```
// 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);
        double SecondaryParameters_Dialog_cancelled = SecondaryParameters[7];

        if (SecondaryParameters_Dialog_cancelled == 1) {
            return;
        }


// selection of save-directory, automatic naming to avoid overwriting of excisting data

        ImagePlus activeImage = WindowManager.getCurrentImage();
        String title = activeImage.getShortTitle();
        String[] createDirectory = createDirectory(title);
        String directory = createDirectory[0];
        title = createDirectory[1];
        String folder_status = createDirectory[2];

        if (folder_status.equals("not created")) {
            activeImage.close();
            return;
        }


// ROI manager reset, Results reset

        RoiManager ROI = RoiManager.getRoiManager();
        ROI.reset();
        IJ.run("Clear Results");
        activeImage.setSlice(1);


// pre-processing (LUT, naming) and splitting of channels depending on previous user selection of primary parameters

        if (one_channel) {
            IJ.run(activeImage, "royal", "");
            IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel");
        }

        else { // = (!one_channel)
            activeImage.setTitle(title + ".tif");
            IJ.run(activeImage, "Split Channels", "");

            if (channel1_main) {
                activeImage = WindowManager.getImage("C2-" + title + ".tif");
                IJ.run(activeImage, "royal", "");
                IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel");
                activeImage.changes = false;
                activeImage.close();

                activeImage = WindowManager.getImage("C1-" + title + ".tif");
                IJ.run(activeImage, "royal", "");
                IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel");
            }

            else { // = (!channel1_main)
                activeImage = WindowManager.getImage("C1-" + title + ".tif");
                IJ.run(activeImage, "royal", "");
                IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel");
                activeImage.changes = false;
                activeImage.close();

                activeImage = WindowManager.getImage("C2-" + title + ".tif");
                IJ.run(activeImage, "royal", "");
                IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel");
            }
        }


// rotation of images to create a top-view image (if not already defined as top-view, based on previous user selection of primary parameters)

        if (horizontal) {

            activeImage = rotate_horizontal2vertical(activeImage, Z_scan_into_tissue);

            IJ.saveAs(activeImage, "Tiff", directory + title + "_temporary1");

            if (!one_channel) {
                activeImage = IJ.openImage(directory + title + "_support-channel.tif");
                activeImage.show();

                activeImage = rotate_horizontal2vertical(activeImage, Z_scan_into_tissue);

                IJ.saveAs(activeImage, "Tiff", directory + title + "_temporary2");
                activeImage.changes = false;
                activeImage.close();

                activeImage = WindowManager.getCurrentImage();
            }
        }


// definition of variables for analysis domain center point definition (for automated and manual selection, based on previous user selection of primary parameters)

        Calibration activeImageCalibration;

        double double_CenterPointX_microns;
        double double_CenterPointY_microns;

        double voxel_width;
        double voxel_height;

        double double_CenterPointX_pixels;
        double double_CenterPointY_pixels;

        int int_CenterPointX_pixels = 0;
        int int_CenterPointY_pixels = 0;


// (semi-)automated definition of the analysis domain center point via thresholding (if previously selected by the user)

        if (automatic_centroid) {
            IJ.setThreshold(activeImage, lower_threshold, upper_threshold);

            for (int subcounter = 1; subcounter <= activeImage.getStackSize(); subcounter++) {
                IJ.setSlice(subcounter);
                IJ.run(activeImage, "Create Selection", "");
                Roi selection_type = activeImage.getRoi();

                if (selection_type != null) {
                    IJ.run(activeImage, "Clear Outside", "slice");
                }

                if (selection_type == null) {
                    IJ.run(activeImage, "Select All", "");
                    IJ.run(activeImage, "Clear", "slice");
                }
            }

            IJ.resetThreshold(activeImage);
            IJ.run(activeImage, "Select None", "");

            IJ.run(activeImage, "Median 3D...", "x=" + filter_radius + " y=" + filter_radius + " z=" + filter_radius);

            activeImage = maxProjection(activeImage);
            IJ.run(activeImage, "royal", "");

            IJ.setThreshold(activeImage, lower_threshold, upper_threshold);
            IJ.run(activeImage, "Analyze Particles...", "size=10-Infinity show=Masks include clear include add");

            if (horizontal) {
                WindowManager.getImage("Mask of MAX_" + title + "_temporary1.tif").close();
            }

            else { // = (!horizontal)
                WindowManager.getImage("Mask of MAX_" + title + "_main-channel.tif").close();
            }

            WindowManager.getCurrentImage().setTitle("ROI selection window - " + (int)Math.round(SecondaryParameters[1]) + " %");

            ROI.deselect();
            int number_ROIs = ROI.getCount();

            int columns = 3;
            int rows_remainder = number_ROIs % columns;
            int rows = 0;

            if (rows_remainder == 1 || rows_remainder == 2) {
                rows = number_ROIs / columns + 1;
            }

            if (rows_remainder == 0) {
                rows = number_ROIs / columns;
            }

            String[] CheckboxLabels = new String[number_ROIs];
            boolean[] CheckboxDefaults = new boolean[number_ROIs];

            for (int ROI_index = 0; ROI_index < number_ROIs; ROI_index++) {
                int ROI_number = ROI_index + 1;
                CheckboxLabels[ROI_index] = "ROI " + ROI_number;
                CheckboxDefaults[ROI_index] = false;
            }

            GenericDialog ROI_Dialog = new GenericDialog("ROI selection");

            ROI_Dialog.addCheckboxGroup(rows, 3, CheckboxLabels, CheckboxDefaults);

            ROI_Dialog.setCancelLabel("Threshold");
            ROI_Dialog.showDialog();



// 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)

            boolean ThresholdReset;
            boolean ThresholdReset_repeat = false;

            if (ROI_Dialog.wasCanceled()) {
                ThresholdReset = true;

                double percentage = SecondaryParameters[1];
                double new_percentage = percentage;

                do {
                    if (ThresholdReset_repeat) {
                        percentage = new_percentage;
                    }

                    GenericDialog ThresholdReset_Dialog = new GenericDialog("Reset lower threshold");

                    ThresholdReset_Dialog.setInsets(10, 10, 0);
                    ThresholdReset_Dialog.addMessage("Lower threshold: ");
                    ThresholdReset_Dialog.setInsets(5, 10, 7);
                    ThresholdReset_Dialog.addNumericField("", percentage, 0, 3, "% of upper threshold ");

                    ThresholdReset_Dialog.showDialog();

                    if (ThresholdReset_Dialog.wasCanceled()) {
                        return;
                    }

                new_percentage = ThresholdReset_Dialog.getNextNumber();
                lower_threshold = upper_threshold / 100 * new_percentage;

                if (horizontal) {
                    activeImage = IJ.openImage(directory + title + "_temporary1.tif");
                }

                else { // = (!horizontal)
                    activeImage = IJ.openImage(directory + title + "_main-channel.tif");
                }

                activeImage.show();

                IJ.setThreshold(activeImage, lower_threshold, upper_threshold);

                for (int subcounter = 1; subcounter <= activeImage.getStackSize(); subcounter++) {
                    IJ.setSlice(subcounter);
                    IJ.run(activeImage, "Create Selection", "");
                    Roi selection_type = activeImage.getRoi();

                    if (selection_type != null) {
                        IJ.run(activeImage, "Clear Outside", "slice");
                    }

                    if (selection_type == null) {
                        IJ.run(activeImage, "Select All", "");
                        IJ.run(activeImage, "Clear", "slice");
                    }
                }

                IJ.resetThreshold(activeImage);
