CursorValues[2];
821
822
823         if (CursorModifiers == 16) {
824
825             if (currentSlice == stack_size) {
826
827                 CirclePointsX[subcounter][currentSlice - 1] = CursorX;
828                 CirclePointsY[subcounter][currentSlice - 1] = CursorY;
829                 activeImageProcessor.fillOval(CursorX - 2, CursorY - 2, 5, 5);
830
831                 currentSlice++;
832
833             if (subcounter == 0 || subcounter == 1) {
834                 WaitForUserDialog PointSelection_Dialog2 = new WaitForUserDialog("Circle
Point Definition", " \nDefine next point-array for meristem unrolling. \n \nClick 'OK' to
continue with circle point definition.\n ");
835                 PointSelection_Dialog2.show();
836             }
837
838             if (subcounter == 2) {
839                 WaitForUserDialog PointSelection_Dialog3 = new WaitForUserDialog("Circle
Point Definition", " \nCircle point definition has been completed. \n \nClick 'OK' to continue
with subsequent analysis.\n ");
840                 PointSelection_Dialog3.show();
841             }
842         } else {
843

```

```

844     CirclePointsX[subcounter][currentSlice - 1] = CursorX;
845     CirclePointsY[subcounter][currentSlice - 1] = CursorY;
846     activeImageProcessor.fillOval(CursorX - 2, CursorY - 2, 5, 5);
847
848     currentSlice++;
849
850     IJ.wait(CirclePointSelectionDelay);
851   }
852 }
853 }
854 }
855
856 if (one_channel) {
857   IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_circle-points");
858
859 } else { // != (!one_channel)
860   IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_circle-points");
861 }
862
863 activeImage.close();
864
865
866
867
868 // calculation of image canvas size needed after polar transformation
869
870 double[] CenterPointX_array = new double[stack_size];
871 double[] CenterPointY_array = new double[stack_size];
872 double[] radius_array = new double[stack_size];
873 int[] highest_CirclePoint_Y = new int[stack_size];
874
875 for (int new_z = 0; new_z < stack_size; new_z++) {
876
877   int CirclePointX_1 = CirclePointsX[0][new_z];
878   int CirclePointY_1 = CirclePointsY[0][new_z];
879
880   int CirclePointX_2 = CirclePointsX[1][new_z];
881   int CirclePointY_2 = CirclePointsY[1][new_z];
882
883   int CirclePointX_3 = CirclePointsX[2][new_z];
884   int CirclePointY_3 = CirclePointsY[2][new_z];
885
886   if (CirclePointY_1 <= CirclePointY_2 && CirclePointY_1 <= CirclePointY_3) {
887     highest_CirclePoint_Y[new_z] = CirclePointY_1;
888   }
889
890   if (CirclePointY_2 <= CirclePointY_1 && CirclePointY_2 <= CirclePointY_3) {
891     highest_CirclePoint_Y[new_z] = CirclePointY_2;
892   }
893
894   if (CirclePointY_3 <= CirclePointY_1 && CirclePointY_3 <= CirclePointY_2) {
895     highest_CirclePoint_Y[new_z] = CirclePointY_3;
896   }
897
898   CenterPointX_array[new_z] = ((Math.pow(CirclePointX_1, 2) + Math.pow(CirclePointY_1, 2)
899     ) * (CirclePointY_2 - CirclePointY_3) + (Math.pow(CirclePointX_2, 2) + Math.pow(CirclePointY_2,
900     2)) * (CirclePointY_3 - CirclePointY_1) + (Math.pow(CirclePointX_3, 2) + Math.pow(CirclePointY_3
901     , 2)) * (CirclePointY_1 - CirclePointY_2)) / (2 * (CirclePointX_1 * (CirclePointY_2 - CirclePointY_3)
902     + CirclePointX_2 * (CirclePointY_3 - CirclePointY_1) + CirclePointX_3 * (CirclePointY_1 -
903     CirclePointY_2)));
904
905   CenterPointY_array[new_z] = ((Math.pow(CirclePointX_1, 2) + Math.pow(CirclePointY_1, 2)
906     ) * (CirclePointX_3 - CirclePointX_2) + (Math.pow(CirclePointX_2, 2) + Math.pow(CirclePointY_2,
907     2)) * (CirclePointX_1 - CirclePointX_3) + (Math.pow(CirclePointX_3, 2) + Math.pow(CirclePointY_3
908     , 2)) * (CirclePointX_2 - CirclePointX_1)) / (2 * (CirclePointX_1 * (CirclePointY_2 - CirclePointY_3)
909     + CirclePointX_2 * (CirclePointY_3 - CirclePointY_1) + CirclePointX_3 * (CirclePointY_1 -
910     CirclePointY_2)));
911
912   radius_array[new_z] = (Math.sqrt(Math.pow((CirclePointX_1 - CenterPointX_array[new_z]), 2)
913     + Math.pow((CirclePointY_1 - CenterPointY_array[new_z]), 2)) + Math.sqrt(Math.pow(
914     CirclePointX_2 - CenterPointX_array[new_z], 2) + Math.pow((CirclePointY_2 - CenterPointY_array[
915     new_z]), 2)) + Math.sqrt(Math.pow((CirclePointX_3 - CenterPointX_array[new_z]), 2) + Math.pow(
916     CirclePointY_3 - CenterPointY_array[new_z], 2))) / 3;
917 }

```

```

903
904
905 // if peripheral analysis is active the whole analysis is duplicated from this point on
906
907     for (int counter = 0; counter <= 1; counter++) {
908
909         String peripheral_analysis_name_variabel;
910
911         if (peripheral_analysis && counter == 0) {
912             peripheral_analysis_name_variabel = "";
913         }
914
915         else { // = (peripheral_analysis && counter == 1)
916             peripheral_analysis_name_variabel = "_periphery";
917         }
918
919         if (!peripheral_analysis) {
920             peripheral_analysis_name_variabel = "";
921         }
922
923
924 // removal of all signal intensity below the center point of the circle (pole for polar transformation)
925
926     activeImage = IJ.openImage(directory + title + "_main-channel_not-unrolled" +
927     peripheral_analysis_name_variabel + ".tif");
928     activeImage.show();
929
930     for (int subcounter = 1; subcounter <= stack_size; subcounter++) {
931
932         activeImage.setSlice(subcounter);
933
934         if (CenterPointY_array[subcounter - 1] > activeImage.getHeight()) {
935             activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.getWidth()
936             (), activeImage.getHeight() - highest_CirclePoint_Y[subcounter - 1]);
937
938             if (highest_CirclePoint_Y[subcounter - 1] == 0) {
939                 activeImage.setRoi(0, 0, activeImage.getWidth(), activeImage.getHeight() -
940                 highest_CirclePoint_Y[subcounter - 1]);
941             }
942
943             } else {
944                 activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.getWidth()
945                 (), (int) Math.round(CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter - 1]));
946             }
947
948             IJ.run(activeImage, "Clear Outside", "slice");
949             IJ.run(activeImage, "Select None", "");
950             ROI.reset();
951         }
952
953         IJ.run(activeImage, "HiLo", "");
954         IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_not-unrolled" +
955         peripheral_analysis_name_variabel);
956
957         activeImage.hide();
958
959 // polar transformation
960
961     ImagePlus inputImage = activeImage;
962
963     activeImage = transform_polarTransformation(inputImage, stack_size, CenterPointX_array,
964     CenterPointY_array, radius_array);
965     IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_unrolled-3D" +
966     peripheral_analysis_name_variabel);
967
968 // reduction of information dimension via subsequent rotations and projections (3D to 1D)

```

