8. Annex: Beanshell code

```
////////////////////////////////////////////////////////////////////////////////////////
// Name         : CAM_Anaylzer.ijm
// Creator      : Jalil Zerdani, master student, Medicine,UNIL
// Purpose      : To segment and analyze 8-bit images of blood vessels in CAM experiments
// Creation date : 27.08.14
// Mod date     :
// Reason       :
/////////////////////////////////////////////////////////////////////////////////////////

// Plugin initialization
print("Running");
run("Action Bar","/plugins/ActionBar/CamAnalyzer/CAM_Analyzer.ijm");
exit();

<beanshell> //Setup Beanshell

/********************************************************/
// Menu
/********************************************************/
<text><html><font size=FSL color=#1E8E22>CAM Analyzer

/********************************************************/
// Selects the root directory containing the "orig" folder (button)
/********************************************************/
<line>
<button>
label=1 Select Images Directory
arg=<bsh>
        // Dependencies
        import ij.IJ;
        import ij.*;
        import ij.ImagePlus;
        import ij.plugin.Duplicator;
        import ij.plugin.ImageCalculator;
        import ij.WindowManager;
        import ij.plugin.frame.RoiManager;
        import ij.util.ArrayUtil;
        import features.TubenessProcessor;
        import ij.gui.WaitForUserDialog;
        import ij.plugin.frame.RoiManager;
        import ij.plugin.filter.GaussianBlur;
        import ij.plugin.frame.ThresholdAdjuster;
        import ij.io.DirectoryChooser;
        import java.io.File;
        import ij.Prefs;
        import ij.gui.HistogramWindow;
        import ij.gui.Toolbar;
        import ij.measure.ResultsTable;
        import ij.process.ImageStatistics;

        /********************************************************/
        // Creates a new directory
        // String path: the absolute path where to create the directory
        /********************************************************/
        void createDir(String path)
        {
                File dir = new File(path);
                dir.mkdir();
                IJ.log("Directory " + dir + " created");
        }

        // Asks the user to point the root directory
        IJ.log("***** CAM Analyzer Loaded *****");
        DirectoryChooser dc = new DirectoryChooser("Select where are your images to preprocess");
        String rootDir = dc.getDirectory();
        IJ.log("Root Directory for images: " + rootDir);
        // Windows version
        // Directories used by CAM Analyzer
        //origImagesPath = dc.getDirectory() +"orig\\";                                  // Original images
        //preprocDirPath = dc.getDirectory() + "1 preprocessed\\";          // Preprocessed images
        //tubenessDirPath = dc.getDirectory() + "2 tubeness\\";                   // Tubeness images
        //maxProjDirPath = dc.getDirectory() + "3 maxProj\\";                     // Maximum projection of tubenesses
        //thresholdedDirPath = dc.getDirectory() + "4 thresholded\\";       // Thresholded images
        //cleanedDirPath = dc.getDirectory() + "5 cleaned\\";                     // Images without small particles
        //skeletonDirPath = dc.getDirectory() + "6 skeletonized\\";         // Skeletonized images
        //localThickDirPath = dc.getDirectory() + "7 localThickness\\";     // Local thickness images
        //thickDistDirPath = dc.getDirectory() + "8 thicknessDist\\";  // Distribution of local thicknesses
        //resultsDirPath = dc.getDirectory() + "Results\\";                       // Root dir for results of analyses
        //skeletonResultsDirPath = resultsDirPath + "skeleton\\";           // Results of skeleton analyses
        //thickDistResultsDirPath = resultsDirPath + "thickDist\\";         // Results of thickness distributions
        //shollResultsDirPath = resultsDirPath + "sholl\\";                       // Results of Sholl analysis
        //vesselAreaDirPath = resultsDirPath + "vesselArea\\";                    // Results for the area covered by vessels

        //Mac version
        // Directories used by CAM Analyzer
        origImagesPath = dc.getDirectory() +"orig/";                              // Original images
        preprocDirPath = dc.getDirectory() + "1 preprocessed/";             // Preprocessed images
        tubenessDirPath = dc.getDirectory() + "2 tubeness/";                // Tubeness images
