OpenCASA

v0.8

Generated by Doxygen 1.8.12

Fri Nov 10 2017 13:51:20

Contents

Chapter 1

Namespace Index

1.1 Packages

Here are the packages with brief descriptions (if available):

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Package analysis

Classes

- class Chemotaxis
- class Motility

5.2 Package data

Classes

- class OscillatoryWalker
- class Params
- class PersistentRandomWalker
- class SerializableList
- class Simulation
- class Spermatozoon
- class Trial

5.3 Package functions

Classes

- class ComputerVision
- class FileManager
- class Kinematics
- class Paint
- class SignalProcessing
- class TrialManager
- class Utils
- class VideoRecognition

5.4 Package gui

Classes

- class ImageAnalysisWindow
- class MainWindow
- class MorphWindow
- class SettingsWindow
- class ViabilityWindow

Chapter 6

Class Documentation

6.1 ViabilityWindow.Channel Enum Reference

Public Attributes

- BLUE
- GREEN
- RED
- NONE

6.1.1 Detailed Description

Definition at line 41 of file ViabilityWindow.java.

6.1.2 Member Data Documentation

6.1.2.1 BLUE

`BLUE`

Definition at line 42 of file ViabilityWindow.java.

6.1.2.2 GREEN

`GREEN`

Definition at line 42 of file ViabilityWindow.java.

6.1.2.3 NONE

`NONE`

Definition at line 42 of file ViabilityWindow.java.

6.1.2.4 RED

`RED`

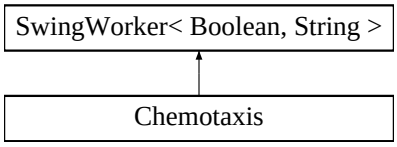
Definition at line 42 of file ViabilityWindow.java.

The documentation for this enum was generated from the following file:

- ViabilityWindow.java

6.2 Chemotaxis Class Reference

Inheritance diagram for Chemotaxis:



Classes

- enum TypeOfAnalysis

Public Member Functions

- Boolean doInBackground ()
- void selectAnalysis ()

Protected Member Functions

- void done ()

Private Member Functions

- ResultsTable analyseCondition (Map< String, Trial > controls, Map< String, Trial > tests)
- void analyseDirectory ()
- void analyseFile ()
- void analyseSimulations ()
- ResultsTable bootstrappingAnalysis (Map< String, Trial > controls, Map< String, Trial > tests)
- float calculateChIndex (List< List< Spermatozoon >> theTracks)
- float calculateSLIndex (List< List< Spermatozoon >> theTracks)
- boolean checkPairs (Map< String, Trial > controls, Map< String, Trial > tests)
- int [ ] circularHistogram (List< Double > angles, int n)
- int [ ] countAngles (SerializableList theTracks)
- int [ ] countInstantDisplacements (List< Spermatozoon > track)
- void drawResults (Trial trial)
- Trial findTrial (String id, Map< String, Trial > trials)

- Map< String, Trial > getControlTrials (String folder)
- List< Double > getListOfAngles (SerializableList theTracks)
- double [ ] getOddsValues (SerializableList tracks)
- List< String > getTestFolders (String folder)
- Map< String, Trial > getTrials (List< String > filenames)
- ResultsTable indexesAnalysis (Map< String, Trial > controls, Map< String, Trial > tests)
- SerializableList mergeTracks (Map< String, Trial > trials)
- int minSampleSize (Map< String, Trial > trials)
- double or (Trial control, Trial test)
- double orThreshold (Map< String, Trial > controls)
- double relativeAngle (Spermatozoon previous, Spermatozoon next)
- void setBootstrappingResults (ResultsTable rt, double or, double th, Trial trial)
- void setIndexesResults (ResultsTable rt, Trial trial, float chIdx, float slIdx)

Private Attributes

- TypeOfAnalysis analysis = TypeOfAnalysis.NONE

6.2.1 Detailed Description

This class implements all the functions related to chemotaxis analysis.

Author

Carlos Alquezar

Definition at line 48 of file Chemotaxis.java.

6.2.2 Member Function Documentation

6.2.2.1 analyseCondition()

```
ResultsTable analyseCondition (
            Map< String, Trial > controls,
            Map< String, Trial > tests )  [private]
```

This method runs the correct analysis looking at the analysis variable set by the user.

Parameters

controls	• control trials
tests	• test trials that are going to be compare with control trials

Returns

- ResultsTable with the results of the analysis

Definition at line 66 of file Chemotaxis.java.

6.2.2.2 analyseDirectory()

```
void analyseDirectory ( )  [private]
```

This method asks user for the main folder that contains the data that is going to be analysed. The content of this folder has to be one folder named "control" with the control videos, and one or more folders corresponding with each condition that user wants to compare with reference controls.

Definition at line 88 of file Chemotaxis.java.

6.2.2.3 analyseFile()

```
void analyseFile ( )  [private]
```

This method asks user for the file that is going to be analysed, extract the corresponding trial and show results.

Definition at line 111 of file Chemotaxis.java.

6.2.2.4 analyseSimulations()

```
void analyseSimulations ( )  [private]
```

This method asks user which simulation parameters have to be used in the simulations, generate the corresponding trials, analyse them and show results

Definition at line 128 of file Chemotaxis.java.

6.2.2.5 bootstrappingAnalysis()

```
ResultsTable bootstrappingAnalysis (
            Map< String, Trial > controls,
            Map< String, Trial > tests )  [private]
```

This method applies the bootstrapping analysis for the given set of trials.

Parameters

controls	• control trials
tests	• test trials that are going to be compare with control trials

Returns

ResultsTable with the bootstrapping analysis

Definition at line 156 of file Chemotaxis.java.

6.2.2.6 calculateChIndex()

```
float calculateChIndex (
            List< List< Spermatozoon >> theTracks )  [private]
```

This method calculates the CH-index for the given set of trajectories.

Parameters

theTracks	• 2D-ArrayList with all the tracks

Returns

Ch-Index

Definition at line 187 of file Chemotaxis.java.

6.2.2.7 calculateSLIndex()

```
float calculateSLIndex (
            List< List< Spermatozoon >> theTracks )  [private]
```

This method calculates the SL-index for the given set of trajectories.

Parameters

theTracks	• 2D-ArrayList that stores all the tracks

Returns

SL-Index

Definition at line 223 of file Chemotaxis.java.

6.2.2.8 checkPairs()

```
boolean checkPairs (
            Map< String, Trial > controls,
            Map< String, Trial > tests )  [private]
```

This method checks if at least exists one trial in each Hashmap with the same ID

Parameters

controls	- Set of control trials
tests	- Set of test trials

Returns

true if the method finds at least one pair with the same ID. Otherwise it returns false.

Definition at line 259 of file Chemotaxis.java.

6.2.2.9 circularHistogram()

```
int [] circularHistogram (
            List< Double > angles,
            int n )  [private]
```

This method calculates the histogram for the given set of angles.

Parameters

angles	• The angles (in degrees).
n	• Total number of bins for the histogram.

Returns

Array of the number of elements for each bin.

Definition at line 281 of file Chemotaxis.java.

6.2.2.10 countAngles()

```
int [] countAngles (
            SerializableList theTracks )  [private]
```

This method counts, for the given set of trajectories, how many angles go in the gradient direction and how many in the opposite (or other) direction, depending on the value of the parameter "Compare Opposite Directions" set by the user on the Settings Window.

Parameters

theTracks	• The array with all trajectories

Returns

An array containing the total count of angles ([0] - upgradient: [1] - other directions).

Definition at line 306 of file Chemotaxis.java.

6.2.2.11 countInstantDisplacements()

```
int [] countInstantDisplacements (
            List< Spermatozoon > track )  [private]
```

This method counts, for the given trajectory, how many angles go in the gradient direction and how many in the opposite (or other) direction, depending on the value of the parameter "Compare Opposite Directions" set by the user on the Settings Window.

Parameters

track	• The single trajectory to analyse

Returns

An array containing the total count of angles ([0] - upgradient: [1] - other directions).

Definition at line 328 of file Chemotaxis.java.

6.2.2.12 doInBackground()

```
Boolean doInBackground ( )
```

This method is inherit from SwingWorker class and it is the starting point after the execute() method is called.

Definition at line 362 of file Chemotaxis.java.

6.2.2.13 done()

```
void done ( )  [protected]
```

This method is executed at the end of the worker thread in the Event Dispatch Thread.

Definition at line 386 of file Chemotaxis.java.

6.2.2.14 drawResults()

```
void drawResults (
            Trial trial )  [private]
```

This method draws a chemotactic cone and a rose diagram for a single trial analysis.

Parameters

trial	

Definition at line 407 of file Chemotaxis.java.

6.2.2.15 findTrial()

```
Trial findTrial (
            String id,
            Map< String, Trial > trials )  [private]
```

This method search in the given set of trials for the one with the given ID.

Parameters

id	• Identifier of the trial to find.
trials	• Set of trials

Returns

the trial found or null otherwise

Definition at line 429 of file Chemotaxis.java.

6.2.2.16 getControlTrials()

```
Map<String, Trial> getControlTrials (
            String folder )  [private]
```

Definition at line 438 of file Chemotaxis.java.

6.2.2.17 getListOfAngles()

```
List<Double> getListOfAngles (
            SerializableList theTracks )  [private]
```

This method calculates all instantaneous displacements for a given set of trajectories.

Parameters

theTracks	• Array with all trajectories.

Returns

List of angles

Definition at line 474 of file Chemotaxis.java.

6.2.2.18 getOddsValues()

```
double [] getOddsValues (
            SerializableList tracks )  [private]
```

This method counts, for the given set of trajectories, how many angles go in the gradient direction and how many in the opposite (or other) direction, depending on the value of the parameter "Compare Opposite Directions" set by the user on the Settings Window. The method stops when all the trajectories have been analysed or the maximum number of angles to analyse is reached.

Parameters

tracks	• All the trajectories to analyse

Returns

An array containing the total count of angles ([0] - upgradient: [1] - other directions).

Definition at line 505 of file Chemotaxis.java.

6.2.2.19 getTestFolders()

```
List<String> getTestFolders (
            String folder )  [private]
```

For a given folder, this method returns a list of all subfolders contained in it, that are not named as "control".

Parameters

folder	

Returns

List of subfolders

Definition at line 531 of file Chemotaxis.java.

6.2.2.20 getTrials()

```
Map<String, Trial> getTrials (
            List< String > filenames )  [private]
```

This method returns all trials extracted from the given set of AVI files.

Parameters

filenames	List of avi filenames to be analysed.

Returns

All extracted trials

Definition at line 551 of file Chemotaxis.java.

6.2.2.21 indexesAnalysis()

```
ResultsTable indexesAnalysis (
            Map< String, Trial > controls,
            Map< String, Trial > tests )  [private]
```

This method calculates Ch-Index and SL-index for the given set of trials

Parameters

controls	• control trials
tests	• tests trials

Returns

ResultsTable with all indexes

Definition at line 574 of file Chemotaxis.java.

6.2.2.22 mergeTracks()

```
SerializableList mergeTracks (
            Map< String, Trial > trials )  [private]
```

This method join all tracks of the given set of trials, into one single array.

Parameters

trials	• set of trials

Returns

List with all tracks

Definition at line 602 of file Chemotaxis.java.

6.2.2.23 minSampleSize()

```
int minSampleSize (
            Map< String, Trial > trials )  [private]
```

This method looks for which trial has the minimum number of instantaneous displacements and return this number.

Parameters

trials	• set of trials

Returns

the value of the minimum number of instantaneous displacements for one trial in the given set of trials.

Definition at line 621 of file Chemotaxis.java.

6.2.2.24 or()

```
double or (
            Trial control,
            Trial test )  [private]
```

This method calculates odds ratio for a given pair control-test

Parameters

control	• control trial
test	• test trial to be analysed

Returns

Odds Ratio

Definition at line 643 of file Chemotaxis.java.

6.2.2.25 orThreshold()

```
double orThreshold (
            Map< String, Trial > controls )  [private]
```

This method calculates the control threshold used for bootstrapping analysis.

Parameters

controls	• Set of control trials

Returns

Threshold for bootstrapping analysis

Definition at line 666 of file Chemotaxis.java.

6.2.2.26 relativeAngle()

```
double relativeAngle (
            Spermatozoon previous,
            Spermatozoon next )  [private]
```

This method calculates the relative angle between the given displacement made by a cell, and the gradient direction.

Parameters

previous	• previous coordinates of the cell before displacement
next	• next coordinates of the cell after displacement

Returns

relative angle in the interval [-PI,PI]

Definition at line 701 of file Chemotaxis.java.

6.2.2.27 selectAnalysis()

```
void selectAnalysis ( )
```

This method opens a set of dialogs to ask the user which analysis has to be carried on.

Definition at line 724 of file Chemotaxis.java.

6.2.2.28 setBootstrappingResults()

```
void setBootstrappingResults (
            ResultsTable rt,
            double or,
            double th,
            Trial trial )  [private]
```

This method adds to the given results table, a new row with the given parameters.

Parameters

rt	• ResultsTable to be appended.
or	• OddsRatio.
th	• Threshold used to calculate the given oddsRatio.
trial	• Trial analysed.

Definition at line 781 of file Chemotaxis.java.

6.2.2.29 setIndexesResults()

```
void setIndexesResults (
            ResultsTable rt,
            Trial trial,
            float chIdx,
            float slIdx )  [private]
```

This method adds to the given results table, a new row with the given parameters.

Parameters

rt	• ResultsTable to be appended.
trial	• Trial analysed.
chIdx	• Ch-Index of the given trial.
slIdx	• SL-Index of the given trial.

Definition at line 818 of file Chemotaxis.java.

6.2.3 Member Data Documentation

6.2.3.1 analysis

```
TypeOfAnalysis analysis = TypeOfAnalysis.NONE  [private]
```

Definition at line 54 of file Chemotaxis.java.

The documentation for this class was generated from the following file:

- Chemotaxis.java

6.3 ComputerVision Class Reference

Public Member Functions

- double autoThresholdImagePlus (ImagePlus imp)
- double autoThresholdImagePlus (ImagePlus imp, String thresholdMethod)
- void convertToGrayscale (ImagePlus imp)
- void convertToRGB (ImagePlus imp)
- ImagePlus getBlueChannel (ImagePlus impColor)
- ImagePlus getGreenChannel (ImagePlus impColor)
- float getMeanGrayValue (Spermatozoon part, ImagePlus impGray, ImagePlus impTh)
- ImagePlus getRedChannel (ImagePlus impColor)
- void outlineThresholdImage (ImagePlus imp)
- void thresholdImagePlus (ImagePlus imp, double lowerThreshold)
- void thresholdImageProcessor (ImageProcessor ip, double lowerThreshold, double upperThreshold)
- void thresholdStack (ImagePlus imp)

6.3.1 Detailed Description

Author

Carlos Alquezar

Definition at line 37 of file ComputerVision.java.

6.3.2 Member Function Documentation

6.3.2.1 autoThresholdImagePlus() [1/2]

```
double autoThresholdImagePlus (
            ImagePlus imp )
```

Parameters

imp	

Returns

Definition at line 45 of file ComputerVision.java.

6.3.2.2 autoThresholdImagePlus() [2/2]

```
double autoThresholdImagePlus (
            ImagePlus imp,
            String thresholdMethod )
```

Parameters

imp	
thresholdMethod	

Returns

Definition at line 56 of file ComputerVision.java.

6.3.2.3 convertToGrayscale()

```
void convertToGrayscale (
            ImagePlus imp )
```

Parameters

imp	ImagePlus

This functions converts imp to grayscale.

Definition at line 79 of file ComputerVision.java.

6.3.2.4 convertToRGB()

```
void convertToRGB (
            ImagePlus imp )
```

Parameters

imp	ImagePlus

This functions converts imp to grayscale.

Definition at line 91 of file ComputerVision.java.

6.3.2.5 getBlueChannel()

```
ImagePlus getBlueChannel (
            ImagePlus impColor )
```

Parameters

impColor	

Returns

Definition at line 103 of file ComputerVision.java.

6.3.2.6 getGreenChannel()

```
ImagePlus getGreenChannel (
            ImagePlus impColor )
```

Parameters

impColor	

Returns

Definition at line 113 of file ComputerVision.java.

6.3.2.7 getMeanGrayValue()

```
float getMeanGrayValue (
            Spermatozoon part,
            ImagePlus impGray,
            ImagePlus impTh )
```

Parameters

part	
impGray	
impTh	

Returns

Definition at line 125 of file ComputerVision.java.

6.3.2.8 getRedChannel()

```
ImagePlus getRedChannel (
            ImagePlus impColor )
```

Parameters

impColor	

Returns

Definition at line 153 of file ComputerVision.java.

6.3.2.9 outlineThresholdImage()

```
void outlineThresholdImage (
            ImagePlus imp )
```

Definition at line 163 of file ComputerVision.java.

6.3.2.10 thresholdImagePlus()

```
void thresholdImagePlus (
            ImagePlus imp,
            double lowerThreshold )
```

Parameters

imp	
lowerThreshold	

Definition at line 175 of file ComputerVision.java.

6.3.2.11 thresholdImageProcessor()

```
void thresholdImageProcessor (
            ImageProcessor ip,
            double lowerThreshold,
            double upperThreshold )
```

Parameters

ip	
lowerThreshold	
upperThreshold	

Definition at line 189 of file ComputerVision.java.

6.3.2.12 thresholdStack()

```
void thresholdStack (
            ImagePlus imp )
```

Parameters

ImagePlus	This function makes binary 'imp' applying an statistical threshold

Definition at line 206 of file ComputerVision.java.

The documentation for this class was generated from the following file:

- ComputerVision.java

6.4 FileManager Class Reference

Public Member Functions

- FileManager ()
- ImagePlus getAVI (String path)
- String getFilename (String path)
- String [ ] getContent (String path)
- List< String > getFiles (String path)
- List< String > getSubfolders (String path)
- String getParentDirectory (String path)
- boolean isAVI (String filename)
- List< ImagePlus > loadImageDirectory ()
- List< ImagePlus > loadImageDirectory (String dir)
- List< ImagePlus > loadImageFile ()
- String removeExtension (String filename)
- String selectFile ()
- String selectFolder ()

6.4.1 Detailed Description

Author

Carlos Alquezar

Definition at line 37 of file FileManager.java.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 FileManager()

```
FileManager ( )
```

Definition at line 40 of file FileManager.java.

6.4.3 Member Function Documentation

6.4.3.1 getAVI()

```
ImagePlus getAVI (
            String path )
```

Parameters

path	

Returns

Definition at line 47 of file FileManager.java.

6.4.3.2 getContent()

```
String [] getContent (
            String path )
```

Parameters

dir	

Returns

Definition at line 66 of file FileManager.java.

6.4.3.3 getFilename()

```
String getFilename (
            String path )
```

Parameters

path	

Returns

Definition at line 57 of file FileManager.java.

6.4.3.4 getFiles()

```
List<String> getFiles (
            String path )
```

Definition at line 79 of file FileManager.java.

6.4.3.5 getParentDirectory()

```
String getParentDirectory (
            String path )
```

Parameters

path	

Returns

Definition at line 105 of file FileManager.java.

6.4.3.6 getSubfolders()

```
List<String> getSubfolders (
            String path )
```

Definition at line 90 of file FileManager.java.

6.4.3.7 isAVI()

```
boolean isAVI (
            String filename )
```

Parameters

filename	

Returns

Definition at line 114 of file FileManager.java.

6.4.3.8 loadImageDirectory() [1/2]

```
List<ImagePlus> loadImageDirectory ( )
```

Returns

Definition at line 125 of file FileManager.java.

6.4.3.9 loadImageDirectory() [2/2]

```
List<ImagePlus> loadImageDirectory (
            String dir )
```

Returns

Definition at line 134 of file FileManager.java.

6.4.3.10 loadImageFile()

```
List<ImagePlus> loadImageFile ( )
```

Returns

an array with only one ImagePlus, compatible with the input specification of other functions

Definition at line 168 of file FileManager.java.

6.4.3.11 removeExtension()

```
String removeExtension (
            String filename )
```

Definition at line 189 of file FileManager.java.

6.4.3.12 selectFile()

```
String selectFile ( )
```

Returns

Definition at line 212 of file FileManager.java.

6.4.3.13 selectFolder()

```
String selectFolder ( )
```

Returns

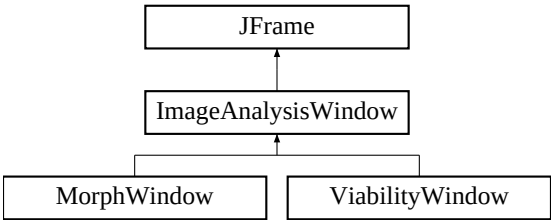
Definition at line 227 of file FileManager.java.

The documentation for this class was generated from the following file:

- FileManager.java

6.5 ImageAnalysisWindow Class Reference

Inheritance diagram for ImageAnalysisWindow:



Classes

- enum TypeOfAnalysis

Public Member Functions

- ImageAnalysisWindow ()
- void deselectAll ()
- void idenfitySperm ()
- void initImage ()
- void reset ()
- int run ()
- void selectAll ()
- void selectAll (List< Spermatozoon > sperm)
- int selectAnalysis ()
- void setChangeListener (ChangeListener ch, JSlider sld)
- void setImage ()
- void setImage (int index)
- void setImages (List< ImagePlus > i)
- void setMouseListener (MouseListener ml)
- void setRawImage ()
- void setResizeFactor ()
- void showWindow ()
- void thresholdImagePlus (ImagePlus imp)

Protected Member Functions

- void drawImage ()
- void nextAction ()
- void previousAction ()
- void processImage (boolean eventType)
- void genericRadioButtonsAction ()

Protected Attributes

- ImagePlus impDraw = null
- ImagePlus impGray = null
- ImagePlus impOrig = null
- ImagePlus impOutline = null
- ImagePlus impTh = null
- JSlider sldThreshold
- JSlider sldRedThreshold
- JSlider sldGreenThreshold
- JSlider sldBlueThreshold
- JRadioButton btnOtsu
- JRadioButton btnMinimum
- ButtonGroup btnGroup
- JButton prevBtn
- JButton nextBtn
- List< Spermatozoon > spermatozoa = new ArrayList<Spermatozoon>()
- double threshold = -1.0
- double redThreshold = -1.0
- double greenThreshold = -1.0
- double blueThreshold = -1.0
- String thresholdMethod = "Otsu"
- double xFactor
- double yFactor

Private Member Functions

- int analyseDirectory ()
- int analyseFile ()
- void configureSliderBar (JSlider sld)
- void setSlidersAutoThreshold ()

Private Attributes

- TypeOfAnalysis analysis = TypeOfAnalysis.NONE
- List< ImagePlus > images
- int imgIndex
- JLabel imgLabel
- double resizeFactor
- JLabel title

6.5.1 Detailed Description

This class implements all the functions related to viability analysis.

Author

Carlos Alquezar

Definition at line 57 of file ImageAnalysisWindow.java.

6.5.2 Constructor & Destructor Documentation

6.5.2.1 ImageAnalysisWindow()

```
ImageAnalysisWindow ( )
```

Definition at line 104 of file ImageAnalysisWindow.java.

6.5.3 Member Function Documentation

6.5.3.1 analyseDirectory()

```
int analyseDirectory ( )  [private]
```

Definition at line 130 of file ImageAnalysisWindow.java.

6.5.3.2 analyseFile()

```
int analyseFile ( )  [private]
```

Definition at line 142 of file ImageAnalysisWindow.java.

6.5.3.3 configureSliderBar()

```
void configureSliderBar (
            JSlider sld )  [private]
```

Definition at line 339 of file ImageAnalysisWindow.java.

6.5.3.4 deselectAll()

```
void deselectAll ( )
```

This method deselect all spermatozoa.

Definition at line 157 of file ImageAnalysisWindow.java.

6.5.3.5 drawImage()

```
void drawImage ( )  [protected]
```

Definition at line 163 of file ImageAnalysisWindow.java.

6.5.3.6 genericRadioButtonsAction()

```
void genericRadioButtonsAction ( )  [protected]
```

Definition at line 337 of file ImageAnalysisWindow.java.

6.5.3.7 idenfitySperm()

```
void idenfitySperm ( )
```

This method set a unique identifier for each spermatozoon in the spermatozoa list

Definition at line 168 of file ImageAnalysisWindow.java.

6.5.3.8 initImage()

```
void initImage ( )
```

This method sets the initial image to be showed.

Definition at line 180 of file ImageAnalysisWindow.java.

6.5.3.9 nextAction()

```
void nextAction ( )  [protected]
```

Definition at line 186 of file ImageAnalysisWindow.java.

6.5.3.10 previousAction()

```
void previousAction ( )  [protected]
```

Definition at line 189 of file ImageAnalysisWindow.java.

6.5.3.11 processImage()

```
void processImage (
            boolean eventType )  [protected]
```

Definition at line 192 of file ImageAnalysisWindow.java.

6.5.3.12 reset()

```
void reset ( )
```

Definition at line 194 of file ImageAnalysisWindow.java.

6.5.3.13 run()

```
int run ( )
```

Definition at line 209 of file ImageAnalysisWindow.java.

6.5.3.14 selectAll() [1/2]

```
void selectAll ( )
```

Definition at line 227 of file ImageAnalysisWindow.java.

6.5.3.15 selectAll() [2/2]

```
void selectAll (
            List< Spermatozoon > sperm )
```

This method deselect all spermatozoa.

Definition at line 234 of file ImageAnalysisWindow.java.

6.5.3.16 selectAnalysis()

```
int selectAnalysis ( )
```

This method opens a set of dialogs to ask the user which analysis has to be carried on.

Definition at line 245 of file ImageAnalysisWindow.java.

6.5.3.17 setChangeListener()

```
void setChangeListener (
            ChangeListener ch,
            JSlider sld )
```

Definition at line 266 of file ImageAnalysisWindow.java.

6.5.3.18 setImage() [1/2]

```
void setImage ( )
```

This method sets the first image on the list and show it on screen.

Definition at line 274 of file ImageAnalysisWindow.java.

6.5.3.19 setImage() [2/2]

```
void setImage (
            int index )
```

This method sets the image at corresponding index on the list and show it on screen.

Parameters

index	• index of the image on the list of loaded images.

Definition at line 296 of file ImageAnalysisWindow.java.

6.5.3.20 setImages()

```
void setImages (
            List< ImagePlus > i )
```

This method sets the images attribute with the given list of ImagePlus.

Parameters

i	

Definition at line 312 of file ImageAnalysisWindow.java.

6.5.3.21 setMouseListener()

```
void setMouseListener (
            MouseListener ml )
```

Definition at line 316 of file ImageAnalysisWindow.java.

6.5.3.22 setRawImage()

```
void setRawImage ( )
```

Definition at line 320 of file ImageAnalysisWindow.java.

6.5.3.23 setResizeFactor()

```
void setResizeFactor ( )
```

This method calculates the resize factor due to the original image size and the showed image size.

Definition at line 328 of file ImageAnalysisWindow.java.

6.5.3.24 setSlidersAutoThreshold()

```
void setSlidersAutoThreshold ( )  [private]
```

Definition at line 477 of file ImageAnalysisWindow.java.

6.5.3.25 showWindow()

```
void showWindow ( )
```

This method creates and shows the window.

Definition at line 351 of file ImageAnalysisWindow.java.

6.5.3.26 thresholdImagePlus()

```
void thresholdImagePlus (
            ImagePlus imp )
```

This method choose between autoThreshold or apply a particular threshold to the given ImagePlus depending if this value has been set before or not.

Parameters

imp	ImagePlus to be thresholded.

Definition at line 494 of file ImageAnalysisWindow.java.

6.5.4 Member Data Documentation

6.5.4.1 analysis

```
TypeOfAnalysis analysis = TypeOfAnalysis.NONE  [private]
```

Definition at line 63 of file ImageAnalysisWindow.java.

6.5.4.2 blueThreshold

```
double blueThreshold = -1.0  [protected]
```

Definition at line 95 of file ImageAnalysisWindow.java.

6.5.4.3 btnGroup

```
ButtonGroup btnGroup  [protected]
```

Definition at line 87 of file ImageAnalysisWindow.java.

6.5.4.4 btnMinimum

```
JRadioButton btnMinimum  [protected]
```

Definition at line 86 of file ImageAnalysisWindow.java.

6.5.4.5 btnOtsu

```
JRadioButton btnOtsu  [protected]
```

Definition at line 85 of file ImageAnalysisWindow.java.

6.5.4.6 greenThreshold

```
double greenThreshold = -1.0  [protected]
```

Definition at line 94 of file ImageAnalysisWindow.java.

6.5.4.7 images

```
List<ImagePlus> images  [private]
```

Definition at line 66 of file ImageAnalysisWindow.java.

6.5.4.8 imgIndex

```
int imgIndex  [private]
```

Definition at line 67 of file ImageAnalysisWindow.java.

6.5.4.9 imgLabel

```
JLabel imgLabel  [private]
```

Definition at line 68 of file ImageAnalysisWindow.java.

6.5.4.10 impDraw

```
ImagePlus impDraw = null  [protected]
```

ImagePlus used to draw over them

Definition at line 70 of file ImageAnalysisWindow.java.

6.5.4.11 impGray

```
ImagePlus impGray = null  [protected]
```

ImagePlus used to calculate mean gray values

Definition at line 72 of file ImageAnalysisWindow.java.

6.5.4.12 impOrig

```
ImagePlus impOrig = null  [protected]
```

ImagePlus used to store the original images

Definition at line 74 of file ImageAnalysisWindow.java.

6.5.4.13 impOutline

```
ImagePlus impOutline = null  [protected]
```

ImagePlus used to store outlines

Definition at line 76 of file ImageAnalysisWindow.java.

6.5.4.14 impTh

```
ImagePlus impTh = null  [protected]
```

ImagePlus used to identify spermatozoa

Definition at line 78 of file ImageAnalysisWindow.java.

6.5.4.15 nextBtn

```
JButton nextBtn  [protected]
```

Definition at line 89 of file ImageAnalysisWindow.java.

6.5.4.16 prevBtn

```
JButton prevBtn  [protected]
```

Definition at line 88 of file ImageAnalysisWindow.java.

6.5.4.17 redThreshold

```
double redThreshold = -1.0  [protected]
```

Definition at line 93 of file ImageAnalysisWindow.java.

6.5.4.18 resizeFactor

```
double resizeFactor  [private]
```

Definition at line 80 of file ImageAnalysisWindow.java.

6.5.4.19 sldBlueThreshold

```
JSlider sldBlueThreshold  [protected]
```

Definition at line 84 of file ImageAnalysisWindow.java.

6.5.4.20 sldGreenThreshold

```
JSlider sldGreenThreshold  [protected]
```

Definition at line 83 of file ImageAnalysisWindow.java.

6.5.4.21 sldRedThreshold

```
JSlider sldRedThreshold  [protected]
```

Definition at line 82 of file ImageAnalysisWindow.java.

6.5.4.22 sldThreshold

```
JSlider sldThreshold  [protected]
```

Definition at line 81 of file ImageAnalysisWindow.java.

6.5.4.23 spermatozoa

```
List<Spermatozoon> spermatozoa = new ArrayList<Spermatozoon>()  [protected]
```

Definition at line 91 of file ImageAnalysisWindow.java.

6.5.4.24 threshold

```
double threshold = -1.0  [protected]
```

Definition at line 92 of file ImageAnalysisWindow.java.

6.5.4.25 thresholdMethod

```
String thresholdMethod = "Otsu"  [protected]
```

Definition at line 96 of file ImageAnalysisWindow.java.

6.5.4.26 title

```
JLabel title  [private]
```

Definition at line 97 of file ImageAnalysisWindow.java.

6.5.4.27 xFactor

```
double xFactor  [protected]
```

Definition at line 99 of file ImageAnalysisWindow.java.

6.5.4.28 yFactor

```
double yFactor  [protected]
```

Definition at line 102 of file ImageAnalysisWindow.java.