```

968
969     activeImage = sumProjection(activeImage);
970     IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_unrolled-2D" +
peripheral_analysis_name_variabel);
971
972     IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=90 x-angle=0");
973     activeImage.close();
974     activeImage = WindowManager.getCurrentImage();
975
976     activeImage = sumProjection(activeImage);
977     IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_unrolled-1D");
978
979
980 // extraction of all signal intensity in 1D image to create a line plot
981
982     activeImageProcessor = activeImage.getProcessor();
983
984     float[] ProfilePlotXValues = new float[activeImage.getHeight()];
985     float[] mainProfilePlotYValues = new float[ProfilePlotXValues.length];
986     float[] supportProfilePlotYValues = new float[ProfilePlotXValues.length];
987     float[] selectProfilePlotYValues = new float[ProfilePlotXValues.length];
988
989     for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {
990         ProfilePlotXValues[subcounter] = subcounter;
991         mainProfilePlotYValues[subcounter] = activeImageProcessor.getPixelValue(0, subcounter)
992 ;
993     }
994
995     activeImage.close();
996
997 // repetition of polar transformation, dimension reduction and information extraction in case there
998 // was a secondary channel
999
1000    if (!one_channel) {
1001
1002        activeImage = IJ.openImage(directory + title + "_support-channel_not-unrolled" +
peripheral_analysis_name_variabel + ".tif");
1003        activeImage.show();
1004
1005        for (int subcounter = 1; subcounter <= stack_size; subcounter++) {
1006
1007            activeImage.setSlice(subcounter);
1008
1009            if (CenterPointY_array[subcounter - 1] > activeImage.getHeight()) {
1010                activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.
getWidth(), activeImage.getHeight() - highest_CirclePoint_Y[subcounter - 1]);
1011
1012                if (highest_CirclePoint_Y[subcounter - 1] == 0) {
1013                    activeImage.setRoi(0, 0, activeImage.getWidth(), activeImage.getHeight() -
highest_CirclePoint_Y[subcounter - 1]);
1014                }
1015
1016                } else {
1017                    activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.
getWidth(), (int) Math.round(CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter -
1]));
1018
1019                    if (highest_CirclePoint_Y[subcounter - 1] == 0) {
1020                        activeImage.setRoi(0, 0, activeImage.getWidth(), (int) Math.round(
CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter - 1]));
1021                    }
1022
1023                    IJ.run(activeImage, "Clear Outside", "slice");
1024                    IJ.run(activeImage, "Select None", "");
1025                    ROI.reset();
1026                }
1027
1028                IJ.run(activeImage, "HiLo", "");
1029                IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_not-unrolled" +
peripheral_analysis_name_variabel);
1030

```

```

1031     activeImage.hide();
1032
1033     inputImage = activeImage;
1034
1035     activeImage = transform_polarTransformation(inputImage, stack_size, CenterPointX_array,
1036     CenterPointY_array, radius_array);
1037     IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-3D" +
1038     peripheral_analysis_name_variabel);
1039
1040     activeImage = sumProjection(activeImage);
1041     IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-2D" +
1042     peripheral_analysis_name_variabel);
1043
1044     activeImage = WindowManager.getCurrentImage();
1045
1046     activeImage = sumProjection(activeImage);
1047     IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-1D");
1048
1049     activeImageProcessor = activeImage.getProcessor();
1050
1051     for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {
1052         supportProfilePlotYValues[subcounter] = activeImageProcessor.getPixelValue(0,
1053         subcounter);
1054     }
1055
1056     activeImage.close();
1057
1058
1059 // plotting of all information in a line plot to select nuclei borders
1060
1061     int PlotDrawingFrameX;
1062     double activeImagePixelWidth;
1063
1064     int[] CursorX_array = new int[4];
1065     int[] PlotXValues_array = new int[4];
1066
1067     String PlotTitle;
1068
1069     if (one_channel) {
1070
1071         PlotTitle = title + "_main-channel_profile-plot" + peripheral_analysis_name_variabel +
1072         ".tif";
1073         System.arraycopy(mainProfilePlotYValues, 0, selectProfilePlotYValues, 0,
1074         ProfilePlotXValues.length);
1075     }
1076
1077     else { // = (!one_channel)
1078
1079         PlotTitle = title + "_support-channel_profile-plot" +
1080         peripheral_analysis_name_variabel + ".tif";
1081         System.arraycopy(supportProfilePlotYValues, 0, selectProfilePlotYValues, 0,
1082         ProfilePlotXValues.length);
1083     }
1084
1085     Plot ProfilePlot = new Plot(PlotTitle, "Distance", "Gray Values");
1086     ProfilePlot.setColor(Color.GRAY);
1087     ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.LINE);
1088
1089     ProfilePlot.setColor(Color.BLACK);
1090     ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.CIRCLE);
1091
1092     ProfilePlot.setFrozen(true);
1093     ProfilePlot.show();
1094
1095     activeImage = ProfilePlot.getImagePlus();
1096     activeImageCanvas = activeImage.getCanvas();
1097     activeImagePixelWidth = activeImage.getCalibration().pixelWidth;
1098
1099     PlotDrawingFrameX = ProfilePlot.getDrawingFrame().x;

```