                IJ.run(activeImage, "Select None", "");

                IJ.run(activeImage, "Median 3D...", "x=" + filter_radius + " y=" + filter_radius + " z=" + filter_radius);

                activeImage = maxProjection(activeImage);
                IJ.run(activeImage, "royal", "");


                IJ.setThreshold(activeImage, lower_threshold, upper_threshold);
                IJ.run(activeImage, "Analyze Particles...", "size=10-Infinity show=Masks include clear include add");

                if (horizontal) {
                    WindowManager.getImage("Mask of MAX_" + title + "_temporary1.tif").close();
                }

                else { // = (!horizontal)
                    WindowManager.getImage("Mask of MAX_" + title + "_main-channel.tif").close();
                }

                WindowManager.getCurrentImage().setTitle("ROI selection window - " + (int)Math.round(new_percentage) + " %");

                ROI.deselect();
                number_ROIs = ROI.getCount();

                columns = 3;
                rows_remainder = number_ROIs % columns;
                rows = 0;

                if (rows_remainder == 1 || rows_remainder == 2) {
                    rows = number_ROIs / columns + 1;
                }

                if (rows_remainder == 0) {
                    rows = number_ROIs / columns;
                }

                CheckboxLabels = new String[number_ROIs];
                CheckboxDefaults = new boolean[number_ROIs];
```

```

                    for (int ROI_index = 0; ROI_index < number_ROIs; ROI_index++) {
                        int ROI_number = ROI_index + 1;
                        CheckboxLabels[ROI_index] = "ROI " + ROI_number;
                        CheckboxDefaults[ROI_index] = false;
                    }

                    ROI_Dialog = new GenericDialog("ROI selection");

                    ROI_Dialog.addCheckboxGroup(rows, 3, CheckboxLabels, CheckboxDefaults);

                    ROI_Dialog.setCancelLabel("Threshold");
                    ROI_Dialog.showDialog();

                    ThresholdReset_repeat = true;

                    if (ROI_Dialog.wasOKed()) {

                        int[] IDlist = WindowManager.getIDList();

                        for (int subcounter = 0; subcounter < IDlist.length - 1; subcounter++) {
                            WindowManager.getImage(IDlist[subcounter]).close();
                        }

                        ThresholdReset = false;
                    }

                } while (ThresholdReset);
            }


// combination of multiple ROIs (if selected) to a single ROI of interest

            for (int ROI_index = 0; ROI_index < number_ROIs; ROI_index++) {

                boolean ROI_selected = ROI_Dialog.getNextBoolean();

                if (!ROI_selected) {
                    ROI.select(ROI_index);
                    ROI.runCommand("Delete");
                    ROI.deselect();
                    ROI_index--;
                    number_ROIs--;
                }
            }

            int number_ROIs_chosen = ROI.getCount();

            if (number_ROIs_chosen != 1) {
                ROI.deselect();
                ROI.runCommand("Combine");
                ROI.deselect();
                ROI.runCommand("Delete");
                ROI.addRoi(activeImage.getRoi());
            }

            WindowManager.getCurrentImage().close();


// definition of the centroid of the ROI of interest, to be used as the center point for the circular analysis domain

            if (horizontal) {
                activeImage = IJ.openImage(directory + title + "_temporary1.tif");
            }

            else { // = (!horizontal)
                activeImage = IJ.openImage(directory + title + "_main-channel.tif");
            }

            activeImage.show();

            activeImage = maxProjection(activeImage);

            ROI.select(0);
            IJ.run("Set Measurements...", "centroid redirect=None decimal=3");
            IJ.run("Measure", "");
            ROI.reset();
            IJ.run("Select None", "");

            ResultsTable centroid_ResultsTable = ResultsTable.getResultsTable();

            double_CenterPointX_microns = centroid_ResultsTable.getValue("X", 0);
            double_CenterPointY_microns = centroid_ResultsTable.getValue("Y", 0);

            IJ.run("Clear Results");

            activeImageCalibration = activeImage.getCalibration();
            voxel_width = activeImageCalibration.pixelWidth;
            voxel_height = activeImageCalibration.pixelHeight;

            double_CenterPointX_pixels = double_CenterPointX_microns / voxel_width;
            double_CenterPointY_pixels = double_CenterPointY_microns / voxel_height;

            int_CenterPointX_pixels = (int) Math.round(double_CenterPointX_pixels);
            int_CenterPointY_pixels = (int) Math.round(double_CenterPointY_pixels);

            ImageProcessor activeImageProcessor = activeImage.getProcessor();
            IJ.setForegroundColor(255, 255, 255);
            activeImageProcessor.fillOval(int_CenterPointX_pixels - 2, int_CenterPointY_pixels - 2, 5, 5);
        }


// manual definition of the analysis domain center point via clicking into the image (if previously selected by the user)

        else { // = (!automated_centroid)

            activeImage = maxProjection(activeImage);

            IJ.setTool("hand");
            ImageCanvas activeImageCanvas = activeImage.getCanvas();
            ImageProcessor activeImageProcessor = activeImage.getProcessor();

            WaitForUserDialog CentroidSelection_Dialog = new WaitForUserDialog("Center Point Definition", " \nClick to define the center point of the analysis domain. \n ");
            CentroidSelection_Dialog.show();

            boolean CenterPoint_selected = false;

            while (!CenterPoint_selected) {

                int[] CursorValues = getCursorLocation(activeImageCanvas);
                int CursorX = CursorValues[0];
                int CursorY = CursorValues[1];
                int CursorModifiers = CursorValues[2];

                if (CursorModifiers == 16) {
                    IJ.setForegroundColor(255, 255, 255);
                    activeImageProcessor.fillOval(CursorX - 2, CursorY - 2, 5, 5);
                    int_CenterPointX_pixels = CursorX;
                    int_CenterPointY_pixels = CursorY;
                    CenterPoint_selected = true;
                }

                activeImage.updateAndDraw();
            }

            double_CenterPointX_pixels = int_CenterPointX_pixels;
            double_CenterPointY_pixels = int_CenterPointY_pixels;

            activeImageCalibration = activeImage.getCalibration();
            voxel_width = activeImageCalibration.pixelWidth;
            voxel_height = activeImageCalibration.pixelHeight;

        double_CenterPointX_microns = double_CenterPointX_pixels * voxel_width;
        double_CenterPointY_microns = double_CenterPointY_pixels * voxel_height;
    }


// specification of the circular analysis domain based on the previously selected or semi-automatically defined center point

    IJ.run(activeImage, "Specify...", "width=" + domain_diameter + " height=" + domain_diameter + " x=" + double_CenterPointX_microns + " y=" + double_CenterPointY_microns + " slice=1 oval centered scaled");

    ROI.addRoi(activeImage.getRoi());
    ROI.select(0);

    IJ.setForegroundColor(255, 255, 255);
    IJ.run(activeImage, "Draw", "stack");

    IJ.run("Select None", "");


// specification of a second circular analysis domain if analysis of the periphery has been selected by the user

    if(peripheral_analysis){

        double peripheral_domain_diameter = domain_diameter + 2 * periphery_width;

        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");

        ROI.addRoi(activeImage.getRoi());
        ROI.select(1);

        IJ.setForegroundColor(255, 255, 255);
        IJ.run(activeImage, "Draw", "stack");

        IJ.run("Select None", "");

        IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_analysis-domain(s)");
        activeImage.changes = false;
        activeImage.close();
    }

    else {
        IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_analysis-domain");
        activeImage.changes = false;
        activeImage.close();
    }


    if (horizontal) {
        activeImage = IJ.openImage(directory + title + "_temporary1.tif");
    }

    else { // = (!horizontal)
        activeImage = IJ.openImage(directory + title + "_main-channel.tif");
    }

    activeImage.show();

    if (peripheral_analysis){
        ROI.select(1);
    }

    else { // = (!peripheral_analysis)
        ROI.select(0);
    }


// 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
```

```
        IJ.run(activeImage, "Clear Outside", "stack");
        IJ.run("Select None", "");

        activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);