The documentation for this class was generated from the following file:

- ImageAnalysisWindow.java

6.6 Kinematics Class Reference

Public Member Functions

- float [ ] alh (List track, List avgTrack)
- float bcf (List track, List avgTrack)
- String getVelocityTrackType (List track)
- float mad (List track)
- boolean motilityTest (List track)
- int [ ] motilityTest (SerializableList theTracks)
- float vcl (List track)
- float vsl (List track)

6.6.1 Detailed Description

Author

Carlos Alquezar

Definition at line 33 of file Kinematics.java.

6.6.2 Member Function Documentation

6.6.2.1 alh()

```
float [] alh (
            List track,
            List avgTrack )
```

Parameters

track	• a track
avgTrack	•

Returns

ALH (mean and max) (um)

Definition at line 42 of file Kinematics.java.

6.6.2.2 bcf()

```
float bcf (
            List track,
            List avgTrack )
```

Parameters

track	• a track
avgTrack	•

Returns

BCF (Hz)

Parameters

track	
avgTrack	

Returns

Definition at line 103 of file Kinematics.java.

6.6.2.3 getVelocityTrackType()

```
String getVelocityTrackType (
            List track )
```

Parameters

track	

Returns

Definition at line 142 of file Kinematics.java.

6.6.2.4 mad()

```
float mad (
            List track )
```

Parameters

track	• a track

Returns

MAD - (degrees)

Definition at line 161 of file Kinematics.java.

6.6.2.5 motilityTest() [1/2]

```
boolean motilityTest (
            List track )
```

Parameters

track	

Returns

Definition at line 185 of file Kinematics.java.

6.6.2.6 motilityTest() [2/2]

```
int [] motilityTest (
            SerializableList theTracks )
```

Parameters

theTracks	

Returns

Definition at line 208 of file Kinematics.java.

6.6.2.7 vcl()

```
float vcl (
            List track )
```

Parameters

track	• a track

Returns

VCL (um/second)

Definition at line 230 of file Kinematics.java.

6.6.2.8 vsl()

```
float vsl (
            List track )
```

Parameters

track	• a track

Returns

VSL (um/second)

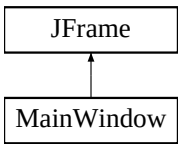
Definition at line 255 of file Kinematics.java.

The documentation for this class was generated from the following file:

- Kinematics.java

6.7 MainWindow Class Reference

Inheritance diagram for MainWindow:



Public Member Functions

- MainWindow (String title) throws HeadlessException

  Constructor. The main graphical user interface is created.

Private Member Functions

- void addButton (final String label, int gridx, int gridy, Color background, String iconPath, JPanel panel)

  This method add to the given JPanel a button with the specified parameters.
- void createGUI ()

  This method creates the main user interface.
- void simulate ()

  Shows a Generic Dialog to ask user which simulation parameters have to be used for the simulation.

Private Attributes

- MainWindow mw

  Self reference used in action listeners to show and hide main window.

Static Private Attributes

- static final long serialVersionUID = 1L

6.7.1 Detailed Description

This window shows all functional modules available.

Author

Carlos Alquezar

Definition at line 48 of file MainWindow.java.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 MainWindow()

```
MainWindow (
            String title ) throws HeadlessException
```

Constructor. The main graphical user interface is created.

Parameters

title	- String that is used as window's title

Definition at line 61 of file MainWindow.java.

6.7.3 Member Function Documentation

6.7.3.1 addButton()

```
void addButton (
            final String label,
            int gridx,
            int gridy,
            Color background,
            String iconPath,
            JPanel panel )  [private]
```

This method add to the given JPanel a button with the specified parameters.

Parameters

label	- String that is shown as button's label
gridx	- relative layout's x location
gridy	- relative layout's y location
background	- Background color
iconPath	- Path relative to the icon's image
panel	- panel where the button is going to be added

Definition at line 82 of file MainWindow.java.

6.7.3.2 createGUI()

```
void createGUI ( )  [private]
```

This method creates the main user interface.

Definition at line 174 of file MainWindow.java.

6.7.3.3 simulate()

```
void simulate ( )  [private]
```

Shows a Generic Dialog to ask user which simulation parameters have to be used for the simulation.

Definition at line 190 of file MainWindow.java.

6.7.4 Member Data Documentation

6.7.4.1 mw

```
MainWindow mw  [private]
```

Self reference used in action listeners to show and hide main window.

Definition at line 55 of file MainWindow.java.

6.7.4.2 serialVersionUID

```
final long serialVersionUID = 1L  [static], [private]
```

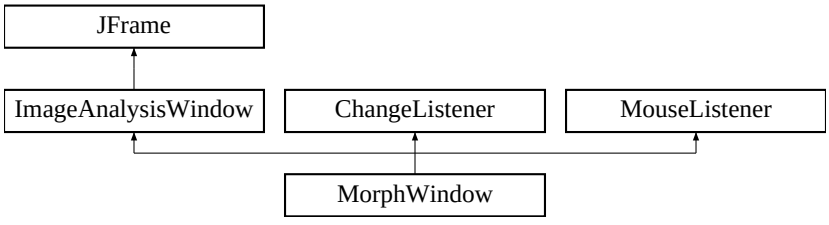
Definition at line 53 of file MainWindow.java.

The documentation for this class was generated from the following file:

- MainWindow.java

6.8 MorphWindow Class Reference

Inheritance diagram for MorphWindow:



Public Member Functions

- MorphWindow () throws HeadlessException
- void checkSelection (int x, int y)
- void close ()
- boolean isClickInside (Spermatozoon sperm, Point click)
- void mouseClicked (MouseEvent e)
- void mouseEntered (MouseEvent e)
- void mouseExited (MouseEvent e)
- void mousePressed (MouseEvent e)
- void mouseReleased (MouseEvent e)
- void processImage (boolean eventType)
- void stateChanged (ChangeEvent inEvent)

Private Member Functions

- void doMouseRefresh ()
- void doSliderRefresh ()
- void generateResults (Spermatozoon spermatozoon)

Private Attributes

- boolean isThresholding = false
- ResultsTable morphometrics = new ResultsTable()

Additional Inherited Members

6.8.1 Detailed Description

This class implements all the functions related to morphometry analysis.

Author

Carlos Alquezar

Definition at line 46 of file MorphWindow.java.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 MorphWindow()

```
MorphWindow ( ) throws HeadlessException
```

Constructor. The main graphical user interface is created.

Definition at line 56 of file MorphWindow.java.

6.8.3 Member Function Documentation

6.8.3.1 checkSelection()

```
void checkSelection (
            int x,
            int y )
```

This method checks if a click has been done over a cell. In that case, the method select/deselect the cell and add the morphometrics to resultsTable if it has been selected.

Parameters

x	
y	

Definition at line 72 of file MorphWindow.java.

6.8.3.2 close()

```
void close ( )
```

This method closes all ImagePlus.

Definition at line 91 of file MorphWindow.java.

6.8.3.3 doMouseRefresh()

```
void doMouseRefresh ( )  [private]
```

This method refreshes the showed image after a mouse click event

Definition at line 102 of file MorphWindow.java.

6.8.3.4 doSliderRefresh()

```
void doSliderRefresh ( )  [private]
```

This method refreshes the showed image after changing the threshold with the sliderbar

Definition at line 124 of file MorphWindow.java.

6.8.3.5 generateResults()

```
void generateResults (
            Spermatozoon spermatozoon )  [private]
```

This method adds the morphometric values of the given spermatozoon to the results table

Parameters

spermatozoon	

Definition at line 145 of file MorphWindow.java.

6.8.3.6 isClickInside()

```
boolean isClickInside (
```

```
            Spermatozoon sperm,
            Point click )
```

This method returns true if the given point is inside the boundaries of the given spermatozoon

Parameters

sperm	• Spermatozoon
click	• Point

Returns

True if the point is inside the boundaries of the spermatozoon. Otherwise, it returns false

Definition at line 194 of file MorphWindow.java.

6.8.3.7 mouseClicked()

```
void mouseClicked (
            MouseEvent e )
```

This method manage a mouse click event.

Definition at line 215 of file MorphWindow.java.

6.8.3.8 mouseEntered()

```
void mouseEntered (
            MouseEvent e )
```

Definition at line 225 of file MorphWindow.java.

6.8.3.9 mouseExited()

```
void mouseExited (
            MouseEvent e )
```

Definition at line 226 of file MorphWindow.java.

6.8.3.10 mousePressed()

```
void mousePressed (
            MouseEvent e )
```

Definition at line 227 of file MorphWindow.java.

6.8.3.11 mouseReleased()

```
void mouseReleased (
            MouseEvent e )
```

Definition at line 228 of file MorphWindow.java.

6.8.3.12 processImage()

```
void processImage (
            boolean eventType )
```

This method updates the showed image depending of the type of event ocurred.

Parameters

eventType	This parameter is used to differentiate between a slider event (true) or a click event (false)

Definition at line 239 of file MorphWindow.java.

6.8.3.13 stateChanged()

```
void stateChanged (
            ChangeEvent inEvent )
```

Listen events from slider

Definition at line 263 of file MorphWindow.java.

6.8.4 Member Data Documentation

6.8.4.1 isThresholding

```
boolean isThresholding = false  [private]
```

Definition at line 49 of file MorphWindow.java.

6.8.4.2 morphometrics

```
ResultsTable morphometrics = new ResultsTable()  [private]
```

Resultstable used to show results

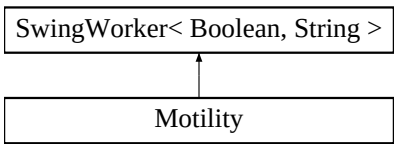
Definition at line 51 of file MorphWindow.java.

The documentation for this class was generated from the following file:

- MorphWindow.java

6.9 Motility Class Reference

Inheritance diagram for Motility:



Classes

- enum TypeOfAnalysis

Public Member Functions

- Motility ()
- void selectAnalysis ()

Protected Member Functions

- Boolean doInBackground () throws Exception

Private Member Functions

- void analyseDirectories ()
- void analyseDirectory ()
- void analyseFile ()
- void calculateAverageMotility (ResultsTable rt, Trial trial)
- void calculateMotility (ResultsTable rt, Trial trial)
- void calculateTotalMotility (ResultsTable rt, String filename)
- Map< String, Trial > getTrials (List< String > filenames)
- void resetParams ()

Private Attributes

- TypeOfAnalysis analysis = TypeOfAnalysis.NONE
- float countProgressiveSperm = 0
- float total_alhMax = 0
- float total_alhMean = 0
- float total_bcf = 0
- float total_dance = 0
- float total_lin = 0
- float total_mad = 0
- float total_motile = 0
- float total_nonMotile = 0
- float total_sperm = 0
- float total_str = 0
- float total_vap = 0
- float total_vcl = 0
- float total_vsl = 0
- float total_wob = 0

6.9.1 Detailed Description

This class implements all the functions related to motility analysis.

Author

Carlos Alquezar

Definition at line 46 of file Motility.java.

6.9.2 Constructor & Destructor Documentation

6.9.2.1 Motility()

```
Motility ( )
```

Definition at line 69 of file Motility.java.

6.9.3 Member Function Documentation

6.9.3.1 analyseDirectories()

```
void analyseDirectories ( )  [private]
```

This method asks user for the directory that contains all subfolders that are going to be analysed. For each subfolder, the average motility parameters will be calculated. The method finish showing a ResultsTable with the motility information.

Definition at line 78 of file Motility.java.

6.9.3.2 analyseDirectory()

```
void analyseDirectory ( )  [private]
```

This method asks user for the directory that contains all AVI files that are going to be analysed. For each file, the individual and average motility parameters will be calculated. The method finish showing the corresponding results tables with the motility information.

Definition at line 107 of file Motility.java.

6.9.3.3 analyseFile()

```
void analyseFile ( )  [private]
```

This method asks user for the file that is going to be analysed, extract the corresponding trial and show results.

Definition at line 129 of file Motility.java.

6.9.3.4 calculateAverageMotility()

```
void calculateAverageMotility (
            ResultsTable rt,
            Trial trial )  [private]
```

This method calculates the average motility values for the given trial.

Parameters

rt	• ResultsTable where the motility information will be added.
trial	• Trial with all trajectories that will be analysed.

Definition at line 161 of file Motility.java.

6.9.3.5 calculateMotility()

```
void calculateMotility (
            ResultsTable rt,
            Trial trial )  [private]
```

This method calculates the individual motility values for the given trial.

Parameters

rt	• ResultsTable where the motility information will be added.
trial	• Trial with all trajectories that will be analysed.

Definition at line 223 of file Motility.java.

6.9.3.6 calculateTotalMotility()

```
void calculateTotalMotility (
            ResultsTable rt,
            String filename )  [private]
```

This method calculates the total average motility values for a folder.

Parameters

rt	• ResultsTable where the motility information will be added.
filename	• the folder name. This information will be added to the results table.

Definition at line 307 of file Motility.java.

6.9.3.7 doInBackground()

```
Boolean doInBackground ( ) throws Exception  [protected]
```

This method is inherit from SwingWorker class and it is the starting point after the execute() method is called.

Definition at line 354 of file Motility.java.

6.9.3.8 getTrials()

```
Map<String, Trial> getTrials (
            List< String > filenames )  [private]
```

This method returns all trials extracted from the given set of AVI files.

Parameters

filenames	List of avi filenames to be analysed.

Returns

All extracted trials

Definition at line 376 of file Motility.java.

6.9.3.9 resetParams()

```
void resetParams ( )  [private]
```

This method resets all motility parameters of the motility analysis

Definition at line 393 of file Motility.java.

6.9.3.10 selectAnalysis()

```
void selectAnalysis ( )
```

This method opens a set of dialogs to ask the user which analysis has to be carried on.

Definition at line 415 of file Motility.java.

6.9.4 Member Data Documentation

6.9.4.1 analysis

```
TypeOfAnalysis analysis = TypeOfAnalysis.NONE  [private]
```

Definition at line 52 of file Motility.java.

6.9.4.2 countProgressiveSperm

```
float countProgressiveSperm = 0  [private]
```

Definition at line 53 of file Motility.java.

6.9.4.3 total_alhMax

```
float total_alhMax = 0  [private]
```

Definition at line 54 of file Motility.java.

6.9.4.4 total_alhMean

```
float total_alhMean = 0  [private]
```

Definition at line 55 of file Motility.java.

6.9.4.5 total_bcf

```
float total_bcf = 0  [private]
```

Definition at line 56 of file Motility.java.

6.9.4.6 total_dance

```
float total_dance = 0  [private]
```

Definition at line 57 of file Motility.java.

6.9.4.7 total_lin

```
float total_lin = 0  [private]
```

Definition at line 58 of file Motility.java.

6.9.4.8 total_mad

```
float total_mad = 0  [private]
```

Definition at line 59 of file Motility.java.

6.9.4.9 total_motile

```
float total_motile = 0  [private]
```

Definition at line 60 of file Motility.java.

6.9.4.10 total_nonMotile

```
float total_nonMotile = 0  [private]
```

Definition at line 61 of file Motility.java.

6.9.4.11 total_sperm

```
float total_sperm = 0  [private]
```

Definition at line 62 of file Motility.java.

6.9.4.12 total_str

```
float total_str = 0  [private]
```

Definition at line 63 of file Motility.java.

6.9.4.13 total_vap

```
float total_vap = 0  [private]
```

Definition at line 64 of file Motility.java.

6.9.4.14 total_vcl

```
float total_vcl = 0  [private]
```

Definition at line 65 of file Motility.java.

6.9.4.15 total_vsl

```
float total_vsl = 0  [private]
```

Definition at line 66 of file Motility.java.

6.9.4.16 total_wob

```
float total_wob = 0  [private]
```

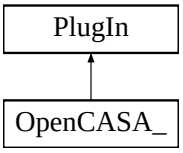
Definition at line 67 of file Motility.java.

The documentation for this class was generated from the following file:

- Motility.java

6.10 OpenCASA_ Class Reference

Inheritance diagram for OpenCASA_:



Public Member Functions

- void run (String arg)

Static Public Member Functions

- static void main (String[] args) throws ClassNotFoundException, InstantiationException, IllegalAccess↩ Exception, UnsupportedLookAndFeelException

6.10.1 Detailed Description

OpenCASA - OpenSource software for Computer Assisted Sperm Analysis

Author

Carlos Alquezar

Definition at line 32 of file OpenCASA_.java.

6.10.2 Member Function Documentation

6.10.2.1 main()

```
static void main (
            String [] args ) throws ClassNotFoundException, InstantiationException, Illegal↩
AccessException, UnsupportedLookAndFeelException  [static]
```

Main method

Definition at line 37 of file OpenCASA_.java.

6.10.2.2 run()

```
void run (
            String arg )
```

This method overrides the superclass run's method. Start point of the plugin.

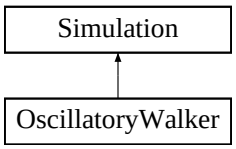
Definition at line 50 of file OpenCASA_.java.

The documentation for this class was generated from the following file:

- OpenCASA_.java

6.11 OscillatoryWalker Class Reference

Inheritance diagram for OscillatoryWalker:



Classes

- class **Cell**

Public Member Functions

- OscillatoryWalker ()
- ImagePlus createSimulation ()
- void run ()

6.11.1 Detailed Description

Author

Carlos Alquezar

Definition at line 37 of file OscillatoryWalker.java.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 OscillatoryWalker()

```
OscillatoryWalker ( )
```

Definition at line 119 of file OscillatoryWalker.java.

6.11.3 Member Function Documentation

6.11.3.1 createSimulation()

```
ImagePlus createSimulation ( )
```

Returns

Definition at line 128 of file OscillatoryWalker.java.

6.11.3.2 run()

```
void run ( )
```

Definition at line 171 of file OscillatoryWalker.java.

The documentation for this class was generated from the following file:

- OscillatoryWalker.java

6.12 Paint Class Reference

Public Member Functions

- void chemotaxisTemplate (ColorProcessor ip, int numTracks, float chIdx, float slIdx, String sampleID)
- void draw (ImagePlus imp, SerializableList theTracks)
- void drawBoundaries (ImagePlus imp, List spermatozoa)
- void drawChemotaxis (Trial trial, float chIdx, float slIdx)
- void drawOutline (ImagePlus impOrig, ImagePlus impTh)
- void drawRoseDiagram (int[ ] histogram, int radius, float chIdx, String sampleID)

6.12.1 Detailed Description

Author

Carlos Alquezar

Definition at line 71 of file Paint.java.

6.12.2 Member Function Documentation

6.12.2.1 chemotaxisTemplate()

```
void chemotaxisTemplate (
            ColorProcessor ip,
            int numTracks,
            float chIdx,
            float slIdx,
            String sampleID )
```

Parameters

ip	
numTracks	
chIdx	
slIdx	
sampleID	

Definition at line 81 of file Paint.java.

6.12.2.2 draw()

```
void draw (
            ImagePlus imp,
            SerializableList theTracks )
```

Parameters

imp	
theTracks	2D-ArrayList with all the tracks

Definition at line 152 of file Paint.java.

6.12.2.3 drawBoundaries()

```
void drawBoundaries (
            ImagePlus imp,
            List spermatozoa )
```

Parameters

imp	
spermatozoa	

Definition at line 218 of file Paint.java.

6.12.2.4 drawChemotaxis()

```
void drawChemotaxis (
            Trial trial,
            float chIdx,
            float slIdx )
```

Parameters

trial.tracks	2D-ArrayList with all the tracks
chIdx	
slIdx	
trial.fieldWidth	
trial.fieldHeight	
trial.ID	

Definition at line 256 of file Paint.java.

6.12.2.5 drawOutline()

```
void drawOutline (
             ImagePlus impOrig,
             ImagePlus impTh )
```

Parameters

impOrig	
impTh	

Definition at line 307 of file Paint.java.

6.12.2.6 drawRoseDiagram()

```
void drawRoseDiagram (
             int [] histogram,
             int radius,
             float chIdx,
             String sampleID )
```

Parameters

histogram	
radius	
chIdx	
sampleID	

Definition at line 332 of file Paint.java.

The documentation for this class was generated from the following file:

- Paint.java

6.13 Params Class Reference

Static Public Member Functions

- static void resetParams ()
- static void saveParams ()

Static Public Attributes

- static float angleAmplitude = 90
- static int angleDelta = 4
- static float angleDirection = 0
- static float borderSize = 20

- static boolean compareOppositeDirections = false
- static String date = ""
- static boolean drawAvgTrajectories = true
- static boolean drawOrigTrajectories = true
- static float frameRate = 100
- static String genericField = ""
- static String male = ""
- static float maxDisplacement = 10
- static int MAXINSTANGLES = 20000
- static float maxSize = 400
- static double micronPerPixel = 1
- static float minSize = 40
- static int minTrackLength = 15
- static int NUMSAMPLES = 100
- static double pixelHeight = 1.0
- static double pixelWidth = 1.0
- static boolean printXY = false
- static float progressMotility = 80
- static float vclLowerTh = 45
- static float vclMin = 70
- static float vclUpperTh = 75
- static int wSize = 9

Static Private Attributes

- static Preferences prefs

6.13.1 Detailed Description

Author

Carlos Alquezar

Definition at line 33 of file Params.java.

6.13.2 Member Function Documentation

6.13.2.1 resetParams()

```
static void resetParams ( )  [static]
```

Definition at line 103 of file Params.java.

6.13.2.2 saveParams()

```
static void saveParams ( )  [static]
```

Definition at line 141 of file Params.java.

6.13.3 Member Data Documentation

6.13.3.1 angleAmplitude

```
float angleAmplitude = 90  [static]
```

Definition at line 36 of file Params.java.

6.13.3.2 angleDelta

```
int angleDelta = 4  [static]
```

This parameter is used to analyze the directionality angle between instant t and instant (t+angleDelta).

Definition at line 41 of file Params.java.

6.13.3.3 angleDirection

```
float angleDirection = 0  [static]
```

Angles used to clasify chemotactic trajectories

Definition at line 43 of file Params.java.

6.13.3.4 borderSize

```
float borderSize = 20  [static]
```

parameters used to compute BCF (equivalent to angleDelta)

Definition at line 47 of file Params.java.

6.13.3.5 compareOppositeDirections

```
boolean compareOppositeDirections = false  [static]
```

Definition at line 49 of file Params.java.

6.13.3.6 date

```
String date = ""  [static]
```

Definition at line 51 of file Params.java.

6.13.3.7 drawAvgTrajectories

```
boolean drawAvgTrajectories = true  [static]
```

Draw original trajectories over the ImagePlus

Definition at line 53 of file Params.java.

6.13.3.8 drawOrigTrajectories

```
boolean drawOrigTrajectories = true  [static]
```

Draw original trajectories over the ImagePlus

Definition at line 55 of file Params.java.

6.13.3.9 frameRate

```
float frameRate = 100  [static]
```

frame rate

Definition at line 57 of file Params.java.

6.13.3.10 genericField

```
String genericField = ""  [static]
```

Definition at line 59 of file Params.java.

6.13.3.11 male

```
String male = ""  [static]
```

Definition at line 61 of file Params.java.

6.13.3.12 maxDisplacement

```
float maxDisplacement = 10  [static]
```

maximum displacement of one spermatozoon between consecutive frames (um)

Definition at line 65 of file Params.java.

6.13.3.13 MAXINSTANGLES

```
int MAXINSTANGLES = 20000  [static]
```

Used to calculate OR ratios

Definition at line 67 of file Params.java.

6.13.3.14 maxSize

```
float maxSize = 400  [static]
```

maximum sperm size

Definition at line 69 of file Params.java.

6.13.3.15 micronPerPixel

```
double micronPerPixel = 1  [static]
```

Microns per pixel

Definition at line 73 of file Params.java.

6.13.3.16 minSize

```
float minSize = 40  [static]
```

minimum sperm size

Definition at line 75 of file Params.java.

6.13.3.17 minTrackLength

```
int minTrackLength = 15  [static]
```

minimum length of sperm track (in frames)

Definition at line 77 of file Params.java.

6.13.3.18 NUMSAMPLES

```
int NUMSAMPLES = 100  [static]
```

Definition at line 79 of file Params.java.

6.13.3.19 pixelHeight

```
double pixelHeight = 1.0  [static]
```

Definition at line 81 of file Params.java.

6.13.3.20 pixelWidth

```
double pixelWidth = 1.0  [static]
```

Definition at line 83 of file Params.java.

6.13.3.21 prefs

```
Preferences prefs  [static], [private]
```

Definition at line 85 of file Params.java.

6.13.3.22 printXY

```
boolean printXY = false  [static]
```

if true, print the xy co-ordinates for all tracks as tsv (tab separated values).

Definition at line 90 of file Params.java.

6.13.3.23 progressMotility

```
float progressMotility = 80  [static]
```

Parameter used to determine progressive motility sperm

Definition at line 92 of file Params.java.

6.13.3.24 vclLowerTh

```
float vclLowerTh = 45  [static]
```

Definition at line 94 of file Params.java.

6.13.3.25 vclMin

```
float vclMin = 70  [static]
```

Motility filter for motile and non motile sperm

Definition at line 96 of file Params.java.

6.13.3.26 vclUpperTh

```
float vclUpperTh = 75  [static]
```

Definition at line 98 of file Params.java.

6.13.3.27 wSize

```
int wSize = 9  [static]
```

Window size for moving average method (um)

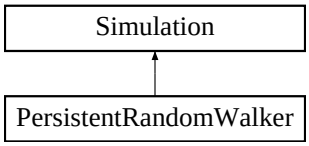
Definition at line 100 of file Params.java.

The documentation for this class was generated from the following file:

- Params.java

6.14 PersistentRandomWalker Class Reference

Inheritance diagram for PersistentRandomWalker:



Classes

- class **Cell**
- class **Obstacle**

Public Member Functions

- PersistentRandomWalker ()
- PersistentRandomWalker (double b, double responsiveCells)
- PersistentRandomWalker (double b, double responsiveCells, int simlength)
- ImagePlus createSimulation ()
- void run ()

6.14.1 Detailed Description

Author

C9l225Definition at line

6.14.2 Constructor & Destructor Documentation

6.14.2.1 PersistentRandomWalker() [1/3]

```
PersistentRandomWalker ( )
```

Definition at line 157 of file PersistentRandomWalker.java.

6.14.2.2 PersistentRandomWalker() [2/3]

```
PersistentRandomWalker (
            double b,
            double responsiveCells )
```

Parameters

b	
responsiveCells	

Definition at line 171 of file PersistentRandomWalker.java.

6.14.2.3 PersistentRandomWalker() [3/3]

```
PersistentRandomWalker (
            double b,
            double responsiveCells,
            int simlength )
```

Parameters

b	
responsiveCells	
simlength	

Definition at line 185 of file PersistentRandomWalker.java.

6.14.3 Member Function Documentation

6.14.3.1 createSimulation()

```
ImagePlus createSimulation ( )
```

Definition at line 200 of file PersistentRandomWalker.java.

6.14.3.2 run()

```
void run ( )
```

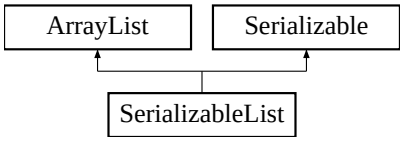
Definition at line 233 of file PersistentRandomWalker.java.

The documentation for this class was generated from the following file:

- PersistentRandomWalker.java

6.15 SerializableList Class Reference

Inheritance diagram for SerializableList:



Public Member Functions

- SerializableList ()
- SerializableList (Collection c)
- SerializableList (int initialCapacity)

6.15.1 Detailed Description

Author

Carlos Alquezar This class extends ArrayList to make it serializable

Definition at line 29 of file SerializableList.java.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 SerializableList() [1/3]

```
SerializableList ( )
```

Definition at line 34 of file SerializableList.java.

6.15.2.2 SerializableList() [2/3]

```
SerializableList (
            Collection c )
```

Parameters

c	

Definition at line 39 of file SerializableList.java.

6.15.2.3 SerializableList() [3/3]

```
SerializableList (
            int initialCapacity )
```

Parameters

initialCapacity	

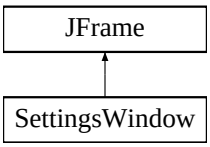
Definition at line 46 of file SerializableList.java.

The documentation for this class was generated from the following file:

- SerializableList.java

6.16 SettingsWindow Class Reference

Inheritance diagram for SettingsWindow:



Public Member Functions

- SettingsWindow (String title) throws HeadlessException
- JPanel createChemotaxisBox ()
- JPanel createGeneralBox ()
- JPanel createMotilityBox ()
- JPanel createVideoBox ()
- void setParameters ()

  Set Params static fields with the values introduced by the user.

Private Member Functions

- JTabbedPane addTabPane ()
- void createButtons ()
- void createGUI ()

6.16.1 Detailed Description

Author

Carlos Alquezar

Definition at line 44 of file SettingsWindow.java.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 SettingsWindow()

```
SettingsWindow (
            String title ) throws HeadlessException
```

Parameters

title	• Window's title

Definition at line 75 of file SettingsWindow.java.

6.16.3 Member Function Documentation

6.16.3.1 addTabPane()

```
JTabbedPane addTabPane ( )  [private]
```

Definition at line 87 of file SettingsWindow.java.

6.16.3.2 createButtons()

```
void createButtons ( )  [private]
```

Definition at line 97 of file SettingsWindow.java.

6.16.3.3 createChemotaxisBox()

```
JPanel createChemotaxisBox ( )
```

Returns

JPanel with all elements

Definition at line 119 of file SettingsWindow.java.

6.16.3.4 createGeneralBox()

```
JPanel createGeneralBox ( )
```

Returns

JPanel with all elements

Definition at line 170 of file SettingsWindow.java.

6.16.3.5 createGUI()

```
void createGUI ( )  [private]
```

Definition at line 225 of file SettingsWindow.java.

6.16.3.6 createMotilityBox()

```
JPanel createMotilityBox ( )
```

Returns

JPanel with all elements

Definition at line 254 of file SettingsWindow.java.

6.16.3.7 createVideoBox()

```
JPanel createVideoBox ( )
```

Returns

JPanel with all elements

Definition at line 304 of file SettingsWindow.java.

6.16.3.8 setParameters()

```
void setParameters ( )
```

Set Params static fields with the values introduced by the user.

Definition at line 355 of file SettingsWindow.java.

The documentation for this class was generated from the following file:

- SettingsWindow.java

6.17 SignalProcessing Class Reference

Public Member Functions

- SerializableList averageTracks (SerializableList theTracks)
- List decimateTrack (List track, int factor)
- List decimateTracks (List theTracks, int factor)
- SerializableList filterTracksByLength (SerializableList theTracks)
- SerializableList filterTracksByMotility (SerializableList theTracks)
- float [ ] movingAverage (float[ ] points, int wSize)
- List movingAverage (List track)
- List movingAverage (List track, int wSize)

6.17.1 Detailed Description

Author

Carlos Alquezar

Definition at line 33 of file SignalProcessing.java.

6.17.2 Member Function Documentation

6.17.2.1 averageTracks()

```
SerializableList averageTracks (
            SerializableList theTracks )
```

Fuction to calculate the average path of all tracks using a moving average filter

Parameters

theTracks	2D-ArrayList with all the tracks

Returns

2D-ArrayList with the averaged tracks

Definition at line 44 of file SignalProcessing.java.

6.17.2.2 decimateTrack()

```
List decimateTrack (
            List track,
            int factor )
```

Fuction to decimate a track

Parameters

track	• a track
factor	• decimation factor

Returns

Decimated track

Definition at line 65 of file SignalProcessing.java.

6.17.2.3 decimateTracks()

```
List decimateTracks (
            List theTracks,
            int factor )
```

Function to decimate all tracks

Parameters

theTracks	- 2D-ArrayList with all the tracks
factor	- decimation factor

Returns

2D-ArrayList with all the tracks decimated

Definition at line 86 of file SignalProcessing.java.