        maxProjDirPath = dc.getDirectory() + "3 maxProj/";                  // Maximum projection of tubenesses
        thresholdedDirPath = dc.getDirectory() + "4 thresholded/";     // Thresholded images
        cleanedDirPath = dc.getDirectory() + "5 cleaned/";                  // Images without small particles
        skeletonDirPath = dc.getDirectory() + "6 skeletonized/";       // Skeletonized images
        localThickDirPath = dc.getDirectory() + "7 localThickness/";// Local thickness images
        thickDistDirPath = dc.getDirectory() + "8 thicknessDist/";     // Distribution of local thicknesses
        resultsDirPath = dc.getDirectory() + "Results/";                    // Root dir for results of analyses
        skeletonResultsDirPath = resultsDirPath + "skeleton/";              // Results of skeleton analyses
        thickDistResultsDirPath = resultsDirPath + "thickDist/";       // Results of thickness distributions
        shollResultsDirPath = resultsDirPath + "sholl/";                    // Results of Sholl analysis
        vesselAreaDirPath = resultsDirPath + "vesselArea/";                       // Results for the area covered by vessels

        // Gets the ROI Manager for the outlining of the scaffold
        RoiManager roiMan = RoiManager.getInstance();
        if(roiMan == null) {
                roiMan = new RoiManager();
        }

        // Resetting and hiding the ROI manager for better usability
        roiMan.reset();
        roiMan.hide();
</bsh>
</line>

/******************************************************/
// Preprocesses the images (button)
/******************************************************/
<line>
<button>
label=2 Preprocess images
arg=<bsh>

        /******************************************************/
        // Normalization of image between 0 and 1 (for thresholding)
        // ImagePlus imp: the image to normalize
        // return the normalized image
        /******************************************************/
        ImagePlus normalizeImp(ImagePlus imp) {

                // Formula for normalization: y = (x - min)/max - min
                stats = imp.getStatistics();
                max = stats.max;
                min = stats.min;
                factor = max-min;
                IJ.run(imp, "Subtract...", "value=" + min);
                IJ.run(imp, "Divide...", "value=" + factor);

                return imp;
        }

        sigmaBgr = 100;                         // Sigma value for blurring the background (Gaussian)
        createDir(preprocDirPath);      // Creates the directory for preprocessed images

        // We process each image in the "orig" folder
        File mainDir = new File(origImagesPath);
        File[] listOfFiles = mainDir.listFiles();
        for (int i = 0; i < listOfFiles.length; i++)
        {
                filename = listOfFiles[i].getName();                    // File name of original image

                // We avoid folders and "non-tifs" files
                if (filename.endsWith(".tif"))
                {
                        ic = new ImageCalculator();                                             // To remove the background
                        impBgr = new ImagePlus();                                               // Blurred background

                        // Removes the noise: 1px gaussian filter
                        IJ.log("Pre-processing of " + filename + " in progress");
                        impOrig = IJ.openImage(origImagesPath + filename);

                        // Remove egg shell fragments bright outlier, values found experimentally
                        IJ.run(impOrig, "Remove Outliers...", "radius=60 threshold=5 which=Bright");

                        // Removes the high frequency noise
                        IJ.run(impOrig, "Gaussian Blur...", "sigma=1");

                        // Removes the background
                        impBgr = new Duplicator().run(impOrig);
                        IJ.run(impBgr, "Gaussian Blur...", "sigma=" + sigmaBgr);
                        impBgrRemoved = ic.run("Divide create 32-bit", impOrig, impBgr);

                        // Normalizes the values between 0 and 1 to compare images between them
                        impPreprocessed = normalizeImp(impBgrRemoved);

                        // Saves the image in the preprocessed folder
                        IJ.saveAs(impPreprocessed, "Tiff", preprocDirPath + filename);

                        // Get back memory
                        impOrig.close();