```

1096
1097
1098     IJ.setTool("hand");
1099
1100    for (int subcounter = 0; subcounter <= 2; subcounter++) {
1101        if (subcounter == 0) {
1102            WaitForUserDialog NucleiBorderSelection_Dialog = new WaitForUserDialog("Cell
1103 layer definition", "\nClick to define the border between the L1 and L2.\n");
1104            NucleiBorderSelection_Dialog.show();
1105        }
1106
1107        if (subcounter == 1) {
1108            WaitForUserDialog NucleiBorderSelection_Dialog = new WaitForUserDialog("Cell
1109 layer definition", "\nClick to define the border between the L2 and L3.\n");
1110            NucleiBorderSelection_Dialog.show();
1111        }
1112
1113        if (subcounter == 2) {
1114            WaitForUserDialog NucleiBorderSelection_Dialog = new WaitForUserDialog("Cell
1115 layer definition", "\nClick to define the right border of the first cell layer of L3.\n");
1116            NucleiBorderSelection_Dialog.show();
1117        }
1118
1119        boolean PlotPoint_selected = false;
1120
1121        while (!PlotPoint_selected) {
1122            activeImage.setSlice(1); //no clue why this is needed, but doesn't work otherwise
1123            int[] CursorValues = getCursorLocation(activeImageCanvas);
1124            int CursorX = CursorValues[0];
1125            int CursorModifiers = CursorValues[2];
1126
1127            if (CursorModifiers == 16) {
1128                CursorX_array[subcounter + 1] = CursorX;
1129                PlotXValues_array[subcounter + 1] = (int) Math.round((CursorX -
1130                    PlotDrawingFrameX) * activeImagePixelWidth);
1131
1132                PlotPoint_selected = true;
1133            }
1134
1135            if (subcounter == 0 || subcounter == 1) {
1136
1137                ProfilePlot = new Plot(PlotTitle, "Distance", "Gray Values");
1138                ProfilePlot.setColor(Color.GRAY);
1139                ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.LINE);
1140
1141                ProfilePlot.setColor(Color.BLACK);
1142                ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.CIRCLE);
1143
1144                ProfilePlot.setColor(Color.RED);
1145
1146                if (subcounter == 0) {
1147                    ProfilePlot.drawLine(PlotXValues_array[subcounter + 1], 0, PlotXValues_array[
1148                        subcounter + 1], selectProfilePlotYValues[PlotXValues_array[subcounter + 1]]);
1149
1150                if (subcounter == 1) {
1151                    ProfilePlot.drawLine(PlotXValues_array[subcounter], 0, PlotXValues_array[subcounter]
1152                        , selectProfilePlotYValues[PlotXValues_array[subcounter]]);
1153                    ProfilePlot.drawLine(PlotXValues_array[subcounter + 1], 0, PlotXValues_array[
1154                        subcounter + 1], selectProfilePlotYValues[PlotXValues_array[subcounter + 1]]);
1155
1156                    int SecondNucleus_distance = CursorX_array[2] - CursorX_array[1];
1157
1158                    CursorX_array[0] = CursorX_array[1] - SecondNucleus_distance;
1159
1160                    if (CursorX_array[0] < PlotDrawingFrameX) {
1161                        CursorX_array[0] = PlotDrawingFrameX;
1162
1163                    PlotXValues_array[0] = (int) Math.round((CursorX_array[0] - PlotDrawingFrameX) *
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3097
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
314
```

```

1161 activeImagePixelWidth);
1162
1163     ProfilePlot.drawLine(PlotXValues_array[0], 0, PlotXValues_array[0],
1164     selectProfilePlotYValues[PlotXValues_array[0]]);
1165 }
1166
1167 if (subcounter == 0 || subcounter == 1) {
1168
1169     ProfilePlot.setFrozen(true);
1170     ProfilePlot.show();
1171
1172     activeImage.close();
1173
1174     activeImage = ProfilePlot.getImagePlus();
1175     activeImageCanvas = activeImage.getCanvas();
1176     activeImagePixelWidth = activeImage.getCalibration().pixelWidth;
1177     PlotDrawingFrameX = ProfilePlot.getDrawingFrame().x;
1178 }
1179
1180 activeImage.close();
1181
1182
1183 ProfilePlot = new Plot(title + "_profile-plot" + peripheral_analysis_name_variabel + ".tif"
1184 , "Distance", "Gray Values");
1185
1186 if (!one_channel) {
1187
1188     ProfilePlot.setColor(Color.LIGHT_GRAY);
1189     ProfilePlot.addPoints(ProfilePlotXValues, supportProfilePlotYValues, Plot.LINE);
1190     ProfilePlot.addPoints(ProfilePlotXValues, supportProfilePlotYValues, Plot.CIRCLE);
1191
1192 }
1193
1194 ProfilePlot.setColor(Color.BLACK);
1195 ProfilePlot.addPoints(ProfilePlotXValues, mainProfilePlotYValues, Plot.LINE);
1196 ProfilePlot.addPoints(ProfilePlotXValues, mainProfilePlotYValues, Plot.CIRCLE);
1197
1198
1199 ProfilePlot.setColor(Color.RED);
1200
1201 for (int subcounter = 0; subcounter <= 3; subcounter++) {
1202
1203     if (mainProfilePlotYValues[PlotXValues_array[subcounter]] > supportProfilePlotYValues[
1204     PlotXValues_array[subcounter]]) {
1205
1206         ProfilePlot.drawLine(PlotXValues_array[subcounter], 0, PlotXValues_array[subcounter]
1207         , mainProfilePlotYValues[PlotXValues_array[subcounter]]);
1208     }
1209
1210     else{
1211
1212         ProfilePlot.drawLine(PlotXValues_array[subcounter], 0, PlotXValues_array[subcounter]
1213         , supportProfilePlotYValues[PlotXValues_array[subcounter]]);
1214     }
1215
1216     if (!one_channel) {
1217
1218         ProfilePlot.setPlotObjectLabel(1, "support channel");
1219         ProfilePlot.setPlotObjectLabel(3, "main channel");
1220     }
1221
1222     else { // = (!one_channel)
1223
1224         ProfilePlot.setPlotObjectLabel(1, "main channel");
1225     }
1226
1227     ProfilePlot.setColor(Color.BLACK);
1228     ProfilePlot.setLegend(" ", Plot.TOP_RIGHT);
1229

```