6.17.2.4 filterTracksByLength()

```
SerializableList filterTracksByLength (
            SerializableList theTracks )
```

Parameters

theTracks	- 2D-ArrayList with all the tracks

Returns

2D-ArrayList with all the tracks that have passed the filter

Definition at line 100 of file SignalProcessing.java.

6.17.2.5 filterTracksByMotility()

```
SerializableList filterTracksByMotility (
            SerializableList theTracks )
```

Parameters

theTracks	- 2D-ArrayList with all the tracks

Returns

2D-ArrayList with all the tracks that have passed the filter

Definition at line 115 of file SignalProcessing.java.

6.17.2.6 movingAverage() [1/3]

```
float [] movingAverage (
            float [] points,
            int wSize )
```

Parameters

points	
wSize	

Returns

Definition at line 131 of file SignalProcessing.java.

6.17.2.7 movingAverage() [2/3]

```
List movingAverage (
            List track )
```

Parameters

track	

Returns

Definition at line 148 of file SignalProcessing.java.

6.17.2.8 movingAverage() [3/3]

```
List movingAverage (
            List track,
            int wSize )
```

Function to calculate the average path of a track using a moving average filter

Parameters

track	Array list that stores one track
wSize	

Returns

ArrayList with the averaged track

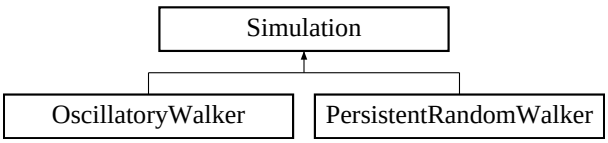
Definition at line 162 of file SignalProcessing.java.

The documentation for this class was generated from the following file:

- SignalProcessing.java

6.18 Simulation Class Reference

Inheritance diagram for Simulation:



Public Member Functions

- abstract ImagePlus createSimulation ()
- abstract void run ()

6.18.1 Detailed Description

Author

Carlos Alquezar

Definition at line 28 of file Simulation.java.

6.18.2 Member Function Documentation

6.18.2.1 createSimulation()

```
abstract ImagePlus createSimulation ( )  [abstract]
```

Returns

6.18.2.2 run()

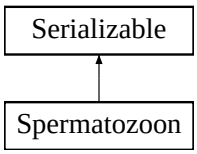
```
abstract void run ( )  [abstract]
```

The documentation for this class was generated from the following file:

- Simulation.java

6.19 Spermatozoon Class Reference

Inheritance diagram for Spermatozoon:



Public Member Functions

- void copy (Spermatozoon source)
- float distance (Spermatozoon s)

Public Attributes

- String id = "*"
- boolean flag = false
- boolean inTrack = false
- int trackNr
- float x
- float y
- int z
- float bx
- float by
- float width
- float height
- boolean selected = false
- float total_area = -1
- float total_perimeter = -1
- float total_feret = -1
- float total_minFeret = -1

Static Private Attributes

- static final long serialVersionUID = 1L

6.19.1 Detailed Description

Author

Carlos Alquezar

Definition at line 27 of file Spermatozoon.java.

6.19.2 Member Function Documentation

6.19.2.1 copy()

```
void copy (
            Spermatozoon source )
```

Parameters

source	- Spermatozoon to be copied

Definition at line 71 of file Spermatozoon.java.

6.19.2.2 distance()

```
float distance (
            Spermatozoon s )
```

Parameters

s	• Spermatozoon used as reference to calculate the distance

Returns

euclidean distance to the Spermatozoon s

Definition at line 95 of file Spermatozoon.java.

6.19.3 Member Data Documentation

6.19.3.1 bx

```
float bx
```

Definition at line 49 of file Spermatozoon.java.

6.19.3.2 by

```
float by
```

Definition at line 51 of file Spermatozoon.java.

6.19.3.3 flag

```
boolean flag = false
```

Definition at line 36 of file Spermatozoon.java.

6.19.3.4 height

```
float height
```

Definition at line 55 of file Spermatozoon.java.

6.19.3.5 id

```
String id = "*"
```

Definition at line 34 of file Spermatozoon.java.

6.19.3.6 inTrack

```
boolean inTrack = false
```

Definition at line 38 of file Spermatozoon.java.

6.19.3.7 selected

```
boolean selected = false
```

Definition at line 58 of file Spermatozoon.java.

6.19.3.8 serialVersionUID

```
final long serialVersionUID = 1L  [static], [private]
```

Definition at line 32 of file Spermatozoon.java.

6.19.3.9 total_area

```
float total_area = -1
```

Definition at line 61 of file Spermatozoon.java.

6.19.3.10 total_feret

```
float total_feret = -1
```

Definition at line 65 of file Spermatozoon.java.

6.19.3.11 total_minFeret

```
float total_minFeret = -1
```

Definition at line 67 of file Spermatozoon.java.

6.19.3.12 total_perimeter

```
float total_perimeter = -1
```

Definition at line 63 of file Spermatozoon.java.

6.19.3.13 trackNr

```
int trackNr
```

Definition at line 40 of file Spermatozoon.java.

6.19.3.14 width

```
float width
```

Definition at line 53 of file Spermatozoon.java.

6.19.3.15 x

```
float x
```

Definition at line 42 of file Spermatozoon.java.

6.19.3.16 y

```
float y
```

Definition at line 44 of file Spermatozoon.java.

6.19.3.17 z

```
int z
```

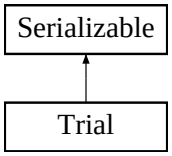
Definition at line 46 of file Spermatozoon.java.

The documentation for this class was generated from the following file:

- Spermatozoon.java

6.20 Trial Class Reference

Inheritance diagram for Trial:



Public Member Functions

- Trial ()
- Trial (String ID, String type, String source, SerializableList t)
- Trial (String ID, String type, String source, SerializableList t, int width, int height)

Public Attributes

- String ID = ""
- String type = ""
- String source = ""
- SerializableList tracks = null
- int fieldWidth = 0
- int fieldHeight = 0

Static Private Attributes

- static final long serialVersionUID = 1L

6.20.1 Detailed Description

Author

Carlos Alquezar

Definition at line 27 of file Trial.java.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 Trial() [1/3]

```
Trial ( )
```

Definition at line 45 of file Trial.java.

6.20.2.2 Trial() [2/3]

```
Trial (
            String ID,
            String type,
            String source,
            SerializableList t )
```

Parameters

ID	
type	
source	
t	

Definition at line 53 of file Trial.java.

6.20.2.3 Trial() [3/3]

```
Trial (
            String ID,
            String type,
            String source,
            SerializableList t,
            int width,
            int height )
```

Parameters

ID	
type	
source	
t	
imp	
motileSperm	

Definition at line 67 of file Trial.java.

6.20.3 Member Data Documentation

6.20.3.1 fieldHeight

```
int fieldHeight = 0
```

Definition at line 43 of file Trial.java.

6.20.3.2 fieldWidth

```
int fieldWidth = 0
```

Definition at line 41 of file Trial.java.

6.20.3.3 ID

```
String ID = ""
```

Definition at line 33 of file Trial.java.

6.20.3.4 serialVersionUID

```
final long serialVersionUID = 1L  [static], [private]
```

Definition at line 31 of file Trial.java.

6.20.3.5 source

```
String source = ""
```

source's filename

Definition at line 37 of file Trial.java.

6.20.3.6 tracks

```
SerializableList tracks = null
```

Definition at line 39 of file Trial.java.

6.20.3.7 type

```
String type = ""
```

Definition at line 35 of file Trial.java.

The documentation for this class was generated from the following file:

- Trial.java

6.21 TrialManager Class Reference

Public Member Functions

- Trial getTrialFromAVI (String path)
- Trial getTrialFromImp (ImagePlus impOrig, String path)
- Map< String, Trial > readTrials ()
- void saveTrials (Map< String, Trial > trials)
- Trial simulateTrial (String trialID, double beta, double responsiveCells)
- Map< String, Trial > simulateTrials (double beta, double responsiveCells, int MAXSIMULATIONS)

6.21.1 Detailed Description

Author

Carlos Alquezar

Definition at line 42 of file TrialManager.java.

6.21.2 Member Function Documentation

6.21.2.1 getTrialFromAVI()

```
Trial getTrialFromAVI (
            String path )
```

Parameters

analysis	
path	

Returns

Definition at line 50 of file TrialManager.java.

6.21.2.2 getTrialFromImp()

```
Trial getTrialFromImp (
            ImagePlus impOrig,
            String path )
```

Parameters

impOrig	
analysis	
trialID	
trialType	
relativePath	

Returns

Definition at line 69 of file TrialManager.java.

6.21.2.3 readTrials()

```
Map<String, Trial> readTrials ( )
```

Returns

Definition at line 92 of file TrialManager.java.

6.21.2.4 saveTrials()

```
void saveTrials (
            Map< String, Trial > trials )
```

Parameters

trials	

Definition at line 112 of file TrialManager.java.

6.21.2.5 simulateTrial()

```
Trial simulateTrial (
            String trialID,
            double beta,
            double responsiveCells )
```

Parameters

trialID	
beta	
responsiveCells	

Returns

Definition at line 143 of file TrialManager.java.

6.21.2.6 simulateTrials()

```
Map<String, Trial> simulateTrials (
            double beta,
            double responsiveCells,
            int MAXSIMULATIONS )
```

Parameters

beta	
responsiveCells	
MAXSIMULATIONS	

Returns

Definition at line 158 of file TrialManager.java.

The documentation for this class was generated from the following file:

- TrialManager.java

6.22 ImageAnalysisWindow.TypeOfAnalysis Enum Reference

Public Attributes

- DIRECTORY
- FILE
- NONE

6.22.1 Detailed Description

Definition at line 59 of file ImageAnalysisWindow.java.

6.22.2 Member Data Documentation

6.22.2.1 DIRECTORY

```
DIRECTORY
```

Definition at line 60 of file ImageAnalysisWindow.java.

6.22.2.2 FILE

```
FILE
```

Definition at line 60 of file ImageAnalysisWindow.java.

6.22.2.3 NONE

NONE

Definition at line 60 of file ImageAnalysisWindow.java.

The documentation for this enum was generated from the following file:

- ImageAnalysisWindow.java

6.23 Motility.TypeOfAnalysis Enum Reference

Public Attributes

- DIRECTORIES
- DIRECTORY
- FILE
- NONE

6.23.1 Detailed Description

Definition at line 48 of file Motility.java.

6.23.2 Member Data Documentation

6.23.2.1 DIRECTORIES

DIRECTORIES

Definition at line 49 of file Motility.java.

6.23.2.2 DIRECTORY

DIRECTORY

Definition at line 49 of file Motility.java.

6.23.2.3 FILE

FILE

Definition at line 49 of file Motility.java.

6.23.2.4 NONE

NONE

Definition at line 49 of file Motility.java.

The documentation for this enum was generated from the following file:

- Motility.java

6.24 Chemotaxis.TypeOfAnalysis Enum Reference

Public Attributes

- BOOTSTRAPPING
- BOOTSTRAPPINGSIMULATIONS
- INDEXESDIRECTORY
- INDEXESFILE
- INDEXESSIMULATIONS
- NONE

6.24.1 Detailed Description

Definition at line 50 of file Chemotaxis.java.

6.24.2 Member Data Documentation

6.24.2.1 BOOTSTRAPPING

BOOTSTRAPPING

Definition at line 51 of file Chemotaxis.java.

6.24.2.2 BOOTSTRAPPINGSIMULATIONS

BOOTSTRAPPINGSIMULATIONS

Definition at line 51 of file Chemotaxis.java.

6.24.2.3 INDEXESDIRECTORY

INDEXESDIRECTORY

Definition at line 51 of file Chemotaxis.java.

6.24.2.4 INDEXESFILE

`INDEXESFILE`

Definition at line 51 of file Chemotaxis.java.

6.24.2.5 INDEXESSIMULATIONS

`INDEXESSIMULATIONS`

Definition at line 51 of file Chemotaxis.java.

6.24.2.6 NONE

`NONE`

Definition at line 51 of file Chemotaxis.java.

The documentation for this enum was generated from the following file:

- Chemotaxis.java

6.25 Utils Class Reference

Public Member Functions

- int analysisSelectionDialog (Object[ ] options, String question, String title)
- int [ ] convertLongArrayToInt (long[ ] orig)
- Spermatozoon getSpermatozoon (String id, List spermatozoa)
- String printXYCoords (List theTracks)

6.25.1 Detailed Description

Author

Carlos Alquezar

Definition at line 33 of file Utils.java.

6.25.2 Member Function Documentation

6.25.2.1 analysisSelectionDialog()

```
int analysisSelectionDialog (
            Object [ ] options,
            String question,
            String title )
```

Parameters

options	
question	
title	

Returns

Definition at line 42 of file Utils.java.

6.25.2.2 convertLongArrayToInt()

```
int [] convertLongArrayToInt (
            long [] orig )
```

Parameters

orig	

Returns

Definition at line 53 of file Utils.java.

6.25.2.3 getSpermatozoon()

```
Spermatozoon getSpermatozoon (
            String id,
            List spermatozoa )
```

Parameters

id	
spermatozoa	

Returns

Definition at line 66 of file Utils.java.

6.25.2.4 printXYCoords()

```
String printXYCoords (
            List theTracks )
```

Parameters

theTracks	2D-ArrayList with all the tracks

Returns

String with the results in tsv format (tab separated values)

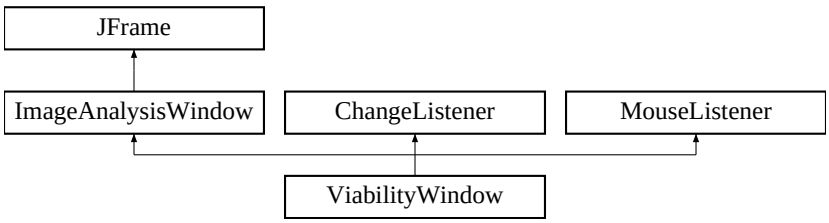
Definition at line 84 of file Utils.java.

The documentation for this class was generated from the following file:

- Utils.java

6.26 ViabilityWindow Class Reference

Inheritance diagram for ViabilityWindow:



Classes

- enum Channel

Public Member Functions

- ViabilityWindow ()
- void mouseClicked (MouseEvent e)
- void mouseEntered (MouseEvent e)
- void mouseExited (MouseEvent e)
- void mousePressed (MouseEvent e)
- void mouseReleased (MouseEvent e)
- void stateChanged (ChangeEvent e)

Protected Member Functions

- void drawImage ()
- void nextAction ()
- void processImage (boolean eventType)

Protected Attributes

- List< Spermatozoon > aliveSpermatozoa = new ArrayList<Spermatozoon>()
- List< Spermatozoon > deadSpermatozoa = new ArrayList<Spermatozoon>()

Private Member Functions

- void doSliderRefresh ()
- void generateResults ()
- List< Spermatozoon > getSpermatozoa (Channel rgbChannel)

Private Attributes

- Channel channel = Channel.NONE
- ImagePlus aliveImpOutline
- ImagePlus deadImpOutline
- boolean isThresholding = false
- ResultsTable results = new ResultsTable()

6.26.1 Detailed Description

Definition at line 39 of file ViabilityWindow.java.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 ViabilityWindow()

```
ViabilityWindow ( )
```

Constructor

Definition at line 56 of file ViabilityWindow.java.

6.26.3 Member Function Documentation

6.26.3.1 doSliderRefresh()

```
void doSliderRefresh ( )  [private]
```

This method refreshes the showed image after changing the threshold with the sliderbar

Definition at line 72 of file ViabilityWindow.java.

6.26.3.2 drawImage()

```
void drawImage ( )  [protected]
```

Definition at line 85 of file ViabilityWindow.java.

6.26.3.3 generateResults()

```
void generateResults ( )  [private]
```

Definition at line 110 of file ViabilityWindow.java.

6.26.3.4 getSpermatozoa()

```
List<Spermatozoon> getSpermatozoa (
            Channel rgbChannel )  [private]
```

Definition at line 135 of file ViabilityWindow.java.

6.26.3.5 mouseClicked()

```
void mouseClicked (
            MouseEvent e )
```

Definition at line 169 of file ViabilityWindow.java.

6.26.3.6 mouseEntered()

```
void mouseEntered (
            MouseEvent e )
```

Definition at line 173 of file ViabilityWindow.java.

6.26.3.7 mouseExited()

```
void mouseExited (
            MouseEvent e )
```

Definition at line 177 of file ViabilityWindow.java.

6.26.3.8 mousePressed()

```
void mousePressed (
            MouseEvent e )
```

Definition at line 181 of file ViabilityWindow.java.

6.26.3.9 mouseReleased()

```
void mouseReleased (
            MouseEvent e )
```

Definition at line 186 of file ViabilityWindow.java.

6.26.3.10 nextAction()

```
void nextAction ( )  [protected]
```

Definition at line 191 of file ViabilityWindow.java.

6.26.3.11 processImage()

```
void processImage (
            boolean eventType )  [protected]
```

Definition at line 196 of file ViabilityWindow.java.

6.26.3.12 stateChanged()

```
void stateChanged (
            ChangeEvent e )
```

Definition at line 216 of file ViabilityWindow.java.

6.26.4 Member Data Documentation

6.26.4.1 aliveImpOutline

```
ImagePlus aliveImpOutline  [private]
```

Definition at line 45 of file ViabilityWindow.java.

6.26.4.2 aliveSpermatozoa

```
List<Spermatozoon> aliveSpermatozoa = new ArrayList<Spermatozoon>()  [protected]
```

Definition at line 46 of file ViabilityWindow.java.

6.26.4.3 channel

```
Channel channel = Channel.NONE  [private]
```

Definition at line 44 of file ViabilityWindow.java.

6.26.4.4 deadImpOutline

```
ImagePlus deadImpOutline  [private]
```

Definition at line 47 of file ViabilityWindow.java.

6.26.4.5 deadSpermatozoa

```
List<Spermatozoon> deadSpermatozoa = new ArrayList<Spermatozoon>()  [protected]
```

Definition at line 48 of file ViabilityWindow.java.

6.26.4.6 isThresholding

```
boolean isThresholding = false  [private]
```

Definition at line 49 of file ViabilityWindow.java.

6.26.4.7 results

```
ResultsTable results = new ResultsTable()  [private]
```

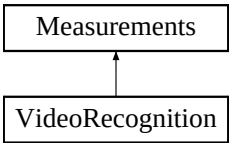
Definition at line 51 of file ViabilityWindow.java.

The documentation for this class was generated from the following file:

- ViabilityWindow.java

6.27 VideoRecognition Class Reference

Inheritance diagram for VideoRecognition:



Public Member Functions

- VideoRecognition ()
- SerializableList analyzeVideo (ImagePlus imp)
- List [ ] detectSpermatozoa (ImagePlus imp)
- SerializableList idenfityTracks (List[ ] spermatozoa, int nFrames)

6.27.1 Detailed Description

Definition at line 64 of file VideoRecognition.java.

6.27.2 Constructor & Destructor Documentation

6.27.2.1 VideoRecognition()

```
VideoRecognition ( )
```

Definition at line 66 of file VideoRecognition.java.

6.27.3 Member Function Documentation

6.27.3.1 analyzeVideo()

```
SerializableList analyzeVideo (
            ImagePlus imp )
```

Parameters

ImagePlus	imp

Returns

Definition at line 73 of file VideoRecognition.java.

6.27.3.2 detectSpermatozoa()

```
List [] detectSpermatozoa (
            ImagePlus imp )
```

Parameters

imp	ImagePlus

Returns

2D-ArrayList with all spermatozoa detected for each frame

Definition at line 108 of file VideoRecognition.java.

6.27.3.3 idenfityTracks()

```
SerializableList idenfityTracks (
            List [] spermatozoa,
            int nFrames )
```

Parameters

spermatozoa	2D-ArrayList with all spermatozoa detected for each frame
nFrames	

Returns

2D-ArrayList with all tracks detected

Definition at line 169 of file VideoRecognition.java.

The documentation for this class was generated from the following file:

- VideoRecognition.java

Chapter 7

File Documentation

7.1 Chemotaxis.java File Reference

Classes

- class Chemotaxis
- enum Chemotaxis.TypeOfAnalysis

Packages

- package analysis

7.2 Chemotaxis.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package analysis;
00020
00021 import java.util.ArrayList;
00022 import java.util.Collections;
00023 import java.util.HashMap;
00024 import java.util.List;
00025 import java.util.ListIterator;
00026 import java.util.Map;
00027 import java.util.Set;
00028
00029 import javax.swing.SwingWorker;
00030
00031 import data.Params;
00032 import data.SerializableList;
00033 import data.Spermatozoon;
```

```
00034 import data.Trial;
00035 import functions.FileManager;
00036 import functions.Paint;
00037 import functions.TrialManager;
00038 import functions.Utils;
00039 import ij.IJ;
00040 import ij.gui.GenericDialog;
00041 import ij.measure.ResultsTable;
00042
00048 public class Chemotaxis extends SwingWorker<Boolean, String> {
00049
00050   private enum TypeOfAnalysis {
00051     BOOTSTRAPPING, BOOTSTRAPPINGSIMULATIONS, INDEXESDIRECTORY, INDEXESFILE, INDEXESSIMULATIONS, NONE
00052   }
00053
00054   private TypeOfAnalysis analysis = TypeOfAnalysis.
      NONE;
00055
00066   private ResultsTable analyseCondition(Map<String, Trial> controls, Map<String, Trial>
      tests) {
00067     ResultsTable rt = new ResultsTable();
00068     switch (analysis) {
00069       case INDEXESDIRECTORY:
00070       case INDEXESSIMULATIONS:
00071         rt = indexesAnalysis(controls, tests);
00072         break;
00073       case BOOTSTRAPPING:
00074       case BOOTSTRAPPINGSIMULATIONS:
00075         rt = bootstrappingAnalysis(controls, tests);
00076         break;
00077       default:
00078     }
00079     return rt;
00080   }
00081
00088   private void analyseDirectory() {
00089     FileManager fm = new FileManager();
00090     String folder = fm.selectFolder();
00091     Map<String, Trial> cTrials = getControlTrials(folder);
00092     List<String> testFolders = getTestFolders(folder);
00093     if(testFolders.size()==0){
00094       IJ.showMessage("No \"test\" folders have been found");
00095       return;
00096     }
00097     for (String f : testFolders) {
00098       List<String> tests = fm.getFiles(f);
00099       Map<String, Trial> tTrials = getTrials(tests);
00100       ResultsTable rt = analyseCondition(cTrials, tTrials);
00101       String condition = fm.getFilename(f);
00102       if(rt!=null)
00103         rt.show(condition);
00104     }
00105   }
00106
00111   private void analyseFile() {
00112     FileManager fm = new FileManager();
00113     String file = fm.selectFile();
00114     TrialManager tm = new TrialManager();
00115     Trial trial = tm.getTrialFromAVI(file);
00116     drawResults(trial);
00117     if(Params.printXY){
00118       Utils utils = new Utils();
00119       IJ.saveString(utils.printXYCoords(trial.tracks),"");
00120     }
00121   }
00122
00128   private void analyseSimulations() {
00129     GenericDialog gd = new GenericDialog("Set Simulation parameters");
00130     gd.addNumericField("Beta", 0, 2);
00131     gd.addNumericField("Responsiveness (%)", 50, 2);
00132     gd.addNumericField("Number of simulations", 50, 0);
00133     gd.showDialog();
00134     if (gd.wasCanceled())
00135       return;
00136     final double BETA = gd.getNextNumber();
00137     final double RESPONSIVENESS = gd.getNextNumber() / 100;// value must be
00138                                                            // between [0,1]
00139     final int TOTALSIMULATIONS = (int) gd.getNextNumber();
00140     TrialManager tm = new TrialManager();
00141     Map<String, Trial> controls = tm.simulateTrials(0, 0, TOTALSIMULATIONS);
00142     Map<String, Trial> tests = tm.simulateTrials(BETA, RESPONSIVENESS, TOTALSIMULATIONS);
00143     ResultsTable rt = analyseCondition(controls, tests);
00144     rt.show("Results from Simulation (Beta: " + BETA + ",Responsiveness: " + RESPONSIVENESS + ")");
00145   }
00146
00156   private ResultsTable bootstrappingAnalysis(Map<String, Trial> controls, Map<String,
       Trial> tests) {
```

```
00157
00158     final int cMin = minSampleSize(controls);
00159     final int tMin = minSampleSize(tests);
00160     Params.MAXINSTANGLES = Math.min(cMin, tMin);
00161     ResultsTable rt = new ResultsTable();
00162     if(!checkPairs(controls,tests)){
00163        String condition = ((Trial) tests.values().toArray()[0]).type;
00164        IJ.showMessage("No pairs (control-"+condition+") with the same ID have been found");
00165        return null;
00166     }
00167     // Calculating OR threshold via resampling
00168     double thControl = orThreshold(controls);
00169     for (String cKey : controls.keySet()) {
00170       Trial cTrial = (Trial) controls.get(cKey);
00171       Trial tTrial = findTrial(cTrial.ID, tests);
00172       if (tTrial != null) {
00173         double or = or(cTrial, tTrial);
00174         setBootstrappingResults(rt, or, thControl, tTrial);
00175       }
00176     }
00177     return rt;
00178   }
00179   /*****************************************************/
00187   private float calculateChIndex(List<List<Spermatozoon>> theTracks) {
00188     int trackNr = 0; // Number of tracks
00189     int nTracks = theTracks.size();
00190     int[] displacements = { 0, 0 };
00191     for (List<Spermatozoon> track : theTracks) {
00192       IJ.showProgress((double) trackNr / nTracks);
00193       IJ.showStatus("Calculating Ch-Index...");
00194       trackNr++;
00195       int[] instD = countInstantDisplacements(track);
00196       displacements[0] += instD[0];
00197       displacements[1] += instD[1];
00198     }
00199     float nUpGradient = displacements[0]; // Number of displacements in the
00200                                           // gradient direction
00201     float nOtherDirs = displacements[1]; // Number of displacements in other
00202                                          // direction
00203     // IJ.log("nUpGradient: "+nUpGradient+"; nOtherDirs: "+nOtherDirs);
00204     float chIdx = 0;
00205     if ((nUpGradient + nOtherDirs) > 0) {
00206       chIdx = nUpGradient / (nUpGradient + nOtherDirs); // (nUpGradient+nOtherDirs)
00207                                                         // = Total number of
00208                                                         // shifts
00209     } else {
00210       chIdx = -1;
00211     }
00212     return chIdx; // return index between [0,1]
00213   }
00214
00215   /*****************************************************/
00223   private float calculateSLIndex(List<List<Spermatozoon>> theTracks) {
00224
00225     float nUpGradient = 0; // Number of shifts in the chemoattractant direction
00226     float nOtherDirs = 0; // Number of shifts in other direction
00227     int trackNr = 0;
00228     int nTracks = theTracks.size();
00229     double angleChemotaxis = (2 * Math.PI + (Params.angleAmplitude / 2) * Math.PI / 180
      ) % (2 * Math.PI);
00230     float ratioSL = 0;
00231     for (List<Spermatozoon> aTrack : theTracks) {
00232       IJ.showProgress((double) trackNr / nTracks);
00233       IJ.showStatus("Calculating SL-Index...");
00234       trackNr++;
00235       Spermatozoon first = (Spermatozoon) aTrack.get(1);
00236       Spermatozoon last = (Spermatozoon) aTrack.get(aTrack.size() - 1);
00237       double angle = relativeAngle(first, last); // Between [-PI,PI]
00238       // Check if the angle is upGradient or not
00239       if (Math.abs(angle) < angleChemotaxis) {
00240         nUpGradient++;
00241       } else {
00242         nOtherDirs++;
00243       }
00244     }
00245     if ((nUpGradient + nOtherDirs) > 0) {
00246       ratioSL = nUpGradient / (nUpGradient + nOtherDirs);
00247     } else {
00248       ratioSL = -1;
00249     }
00250     return ratioSL;
00251   }
00252
00259   private boolean checkPairs(Map<String, Trial> controls, Map<String, Trial> tests){
00260     boolean ok = false;
00261     for (String cKey : controls.keySet()) {
00262       Trial cTrial = (Trial) controls.get(cKey);
```

```
00263       Trial tTrial = findTrial(cTrial.ID, tests);
00264       if (tTrial != null) {
00265         ok=true;
00266         return ok;
00267       }
00268     }
00269     return ok;
00270   }
00271
00281   private int[] circularHistogram(List<Double> angles, int n) {
00282
00283     int[] histogram = new int[n];
00284     for (int i = 0; i < n; i++) {
00285       histogram[i] = 0;
00286     }
00287     final int BINSIZE = 360 / n;
00288     for (int i = 0; i < angles.size(); i++) {
00289       int bin = angles.get(i).intValue() / BINSIZE;
00290       histogram[bin]++;
00291     }
00292     return histogram;
00293   }
00294
00306   private int[] countAngles(SerializableList theTracks) {
00307     int[] angles = { 0, 0 };
00308     for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
00309       List aTrack = (ArrayList) iT.next();
00310       int[] instantAngles = countInstantDisplacements(aTrack);
00311       angles[0] += instantAngles[0];
00312       angles[1] += instantAngles[1];
00313     }
00314     return angles;
00315   }
00316
00328   private int[] countInstantDisplacements(List<Spermatozoon> track) {
00329     int nUpGradient = 0;
00330     int nOtherDir = 0;
00331     int nPoints = track.size();
00332     double angleChemotaxis = (2 * Math.PI + (Params.angleAmplitude / 2) * Math.PI / 180
     ) % (2 * Math.PI);
00333     for (int j = 0; j < (nPoints - Params.angleDelta); j++) {
00334       Spermatozoon oldSpermatozoon = (Spermatozoon) track.get(j);
00335       Spermatozoon newSpermatozoon = (Spermatozoon) track.get(j +
    Params.angleDelta);
00336       double angle = relativeAngle(oldSpermatozoon, newSpermatozoon);// Between interval
     [-PI,PI]
00337       if (Params.compareOppositeDirections) {// We only take into account
     angles in the gradient and opposite direction
00338         if (Math.abs(angle) < angleChemotaxis) {
00339           nUpGradient++;
00340         } else if (Math.abs(angle) > (Math.PI - angleChemotaxis)) {
00341           nOtherDir++;
00342         }
00343       } else {// We take into account all angles in all directions
00344         if (Math.abs(angle) < angleChemotaxis) {
00345           nUpGradient++;
00346         } else {
00347           nOtherDir++;
00348         }
00349       }
00350     }
00351     int[] results = new int[2];
00352     results[0] = nUpGradient;
00353     results[1] = nOtherDir;
00354     return results;
00355   }
00356
00361   @Override
00362   public Boolean doInBackground() {
00363
00364     switch (analysis) {
00365       case INDEXESFILE:
00366         analyseFile();
00367         break;
00368       case INDEXESDIRECTORY:
00369       case BOOTSTRAPPING:
00370         analyseDirectory();
00371         break;
00372       case INDEXESSIMULATIONS:
00373       case BOOTSTRAPPINGSIMULATIONS:
00374         analyseSimulations();
00375         break;
00376     }
00377
00378     return null;
00379   }
00380
```

