```
// import statements

import ij.IJ;
import ij.ImagePlus;
import ij.ImageStack;
import ij.WindowManager;
import ij.gui.*;
import ij.measure.Calibration;
import ij.measure.ResultsTable;
import ij.plugin.Duplicator;
import ij.plugin.PlugIn;
import ij.plugin.ZProjector;
import ij.plugin.frame.RoiManager;
import ij.process.ImageProcessor;

import java.awt.*;
import java.io.File;


// InsertFancyNameHere (IFNH) plugin for Fiji (tested with Fiji version 1.52p)

public class InsertFancyNameHere implements PlugIn {


// main plugin function, secondary functions called within can be found below

    public void run(String arg) {


// check for the number of open images: plugin continues only if exactly one image is open

        if (checkExit()) {
            return;
        }


// primary parameters for subsequent analysis: user input, plugin aborts if dialog is cancelled

        boolean[] PrimaryParameters = PrimaryParameters_Dialog();
        boolean one_channel = PrimaryParameters[0];
        boolean channel1_main = PrimaryParameters[1];
        boolean Z_scan_into_tissue = PrimaryParameters[2];
        boolean horizontal = PrimaryParameters[3];
        boolean automatic_centroid = PrimaryParameters[4];
        boolean advancedOptions = PrimaryParameters[5];
        boolean PrimaryParameters_Dialog_cancelled = PrimaryParameters[6];

        if (PrimaryParameters_Dialog_cancelled) {
            return;
        }


// secondary parameters for subsequent analysis: user input, plugin aborts if dialog is cancelled

        double[] SecondaryParameters = SecondaryParameters_Dialog(automatic_centroid,
advancedOptions);
        double upper_threshold = SecondaryParameters[0];
        double lower_threshold = SecondaryParameters[1] * upper_threshold / 100;
        double filter_radius = SecondaryParameters[2];

        double domain_diameter = SecondaryParameters[3];

        boolean peripheral_analysis = false;
        double peripheral_analysis_double = SecondaryParameters[4];

        if (peripheral_analysis_double == 1) {
            peripheral_analysis = true;
        }

        double periphery_width = SecondaryParameters[5];

        double double_CirclePointSelectionDelay = SecondaryParameters[6];
        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 excisting 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
    primary parameters
104
105         if (one_channel) {
106             IJ.run(activeImage, "royal", "");
107             IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel");
108         }
109
110         else { // = (!one_channel)
111             activeImage.setTitle(title + ".tif");
112             IJ.run(activeImage, "Split Channels", "");
113
114             if (channel1_main) {
115                 activeImage = WindowManager.getImage("C2-" + title + ".tif");
116                 IJ.run(activeImage, "royal", "");
117                 IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel");
118                 activeImage.changes = false;
119                 activeImage.close();
120
121                 activeImage = WindowManager.getImage("C1-" + title + ".tif");
122                 IJ.run(activeImage, "royal", "");
123                 IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel");
124             }
125
126             else { // = (!channel1_main)
127                 activeImage = WindowManager.getImage("C1-" + title + ".tif");
128                 IJ.run(activeImage, "royal", "");
129                 IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel");
130                 activeImage.changes = false;
131                 activeImage.close();
132
133                 activeImage = WindowManager.getImage("C2-" + title + ".tif");
134                 IJ.run(activeImage, "royal", "");
135                 IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel");
136             }
137         }
138
139
140 // rotation of images to create a top-view image (if not already defined as top-view, based on previous
    user selection of primary parameters)
141
142         if (horizontal) {
143
```

```
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
    selection, based on previous user selection of primary parameters)
164
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
    selected by the user)
181
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=" +
     filter_radius);
204
205             activeImage = maxProjection(activeImage);
206             IJ.run(activeImage, "royal", "");
207
208             IJ.setThreshold(activeImage, lower_threshold, upper_threshold);
209             IJ.run(activeImage, "Analyze Particles...", "size=10-Infinity show=Masks include clear
    include add");
210
211             if (horizontal) {
212                 WindowManager.getImage("Mask of MAX_" + title + "_temporary1.tif").close();
```

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

```
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 + "
    z=" + filter_radius);
317
318                 activeImage = maxProjection(activeImage);
319                 IJ.run(activeImage, "royal", "");
320
321
322                 IJ.setThreshold(activeImage, lower_threshold, upper_threshold);
323                 IJ.run(activeImage, "Analyze Particles...", "size=10-Infinity show=Masks include
    clear include add");
324
325                 if (horizontal) {
326                     WindowManager.getImage("Mask of MAX_" + title + "_temporary1.tif").close();
327                 }
328
329                 else { // = (!horizontal)
330                     WindowManager.getImage("Mask of MAX_" + title + "_main-channel.tif").close(
    );