        int stack_size = activeImage.getStackSize();
        int startslice = 1;
        int endslice = stack_size;

        boolean startslice_determined = false;
        boolean endslice_determined = false;

        for (int subcounter = 1; subcounter <= stack_size; subcounter++) {
            IJ.setSlice(subcounter);
            IJ.run("Clear Results");
            IJ.run("Set Measurements...", "integrated redirect=None decimal=3");
            IJ.run("Measure", "");

            ResultsTable startslice_endslice_results_table = ResultsTable.getResultsTable();
            double total_pixel_intensity = startslice_endslice_results_table.getValue("RawIntDen", 0);
            IJ.run("Clear Results");

            if (total_pixel_intensity != 0 && !startslice_determined) {
                startslice = activeImage.getCurrentSlice();
                startslice_determined = true;
            }

            if (total_pixel_intensity == 0 && startslice_determined && !endslice_determined) {
                endslice = activeImage.getCurrentSlice() - 1;
                endslice_determined = true;
            }
        }

        activeImage.close();


        if (horizontal) {
            activeImage = IJ.openImage(directory + title + "_temporary1.tif");
        }

        else { // = (!horizontal)
            activeImage = IJ.openImage(directory + title + "_main-channel.tif");
        }

        activeImage.show();

        ROI.select(0);
        IJ.run(activeImage, "Clear Outside", "stack");
        IJ.run("Select None", "");

        activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);

        ImagePlus duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
        duplicatedImage.show();

        activeImage.changes = false;
        activeImage.close();

        IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_main-channel_not-unrolled");
        duplicatedImage.changes = false;
        duplicatedImage.close();


        if(peripheral_analysis){

            if (horizontal) {
                activeImage = IJ.openImage(directory + title + "_temporary1.tif");
            }

            else {
                activeImage = IJ.openImage(directory + title + "_main-channel.tif");
            }
            activeImage.show();

            ROI.select(1);
            IJ.run(activeImage, "Clear Outside", "stack");
            IJ.run("Select None", "");

            ROI.select(0);
            IJ.run(activeImage, "Clear", "stack");

            IJ.run("Select None", "");

            activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);

            duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
            duplicatedImage.show();

            activeImage.changes = false;
            activeImage.close();

            IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_main-channel_not-unrolled_periphery");
            duplicatedImage.changes = false;
            duplicatedImage.close();

        }

        if (one_channel) {
            ROI.reset();

            if (horizontal) {
                activeImage = IJ.openImage(directory + title + "_temporary1.tif");
            } else {
                activeImage = IJ.openImage(directory + title + "_main-channel.tif");
            }
            activeImage.show();

            activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);

            duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
            duplicatedImage.show();
            activeImage.changes = false;
            activeImage.close();
        }

        if(horizontal) {
            File file = new File(directory + title + "_temporary1.tif");

            if (!file.delete()) {
                IJ.error("Error Message", "File could not be deleted");
            }
        }

        if(one_channel) {
            IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_main-channel_circle-points");
            duplicatedImage.changes = false;
            duplicatedImage.close();
        }

        else { // = (!one_channel)

            if (horizontal) {
                activeImage = IJ.openImage(directory + title + "_temporary2.tif");
            }

            else {
                activeImage = IJ.openImage(directory + title + "_support-channel.tif");
            }
            activeImage.show();

            ROI.select(0);

            IJ.run(activeImage, "Clear Outside", "stack");

        IJ.run("Select None", "");

        activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);

        duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
        duplicatedImage.show();
        activeImage.changes = false;
        activeImage.close();
        IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_support-channel_not-unrolled");
        duplicatedImage.changes = false;
        duplicatedImage.close();

        if (peripheral_analysis) {

            if (horizontal) {
                activeImage = IJ.openImage(directory + title + "_temporary2.tif");
            }

            else {
                activeImage = IJ.openImage(directory + title + "_support-channel.tif");
            }
            activeImage.show();

            ROI.select(1);
            IJ.run(activeImage, "Clear Outside", "stack");
            IJ.run("Select None", "");


            ROI.select(0);
            IJ.run(activeImage, "Clear", "stack");
            IJ.run("Select None", "");

            activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);

            duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
            duplicatedImage.show();
            activeImage.changes = false;
            activeImage.close();
            IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_support-channel_not-unrolled_periphery");
            duplicatedImage.changes = false;
            duplicatedImage.close();

        }

        ROI.reset();


        if (horizontal) {
            activeImage = IJ.openImage(directory + title + "_temporary2.tif");
        }

        else {
            activeImage = IJ.openImage(directory + title + "_support-channel.tif");
        }

        activeImage.show();

        activeImage = rotate_vertical2horizontal(activeImage, Z_scan_into_tissue);

        duplicatedImage = new Duplicator().run(activeImage, startslice, endslice);
        duplicatedImage.show();
        activeImage.changes = false;
        activeImage.close();


        if (horizontal) {
            File file = new File(directory + title + "_temporary2.tif");

            if (!file.delete()) {
                IJ.error("Error Message", "File could not be deleted");
            }
        }
```

```

            IJ.saveAs(duplicatedImage, "Tiff", directory + title + "_support-channel_circle-points");
            duplicatedImage.changes = false;
            duplicatedImage.close();
        }


// selection of circle points for polar transformation

        if (one_channel) {
            activeImage = IJ.openImage(directory + title + "_main-channel_circle-points.tif");
            activeImage.show();
        }

        else { // = (!one_channel)
            activeImage = IJ.openImage(directory + title + "_support-channel_circle-points.tif");
            activeImage.show();
        }

        stack_size = activeImage.getStackSize();

        int[][] CirclePointsX = new int[3][stack_size];
        int[][] CirclePointsY = new int[3][stack_size];

        ImageCanvas activeImageCanvas = activeImage.getCanvas();
        ImageProcessor activeImageProcessor = activeImage.getProcessor();

        IJ.setTool("hand");

        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 ");
        PointSelection_Dialog1.show();

        for (int subcounter = 0; subcounter <= 2; subcounter++) {

            int currentSlice = 1;

            IJ.setForegroundColor(255, 255, 255);

            while (currentSlice <= stack_size) {
                IJ.setSlice(currentSlice);
                int[] CursorValues = getCursorLocation(activeImageCanvas);
                int CursorX = CursorValues[0];
                int CursorY = CursorValues[1];
                int CursorModifiers = CursorValues[2];


                if (CursorModifiers == 16) {

                    if (currentSlice == stack_size) {

                        CirclePointsX[subcounter][currentSlice - 1] = CursorX;
                        CirclePointsY[subcounter][currentSlice - 1] = CursorY;
                        activeImageProcessor.fillOval(CursorX - 2, CursorY - 2, 5, 5);

                        currentSlice++;

                        if (subcounter == 0 || subcounter == 1) {
                            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 ");
                            PointSelection_Dialog2.show();
                        }

                        if (subcounter == 2) {