```
00385   @Override
00386   protected void done() {
00387     // switch (analysis) {
00388     // case INDEXESFILE:
00389     // break;
00390     // case INDEXESDIRECTORY:
00391     // break;
00392     // case BOOTSTRAPPING:
00393     // break;
00394     // case INDEXESSIMULATIONS:
00395     // break;
00396     // case BOOTSTRAPPINGSIMULATIONS:
00397     // break;
00398     // }
00399   }
00400
00407   private void drawResults(Trial trial) {
00408     if (trial == null)
00409       return;
00410     float chIdx = calculateChIndex(trial.tracks);
00411     float slIdx = calculateSLIndex(trial.tracks);
00412     int[] hist = circularHistogram(getListOfAngles(trial.
      tracks), 45);
00413     int radius = trial.fieldWidth;
00414     Paint paint = new Paint();
00415     paint.drawChemotaxis(trial, chIdx, slIdx);
00416     paint.drawRoseDiagram(hist, radius, chIdx, trial.source);
00417   }
00418
00429   private Trial findTrial(String id, Map<String, Trial> trials) {
00430     for (String k : trials.keySet()) {
00431       Trial trial = (Trial) trials.get(k);
00432       if (trial.ID.equalsIgnoreCase(id))
00433         return trial;
00434     }
00435     return null;
00436   }
00437
00438   private Map<String, Trial> getControlTrials(String folder) {
00439     FileManager fm = new FileManager();
00440     List<String> subFolders = fm.getSubfolders(folder);
00441     String controlFolder = null;
00442     for (int i = 0; i < subFolders.size(); i++) {
00443       String tempName = subFolders.get(i).toLowerCase();
00444       tempName = fm.getFilename(tempName);
00445       if (tempName.equals("control")) {
00446         controlFolder = subFolders.get(i);
00447         break;
00448       }
00449     }
00450     if(controlFolder==null){
00451       IJ.showMessage("No \"control\" folder has been found");
00452       return new HashMap<String, Trial>();
00453     }
00454     List<String> controlFiles = fm.getFiles(controlFolder);
00455     Map<String, Trial> cTrials = new HashMap<String, Trial>();
00456     TrialManager tm = new TrialManager();
00457     for (int i = 0; i < controlFiles.size(); i++) {
00458       String file = controlFiles.get(i);
00459       Trial trial = tm.getTrialFromAVI(file);
00460       if (trial != null)
00461         cTrials.put(trial.type + "-_-" + trial.ID, trial);
00462     }
00463     return cTrials;
00464   }
00465
00474   private List<Double> getListOfAngles(SerializableList theTracks) {
00475     List<Double> instAngles = new ArrayList<Double>();
00476     for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
00477       List track = (ArrayList) iT.next();
00478       int nPoints = track.size();
00479       for (int j = 0; j < (nPoints - Params.angleDelta); j++) {
00480         Spermatozoon oldSpermatozoon = (Spermatozoon) track.get(j);
00481         Spermatozoon newSpermatozoon = (Spermatozoon) track.get(j +
      Params.angleDelta);
00482         float diffX = newSpermatozoon.x - oldSpermatozoon.x;
00483         float diffY = newSpermatozoon.y - oldSpermatozoon.y;
00484         double angle = (360 + Math.atan2(diffY, diffX) * 180 / Math.PI) % (360); // Absolute
00485                                                                                  // angle
00486         instAngles.add(angle);
00487       }
00488     }
00489     return instAngles;
00490   }
00491
00505   private double[] getOddsValues(SerializableList tracks) {
00506
```

```
00507     double[] values = new double[] { 0.0, 0.0 };// [0]-upgradient
00508                                                 // displacements;[1]-displacements
00509                                                 // to other directionality
00510     int count = 0;
00511     int index = 0;
00512     while ((count < Params.MAXINSTANGLES) && (index < tracks.size())) {
00513       int[] countInstDirections = countInstantDisplacements((List) tracks.get(
      index));
00514       count += countInstDirections[0] + countInstDirections[1];
00515       values[0] += (double) countInstDirections[0]; // number of instantaneous
00516                                                     // angles in the positive
00517                                                     // direction
00518       values[1] += (double) (countInstDirections[0] + countInstDirections[1]);
00519       index++;
00520     }
00521     return values;
00522   }
00523
00531   private List<String> getTestFolders(String folder) {
00532     FileManager fm = new FileManager();
00533     List<String> testFolders = fm.getSubfolders(folder);
00534     for (int i = 0; i < testFolders.size(); i++) {
00535       String tempName = testFolders.get(i).toLowerCase();
00536       tempName = fm.getFilename(tempName);
00537       if (tempName.equals("control")) {
00538         testFolders.remove(i);
00539       }
00540     }
00541     return testFolders;
00542   }
00543
00551   private Map<String, Trial> getTrials(List<String> filenames) {
00552     // Extract Trials
00553     TrialManager tm = new TrialManager();
00554     Map<String, Trial> trials = new HashMap<String, Trial>();
00555     for (int i = 0; i < filenames.size(); i++) {
00556       String file = filenames.get(i);
00557       Trial trial = tm.getTrialFromAVI(file);
00558       if (trial != null)
00559         trials.put(trial.type + "-_-" + trial.ID, trial); // Expression "-_-" is
00560                                                           // just a separator
00561     }
00562     return trials;
00563   }
00564
00574   private ResultsTable indexesAnalysis(Map<String, Trial> controls, Map<String, Trial> tests
      ) {
00575
00576     Set<String> ckeySet = controls.keySet();
00577     Set<String> tkeySet = tests.keySet();
00578     ResultsTable rt = new ResultsTable();
00579     for (String k : ckeySet) {
00580       Trial trial = (Trial) controls.get(k);
00581       float chIdx = calculateChIndex(trial.tracks);
00582       float slIdx = calculateSLIndex(trial.tracks);
00583       setIndexesResults(rt, trial, chIdx, slIdx);
00584     }
00585     for (String k : tkeySet) {
00586       Trial trial = (Trial) tests.get(k);
00587       float chIdx = calculateChIndex(trial.tracks);
00588       float slIdx = calculateSLIndex(trial.tracks);
00589       setIndexesResults(rt, trial, chIdx, slIdx);
00590     }
00591     return rt;
00592   }
00593
00602   private SerializableList mergeTracks(Map<String, Trial> trials) {
00603
00604     SerializableList tracks = new SerializableList();
00605     for (String k : trials.keySet()) {
00606       Trial trial = (Trial) trials.get(k);
00607       tracks.addAll(trial.tracks);
00608     }
00609     return tracks;
00610   }
00611
00621   private int minSampleSize(Map<String, Trial> trials) {
00622     int minimum = 999999999;
00623     for (String k : trials.keySet()) {
00624       Trial t = (Trial) trials.get(k);
00625       int[] instantAngles = countAngles(t.tracks);
00626       int sampleSize = instantAngles[0] + instantAngles[1];
00627       if (sampleSize < minimum) {
00628         minimum = sampleSize;
00629       }
00630     }
00631     return minimum;
```

```
00632   }
00633
00643   private double or(Trial control, Trial test) {
00644     SerializableList controlTracks = control.tracks;
00645     SerializableList conditionTracks = test.tracks;
00646     double[] numeratorValues = getOddsValues(conditionTracks);// Calculate
00647                                                               // numerator's
00648                                                               // odds value
00649     double[] denominatorValues = getOddsValues(controlTracks);// Calculate
00650                                                               // denominator's
00651                                                               // odds value
00652     double numeratorRatio = numeratorValues[0] / numeratorValues[1];
00653     double denominatorRatio = denominatorValues[0] / denominatorValues[1];
00654     double oddsRatio = numeratorRatio / denominatorRatio;
00655     return oddsRatio;
00656   }
00657
00666   private double orThreshold(Map<String, Trial> controls) {
00667
00668     SerializableList controlTracks = mergeTracks(controls);
00669     List<Double> oRs = new ArrayList<Double>();
00670     for (int i = 0; i < Params.NUMSAMPLES; i++) {
00671       IJ.showProgress((double) i / Params.NUMSAMPLES);
00672       IJ.showStatus("Calculating Control Threshold. Shuffle " + i);
00673       Collections.shuffle(controlTracks);
00674       double[] numeratorValues = getOddsValues(controlTracks);// Calculate
00675                                                               // numerator's
00676                                                               // odds value
00677       Collections.shuffle(controlTracks);
00678       double[] denominatorValues = getOddsValues(controlTracks);// Calculate
00679                                                                 // denominator's
00680                                                                 // odds value
00681       double numeratorRatio = numeratorValues[0] / numeratorValues[1];
00682       double denominatorRatio = denominatorValues[0] / denominatorValues[1];
00683       double oddsRatio = numeratorRatio / denominatorRatio;
00684       oRs.add(oddsRatio);
00685       // IJ.log(""+oddsRatio);
00686     }
00687     Collections.sort(oRs);
00688     return oRs.get((int) (Params.NUMSAMPLES * 0.95));
00689   }
00690
00701   private double relativeAngle(Spermatozoon previous,
      Spermatozoon next) { // With
00702                                                                       // gradient
00703                                                                       // direction
00704     double angleDirection = (2 * Math.PI + Params.angleDirection * Math.PI / 180) % (2
      * Math.PI);
00705     float diffX = next.x - previous.x;
00706     float diffY = next.y - previous.y;
00707     double angle = (2 * Math.PI + Math.atan2(diffY, diffX)) % (2 * Math.PI); // Absolute
00708                                                                               // angle
00709     angle = (2 * Math.PI + angle - angleDirection) % (2 * Math.PI); // Relative
00710                                                                     // angle
00711                                                                     // between
00712                                                                     // interval
00713                                                                     // [0,2*Pi]
00714     if (angle > Math.PI) {
00715       angle = -(2 * Math.PI - angle);
00716     }
00717     return angle; // Between [-PI,PI]
00718   }
00719
00724   public void selectAnalysis() {
00725     // Ask if user wants to analyze a file or directory
00726     Object[] options = { "File", "Directory", " Multiple Simulations" };
00727     String question = "What do you want to analyze?";
00728     String title = "Choose one analysis...";
00729     final int FILE = 0;
00730     final int DIR = 1;
00731     final int SIMULATION = 2;
00732     Utils utils = new Utils();
00733     int sourceSelection = utils.analysisSelectionDialog(options, question, title);
00734     if (sourceSelection < 0) {
00735       return;
00736     } else if (sourceSelection == FILE) {// File
00737       analysis = TypeOfAnalysis.INDEXESFILE; // It's not possible to carry on
00738                                              // bootstrapping analysis in a
00739                                              // single file
00740     } else if (sourceSelection == DIR || sourceSelection == SIMULATION) {// Directory
00741                                                                          // or
00742                                                                          // simulations
00743       // Ask user which analysis wants to apply
00744       Object[] options2 = { "Ch-Index", "BOOTSTRAPPING" };
00745       question = "Which analysis do you want to apply to the data?";
00746       title = "Choose one analysis...";
00747       int analysisSelection = utils.analysisSelectionDialog(options2, question,
```

```
      title);
00748       final int CHINDEX = 0;
00749       final int BOOTSTRAPPING = 1;
00750       if (analysisSelection < 0)
00751         return;
00752       if (sourceSelection == DIR) {
00753         if (analysisSelection == CHINDEX) {
00754           analysis = TypeOfAnalysis.INDEXESDIRECTORY;
00755         } else if (analysisSelection == BOOTSTRAPPING) {
00756           analysis = TypeOfAnalysis.BOOTSTRAPPING;
00757         }
00758       } else if (sourceSelection == SIMULATION) { // Simulations
00759         if (analysisSelection == CHINDEX) {
00760           analysis = TypeOfAnalysis.INDEXESSIMULATIONS;
00761         } else if (analysisSelection == BOOTSTRAPPING) {
00762           analysis = TypeOfAnalysis.BOOTSTRAPPINGSIMULATIONS;
00763         }
00764       }
00765     }
00766   }
00767
00781   private void setBootstrappingResults(ResultsTable rt, double
      or, double th, Trial trial) {
00782     rt.incrementCounter();
00783     rt.addValue("ID", trial.ID);
00784     rt.addValue("OR", or);
00785     rt.addValue("Threshold", th);
00786     if (or > (th)) {
00787       rt.addValue("Result", "POSITIVE");
00788     } else {
00789       rt.addValue("Result", "-");
00790     }
00791     rt.addValue("Type", trial.type);
00792     rt.addValue("Source", trial.source);
00793     if (Params.compareOppositeDirections)
00794       rt.addValue("COD", "YES");
00795     else
00796       rt.addValue("COD", "NO");
00797     if (!Params.male.isEmpty())
00798       rt.addValue("Male", Params.male);
00799     if (!Params.date.isEmpty())
00800       rt.addValue("Date", Params.date);
00801     if (!Params.genericField.isEmpty())
00802       rt.addValue("Generic Field", Params.genericField);
00803   }
00804
00818   private void setIndexesResults(ResultsTable rt, Trial trial, float chIdx, float
      slIdx) {
00819     int nTracks = trial.tracks.size();
00820     rt.incrementCounter();
00821     rt.addValue("nTracks", nTracks);
00822     rt.addValue("Ch-Index", chIdx);
00823     rt.addValue("Sl-Index", slIdx);
00824     rt.addValue("Type", trial.type);
00825     rt.addValue("Direction (Degrees)", Params.angleDirection);
00826     rt.addValue("ArcChemotaxis (Degrees)", Params.angleAmplitude);
00827     rt.addValue("ID", trial.ID);
00828     rt.addValue("Source", trial.source);
00829     if (Params.compareOppositeDirections)
00830       rt.addValue("COD", "YES");
00831     else
00832       rt.addValue("COD", "NO");
00833     if (!Params.male.isEmpty())
00834       rt.addValue("Male", Params.male);
00835     if (!Params.date.isEmpty())
00836       rt.addValue("Date", Params.date);
00837     if (!Params.genericField.isEmpty())
00838       rt.addValue("Generic Field", Params.genericField);
00839   }
00840 }
```

7.3 ComputerVision.java File Reference

Classes

- class ComputerVision

Packages

- package functions

7.4 ComputerVision.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package functions;
00020
00021 import data.Spermatozoon;
00022 import ij.IJ;
00023 import ij.ImagePlus;
00024 import ij.ImageStack;
00025 import ij.plugin.ChannelSplitter;
00026 import ij.process.AutoThresholder;
00027 import ij.process.BinaryProcessor;
00028 import ij.process.ByteProcessor;
00029 import ij.process.ImageConverter;
00030 import ij.process.ImageProcessor;
00031 import ij.process.ImageStatistics;
00032
00037 public class ComputerVision {
00038
00039   /*****************************************************/
00040
00045   public double autoThresholdImagePlus(ImagePlus imp) {
00046     return autoThresholdImagePlus(imp, "Otsu"); // Otsu as a default
00047                                                 // thresholding method
00048   }
00049
00050   /*****************************************************/
00056   public double autoThresholdImagePlus(ImagePlus imp, String thresholdMethod) {
00057     ImageProcessor ip = imp.getProcessor();
00058     double lowerThreshold = 0;
00059     ImageStatistics st = ip.getStatistics();
00060     long[] histlong = st.getHistogram();
00061     Utils utils = new Utils();
00062     int histogram[] = utils.convertLongArrayToInt(histlong);
00063     AutoThresholder at = new AutoThresholder();
00064     lowerThreshold = (double) at.getThreshold(thresholdMethod, histogram);
00065     // Upper threshold set to maximum
00066     double upperThreshold = 255;
00067     // Threshold image processor
00068     thresholdImageProcessor(ip, lowerThreshold, upperThreshold);
00069     return lowerThreshold;
00070   }
00071
00072   /*****************************************************/
00079   public void convertToGrayscale(ImagePlus imp) {
00080     ImageConverter ic = new ImageConverter(imp);
00081     ic.convertToGray8();
00082   }
00083
00084   /*****************************************************/
00091   public void convertToRGB(ImagePlus imp) {
00092     ImageConverter ic = new ImageConverter(imp);
00093     ic.convertToRGB();
00094   }
00095
00096
00097
00098   /*****************************************************/
00103   public ImagePlus getBlueChannel(ImagePlus impColor) {
00104     ImagePlus[] images = ChannelSplitter.split(impColor);
00105     return images[2];
00106   }
00107
00108   /*****************************************************/
00113   public  ImagePlus getGreenChannel(ImagePlus impColor) {
00114     ImagePlus[] images = ChannelSplitter.split(impColor);
00115     return images[1];
00116   }
00117
```

```
00118   /*****************************************************/
00125   public  float getMeanGrayValue(Spermatozoon part, ImagePlus impGray,
      ImagePlus impTh) {
00126
00127     ImageProcessor ipTh = impTh.getProcessor();
00128     ImageProcessor ipGray = impGray.getProcessor();
00129     int bx = (int) part.bx;
00130     int by = (int) part.by;
00131     int width = (int) part.width;
00132     int height = (int) part.height;
00133     float totalGray = 0;
00134     float totalPixels = 0;
00135     for (int x = bx; x < (width + bx); x++) {
00136       IJ.showStatus("scanning pixels...");
00137       for (int y = by; y < (height + by); y++) {
00138         int pixel = ipTh.get(x, y);
00139         if (pixel == 0) {
00140           totalGray += (float) ipGray.get(x, y);
00141           totalPixels++;
00142         }
00143       }
00144     }
00145     return totalGray / totalPixels;
00146   }
00147
00148   /*****************************************************/
00153   public ImagePlus getRedChannel(ImagePlus impColor) {
00154     ImagePlus[] images = ChannelSplitter.split(impColor);
00155     return images[0];
00156   }
00157
00158   /*****************************************************
00159    *
00163   public void outlineThresholdImage(ImagePlus imp) {
00164     convertToGrayscale(imp);
00165     ImageProcessor ip = imp.getProcessor();
00166     BinaryProcessor bp = new BinaryProcessor((ByteProcessor) ip);
00167     bp.outline();
00168   }
00169
00170   /*****************************************************/
00175   public void thresholdImagePlus(ImagePlus imp, double lowerThreshold) {
00176     ImageProcessor ip = imp.getProcessor();
00177     // Upper threshold set to maximum
00178     double upperThreshold = 255;
00179     // Threshold image processor
00180     thresholdImageProcessor(ip, lowerThreshold, upperThreshold);
00181   }
00182
00183   /*****************************************************/
00189   public void thresholdImageProcessor(ImageProcessor ip, double lowerThreshold,
      double upperThreshold) {
00190     // Make binary
00191     int[] lut = new int[256];
00192     for (int j = 0; j < 256; j++) {
00193       if (j >= lowerThreshold && j <= upperThreshold)
00194         lut[j] = (byte) 0;
00195       else
00196         lut[j] = (byte) 255;
00197     }
00198     ip.applyTable(lut);
00199   }
00200
00201   /*****************************************************/
00206   public void thresholdStack(ImagePlus imp) {
00207
00208     ImageStack stack = imp.getStack();
00209     ImageProcessor ip = stack.getProcessor(1);
00210     ImageStatistics st = ip.getStatistics();
00211     double mean = st.mean;
00212     double std = st.stdDev;
00213     // Set threshold as mean + 2 x standard deviation
00214     double lowerThreshold = mean + 2 * std; // std factor: candidate to be a
00215                                             // parameter of the plugin
00216     double upperThreshold = 255;
00217     // Make binary
00218     int[] lut = new int[256];
00219     for (int j = 0; j < 256; j++) {
00220       if (j >= lowerThreshold && j <= upperThreshold)
00221         lut[j] = 0;
00222       else
00223         lut[j] = (byte) 255;
00224     }
00225     int nFrames = imp.getStackSize();
00226     for (int iFrame = 1; iFrame <= nFrames; iFrame++) {
00227       ip = stack.getProcessor(iFrame);
00228       ip.applyTable(lut);
```

```
00229     }
00230   }
00231 }
```

7.5 FileManager.java File Reference

Classes

- class FileManager

Packages

- package functions

7.6 FileManager.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package functions;
00020
00021 import java.io.File;
00022 import java.util.ArrayList;
00023 import java.util.List;
00024
00025 import javax.swing.JFileChooser;
00026 import javax.swing.JOptionPane;
00027
00028 import ij.IJ;
00029 import ij.ImagePlus;
00030 import ij.io.DirectoryChooser;
00031 import third_Party.AVI_Reader;
00032
00037 public class FileManager {
00038
00040   public FileManager() {
00041   }
00047   public ImagePlus getAVI(String path){
00048     AVI_Reader ar = new AVI_Reader();
00049     ar.run(path);
00050     ImagePlus imp = ar.getImagePlus();
00051     return imp;
00052   }
00057   public String getFilename(String path) {
00058     String[] parts = path.split("\\\\");
00059     return removeExtension(parts[parts.length - 1]);
00060   }
00061
00066   public String[] getContent(String path) {
00067     if(path==null)
00068       return null;
00069     File folder = new File(path);
00070     File[] listOfFiles = folder.listFiles();
00071     if(listOfFiles.length<=0)
00072       return null;
```

```
00073     String[] listOfNames = new String[listOfFiles.length];
00074     for (int i = 0; i < listOfFiles.length; i++)
00075       listOfNames[i] = folder.getAbsolutePath() + "\\" + listOfFiles[i].getName();
00076     return listOfNames;
00077   }
00078
00079   public List<String> getFiles(String path){
00080     String[] content = getContent(path);
00081     List<String> files = new ArrayList<String>();
00082     for(int i=0;i<content.length;i++){
00083       if(new File(content[i]).isFile()){
00084         files.add(content[i]);
00085       }
00086     }
00087     return files;
00088   }
00089
00090   public List<String> getSubfolders(String path){
00091
00092     String[] content = getContent(path);
00093     List<String> subfolders = new ArrayList<String>();
00094     for(int i=0;i<content.length;i++){
00095       if(new File(content[i]).isDirectory()){
00096         subfolders.add(content[i]);
00097       }
00098     }
00099     return subfolders;
00100   }
00105   public String getParentDirectory(String path) {
00106     String[] parts = path.split("\\\\");
00107     return parts[parts.length - 2];
00108   }
00109
00114   public boolean isAVI(String filename) {
00115     String[] parts = filename.split("\\.");
00116     if (parts[1].equals("avi"))
00117       return true;
00118     else
00119       return false;
00120   }
00125   public List<ImagePlus> loadImageDirectory() {
00126     String dir = selectFolder();
00127     return loadImageDirectory(dir);
00128   }
00129
00134   public List<ImagePlus> loadImageDirectory(String dir) {
00135
00136     String[] listOfFiles = getContent(dir);
00137     if (listOfFiles == null || listOfFiles.length == 0) {
00138       if (listOfFiles != null)
00139         JOptionPane.showMessageDialog(null, "Please, select a non-empty folder.");
00140       return null;
00141     }
00142     List<ImagePlus> images = new ArrayList<ImagePlus>();
00143     for (int i = 0; i < listOfFiles.length; i++) {
00144       IJ.showProgress((double) i / listOfFiles.length);
00145       IJ.showStatus("Loading image "+ i +"...");
00146       String absoluteFilePath = listOfFiles[i];
00147       if(isAVI(absoluteFilePath))
00148         continue;
00149       String parentsDirectory = getParentDirectory(absoluteFilePath);
00150       ImagePlus imp = IJ.openImage(absoluteFilePath);
00151       if (imp != null) {
00152         imp.setTitle(parentsDirectory + "\\" + imp.getTitle());
00153         images.add(imp);
00154       }
00155       // else - possibly the file is not an image nor AVI
00156     }
00157     if (images.size() < 1) {
00158       JOptionPane.showMessageDialog(null, "Please, select a valid folder.");
00159       return null;
00160     }
00161     return images;
00162   }
00168   public List<ImagePlus> loadImageFile() {
00169     String absoluteFilePath = selectFile();
00170     if (absoluteFilePath == null) {
00171       return null;
00172     }
00173     if(isAVI(absoluteFilePath)){
00174       JOptionPane.showMessageDialog(null, "Please, select a valid file.");
00175       return null;
00176     }
00177     String parentsDirectory = getParentDirectory(absoluteFilePath);
00178     ImagePlus imp = IJ.openImage(absoluteFilePath);
00179     if (imp == null) {
00180       JOptionPane.showMessageDialog(null, "Please, select a valid file.");
```