331                 }
332
333                 WindowManager.getCurrentImage().setTitle("ROI selection window - " + (int)Math.
    round(new_percentage) + " %");
334
335                 ROI.deselect();
336                 number_ROIs = ROI.getCount();
337
338                 columns = 3;
339                 rows_remainder = number_ROIs % columns;
340                 rows = 0;
341
342                 if (rows_remainder == 1 || rows_remainder == 2) {
343                     rows = number_ROIs / columns + 1;
344                 }
345
346                 if (rows_remainder == 0) {
347                     rows = number_ROIs / columns;
348                 }
349
350                 CheckboxLabels = new String[number_ROIs];
351                 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
    analysis domain
412
413         if (horizontal) {
414             activeImage = IJ.openImage(directory + title + "_temporary1.tif");
415         }
416
417         else { // = (!horizontal)
418             activeImage = IJ.openImage(directory + title + "_main-channel.tif");
419         }
420
421         activeImage.show();
422
423         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
    selected by the user)
455
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
    Definition", " \nClick to define the center point of the analysis domain. \n ");
465             CentroidSelection_Dialog.show();
466
467             boolean CenterPoint_selected = false;
468
469             while (!CenterPoint_selected) {
470
471                 int[] CursorValues = getCursorLocation(activeImageCanvas);
472                 int CursorX = CursorValues[0];
473                 int CursorY = CursorValues[1];
474                 int CursorModifiers = CursorValues[2];
475
476                 if (CursorModifiers == 16) {
477                     IJ.setForegroundColor(255, 255, 255);
478                     activeImageProcessor.fillOval(CursorX - 2, CursorY - 2, 5, 5);
479                     int_CenterPointX_pixels = CursorX;
480                     int_CenterPointY_pixels = CursorY;
481                     CenterPoint_selected = true;
482                 }
483
484                 activeImage.updateAndDraw();
485             }
486
487             double_CenterPointX_pixels = int_CenterPointX_pixels;
488             double_CenterPointY_pixels = int_CenterPointY_pixels;
489
490             activeImageCalibration = activeImage.getCalibration();
491             voxel_width = activeImageCalibration.pixelWidth;
492             voxel_height = activeImageCalibration.pixelHeight;
493
```

```
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
    defined center point
500
501         IJ.run(activeImage, "Specify...", "width=" + domain_diameter + " height=" +
    domain_diameter + " x=" + double_CenterPointX_microns + " y=" + double_CenterPointY_microns
     + " slice=1 oval centered scaled");
502
503         ROI.addRoi(activeImage.getRoi());
504         ROI.select(0);
505
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
    the user
513
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=" +
    peripheral_domain_diameter + " x=" + double_CenterPointX_microns + " y=" +
    double_CenterPointY_microns + " slice=1 oval centered scaled");
519
520             ROI.addRoi(activeImage.getRoi());
521             ROI.select(1);
522
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
    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-
    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
778             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)
    ) * (CirclePointY_2 - CirclePointY_3) + (Math.pow(CirclePointX_2, 2) + Math.pow(CirclePointY_2,
    2)) * (CirclePointY_3 - CirclePointY_1) + (Math.pow(CirclePointX_3, 2) + Math.pow(CirclePointY_3
    , 2)) * (CirclePointY_1 - CirclePointY_2)) / (2 * (CirclePointX_1 * (CirclePointY_2 - CirclePointY_3)
     + CirclePointX_2 * (CirclePointY_3 - CirclePointY_1) + CirclePointX_3 * (CirclePointY_1 -
    CirclePointY_2)));