                        impBgr.close();
                        impPreprocessed.close();

                        IJ.log("Pre-processing of " + filename + " done");
                        IJ.log("");
                }
        }

        IJ.log("All images have been successfully pre-processed");
</bsh>
</line>

/********************************************************/
// Allow the user to outline the scaffold manually (button)
/********************************************************/
<line>
<button>
label=3 Outline Scaffold
arg=<bsh>

        // We process each image in the "preprocess" folder
        File preprocDir = new File(preprocDirPath);
        File[] listOfFiles = preprocDir.listFiles();
```

```
    for (int i = 0; i < listOfFiles.length; i++)
    {
        filename = listOfFiles[i].getName(); // File name of preprocessed image

        // We avoid folders and "non-tifs" files
        if (filename.endsWith(".tif"))
        {
            IJ.log("Pre-processing of " + filename + " in progress");
            impPreprocessed = IJ.openImage(preprocDirPath + filename);
            impPreprocessed.show();

            // Message for the user
            scaffoldDlg = new WaitForUserDialog("Scaffold Outline", "Outline the scaffold with a margin using a circle if
needed");
            scaffoldDlg.show();

            // We verify that the "oval" tool is selected
            if(Toolbar.getToolId() == Toolbar.OVAL || Toolbar.getToolId() == -1)
            {
                // Saves the image in the preprocessed folder
                IJ.saveAs(impPreprocessed, "Tiff", preprocDirPath + filename);
                impPreprocessed.close();
            }
            impPreprocessed.close();
        }
    }

    IJ.log("Scaffold Outline Done!");
</bsh>
</line>

/*********************************************************/
// Segments the blood vessels in the image
/*********************************************************/
<line>
<button>
label=4 Detect blood vessels
arg=<bsh>

    /*********************************************************/
    // Extracts the tube-like shapes in an image
    // ImagePlus impOrig: the image where to look for tubes
    // double sigma     : diameter of tubes to extract
    // returns the tubeness image
    /*********************************************************/
    ImagePlus tubenessImp (ImagePlus impOrig, double sigma)
    {
        // Inverts the image for the tubeness
        ImagePlus imp = new Duplicator().run(impOrig);
        Prefs.blackBackground = true;
        imp.hide();
        IJ.run(imp, "Invert", "");

        // Takes the tubeness in the image
        TubenessProcessor tp = new TubenessProcessor(sigma, true);
        ImagePlus impTube = tp.generateImage(imp);
        impTube.setTitle("Tubeness of "+ imp.getTitle());

        return impTube;
    }

    /*********************************************************/
    // Removes particles smaller than a given size
    // ImagePlus imp   : image to clean
    // int particleSize: all particles smaller than this are removed
    // returns the cleaned image
    /*********************************************************/
```

```
ImagePlus cleanArtefacts (ImagePlus imp, int particleSize)
{
		//impCleaned = new Duplicator().run(imp);
		IJ.run(imp, "Invert", "");
		IJ.run(imp, "Analyze Particles...", "size=" + particleSize + "-Infinity show=Masks");

		return imp;
}

/***********************************************/
// Transforms an 8-bit grey image in an 8-bit black & white image
// with a given thresholding method
// ImagePlus imp    : image to be thresholded
// String method    : the method found by ThresholFinder to fit the best
// double intercept : the intercept value given by ThresholdFinder
// double slope     : the slope value given by ThresholdFinder
// returns the thresholded image
/***********************************************/
ImagePlus thresholdImage (ImagePlus imp, String method, double intercept, double slope)
{
		double min = 0.0;	// The minimum value of the threshold
		double max = 0.0;	// The maximum value of the threshold
		double max2 = 0.0;  // The new maximum value of the threshold

		// Uses the method, the intercept and slope values found
		// by ThresholdFinder to threshold the image
		Prefs.blackBackground = true;
		IJ.setAutoThreshold(imp, method);
		min = imp.getProcessor().getMinThreshold();
		max = imp.getProcessor().getMaxThreshold();
		max2 = (slope * max) + intercept;
		IJ.setThreshold(imp, min, max2);