```
00181       return null;
00182     }
00183     imp.setTitle(parentsDirectory + "\\" + imp.getTitle());
00184     List<ImagePlus> images = new ArrayList<ImagePlus>();
00185     images.add(imp);
00186     return images;
00187   }
00188
00189   public String removeExtension(String filename){
00190     String[] parts = filename.split("\\.");
00191     return parts[0];
00192   }
00193 //  /**
00194 //   *
00195 //   * @return
00196 //   */
00197 //  public List<ImagePlus> loadImages() {
00198 //    // Ask user which analysis wants to apply
00199 //    int userSelection = dialog();
00200 //    if (userSelection < 0)
00201 //      return null;
00202 //    if (userSelection == 0)
00203 //      return loadImageFile();
00204 //    else if (userSelection == 1)
00205 //      return loadImageDirectory();
00206 //    else
00207 //      return null;
00208 //  }
00212   public String selectFile() {
00213     JFileChooser chooser = new JFileChooser();
00214     chooser.setDialogTitle("Select a file...");
00215     chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
00216     chooser.setAcceptAllFileFilterUsed(false);
00217     if (chooser.showOpenDialog(null) == JFileChooser.APPROVE_OPTION) {
00218       File file = chooser.getSelectedFile();
00219       return file.getAbsolutePath();
00220     }
00221     return null;
00222   }
00223
00227   public String selectFolder() {
00228     DirectoryChooser dc = new DirectoryChooser("Select folder...");
00229     return dc.getDirectory();
00230   }
00231 }
```

7.7 ImageAnalysisWindow.java File Reference

Classes

- class ImageAnalysisWindow
- enum ImageAnalysisWindow.TypeOfAnalysis

Packages

- package gui

7.8 ImageAnalysisWindow.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
```

```
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package gui;
00020
00021 import java.awt.Color;
00022 import java.awt.Dimension;
00023 import java.awt.GridBagConstraints;
00024 import java.awt.GridBagLayout;
00025 import java.awt.Toolkit;
00026 import java.awt.event.ActionEvent;
00027 import java.awt.event.ActionListener;
00028 import java.awt.event.MouseListener;
00029 import java.util.ArrayList;
00030 import java.util.List;
00031 import java.util.ListIterator;
00032
00033 import javax.swing.ButtonGroup;
00034 import javax.swing.ImageIcon;
00035 import javax.swing.JButton;
00036 import javax.swing.JFrame;
00037 import javax.swing.JLabel;
00038 import javax.swing.JPanel;
00039 import javax.swing.JRadioButton;
00040 import javax.swing.JSeparator;
00041 import javax.swing.JSlider;
00042 import javax.swing.SwingConstants;
00043 import javax.swing.event.ChangeListener;
00044
00045 import data.Spermatozoon;
00046 import functions.ComputerVision;
00047 import functions.FileManager;
00048 import functions.Utils;
00049 import ij.ImagePlus;
00050 import ij.process.ImageProcessor;
00051
00057 public class ImageAnalysisWindow extends JFrame {
00058
00059   private enum TypeOfAnalysis {
00060     DIRECTORY, FILE, NONE
00061   }
00062
00063   private TypeOfAnalysis analysis = TypeOfAnalysis.
      NONE;
00064
00066   private List<ImagePlus>      images;
00067   private int                  imgIndex;
00068   private JLabel               imgLabel;
00070   protected ImagePlus          impDraw         = null;
00072   protected ImagePlus          impGray         = null;
00074   protected ImagePlus          impOrig         = null;
00076   protected ImagePlus          impOutline      = null;
00078   protected ImagePlus          impTh           = null;
00080   private double               resizeFactor;
00081   protected JSlider            sldThreshold;
00082   protected JSlider            sldRedThreshold;
00083   protected JSlider            sldGreenThreshold;
00084   protected JSlider            sldBlueThreshold;
00085   protected JRadioButton btnOtsu;
00086   protected JRadioButton btnMinimum;
00087   protected ButtonGroup btnGroup;
00088   protected JButton prevBtn;
00089   protected JButton nextBtn;
00090
00091   protected List<Spermatozoon> spermatozoa     = new ArrayList<Spermatozoon>();
00092   protected double             threshold       = -1.0;
00093   protected double             redThreshold      = -1.0;
00094   protected double             greenThreshold      = -1.0;
00095   protected double             blueThreshold      = -1.0;
00096   protected String             thresholdMethod = "Otsu";
00097   private JLabel               title;
00099   protected double             xFactor;
00100
00102   protected double yFactor;
00103
00104   public ImageAnalysisWindow() {
00105     imgLabel = new JLabel();
00106     imgIndex = 0;
00107     //The size of the showed image will be set to 60% of the screen size
00108     resizeFactor = 0.6;
00109     //its necessary to initialize here the slider bar in order to enable
00110     //the change listener selection for an inherit class
00111     sldThreshold = new JSlider(JSlider.HORIZONTAL, 0, 255, 60);
```

```
00112     sldRedThreshold = new JSlider(JSlider.HORIZONTAL, 0, 255, 60);
00113     sldRedThreshold.setForeground(Color.RED);
00114     sldGreenThreshold = new JSlider(JSlider.HORIZONTAL, 0, 255, 60);
00115     sldGreenThreshold.setForeground(Color.GREEN);
00116     sldBlueThreshold = new JSlider(JSlider.HORIZONTAL, 0, 255, 60);
00117     sldBlueThreshold.setForeground(Color.BLUE);
00118     sldThreshold.setVisible(false); //By default
00119     sldRedThreshold.setVisible(false); //By default
00120     sldGreenThreshold.setVisible(false);//By default
00121     sldBlueThreshold.setVisible(false);//By default
00122     imgLabel = new JLabel();// The same as slider bar
00123     btnOtsu = new JRadioButton("Otsu");
00124     btnMinimum = new JRadioButton("Minimum");
00125     btnGroup = new ButtonGroup();
00126     prevBtn = new JButton("Previous");
00127     nextBtn = new JButton("Next");
00128   }
00129
00130   private int analyseDirectory() {
00131     FileManager fm = new FileManager();
00132     List<ImagePlus> images = fm.loadImageDirectory();
00133     if (images != null) {
00134       setImages(images);
00135       showWindow();
00136       return 0;
00137     } else {
00138       return -1;
00139     }
00140   }
00141
00142   private int analyseFile() {
00143     FileManager fm = new FileManager();
00144     List<ImagePlus> images = fm.loadImageFile();
00145     if (images != null) {
00146       setImages(images);
00147       showWindow();
00148       return 0;
00149     } else {
00150       return -1;
00151     }
00152   }
00153
00157   public void deselectAll() {
00158     for (ListIterator j = spermatozoa.listIterator(); j.hasNext();) {
00159       Spermatozoon spermatozoon = (Spermatozoon) j.next();
00160       spermatozoon.selected = false;
00161     }
00162   }
00163   protected void drawImage() {}
00168   public void idenfitySperm() {
00169     int SpermNr = 0;
00170     for (ListIterator<Spermatozoon> j = spermatozoa.listIterator(); j.hasNext();) {
00171       Spermatozoon sperm = (Spermatozoon) j.next();
00172       SpermNr++;
00173       sperm.id = "" + SpermNr;
00174     }
00175   }
00176
00180   public void initImage() {
00181     setImage(0); // Initialization with the first image available
00182     processImage(false);
00183     drawImage();
00184   }
00185
00186   protected void nextAction() {
00187   }
00188
00189   protected void previousAction() {
00190   }
00191
00192   protected void processImage(boolean eventType) {}
00193
00194   public void reset() {
00195     if (impOrig != null)
00196       impOrig.close();
00197     if (impDraw != null)
00198       impDraw.close();
00199     if (impGray != null)
00200       impGray.close();
00201     if (impTh != null)
00202       impTh.close();
00203     if (impOutline != null)
00204       impOutline.close();
00205     threshold = -1.0;
00206     spermatozoa.clear();
00207   }
00208
```

```
00209   public int run() {
00210     int out = selectAnalysis();
00211     if (out < 0)
00212       return out;
00213     switch (analysis) {
00214       case FILE:
00215         out = analyseFile();
00216         break;
00217       case DIRECTORY:
00218         out = analyseDirectory();
00219         break;
00220       default:
00221         out = -2;
00222         break;
00223     }
00224     return out;
00225   }
00226
00227   public void selectAll() {
00228     selectAll(spermatozoa);
00229   }
00230
00234   public void selectAll(List<Spermatozoon> sperm) {
00235     for (ListIterator j = sperm.listIterator(); j.hasNext();) {
00236       Spermatozoon spermatozoon = (Spermatozoon) j.next();
00237       spermatozoon.selected = true;
00238     }
00239   }
00240
00245   public int selectAnalysis() {
00246     // Ask if user wants to analyze a file or directory
00247     Object[] options = { "File", "Directory" };
00248     String question = "What do you want to analyze?";
00249     String title = "Choose one analysis...";
00250     final int FILE = 0;
00251     final int DIR = 1;
00252     final int MULTIDIR = 2;
00253     Utils utils = new Utils();
00254     int sourceSelection = utils.analysisSelectionDialog(options, question, title);
00255     if (sourceSelection < 0) {
00256       analysis = TypeOfAnalysis.NONE;
00257       return -1;
00258     } else if (sourceSelection == FILE) {
00259       analysis = TypeOfAnalysis.FILE;
00260     } else if (sourceSelection == DIR) {
00261       analysis = TypeOfAnalysis.DIRECTORY;
00262     }
00263     return 0;
00264   }
00265
00266   public void setChangeListener(ChangeListener ch,JSlider sld) {
00267     sld.addChangeListener(ch);
00268   }
00269
00270   /*****************************************************/
00274   public void setImage() {
00275     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
00276     double w = screenSize.getWidth();
00277     double h = screenSize.getHeight();
00278     int targetWidth = (int) (w * resizeFactor);
00279     int targetHeight = (int) (h * resizeFactor);
00280     ImageProcessor ip = impDraw.getProcessor();
00281     ip.setInterpolationMethod(ImageProcessor.BILINEAR);
00282     ip = ip.resize(targetWidth, targetHeight);
00283     impDraw.setProcessor(ip);
00284     imgLabel.setIcon(new ImageIcon(impDraw.getImage()));
00285     imgLabel.repaint();
00286   }
00287
00288   /*****************************************************/
00296   public void setImage(int index) {
00297     if (index < 0 || index >= images.size())
00298       return;
00299     impOrig = images.get(index).duplicate();
00300     impOrig.setTitle(images.get(index).getTitle());
00301     impDraw = impOrig.duplicate();
00302     title.setText(impOrig.getTitle());
00303     setImage();
00304     setResizeFactor();
00305   }
00306
00312   public void setImages(List<ImagePlus> i) {
00313     images = i;
00314   }
00315
00316   public void setMouseListener(MouseListener ml) {
00317     imgLabel.addMouseListener(ml);
```

```
00318   }
00319
00320   public void setRawImage() {
00321     setImage(imgIndex);
00322   }
00323
00328   public void setResizeFactor() {
00329     double origW = impOrig.getWidth();
00330     double origH = impOrig.getHeight();
00331     double resizeW = impDraw.getWidth();
00332     double resizeH = impDraw.getHeight();
00333     xFactor = origW / resizeW;
00334     yFactor = origH / resizeH;
00335   }
00336
00337   protected void genericRadioButtonsAction(){}
00338
00339   private void configureSliderBar(JSlider sld){
00340     sld.setMinorTickSpacing(2);
00341     sld.setMajorTickSpacing(10);
00342     sld.setPaintTicks(true);
00343     sld.setPaintLabels(true);
00344     // We'll just use the standard numeric labels for now...
00345     sld.setLabelTable(sld.createStandardLabels(10));
00346   }
00347   /*****************************************************/
00351   public void showWindow() {
00352
00353     JPanel panel = new JPanel(new GridBagLayout());
00354     GridBagConstraints c = new GridBagConstraints();
00355     c.fill = GridBagConstraints.HORIZONTAL;
00356
00357     // RADIO BUTTONS
00358     btnOtsu.setSelected(true);
00359     btnOtsu.addActionListener(new ActionListener() {
00360       public void actionPerformed(ActionEvent e) {
00361         threshold = -1.0;
00362         thresholdMethod = "Otsu";
00363         processImage(false);
00364         genericRadioButtonsAction();
00365       }
00366     });
00367
00368     btnMinimum.addActionListener(new ActionListener() {
00369       public void actionPerformed(ActionEvent e) {
00370         threshold = -1.0;
00371         thresholdMethod = "Minimum";
00372         processImage(false);
00373         genericRadioButtonsAction();
00374       }
00375     });
00376     // Group the radio buttons.
00377
00378     btnGroup.add(btnOtsu);
00379     btnGroup.add(btnMinimum);
00380     c.gridx = 0;
00381     c.gridy = 0;
00382     c.gridwidth = 1;
00383     panel.add(btnOtsu, c);
00384     c.gridy = 1;
00385     panel.add(btnMinimum, c);
00386     // THRESHOLD SLIDERBARS
00387     configureSliderBar(sldThreshold);
00388     configureSliderBar(sldRedThreshold);
00389     configureSliderBar(sldGreenThreshold);
00390     configureSliderBar(sldBlueThreshold);
00391
00392     c.fill = GridBagConstraints.HORIZONTAL;
00393     c.gridx = 1;
00394     c.gridy = 0;
00395     c.gridwidth = 10;
00396     c.gridheight = 2;
00397     c.ipady = 10;
00398     panel.add(sldRedThreshold, c);
00399     panel.add(sldThreshold, c); // this two sliders are mutually exclusives
00400     c.gridy = 2;
00401     panel.add(sldGreenThreshold, c);
00402     c.gridy = 4;
00403     panel.add(sldBlueThreshold, c);
00404
00405     c.gridx = 0;
00406     c.gridy = 6;
00407     c.gridwidth = 10;
00408     c.gridheight = 1;
00409     panel.add(new JSeparator(SwingConstants.HORIZONTAL), c);
00410
00411     title = new JLabel();
```

```
00412     c.gridx = 2;
00413     c.gridy = 7;
00414     c.gridwidth = 6;
00415     c.gridheight = 1;
00416     c.ipady = 10;
00417     panel.add(title, c);
00418
00419     c.gridx = 2;
00420     c.gridy = 8;
00421     c.gridwidth = 6;
00422     c.gridheight = 1;
00423     c.ipady = 10;
00424     panel.add(imgLabel, c);
00425     initImage(); // Initialization with the first image available
00426
00427     c.gridx = 0;
00428     c.gridy = 9;
00429     c.gridwidth = 10;
00430     c.gridheight = 1;
00431     panel.add(new JSeparator(SwingConstants.HORIZONTAL), c);
00432
00433     // Add action listener
00434     prevBtn.addActionListener(new ActionListener() {
00435       public void actionPerformed(ActionEvent e) {
00436         if (imgIndex > 0) {
00437           nextBtn.setEnabled(true);
00438           previousAction();
00439           reset();
00440           setImage(--imgIndex);
00441           processImage(false);
00442         }else if(imgIndex==0){
00443           previousAction();
00444           prevBtn.setEnabled(false);
00445         }
00446       }
00447     });
00448     c.gridx = 0;
00449     c.gridy = 10;
00450     c.gridwidth = 1;
00451     c.gridheight = 1;
00452     panel.add(prevBtn, c);
00453
00454     // Add action listener
00455     nextBtn.addActionListener(new ActionListener() {
00456       public void actionPerformed(ActionEvent e) {
00457         if (imgIndex < (images.size() - 1)) {
00458           prevBtn.setEnabled(true);
00459           nextAction();
00460           reset();
00461           setImage(++imgIndex);
00462           processImage(false);
00463         }else if(imgIndex==(images.size()-1)){
00464           nextAction();
00465           nextBtn.setEnabled(false);
00466         }
00467       }
00468     });
00469     c.gridx = 9;
00470     c.gridy = 10;
00471     panel.add(nextBtn, c);
00472
00473     this.setContentPane(panel);
00474     this.pack();
00475     this.setVisible(true);
00476   }
00477   private void setSlidersAutoThreshold(){
00478     redThreshold = threshold;
00479     greenThreshold = threshold;
00480     blueThreshold = threshold;
00481     sldThreshold.setValue((int) threshold);
00482     sldRedThreshold.setValue((int) threshold);
00483     sldGreenThreshold.setValue((int) threshold);
00484     sldBlueThreshold.setValue((int) threshold);
00485   }
00486   /*****************************************************/
00494   public void thresholdImagePlus(ImagePlus imp) {
00495     ComputerVision cv = new ComputerVision();
00496     if (threshold == -1) {
00497       threshold = cv.autoThresholdImagePlus(imp, thresholdMethod);
00498       setSlidersAutoThreshold();
00499     } else {
00500       cv.thresholdImagePlus(imp, threshold);
00501     }
00502   }
00503
00504 }
```

7.9 Kinematics.java File Reference

Classes

- class Kinematics

Packages

- package functions

7.10 Kinematics.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package functions;
00020
00021 import java.util.ArrayList;
00022 import java.util.List;
00023 import java.util.ListIterator;
00024
00025 import data.Params;
00026 import data.SerializableList;
00027 import data.Spermatozoon;
00028
00033 public class Kinematics {
00034   /*****************************************************/
00042   public float[] alh(List track, List avgTrack) {
00043
00044     int length = avgTrack.size();
00045     float alh[] = new float[2];
00046     float alhMax = 0;
00047     float alhMean = 0;
00048     for (int i = 0; i < length; i++) {
00049       Spermatozoon origSpermatozoon = (Spermatozoon) track.get(i +
      Params.wSize / 2 - 1);
00050       Spermatozoon avgSpermatozoon = (Spermatozoon) avgTrack.get(i);
00051       float distance = origSpermatozoon.distance(avgSpermatozoon);
00052       alhMean += distance;
00053       if (distance > alhMax)
00054         alhMax = distance;
00055     }
00056     // Mean value
00057     alhMean = alhMean / length;
00058     // convert pixels to micrometers
00059     alh[0] = alhMean * (float) Params.micronPerPixel;
00060     alh[1] = alhMax * (float) Params.micronPerPixel;
00061
00062     return alh;
00063   }
00064
00065   /*****************************************************/
00066
00074   // public float bcf(List track,List avgTrack){
00075   //
00076   // int length = avgTrack.size();
00077   // int intersections=0;
00078   // // bcf_shift equal to 1 is not enougth to catch all beat-cross
00079   // for (int i=1;i<length;i=i+1+Params.bcf_shift){
```

```
00080   // Spermatozoon origP0 =
00081   // (Spermatozoon)track.get(i-Params.bcf_shift+Params.wSize/2-1);
00082   // Spermatozoon origP1 = (Spermatozoon)track.get(i+Params.wSize/2-1);
00083   // Spermatozoon avgP0 = (Spermatozoon)avgTrack.get(i-Params.bcf_shift);
00084   // Spermatozoon avgP1 = (Spermatozoon)avgTrack.get(i);
00085   // Line2D origLine = new Line2D.Float();
00086   // origLine.setLine(origP0.x,origP0.y,origP1.x,origP1.y);
00087   // Line2D avgLine = new Line2D.Float();
00088   // avgLine.setLine(avgP0.x,avgP0.y,avgP1.x,avgP1.y);
00089   //
00090   // boolean intersection = origLine.intersectsLine(avgLine);
00091   // if(intersection)
00092   // intersections++;
00093   // }
00094   // float bcf_value = (float)intersections*Params.frameRate/(float)length;
00095   //
00096   // return bcf_value;
00097   // }
00103   public float bcf(List track, List avgTrack) {
00104
00105     int nAvgPoints = avgTrack.size();
00106     float[] dists = new float[nAvgPoints];
00107     int[] xPoints = new int[nAvgPoints];
00108     for (int i = 0; i < nAvgPoints; i++) {
00109       Spermatozoon origS = (Spermatozoon) track.get(i +
      Params.wSize / 2 - 1);
00110       Spermatozoon avgS = (Spermatozoon) avgTrack.get(i);
00111       dists[i] = origS.distance(avgS);
00112     }
00113     SignalProcessing sp = new SignalProcessing();
00114     dists = sp.movingAverage(dists, 2);
00115     int intersections = countLocalMaximas(dists);
00116     float bcf_value = (float) intersections * Params.frameRate / (float) nAvgPoints;
00117     return bcf_value;
00118   }
00119
00124   int countLocalMaximas(float[] points) {
00125     int nPoints = points.length;
00126     int count = 0;
00127     for (int i = 2; i < nPoints; i++) {
00128       float x0 = points[i - 2];
00129       float x1 = points[i - 1];
00130       float x2 = points[i];
00131       if (((x1 > x0) & (x1 > x2)) || ((x1 < x0) & (x1 < x2)))
00132         count++;
00133     }
00134     return count;
00135   }
00136
00137   /*****************************************************/
00142   public String getVelocityTrackType(List track) {
00143
00144     SignalProcessing sp = new SignalProcessing();
00145     List avgTrack = sp.movingAverage(track);
00146     float vap = vcl(avgTrack);
00147     if ((vsl(track) < Params.vclLowerTh) || (vcl(track) <
      Params.vclLowerTh) || (vap < Params.vclLowerTh))
00148       return "Slow";
00149     else if ((vsl(track) > Params.vclUpperTh) || (vcl(track) >
      Params.vclUpperTh) || (vap > Params.vclUpperTh))
00150       return "Fast";
00151     else
00152       return "Normal";
00153   }
00154
00155   /*****************************************************/
00161   public float mad(List track) {
00162
00163     int length = track.size();
00164     ListIterator jT = track.listIterator();
00165     Spermatozoon oldSpermatozoon = (Spermatozoon) jT.next();
00166     float totalDegrees = 0;
00167     for (int i = 1; i < length; i++) {
00168       Spermatozoon newSpermatozoon = (Spermatozoon) track.get(i);
00169       float diffX = newSpermatozoon.x - oldSpermatozoon.x;
00170       float diffY = newSpermatozoon.y - oldSpermatozoon.y;
00171       double angle = (2 * Math.PI + Math.atan2(diffY, diffX)) % (2 * Math.PI);
00172       totalDegrees += angle;
00173       oldSpermatozoon = newSpermatozoon;
00174     }
00175     // mean angle
00176     float meanAngle = totalDegrees / (length - 1);
00177     return meanAngle;
00178   }
00179
00180   /*****************************************************/
00185   public boolean motilityTest(List track) {
```

```
00186
00187     Kinematics K = new Kinematics();
00188     int nPoints = track.size();
00189     Spermatozoon firstSpermatozoon = (Spermatozoon) track.get(0);
00190     Spermatozoon lastSpermatozoon = (Spermatozoon) track.get(nPoints - 1);
00191     float distance = lastSpermatozoon.distance(firstSpermatozoon);
00192     SignalProcessing sp = new SignalProcessing();
00193     List avgTrack = sp.movingAverage(track);
00194     float vap = K.vcl(avgTrack) / K.vsl(avgTrack);
00195     // Kinematics filter
00196     double minPixelDistance = 10;// 10/Params.micronPerPixel;
00197     if (K.vcl(track) > Params.vclMin && (distance > minPixelDistance) && (vap > 0))
00198       return true;
00199     else
00200       return false;
00201   }
00202
00203   /*****************************************************/
00208   public int[] motilityTest(SerializableList theTracks) {
00209     int motile = 0;
00210     int nonMotile = 0;
00211     for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
00212       List aTrack = (ArrayList) iT.next();
00213       if (motilityTest(aTrack))
00214         motile++;
00215       else
00216         nonMotile++;
00217     }
00218     int[] results = new int[2];
00219     results[0] = motile;
00220     results[1] = nonMotile;
00221     return results;
00222   }
00223
00224   /*****************************************************/
00230   public float vcl(List track) {
00231
00232     int length = track.size();
00233     ListIterator jT = track.listIterator();
00234     Spermatozoon oldSpermatozoon = (Spermatozoon) jT.next();
00235     float distance = 0;
00236     for (; jT.hasNext();) {
00237       Spermatozoon newSpermatozoon = (Spermatozoon) jT.next();
00238       distance += newSpermatozoon.distance(oldSpermatozoon);
00239       oldSpermatozoon = newSpermatozoon;
00240     }
00241     // convert pixels to micrometers
00242     distance = distance * (float) Params.micronPerPixel;
00243     // Seconds
00244     float elapsedTime = (length - 1) / Params.frameRate;
00245     // return um/second
00246     return distance / elapsedTime;
00247   }
00248
00249   /*****************************************************/
00255   public float vsl(List track) {
00256
00257     int length = track.size();
00258     Spermatozoon first = (Spermatozoon) track.get(0);
00259     Spermatozoon last = (Spermatozoon) track.get(length - 1);
00260     // Distance (pixels)
00261     float distance = last.distance(first);
00262     // convert pixels to micrometers
00263     distance = distance * (float) Params.micronPerPixel;
00264     // Seconds
00265     float elapsedTime = (length - 1) / Params.frameRate;
00266     // return um/second
00267     return distance / elapsedTime;
00268   }
00269
00270 }
```

7.11 MainWindow.java File Reference

Classes

- class MainWindow

Packages

- package gui

7.12 MainWindow.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package gui;
00020
00021 import java.awt.Color;
00022 import java.awt.Dimension;
00023 import java.awt.GridBagConstraints;
00024 import java.awt.GridBagLayout;
00025 import java.awt.HeadlessException;
00026 import java.awt.Image;
00027 import java.awt.event.ActionEvent;
00028 import java.awt.event.ActionListener;
00029
00030 import javax.imageio.ImageIO;
00031 import javax.swing.ImageIcon;
00032 import javax.swing.JButton;
00033 import javax.swing.JFrame;
00034 import javax.swing.JPanel;
00035
00036 import analysis.Chemotaxis;
00037 import analysis.Motility;
00038 import data.Params;
00039 import data.PersistentRandomWalker;
00040 import data.Simulation;
00041 import ij.IJ;
00042 import ij.gui.GenericDialog;
00043
00048 public class MainWindow extends JFrame {
00049
00053   private static final long serialVersionUID = 1L;
00055   private MainWindow mw;
00056
00061   public MainWindow(String title) throws HeadlessException {
00062     super(title);
00063     createGUI();
00064     this.setPreferredSize(new Dimension(600, 300));
00065     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
00066     this.pack();
00067     this.setVisible(true);
00068     this.setLocationRelativeTo(null);
00069     mw = this;
00070     Params.resetParams();
00071   }
00072
00082   private void addButton(final String label, int gridx, int gridy, Color background, String
      iconPath, JPanel panel) {
00083
00084     GridBagConstraints c = new GridBagConstraints();
00085     c.weightx = 0.5;
00086     c.fill = GridBagConstraints.HORIZONTAL;
00087     c.ipady = 0;
00088     c.gridx = gridx;
00089     c.gridy = gridy;
00090     JButton btn = new JButton(label);
00091     btn.setBackground(background);
00092     try {
00093       // Image img =
00094       // ImageIO.read(getClass().getResource("/resources/motility.png"));
00095       Image img = ImageIO.read(getClass().getResource(iconPath));
```

```
00096       btn.setIcon(new ImageIcon(img));
00097     } catch (Exception ex) {
00098       IJ.handleException(ex);
00099 //      System.out.println(ex);
00100     }
00101     // Add action listener
00102     btn.addActionListener(new ActionListener() {
00103       public void actionPerformed(ActionEvent e) {
00104         if (label.equals("Chemotaxis")) {
00105           Chemotaxis ch = new Chemotaxis();
00106           try {
00107             mw.setVisible(false);
00108             ch.selectAnalysis();
00109             ch.execute();
00110             mw.setVisible(true);
00111           } catch (Exception e1) {
00112             IJ.handleException(e1);
00113           }
00114         } else if (label.equals("Motility")) {
00115           Motility mot = new Motility();
00116           try {
00117             mw.setVisible(false);
00118             mot.selectAnalysis();
00119             mot.execute();
00120             mw.setVisible(true);
00121           } catch (Exception e1) {
00122             IJ.handleException(e1);
00123           }
00124         } else if (label.equals("Viability")) {
00125           mw.setVisible(false);
00126           ViabilityWindow viabilityW = new ViabilityWindow();
00127           viabilityW.addWindowListener(new java.awt.event.WindowAdapter() {
00128             @Override
00129             public void windowClosing(java.awt.event.WindowEvent windowEvent) {
00130               if (mw != null)
00131                 mw.setVisible(true);
00132             }
00133           });
00134           int out = viabilityW.run();
00135           if(out<0){
00136             mw.setVisible(true);
00137           }
00138         } else if (label.equals("Morphometry")) {
00139           mw.setVisible(false);
00140           MorphWindow morphW = new MorphWindow();
00141           morphW.addWindowListener(new java.awt.event.WindowAdapter() {
00142             @Override
00143             public void windowClosing(java.awt.event.WindowEvent windowEvent) {
00144               if (mw != null)
00145                 mw.setVisible(true);
00146             }
00147           });
00148           int out = morphW.run();
00149           if(out<0){
00150             mw.setVisible(true);
00151           }
00152         } else if (label.equals("Simulation")) {
00153           simulate();
00154         } else if (label.equals("Settings")) {
00155           SettingsWindow sw = new SettingsWindow("Settings");
00156           sw.addWindowListener(new java.awt.event.WindowAdapter() {
00157             @Override
00158             public void windowClosing(java.awt.event.WindowEvent windowEvent) {
00159               if (mw != null)
00160                 mw.setVisible(true);
00161             }
00162           });
00163           mw.setVisible(false);
00164           //sw.run();
00165         }
00166       }
00167     });
00168     panel.add(btn, c);
00169   }
00170
00174   private void createGUI() {
00175     String parentDir = "";
00176 //    String parentDir = "/resources";
00177     JPanel panel = new JPanel(new GridBagLayout());
00178     addButton("Motility", 0, 0, new Color(255, 255, 255), parentDir+"/motility.png", panel);
00179     addButton("Chemotaxis", 1, 0, new Color(255, 255, 255), parentDir+"/chemotaxis.png", panel);
00180     addButton("Viability", 0, 1, new Color(255, 255, 255), parentDir+"/viability.png", panel);
00181     addButton("Morphometry", 1, 1, new Color(255, 255, 255), parentDir+"/morphometry.png", panel);
00182     addButton("Simulation", 0, 2, new Color(255, 255, 255), parentDir+"/settings.png", panel);
00183     addButton("Settings", 1, 2, new Color(255, 204, 153), parentDir+"/settings.png", panel);
00184     this.setContentPane(panel);
00185   }
```

```
00186
00190   private void simulate(){
00191     GenericDialog gd = new GenericDialog("Set Simulation parameters");
00192     gd.addNumericField("Beta", 0, 2);
00193     gd.addNumericField("Responsiveness (%)", 50, 2);
00194     gd.addNumericField("Length of the simulation (frames)", 500, 0);
00195     gd.showDialog();
00196     if (gd.wasCanceled())
00197       return;
00198     double beta = gd.getNextNumber();
00199     double responsiveness = gd.getNextNumber() / 100; // value must be between [0,1]
00200     int length = (int) gd.getNextNumber();
00201     Simulation sim = new PersistentRandomWalker(beta, responsiveness,
      length);
00202     try {
00203       sim.run();
00204     } catch (Exception e1) {
00205       IJ.handleException(e1);
00206 //      e1.printStackTrace();
00207     }
00208   }
00209 }
```

7.13 MorphWindow.java File Reference

Classes

- class MorphWindow

Packages

- package gui

7.14 MorphWindow.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package gui;
00020
00021 import java.awt.HeadlessException;
00022 import java.awt.Point;
00023 import java.awt.Rectangle;
00024 import java.awt.event.MouseEvent;
00025 import java.awt.event.MouseListener;
00026 import java.util.List;
00027 import java.util.ListIterator;
00028
00029 import javax.swing.event.ChangeEvent;
00030 import javax.swing.event.ChangeListener;
00031
00032 import data.Params;
00033 import data.Spermatozoon;
00034 import functions.ComputerVision;
00035 import functions.FileManager;
```

```
00036 import functions.Paint;
00037 import functions.Utils;
00038 import functions.VideoRecognition;
00039 import ij.measure.ResultsTable;
00040
00046 public class MorphWindow extends ImageAnalysisWindow implements
      ChangeListener, MouseListener {
00047
00048
00049   private boolean         isThresholding  = false;
00051   private ResultsTable       morphometrics = new ResultsTable();
00052
00056   public MorphWindow() throws HeadlessException {
00057     super();
00058     sldThreshold.setVisible(true);
00059     setChangeListener(this,sldThreshold);
00060     setMouseListener(this);
00061   }
00062
00063   /*****************************************************/
00072   public void checkSelection(int x, int y) {
00073     Point click = new Point(x, y);
00074     Utils utils = new Utils();
00075     for (ListIterator j = spermatozoa.listIterator(); j.hasNext();) {
00076       Spermatozoon sperm = (Spermatozoon) j.next();
00077       if (isClickInside(sperm, click)) {
00078         sperm.selected = !sperm.selected;
00079         if (sperm.selected) {
00080           Spermatozoon spermatozoon = utils.getSpermatozoon(sperm.
      id, spermatozoa);
00081           generateResults(spermatozoon);
00082         }
00083         break;
00084       }
00085     }
00086   }
00087
00091   public void close() {
00092     impOrig.changes = false; // This is necessary to avoid Save changes? dialog when closing
00093     impDraw.changes = false; // This is necessary to avoid Save changes? dialog when closing
00094     impOrig.close();
00095     impDraw.close();
00096   }
00097
00098   /*****************************************************/
00102   private void doMouseRefresh() {
00103
00104     if (!isThresholding) {
00105       isThresholding = true;
00106       Thread t1 = new Thread(new Runnable() {
00107         public void run() {
00108           impDraw = impOrig.duplicate();
00109           Paint paint = new Paint();
00110           paint.drawOutline(impDraw, impOutline);
00111           paint.drawBoundaries(impDraw, spermatozoa);
00112           setImage();
00113           isThresholding = false;
00114         }
00115       });
00116       t1.start();
00117     }
00118   }
00119
00124   private void doSliderRefresh() {
00125     if (!isThresholding) {
00126       isThresholding = true;
00127       Thread t1 = new Thread(new Runnable() {
00128         public void run() {
00129           deselectAll();
00130           processImage(true);
00131           isThresholding = false;
00132         }
00133       });
00134       t1.start();
00135     }
00136   }
00137
00138   /*****************************************************/
00145   private void generateResults(Spermatozoon spermatozoon) {
00146
00147     ComputerVision cv = new ComputerVision();
00148     double total_meanGray = (double) cv.getMeanGrayValue(spermatozoon,
      impGray, impTh);
00149     double total_area = spermatozoon.total_area * Math.pow(Params.
      micronPerPixel, 2);
00150     double total_perimeter = spermatozoon.total_perimeter *
      Params.micronPerPixel;
```

```
00151     double total_feret = spermatozoon.total_feret * Params.
      micronPerPixel;
00152     double total_minFeret = spermatozoon.total_minFeret * Params.
      micronPerPixel;
00153     double total_ellipticity = total_feret / total_minFeret;
00154     double total_roughness = 4 * Math.PI * total_area / (Math.pow(total_perimeter, 2));
00155     double total_elongation = (total_feret - total_minFeret) / (total_feret + total_minFeret);
00156     double total_regularity = (Math.PI * total_feret * total_minFeret) / (4 * total_area);
00157
00158     morphometrics.incrementCounter();
00159     morphometrics.addValue("ID", spermatozoon.id);
00160     morphometrics.addValue("Threshold", threshold);
00161     morphometrics.addValue("MeanGray", total_meanGray);
00162     morphometrics.addValue("Area(um^2)", total_area);
00163     morphometrics.addValue("Perimeter(um)", total_perimeter);
00164     morphometrics.addValue("Length(um)", total_feret);
00165     morphometrics.addValue("Width(um)", total_minFeret);
00166     morphometrics.addValue("Ellipticity", total_ellipticity);
00167     morphometrics.addValue("Roughness", total_roughness);
00168     morphometrics.addValue("Elongation", total_elongation);
00169     morphometrics.addValue("Regularity", total_regularity);
00170     FileManager fm = new FileManager();
00171     morphometrics.addValue("Sample", fm.getParentDirectory(
      impOrig.getTitle()));
00172     morphometrics.addValue("Filename", fm.getFilename(impOrig.getTitle()));
00173     if (!Params.male.isEmpty())
00174       morphometrics.addValue("Male", Params.male);
00175     if (!Params.date.isEmpty())
00176       morphometrics.addValue("Date", Params.date);
00177     if (!Params.genericField.isEmpty())
00178       morphometrics.addValue("Generic Field", Params.genericField);
00179     morphometrics.show("Morphometrics");
00180   }
00181
00182   /*****************************************************/
00194   public boolean isClickInside(Spermatozoon sperm, Point click) {
00195     // Get boundaries
00196     double offsetX = (double) sperm.bx;
00197     double offsetY = (double) sperm.by;
00198     int w = (int) sperm.width;
00199     int h = (int) sperm.height;
00200     // correct offset
00201     int pX = (int) (click.getX() - offsetX);
00202     int pY = (int) (click.getY() - offsetY);
00203     // IJ.log("offsetX: "+offsetX+" ; offsetY: "+offsetY+" ;w: "+w+"; h:
00204     // "+h+"px: "+pX+"; py: "+pY);
00205     Rectangle r = new Rectangle(w, h);
00206     return r.contains(new Point(pX, pY));
00207   }
00208
00209   /*****************************************************
00210    * MOUSE LISTENER
00211    *****************************************************/
00215   public void mouseClicked(MouseEvent e) {
00216     int x = e.getX();
00217     int y = e.getY();
00218     // System.out.println("X: "+ x+"; Y: "+ y);
00219     int realX = (int) (x * xFactor);
00220     int realY = (int) (y * yFactor);
00221     // System.out.println("realX: "+ realX+"; realY: "+ realY);
00222     checkSelection(realX, realY);
00223     doMouseRefresh();
00224   }
00225   public void mouseEntered(MouseEvent e) {}
00226   public void mouseExited(MouseEvent e) {}
00227   public void mousePressed(MouseEvent e) {}
00228   public void mouseReleased(MouseEvent e) {}
00229
00230   /*****************************************************/
00239   public void processImage(boolean eventType) {
00240     if (eventType || threshold == -1) {// If true, the threshold has changed or it needs to be
      calculated
00241       ComputerVision cv = new ComputerVision();
00242       impTh = impOrig.duplicate();
00243       cv.convertToGrayscale(impTh);
00244       impGray = impTh.duplicate();
00245       thresholdImagePlus(impTh);
00246       VideoRecognition vr = new VideoRecognition();
00247       List<Spermatozoon>[] sperm = vr.detectSpermatozoa(impTh);
00248       if (sperm != null)
00249         spermatozoa = sperm[0];
00250       // Calculate outlines
00251       impOutline = impTh.duplicate();
00252       cv.outlineThresholdImage(impOutline);
00253       idenfitySperm();
00254     }
00255     impDraw = impOrig.duplicate();
```