899             CenterPointY_array[new_z] = ((Math.pow(CirclePointX_1, 2) + Math.pow(CirclePointY_1, 2)
    ) * (CirclePointX_3 - CirclePointX_2) + (Math.pow(CirclePointX_2, 2) + Math.pow(CirclePointY_2,
    2)) * (CirclePointX_1 - CirclePointX_3) + (Math.pow(CirclePointX_3, 2) + Math.pow(CirclePointY_3
    , 2)) * (CirclePointX_2 - CirclePointX_1)) / (2 * (CirclePointX_1 * (CirclePointY_2 - CirclePointY_3)
     + CirclePointX_2 * (CirclePointY_3 - CirclePointY_1) + CirclePointX_3 * (CirclePointY_1 -
    CirclePointY_2)));
900
901             radius_array[new_z] = (Math.sqrt(Math.pow((CirclePointX_1 - CenterPointX_array[new_z]), 2)
     + Math.pow((CirclePointY_1 - CenterPointY_array[new_z]), 2)) + Math.sqrt(Math.pow((
    CirclePointX_2 - CenterPointX_array[new_z]), 2) + Math.pow((CirclePointY_2 - CenterPointY_array[
    new_z]), 2)) + Math.sqrt(Math.pow((CirclePointX_3 - CenterPointX_array[new_z]), 2) + Math.pow((
    CirclePointY_3 - CenterPointY_array[new_z]), 2))) / 3;
902         }
```

```
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" +
    peripheral_analysis_name_variabel + ".tif");
927             activeImage.show();
928
929             for (int subcounter = 1; subcounter <= stack_size; subcounter++) {
930
931                 activeImage.setSlice(subcounter);
932
933                 if (CenterPointY_array[subcounter - 1] > activeImage.getHeight()) {
934                     activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.getWidth
    (), activeImage.getHeight() - highest_CirclePoint_Y[subcounter - 1]);
935
936                     if (highest_CirclePoint_Y[subcounter - 1] == 0) {
937                         activeImage.setRoi(0, 0, activeImage.getWidth(), activeImage.getHeight() -
    highest_CirclePoint_Y[subcounter - 1]);
938                     }
939
940                 } else {
941                     activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.getWidth
    (), (int) Math.round(CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter - 1]));
942
943                     if (highest_CirclePoint_Y[subcounter - 1] == 0) {
944                         activeImage.setRoi(0, 0, activeImage.getWidth(), (int) Math.round(
    CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter - 1]));
945                     }
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" +
    peripheral_analysis_name_variabel);
955
956             activeImage.hide();
957
958
959 // polar transformation
960
961             ImagePlus inputImage = activeImage;
962
963             activeImage = transform_polarTransformation(inputImage, stack_size, CenterPointX_array,
    CenterPointY_array, radius_array);
964             IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_unrolled-3D" +
    peripheral_analysis_name_variabel);
965
966
967 // 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             activeImage.close();
995
996
997 // repetition of polar transformation, dimension reduction and information extraction in case there
    was a secondary channel
998
999             if (!one_channel) {
1000
1001                 activeImage = IJ.openImage(directory + title + "_support-channel_not-unrolled" +
    peripheral_analysis_name_variabel + ".tif");
1002                 activeImage.show();
1003
1004                 for (int subcounter = 1; subcounter <= stack_size; subcounter++) {
1005
1006                     activeImage.setSlice(subcounter);
1007
1008                     if (CenterPointY_array[subcounter - 1] > activeImage.getHeight()) {
1009                         activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.
    getWidth(), activeImage.getHeight() - highest_CirclePoint_Y[subcounter - 1]);
1010
1011                         if (highest_CirclePoint_Y[subcounter - 1] == 0) {
1012                             activeImage.setRoi(0, 0, activeImage.getWidth(), activeImage.getHeight() -
    highest_CirclePoint_Y[subcounter - 1]);
1013                         }
1014
1015                     } else {
1016                         activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.