                            WaitForUserDialog PointSelection_Dialog3 = new WaitForUserDialog("Circle Point Definition", " \nCircle point definition has been completed. \n \nClick 'OK' to continue with subsequent analysis.\n ");
                            PointSelection_Dialog3.show();
                        }
                    } else {

                        CirclePointsX[subcounter][currentSlice - 1] = CursorX;
                        CirclePointsY[subcounter][currentSlice - 1] = CursorY;
                        activeImageProcessor.fillOval(CursorX - 2, CursorY - 2, 5, 5);

                        currentSlice++;

                        IJ.wait(CirclePointSelectionDelay);
                    }
                }
            }
        }

        if (one_channel) {
            IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_circle-points");

        } else { // = (!one_channel)
            IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_circle-points");
        }

        activeImage.close();




// calculation of image canvas size needed after polar transformation

        double[] CenterPointX_array = new double[stack_size];
        double[] CenterPointY_array = new double[stack_size];
        double[] radius_array = new double[stack_size];
        int[] highest_CirclePoint_Y = new int[stack_size];

        for (int new_z = 0; new_z < stack_size; new_z++) {

            int CirclePointX_1 = CirclePointsX[0][new_z];
            int CirclePointY_1 = CirclePointsY[0][new_z];

            int CirclePointX_2 = CirclePointsX[1][new_z];
            int CirclePointY_2 = CirclePointsY[1][new_z];

            int CirclePointX_3 = CirclePointsX[2][new_z];
            int CirclePointY_3 = CirclePointsY[2][new_z];

            if (CirclePointY_1 <= CirclePointY_2 && CirclePointY_1 <= CirclePointY_3) {
                highest_CirclePoint_Y[new_z] = CirclePointY_1;
            }

            if (CirclePointY_2 <= CirclePointY_1 && CirclePointY_2 <= CirclePointY_3) {
                highest_CirclePoint_Y[new_z] = CirclePointY_2;
            }

            if (CirclePointY_3 <= CirclePointY_1 && CirclePointY_3 <= CirclePointY_2) {
                highest_CirclePoint_Y[new_z] = CirclePointY_3;
            }

            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)));
            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)));

            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;
        }
```

```


// if peripheral analysis is active the whole analysis is duplicated from this point on

        for (int counter = 0; counter <= 1; counter++) {

            String peripheral_analysis_name_variabel;

            if (peripheral_analysis && counter == 0) {
                peripheral_analysis_name_variabel = "";
            }

            else { // = (peripheral_analysis && counter == 1)
                peripheral_analysis_name_variabel = "_periphery";
            }

            if (!peripheral_analysis) {
                peripheral_analysis_name_variabel = "";
            }


// removal of all signal intensity below the center point of the circle (pole for polar transformation)

            activeImage = IJ.openImage(directory + title + "_main-channel_not-unrolled" + peripheral_analysis_name_variabel + ".tif");
            activeImage.show();

            for (int subcounter = 1; subcounter <= stack_size; subcounter++) {

                activeImage.setSlice(subcounter);

                if (CenterPointY_array[subcounter - 1] > activeImage.getHeight()) {
                    activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.getWidth(), activeImage.getHeight() - highest_CirclePoint_Y[subcounter - 1]);

                    if (highest_CirclePoint_Y[subcounter - 1] == 0) {
                        activeImage.setRoi(0, 0, activeImage.getWidth(), activeImage.getHeight() - highest_CirclePoint_Y[subcounter - 1]);
                    }

                } else {
                    activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.getWidth(), (int) Math.round(CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter - 1]));

                    if (highest_CirclePoint_Y[subcounter - 1] == 0) {
                        activeImage.setRoi(0, 0, activeImage.getWidth(), (int) Math.round(CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter - 1]));
                    }
                }

                IJ.run(activeImage, "Clear Outside", "slice");
                IJ.run(activeImage, "Select None", "");
                ROI.reset();
            }

            IJ.run(activeImage, "HiLo", "");
            IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_not-unrolled" + peripheral_analysis_name_variabel);

            activeImage.hide();


// polar transformation

            ImagePlus inputImage = activeImage;

            activeImage = transform_polarTransformation(inputImage, stack_size, CenterPointX_array, CenterPointY_array, radius_array);
            IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_unrolled-3D" + peripheral_analysis_name_variabel);


// reduction of information dimension via subsequent rotations and projections (3D to 1D)
```

```

            activeImage = sumProjection(activeImage);
            IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_unrolled-2D" + peripheral_analysis_name_variabel);

            IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=90 x-angle=0");
            activeImage.close();
            activeImage = WindowManager.getCurrentImage();

            activeImage = sumProjection(activeImage);
            IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_unrolled-1D");


// extraction of all signal intensity in 1D image to create a line plot

            activeImageProcessor = activeImage.getProcessor();

            float[] ProfilePlotXValues = new float[activeImage.getHeight()];
            float[] mainProfilePlotYValues = new float[ProfilePlotXValues.length];
            float[] supportProfilePlotYValues = new float[ProfilePlotXValues.length];
            float[] selectProfilePlotYValues = new float[ProfilePlotXValues.length];

            for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {
                ProfilePlotXValues[subcounter] = subcounter;
                mainProfilePlotYValues[subcounter] = activeImageProcessor.getPixelValue(0, subcounter);
            }

            activeImage.close();


// repetition of polar transformation, dimension reduction and information extraction in case there was a secondary channel

            if (!one_channel) {

                activeImage = IJ.openImage(directory + title + "_support-channel_not-unrolled" + peripheral_analysis_name_variabel + ".tif");
                activeImage.show();

                for (int subcounter = 1; subcounter <= stack_size; subcounter++) {

                    activeImage.setSlice(subcounter);

                    if (CenterPointY_array[subcounter - 1] > activeImage.getHeight()) {
                        activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.getWidth(), activeImage.getHeight() - highest_CirclePoint_Y[subcounter - 1]);

                        if (highest_CirclePoint_Y[subcounter - 1] == 0) {
                            activeImage.setRoi(0, 0, activeImage.getWidth(), activeImage.getHeight() - highest_CirclePoint_Y[subcounter - 1]);
                        }

                    } else {
                        activeImage.setRoi(0, highest_CirclePoint_Y[subcounter - 1] - 10, activeImage.getWidth(), (int) Math.round(CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter - 1]));

                        if (highest_CirclePoint_Y[subcounter - 1] == 0) {
                            activeImage.setRoi(0, 0, activeImage.getWidth(), (int) Math.round(CenterPointY_array[subcounter - 1] - highest_CirclePoint_Y[subcounter - 1]));
                        }
                    }

                    IJ.run(activeImage, "Clear Outside", "slice");
                    IJ.run(activeImage, "Select None", "");
                    ROI.reset();
                }

                IJ.run(activeImage, "HiLo", "");
                IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_not-unrolled" + peripheral_analysis_name_variabel);

            activeImage.hide();

            inputImage = activeImage;

            activeImage = transform_polarTransformation(inputImage, stack_size, CenterPointX_array, CenterPointY_array, radius_array);
            IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-3D" + peripheral_analysis_name_variabel);

            activeImage = sumProjection(activeImage);
            IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-2D" + peripheral_analysis_name_variabel);

            IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=90 x-angle=0");
            activeImage.close();
            activeImage = WindowManager.getCurrentImage();

            activeImage = sumProjection(activeImage);
            IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-1D");

            activeImageProcessor = activeImage.getProcessor();

            for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {
                supportProfilePlotYValues[subcounter] = activeImageProcessor.getPixelValue(0, subcounter);
            }

            activeImage.close();

        }


// plotting of all information in a line plot to select nuclei borders

        int PlotDrawingFrameX;
        double activeImagePixelWidth;

        int[] CursorX_array = new int[4];
        int[] PlotXValues_array = new int[4];

        String PlotTitle;

        if (one_channel) {

            PlotTitle = title + "_main-channel_profile-plot" + peripheral_analysis_name_variabel + ".tif";
            System.arraycopy(mainProfilePlotYValues, 0, selectProfilePlotYValues, 0, ProfilePlotXValues.length);
        }

        else { // = (!one_channel)

            PlotTitle = title + "_support-channel_profile-plot" + peripheral_analysis_name_variabel + ".tif";
            System.arraycopy(supportProfilePlotYValues, 0, selectProfilePlotYValues, 0, ProfilePlotXValues.length);
        }

        Plot ProfilePlot = new Plot(PlotTitle, "Distance", "Gray Values");
        ProfilePlot.setColor(Color.GRAY);
        ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.LINE);

        ProfilePlot.setColor(Color.BLACK);
        ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.CIRCLE);

        ProfilePlot.setFrozen(true);
        ProfilePlot.show();

        activeImage = ProfilePlot.getImagePlus();
        activeImageCanvas = activeImage.getCanvas();
        activeImagePixelWidth = activeImage.getCalibration().pixelWidth;

        PlotDrawingFrameX = ProfilePlot.getDrawingFrame().x;
```

```


            IJ.setTool("hand");

            for (int subcounter = 0; subcounter <= 2; subcounter++) {

                if (subcounter == 0) {
                    WaitForUserDialog NucleiBorderSelection_Dialog = new WaitForUserDialog("Cell layer definition", " \nClick to define the border between the L1 and L2. \n ");
                    NucleiBorderSelection_Dialog.show();
                }

                if (subcounter == 1) {
                    WaitForUserDialog NucleiBorderSelection_Dialog = new WaitForUserDialog("Cell layer definition", " \nClick to define the border between the L2 and L3. \n ");
                    NucleiBorderSelection_Dialog.show();
                }

                if (subcounter == 2) {
                    WaitForUserDialog NucleiBorderSelection_Dialog = new WaitForUserDialog("Cell layer definition", " \nClick to define the right border of the first cell layer of L3. \n ");
                    NucleiBorderSelection_Dialog.show();
                }

                boolean PlotPoint_selected = false;

                while (!PlotPoint_selected) {
                    activeImage.setSlice(1); //no clue why this is needed, but doesn't work otherwise
                    int[] CursorValues = getCursorLocation(activeImageCanvas);
                    int CursorX = CursorValues[0];
                    int CursorModifiers = CursorValues[2];

                    if (CursorModifiers == 16) {
                        CursorX_array[subcounter + 1] = CursorX;
                        PlotXValues_array[subcounter + 1] = (int) Math.round((CursorX - PlotDrawingFrameX) * activeImagePixelWidth);

                        PlotPoint_selected = true;
                    }
                }

                if (subcounter == 0 || subcounter == 1) {

                    ProfilePlot = new Plot(PlotTitle, "Distance", "Gray Values");
                    ProfilePlot.setColor(Color.GRAY);
                    ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.LINE);

                    ProfilePlot.setColor(Color.BLACK);
                    ProfilePlot.addPoints(ProfilePlotXValues, selectProfilePlotYValues, Plot.CIRCLE);

                    ProfilePlot.setColor(Color.RED);
                }

                if (subcounter == 0) {
                    ProfilePlot.drawLine(PlotXValues_array[subcounter + 1], 0, PlotXValues_array[subcounter + 1], selectProfilePlotYValues[PlotXValues_array[subcounter + 1]]);
                }

                if (subcounter == 1) {
                    ProfilePlot.drawLine(PlotXValues_array[subcounter], 0, PlotXValues_array[subcounter], selectProfilePlotYValues[PlotXValues_array[subcounter]]);
                    ProfilePlot.drawLine(PlotXValues_array[subcounter + 1], 0, PlotXValues_array[subcounter + 1], selectProfilePlotYValues[PlotXValues_array[subcounter + 1]]);

                    int SecondNucleus_distance = CursorX_array[2] - CursorX_array[1];

                    CursorX_array[0] = CursorX_array[1] - SecondNucleus_distance;

                    if (CursorX_array[0] < PlotDrawingFrameX) {
                        CursorX_array[0] = PlotDrawingFrameX;
                    }

                    PlotXValues_array[0] = (int) Math.round((CursorX_array[0] - PlotDrawingFrameX) *
activeImagePixelWidth);

                ProfilePlot.drawLine(PlotXValues_array[0], 0, PlotXValues_array[0], selectProfilePlotYValues[PlotXValues_array[0]]);
            }

            if (subcounter == 0 || subcounter == 1) {

                ProfilePlot.setFrozen(true);
                ProfilePlot.show();

                activeImage.close();

                activeImage = ProfilePlot.getImagePlus();
                activeImageCanvas = activeImage.getCanvas();
                activeImagePixelWidth = activeImage.getCalibration().pixelWidth;
                PlotDrawingFrameX = ProfilePlot.getDrawingFrame().x;
            }
        }

        activeImage.close();


        ProfilePlot = new Plot(title + "_profile-plot" + peripheral_analysis_name_variabel + ".tif", "Distance", "Gray Values");


        if (!one_channel) {

            ProfilePlot.setColor(Color.LIGHT_GRAY);
            ProfilePlot.addPoints(ProfilePlotXValues, supportProfilePlotYValues, Plot.LINE);
            ProfilePlot.addPoints(ProfilePlotXValues, supportProfilePlotYValues, Plot.CIRCLE);

        }

        ProfilePlot.setColor(Color.BLACK);
        ProfilePlot.addPoints(ProfilePlotXValues, mainProfilePlotYValues, Plot.LINE);
        ProfilePlot.addPoints(ProfilePlotXValues, mainProfilePlotYValues, Plot.CIRCLE);


        ProfilePlot.setColor(Color.RED);

        for (int subcounter = 0; subcounter <= 3; subcounter++) {

            if (mainProfilePlotYValues[PlotXValues_array[subcounter]] > supportProfilePlotYValues[PlotXValues_array[subcounter]]) {

                ProfilePlot.drawLine(PlotXValues_array[subcounter], 0, PlotXValues_array[subcounter], mainProfilePlotYValues[PlotXValues_array[subcounter]]);
            }

            else{

                ProfilePlot.drawLine(PlotXValues_array[subcounter], 0, PlotXValues_array[subcounter], supportProfilePlotYValues[PlotXValues_array[subcounter]]);
            }

        }

        if (!one_channel) {

            ProfilePlot.setPlotObjectLabel(1, "support channel");
            ProfilePlot.setPlotObjectLabel(3, "main channel");
        }

        else { // = (!one_channel)

            ProfilePlot.setPlotObjectLabel(1, "main channel");
        }

        ProfilePlot.setColor(Color.BLACK);
        ProfilePlot.setLegend(" ", Plot.TOP_RIGHT);

        ProfilePlot.show();
        ProfilePlot.setLimitsToFit(true);
        ProfilePlot.updateImage();
        ProfilePlot.setFrozen(true);

        activeImage = ProfilePlot.getImagePlus();
        IJ.saveAs(activeImage, "Tiff", directory + title + "_profile-plot" + peripheral_analysis_name_variabel);
        activeImage.close();


// measurement of layer specific fluorescence intensity in the 2D image, based on the nuclei borders selected

        activeImage = IJ.openImage(directory + title + "_main-channel_unrolled-2D" + peripheral_analysis_name_variabel + ".tif");
        activeImage.show();

        activeImage.setRoi(0, PlotXValues_array[0], activeImage.getWidth(), PlotXValues_array[1] - PlotXValues_array[0]);
        ROI.addRoi(activeImage.getRoi());
        ROI.select(0);
        ROI.runCommand("Rename", "1st nucleus");
        IJ.run("Set Measurements...", "integrated redirect=None decimal=3");
        IJ.run("Measure", "");
        ResultsTable firstNucleus_results_table = ResultsTable.getResultsTable();
        double firstNucleus_intensity = firstNucleus_results_table.getValue("RawIntDen", 0);
        IJ.run("Clear Results");
        IJ.run(activeImage, "Select None", "");


        activeImage.setRoi(0, PlotXValues_array[1], activeImage.getWidth(), PlotXValues_array[2] - PlotXValues_array[1]);
        ROI.addRoi(activeImage.getRoi());
        ROI.select(1);
        ROI.runCommand("Rename", "2nd nucleus");
        IJ.run("Set Measurements...", "integrated redirect=None decimal=3");
        IJ.run("Measure", "");
        ResultsTable secondNucleus_results_table = ResultsTable.getResultsTable();