```
00256     Paint paint = new Paint();
00257     paint.drawOutline(impDraw, impOutline);
00258     paint.drawBoundaries(impDraw, spermatozoa);
00259     setImage();
00260   }
00261
00263   public void stateChanged(ChangeEvent inEvent) {
00264     Object auxWho = inEvent.getSource();
00265     if ((auxWho == sldThreshold)) {
00266       // Updating threshold value from slider
00267       threshold = sldThreshold.getValue();
00268       doSliderRefresh();
00269     }
00270   }
00271 }
```

7.15 Motility.java File Reference

Classes

- class Motility
- enum Motility.TypeOfAnalysis

Packages

- package analysis

7.16 Motility.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package analysis;
00020
00021 import java.util.HashMap;
00022 import java.util.List;
00023 import java.util.ListIterator;
00024 import java.util.Map;
00025
00026 import javax.swing.SwingWorker;
00027
00028 import data.Params;
00029 import data.Spermatozoon;
00030 import data.Trial;
00031 import functions.FileManager;
00032 import functions.Kinematics;
00033 import functions.Paint;
00034 import functions.SignalProcessing;
00035 import functions.TrialManager;
00036 import functions.Utils;
00037 import ij.IJ;
00038 import ij.ImagePlus;
00039 import ij.measure.ResultsTable;
00040
00046 public class Motility extends SwingWorker<Boolean, String> {
```

```
00047
00048   private enum TypeOfAnalysis {
00049     DIRECTORIES, DIRECTORY, FILE, NONE
00050   }
00051
00052   private TypeOfAnalysis analysis              =
    TypeOfAnalysis.NONE;
00053   private float          countProgressiveSperm = 0;
00054   private float          total_alhMax          = 0;
00055   private float          total_alhMean         = 0;
00056   private float          total_bcf             = 0;
00057   private float          total_dance           = 0;
00058   private float          total_lin             = 0;
00059   private float          total_mad             = 0;
00060   private float          total_motile          = 0;
00061   private float          total_nonMotile       = 0;
00062   private float          total_sperm           = 0;
00063   private float          total_str             = 0;
00064   private float          total_vap             = 0;
00065   private float          total_vcl             = 0;
00066   private float          total_vsl             = 0;
00067   private float          total_wob             = 0;
00068
00069   public Motility() {
00070   }
00071
00078   private void analyseDirectories() {
00079     FileManager fm = new FileManager();
00080     String folder = fm.selectFolder();
00081     List<String> subfolders = fm.getSubfolders(folder);
00082     ResultsTable rtTotal = new ResultsTable();
00083     int i = 0;
00084     for (String s : subfolders) {
00085       IJ.showProgress((double) i / subfolders.size());
00086       IJ.showStatus("Analizing folder " + i++ + "...");
00087       List<String> files = fm.getFiles(s);
00088       Map<String, Trial> trials = getTrials(files);
00089       for (String key : trials.keySet()) {
00090         Trial trial = (Trial) trials.get(key);
00091         // Motility results
00092         calculateMotility(new ResultsTable(), trial);
00093         calculateAverageMotility(new ResultsTable(), trial);
00094       }
00095       calculateTotalMotility(rtTotal, s);
00096       resetParams();
00097     }
00098     rtTotal.show("Total Motility");
00099   }
00100
00107   private void analyseDirectory() {
00108     FileManager fm = new FileManager();
00109     String folder = fm.selectFolder();
00110     List<String> files = fm.getFiles(folder);
00111     Map<String, Trial> trials = getTrials(files);
00112     ResultsTable rtIndividual = new ResultsTable();
00113     ResultsTable rtAverage = new ResultsTable();
00114     for (String key : trials.keySet()) {
00115       Trial trial = (Trial) trials.get(key);
00116       // Motility results
00117       calculateMotility(rtIndividual, trial);
00118       calculateAverageMotility(rtAverage, trial);
00119       resetParams();
00120     }
00121     rtIndividual.show("Individual motility");
00122     rtAverage.show("Average motility");
00123   }
00124
00129   private void analyseFile() {
00130     FileManager fm = new FileManager();
00131     String file = fm.selectFile();
00132     TrialManager tm = new TrialManager();
00133     Trial trial = tm.getTrialFromAVI(file);
00134     // Calculate motility
00135     ResultsTable rtIndividual = new ResultsTable();
00136     ResultsTable rtAverage = new ResultsTable();
00137     calculateMotility(rtIndividual, trial);
00138     calculateAverageMotility(rtAverage, trial);
00139     // Show results
00140     rtIndividual.show("Individual Motility");
00141     rtAverage.show("Average Motility");
00142     // Draw trajectories
00143     ImagePlus imp = fm.getAVI(file);
00144     Paint paint = new Paint();
00145     paint.draw(imp, trial.tracks);
00146     imp.show();
00147     if(Params.printXY){
00148       Utils utils = new Utils();
```

```
00149       IJ.saveString(utils.printXYCoords(trial.tracks),"");
00150     }
00151   }
00152
00161   private void calculateAverageMotility(ResultsTable rt,
      Trial trial) {
00162
00163     SignalProcessing sp = new SignalProcessing();
00164     List<List<Spermatozoon>> filteredTracks = sp.filterTracksByMotility(trial.
      tracks);
00165     float nTracks = filteredTracks.size(); // Only take into account those who passed the motility test
00166     float vsl_mean = total_vsl / nTracks;
00167     float vcl_mean = total_vcl / nTracks;
00168     float vap_mean = total_vap / nTracks;
00169     float lin_mean = total_lin / nTracks;
00170     float wob_mean = total_wob / nTracks;
00171     float str_mean = total_str / nTracks;
00172     float alhMean_mean = total_alhMean / nTracks;
00173     float alhMax_mean = total_alhMax / nTracks;
00174     float bcf_mean = total_bcf / nTracks;
00175     float dance_mean = total_dance / nTracks;
00176     float mad_mean = total_mad / nTracks;
00177     // % progressive Motile sperm
00178     float progressiveMotPercent = countProgressiveSperm / (float) nTracks;
00179     // % motility
00180     Kinematics K = new Kinematics();
00181     int[] motileSperm = K.motilityTest(trial.tracks);
00182     int countMotileSperm = motileSperm[0];
00183     total_motile += countMotileSperm;
00184     int countNonMotileSperm = motileSperm[1];
00185     total_nonMotile += countNonMotileSperm;
00186     float motility_value = (float) countMotileSperm / ((float) (countMotileSperm + countNonMotileSperm));
00187     total_sperm += nTracks;
00188
00189     rt.incrementCounter();
00190     rt.addValue("Motile trajectories", nTracks);
00191     rt.addValue("VSL Mean (um/s)", vsl_mean);
00192     rt.addValue("VCL Mean (um/s)", vcl_mean);
00193     rt.addValue("VAP Mean (um/s)", vap_mean);
00194     rt.addValue("LIN Mean ", lin_mean);
00195     rt.addValue("WOB Mean ", wob_mean);
00196     rt.addValue("STR Mean ", str_mean);
00197     rt.addValue("ALH_Mean Mean (um)", alhMean_mean);
00198     rt.addValue("ALH_Max Mean (um)", alhMax_mean);
00199     rt.addValue("BCF Mean (Hz)", bcf_mean);
00200     rt.addValue("DANCE Mean (um^2/s)", dance_mean);
00201     rt.addValue("MAD Mean (degrees)", mad_mean);
00202     rt.addValue("Progressive Motility (%)", progressiveMotPercent * 100);
00203     rt.addValue("Motility (%)", motility_value * 100);
00204     rt.addValue("Sample", trial.type);
00205     rt.addValue("ID", trial.ID);
00206     rt.addValue("Source", trial.source);
00207     if (!Params.male.isEmpty())
00208       rt.addValue("Male", Params.male);
00209     if (!Params.date.isEmpty())
00210       rt.addValue("Date", Params.date);
00211     if (!Params.genericField.isEmpty())
00212       rt.addValue("Generic Field", Params.genericField);
00213   }
00214
00223   private void calculateMotility(ResultsTable rt, Trial trial) {
00224
00225     SignalProcessing sp = new SignalProcessing();
00226     Kinematics K = new Kinematics();
00227     // Only pass from here tracks with a minimum level of motility
00228     List<List<Spermatozoon>> filteredTracks = sp.filterTracksByMotility(trial.
      tracks);
00229     // Calculate values for each track
00230     for (ListIterator iT = filteredTracks.listIterator(); iT.hasNext();) {
00231       List aTrack = (List) iT.next();
00232       List avgTrack = sp.movingAverage(aTrack);
00233       float length = (float) aTrack.size();
00234       // VSL
00235       float vsl_value = K.vsl(aTrack);
00236       total_vsl += vsl_value;
00237       // VCL
00238       float vcl_value = K.vcl(aTrack);
00239       total_vcl += vcl_value;
00240       // VAP is equivalent to calculate vcl from averaged track
00241       float vap_value = K.vcl(avgTrack);
00242       total_vap += vap_value;
00243       // Linearity
00244       float lin_value = (vsl_value / vcl_value) * 100;
00245       total_lin += lin_value;
00246       // Wobble
00247       float wob_value = (vap_value / vcl_value) * 100;
00248       total_wob += wob_value;
```

```
00249       // Straightness
00250       float str_value = (vsl_value / vap_value) * 100;
00251       total_str += str_value;
00252       // Amplitude of lateral head
00253       float[] alh_values = K.alh(aTrack, avgTrack);
00254       total_alhMean += alh_values[0];
00255       total_alhMax += alh_values[1];
00256       // Beat-cross frequency
00257       float bcf_value = K.bcf(aTrack, avgTrack);
00258       total_bcf += bcf_value;
00259       // Progressive motility
00260       String progressMotility_value = "NO";
00261       if (str_value > Params.progressMotility) {
00262         progressMotility_value = "YES";
00263         countProgressiveSperm++;
00264       }
00265       // DANCE
00266       float dance_value = vcl_value * alh_values[0];// vcl*alh_mean
00267       total_dance += dance_value;
00268       // MAD
00269       float mad_value = K.mad(aTrack);
00270       total_mad += mad_value;
00271
00272       rt.incrementCounter();
00273       rt.addValue("Length (frames)", length);
00274       rt.addValue("VSL (um/s)", vsl_value);
00275       rt.addValue("VCL (um/s)", vcl_value);
00276       rt.addValue("VAP (um/s)", vap_value);
00277       rt.addValue("LIN", lin_value);
00278       rt.addValue("WOB", wob_value);
00279       rt.addValue("STR", str_value);
00280       rt.addValue("ALH_Mean (um)", alh_values[0]);
00281       rt.addValue("ALH_Max (um)", alh_values[1]);
00282       rt.addValue("BCF (Hz)", bcf_value);
00283       rt.addValue("DANCE (um^2/s)", dance_value);
00284       rt.addValue("MAD (degrees)", mad_value);
00285       rt.addValue("Progress Motility", progressMotility_value);
00286       rt.addValue("Sample", trial.type);
00287       rt.addValue("ID", trial.ID);
00288       rt.addValue("Source", trial.source);
00289       if (!Params.male.isEmpty())
00290         rt.addValue("Male", Params.male);
00291       if (!Params.date.isEmpty())
00292         rt.addValue("Date", Params.date);
00293       if (!Params.genericField.isEmpty())
00294         rt.addValue("Generic Field", Params.genericField);
00295     }
00296   }
00297
00307   private void calculateTotalMotility(ResultsTable rt, String filename) {
00308
00309     float vsl_mean = total_vsl / total_sperm;
00310     float vcl_mean = total_vcl / total_sperm;
00311     float vap_mean = total_vap / total_sperm;
00312     float lin_mean = total_lin / total_sperm;
00313     float wob_mean = total_wob / total_sperm;
00314     float str_mean = total_str / total_sperm;
00315     float alhMean_mean = total_alhMean / total_sperm;
00316     float alhMax_mean = total_alhMax / total_sperm;
00317     float bcf_mean = total_bcf / total_sperm;
00318     float dance_mean = total_dance / total_sperm;
00319     float mad_mean = total_mad / total_sperm;
00320     // % progressive Motile sperm
00321     float progressiveMotPercent = countProgressiveSperm / (float) total_sperm;
00322     // % motility
00323     float motility_value = (float) total_motile / ((float) (total_motile +
      total_nonMotile));
00324
00325     rt.incrementCounter();
00326     rt.addValue("Motile trajectories", total_sperm);
00327     rt.addValue("VSL Mean (um/s)", vsl_mean);
00328     rt.addValue("VCL Mean (um/s)", vcl_mean);
00329     rt.addValue("VAP Mean (um/s)", vap_mean);
00330     rt.addValue("LIN Mean ", lin_mean);
00331     rt.addValue("WOB Mean ", wob_mean);
00332     rt.addValue("STR Mean ", str_mean);
00333     rt.addValue("ALH_Mean Mean (um)", alhMean_mean);
00334     rt.addValue("ALH_Max Mean (um)", alhMax_mean);
00335     rt.addValue("BCF Mean (Hz)", bcf_mean);
00336     rt.addValue("DANCE Mean (um^2/s)", dance_mean);
00337     rt.addValue("MAD Mean (degrees)", mad_mean);
00338     rt.addValue("Progressive Motility (%)", progressiveMotPercent * 100);
00339     rt.addValue("Motility (%)", motility_value * 100);
00340     rt.addValue("Filename", filename);
00341     if (!Params.male.isEmpty())
00342       rt.addValue("Male", Params.male);
00343     if (!Params.date.isEmpty())
```

```
00344       rt.addValue("Date", Params.date);
00345     if (!Params.genericField.isEmpty())
00346       rt.addValue("Generic Field", Params.genericField);
00347   }
00348
00353   @Override
00354   protected Boolean doInBackground() throws Exception {
00355     switch (analysis) {
00356       case FILE:
00357         analyseFile();
00358         break;
00359       case DIRECTORY:
00360         analyseDirectory();
00361         break;
00362       case DIRECTORIES:
00363         analyseDirectories();
00364         break;
00365     }
00366     return null;
00367   }
00368
00376   private Map<String, Trial> getTrials(List<String> filenames) {
00377     // Extract Trials
00378     TrialManager tm = new TrialManager();
00379     Map<String, Trial> trials = new HashMap<String, Trial>();
00380     for (int i = 0; i < filenames.size(); i++) {
00381       String file = filenames.get(i);
00382       Trial trial = tm.getTrialFromAVI(file);
00383       if (trial != null)
00384         trials.put(trial.type + "-_-" + trial.ID, trial); // Expression "-_-" is
00385                                                           // just a separator
00386     }
00387     return trials;
00388   }
00389
00393   private void resetParams() {
00394     total_sperm = 0;
00395     total_vsl = 0;
00396     total_vcl = 0;
00397     total_vap = 0;
00398     total_lin = 0;
00399     total_wob = 0;
00400     total_str = 0;
00401     total_alhMean = 0;
00402     total_alhMax = 0;
00403     total_bcf = 0;
00404     total_dance = 0;
00405     total_mad = 0;
00406     total_motile = 0;
00407     total_nonMotile = 0;
00408     countProgressiveSperm = 0;
00409   }
00410
00415   public void selectAnalysis() {
00416     // Ask if user wants to analyze a file or directory
00417     Object[] options = { "File", "Directory", "Multiple directories" };
00418     String question = "What do you want to analyze?";
00419     String title = "Choose one analysis...";
00420     final int FILE = 0;
00421     final int DIR = 1;
00422     final int MULTIDIR = 2;
00423     Utils utils = new Utils();
00424     int sourceSelection = utils.analysisSelectionDialog(options, question, title);
00425     if (sourceSelection < 0) {
00426       return;
00427     } else if (sourceSelection == FILE) {
00428       analysis = TypeOfAnalysis.FILE;
00429     } else if (sourceSelection == DIR) {
00430       analysis = TypeOfAnalysis.DIRECTORY;
00431     } else if (sourceSelection == MULTIDIR) {
00432       analysis = TypeOfAnalysis.DIRECTORIES;
00433     }
00434   }
00435 }
```

7.17 OpenCASA_.java File Reference

Classes

- class OpenCASA_

7.18 OpenCASA_.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 import javax.swing.UIManager;
00020 import javax.swing.UnsupportedLookAndFeelException;
00021
00022 import gui.MainWindow;
00023 import ij.IJ;
00024 import ij.ImageJ;
00025 import ij.plugin.PlugIn;
00026
00032 public class OpenCASA_ implements PlugIn {
00033
00037   public static void main(String[] args)
00038       throws ClassNotFoundException, InstantiationException, IllegalAccessException,
      UnsupportedLookAndFeelException {
00039     UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
00040     final Class<?> clazz = OpenCASA_.class;
00041     new ImageJ();// start ImageJ
00042     IJ.runPlugIn(clazz.getName(), "");// run the plugin
00043   }
00044
00049   @Override
00050   public void run(String arg) {
00051     (new MainWindow("OpenCASA")).setVisible(true);
00052   }
00053 }
```

7.19 OscillatoryWalker.java File Reference

Classes

- class OscillatoryWalker
- class **OscillatoryWalker.Cell**

Packages

- package data

7.20 OscillatoryWalker.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
```

```
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package data;
00020
00021 import java.awt.Color;
00022 import java.util.ArrayList;
00023 import java.util.List;
00024 import java.util.Random;
00025
00026 import functions.Kinematics;
00027 import functions.SignalProcessing;
00028 import ij.ImagePlus;
00029 import ij.ImageStack;
00030 import ij.process.ByteProcessor;
00031 import ij.process.ImageProcessor;
00032
00037 public class OscillatoryWalker extends Simulation {
00038
00039   class Cell {
00040
00041     float  amplitude;
00042     float  dist;
00043     double f;
00044     double phi;
00045     int    sizex;
00046     int    sizey;
00047     float  t;
00048     double T;
00049     double w;
00050     float  y;
00051
00052     Cell() {
00053       sizex = 10;
00054       sizey = 8;
00055       t = 0;
00056       y = height / 2;
00057       amplitude = 100;
00058       T = 350;
00059       f = 1 / T;// 0.01;
00060       w = 2 * Math.PI * f;
00061       phi = 0;
00062     }
00063
00071     double distance(float x1, float y1, float x2, float y2) {
00072       return Math.sqrt(Math.pow(x2 - x1, 2) + Math.pow(y2 - y1, 2));
00073     }
00074
00078     void update(ImageProcessor ip) {
00079
00080       float prevT = t;
00081       float prevY = y;
00082       Random rand = new Random();
00083       // double epsilon = 2*rand.nextGaussian();
00084       // Update variables
00085       t += 1;// T/width;
00086       // Sinusoidal function
00087       y = (float) (amplitude * Math.sin(w * t + phi)) + height / 2;
00088       // float x = (float) (amplitude*Math.cos(w*t+phi)+epsilon)+height/2;
00089       // Triangular function
00090       // y=(float)
00091       // ((2*amplitude/Math.PI)*Math.asin(Math.sin(2*Math.PI*t/T)))+height/2;
00092       dist += distance(prevT, prevY, t, y);
00093       // Draw Cell
00094       ip.fillOval((int) t, (int) y, sizex, sizey);
00095       // Save position
00096       Spermatozoon p = new Spermatozoon();
00097       p.x = t;
00098       p.y = y;
00099       track.add(p);
00100     }
00101   }
00102
00104   int    cellCount = 1;
00106   int    height    = 800;
00108   int    SIMLENGTH = 700;
00110   Cell[] sperm     = new Cell[cellCount];
00111   // Point[][] tracks = new Point[cellCount][SIMLENGTH];
00113   List<Spermatozoon> track = new ArrayList<Spermatozoon>();
00114
00116   int width = 800;
00117
```

```
00119   public OscillatoryWalker() {
00120     for (int x = cellCount - 1; x >= 0; x--) {
00121       sperm[x] = new Cell();
00122     }
00123   }
00124
00128   public ImagePlus createSimulation() {
00129     ImageStack imStack = new ImageStack(width, height);
00130     for (int i = 0; i < SIMLENGTH; i++) {
00131       ImageProcessor ip = new ByteProcessor(width, height);
00132       draw(ip);
00133       imStack.addSlice(ip);
00134     }
00135     Kinematics kinematics = new Kinematics();
00136     SignalProcessing sp = new SignalProcessing();
00137     for (int x = cellCount - 1; x >= 0; x--) {
00138       // System.out.println("Distance: "+sperm[x].dist);
00139       // System.out.println("Time: "+sperm[x].t);
00140       double vsl = track.get(0).distance(track.get(track.size() - 1)) / track.size();
00141       double vcl = sperm[x].dist / sperm[x].t;
00142       List<Spermatozoon> avgTrack = sp.movingAverage(track);
00143       double vap = kinematics.vcl(avgTrack);
00144       double lin = vsl / vcl;
00145       double wob = vap / vcl;
00146       System.out.println("VSL: " + vsl);
00147       System.out.println("VCL: " + vcl);
00148       System.out.println("VAP: " + vap);
00149       System.out.println("LIN: " + lin);
00150       System.out.println("WOB: " + wob);
00151     }
00152     return new ImagePlus("Simulation", imStack);
00153   }
00154
00159   void draw(ImageProcessor ip) {
00160     ip.setColor(Color.black);
00161     ip.fill();
00162     ip.setColor(Color.white);
00163     for (int x = cellCount - 1; x >= 0; x--) {
00164       sperm[x].update(ip);
00165     }
00166   }
00167
00171   public void run() {
00172     createSimulation().show();
00173   }
00174
00175 }
```

7.21 Paint.java File Reference

Classes

- class Paint

Packages

- package functions

7.22 Paint.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
```

```
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017
00018 Part of this code (draw and doOffset methods in particular) is a modification of a previous code written
00019 by Jonas Wilson-Leedy and Rolf Ingermann and publish in CASA_ plugin for ImageJ.
00020 Copyright © 2003 The Regents of the University of California and the Howard Hughes Medical Institute.
00021
00022 All Rights Reserved.
00023
00024 Permission to use, copy, modify, and distribute this software and its documentation for educational,
       research and
00025 non-profit purposes, without fee, and without a written agreement is hereby granted, provided that the
       above copyright
00026 notice, this paragraph and the following three paragraphs appear in all copies.
00027
00028 Permission to incorporate this software into commercial products may be obtained by contacting the Office
       of
00029 Technology Management at the University of California San Francisco [Sunita Rajdev, Ph.D., Licensing
       Officer,
00030 UCSF Office of Technology Management. 185 Berry St, Suite 4603, San Francisco, CA 94107].
00031
00032 This software program and documentation are copyrighted by The Regents of the University of California
00033 acting on behalf of the University of California San Francisco via its Office of Technology Management and
       the
00034 Howard Hughes Medical Institute (collectively, the Institution).  The software program and documentation
       are
00035 supplied "as is", without any accompanying services from the Institution. The Institution does not warrant
       that the
00036 operation of the program will be uninterrupted or error-free. The end-user understands that the program was
       developed
00037 for research purposes and is advised not to rely exclusively on the program for any reason.
00038
00039 IN NO EVENT SHALL THE INSTITUTION BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR
       CONSEQUENTIAL
00040 DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE
       INSTITUTION
00041 HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THE  INSTITUTION SPECIFICALLY DISCLAIMS ANY WARRANTIES,
       INCLUDING,
00042 BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE
       SOFTWARE
00043 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE  INSTITUTION HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE,
       SUPPORT,
00044 UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
00045
00046 */
00047
00048 package functions;
00049
00050 import java.awt.Color;
00051 import java.awt.Font;
00052 import java.awt.Polygon;
00053 import java.util.ArrayList;
00054 import java.util.List;
00055 import java.util.ListIterator;
00056
00057 import data.Params;
00058 import data.SerializableList;
00059 import data.Spermatozoon;
00060 import data.Trial;
00061 import ij.IJ;
00062 import ij.ImagePlus;
00063 import ij.ImageStack;
00064 import ij.process.ColorProcessor;
00065 import ij.process.ImageProcessor;
00066
00071 public class Paint {
00072
00073   /*****************************************************/
00081   public void chemotaxisTemplate(ColorProcessor ip, int numTracks, float chIdx, float
      slIdx, String sampleID) {
00082     // Alpha version of this method
00083     ip.setLineWidth(4);
00084     // center coords. of the cone used to clasify chemotactic trajectories
00085     int xCenter = ip.getWidth() / 2;
00086     int yCenter = ip.getHeight() / 2;
00087     float upperAngle = (float) (Params.angleDirection +
      Params.angleAmplitude / 2 + 360) % 360;
00088     upperAngle = upperAngle * (float) Math.PI / 180; // calculate and convert
00089                                                      // to radians
00090     float lowerAngle = (float) (Params.angleDirection -
      Params.angleAmplitude / 2 + 360) % 360;
00091     lowerAngle = lowerAngle * (float) Math.PI / 180; // convert to radians
00092     // Upper Line
```

```
00093     int upperLineX = xCenter + (int) (1000 * Math.cos(upperAngle));
00094     int upperLineY = yCenter - (int) (1000 * Math.sin(upperAngle));
00095     // Lower Line
00096     int lowerLineX = xCenter + (int) (1000 * Math.cos(lowerAngle));
00097     int lowerLineY = yCenter - (int) (1000 * Math.sin(lowerAngle));
00098     // Draw Chemotaxis Cone
00099     ip.moveTo((int) xCenter, (int) yCenter);
00100     ip.lineTo((int) upperLineX, (int) upperLineY);
00101     ip.moveTo((int) xCenter, (int) yCenter);
00102     ip.lineTo((int) lowerLineX, (int) lowerLineY);
00103     // Reses line width
00104     ip.setLineWidth(1);
00105     ip.setFont(new Font("SansSerif", Font.PLAIN, 16));
00106     ip.moveTo(10, 30);
00107     ip.setColor(Color.blue);
00108     ip.drawString("Sample: ");
00109     ip.moveTo(70, 30);
00110     ip.setColor(Color.black);
00111     ip.drawString(sampleID);
00112     ip.moveTo(10, 50);
00113     ip.setColor(Color.blue);
00114     ip.drawString("Number of tracks: ");
00115     ip.moveTo(135, 50);
00116     ip.setColor(Color.black);
00117     ip.drawString("" + numTracks);
00118 //    ip.moveTo(10, 70);
00119 //    ip.setColor(Color.red);
00120 //    ip.drawString("Ch-Index: ");
00121 //    ip.moveTo(70, 70);
00122 //    ip.setColor(Color.black);
00123 //    ip.drawString("" + chIdx);
00124     ip.moveTo(10, 70);
00125     ip.setColor(new Color(34, 146, 234));
00126     ip.drawString("SL-Index: ");
00127     ip.moveTo(80, 70);
00128     ip.setColor(Color.black);
00129     ip.drawString("" + slIdx);
00130   }
00131
00138   int doOffset(int center, int maxSize, int displacement) {
00139     if ((center - displacement) < 2 * displacement) {
00140       return (center + 4 * displacement);
00141     } else {
00142       return (center - displacement);
00143     }
00144   }
00145
00146   /*****************************************************/
00152   public void draw(ImagePlus imp, SerializableList theTracks) {
00153
00154     ComputerVision cv = new ComputerVision();
00155     cv.convertToRGB(imp);
00156     int nFrames = imp.getStackSize();
00157     ImageStack stack = imp.getStack();
00158     if (imp.getCalibration().scaled()) {
00159       IJ.showMessage("MultiTracker", "Cannot display paths if image is spatially calibrated");
00160       return;
00161     }
00162     int upRes = 1;
00163     String strPart;
00164     int displayTrackNr = 0;
00165     SignalProcessing sp = new SignalProcessing();
00166     SerializableList avgTracks = sp.averageTracks(theTracks);
00167     Kinematics kinematics = new Kinematics();
00168     // Draw on each frame
00169     for (int iFrame = 1; iFrame <= nFrames; iFrame++) {
00170       IJ.showProgress((double) iFrame / nFrames);
00171       IJ.showStatus("Drawing Tracks (frame "+iFrame+"/"+nFrames+")...");
00172       int yWidth = stack.getWidth();
00173       ImageProcessor ip = stack.getProcessor(iFrame);
00174       ip.setFont(new Font("SansSerif", Font.PLAIN, 16));
00175       displayTrackNr = 0;
00176       for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
00177         List zTrack = (ArrayList) iT.next();
00178         displayTrackNr++;
00179         ListIterator jT = zTrack.listIterator();
00180         Spermatozoon oldSpermatozoon = (Spermatozoon) jT.next();
00181         for (; jT.hasNext();) {
00182           Spermatozoon newSpermatozoon = (Spermatozoon) jT.next();
00183           if (kinematics.getVelocityTrackType(zTrack) == "Slow")
00184             ip.setColor(Color.white);
00185           else if (kinematics.getVelocityTrackType(zTrack) == "Normal")
00186             ip.setColor(Color.yellow);
00187           else if (kinematics.getVelocityTrackType(zTrack) == "Fast")
00188             ip.setColor(Color.red);
00189           // ip.setValue(color);
00190           if (Params.drawOrigTrajectories) {
```

```
00191             ip.moveTo((int) oldSpermatozoon.x * upRes, (int) oldSpermatozoon.y * upRes);
00192             ip.lineTo((int) newSpermatozoon.x * upRes, (int) newSpermatozoon.y * upRes);
00193           }
00194           oldSpermatozoon = newSpermatozoon;
00195           // Draw track numbers
00196           if (newSpermatozoon.z == iFrame) {
00197             strPart = "" + displayTrackNr;
00198             ip.setColor(Color.black);
00199             // we could do someboundary testing here to place the labels
00200             // better when we are close to the edge
00201             ip.moveTo((int) (oldSpermatozoon.x / Params.pixelWidth + 0),
00202                 doOffset((int) (oldSpermatozoon.y / Params.pixelHeight), yWidth, 5));
00203             ip.setColor(Color.white);
00204             ip.drawString(strPart);
00205           }
00206         }
00207       }
00208 //      System.out.println("Drawind frame: " + iFrame);
00209     }
00210     imp.updateAndRepaintWindow();
00211   }
00212
00213   /***************************************************/
00218   public void drawBoundaries(ImagePlus imp, List spermatozoa) {
00219     int yWidth = imp.getWidth();
00220     IJ.showStatus("Drawing boundaries...");
00221     ImageProcessor ip = imp.getProcessor();
00222 //    ip.setColor(Color.white);
00223     for (ListIterator j = spermatozoa.listIterator(); j.hasNext();) {
00224       Spermatozoon sperm = (Spermatozoon) j.next();
00225       ip.setLineWidth(2);
00226       if (sperm.selected)
00227         ip.drawRect((int) sperm.bx, (int) sperm.by, (int) sperm.width, (int) sperm.
    height);
00228       ip.setLineWidth(1);
00229       // Draw numbers
00230       ip.setFont(new Font("SansSerif", Font.PLAIN, 32));
00231       // we could do someboundary testing here to place the labels better
00232       // when we are close to the edge
00233       ip.moveTo((int) (sperm.x), doOffset((int) (sperm.y), yWidth, 5));
00234       try {
00235         ip.drawString("" + sperm.id);
00236       } catch (Exception e) {
00237 //        IJ.handleException(e);
00238         e.printStackTrace();
00239         // ip.drawString throws eventually an exception.
00240         // Possibly it is a bug in the ImageProcessor implementation of this
00241         // ImageJ version
00242       }
00243     }
00244   }
00245
00246   /***************************************************/
00256   public void drawChemotaxis(Trial trial, float chIdx, float slIdx) {
00257
00258     SignalProcessing sp = new SignalProcessing();
00259     SerializableList avgTracks = sp.averageTracks(trial.
    tracks);
00260     // We create another ImageProcesor to draw chemotactic cone and relative
00261     // trajectories
00262     ColorProcessor ipRelTraj = new ColorProcessor(trial.fieldWidth, trial.
    fieldHeight);
00263     ipRelTraj.setColor(Color.white);
00264     ipRelTraj.fill();
00265     int xCenter = trial.fieldWidth / 2;
00266     int yCenter = trial.fieldHeight / 2;
00267     // Draw cone used to clasify chemotactic trajectories
00268     ipRelTraj.setColor(Color.green);
00269     chemotaxisTemplate(ipRelTraj, avgTracks.size(), chIdx, slIdx, trial.
    ID);
00270     ipRelTraj.setColor(Color.red);
00271     ipRelTraj.setLineWidth(4);
00272     ipRelTraj.moveTo(xCenter, yCenter);
00273     int rx = (int) (1000 * Math.cos(Params.angleDirection * Math.PI / 180));
00274     int ry = (int) (1000 * Math.sin(Params.angleDirection * Math.PI / 180));
00275     ipRelTraj.lineTo(xCenter + rx, yCenter - ry);
00276     ipRelTraj.setLineWidth(1);
00277     // Draw average paths
00278     IJ.showStatus("Drawing Tracks...");
00279     for (ListIterator iT = avgTracks.listIterator(); iT.hasNext();) {
00280       List zTrack = (ArrayList) iT.next();
00281       ListIterator jT = zTrack.listIterator();
00282       Spermatozoon oldSpermatozoon = (Spermatozoon) jT.next();
00283       // Variables used to
00284       Spermatozoon firstSpermatozoon = new Spermatozoon();
00285       firstSpermatozoon.copy(oldSpermatozoon);
00286       int xLast = xCenter;
```