		IJ.run(imp, "Convert to Mask", "");

		return imp;
}

/***********************************************/
// Segments the vessels in the image
// returns the thresholded image
/***********************************************/
void detectVessels()
{
		sigmaFactor = 1.61;			// Linear factor found experimentally
		defaultSigma = 0.1442;		// Default value of sigma from tubeness module

		// We process all images in the "preprocess" directory
		File preprocDir = new File(preprocDirPath);
		File[] listOfFiles = preprocDir.listFiles();

		// Creates the directories for tubeness and the maximum projection of tubenesses
		createDir(tubenessDirPath);
		createDir(maxProjDirPath);

		// We produce 5 images with different sigma values that will be
		// put together to make a maximum projection image
		for (int i = 0; i < listOfFiles.length; i++)
		{
				filename = listOfFiles[i].getName(); // File name of original image

				// We avoid folders and "non-tifs" files
				if (filename.endsWith(".tif"))
				{
						IJ.log("Extraction of blood vessels in " + filename + " in progress");
						impPreproc = IJ.openImage(preprocDirPath + filename);

						// To save the images
```

```
            // Saves the ROI to put it back after the processing
            scaffoldRoi = impPreproc.getRoi();
            if (scaffoldRoi != null) {
                IJ.log("there is a ROI");

                //Remove ROI to make sure processing is on whole image
                impPreproc.deleteRoi();
            }

            int j = 1;  // loop counter for faster convergence

            do
            {
                // Value of the slices where the vessels are the best detected
                sliceBest = 5.1429 * j * j - 7.6571 * j + 10.4;

                // Sigma for tubeness
                sigma = sliceBest * sigmaFactor * defaultSigma;

                // Takes the tubeness
                impTube = tubenessImp(impPreproc, sigma);

                // Intensity that is varying along with sigma found by regression
                normIntensity = (0.754 * Math.log(sigma)) + 0.0077;

                // To take into account that the bigger the sigma, the higher the intensity
                // we normalize the image by its value
                IJ.run(impTube, "Divide...", "value=" + normIntensity);

                // Saves the tubeness image
                IJ.saveAs(impTube, "Tiff", tubenessDirPath + filenameNoTif + " " + j);

                // Get back memory
                impPreproc.close();
                impTube.close();

                IJ.log("Vessels in " + filename + " detected");
                IJ.log("");

                j++;
            } while (sliceBest < 100);

            // We take the 5 images produced to assemble them using the maximum Z projection
            filenameNoTif = filename.substring(0, filename.length() - 4);
            IJ.run("Image Sequence...", "open=[" + tubenessDirPath + filenameNoTif + " 1.tif] sort");
            IJ.run("Z Project...", "projection=[Max Intensity]");

            // Puts back the ROI after processing
            if (scaffoldRoi != null)
                IJ.getImage().setRoi(scaffoldRoi);

            // Saves the maximum projection of the 5 tubeness images
            IJ.saveAs("Tiff", maxProjDirPath + filename);
            IJ.run("Close All");

            // Deletes the 5 tubeness images to make room to the next round of tubeness
            File tubenessDir = new File(tubenessDirPath);
            File[] tubenessFiles = tubenessDir.listFiles();
            for (int i = 0; i < tubenessFiles.length; i++)
                rm(tubenessDirPath + tubenessFiles[i].getName());

            IJ.log("Extraction of blood vessels in " + filename + " done");
            IJ.log("");
        }
    }
    IJ.log("Blood vessels extracted in all images");
}
```

```
/********************************************************/
// Thresholds the images based on the method and values found
// with ThresholdFinder
/********************************************************/
void threshold()
{
        // creates the directory for the thresholded images
        createDir(thresholdedDirPath);

        double a = -0.003;                                   // Intercept found with ThresholdFinder
        double b = 1.305;                                            // Slope found with ThresholdFinder
        String ThresholdMethod = "Li Dark";        // Thresholding method found with TresholdFinder