```

1229 ProfilePlot.show();
1230 ProfilePlot.setLimitsToFit(true);
1231 ProfilePlot.updateImage();
1232 ProfilePlot.setFrozen(true);
1233
1234 activeImage = ProfilePlot.getImagePlus();
1235 IJ.saveAs(activeImage, "Tiff", directory + title + "_profile-plot" +
peripheral_analysis_name_variabel);
1236 activeImage.close();
1237
1238
1239 // measurement of layer specific fluorescence intensity in the 2D image, based on the nuclei borders
selected
1240
1241 activeImage = IJ.openImage(directory + title + "_main-channel_unrolled-2D" +
peripheral_analysis_name_variabel + ".tif");
1242 activeImage.show();
1243
1244 activeImage.setRoi(0, PlotXValues_array[0], activeImage.getWidth(), PlotXValues_array[1] -
PlotXValues_array[0]);
1245 ROI.addRoi(activeImage.getRoi());
1246 ROI.select(0);
1247 ROI.runCommand("Rename", "1st nucleus");
1248 IJ.run("Set Measurements...", "integrated redirect=None decimal=3");
1249 IJ.run("Measure", "");
1250 ResultsTable firstNucleus_results_table = ResultsTable.getResultsTable();
1251 double firstNucleus_intensity = firstNucleus_results_table.getValue("RawIntDen", 0);
1252 IJ.run("Clear Results");
1253 IJ.run(activeImage, "Select None", "");
1254
1255
1256 activeImage.setRoi(0, PlotXValues_array[1], activeImage.getWidth(), PlotXValues_array[2] -
PlotXValues_array[1]);
1257 ROI.addRoi(activeImage.getRoi());
1258 ROI.select(1);
1259 ROI.runCommand("Rename", "2nd nucleus");
1260 IJ.run("Set Measurements...", "integrated redirect=None decimal=3");
1261 IJ.run("Measure", "");
1262 ResultsTable secondNucleus_results_table = ResultsTable.getResultsTable();
1263 double secondNucleus_intensity = secondNucleus_results_table.getValue("RawIntDen", 0);
1264 IJ.run("Clear Results");
1265 IJ.run(activeImage, "Select None", "");
1266
1267
1268 activeImage.setRoi(0, PlotXValues_array[2], activeImage.getWidth(), PlotXValues_array[3] -
PlotXValues_array[2]);
1269 ROI.addRoi(activeImage.getRoi());
1270 ROI.select(2);
1271 ROI.runCommand("Rename", "3rd nucleus");
1272 IJ.run("Set Measurements...", "integrated redirect=None decimal=3");
1273 IJ.run("Measure", "");
1274 ResultsTable thirdNucleus_results_table = ResultsTable.getResultsTable();
1275 double thirdNucleus_intensity = thirdNucleus_results_table.getValue("RawIntDen", 0);
1276 IJ.run("Clear Results");
1277 IJ.run(activeImage, "Select None", "");
1278
1279 ROI.runCommand("Save", directory + title + "_nuclei-borders_ROIs.zip");
1280 ROI.reset();
1281
1282 activeImageProcessor = activeImage.getProcessor();
1283 activeImageProcessor.setColor(Color.white);
1284
1285 for (int subcounter = 0; subcounter <= 3; subcounter++) {
1286
1287 activeImageProcessor.drawLine(0, PlotXValues_array[subcounter], activeImage.getWidth(),
PlotXValues_array[subcounter]);
1288 }
1289
1290 IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_unrolled-2D" +
peripheral_analysis_name_variabel + "_nuclei-borders");
1291 activeImage.close();
1292
1293

```

```

1294     if (!one_channel) {
1295
1296         activeImage = IJ.openImage(directory + title + "_support-channel_unrolled-2D" +
peripheral_analysis_name_variabel + ".tif");
1297         activeImage.show();
1298
1299         activeImageProcessor = activeImage.getProcessor();
1300         activeImageProcessor.setColor(Color.white);
1301
1302         for (int subcounter = 0; subcounter <= 3; subcounter++) {
1303
1304             activeImageProcessor.drawLine(0, PlotXValues_array[subcounter], activeImage.
getWidth(), PlotXValues_array[subcounter]);
1305         }
1306
1307         IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-2D" +
peripheral_analysis_name_variabel + "_nuclei-borders");
1308         activeImage.close();
1309     }
1310
1311
1312 // calculation of different intensity ratios and storage of all data in a results table to be saved
1313
1314     double firstsecondthirdNucleus_intensity = firstNucleus_intensity +
secondNucleus_intensity + thirdNucleus_intensity;
1315     double firstNucleusPercentage = firstNucleus_intensity * 100 /
firstsecondthirdNucleus_intensity;
1316     double secondNucleusPercentage = secondNucleus_intensity * 100 /
firstsecondthirdNucleus_intensity;
1317     double thirdNucleusPercentage = thirdNucleus_intensity * 100 /
firstsecondthirdNucleus_intensity;
1318
1319
1320     ResultsTable AnalysisData = new ResultsTable();
1321
1322     AnalysisData.setValue(" ", 0, title + " [intensity]");
1323     AnalysisData.setValue(" ", 1, title + " [intensity %]");
1324     AnalysisData.setValue(" ", 2, " ");
1325     AnalysisData.setValue(" ", 3, "ratios for pWUS (L3 expression)");
1326     AnalysisData.setValue(" ", 4, title + " [intensity ratio]");
1327     AnalysisData.setValue(" ", 5, " ");
1328     AnalysisData.setValue(" ", 6, "ratios for pML1 (L1 expression)");
1329     AnalysisData.setValue(" ", 7, title + " [intensity ratio]");
1330
1331     AnalysisData.setValue("1st Nucleus", 0, firstNucleus_intensity);
1332     AnalysisData.setValue("1st Nucleus", 1, firstNucleusPercentage);
1333     AnalysisData.setValue("1st Nucleus", 2, " ");
1334     AnalysisData.setValue("1st Nucleus", 3, "1st to 3rd");
1335     AnalysisData.setValue("1st Nucleus", 4, firstNucleus_intensity / thirdNucleus_intensity);
1336     AnalysisData.setValue("1st Nucleus", 5, " ");
1337     AnalysisData.setValue("1st Nucleus", 6, "3rd to 1st");
1338     AnalysisData.setValue("1st Nucleus", 7, thirdNucleus_intensity / firstNucleus_intensity);
1339
1340     AnalysisData.setValue("2nd Nucleus", 0, secondNucleus_intensity);
1341     AnalysisData.setValue("2nd Nucleus", 1, secondNucleusPercentage);
1342     AnalysisData.setValue("2nd Nucleus", 2, " ");
1343     AnalysisData.setValue("2nd Nucleus", 3, "2nd to 3rd");
1344     AnalysisData.setValue("2nd Nucleus", 4, secondNucleus_intensity / thirdNucleus_intensity
);
1345
1346     AnalysisData.setValue("2nd Nucleus", 5, " ");
1347     AnalysisData.setValue("2nd Nucleus", 6, "2nd to 1st");
1348     AnalysisData.setValue("2nd Nucleus", 7, secondNucleus_intensity / firstNucleus_intensity
);
1349
1350     AnalysisData.setValue("3rd Nucleus", 0, thirdNucleus_intensity);
1351     AnalysisData.setValue("3rd Nucleus", 1, thirdNucleusPercentage);
1352     AnalysisData.setValue("3rd Nucleus", 2, " ");
1353     AnalysisData.setValue("3rd Nucleus", 3, "1st to 2nd");
1354     AnalysisData.setValue("3rd Nucleus", 4, firstNucleus_intensity / secondNucleus_intensity);
1355     AnalysisData.setValue("3rd Nucleus", 5, " ");
1356     AnalysisData.setValue("3rd Nucleus", 6, "3rd to 2nd");
1357     AnalysisData.setValue("3rd Nucleus", 7, thirdNucleus_intensity / secondNucleus_intensity
);

```