```
00287       int yLast = yCenter;
00288       for (; jT.hasNext();) {
00289         Spermatozoon newSpermatozoon = (Spermatozoon) jT.next();
00290         ipRelTraj.setColor(Color.black);
00291         ipRelTraj.moveTo(xLast, yLast);
00292         xLast = (int) (xCenter + (newSpermatozoon.x - firstSpermatozoon.x));
00293         yLast = (int) (yCenter - (newSpermatozoon.y - firstSpermatozoon.y));
00294         ipRelTraj.lineTo(xLast, yLast);
00295         oldSpermatozoon = newSpermatozoon;
00296       }
00297       ipRelTraj.drawOval(xLast - 3, yLast, 6, 6);
00298     }
00299     new ImagePlus("Chemotactic Ratios", ipRelTraj).show();
00300   }
00301
00302   /*****************************************************/
00307   public void drawOutline(ImagePlus impOrig, ImagePlus impTh) {
00308
00309     IJ.showStatus("Changing background...");
00310     ColorProcessor ipOrig = (ColorProcessor) impOrig.getProcessor();
00311     ipOrig.setColor(Color.yellow);
00312     ImageProcessor ipTh = impTh.getProcessor();
00313     int ipWidth = ipOrig.getWidth();
00314     int ipHeight = ipOrig.getHeight();
00315     for (int x = 0; x < ipWidth; x++) {
00316       IJ.showStatus("scanning pixels...");
00317       for (int y = 0; y < ipHeight; y++) {
00318         int pixel = ipTh.get(x, y);
00319         if (pixel == 0)// It's background
00320           ipOrig.drawPixel(x, y);
00321       }
00322     }
00323   }
00324
00325   /*****************************************************/
00332   public void drawRoseDiagram(int[] histogram, int radius, float chIdx, String sampleID) {
00333
00334     // Calculate maximum value of the histogram
00335     // to use it later for normalization
00336     double max = 0;
00337     for (int i = 0; i < histogram.length; i++)
00338       if (histogram[i] > max)
00339         max = histogram[i];
00340     double normFactor = radius / max;
00341     int xCenter = radius;
00342     int yCenter = radius;
00343     ColorProcessor roseDiagram = new ColorProcessor(2 * radius, 2 * radius);
00344     roseDiagram.setColor(Color.white);
00345     roseDiagram.fill();
00346     roseDiagram.setColor(new Color((int) 0, 0, 255, 10));
00347     roseDiagram.setLineWidth(1);
00348     int NBINS = histogram.length;
00349     double angleBin = 2 * Math.PI / (double) NBINS;
00350     // Draw on triangle for each bin
00351     for (double i = 0; i < NBINS; i++) {
00352       int value = histogram[(int) i];
00353       int r = (int) (value * normFactor);
00354       Polygon p = new Polygon();
00355       p.addPoint(xCenter, yCenter); // First vertex
00356       int x = (int) (r * Math.cos(i * angleBin));
00357       int y = (int) (r * Math.sin(i * angleBin));
00358       p.addPoint(xCenter + x, yCenter - y); // Second vertex
00359       x = (int) (r * Math.cos((i + 1) * angleBin));
00360       y = (int) (r * Math.sin((i + 1) * angleBin));
00361       p.addPoint(xCenter + x, yCenter - y); // Third vertex
00362       roseDiagram.fillPolygon(p);
00363     }
00364     roseDiagram.setColor(Color.gray);
00365     roseDiagram.setFont(new Font("SansSerif", Font.PLAIN, 22));
00366     // Draw line at each 30°
00367     for (double i = 0; i < 12; i++) {
00368       int x = (int) (radius * Math.cos(i * (2 * Math.PI / 12)));
00369       int y = (int) (radius * Math.sin(i * (2 * Math.PI / 12)));
00370       roseDiagram.moveTo(xCenter, yCenter);
00371       roseDiagram.lineTo(xCenter + x, yCenter - y);
00372       roseDiagram.moveTo(xCenter + x, yCenter - y);
00373       roseDiagram.drawString("" + i * 30);
00374     }
00375     roseDiagram.setColor(Color.gray);
00376     // Draw three concentric circles as reference values
00377     roseDiagram.drawOval(0, 0, 2 * radius, 2 * radius); // First circle
00378     roseDiagram.setColor(Color.black);
00379     int rx = (int) (radius * Math.cos(Math.PI / 3));
00380     int ry = (int) (radius * Math.sin(Math.PI / 3));
00381     roseDiagram.moveTo(xCenter + rx, yCenter - ry);
00382     roseDiagram.setFont(new Font("SansSerif", Font.PLAIN, 30));
00383     roseDiagram.drawString("" + (int) max);// Draw reference value
```

```
00384     roseDiagram.setColor(Color.gray);
00385     roseDiagram.drawOval(radius - 2 * radius / 3, radius - 2 * radius / 3, 4 * radius / 3, 4 * radius / 3);
      // Second
00386
      // circle
00387     roseDiagram.setColor(Color.black);
00388     int r = radius - radius / 3;
00389     rx = (int) (r * Math.cos(Math.PI / 3));
00390     ry = (int) (r * Math.sin(Math.PI / 3));
00391     roseDiagram.moveTo(xCenter + rx, yCenter - ry);
00392     roseDiagram.drawString("" + (int) (2 * max / 3));// Draw reference value
00393     roseDiagram.setColor(Color.gray);
00394     roseDiagram.drawOval(radius - radius / 3, radius - radius / 3, 2 * radius / 3, 2 * radius / 3); //
      Third
00395                                                                                                    //
      circle
00396     roseDiagram.setColor(Color.black);
00397     r = radius - 2 * radius / 3;
00398     rx = (int) (r * Math.cos(Math.PI / 3));
00399     ry = (int) (r * Math.sin(Math.PI / 3));
00400     roseDiagram.moveTo(xCenter + rx, yCenter - ry);
00401     roseDiagram.drawString("" + (int) (max / 3));// Draw reference value
00402     // Draw gradiend direction
00403     roseDiagram.setColor(Color.red);
00404     roseDiagram.setLineWidth(4);
00405     roseDiagram.moveTo(xCenter, yCenter);
00406     rx = (int) (radius * Math.cos(Params.angleDirection * Math.PI / 180));
00407     ry = (int) (radius * Math.sin(Params.angleDirection * Math.PI / 180));
00408     roseDiagram.lineTo(xCenter + rx, yCenter - ry);
00409     roseDiagram.setColor(new Color(0, 0, 255, 0));
00410     // Draw chemotaxis cone
00411     roseDiagram.setColor(Color.green);
00412     roseDiagram.setLineWidth(8);
00413     double upperAngle = (360 + Params.angleDirection +
      Params.angleAmplitude / 2) % 360;
00414     upperAngle *= Math.PI / 180;
00415     int x = (int) (radius * Math.cos(upperAngle));
00416     int y = (int) (radius * Math.sin(upperAngle));
00417     roseDiagram.moveTo(xCenter, yCenter);
00418     roseDiagram.lineTo(xCenter + x, yCenter - y);
00419     double lowerAngle = (360 + Params.angleDirection -
      Params.angleAmplitude / 2) % 360;
00420     lowerAngle *= Math.PI / 180;
00421     x = (int) (radius * Math.cos(lowerAngle));
00422     y = (int) (radius * Math.sin(lowerAngle));
00423     roseDiagram.moveTo(xCenter, yCenter);
00424     roseDiagram.lineTo(xCenter + x, yCenter - y);
00425     // Draw sample info
00426     roseDiagram.setLineWidth(1);
00427     roseDiagram.setFont(new Font("SansSerif", Font.PLAIN, 30));
00428     roseDiagram.moveTo(10, 30);
00429     roseDiagram.setColor(Color.blue);
00430     roseDiagram.drawString("Sample: " + sampleID);
00431     roseDiagram.moveTo(10, 70);
00432     roseDiagram.drawString("Ch-Index: " + chIdx);
00433
00434     new ImagePlus("Chemotactic Ratios", roseDiagram).show();
00435   }
00436
00437 }
```

7.23 Params.java File Reference

Classes

- class Params

Packages

- package data

7.24 Params.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package data;
00020
00021 import java.io.FileInputStream;
00022 import java.io.FileOutputStream;
00023 import java.io.ObjectInputStream;
00024 import java.io.ObjectOutputStream;
00025 import java.util.prefs.Preferences;
00026
00027 import ij.IJ;
00028
00033 public class Params {
00034
00036   public static float   angleAmplitude       = 90;
00041   public static int     angleDelta           = 4;
00043   public static float   angleDirection       = 0;
00045 //  public static int     bcf_shift            = 0;
00047   public static float   borderSize           = 20;
00049   public static boolean compareOppositeDirections = false;
00051   public static String  date                 = "";
00053   public static boolean drawAvgTrajectories  = true;
00055   public static boolean drawOrigTrajectories = true;
00057   public static float   frameRate            = 100;
00059   public static String  genericField         = "";
00061   public static String  male                 = "";
00065   public static float   maxDisplacement      = 10;   // um
00067   public static int     MAXINSTANGLES        = 20000;
00069   public static float   maxSize              = 400;
00070   // 10x ==> 0.58
00071   // 40x ==> 0.1455
00073   public static double       micronPerPixel   = 1;    // 0.58; 10x ISAS
00075   public static float        minSize          = 40;
00077   public static int          minTrackLength   = 15;
00079   public static int          NUMSAMPLES       = 100;
00081   public static double       pixelHeight      = 1.0;
00083   public static double       pixelWidth       = 1.0;
00085   private static Preferences prefs;
00090   public static boolean      printXY          = false;
00092   public static float        progressMotility = 80;
00094   public static float        vclLowerTh       = 45;
00096   public static float        vclMin           = 70;
00098   public static float        vclUpperTh       = 75;
00100   public static int          wSize            = 9;
00101
00103   public static void resetParams() {
00104
00105     try {
00106       FileInputStream streamIn = new FileInputStream(System.getProperty("user.dir") + "\\prefs.config");
00107       ObjectInputStream objectinputstream = new ObjectInputStream(streamIn);
00108       prefs.importPreferences(objectinputstream);
00109     } catch (Exception e) {
00110       IJ.handleException(e);
00111 //      System.out.println("Fallo de lectura");
00112     }
00113     if (prefs == null)
00114       prefs = Preferences.userNodeForPackage(Params.class);
00115     minSize = prefs.getFloat("minSize", minSize);
00116     maxSize = prefs.getFloat("maxSize", maxSize);
00117     minTrackLength = prefs.getInt("minTrackLength", minTrackLength);
00118     maxDisplacement = prefs.getFloat("maxDisplacement", maxDisplacement);
00119     wSize = prefs.getInt("wSize", wSize);
00120     vclMin = prefs.getFloat("vclMin", vclMin);
00121     vclLowerTh = prefs.getFloat("vclLowerTh", vclLowerTh);
00122     vclUpperTh = prefs.getFloat("vclUpperTh", vclUpperTh);
00123     angleDelta = prefs.getInt("angleDelta", angleDelta);
00124     angleDirection = prefs.getFloat("angleDirection", angleDirection);
00125     angleAmplitude = prefs.getFloat("angleAmplitude", angleAmplitude);
```

```
00126     compareOppositeDirections = prefs.getBoolean("compareOppositeDirections", compareOppositeDirections);
00127     printXY = prefs.getBoolean("printXY", printXY);
00128     frameRate = prefs.getFloat("frameRate", frameRate);
00129 //    male = prefs.get("male", male);
00130 //    date = prefs.get("date", date);
00131 //    genericField = prefs.get("genericField", genericField);
00132 //    bcf_shift = prefs.getInt("bcf_shift", bcf_shift);
00133     progressMotility = prefs.getFloat("progressMotility", progressMotility);
00134     micronPerPixel = prefs.getDouble("micronPerPixel", micronPerPixel);
00135     NUMSAMPLES = prefs.getInt("NUMSAMPLES", NUMSAMPLES);
00136   }
00137
00141   public static void saveParams() {
00142
00143     prefs = Preferences.userNodeForPackage(Params.class);
00144     prefs.putFloat("minSize", minSize);
00145     prefs.putFloat("maxSize", maxSize);
00146     prefs.putInt("minTrackLength", minTrackLength);
00147     prefs.putFloat("maxDisplacement", maxDisplacement);
00148     prefs.putInt("wSize", wSize);
00149     prefs.putFloat("vclMin", vclMin);
00150     prefs.putFloat("vclLowerTh", vclLowerTh);
00151     prefs.putFloat("vclUpperTh", vclUpperTh);
00152     prefs.putInt("angleDelta", angleDelta);
00153     prefs.putFloat("angleDirection", angleDirection);
00154     prefs.putFloat("angleAmplitude", angleAmplitude);
00155     prefs.putBoolean("compareOppositeDirections", compareOppositeDirections);
00156     prefs.putBoolean("printXY", printXY);
00157     prefs.putFloat("frameRate", frameRate);
00158 //    prefs.put("male", male);
00159 //    prefs.put("date", date);
00160 //    prefs.put("genericField", genericField);
00161 //    prefs.putInt("bcf_shift", bcf_shift);
00162     prefs.putFloat("progressMotility", progressMotility);
00163     prefs.putDouble("micronPerPixel", micronPerPixel);
00164     prefs.putInt("NUMSAMPLES", NUMSAMPLES);
00165
00166     try {
00167       FileOutputStream fos = new FileOutputStream(System.getProperty("user.dir") + "\\prefs.config");
00168       ObjectOutputStream oos = new ObjectOutputStream(fos);
00169       prefs.exportSubtree(oos);
00170       oos.close();
00171       fos.close();
00172     } catch (Exception e) {
00173       IJ.handleException(e);
00174 //      e.printStackTrace();
00175     }
00176   }
00177 }
```

7.25 PersistentRandomWalker.java File Reference

Classes

- class PersistentRandomWalker
- class **PersistentRandomWalker.Cell**
- class **PersistentRandomWalker.Obstacle**

Packages

- package data

7.26 PersistentRandomWalker.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
```

```
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package data;
00020
00021 import java.awt.Color;
00022 import java.awt.Point;
00023 import java.util.Random;
00024
00025 import ij.ImagePlus;
00026 import ij.ImageStack;
00027 import ij.process.ByteProcessor;
00028 import ij.process.ImageProcessor;
00029
00034 public class PersistentRandomWalker extends Simulation {
00035
00040   class Cell {
00041
00043     int sizex;
00045     int sizey;
00047     float x;
00049     float y;
00051     double angle;
00053     float speed;
00055     double Drot;
00057     double beta;
00059     double ro;
00061     Cell() {
00062       Random rand = new Random();
00063       sizex = 10;
00064       sizey = 8;
00065       x = rand.nextInt(w);
00066       y = rand.nextInt(h);
00067       angle = 0;// random(-PI,PI);
00068       speed = 3;// 4;
00069       Drot = 0.1;
00070       beta = 0;
00071       ro = 1 / Drot;
00072     }
00073
00078     Cell(double b, double responsiveCells) {
00079       Random rand = new Random();
00080       sizex = 10;
00081       sizey = 8;
00082       x = rand.nextInt(w);
00083       y = rand.nextInt(h);
00084       angle = 0;// random(-PI,PI);
00085       speed = 3;// 4;
00086       Drot = 0.1;
00087       // beta=0;//Control
00088       // Chemotaxis
00089       if (rand.nextFloat() < responsiveCells) // Only x% of the population is
00090                                               // chemoattracted
00091         beta = b;
00092       else
00093         beta = 0;
00094       ro = 1 / Drot;
00095     }
00096
00100     void update(ImageProcessor ip) {
00101       Random rand = new Random();
00102       double epsilon = rand.nextGaussian();
00103       // Persistent random walker's differential equation
00104       double da = -(beta / ro) * Math.sin(angle) + epsilon * Math.sqrt(2 * Drot);
00105       // Update variables
00106       angle += da;
00107       angle = angle % (2 * Math.PI);
00108       float dx = (float) (speed * Math.cos(angle));
00109       float dy = (float) (speed * Math.sin(angle));
00110       x += dx;
00111       y += dy;
00112       // Draw Cell
00113       ip.fillOval((int) x, (int) y, sizex, sizey);
00114
00115     }
00116   }
00117
```

```
00122   class Obstacle {
00123
00124     int x;
00125     int y;
00126     int radius;
00127
00128     Obstacle() {
00129       Random rand = new Random();
00130       x = rand.nextInt(w);
00131       y = rand.nextInt(h);
00132       radius = rand.nextInt(100);
00133     }
00134
00135     void update(ImageProcessor ip) {
00136       ip.fillOval(x, y, radius, radius);
00137     }
00138   }
00139
00141   int w = 800;
00143   int h = 800;
00145   int cellCount = 100;
00147   int obstaclesCount = 0;
00149   Cell[] sperm = new Cell[cellCount];
00151   Obstacle[] obstacles = new Obstacle[obstaclesCount];
00153   int SIMLENGTH = 500;
00155   Point[][] tracks = new Point[cellCount][SIMLENGTH];
00157   public PersistentRandomWalker() {
00158
00159     for (int x = cellCount - 1; x >= 0; x--) {
00160       sperm[x] = new Cell();
00161     }
00162     for (int x = obstaclesCount - 1; x >= 0; x--) {
00163       obstacles[x] = new Obstacle();
00164     }
00165   }
00166
00171   public PersistentRandomWalker(double b, double responsiveCells) {
00172     for (int x = cellCount - 1; x >= 0; x--) {
00173       sperm[x] = new Cell(b, responsiveCells);
00174     }
00175     for (int x = obstaclesCount - 1; x >= 0; x--) {
00176       obstacles[x] = new Obstacle();
00177     }
00178   }
00179
00185   public PersistentRandomWalker(double b, double responsiveCells, int simlength) {
00186     SIMLENGTH = simlength;
00187     for (int x = cellCount - 1; x >= 0; x--) {
00188       sperm[x] = new Cell(b, responsiveCells);
00189     }
00190     for (int x = obstaclesCount - 1; x >= 0; x--) {
00191       obstacles[x] = new Obstacle();
00192     }
00193   }
00194
00195   /*
00196    * (non-Javadoc)
00197    *
00198    * @see data.Simulation#createSimulation()
00199    */
00200   public ImagePlus createSimulation() {
00201     ImageStack imStack = new ImageStack(w, h);
00202     for (int i = 0; i < SIMLENGTH; i++) {
00203       ImageProcessor ip = new ByteProcessor(w, h);
00204       draw(ip);
00205       imStack.addSlice(ip);
00206     }
00207     return new ImagePlus("PersistentRandomWalker", imStack);
00208   }
00209
00210   /*
00211    * (non-Javadoc)
00212    *
00213    * @see data.Simulation#draw(ij.process.ImageProcessor)
00214    */
00215   void draw(ImageProcessor ip) {
00216     ip.setColor(Color.black);
00217     ip.fill();
00218     ip.setColor(Color.white);
00219
00220     for (int x = obstaclesCount - 1; x >= 0; x--) {
00221       obstacles[x].update(ip);
00222     }
00223     for (int x = cellCount - 1; x >= 0; x--) {
00224       sperm[x].update(ip);
00225     }
00226   }
```

```
00227
00228   /*
00229    * (non-Javadoc)
00230    *
00231    * @see data.Simulation#run()
00232    */
00233   public void run() {
00234     createSimulation().show();
00235   }
00236 }
```

7.27 SerializableList.java File Reference

Classes

- class SerializableList

Packages

- package data

7.28 SerializableList.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package data;
00020
00021 import java.io.Serializable;
00022 import java.util.ArrayList;
00023 import java.util.Collection;
00024
00029 public class SerializableList extends ArrayList implements Serializable {
00030
00034   public SerializableList() {}
00035
00039   public SerializableList(Collection c) {
00040     super(c);
00041   }
00042
00046   public SerializableList(int initialCapacity) {
00047     super(initialCapacity);
00048   }
00049
00050 }
```

7.29 SettingsWindow.java File Reference

Classes

- class SettingsWindow

Packages

- package gui

7.30 SettingsWindow.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package gui;
00020
00021 import java.awt.Dimension;
00022 import java.awt.GridBagConstraints;
00023 import java.awt.GridBagLayout;
00024 import java.awt.HeadlessException;
00025 import java.awt.Toolkit;
00026 import java.awt.event.ActionEvent;
00027 import java.awt.event.ActionListener;
00028 import java.awt.event.WindowEvent;
00029
00030 import javax.swing.JButton;
00031 import javax.swing.JCheckBox;
00032 import javax.swing.JFrame;
00033 import javax.swing.JLabel;
00034 import javax.swing.JPanel;
00035 import javax.swing.JTabbedPane;
00036 import javax.swing.JTextField;
00037
00038 import data.Params;
00039
00044 public class SettingsWindow extends JFrame {
00045
00046   JTextField     angleAmplitudeTF      = new JTextField("" + Params.
      angleAmplitude, 4);
00047   JTextField     angleDeltaTF          = new JTextField("" + Params.
      angleDelta, 4);
00048   JTextField     angleDirectionTF      = new JTextField("" + Params.
      angleDirection, 4);
00049   //JTextField     bcfShiftTF            = new JTextField("" + Params.bcf_shift, 4);
00050   JButton        cancelBtn;
00051   JCheckBox      compareOppositeDirCB  = new JCheckBox();
00052   JTextField     dateTF                = new JTextField(Params.date, 8);
00053   JTextField     frameRateTF           = new JTextField("" + Params.
      frameRate, 4);
00054   JTextField     genericTF             = new JTextField(Params.
      genericField, 8);
00055   JTextField     maleTF                = new JTextField(Params.male, 8);
00056   JTextField     maxDisplacementTF     = new JTextField("" + Params.
      maxDisplacement, 4);
00057   JTextField     maxSizeTF             = new JTextField("" + Params.
      maxSize, 4);
00058   JTextField     micronPerPixelTF      = new JTextField("" + Params.
      micronPerPixel, 4);
00059   JTextField     minSizeTF             = new JTextField("" + Params.
      minSize, 4);
00060   JTextField     minTrackLengthTF      = new JTextField("" + Params.
      minTrackLength, 4);
00061   JTextField     numSamplesBootsTF     = new JTextField("" + Params.
      NUMSAMPLES, 4);
00062   JCheckBox      printXYCB             = new JCheckBox();
00063   JTextField     progressiveMotilityTF = new JTextField("" + Params.
      progressMotility, 4);
00064   JButton        saveBtn;
00065   SettingsWindow sw; // Self reference used in action listeners
00066   JTextField     vclLowerThTF          = new JTextField("" + Params.
      vclLowerTh, 4);
```

```
00067   JTextField     vclMinTF              = new JTextField("" + Params.vclMin, 4);
00068   JTextField     vclUpperThTF          = new JTextField("" + Params.
    vclUpperTh, 4);
00069   JTextField     windowSizeTF          = new JTextField("" + Params.wSize, 4);
00070
00075   public SettingsWindow(String title) throws HeadlessException {
00076     super(title);
00077     sw = this;
00078     createGUI();
00079     this.setVisible(true);
00080     // this.setLocationRelativeTo(null);
00081     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
00082     int w = (int) screenSize.getWidth();
00083     int h = (int) screenSize.getHeight();
00084     this.setMinimumSize(new Dimension(w/3, h/3));
00085   }
00086
00087   private JTabbedPane addTabPane() {
00088     JTabbedPane tabbedPane = new JTabbedPane();
00089     tabbedPane.addTab("General", createGeneralBox());
00090     tabbedPane.addTab("Video", createVideoBox());
00091     tabbedPane.addTab("Chemotaxis", createChemotaxisBox());
00092     tabbedPane.addTab("Motility", createMotilityBox());
00093     return tabbedPane;
00094
00095   }
00096
00097   private void createButtons() {
00098     saveBtn = new JButton("Save");
00099     // Add action listener
00100     saveBtn.addActionListener(new ActionListener() {
00101       public void actionPerformed(ActionEvent e) {
00102         setParameters();
00103         Params.saveParams();
00104         sw.dispatchEvent(new WindowEvent(sw, WindowEvent.WINDOW_CLOSING));
00105       }
00106     });
00107     cancelBtn = new JButton("Cancel");
00108     // Add action listener
00109     cancelBtn.addActionListener(new ActionListener() {
00110       public void actionPerformed(ActionEvent e) {
00111         sw.dispatchEvent(new WindowEvent(sw, WindowEvent.WINDOW_CLOSING));
00112       }
00113     });
00114   }
00115
00119   public JPanel createChemotaxisBox() {
00120     JPanel box = new JPanel();
00121     box.setLayout(new GridBagLayout());
00122     // box.setBackground(new Color(204, 229, 255));
00123     GridBagConstraints c = new GridBagConstraints();
00124     // c.fill = GridBagConstraints.HORIZONTAL;
00125     c.gridx = 0;
00126     c.gridy = 0;
00128     JLabel label = new JLabel("Chemotactic direction (degrees): ");
00129     box.add(label, c);
00130     c.gridx = 1;
00131     box.add(angleDirectionTF, c);
00133     c.gridy += 1;
00134     label = new JLabel("Chemotactic cone's amplitude (Degrees): ");
00135     c.gridx = 0;
00136     box.add(label, c);
00137     c.gridx = 1;
00138     box.add(angleAmplitudeTF, c);
00140     c.gridy += 1;
00141     label = new JLabel("Number of bootstrapping resamples: ");
00142     c.gridx = 0;
00143     box.add(label, c);
00144     c.gridx = 1;
00145     box.add(numSamplesBootsTF, c);
00147     c.gridy += 1;
00148     label = new JLabel("Angle Delta (frames): ");
00149     c.gridx = 0;
00150     box.add(label, c);
00151     c.gridx = 1;
00152     box.add(angleDeltaTF, c);
00154     c.gridy += 1;
00155     label = new JLabel("Compare opposite directions: ");
00156     c.gridx = 0;
00157     box.add(label, c);
00158     c.gridx = 1;
00159     compareOppositeDirCB.setSelected(Params.compareOppositeDirections);
00160     box.add(compareOppositeDirCB, c);
00162     // box.setBorder(BorderFactory.createTitledBorder("Chemotaxis"));
00163
00164     return box;
00165   }
```

```
00166
00170   public JPanel createGeneralBox() {
00171     JPanel box = new JPanel();
00172     box.setLayout(new GridBagLayout());
00173     // box.setBackground(new Color(229,255,204));
00174     GridBagConstraints c = new GridBagConstraints();
00175     // c.fill = GridBagConstraints.HORIZONTAL;
00176     c.gridx = 0;
00177     c.gridy = 0;
00179     JLabel label = new JLabel("Microns per Pixel: ");
00180     c.gridx = 0;
00181     box.add(label, c);
00182     c.gridx = 1;
00183     box.add(micronPerPixelTF, c);
00185     c.gridy += 1;
00186     label = new JLabel("Minimum cell size (um^2): ");
00187     c.gridx = 0;
00188     box.add(label, c);
00189     c.gridx = 1;
00190     box.add(minSizeTF, c);
00192     c.gridy += 1;
00193     label = new JLabel("Maximum cell size (um^2): ");
00194     c.gridx = 0;
00195     box.add(label, c);
00196     c.gridx = 1;
00197     box.add(maxSizeTF, c);
00199     c.gridy += 1;
00200     label = new JLabel("Male: ");
00201     c.gridx = 0;
00202     box.add(label, c);
00203     c.gridx = 1;
00204     box.add(maleTF, c);
00206     c.gridy += 1;
00207     label = new JLabel("Date: ");
00208     c.gridx = 0;
00209     box.add(label, c);
00210     c.gridx = 1;
00211     box.add(dateTF, c);
00213     c.gridy += 1;
00214     label = new JLabel("Generic: ");
00215     c.gridx = 0;
00216     box.add(label, c);
00217     c.gridx = 1;
00218     box.add(genericTF, c);
00220     // box.setBorder(BorderFactory.createTitledBorder("General"));
00221
00222     return box;
00223   }
00224
00225   private void createGUI() {
00226     this.setLayout(new GridBagLayout());
00227     GridBagConstraints c = new GridBagConstraints();
00228     c.fill = GridBagConstraints.HORIZONTAL;
00230     c.gridx = 1;
00231     c.gridy = 0;
00232     c.ipadx = 2;
00233     c.gridheight = 8;
00234     c.gridwidth = 8;
00235     // c.gridwidth = 6;
00236     JTabbedPane tabbedPane = addTabPane();
00237     this.add(tabbedPane, c);
00239     c.gridheight = 1;
00240     c.gridwidth = 1;
00241     createButtons();
00242     c.gridx = 0;
00243     c.gridy = 8;
00244     this.add(cancelBtn, c);
00245     c.gridx = 9;
00246     c.gridy = 8;
00247     this.add(saveBtn, c);
00248
00249   }
00250
00254   public JPanel createMotilityBox() {
00255     JPanel box = new JPanel();
00256     box.setLayout(new GridBagLayout());
00257     // box.setBackground(new Color(229,255,204));
00258     GridBagConstraints c = new GridBagConstraints();
00259     // c.fill = GridBagConstraints.HORIZONTAL;
00260     c.gridx = 0;
00261     c.gridy = 0;
00263 //    JLabel label = new JLabel("Minimum shift for BCF (frames): ");
00264 //    box.add(label, c);
00265 //    c.gridx = 1;
00266 //    box.add(bcfShiftTF, c);
00268     c.gridy += 1;
00269     JLabel label = new JLabel("Progressive motility (STR>%): ");
```

```
00270     c.gridx = 0;
00271     box.add(label, c);
00272     c.gridx = 1;
00273     box.add(progressiveMotilityTF, c);
00275     c.gridy += 1;
00276     label = new JLabel("Minimum vcl (um/s): ");
00277     c.gridx = 0;
00278     box.add(label, c);
00279     c.gridx = 1;
00280     box.add(vclMinTF, c);
00282     c.gridy += 1;
00283     label = new JLabel("vcl lower threshold (um/s): ");
00284     c.gridx = 0;
00285     box.add(label, c);
00286     c.gridx = 1;
00287     box.add(vclLowerThTF, c);
00289     c.gridy += 1;
00290     label = new JLabel("vcl upper threshold (um/s): ");
00291     c.gridx = 0;
00292     box.add(label, c);
00293     c.gridx = 1;
00294     box.add(vclUpperThTF, c);
00296     // box.setBorder(BorderFactory.createTitledBorder("Motility"));
00297
00298     return box;
00299   }
00300
00304   public JPanel createVideoBox() {
00305     JPanel box = new JPanel();
00306     box.setLayout(new GridBagLayout());
00307     // box.setBackground(new Color(204, 229, 255));
00308     GridBagConstraints c = new GridBagConstraints();
00309     c.fill = GridBagConstraints.HORIZONTAL;
00310     c.gridx = 0;
00311     c.gridy = 0;
00313     JLabel label = new JLabel("Frame Rate (frames/s): ");
00314     box.add(label, c);
00315     c.gridx = 1;
00316     box.add(frameRateTF, c);
00318     c.gridy += 1;
00319     label = new JLabel("Minimum Track Length(frames): ");
00320     c.gridx = 0;
00321     box.add(label, c);
00322     c.gridx = 1;
00323     box.add(minTrackLengthTF, c);
00325     c.gridy += 1;
00326     label = new JLabel("Maximum displacement between frames (um): ");
00327     c.gridx = 0;
00328     box.add(label, c);
00329     c.gridx = 1;
00330     box.add(maxDisplacementTF, c);
00332     c.gridy += 1;
00333     label = new JLabel("Window Size (frames): ");
00334     c.gridx = 0;
00335     box.add(label, c);
00336     c.gridx = 1;
00337     box.add(windowSizeTF, c);
00339     c.gridy += 1;
00340     label = new JLabel("Print XY coords: ");
00341     c.gridx = 0;
00342     box.add(label, c);
00343     c.gridx = 1;
00344     printXYCB.setSelected(Params.printXY);
00345     box.add(printXYCB, c);
00347     // box.setBorder(BorderFactory.createTitledBorder("Recognition"));
00348
00349     return box;
00350   }
00351
00355   public void setParameters() {
00356     // General
00357     Params.frameRate = Float.parseFloat(frameRateTF.getText());
00358     Params.micronPerPixel = Double.parseDouble(micronPerPixelTF.getText());
00359     Params.male = maleTF.getText();
00360     Params.date = dateTF.getText();
00361     Params.genericField = genericTF.getText();
00362     Params.minSize = Float.parseFloat(minSizeTF.getText());
00363     Params.maxSize = Float.parseFloat(maxSizeTF.getText());
00364     Params.minTrackLength = Integer.parseInt(minTrackLengthTF.getText());
00365     Params.maxDisplacement = Float.parseFloat(maxDisplacementTF.getText()); // um =>
      pixels
00366     Params.wSize = Integer.parseInt(windowSizeTF.getText());
00367     Params.vclMin = Float.parseFloat(vclMinTF.getText());
00368     Params.vclLowerTh = Float.parseFloat(vclLowerThTF.getText());
00369     Params.vclUpperTh = Float.parseFloat(vclUpperThTF.getText());
00370     Params.angleDelta = Integer.parseInt(angleDeltaTF.getText());
00371     Params.angleDirection = Float.parseFloat(angleDirectionTF.getText());
```