        // We threshold the maximum projection of images
        File maxProjDir = new File(maxProjDirPath);
        File[] listOfFiles = maxProjDir.listFiles();
        for (int i = 0; i < listOfFiles.length; i++)
        {
                filename = listOfFiles[i].getName(); // File name of max projection image

                // We avoid folders and "non-tifs" files
                if (filename.endsWith(".tif"))
                {
                        IJ.log("Thresholding of " + filename + " in progress");
                        impMaxProj = IJ.openImage(maxProjDirPath + filename);

                        // Saves the ROI to put it back after the processing
                        scaffoldRoi = impMaxProj.getRoi();
                        if (scaffoldRoi != null) {
                                IJ.log("there is a ROI");

                                //Remove ROI to make sure processing is on whole image
                                impMaxProj.deleteRoi();
                        }

                        // Thresholds the image
                        impThresholded = thresholdImage (impMaxProj, ThresholdMethod, a, b);

                        // Puts back the ROI after processing
                        if (scaffoldRoi != null)
                                impThresholded.setRoi(scaffoldRoi);

                        // Saves the thresholded image
                        IJ.saveAs(impThresholded, "Tiff", thresholdedDirPath + filename);

                        // Get back memory
                        impMaxProj.close();
                }
        }
        IJ.log("All images thresholded successfully");
}


/********************************************************/
// Cleans the thresholded image of small particles
/********************************************************/
void clean()
{
        // Creates the directory for cleaned images
        createDir(cleanedDirPath);

        // Cleans all the images in the thresholded folder
        File thresholdedDir = new File(thresholdedDirPath);
        File[] listOfFiles = thresholdedDir.listFiles();

        for (int i = 0; i < listOfFiles.length; i++)
        {
            filename = listOfFiles[i].getName(); // File name of thresholded image

            // We avoid folders and "non-tifs" files
            if (filename.endsWith(".tif"))
            {
                IJ.log("Cleaning of " + filename + " in progress");
                impThresholded = IJ.openImage(thresholdedDirPath + filename);

                // Saves the ROI to put it back after the processing
                scaffoldRoi = impThresholded.getRoi();

                if (scaffoldRoi != null) {

                    IJ.log("there is a ROI");
                    //Remove ROI to make sure processing is on whole image
                    impThresholded.deleteRoi();
                }

                // Remove small artefacts
                impCleaned = cleanArtefacts(impThresholded, 300);
                impMaskCleaned = IJ.getImage();

                // Puts back the ROI after processing and
                // removes the artefacts coming fom the scaffold
                if (scaffoldRoi != null)
                {
                    IJ.run(impMaskCleaned, "Invert", "");
                    impMaskCleaned.setRoi(scaffoldRoi);
                    IJ.run(impMaskCleaned, "Cut", "");
                    impMaskCleaned.deleteRoi();
                    IJ.run(impMaskCleaned, "Invert", "");
                    impMaskCleaned.setRoi(scaffoldRoi);
                }

                // Saves the cleaned image
                IJ.saveAs(impMaskCleaned, "Tiff", cleanedDirPath + filename);

                // Get back memory
                impThresholded.close();
                impCleaned.close();
                impMaskCleaned.close();
            }
        }

        IJ.log("All images cleaned successfully");
    }

/********************************************************/
// Main for vessel detection
/********************************************************/
    detectVessels();    // Segments vessels
    threshold();            // Threholds the images
    clean();                // Cleans the small particles (artifacts)
</bsh>
</line>

/********************************************************/
// Extract numerical values from thresholded & cleaned images (button)
/********************************************************/
<line>
<button>
label=5 Analyze blood vessels
arg=<bsh>

    /********************************************************/