```

1357     AnalysisData.setValue("total", 0, firstsecondthirdNucleus_intensity);
1358     AnalysisData.setValue("total", 1, firstNucleusPercentage + secondNucleusPercentage +
1359     thirdNucleusPercentage);
1360     AnalysisData.setValue("total", 2, "");
1361     AnalysisData.setValue("total", 3, "3rd to 3rd");
1362     AnalysisData.setValue("total", 4, thirdNucleus_intensity / thirdNucleus_intensity);
1363     AnalysisData.setValue("total", 5, "");
1364     AnalysisData.setValue("total", 6, "1st to 1st");
1365     AnalysisData.setValue("total", 7, firstNucleus_intensity / firstNucleus_intensity);
1366
1367     AnalysisData.save(directory + title + "_analysis-data" + peripheral_analysis_name_variabel
1368     + ".csv");
1369
1370 // saving all plot data, including selected nuclei borders, in a results table
1371
1372     ResultsTable PlotData = new ResultsTable();
1373
1374     String[] BorderPoints = new String[ProfilePlotXValues.length];
1375
1376     for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {
1377
1378         if (ProfilePlotXValues[subcounter] == (PlotXValues_array[0]) || ProfilePlotXValues[
1379         subcounter] == (PlotXValues_array[1]) || ProfilePlotXValues[subcounter] == (PlotXValues_array[2]) ||
1380         ProfilePlotXValues[subcounter] == (PlotXValues_array[3])) {
1381             BorderPoints[subcounter] = "selected border";
1382         } else {
1383             BorderPoints[subcounter] = " ";
1384         }
1385
1386         for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {
1387             PlotData.setValue("X-value", subcounter, ProfilePlotXValues[subcounter]);
1388             PlotData.setValue("main-channel Y-value", subcounter, mainProfilePlotYValues[
1389             subcounter]);
1390
1391             if (!one_channel) {
1392                 PlotData.setValue("support-channel Y-value", subcounter,
1393                 supportProfilePlotYValues[subcounter]);
1394             }
1395
1396             PlotData.save(directory + title + "_profile-plot-data" + peripheral_analysis_name_variabel
1397             + ".csv");
1398
1399             if (!peripheral_analysis) {
1400                 counter = 1;
1401             }
1402
1403             if (peripheral_analysis) {
1404
1405                 IJ.openImage(directory + title + "_profile-plot_periphery.tif").show();
1406             }
1407
1408             IJ.openImage(directory + title + "_profile-plot.tif").show();
1409
1410             WaitForUserDialog QuitPlugin_Dialog = new WaitForUserDialog("Quit Plugin", " \n
1411             Analysis has run successfully. Click 'OK' to quit the plugin. \n ");
1412             QuitPlugin_Dialog.show();
1413
1414             IJ.run("Close All", "");
1415
1416 // end of the main function
1417
1418
1419
1420 // secondary functions to be called in the main function above
1421

```

```

1422
1423 // function to check if the right amount of images is open
1424
1425 private boolean checkExit() {
1426
1427     boolean exit = false;
1428
1429     int ImageWindowNumber = WindowManager.getImageCount();
1430
1431     if (ImageWindowNumber == 0) {
1432         IJ.error("Error Message", "Please open an image to be analyzed.");
1433         exit = true;
1434     }
1435
1436     if (ImageWindowNumber != 0 && ImageWindowNumber != 1) {
1437         IJ.error("Error Message", "Please open only one image.");
1438         exit = true;
1439     }
1440
1441     return exit;
1442 }
1443
1444
1445 // dialog for selection of primary parameters
1446
1447 private boolean[] PrimaryParameters_Dialog() {
1448
1449
1450     GenericDialog PrimaryParameters_Dialog = new GenericDialog("Analysis Setup");
1451
1452
1453     PrimaryParameters_Dialog.centerDialog(true);
1454
1455     PrimaryParameters_Dialog.setInsets(10, 10, 5);
1456     PrimaryParameters_Dialog.addMessage("Image Acquisition Parameters:");
1457
1458     String[] acquisition_channel_number_choices = {"one channel", "two channels"};
1459     PrimaryParameters_Dialog.addRadioButtonGroup("Acquisition channels:",
1460     acquisition_channel_number_choices, 2, 1, "one channel");
1461
1462     String[] main_signal_channel_choices = {"channel 1", "channel 2"};
1463     PrimaryParameters_Dialog.addRadioButtonGroup("Main signal channel:",
1464     main_signal_channel_choices, 2, 1, "channel 1");
1465
1466     String[] Z_scan_direction_choices = {"into the tissue", "out of the tissue"};
1467     PrimaryParameters_Dialog.addRadioButtonGroup("Z-scan direction:",
1468     Z_scan_direction_choices, 2, 1, "into the tissue");
1469
1470     String[] meristem_orientation_choices = {"native side-view", "native top-view"};
1471     PrimaryParameters_Dialog.addRadioButtonGroup("Meristem orientation:",
1472     meristem_orientation_choices, 2, 1, "native side-view");
1473
1474
1475     PrimaryParameters_Dialog.setInsets(30, 10, 5);
1476     PrimaryParameters_Dialog.addMessage("Image Analysis Settings:");
1477
1478     String[] centroid_determination_choices = {"automatically", "manually"};
1479     PrimaryParameters_Dialog.addRadioButtonGroup("Domain column center point:",
1480     centroid_determination_choices, 2, 1, "automatically");
1481
1482
1483     PrimaryParameters_Dialog.setInsets(15, 5, 5);
1484     PrimaryParameters_Dialog.addCheckbox("Show advanced options", false);
1485
1486     PrimaryParameters_Dialog.setHelpLabel("Contact");
1487     PrimaryParameters_Dialog.addHelp("https://orcid.org/0000-0003-1307-4060");
1488     PrimaryParameters_Dialog.showDialog();
1489

```