        double secondNucleus_intensity = secondNucleus_results_table.getValue("RawIntDen", 0);
        IJ.run("Clear Results");
        IJ.run(activeImage, "Select None", "");


        activeImage.setRoi(0, PlotXValues_array[2], activeImage.getWidth(), PlotXValues_array[3] - PlotXValues_array[2]);
        ROI.addRoi(activeImage.getRoi());
        ROI.select(2);
        ROI.runCommand("Rename", "3rd nucleus");
        IJ.run("Set Measurements...", "integrated redirect=None decimal=3");
        IJ.run("Measure", "");
        ResultsTable thirdNucleus_results_table = ResultsTable.getResultsTable();
        double thirdNucleus_intensity = thirdNucleus_results_table.getValue("RawIntDen", 0);
        IJ.run("Clear Results");
        IJ.run(activeImage, "Select None", "");

        ROI.runCommand("Save", directory + title + "_nuclei-borders_ROIs.zip");
        ROI.reset();

        activeImageProcessor = activeImage.getProcessor();
        activeImageProcessor.setColor(Color.white);

        for (int subcounter = 0; subcounter <= 3; subcounter++) {

            activeImageProcessor.drawLine(0, PlotXValues_array[subcounter], activeImage.getWidth(), PlotXValues_array[subcounter]);
        }

        IJ.saveAs(activeImage, "Tiff", directory + title + "_main-channel_unrolled-2D" + peripheral_analysis_name_variabel + "_nuclei-borders");
        activeImage.close();


```

```
            if (!one_channel) {

                activeImage = IJ.openImage(directory + title + "_support-channel_unrolled-2D" + peripheral_analysis_name_variabel + ".tif");
                activeImage.show();

                activeImageProcessor = activeImage.getProcessor();
                activeImageProcessor.setColor(Color.white);

                for (int subcounter = 0; subcounter <= 3; subcounter++) {

                    activeImageProcessor.drawLine(0, PlotXValues_array[subcounter], activeImage.getWidth(), PlotXValues_array[subcounter]);
                }

                IJ.saveAs(activeImage, "Tiff", directory + title + "_support-channel_unrolled-2D" + peripheral_analysis_name_variabel + "_nuclei-borders");
                activeImage.close();
            }


// calculation of different intensity ratios and storage of all data in a results table to be saved

            double firstsecondthirdNucleus_intensity = firstNucleus_intensity + secondNucleus_intensity + thirdNucleus_intensity;
            double firstNucleusPercentage = firstNucleus_intensity * 100 / firstsecondthirdNucleus_intensity;
            double secondNucleusPercentage = secondNucleus_intensity * 100 / firstsecondthirdNucleus_intensity;
            double thirdNucleusPercentage = thirdNucleus_intensity * 100 / firstsecondthirdNucleus_intensity;


            ResultsTable AnalysisData = new ResultsTable();

            AnalysisData.setValue("", 0, title + " [intensity]");
            AnalysisData.setValue("", 1, title + " [intensity %]");
            AnalysisData.setValue("", 2, "");
            AnalysisData.setValue("", 3, "ratios for pWUS (L3 expression)");
            AnalysisData.setValue("", 4, title + " [intensity ratio]");
            AnalysisData.setValue("", 5, "");
            AnalysisData.setValue("", 6, "ratios for pML1 (L1 expression)");
            AnalysisData.setValue("", 7, title + " [intensity ratio]");

            AnalysisData.setValue("1st Nucleus", 0, firstNucleus_intensity);
            AnalysisData.setValue("1st Nucleus", 1, firstNucleusPercentage);
            AnalysisData.setValue("1st Nucleus", 2, "");
            AnalysisData.setValue("1st Nucleus", 3, "1st to 3rd");
            AnalysisData.setValue("1st Nucleus", 4, firstNucleus_intensity / thirdNucleus_intensity);
            AnalysisData.setValue("1st Nucleus", 5, "");
            AnalysisData.setValue("1st Nucleus", 6, "3rd to 1st");
            AnalysisData.setValue("1st Nucleus", 7, thirdNucleus_intensity / firstNucleus_intensity);

            AnalysisData.setValue("2nd Nucleus", 0, secondNucleus_intensity);
            AnalysisData.setValue("2nd Nucleus", 1, secondNucleusPercentage);
            AnalysisData.setValue("2nd Nucleus", 2, "");
            AnalysisData.setValue("2nd Nucleus", 3, "2nd to 3rd");
            AnalysisData.setValue("2nd Nucleus", 4, secondNucleus_intensity / thirdNucleus_intensity);
            AnalysisData.setValue("2nd Nucleus", 5, "");
            AnalysisData.setValue("2nd Nucleus", 6, "2nd to 1st");
            AnalysisData.setValue("2nd Nucleus", 7, secondNucleus_intensity / firstNucleus_intensity);

            AnalysisData.setValue("3rd Nucleus", 0, thirdNucleus_intensity);
            AnalysisData.setValue("3rd Nucleus", 1, thirdNucleusPercentage);
            AnalysisData.setValue("3rd Nucleus", 2, "");
            AnalysisData.setValue("3rd Nucleus", 3, "1st to 2nd");
            AnalysisData.setValue("3rd Nucleus", 4, firstNucleus_intensity / secondNucleus_intensity);
            AnalysisData.setValue("3rd Nucleus", 5, "");
            AnalysisData.setValue("3rd Nucleus", 6, "3rd to 2nd");
            AnalysisData.setValue("3rd Nucleus", 7, thirdNucleus_intensity / secondNucleus_intensity);

            AnalysisData.setValue("total", 0, firstsecondthirdNucleus_intensity);
            AnalysisData.setValue("total", 1, firstNucleusPercentage + secondNucleusPercentage + thirdNucleusPercentage);
            AnalysisData.setValue("total", 2, "");
            AnalysisData.setValue("total", 3, "3rd to 3rd");
            AnalysisData.setValue("total", 4, thirdNucleus_intensity / thirdNucleus_intensity);
            AnalysisData.setValue("total", 5, "");
            AnalysisData.setValue("total", 6, "1st to 1st");
            AnalysisData.setValue("total", 7, firstNucleus_intensity / firstNucleus_intensity);

            AnalysisData.save(directory + title + "_analysis-data" + peripheral_analysis_name_variabel + ".csv");


// saving all plot data, including selected nuclei borders, in a results table

            ResultsTable PlotData = new ResultsTable();

            String[] BorderPoints = new String[ProfilePlotXValues.length];

            for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {

                if (ProfilePlotXValues[subcounter] == (PlotXValues_array[0]) || ProfilePlotXValues[subcounter] == (PlotXValues_array[1]) || ProfilePlotXValues[subcounter] == (PlotXValues_array[2]) || ProfilePlotXValues[subcounter] == (PlotXValues_array[3])) {
                    BorderPoints[subcounter] = "selected border";
                } else {
                    BorderPoints[subcounter] = "";
                }
            }

            for (int subcounter = 0; subcounter < ProfilePlotXValues.length; subcounter++) {
                PlotData.setValue("X-value", subcounter, ProfilePlotXValues[subcounter]);
                PlotData.setValue("main-channel Y-value", subcounter, mainProfilePlotYValues[subcounter]);

                if (!one_channel) {
                    PlotData.setValue("support-channel Y-value", subcounter, supportProfilePlotYValues[subcounter]);
                }

                PlotData.setValue("", subcounter, BorderPoints[subcounter]);
            }

            PlotData.save(directory + title + "_profile-plot-data" + peripheral_analysis_name_variabel + ".csv");

            if (!peripheral_analysis) {
                counter = 1;
            }
        }

        if (peripheral_analysis) {

            IJ.openImage(directory + title + "_profile-plot_periphery.tif").show();
        }

        IJ.openImage(directory + title + "_profile-plot.tif").show();

        WaitForUserDialog QuitPlugin_Dialog = new WaitForUserDialog("Quit Plugin", " \n Analysis has run successfully. Click 'OK' to quit the plugin. \n ");
        QuitPlugin_Dialog.show();

        IJ.run("Close All", "");

// end of the main function

    }


// secondary functions to be called in the main function above


// function to check it the right amount of images is open

    private boolean checkExit() {

        boolean exit = false;

        int ImageWindowNumber = WindowManager.getImageCount();

        if (ImageWindowNumber == 0) {
            IJ.error("Error Message", "Please open an image to be analyzed.");
            exit = true;
        }

        if (ImageWindowNumber != 0 && ImageWindowNumber != 1) {
            IJ.error("Error Message", "Please open only one image.");
            exit = true;
        }

        return exit;
    }


// dialog for selection of primary parameters

    private boolean[] PrimaryParameters_Dialog() {


        GenericDialog PrimaryParameters_Dialog = new GenericDialog("Analysis Setup");


        PrimaryParameters_Dialog.centerDialog(true);

        PrimaryParameters_Dialog.setInsets(10, 10, 5);
        PrimaryParameters_Dialog.addMessage("Image Acquisition Parameters:");

        String[] acquisition_channel_number_choices = {"one channel", "two channels"};