    // Measures the area in pixels occupied by vessels
    /********************************************************/
    void measureVesselArea() {
```

```
    // Creates the directory to save the results
    createDir(vesselAreaDirPath);

    // Calculates the area taken by the vessels from cleaned images
    File cleanedDir = new File(cleanedDirPath);
    File[] listOfFiles = cleanedDir.listFiles();

    for (int i = 0; i < listOfFiles.length; i++)
    {
        filename = listOfFiles[i].getName(); // File name of cleaned image

        // We avoid folders and "non-tifs" files
        if (filename.endsWith(".tif"))
        {
            IJ.log("Measure area of " + filename + " in progress");

            impCleaned = IJ.openImage(cleanedDirPath + filename);

            //Remove ROI to make sure processing is on whole image
            if (impCleaned.getRoi() != null)
                impCleaned.deleteRoi();

            // Gets the histogram from cleaned images and saves the result
            histWindow = new HistogramWindow("Histogram", impCleaned, 255);
            ResultsTable results = new ResultsTable();
            results = histWindow.getResultsTable();
            filenameNoTif = filename.substring(0, filename.length() - 4);
            results.save(vesselAreaDirPath + filenameNoTif + ".xls");

            // Get back memory
            histWindow.close();
            impCleaned.close();
        }
    }

}

/*******************************************************/
// Gets the connectivity network (skeletonization) from
// thresholded and cleaned images to measure connectivity
/*******************************************************/
void skeletonize() {

    IJ.log("Skeletonization of images started");

    // Creates the directory for skeletonized images and results
    createDir(skeletonDirPath);
    createDir(skeletonResultsDirPath);

    // Skeletonize all cleaned images
    File cleanedDir = new File(cleanedDirPath);
    File[] listOfFiles = cleanedDir.listFiles();

    for (int i = 0; i < listOfFiles.length; i++)
    {
        filename = listOfFiles[i].getName(); // File name of skeletonized image

        // We avoid folders and "non-tifs" files
        if (filename.endsWith(".tif"))
        {
            IJ.log("Skeletonization of " + filename + " in progress");
            impCleaned = IJ.openImage(cleanedDirPath + filename);

            // Save ROI somewhere if it's there (Oli)
            scaffoldRoi = impCleaned.getRoi();

            // Saves the ROI to put it back after the processing
            if (scaffoldRoi != null) {
                IJ.log("there is a ROI");

                //Remove ROI to make sure processing is on whole image
                impCleaned.deleteRoi();
            }

            // Skeletonize needs a black on white image
            IJ.run(impCleaned, "Invert", "");
            IJ.run(impCleaned, "Skeletonize", "");
            IJ.run(impCleaned, "Invert", "");

            // Puts back the ROI after processing
            if (scaffoldRoi != null)
                impCleaned.setRoi(scaffoldRoi);

            // Saves the skeletonized image
            IJ.saveAs(impCleaned, "Tiff", skeletonDirPath + filename);

            //Analysis of skeleton
            IJ.run(impCleaned, "Analyze Skeleton (2D/3D)", "prune=none");

            // Saves the results
            filenameNoTif = filename.substring(0, filename.length() - 4);
            IJ.saveAs("Results", skeletonResultsDirPath + filenameNoTif + ".xls");

            // Get back memory
            IJ.run(impCleaned, "Close All", "");
            IJ.log("Skeletonization of " + filename + " done");
        }
    }

    IJ.log("All images skeletonized successfully");
}

/*********************************************************/
// Gets the local thickness to measure vessel diameters
/*********************************************************/
void getLocalThickness() {

    IJ.log("Calculation of local thickness for all images started");

    // Creates the directory for the local thickness images
    createDir(localThickDirPath);

    // Gets the local thickness from all cleaned images
    File cleanedDir = new File(cleanedDirPath);
    File[] listOfFiles = cleanedDir.listFiles();

    for (int i = 0; i < listOfFiles.length; i++)
    {
        filename = listOfFiles[i].getName(); // File name of local thickness image

        // We avoid folders and "non-tifs" files
        if (filename.endsWith(".tif"))
        {