```

1490 // boolean[] PrimaryParameters[0] = boolean one_channel = true = "one channel"
1491 // boolean[] PrimaryParameters[0] = boolean one_channel = false = "two channels"
1492
1493 if (PrimaryParameters_Dialog.getNextRadioButton().equals("one channel")) {
1494     PrimaryParameters[0] = true;
1495 }
1496
1497
1498 // boolean[] PrimaryParameters[1] = boolean channel1_main = true = "channel 1"
1499 // boolean[] PrimaryParameters[1] = boolean channel1_main = false = "channel 2"
1500
1501 if (PrimaryParameters_Dialog.getNextRadioButton().equals("channel 1")) {
1502     PrimaryParameters[1] = true;
1503 }
1504
1505
1506 // boolean[] PrimaryParameters[2] = boolean Z_scan_into_tissue = true = "into the tissue"
1507 // boolean[] PrimaryParameters[2] = boolean bottom_to_top = false = "out from the tissue"
1508
1509 if (PrimaryParameters_Dialog.getNextRadioButton().equals("into the tissue")) {
1510     PrimaryParameters[2] = true;
1511 }
1512
1513
1514 // boolean[] PrimaryParameters[3] = boolean horizontal = true = "native side-view"
1515 // boolean[] PrimaryParameters[3] = boolean horizontal = false = "native top-view"
1516
1517 if (PrimaryParameters_Dialog.getNextRadioButton().equals("native side-view")) {
1518     PrimaryParameters[3] = true;
1519 }
1520
1521
1522 // boolean[] PrimaryParameters[4] = boolean automatic_centroid = true = "automatic
determination"
1523 // boolean[] PrimaryParameters[4] = boolean automatic_centroid = false = "manual
determination"
1524
1525 if (PrimaryParameters_Dialog.getNextRadioButton().equals("automatically")) {
1526     PrimaryParameters[4] = true;
1527 }
1528
1529
1530 // boolean[] PrimaryParameters[5] = boolean advancedOptions = true = checkbox checked
1531 // boolean[] PrimaryParameters[5] = boolean advancedOptions = false = checkbox unchecked
1532
1533 PrimaryParameters[5] = PrimaryParameters_Dialog.getNextBoolean();
1534
1535
1536 if (PrimaryParameters_Dialog.wasCanceled()){
1537     PrimaryParameters[6] = true;
1538     WindowManager.getCurrentImage().close();
1539 }
1540
1541 return PrimaryParameters;
1542 }
1543
1544
1545 // dialog for selection of secondary parameters
1546
1547 private double[] SecondaryParameters_Dialog(boolean automatic_centroid, boolean
advancedOptions) {
1548
1549     GenericDialog SecondaryParameters_Dialog = new GenericDialog("Analysis Setup");
1550
1551     if (automatic_centroid) {
1552         SecondaryParameters_Dialog.setInsets(10, 10, 0);
1553         SecondaryParameters_Dialog.addMessage("Upper Threshold: ");
1554         SecondaryParameters_Dialog.setInsets(5, 10, 7);
1555         SecondaryParameters_Dialog.addNumericField("", 4095, 0, 5, "intensity value");
1556
1557         SecondaryParameters_Dialog.setInsets(10, 10, 0);
1558         SecondaryParameters_Dialog.addMessage("Lower threshold: ");
1559         SecondaryParameters_Dialog.setInsets(5, 10, 7);

```

```

1560 SecondaryParameters_Dialog.addNumericField(" ", 35, 0, 3, "% of upper threshold");
1561
1562 SecondaryParameters_Dialog.setInsets(10, 10, 0);
1563 SecondaryParameters_Dialog.addMessage("3D median filter: ");
1564 SecondaryParameters_Dialog.setInsets(5, 10, 7);
1565 SecondaryParameters_Dialog.addNumericField(" ", 1, 1, 3, "radius");
1566 }
1567
1568 SecondaryParameters_Dialog.setInsets(10, 10, 0);
1569 SecondaryParameters_Dialog.addMessage("Domain diameter: ");
1570 SecondaryParameters_Dialog.setInsets(5, 10, 7);
1571 SecondaryParameters_Dialog.addNumericField(" ", 20, 0, 3, "microns");
1572
1573 if (!automatic_centroid) {
1574     SecondaryParameters_Dialog.setInsets(10, 10, 0);
1575     String[] peripheral_analysis_choices = {"yes", "no"};
1576     SecondaryParameters_Dialog.addRadioButtonGroup("Peripheral Analysis:", peripheral_analysis_choices, 2, 1, "yes");
1577 }
1578 SecondaryParameters_Dialog.setInsets(10, 10, 0);
1579 SecondaryParameters_Dialog.addMessage("Periphery width: ");
1580 SecondaryParameters_Dialog.setInsets(5, 10, 7);
1581 SecondaryParameters_Dialog.addNumericField(" ", 10, 0, 3, "microns");
1582 }
1583
1584 SecondaryParameters_Dialog.setInsets(10, 10, 0);
1585 SecondaryParameters_Dialog.addMessage("Circle point selection delay: ");
1586 SecondaryParameters_Dialog.setInsets(5, 10, 7);
1587 SecondaryParameters_Dialog.addNumericField(" ", 200, 0, 5, "milliseconds");
1588
1589
1590 SecondaryParameters_Dialog.setHelpLabel("Contact");
1591 SecondaryParameters_Dialog.addHelp("https://orcid.org/0000-0003-1307-4060");
1592
1593 if (advancedOptions) {
1594     SecondaryParameters_Dialog.showDialog();
1595 }
1596
1597
1598 double[] SecondaryParameters = new double[8];
1599
1600 if (automatic_centroid) {
1601     SecondaryParameters[0] = SecondaryParameters_Dialog.getNextNumber();
1602     SecondaryParameters[1] = SecondaryParameters_Dialog.getNextNumber();
1603     SecondaryParameters[2] = SecondaryParameters_Dialog.getNextNumber();
1604 }
1605
1606 else {
1607     SecondaryParameters[0] = 0;
1608     SecondaryParameters[1] = 0;
1609     SecondaryParameters[2] = 0;
1610 }
1611
1612 SecondaryParameters[3] = SecondaryParameters_Dialog.getNextNumber();
1613
1614 if (!automatic_centroid) {
1615
1616     if (SecondaryParameters_Dialog.getNextRadioButton().equals("yes")) {
1617         SecondaryParameters[4] = 1;
1618     }
1619
1620     else {
1621         SecondaryParameters[4] = 0;
1622     }
1623
1624     SecondaryParameters[5] = SecondaryParameters_Dialog.getNextNumber();
1625 }
1626
1627 else {
1628     SecondaryParameters[4] = 0;
1629     SecondaryParameters[5] = 0;
1630 }
1631

```