    getWidth(), (int) Math.round(CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter
    - 1]));
1017
1018                         if (highest_CirclePoint_Y[subcounter - 1] == 0) {
1019                             activeImage.setRoi(0, 0, activeImage.getWidth(), (int) Math.round(
    CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter - 1]));
1020                         }
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,
     CenterPointY_array, radius_array);
1036                IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-3D" +
    peripheral_analysis_name_variabel);
1037
1038                activeImage = sumProjection(activeImage);
1039                IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-2D" +
    peripheral_analysis_name_variabel);
1040
1041                IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=90 x-angle=0");
1042                activeImage.close();
1043                activeImage = WindowManager.getCurrentImage();
1044
1045                activeImage = sumProjection(activeImage);
1046                IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-1D");
1047
1048                activeImageProcessor = activeImage.getProcessor();
1049
1050                for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {
1051                    supportProfilePlotYValues[subcounter] = activeImageProcessor.getPixelValue(0,
    subcounter);
1052                }
1053
1054                activeImage.close();
1055
1056            }
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 +
     ".tif";
1072                System.arraycopy(mainProfilePlotYValues, 0, selectProfilePlotYValues, 0,
    ProfilePlotXValues.length);
1073            }
1074
1075            else { // = (!one_channel)
1076
1077                PlotTitle = title + "_support-channel_profile-plot" +
    peripheral_analysis_name_variabel + ".tif";
1078                System.arraycopy(supportProfilePlotYValues, 0, selectProfilePlotYValues, 0,
    ProfilePlotXValues.length);
1079            }
1080
1081            Plot ProfilePlot = new Plot(PlotTitle, "Distance", "Gray Values");
1082            ProfilePlot.setColor(Color.GRAY);
1083            ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.LINE);
1084
1085            ProfilePlot.setColor(Color.BLACK);
1086            ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.CIRCLE);
1087
1088            ProfilePlot.setFrozen(true);
1089            ProfilePlot.show();
1090
1091            activeImage = ProfilePlot.getImagePlus();
1092            activeImageCanvas = activeImage.getCanvas();
1093            activeImagePixelWidth = activeImage.getCalibration().pixelWidth;
1094
1095            PlotDrawingFrameX = ProfilePlot.getDrawingFrame().x;
```

```
1096
1097
1098            IJ.setTool("hand");
1099
1100            for (int subcounter = 0; subcounter <= 2; subcounter++) {
1101
1102                if (subcounter == 0) {
1103                    WaitForUserDialog NucleiBorderSelection_Dialog = new WaitForUserDialog("Cell
    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
    layer definition", " \nClick to define the border between the L2 and L3. \n ");
1109                    NucleiBorderSelection_Dialog.show();
1110                }
1111
1112                if (subcounter == 2) {
1113                    WaitForUserDialog NucleiBorderSelection_Dialog = new WaitForUserDialog("Cell
    layer definition", " \nClick to define the right border of the first cell layer of L3. \n ");
1114                    NucleiBorderSelection_Dialog.show();
1115                }
1116
1117                boolean PlotPoint_selected = false;
1118
1119                while (!PlotPoint_selected) {
1120                    activeImage.setSlice(1); //no clue why this is needed, but doesn't work otherwise
1121                    int[] CursorValues = getCursorLocation(activeImageCanvas);
1122                    int CursorX = CursorValues[0];
1123                    int CursorModifiers = CursorValues[2];
1124
1125                    if (CursorModifiers == 16) {
1126                        CursorX_array[subcounter + 1] = CursorX;
1127                        PlotXValues_array[subcounter + 1] = (int) Math.round((CursorX -
    PlotDrawingFrameX) * activeImagePixelWidth);
1128
1129                        PlotPoint_selected = true;
1130                    }
1131                }
1132
1133                if (subcounter == 0 || subcounter == 1) {