            IJ.log("Calculation of local thickness of " + filename + " in progress");
            impCleaned = IJ.openImage(cleanedDirPath + filename);

            // Saves the ROI to put it back after the processing
            scaffoldRoi = impCleaned.getRoi();

            if (scaffoldRoi != null) {
                IJ.log("there is a ROI");

                //Remove ROI to make sure processing is on whole image
                impCleaned.deleteRoi();
            }
```

```
                // Gets the local thickness
                IJ.run(impCleaned, "Local Thickness (complete process)", "threshold=128");
                impLocalThick = IJ.getImage();

                // Puts back the ROI after processing
                if (scaffoldRoi != null)
                    impLocalThick.setRoi(scaffoldRoi);

                // Saves the local thickness image
                IJ.saveAs(impLocalThick, "Tiff", localThickDirPath + filename);

                // Get back memory
                IJ.run(impCleaned, "Close All", "");
                impCleaned.close();
                impLocalThick.close();

                IJ.log("Local thickness of " + filename + " calculated");
            }
        }

        IJ.log("Local thickness calculated for all images successfully");
}

/*******************************************************/
// Gets the distribution of diameters of blood vessels
/*******************************************************/
void getThicknessDistribution() {

        int nBins = 10;                     // Nb of bins for the distribution (histogram)

        IJ.log("Calculation of thickness distribution for all images started");

        // Creates directories for the thickness distribution and results
        createDir(thickDistDirPath);
        createDir(thickDistResultsDirPath);

        // Goes through all skeletonized images to get the distribution
        File skeletonDir = new File(skeletonDirPath);
        File[] listOfFiles = skeletonDir.listFiles();

        for (int i = 0; i < listOfFiles.length; i++)
        {
            filename = listOfFiles[i].getName(); // File name of cleaned image

            // We avoid folders and "non-tifs" files
            if (filename.endsWith(".tif"))
            {
                IJ.log("Calculation of local thickness of " + filename + " in progress");

                impSkeleton = IJ.openImage(skeletonDirPath + filename);
                impLocalThick = IJ.openImage(localThickDirPath + filename);

                // Saves the ROI to put it back after the processing
                scaffoldRoi = impSkeleton.getRoi();

                if (scaffoldRoi != null) {
                    IJ.log("there is a ROI");

                    //Remove ROI to make sure processing is on whole image
                    impSkeleton.deleteRoi();
                    impLocalThick.deleteRoi();
                }

                // Since the thickness is represented on a color map
                // the multiplication of the skeleton (1pix diameter) by the local thickness
                // allows to get the real thickness distribution
                ic = new ImageCalculator();
            impThickDist = ic.run("Multiply create 32-bit",impLocalThick, impSkeleton);
            IJ.saveAs(impThickDist, "Tiff", thickDistDirPath + filename);

            // Saves the histogram
            histWindow = new HistogramWindow("Histogram", impThickDist, nBins);
            ResultsTable results = new ResultsTable();
            results = histWindow.getResultsTable();
            filenameNoTif = filename.substring(0, filename.length() - 4);
            results.save(thickDistResultsDirPath + filenameNoTif + ".xls");

            // Get back memory
            histWindow.close();
            impLocalThick.close();
            impThickDist.close();
            impSkeleton.close();
        }
    }
}

/********************************************************/
// Gets the Sholl analysis
/********************************************************/
getShollAnalysis()
{
    // Creates the directory for the Sholl analysis
    createDir(shollResultsDirPath);

    // Gets the Sholl analysis from all skeletonized images
    File skeletonDir = new File(skeletonDirPath);
    File[] listOfFiles = skeletonDir.listFiles();

    for (int i = 0; i < listOfFiles.length; i++)
    {
        filename = listOfFiles[i].getName(); // File name of cleaned image

        // We avoid folders and "non-tifs" files
        if (filename.endsWith(".tif"))