```

1632 SecondaryParameters[6] = SecondaryParameters_Dialog.getNextNumber();
1633
1634 SecondaryParameters[7] = 0;
1635
1636 if (SecondaryParameters_Dialog.wasCanceled()){
1637     SecondaryParameters[7] = 1;
1638     WindowManager.getCurrentImage().close();
1639 }
1640
1641 return SecondaryParameters;
1642 }
1643
1644
1645 // creation of a new folder at the safe directory, including naming and check for pre-existing folders
1646 // with the same name
1647 private String[] createDirectory(String title) {
1648
1649     String [] createDirectory = new String[3];
1650
1651     String updated_title = title;
1652
1653     String folder_directory = IJ.getDirectory("Choose Folder Directory") + title;
1654     File folder = new File(folder_directory);
1655     boolean folder_exists_already = folder.exists();
1656
1657     int folder_directory_appendix = 2;
1658     String updated_folder_directory = folder_directory;
1659
1660     while (folder_exists_already) {
1661         updated_folder_directory = folder_directory + " " + folder_directory_appendix;
1662         updated_title = title + " " + folder_directory_appendix;
1663         folder = new File(updated_folder_directory);
1664         folder_exists_already = folder.exists();
1665
1666         if (folder_exists_already) {
1667             folder_directory_appendix++;
1668         }
1669     }
1670
1671     boolean folder_successfully_created = folder.mkdir();
1672
1673     String folder_status = "created";
1674
1675     if (!folder_successfully_created) {
1676         IJ.error("Error Message", "Folder could not be created.");
1677         folder_status = "not created";
1678     }
1679
1680     createDirectory[0] = updated_folder_directory + System.getProperty("file.separator");
1681     createDirectory[1] = updated_title;
1682     createDirectory[2] = folder_status;
1683
1684     return createDirectory;
1685 }
1686
1687
1688 // function for a rotation from horizontal to vertical using TransformJ Turn
1689
1690 private ImagePlus rotate_horizontal2vertical(ImagePlus activeImage, boolean Z_scan_into_tissue)
1691 {
1692
1693     if (Z_scan_into_tissue) {
1694         IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=90");
1695     }
1696
1697     if (!Z_scan_into_tissue) {
1698         IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=270");
1699     }
1700
1701     activeImage.changes = false;
1702     activeImage.close();
1703     activeImage = WindowManager.getCurrentImage();

```

```

1703
1704     IJ.run(activeImage, "TransformJ Turn", "z-angle=180 y-angle=0 x-angle=0");
1705
1706     activeImage.changes = false;
1707     activeImage.close();
1708     activeImage = WindowManager.getCurrentImage();
1709
1710     return activeImage;
1711 }
1712
1713
1714 // function for a rotation from vertical to horizontal using TransformJ Turn
1715
1716 private ImagePlus rotate_vertical2horizontal(ImagePlus activeImage, boolean Z_scan_into_tissue)
{
1717
1718     IJ.run(activeImage, "TransformJ Turn", "z-angle=180 y-angle=0 x-angle=0");
1719
1720     activeImage.changes = false;
1721     activeImage.close();
1722     activeImage = WindowManager.getCurrentImage();
1723
1724     if (Z_scan_into_tissue) {
1725         IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=270");
1726     }
1727
1728     if (!Z_scan_into_tissue) {
1729         IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=90");
1730     }
1731
1732     activeImage.changes = false;
1733     activeImage.close();
1734     activeImage = WindowManager.getCurrentImage();
1735
1736     return activeImage;
1737 }
1738
1739
1740 // function for a maximum-projection
1741
1742 private ImagePlus maxProjection(ImagePlus activeImage) {
1743
1744     ImagePlus projection = ZProjector.run(activeImage, "max");
1745     projection.show();
1746     activeImage.changes = false;
1747     activeImage.close();
1748
1749     return projection;
1750 }
1751
1752
1753 // function for a sum-projection
1754
1755 private ImagePlus sumProjection(ImagePlus activeImage) {
1756
1757     ImagePlus projection = ZProjector.run(activeImage, "sum");
1758     IJ.run(projection, "royal", "");
1759     projection.show();
1760     activeImage.changes = false;
1761     activeImage.close();
1762     return projection;
1763 }
1764
1765
1766 // function to get the cursor position within an image canvas
1767
1768 private int[] getCursorLocation(ImageCanvas activeImageCanvas) {
1769
1770     Point CursorPosition = activeImageCanvas.getCursorPosition();
1771
1772     int [] CursorValues = new int[3];
1773
1774     CursorValues[0] = CursorPosition.x;

```

```

1775     CursorValues[1] = CursorPosition.y;
1776     CursorValues[2] = activeImageCanvas.getModifiers();
1777
1778     return CursorValues;
1779 }
1780
1781
1782 // function for polar transformation
1783
1784     private ImagePlus transform_polarTransformation(ImagePlus inputImage, int stack_size, double[]
1785     CenterPointX_array, double[] CenterPointY_array, double[] radius_array) {
1786
1787     ImageStack inputImageStack = inputImage.getStack();
1788     int inputImage_width = inputImage.getWidth();
1789     int inputImage_height = inputImage.getHeight();
1790
1791     double[][][] radius_unrolled_array = new double[stack_size][inputImage_height][
1792     inputImage_width];
1793     double[][][] phi_array = new double[stack_size][inputImage_height][inputImage_width];
1794
1795     for (int new_z = 0; new_z < stack_size; new_z++) {
1796
1797         for (int new_y = 0; new_y < inputImage_height; new_y++) {
1798
1799             for (int new_x = 0; new_x < inputImage_width; new_x++) {
1800
1801                 radius_unrolled_array[new_z][new_y][new_x] = Math.sqrt(Math.pow((new_x -
1802                 CenterPointX_array[new_z]), 2) + Math.pow((new_y - CenterPointY_array[new_z]), 2));
1803                 phi_array[new_z][new_y][new_x] = Math.atan2((new_y - CenterPointY_array[new_z]), (
1804                 new_x - CenterPointX_array[new_z]));
1805             }
1806         }
1807     }
1808
1809     int[][][] temp_unrolled_y = new int[stack_size][inputImage_height][inputImage_width];
1810     int[][][] temp_unrolled_x = new int[stack_size][inputImage_height][inputImage_width];
1811     double [[[value]]] value = new double[stack_size][inputImage_height][inputImage_width];
1812     int temp_unrolledImage_height = 10000000;
1813     int temp_unrolledImage_width = 5000000;
1814
1815     for (int new_z = 0; new_z < stack_size; new_z++) {
1816
1817         for (int new_y = 0; new_y < inputImage_height; new_y++) {
1818
1819             for (int new_x = 0; new_x < inputImage_width; new_x++) {
1820
1821                 value[new_z][new_y][new_x] = inputImageStack.getVoxel(new_x, new_y, new_z);
1822
1823                 if (value[new_z][new_y][new_x] != 0) {
1824
1825                     temp_unrolled_y[new_z][new_y][new_x] = (int) Math.round(
1826                     temp_unrolledImage_height / 2 - (radius_unrolled_array[new_z][new_y][new_x] - radius_array[new_z]));
1827
1828                     temp_unrolled_x[new_z][new_y][new_x] = (int) Math.round(
1829                     temp_unrolledImage_width / 2 + phi_array[new_z][new_y][new_x] * radius_array[new_z]);
1830
1831             }
1832
1833         }
1834
1835         double min_temp_unrolled_y = temp_unrolledImage_height;
1836         double max_temp_unrolled_y = 0;
1837         double min_temp_unrolled_x = temp_unrolledImage_width;
1838         double max_temp_unrolled_x = 0;
1839
1840         for (int new_z = 0; new_z < stack_size; new_z++) {
1841
1842             for (int new_y = 0; new_y < inputImage_height; new_y++) {
1843
1844                 for (int new_x = 0; new_x < inputImage_width; new_x++) {

```