        PrimaryParameters_Dialog.addRadioButtonGroup("Acquisition channels:", acquisition_channel_number_choices, 2, 1, "one channel");

        String[] main_signal_channel_choices = {"channel 1", "channel 2"};
        PrimaryParameters_Dialog.addRadioButtonGroup("Main signal channel:", main_signal_channel_choices, 2, 1, "channel 1");

        String[] Z_scan_direction_choices = {"into the tissue", "out of the tissue"};
        PrimaryParameters_Dialog.addRadioButtonGroup("Z-scan direction:", Z_scan_direction_choices, 2, 1, "into the tissue");

        String[] meristem_orientation_choices = {"native side-view", "native top-view"};
        PrimaryParameters_Dialog.addRadioButtonGroup("Meristem orientation:", meristem_orientation_choices, 2, 1, "native side-view");


        PrimaryParameters_Dialog.setInsets(30, 10, 5);
        PrimaryParameters_Dialog.addMessage("Image Analysis Settings:");

        String[] centroid_determination_choices = {"automatically", "manually"};
        PrimaryParameters_Dialog.addRadioButtonGroup("Domain column center point:", centroid_determination_choices, 2, 1, "automatically");


        PrimaryParameters_Dialog.setInsets(15, 5, 5);
        PrimaryParameters_Dialog.addCheckbox("Show advanced options", false);

        PrimaryParameters_Dialog.setHelpLabel("Contact");
        PrimaryParameters_Dialog.addHelp("https://orcid.org/0000-0003-1307-4060");
        PrimaryParameters_Dialog.showDialog();


        boolean[] PrimaryParameters = new boolean[7];



        // boolean[] PrimaryParameters[0] = boolean one_channel = true = "one channel"
        // boolean[] PrimaryParameters[0] = boolean one_channel = false = "two channels"

        if (PrimaryParameters_Dialog.getNextRadioButton().equals("one channel")) {
            PrimaryParameters[0] = true;
        }


        // boolean[] PrimaryParameters[1] = boolean channel1_main = true = "channel 1"
        // boolean[] PrimaryParameters[1] = boolean channel1_main = false = "channel 2"

        if (PrimaryParameters_Dialog.getNextRadioButton().equals("channel 1")) {
            PrimaryParameters[1] = true;
        }


        // boolean[] PrimaryParameters[2] = boolean Z_scan_into_tissue = true = "into the tissue"
        // boolean[] PrimaryParameters[2] = boolean bottom_to_top = false = "out from the tissue"

        if (PrimaryParameters_Dialog.getNextRadioButton().equals("into the tissue")) {
            PrimaryParameters[2] = true;
        }


        // boolean[] PrimaryParameters[3] = boolean horizontal = true = "native side-view"
        // boolean[] PrimaryParameters[3] = boolean horizontal = false = "native top-view"

        if (PrimaryParameters_Dialog.getNextRadioButton().equals("native side-view")) {
            PrimaryParameters[3] = true;
        }


        // boolean[] PrimaryParameters[4] = boolean automatic_centroid = true = "automatic determination"
        // boolean[] PrimaryParameters[4] = boolean automatic_centroid = false = "manual determination"

        if (PrimaryParameters_Dialog.getNextRadioButton().equals("automatically")) {
            PrimaryParameters[4] = true;
        }


        // boolean[] PrimaryParameters[5] = boolean advancedOptions = true = checkbox checked
        // boolean[] PrimaryParameters[5] = boolean advancedOptions = false = checkbox unchecked

        PrimaryParameters[5] = PrimaryParameters_Dialog.getNextBoolean();


        if (PrimaryParameters_Dialog.wasCanceled()){
            PrimaryParameters[6] = true;
            WindowManager.getCurrentImage().close();
        }

        return PrimaryParameters;
    }


// dialog for selection of secondary parameters

    private double[] SecondaryParameters_Dialog(boolean automatic_centroid, boolean advancedOptions) {

        GenericDialog SecondaryParameters_Dialog = new GenericDialog("Analysis Setup");

        if (automatic_centroid) {
            SecondaryParameters_Dialog.setInsets(10, 10, 0);
            SecondaryParameters_Dialog.addMessage("Upper Threshold: ");
            SecondaryParameters_Dialog.setInsets(5, 10, 7);
            SecondaryParameters_Dialog.addNumericField("", 4095, 0, 5, "intensity value");

            SecondaryParameters_Dialog.setInsets(10, 10, 0);
            SecondaryParameters_Dialog.addMessage("Lower threshold: ");
            SecondaryParameters_Dialog.setInsets(5, 10, 7);
            SecondaryParameters_Dialog.addNumericField("", 35, 0, 3, "% of upper threshold");

            SecondaryParameters_Dialog.setInsets(10, 10, 0);
            SecondaryParameters_Dialog.addMessage("3D median filter: ");
            SecondaryParameters_Dialog.setInsets(5, 10, 7);
            SecondaryParameters_Dialog.addNumericField("", 1, 1, 3, "radius");
        }

        SecondaryParameters_Dialog.setInsets(10, 10, 0);
        SecondaryParameters_Dialog.addMessage("Domain diameter: ");
        SecondaryParameters_Dialog.setInsets(5, 10, 7);
        SecondaryParameters_Dialog.addNumericField("", 20, 0, 3, "microns");

        if (!automatic_centroid) {
            SecondaryParameters_Dialog.setInsets(10, 10, 0);
            String[] peripheral_analysis_choices = {"yes", "no"};
            SecondaryParameters_Dialog.addRadioButtonGroup("Peripheral Analysis:", peripheral_analysis_choices, 2, 1, "yes");

            SecondaryParameters_Dialog.setInsets(10, 10, 0);
            SecondaryParameters_Dialog.addMessage("Periphery width: ");
            SecondaryParameters_Dialog.setInsets(5, 10, 7);
            SecondaryParameters_Dialog.addNumericField("", 10, 0, 3, "microns");
        }

        SecondaryParameters_Dialog.setInsets(10, 10, 0);
        SecondaryParameters_Dialog.addMessage("Circle point selection delay: ");
        SecondaryParameters_Dialog.setInsets(5, 10, 7);
        SecondaryParameters_Dialog.addNumericField("", 200, 0, 5, "milliseconds");


        SecondaryParameters_Dialog.setHelpLabel("Contact");
        SecondaryParameters_Dialog.addHelp("https://orcid.org/0000-0003-1307-4060");

        if (advancedOptions) {
            SecondaryParameters_Dialog.showDialog();
        }


        double[] SecondaryParameters = new double[8];

        if (automatic_centroid) {
            SecondaryParameters[0] = SecondaryParameters_Dialog.getNextNumber();
            SecondaryParameters[1] = SecondaryParameters_Dialog.getNextNumber();
            SecondaryParameters[2] = SecondaryParameters_Dialog.getNextNumber();
        }

        else {
            SecondaryParameters[0] = 0;
            SecondaryParameters[1] = 0;
            SecondaryParameters[2] = 0;
        }

        SecondaryParameters[3] = SecondaryParameters_Dialog.getNextNumber();

        if (!automatic_centroid) {

            if (SecondaryParameters_Dialog.getNextRadioButton().equals("yes")) {
                SecondaryParameters[4] = 1;
            }

            else {
                SecondaryParameters[4] = 0;
            }

            SecondaryParameters[5] = SecondaryParameters_Dialog.getNextNumber();
        }

        else {
            SecondaryParameters[4] = 0;
            SecondaryParameters[5] = 0;
        }

        SecondaryParameters[6] = SecondaryParameters_Dialog.getNextNumber();

        SecondaryParameters[7] = 0;

        if (SecondaryParameters_Dialog.wasCanceled()){
            SecondaryParameters[7] = 1;
            WindowManager.getCurrentImage().close();
        }

        return SecondaryParameters;
    }


// creation of a new folder at the safe directory, including naming and check for pre-existing folders with the same name

    private String[] createDirectory(String title) {

        String [] createDirectory = new String[3];

        String updated_title = title;

        String folder_directory = IJ.getDirectory("Choose Folder Directory") + title;
        File folder = new File(folder_directory);
        boolean folder_exists_already = folder.exists();

        int folder_directory_appendix = 2;
        String updated_folder_directory = folder_directory;

        while (folder_exists_already) {
            updated_folder_directory = folder_directory + "_" + folder_directory_appendix;
            updated_title = title + "_" + folder_directory_appendix;
            folder = new File(updated_folder_directory);
            folder_exists_already = folder.exists();

            if (folder_exists_already) {
                folder_directory_appendix++;
            }
        }

        boolean folder_successfully_created = folder.mkdir();

        String folder_status = "created";

        if (!folder_successfully_created) {
            IJ.error("Error Message", "Folder could not be created.");
            folder_status = "not created";
        }

        createDirectory[0] = updated_folder_directory + System.getProperty("file.separator");
        createDirectory[1] = updated_title;
        createDirectory[2] = folder_status;

        return createDirectory;
    }