1134
1135                    ProfilePlot = new Plot(PlotTitle, "Distance", "Gray Values");
1136                    ProfilePlot.setColor(Color.GRAY);
1137                    ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.LINE);
1138
1139                    ProfilePlot.setColor(Color.BLACK);
1140                    ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.CIRCLE);
1141
1142                    ProfilePlot.setColor(Color.RED);
1143                }
1144
1145                if (subcounter == 0) {
1146                    ProfilePlot.drawLine(PlotXValues_array[subcounter + 1], 0, PlotXValues_array[
    subcounter + 1], selectProfilePlotYValues[PlotXValues_array[subcounter + 1]]);
1147                }
1148
1149                if (subcounter == 1) {
1150                    ProfilePlot.drawLine(PlotXValues_array[subcounter], 0, PlotXValues_array[subcounter]
    , selectProfilePlotYValues[PlotXValues_array[subcounter]]);
1151                    ProfilePlot.drawLine(PlotXValues_array[subcounter + 1], 0, PlotXValues_array[
    subcounter + 1], selectProfilePlotYValues[PlotXValues_array[subcounter + 1]]);
1152
1153                    int SecondNucleus_distance = CursorX_array[2] - CursorX_array[1];
1154
1155                    CursorX_array[0] = CursorX_array[1] - SecondNucleus_distance;
1156
1157                    if (CursorX_array[0] < PlotDrawingFrameX) {
1158                        CursorX_array[0] = PlotDrawingFrameX;
1159                    }
1160
1161                    PlotXValues_array[0] = (int) Math.round((CursorX_array[0] - PlotDrawingFrameX) *
```

```
1161 activeImagePixelWidth);
1162
1163                     ProfilePlot.drawLine(PlotXValues_array[0], 0, PlotXValues_array[0],
     selectProfilePlotYValues[PlotXValues_array[0]]);
1164                 }
1165
1166                 if (subcounter == 0 || subcounter == 1) {
1167
1168                     ProfilePlot.setFrozen(true);
1169                     ProfilePlot.show();
1170
1171                     activeImage.close();
1172
1173                     activeImage = ProfilePlot.getImagePlus();
1174                     activeImageCanvas = activeImage.getCanvas();
1175                     activeImagePixelWidth = activeImage.getCalibration().pixelWidth;
1176                     PlotDrawingFrameX = ProfilePlot.getDrawingFrame().x;
1177                 }
1178             }
1179
1180             activeImage.close();
1181
1182
1183             ProfilePlot = new Plot(title + "_profile-plot" + peripheral_analysis_name_variabel + ".tif"
     , "Distance", "Gray Values");
1184
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[
     PlotXValues_array[subcounter]]) {
1204
1205                     ProfilePlot.drawLine(PlotXValues_array[subcounter], 0, PlotXValues_array[subcounter]
     , mainProfilePlotYValues[PlotXValues_array[subcounter]]);
1206                 }
1207
1208                 else{
1209
1210                     ProfilePlot.drawLine(PlotXValues_array[subcounter], 0, PlotXValues_array[subcounter]
     , supportProfilePlotYValues[PlotXValues_array[subcounter]]);
1211                 }
1212
1213             }
1214
1215             if (!one_channel) {
1216
1217                 ProfilePlot.setPlotObjectLabel(1, "support channel");
1218                 ProfilePlot.setPlotObjectLabel(3, "main channel");
1219             }
1220
1221             else { // = (!one_channel)
1222
1223                 ProfilePlot.setPlotObjectLabel(1, "main channel");
1224             }
1225
1226             ProfilePlot.setColor(Color.BLACK);
1227             ProfilePlot.setLegend("", Plot.TOP_RIGHT);
1228
```

```
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            AnalysisData.setValue("2nd Nucleus", 5, "");
1346            AnalysisData.setValue("2nd Nucleus", 6, "2nd to 1st");
1347            AnalysisData.setValue("2nd Nucleus", 7, secondNucleus_intensity / firstNucleus_intensity)
    ;
1348
1349            AnalysisData.setValue("3rd Nucleus", 0, thirdNucleus_intensity);
1350            AnalysisData.setValue("3rd Nucleus", 1, thirdNucleusPercentage);
1351            AnalysisData.setValue("3rd Nucleus", 2, "");
1352            AnalysisData.setValue("3rd Nucleus", 3, "1st to 2nd");
1353            AnalysisData.setValue("3rd Nucleus", 4, firstNucleus_intensity / secondNucleus_intensity);
1354            AnalysisData.setValue("3rd Nucleus", 5, "");
1355            AnalysisData.setValue("3rd Nucleus", 6, "3rd to 2nd");