        {
            IJ.log("Calculation of Sholl Analysis " + filename + " in progress");

            impSkeleton = IJ.openImage(skeletonDirPath + filename);
            impSkeleton.show();

            // Saves the ROI to put it back after the processing
            scaffoldRoi = impSkeleton.getRoi();

            // To find the center of the image
            impWidth = impSkeleton.getWidth();
            impHeight = impSkeleton.getHeight();

            // If there is a scaffold outline, we choose the center of the scaffold
            // as the start of the Sholl analysis
            if (scaffoldRoi != null) {

                centerX = scaffoldRoi.getBounds().getX() + scaffoldRoi.getBounds().getWidth() / 2;
                centerY = scaffoldRoi.getBounds().getY() + scaffoldRoi.getBounds().getHeight() / 2;

                impSkeleton.deleteRoi();
                impSkeleton.setRoi(new Line(centerX, centerY, impWidth / 2, 0));

                // To get the radius from the center of scaffold to the border of the image
                distX = Math.abs(centerX - impWidth / 2);
                distY = impHeight / 2;
                shollRadius = Math.sqrt(Math.pow(distX, 2) + Math.pow(distY, 2));

            // If there is no scaffold outline, the center for the start of the
            // Sholl analysis is the center of the image
            } else {
```

```
                                                  impSkeleton.setRoi(new Line(impWidth/2, impHeight/2, impWidth / 2, 0));
                                                  shollRadius = impHeight/2;
                                        }

                                        // Get the Sholl Analysis and saves it
                                        IJ.run("Clear Results");
                                        IJ.run(impSkeleton, "Sholl Analysis...", "starting=10 ending=" + shollRadius + " radius_step=0
#_samples=1 integration=Mean enclosing=1 #_primary=4 fit linear polynomial=[Best fitting degree] most log-log normalizer=Area
background=228");
                                        filenameNoTif = filename.substring(0, filename.length() - 4);
                                        IJ.selectWindow(filenameNoTif + "_Sholl-Profiles");
                                        IJ.saveAs("Results", shollResultsDirPath + filenameNoTif + ".xls");

                                        // Get back memory
                                        impSkeleton.close();
                                        IJ.run("Close All", "");
                              }
                    }

                    IJ.log("Sholl analysis calculated for all images successfully");
          }

          // Main for the data analysis
          createDir(resultsDirPath);    // Creates the folder for the results
          measureVesselArea();                    // Measures the area occupied by blood vessels
          skeletonize();                                    // Skeletonize the images
          getLocalThickness();                    // Gets the local thickness
          getThicknessDistribution();   // Gets the distribution of local thicknesses
          getShollAnalysis();                     // Gets the Sholl analysis
</bsh>
</line>
```

9. Annex: Software Manual

1. Install Fiji 1.50b
2. Install Actionbar
3. Copy the "CAM_Analyzer.ijm" file in the Fiji subfolder "/plugin/ActionBar/"
4. Create a parent folder with the date of the experiment: "Day-Month-Year"
5. Inside parent folder, create a folder called "orig"
6. Copy the images to be analyzed in this "orig" folder
7. Launch Fiji
8. Lauch CAM Analyzer from the menu "Plugins" ➔ Action Bar ➔ Cam Analyzer
9. Click on "1 Select Images Directory" and select the "Day-Month-Year" folder
10. Click on "2 Preprocess Images" and wait until the end of the process
11. If the images contain scaffolds, click on "Outline scaffold", then when an image contains a scaffold outline it with a small margin around it using the "oval" selecting tool of Fiji and the "shift" key (Mac OSX) to draw a perfect circle.
12. If there are no scaffold or your are finished with outlining the images, click on "Detect Blood Vessels" and wait until the end of the process
13. Once the detection is done, you can click on "Analyze Blood Vessels" and wait until the end of the process
14. All images obtained at each step can be found in the subfolders of the "Day-Month-Year" folder. All results are found as Excel files in the "Results" subfolder.