// function for a rotation from horizontal to vertical using TransformJ Turn

    private ImagePlus rotate_horizontal2vertical(ImagePlus activeImage, boolean Z_scan_into_tissue) {

        if (Z_scan_into_tissue) {
            IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=90");
        }

        if (!Z_scan_into_tissue) {
            IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=270");
        }

        activeImage.changes = false;
        activeImage.close();
        activeImage = WindowManager.getCurrentImage();

        IJ.run(activeImage, "TransformJ Turn", "z-angle=180 y-angle=0 x-angle=0");

        activeImage.changes = false;
        activeImage.close();
        activeImage = WindowManager.getCurrentImage();

        return activeImage;
    }


// function for a rotation from vertical to horizontal using TransformJ Turn

    private ImagePlus rotate_vertical2horizontal(ImagePlus activeImage, boolean Z_scan_into_tissue) {

        IJ.run(activeImage, "TransformJ Turn", "z-angle=180 y-angle=0 x-angle=0");

        activeImage.changes = false;
        activeImage.close();
        activeImage = WindowManager.getCurrentImage();

        if (Z_scan_into_tissue) {
            IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=270");
        }

        if (!Z_scan_into_tissue) {
            IJ.run(activeImage, "TransformJ Turn", "z-angle=0 y-angle=0 x-angle=90");
        }

        activeImage.changes = false;
        activeImage.close();
        activeImage = WindowManager.getCurrentImage();

        return activeImage;
    }


// function for a maximum-projection

    private ImagePlus maxProjection(ImagePlus  activeImage) {

        ImagePlus projection = ZProjector.run(activeImage, "max");
        projection.show();
        activeImage.changes = false;
        activeImage.close();

        return projection;
    }


// function for a sum-projection

    private ImagePlus sumProjection(ImagePlus  activeImage) {

        ImagePlus projection = ZProjector.run(activeImage, "sum");
        IJ.run(projection, "royal", "");
        projection.show();
        activeImage.changes = false;
        activeImage.close();
        return projection;
    }


// function to get the cursor position within an image canvas

    private int[] getCursorLocation(ImageCanvas activeImageCanvas) {

        Point CursorPosition = activeImageCanvas.getCursorLoc();

        int [] CursorValues = new int[3];

        CursorValues[0] = CursorPosition.x;
        CursorValues[1] = CursorPosition.y;
        CursorValues[2] = activeImageCanvas.getModifiers();

        return CursorValues;
    }


// function for polar transformation

    private ImagePlus transform_polarTransformation(ImagePlus inputImage, int stack_size, double[] CenterPointX_array, double[] CenterPointY_array, double[] radius_array) {

        ImageStack inputImageStack = inputImage.getStack();
        int inputImage_width = inputImage.getWidth();
        int inputImage_height = inputImage.getHeight();

        double[][][] radius_unrolled_array = new double[stack_size][inputImage_height][inputImage_width];
        double[][][] phi_array = new double[stack_size][inputImage_height][inputImage_width];

        for (int new_z = 0; new_z < stack_size; new_z++) {

            for (int new_y = 0; new_y < inputImage_height; new_y++) {


                for (int new_x = 0; new_x < inputImage_width; new_x++) {

                    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));
                    phi_array[new_z][new_y][new_x] = Math.atan2((new_y - CenterPointY_array[new_z]), (new_x - CenterPointX_array[new_z]));

                }
            }
        }

        int[][][] temp_unrolled_y = new int[stack_size][inputImage_height][inputImage_width];
        int[][][] temp_unrolled_x = new int[stack_size][inputImage_height][inputImage_width];
        double [][][] value = new double[stack_size][inputImage_height][inputImage_width];
        int temp_unrolledImage_height = 10000000;
        int temp_unrolledImage_width = 5000000;

        for (int new_z = 0; new_z < stack_size; new_z++) {

            for (int new_y = 0; new_y < inputImage_height; new_y++) {

                for (int new_x = 0; new_x < inputImage_width; new_x++) {

                    value[new_z][new_y][new_x] = inputImageStack.getVoxel(new_x, new_y, new_z);

                    if (value[new_z][new_y][new_x] != 0) {

                        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]));
                        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]);
                    }
                }

            }
        }

        double min_temp_unrolled_y = temp_unrolledImage_height;
        double max_temp_unrolled_y = 0;
        double min_temp_unrolled_x = temp_unrolledImage_width;
        double max_temp_unrolled_x = 0;

        for (int new_z = 0; new_z < stack_size; new_z++) {

            for (int new_y = 0; new_y < inputImage_height; new_y++) {

                for (int new_x = 0; new_x < inputImage_width; new_x++) {

                    if (max_temp_unrolled_y < temp_unrolled_y[new_z][new_y][new_x]) {
                        max_temp_unrolled_y = temp_unrolled_y[new_z][new_y][new_x];
                    }

                    if (min_temp_unrolled_y > temp_unrolled_y[new_z][new_y][new_x] && temp_unrolled_y[new_z][new_y][new_x] != 0) {
                        min_temp_unrolled_y = temp_unrolled_y[new_z][new_y][new_x];
                    }

                    if (max_temp_unrolled_x < temp_unrolled_x[new_z][new_y][new_x]) {
                        max_temp_unrolled_x = temp_unrolled_x[new_z][new_y][new_x];
                    }

                    if (min_temp_unrolled_x > temp_unrolled_x[new_z][new_y][new_x] && temp_unrolled_x[new_z][new_y][new_x] != 0) {
                        min_temp_unrolled_x = temp_unrolled_x[new_z][new_y][new_x];
                    }
                }
            }
        }

        int unrolledImage_height = (int) Math.round(max_temp_unrolled_y - min_temp_unrolled_y) + 1;
        int unrolledImage_width = (int) Math.round(max_temp_unrolled_x - min_temp_unrolled_x) + 1;

        int[][][] unrolled_y_array = new int[stack_size][inputImage_height][inputImage_width];
        int[][][] unrolled_x_array = new int[stack_size][inputImage_height][inputImage_width];

        for (int new_z = 0; new_z < stack_size; new_z++) {

            for (int new_y = 0; new_y < inputImage_height; new_y++) {

                for (int new_x = 0; new_x < inputImage_width; new_x++) {

                    if (value[new_z][new_y][new_x] != 0) {

                        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]));
                        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]);
                    }
                }
            }
        }

        int min_unrolled_y = unrolledImage_height;
        int min_unrolled_x = unrolledImage_width;

        for (int new_z = 0; new_z < stack_size; new_z++) {

            for (int new_y = 0; new_y < inputImage_height; new_y++) {

                for (int new_x = 0; new_x < inputImage_width; new_x++) {

                    if (min_unrolled_y > unrolled_y_array[new_z][new_y][new_x] && unrolled_y_array[new_z][new_y][new_x] != 0) {
                        min_unrolled_y = unrolled_y_array[new_z][new_y][new_x];
                    }

                    if (min_unrolled_x > unrolled_x_array[new_z][new_y][new_x] && unrolled_x_array[new_z][new_y][new_x] != 0) {
                        min_unrolled_x = unrolled_x_array[new_z][new_y][new_x];
                    }
                }
            }
        }

        ImagePlus unrolledImage = IJ.createImage("Output Image", unrolledImage_width, unrolledImage_height, stack_size, 16);
        ImageStack unrolledImageStack = unrolledImage.getStack();

        for (int new_z = 0; new_z < stack_size; new_z++) {

            for (int new_y = 0; new_y < inputImage_height; new_y++) {

                for (int new_x = 0; new_x < inputImage_width; new_x++) {

                    if (value[new_z][new_y][new_x] != 0) {

                        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])));
                        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]));


                        if (unrolledImageStack.getVoxel(unrolled_x, unrolled_y, new_z) == 0) {
                            unrolledImageStack.setVoxel(unrolled_x, unrolled_y, new_z, value[new_z][new_y][new_x]);
                        } else {
                            double existingPixel = unrolledImageStack.getVoxel(unrolled_x, unrolled_y, new_z);
                            unrolledImageStack.setVoxel(unrolled_x, unrolled_y, new_z, existingPixel + value[new_z][new_y][new_x]);
                        }
                    }
                }
            }
        }
        inputImage.close();
        return unrolledImage;
    }

}
```