1356            AnalysisData.setValue("3rd Nucleus", 7, thirdNucleus_intensity / secondNucleus_intensity)
    ;
```

```
1357
1358            AnalysisData.setValue("total", 0, firstsecondthirdNucleus_intensity);
1359            AnalysisData.setValue("total", 1, firstNucleusPercentage + secondNucleusPercentage +
    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
     + ".csv");
1368
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[
    subcounter] == (PlotXValues_array[1]) || ProfilePlotXValues[subcounter] == (PlotXValues_array[2]) ||
    ProfilePlotXValues[subcounter] == (PlotXValues_array[3])) {
1379                    BorderPoints[subcounter] = "selected border";
1380                } else {
1381                    BorderPoints[subcounter] = "";
1382                }
1383            }
1384
1385            for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {
1386                PlotData.setValue("X-value", subcounter, ProfilePlotXValues[subcounter]);
1387                PlotData.setValue("main-channel Y-value", subcounter, mainProfilePlotYValues[
    subcounter]);
1388
1389                if (!one_channel) {
1390                    PlotData.setValue("support-channel Y-value", subcounter,
    supportProfilePlotYValues[subcounter]);
1391                }
1392
1393                PlotData.setValue("", subcounter, BorderPoints[subcounter]);
1394            }
1395
1396            PlotData.save(directory + title + "_profile-plot-data" + peripheral_analysis_name_variabel
    + ".csv");
1397
1398            if (!peripheral_analysis) {
1399                counter = 1;
1400            }
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
    Analysis has run successfully. Click 'OK' to quit the plugin. \n ");
1411        QuitPlugin_Dialog.show();
1412
1413        IJ.run("Close All", "");
1414
1415  // end of the main function
1416
1417    }
1418
1419
1420  // secondary functions to be called in the main function above
1421
```

```
1422
1423  // function to check it 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:",
    acquisition_channel_number_choices, 2, 1, "one channel");
1460
1461          String[] main_signal_channel_choices = {"channel 1", "channel 2"};
1462          PrimaryParameters_Dialog.addRadioButtonGroup("Main signal channel:",
    main_signal_channel_choices, 2, 1, "channel 1");
1463
1464          String[] Z_scan_direction_choices = {"into the tissue", "out of the tissue"};
1465          PrimaryParameters_Dialog.addRadioButtonGroup("Z-scan direction:",
    Z_scan_direction_choices, 2, 1, "into the tissue");
1466
1467          String[] meristem_orientation_choices = {"native side-view", "native top-view"};
1468          PrimaryParameters_Dialog.addRadioButtonGroup("Meristem orientation:",
    meristem_orientation_choices, 2, 1, "native side-view");
1469
1470
1471          PrimaryParameters_Dialog.setInsets(30, 10, 5);
1472          PrimaryParameters_Dialog.addMessage("Image Analysis Settings:");
1473
1474          String[] centroid_determination_choices = {"automatically", "manually"};
1475          PrimaryParameters_Dialog.addRadioButtonGroup("Domain column center point:",
    centroid_determination_choices, 2, 1, "automatically");
1476
1477
1478          PrimaryParameters_Dialog.setInsets(15, 5, 5);
1479          PrimaryParameters_Dialog.addCheckbox("Show advanced options", false);
1480
1481          PrimaryParameters_Dialog.setHelpLabel("Contact");
1482          PrimaryParameters_Dialog.addHelp("https://orcid.org/0000-0003-1307-4060");
1483          PrimaryParameters_Dialog.showDialog();
1484
1485
1486          boolean[] PrimaryParameters = new boolean[7];
1487
1488
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
    with the same name
1646
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        if (Z_scan_into_tissue) {
1693            IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=90");
1694        }
1695
1696        if (!Z_scan_into_tissue) {
1697            IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=270");
1698        }
1699
1700        activeImage.changes = false;
1701        activeImage.close();
1702        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.getCursorLoc();
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[]
    CenterPointX_array, double[] CenterPointY_array, double[] radius_array) {
1785
1786        ImageStack inputImageStack = inputImage.getStack();