```

1841     if (max_temp_unrolled_y < temp_unrolled_y[new_z][new_y][new_x]) {
1842         max_temp_unrolled_y = temp_unrolled_y[new_z][new_y][new_x];
1843     }
1844
1845     if (min_temp_unrolled_y > temp_unrolled_y[new_z][new_y][new_x] &&
1846 temp_unrolled_y[new_z][new_y][new_x] != 0) {
1847         min_temp_unrolled_y = temp_unrolled_y[new_z][new_y][new_x];
1848     }
1849
1850     if (max_temp_unrolled_x < temp_unrolled_x[new_z][new_y][new_x]) {
1851         max_temp_unrolled_x = temp_unrolled_x[new_z][new_y][new_x];
1852     }
1853
1854     if (min_temp_unrolled_x > temp_unrolled_x[new_z][new_y][new_x] &&
1855 temp_unrolled_x[new_z][new_y][new_x] != 0) {
1856         min_temp_unrolled_x = temp_unrolled_x[new_z][new_y][new_x];
1857     }
1858 }
1859 }
1860
1861 int unrolledImage_height = (int) Math.round(max_temp_unrolled_y - min_temp_unrolled_y) +
1;
1862 int unrolledImage_width = (int) Math.round(max_temp_unrolled_x - min_temp_unrolled_x) +
1;
1863
1864 int[][][] unrolled_y_array = new int[stack_size][inputImage_height][inputImage_width];
1865 int[][][] unrolled_x_array = new int[stack_size][inputImage_height][inputImage_width];
1866
1867 for (int new_z = 0; new_z < stack_size; new_z++) {
1868
1869     for (int new_y = 0; new_y < inputImage_height; new_y++) {
1870
1871         for (int new_x = 0; new_x < inputImage_width; new_x++) {
1872
1873             if (value[new_z][new_y][new_x] != 0) {
1874
1875                 unrolled_y_array[new_z][new_y][new_x] = (int) Math.round(unrolledImage_height /
2 - (radius_unrolled_array[new_z][new_y][new_x] - radius_array[new_z]));
1876                 unrolled_x_array[new_z][new_y][new_x] = (int) Math.round(unrolledImage_width /
2 + phi_array[new_z][new_y][new_x] * radius_array[new_z]);
1877             }
1878         }
1879     }
1880 }
1881
1882 int min_unrolled_y = unrolledImage_height;
1883 int min_unrolled_x = unrolledImage_width;
1884
1885 for (int new_z = 0; new_z < stack_size; new_z++) {
1886
1887     for (int new_y = 0; new_y < inputImage_height; new_y++) {
1888
1889         for (int new_x = 0; new_x < inputImage_width; new_x++) {
1890
1891             if (min_unrolled_y > unrolled_y_array[new_z][new_y][new_x] && unrolled_y_array[
new_z][new_y][new_x] != 0) {
1892                 min_unrolled_y = unrolled_y_array[new_z][new_y][new_x];
1893             }
1894
1895             if (min_unrolled_x > unrolled_x_array[new_z][new_y][new_x] && unrolled_x_array[
new_z][new_y][new_x] != 0) {
1896                 min_unrolled_x = unrolled_x_array[new_z][new_y][new_x];
1897             }
1898         }
1899     }
1900 }
1901
1902 ImagePlus unrolledImage = IJ.createImage("Output Image", unrolledImage_width,
unrolledImage_height, stack_size, 16);
1903 ImageStack unrolledImageStack = unrolledImage.getStack();
1904

```

```
1905     for (int new_z = 0; new_z < stack_size; new_z++) {  
1906         for (int new_y = 0; new_y < inputImage_height; new_y++) {  
1908             for (int new_x = 0; new_x < inputImage_width; new_x++) {  
1910                 if (value[new_z][new_y][new_x] != 0) {  
1912                     int unrolled_y = -1 * min_unrolled_y + ((int) Math.round(unrolledImage_height / 2  
1913                         - (radius_unrolled_array[new_z][new_y][new_x] - radius_array[new_z])));  
1914                     int unrolled_x = -1 * min_unrolled_x + ((int) Math.round(unrolledImage_width / 2  
1915                         + phi_array[new_z][new_y][new_x] * radius_array[new_z]));  
1916  
1917                     if (unrolledImageStack.getVoxel(unrolled_x, unrolled_y, new_z) == 0) {  
1918                         unrolledImageStack.setVoxel(unrolled_x, unrolled_y, new_z, value[new_z][  
1919                             new_y][new_x]);  
1920                     } else {  
1921                         double existingPixel = unrolledImageStack.getVoxel(unrolled_x, unrolled_y,  
1922                             new_z);  
1923                         unrolledImageStack.setVoxel(unrolled_x, unrolled_y, new_z, existingPixel +  
1924                             value[new_z][new_y][new_x]);  
1925                     }  
1926                 }  
1927             inputImage.close();  
1928         return unrolledImage;  
1929     }  
1930 }  
1931 }
```