1787        int inputImage_width = inputImage.getWidth();
1788        int inputImage_height = inputImage.getHeight();
1789
1790        double[][][] radius_unrolled_array = new double[stack_size][inputImage_height][
    inputImage_width];
1791        double[][][] phi_array = new double[stack_size][inputImage_height][inputImage_width];
1792
1793        for (int new_z = 0; new_z < stack_size; new_z++) {
1794
1795            for (int new_y = 0; new_y < inputImage_height; new_y++) {
1796
1797
1798                for (int new_x = 0; new_x < inputImage_width; new_x++) {
1799
1800                    radius_unrolled_array[new_z][new_y][new_x] = Math.sqrt(Math.pow((new_x -
    CenterPointX_array[new_z]), 2) + Math.pow((new_y - CenterPointY_array[new_z]), 2));
1801                    phi_array[new_z][new_y][new_x] = Math.atan2((new_y - CenterPointY_array[new_z]), (
    new_x - CenterPointX_array[new_z]));
1802
1803                }
1804            }
1805        }
1806
1807        int[][][] temp_unrolled_y = new int[stack_size][inputImage_height][inputImage_width];
1808        int[][][] temp_unrolled_x = new int[stack_size][inputImage_height][inputImage_width];
1809        double [][][] value = new double[stack_size][inputImage_height][inputImage_width];
1810        int temp_unrolledImage_height = 10000000;
1811        int temp_unrolledImage_width = 5000000;
1812
1813        for (int new_z = 0; new_z < stack_size; new_z++) {
1814
1815            for (int new_y = 0; new_y < inputImage_height; new_y++) {
1816
1817                for (int new_x = 0; new_x < inputImage_width; new_x++) {
1818
1819                    value[new_z][new_y][new_x] = inputImageStack.getVoxel(new_x, new_y, new_z);
1820
1821                    if (value[new_z][new_y][new_x] != 0) {
1822
1823                        temp_unrolled_y[new_z][new_y][new_x] = (int) Math.round(
    temp_unrolledImage_height / 2 - (radius_unrolled_array[new_z][new_y][new_x] - radius_array[new_z]))
    ;
1824                        temp_unrolled_x[new_z][new_y][new_x] = (int) Math.round(
    temp_unrolledImage_width / 2 + phi_array[new_z][new_y][new_x] * radius_array[new_z]);
1825                    }
1826                }
1827
1828            }
1829        }
1830
1831        double min_temp_unrolled_y = temp_unrolledImage_height;
1832        double max_temp_unrolled_y = 0;
1833        double min_temp_unrolled_x = temp_unrolledImage_width;
1834        double max_temp_unrolled_x = 0;
1835
1836        for (int new_z = 0; new_z < stack_size; new_z++) {
1837
1838            for (int new_y = 0; new_y < inputImage_height; new_y++) {
1839
1840                for (int new_x = 0; new_x < inputImage_width; new_x++) {
```

```
1841
1842                    if (max_temp_unrolled_y < temp_unrolled_y[new_z][new_y][new_x]) {
1843                        max_temp_unrolled_y = temp_unrolled_y[new_z][new_y][new_x];
1844                    }
1845
1846                    if (min_temp_unrolled_y > temp_unrolled_y[new_z][new_y][new_x] &&
    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] &&
    temp_unrolled_x[new_z][new_y][new_x] != 0) {
1855                        min_temp_unrolled_x = temp_unrolled_x[new_z][new_y][new_x];
1856                    }
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
1907            for (int new_y = 0; new_y < inputImage_height; new_y++) {
1908
1909                for (int new_x = 0; new_x < inputImage_width; new_x++) {
1910
1911                    if (value[new_z][new_y][new_x] != 0) {
1912
1913                        int unrolled_y = -1 * min_unrolled_y + ((int) Math.round(unrolledImage_height / 2
     - (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
    + phi_array[new_z][new_y][new_x] * radius_array[new_z]));
1915
1916
1917                        if (unrolledImageStack.getVoxel(unrolled_x, unrolled_y, new_z) == 0) {
1918                            unrolledImageStack.setVoxel(unrolled_x, unrolled_y, new_z, value[new_z][
    new_y][new_x]);
1919                        } else {
1920                            double existingPixel = unrolledImageStack.getVoxel(unrolled_x, unrolled_y,
    new_z);
1921                            unrolledImageStack.setVoxel(unrolled_x, unrolled_y, new_z, existingPixel +
    value[new_z][new_y][new_x]);
1922                        }
1923                    }
1924                }
1925            }
1926        }
1927        inputImage.close();
1928        return unrolledImage;
1929    }
1930
1931 }
```