```
00372     Params.angleAmplitude = Float.parseFloat(angleAmplitudeTF.getText());
00373     Params.NUMSAMPLES = Integer.parseInt(numSamplesBootsTF.getText());
00374     Params.compareOppositeDirections = compareOppositeDirCB.isSelected();
00375 //    Params.bcf_shift = Integer.parseInt(bcfShiftTF.getText());
00376     Params.progressMotility = Float.parseFloat(progressiveMotilityTF.getText());
00377     Params.printXY = printXYCB.isSelected();
00378   }
00379 }
```

7.31 SignalProcessing.java File Reference

Classes

- class SignalProcessing

Packages

- package functions

7.32 SignalProcessing.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package functions;
00020
00021 import java.util.ArrayList;
00022 import java.util.List;
00023 import java.util.ListIterator;
00024
00025 import data.Params;
00026 import data.SerializableList;
00027 import data.Spermatozoon;
00028
00033 public class SignalProcessing {
00034
00035   /*****************************************************/
00044   public SerializableList averageTracks(
      SerializableList theTracks) {
00045
00046     SerializableList avgTracks = new SerializableList();
00047     for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
00048       List aTrack = (ArrayList) iT.next();
00049       List avgTrack = movingAverage(aTrack);
00050       avgTracks.add(avgTrack);
00051     }
00052     return avgTracks;
00053   }
00054
00055   /*****************************************************/
00065   public List decimateTrack(List track, int factor) {
00066     List decimatedTrack = new ArrayList();
00067     for (ListIterator iT = track.listIterator(); iT.hasNext();) {
00068       Spermatozoon p = (Spermatozoon) iT.next();
00069       decimatedTrack.add(p);
```

```
00070       for (int i = 1; i < factor; i++) {
00071         if (iT.hasNext())
00072           p = (Spermatozoon) iT.next();
00073       }
00074     }
00075     return decimatedTrack;
00076   }
00077
00078   /*************************************************/
00086   public List decimateTracks(List theTracks, int factor) {
00087     List decimatedTracks = new ArrayList();
00088     for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
00089       List aTrack = (ArrayList) iT.next();
00090       decimatedTracks.add(decimateTrack(aTrack, factor));
00091     }
00092     return decimatedTracks;
00093   }
00094
00095   /*************************************************/
00100   public SerializableList filterTracksByLength(
      SerializableList theTracks) {
00101     SerializableList filteredTracks = new SerializableList();
00102     for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
00103       List aTrack = (ArrayList) iT.next();
00104       if (aTrack.size() >= Params.minTrackLength)
00105         filteredTracks.add(aTrack);
00106     }
00107     return filteredTracks;
00108   }
00109
00110   /*************************************************/
00115   public SerializableList filterTracksByMotility(
      SerializableList theTracks) {
00116     SerializableList filteredTracks = new SerializableList();
00117     Kinematics K = new Kinematics();
00118     for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
00119       List aTrack = (ArrayList) iT.next();
00120       if (K.motilityTest(aTrack))
00121         filteredTracks.add(aTrack);
00122     }
00123     return filteredTracks;
00124   }
00125
00131   public float[] movingAverage(float[] points, int wSize) {
00132     int nPoints = points.length;
00133     int count = 0;
00134     float[] avgPoints = new float[nPoints - wSize + 1];
00135     for (int i = wSize - 1; i < nPoints; i++) {
00136       for (int k = wSize - 1; k >= 0; k--) {
00137         avgPoints[i - wSize + 1] += points[i - k];
00138       }
00139       avgPoints[i - wSize + 1] /= (float) wSize;
00140     }
00141     return avgPoints;
00142   }
00143
00148   public List movingAverage(List track) {
00149     return movingAverage(track, Params.wSize);
00150   }
00151
00152   /*************************************************/
00162   public List movingAverage(List track, int wSize) {
00163     int nPoints = track.size();
00164     List avgTrack = new ArrayList();
00165     for (int j = wSize - 1; j < nPoints; j++) {
00166       int avgX = 0;
00167       int avgY = 0;
00168       for (int k = wSize - 1; k >= 0; k--) {
00169         Spermatozoon aSpermatozoon = (Spermatozoon) track.get(j - k);
00170         avgX += (int) aSpermatozoon.x;
00171         avgY += (int) aSpermatozoon.y;
00172       }
00173       avgX = avgX / wSize;
00174       avgY = avgY / wSize;
00175       Spermatozoon newSpermatozoon = new Spermatozoon();
00176       newSpermatozoon.x = (float) avgX;
00177       newSpermatozoon.y = (float) avgY;
00178       avgTrack.add(newSpermatozoon);
00179     }
00180     return avgTrack;
00181   }
00182 }
```

7.33 Simulation.java File Reference

Classes

- class Simulation

Packages

- package data

7.34 Simulation.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package data;
00020
00021 import ij.ImagePlus;
00022 import ij.process.ImageProcessor;
00023
00028 public abstract class Simulation {
00029
00033   abstract public ImagePlus createSimulation();
00034
00038   abstract void draw(ImageProcessor ip);
00039
00043   abstract public void run();
00044 }
```

7.35 Spermatozoon.java File Reference

Classes

- class Spermatozoon

Packages

- package data

7.36 Spermatozoon.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package data;
00020
00021 import java.io.Serializable;
00022
00027 public class Spermatozoon implements Serializable {
00028
00032   private static final long serialVersionUID = 1L;
00034   public String id = "*";
00036   public boolean flag = false;
00038   public boolean inTrack = false;
00040   public int trackNr;
00042   public float x;
00044   public float y;
00046   public int z;
00047   // Boundary data
00049   public float bx;
00051   public float by;
00053   public float width;
00055   public float height;
00056   // Selection variables
00058   public boolean selected = false;
00059   // Morphometrics
00061   public float total_area = -1;
00063   public float total_perimeter = -1;
00065   public float total_feret = -1;
00067   public float total_minFeret = -1;
00071   public void copy(Spermatozoon source) {
00072     this.id = source.id;
00073     this.x = source.x;
00074     this.y = source.y;
00075     this.z = source.z;
00076     this.trackNr = source.trackNr;
00077     this.inTrack = source.inTrack;
00078     this.flag = source.flag;
00079     this.bx = source.bx;
00080     this.by = source.by;
00081     this.width = source.width;
00082     this.height = source.height;
00083     this.selected = source.selected;
00084     this.total_area = source.total_area;
00085     this.total_perimeter = source.total_perimeter;
00086     this.total_feret = source.total_feret;
00087     this.total_minFeret = source.total_minFeret;
00088   }
00089
00095   public float distance(Spermatozoon s) {
00096     return (float) Math.sqrt(Math.pow(this.x - s.x, 2) + Math.pow(this.y - s.y, 2));
00097   }
00098
00099 }
```

7.37 Trial.java File Reference

Classes

- class Trial

Packages

- package data

7.38 Trial.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package data;
00020
00021 import java.io.Serializable;
00022
00027 public class Trial implements Serializable {
00031   private static final long serialVersionUID = 1L;
00033   public String ID = "";
00035   public String type = "";
00037   public String source = "";
00039   public SerializableList tracks = null;
00041   public int fieldWidth = 0;
00043   public int fieldHeight = 0;
00045   public Trial() {
00046   }
00053   public Trial(String ID, String type, String source, SerializableList t) {
00054     this.ID = ID;
00055     this.type = type;
00056     this.source = source;
00057     this.tracks = t;
00058   }
00067   public Trial(String ID, String type, String source, SerializableList t,int width,
      int height) {
00068     this.ID = ID;
00069     this.type = type;
00070     this.source = source;
00071     this.tracks = t;
00072     this.fieldWidth = width;
00073     this.fieldHeight = height;
00074   }
00075
00076 }
```

7.39 TrialManager.java File Reference

Classes

- class TrialManager

Packages

- package functions

7.40 TrialManager.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package functions;
00020
00021 import java.io.FileInputStream;
00022 import java.io.FileOutputStream;
00023 import java.io.IOException;
00024 import java.io.ObjectInputStream;
00025 import java.io.ObjectOutputStream;
00026 import java.util.HashMap;
00027 import java.util.Map;
00028
00029 import javax.swing.JFileChooser;
00030
00031 import data.PersistentRandomWalker;
00032 import data.SerializableList;
00033 import data.Simulation;
00034 import data.Trial;
00035 import ij.IJ;
00036 import ij.ImagePlus;
00037
00042 public class TrialManager {
00043
00044
00050   public Trial getTrialFromAVI(String path) {
00051     if (path == null)
00052       return null;
00053     FileManager fm = new FileManager();
00054     if (!fm.isAVI(path))
00055       return new Trial();
00056     // Load video
00057     ImagePlus imp = fm.getAVI(path);
00058     return getTrialFromImp(imp, path);
00059   }
00060
00069   public Trial getTrialFromImp(ImagePlus impOrig, String path) {
00070     // Analyze the video
00071     // It's necessary to duplicate the ImagePlus if
00072     // we want to draw later sperm trajectories in the original video
00073     ImagePlus imp = impOrig.duplicate();
00074     //Extract trajectories
00075     VideoRecognition vr = new VideoRecognition();
00076     SerializableList tracks = vr.analyzeVideo(imp);
00077     //Set metadata
00078     FileManager fm = new FileManager();
00079     String filename = fm.getFilename(path);
00080     String ID = filename.toLowerCase();
00081     String type = fm.getParentDirectory(path).toLowerCase();// the call toLowerCase() is
      to avoid user mistakes
00082                                                             // while naming folders. This variable
      is useful
00083                                                             // in directory analysis
00084     //Create and return trial
00085     Trial trial =  new Trial(ID, type, path, tracks, impOrig.getWidth(), impOrig.getHeight());
00086     return trial;
00087   }
00088
00092   public Map<String, Trial> readTrials() {
00093     Map<String, Trial> trials = null;
00094     try {
00095       FileManager fm = new FileManager();
00096       String file = fm.selectFile();
00097       if (file == null)
00098         return null;
00099       FileInputStream streamIn = new FileInputStream(file);
00100       ObjectInputStream objectinputstream = new ObjectInputStream(streamIn);
00101       trials = (HashMap<String, Trial>) objectinputstream.readObject();
00102     } catch (Exception e) {
```

```
00103 //      e.printStackTrace();
00104       IJ.handleException(e);
00105     }
00106     return trials;
00107   }
00108
00112   public void saveTrials(Map<String, Trial> trials) {
00113
00114     String filename = "";
00115     String dir = "";
00116     JFileChooser c = new JFileChooser();
00117     int rVal = c.showSaveDialog(null);
00118     if (rVal == JFileChooser.APPROVE_OPTION) {
00119       filename = c.getSelectedFile().getName();
00120       dir = c.getCurrentDirectory().toString();
00121     }
00122     System.out.println(dir);
00123     try {
00124       // String folder = Utils.selectFolder();
00125       if (dir == null || dir.equals(""))
00126         return;
00127       FileOutputStream fos = new FileOutputStream(dir + "\\" + filename);
00128       ObjectOutputStream oos = new ObjectOutputStream(fos);
00129       oos.writeObject(trials);
00130       oos.close();
00131       fos.close();
00132     } catch (IOException ioe) {
00133       ioe.printStackTrace();
00134     }
00135   }
00136
00143   public Trial simulateTrial(String trialID, double beta, double responsiveCells) {
00144     Simulation sim = new PersistentRandomWalker(beta, responsiveCells);
00145     ImagePlus imp = sim.createSimulation();
00146     String trialType = "Beta: "+Double.toString(beta)+";Resp: "+Double.toString(responsiveCells);
00147     String simName = trialType+"\\"+trialID;
00148     Trial tr = getTrialFromImp(imp,simName);
00149     return tr;
00150   }
00151
00158   public Map<String, Trial> simulateTrials(double beta, double responsiveCells,int
      MAXSIMULATIONS) {
00159     Map<String, Trial> trials = new HashMap<String, Trial>();
00160     for (int i = 0; i < MAXSIMULATIONS; i++) {
00161       IJ.showProgress((double) i / (double)MAXSIMULATIONS);
00162       IJ.showStatus("Simulating trial "+i+"...");
00163       Trial tr = simulateTrial(Integer.toString(i),beta,responsiveCells);
00164       trials.put(tr.ID, tr);
00165     }
00166     return trials;
00167   }
00168
00169 }
```

7.41 Utils.java File Reference

Classes

- class Utils

Packages

- package functions

7.42 Utils.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
```

```
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package functions;
00020
00021 import java.util.List;
00022 import java.util.ListIterator;
00023
00024 import javax.swing.JOptionPane;
00025
00026 import data.Spermatozoon;
00027 import ij.IJ;
00028
00033 public class Utils {
00034
00042   public int analysisSelectionDialog(Object[] options, String question, String title
      ) {
00043     int n = JOptionPane.showOptionDialog(null, question, title, JOptionPane.YES_NO_OPTION, JOptionPane.
      QUESTION_MESSAGE,
00044         null, // do not use a custom Icon
00045         options, // the titles of buttons
00046         options[0]); // default button title
00047     return n;
00048   }
00053   public int[] convertLongArrayToInt(long[] orig) {
00054     int[] arrayInt = new int[orig.length];
00055     for (int i = 0; i < orig.length; i++)
00056       arrayInt[i] = (int) orig[i];
00057     return arrayInt;
00058   }
00059
00060   /*****************************************************/
00066   public Spermatozoon getSpermatozoon(String id, List spermatozoa) {
00067     Spermatozoon spermatozoon = null;
00068     for (ListIterator j = spermatozoa.listIterator(); j.hasNext();) {
00069       Spermatozoon candidate = (Spermatozoon) j.next();
00070       if (candidate.id.equals(id) && id != "***") {
00071         spermatozoon = candidate;
00072         break;
00073       }
00074     }
00075     return spermatozoon;
00076   }
00077
00078   /*****************************************************/
00084   public String printXYCoords(List theTracks) {
00085     int nTracks = theTracks.size();
00086     // strings to print out all of the data gathered, point by point
00087     String xyPts = " ";
00088     // initialize variables
00089     int trackNr = 0;
00090     int displayTrackNr = 0;
00091     int line = 1;
00092     String output = "Line" + "\tTrack" + "\tRelative_Frame" + "\tX" + "\tY";
00093     // loop through all sperm tracks
00094     for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
00095       int frame = 0;
00096       trackNr++;
00097       IJ.showProgress((double) trackNr / nTracks);
00098       IJ.showStatus("Analyzing Tracks...");
00099       List bTrack = (List) iT.next();
00100       // keeps track of the current track
00101       displayTrackNr++;
00102       ListIterator jT = bTrack.listIterator();
00103       Spermatozoon oldSpermatozoon = (Spermatozoon) jT.next();
00104       Spermatozoon firstSpermatozoon = new Spermatozoon();
00105       firstSpermatozoon.copy(oldSpermatozoon);
00106       // For each instant (Spermatozoon) in the track
00107       String outputline = "";
00108       for (; jT.hasNext();) {
00109         Spermatozoon newSpermatozoon = (Spermatozoon) jT.next();
00110         xyPts = "\t" + displayTrackNr + "\t" + frame + "\t" + (int) newSpermatozoon.
      x + "\t" + (int) newSpermatozoon.y;
00111         frame++;
00112         oldSpermatozoon = newSpermatozoon;
00113         outputline += "\n" + line + xyPts;
```

```
00114         line++;
00115       }
00116       output += outputline;
00117     }
00118     return output;
00119   }
00120
00121 }
```

7.43 ViabilityWindow.java File Reference

Classes

- class ViabilityWindow
- enum ViabilityWindow.Channel

Packages

- package gui

7.44 ViabilityWindow.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017 */
00018
00019 package gui;
00020
00021 import java.awt.Color;
00022 import java.awt.event.MouseEvent;
00023 import java.awt.event.MouseListener;
00024 import java.util.ArrayList;
00025 import java.util.List;
00026
00027 import javax.swing.event.ChangeEvent;
00028 import javax.swing.event.ChangeListener;
00029
00030 import data.Params;
00031 import data.Spermatozoon;
00032 import functions.ComputerVision;
00033 import functions.FileManager;
00034 import functions.Paint;
00035 import functions.VideoRecognition;
00036 import ij.ImagePlus;
00037 import ij.measure.ResultsTable;
00038
00039 public class ViabilityWindow extends ImageAnalysisWindow implements
      ChangeListener, MouseListener {
00040
00041   private enum Channel {
00042     BLUE, GREEN, RED, NONE
00043   }
00044   private Channel channel = Channel.NONE;
00045   private ImagePlus            aliveImpOutline;
00046   protected List<Spermatozoon> aliveSpermatozoa = new ArrayList<Spermatozoon>();
00047   private ImagePlus            deadImpOutline;
```

```
00048   protected List<Spermatozoon> deadSpermatozoa  = new ArrayList<Spermatozoon>();
00049   private boolean              isThresholding   = false;
00050
00051   private ResultsTable results = new ResultsTable();
00052
00056   public ViabilityWindow() {
00057     super();
00058     sldRedThreshold.setVisible(true);
00059     sldGreenThreshold.setVisible(true);
00060     sldRedThreshold.addMouseListener(this);
00061     sldGreenThreshold.addMouseListener(this);
00062     sldBlueThreshold.addMouseListener(this);
00063     setChangeListener(this,sldRedThreshold);
00064     setChangeListener(this,sldGreenThreshold);
00065     setMouseListener(this);
00066   }
00067
00072   private void doSliderRefresh() {
00073     if (!isThresholding) {
00074       isThresholding = true;
00075       Thread t1 = new Thread(new Runnable() {
00076         public void run() {
00077           processImage(true);
00078           isThresholding = false;
00079         }
00080       });
00081       t1.start();
00082     }
00083   }
00084
00085   protected void drawImage() {
00086     // Draw cells on image
00087     impDraw = impOrig.duplicate();
00088     Paint paint = new Paint();
00089     if(channel==Channel.GREEN){
00090       paint.drawOutline(impDraw, aliveImpOutline);
00091       impDraw.setColor(Color.green);
00092       paint.drawBoundaries(impDraw, aliveSpermatozoa);
00093     }else if(channel==Channel.RED){
00094       paint.drawOutline(impDraw, deadImpOutline);
00095       impDraw.setColor(Color.red);
00096       paint.drawBoundaries(impDraw, deadSpermatozoa);
00097     }else if(channel==Channel.BLUE){
00098       //Not used in this version of the module
00099     }else if(channel==Channel.NONE){
00100       paint.drawOutline(impDraw, aliveImpOutline);
00101       impDraw.setColor(Color.green);
00102       paint.drawBoundaries(impDraw, aliveSpermatozoa);
00103       paint.drawOutline(impDraw, deadImpOutline);
00104       impDraw.setColor(Color.red);
00105       paint.drawBoundaries(impDraw, deadSpermatozoa);
00106     }
00107     setImage();
00108   }
00109
00110   private void generateResults() {
00111
00112     int aliveCount = aliveSpermatozoa.size();
00113     int deadCount = deadSpermatozoa.size();
00114     results.incrementCounter();
00115     results.addValue("Alives", aliveCount);
00116     results.addValue("Deads", deadCount);
00117     int total = aliveCount + deadCount;
00118     results.addValue("Total", total);
00119     float percAlives = ((float) aliveCount) / ((float) total) * 100;
00120     results.addValue("% Alives", percAlives);
00121     results.addValue("% Deads", 100-percAlives);
00122     FileManager fm = new FileManager();
00123     results.addValue("Sample", fm.getParentDirectory(impOrig.getTitle()));
00124     results.addValue("Filename", fm.getFilename(impOrig.getTitle()));
00125     if (!Params.male.isEmpty())
00126       results.addValue("Male", Params.male);
00127     if (!Params.date.isEmpty())
00128       results.addValue("Date", Params.date);
00129     if (!Params.genericField.isEmpty())
00130       results.addValue("Generic Field", Params.genericField);
00131     results.show("Viability results");
00132
00133   }
00134
00135   private List<Spermatozoon> getSpermatozoa(Channel rgbChannel) {
00136     ComputerVision cv = new ComputerVision();
00137
00138     if (rgbChannel == Channel.RED){
00139       impTh = cv.getRedChannel(impOrig.duplicate());
00140       if(threshold!=-1)
00141         threshold = redThreshold;
```

```
00142     }
00143     else if (rgbChannel == Channel.GREEN){
00144       impTh = cv.getGreenChannel(impOrig.duplicate());
00145       if(threshold!=-1)
00146         threshold = greenThreshold;
00147     }
00148     else if (rgbChannel == Channel.BLUE){
00149       impTh = cv.getBlueChannel(impOrig.duplicate());
00150       if(threshold!=-1)
00151         threshold = blueThreshold;
00152     }
00153 
00154     cv.convertToGrayscale(impTh);
00155     thresholdImagePlus(impTh);
00156     // this will be useful for painting outlines later
00157     if (rgbChannel == Channel.RED)
00158       deadImpOutline = impTh;
00159     else if (rgbChannel == Channel.GREEN)
00160       aliveImpOutline = impTh;
00161     else if (rgbChannel == Channel.BLUE)
00162       aliveImpOutline = impTh;
00163     VideoRecognition vr = new VideoRecognition();
00164     List<Spermatozoon>[] sperm = vr.detectSpermatozoa(impTh);
00165     return sperm[0];
00166   }
00167 
00168   @Override
00169   public void mouseClicked(MouseEvent e) {
00170   }
00171 
00172   @Override
00173   public void mouseEntered(MouseEvent e) {
00174   }
00175 
00176   @Override
00177   public void mouseExited(MouseEvent e) {
00178   }
00179 
00180   @Override
00181   public void mousePressed(MouseEvent e) {
00182     setRawImage();
00183   }
00184 
00185   @Override
00186   public void mouseReleased(MouseEvent e) {
00187     channel = channel.NONE;
00188     drawImage();
00189   }
00190 
00191   protected void nextAction() {
00192     generateResults();
00193   }
00194 
00195 
00196   protected void processImage(boolean eventType) {
00197     // If eventType == true, the threshold has changed or it needs to be
00198     // calculated
00199     // In that class, eventType is always true
00200     aliveSpermatozoa = getSpermatozoa(Channel.GREEN);
00201     deadSpermatozoa = getSpermatozoa(Channel.RED);
00202     if (aliveSpermatozoa != null && deadSpermatozoa != null) {
00203       spermatozoa = new ArrayList<Spermatozoon>(aliveSpermatozoa);
00204       spermatozoa.addAll(deadSpermatozoa);
00205       selectAll();// set as selected all spermatozoa to allow boundary painting
00206       idenfitySperm();
00207     }
00208     // Calculate outlines
00209     ComputerVision cv = new ComputerVision();
00210     cv.outlineThresholdImage(aliveImpOutline);
00211     cv.outlineThresholdImage(deadImpOutline);
00212     drawImage();
00213   }
00214 
00215   @Override
00216   public void stateChanged(ChangeEvent e) {
00217     Object auxWho = e.getSource();
00218     if ((auxWho == sldRedThreshold)) {
00219       channel = Channel.RED;
00220       redThreshold = sldRedThreshold.getValue();
00221       doSliderRefresh();
00222     }
00223     else if ((auxWho == sldGreenThreshold)) {
00224       channel = Channel.GREEN;
00225       // Updating threshold value from slider
00226       greenThreshold = sldGreenThreshold.getValue();
00227       doSliderRefresh();
00228     }else if(auxWho == sldBlueThreshold){
```

```
00229       channel = Channel.BLUE;
00230       blueThreshold = sldBlueThreshold.getValue();
00231       doSliderRefresh();
00232     }
00233   }
00234
00235 }
```

7.45 VideoRecognition.java File Reference

Classes

- class VideoRecognition

Packages

- package functions

7.46 VideoRecognition.java

```
00001 /*
00002  *   OpenCASA software v0.8 for video and image analysis
00003  *   Copyright (C) 2017  Carlos Alquézar
00004  *
00005  *   This program is free software: you can redistribute it and/or modify
00006  *   it under the terms of the GNU General Public License as published by
00007  *   the Free Software Foundation, either version 3 of the License, or
00008  *   (at your option) any later version.
00009  *
00010  *   This program is distributed in the hope that it will be useful,
00011  *   but WITHOUT ANY WARRANTY; without even the implied warranty of
00012  *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00013  *   GNU General Public License for more details.
00014  *
00015  *   You should have received a copy of the GNU General Public License
00016  *   along with this program.  If not, see <https://www.gnu.org/licenses/>.
00017  *
00018
00019 Part of this code (detectSpermatozoa and idendifyTracks methods in particular) is a modification of a
       previous code
00020 written by Jonas Wilson-Leedy and Rolf Ingermann and publish in CASA_ plugin for ImageJ.
00021 Copyright © 2003 The Regents of the University of California and the Howard Hughes Medical Institute.
00022
00023 All Rights Reserved.
00024
00025 Permission to use, copy, modify, and distribute this software and its documentation for educational,
       research and
00026 non-profit purposes, without fee, and without a written agreement is hereby granted, provided that the
       above copyright
00027 notice, this paragraph and the following three paragraphs appear in all copies.
00028
00029 Permission to incorporate this software into commercial products may be obtained by contacting the Office
       of
00030 Technology Management at the University of California San Francisco [Sunita Rajdev, Ph.D., Licensing
       Officer,
00031 UCSF Office of Technology Management. 185 Berry St, Suite 4603, San Francisco, CA 94107].
00032
00033 This software program and documentation are copyrighted by The Regents of the University of California
00034 acting on behalf of the University of California San Francisco via its Office of Technology Management and
       the
00035 Howard Hughes Medical Institute (collectively, the Institution).  The software program and documentation
       are
00036 supplied "as is", without any accompanying services from the Institution. The Institution does not warrant
       that the
00037 operation of the program will be uninterrupted or error-free. The end-user understands that the program was
       developed
00038 for research purposes and is advised not to rely exclusively on the program for any reason.
00039
00040 IN NO EVENT SHALL THE INSTITUTION BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR
       CONSEQUENTIAL
```

```
00041 DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE
       INSTITUTION
00042 HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. THE  INSTITUTION SPECIFICALLY DISCLAIMS ANY WARRANTIES,
       INCLUDING,
00043 BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE
       SOFTWARE
00044 PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE  INSTITUTION HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE,
       SUPPORT,
00045 UPDATES, ENHANCEMENTS, OR MODIFICATIONS.
00046 */
00047
00048 package functions;
00049
00050 import java.util.ArrayList;
00051 import java.util.List;
00052 import java.util.ListIterator;
00053
00054 import data.Params;
00055 import data.SerializableList;
00056 import data.Spermatozoon;
00057 import ij.IJ;
00058 import ij.ImagePlus;
00059 import ij.ImageStack;
00060 import ij.measure.Measurements;
00061 import ij.measure.ResultsTable;
00062 import ij.plugin.filter.ParticleAnalyzer;
00063
00064 public class VideoRecognition implements Measurements {
00065
00066   public VideoRecognition() {}
00067
00073   public SerializableList analyzeVideo(ImagePlus imp) {
00074     if (imp == null)
00075       return new SerializableList();
00076     System.out.println("converToGrayScale...");
00077     ComputerVision cv = new ComputerVision();
00078     cv.convertToGrayscale(imp);
00079     // *************************************************************
00080     // * Automatic Thresholding
00081     // *************************************************************
00082     System.out.println("thresholdStack...");
00083     cv.thresholdStack(imp);
00084     // *************************************************************
00085     // * Record particle positions for each frame in an ArrayList
00086     // *************************************************************
00087     System.out.println("detectSpermatozoa...");
00088     List[] theParticles =  detectSpermatozoa(imp);
00089     // *************************************************************
00090     // * Now assemble tracks out of the spermatozoa lists
00091     // * Also record to which track a particle belongs in ArrayLists
00092     // *************************************************************
00093     System.out.println("identifyTracks...");
00094     SerializableList theTracks = idenfityTracks(theParticles, imp.
      getStackSize());
00095     // Filtering tracks by length
00096     SignalProcessing sp = new SignalProcessing();
00097     theTracks = sp.filterTracksByLength(theTracks);
00098     // IJ.saveString(Utils.printXYCoords(theTracks),"");
00099     return theTracks;
00100   }
00101
00102   /*****************************************************/
00108   public List[] detectSpermatozoa(ImagePlus imp) {
00109
00110     int nFrames = imp.getStackSize();
00111     ImageStack stack = imp.getStack();
00112     int options = 0; // set all PA options false
00113     int measurements = MEAN + CENTROID + RECT + AREA + PERIMETER + FERET;
00114     // Initialize results table
00115     ResultsTable rt = new ResultsTable();
00116     rt.reset();
00117     int minSize = (int) (Params.minSize * Math.pow((1 / Params.
      micronPerPixel), 2));
00118     int maxSize = (int) (Params.maxSize * Math.pow((1 / Params.
      micronPerPixel), 2));
00119     // create storage for Spermatozoa positions
00120     List[] spermatozoa = new ArrayList[nFrames];
00121     // *************************************************************
00122     // * Record spermatozoa positions for each frame in an ArrayList
00123     // *************************************************************/
00124     for (int iFrame = 1; iFrame <= nFrames; iFrame++) {
00125       IJ.showProgress((double) iFrame / nFrames);
00126       IJ.showStatus("Identifying spermatozoa per frame...");
00127       spermatozoa[iFrame - 1] = new ArrayList();
00128       rt.reset();
00129       ParticleAnalyzer pa = new ParticleAnalyzer(options, measurements, rt, minSize, maxSize);
00130       pa.analyze(imp, stack.getProcessor(iFrame));
```

```
00131       float[] sxRes = rt.getColumn(ResultsTable.X_CENTROID);
00132       float[] syRes = rt.getColumn(ResultsTable.Y_CENTROID);
00133       float[] bxRes = rt.getColumn(ResultsTable.ROI_X);
00134       float[] byRes = rt.getColumn(ResultsTable.ROI_Y);
00135       float[] widthRes = rt.getColumn(ResultsTable.ROI_WIDTH);
00136       float[] heightRes = rt.getColumn(ResultsTable.ROI_HEIGHT);
00137       float[] areaRes = rt.getColumn(ResultsTable.AREA);
00138       float[] perimeterRes = rt.getColumn(ResultsTable.PERIMETER);
00139       float[] feretRes = rt.getColumn(ResultsTable.FERET);
00140       float[] minFeretRes = rt.getColumn(ResultsTable.MIN_FERET);
00141       if (sxRes == null) //Nothing detected
00142         continue;//jump to next frame
00143       for (int iPart = 0; iPart < sxRes.length; iPart++) {
00144         Spermatozoon aSpermatozoon = new Spermatozoon();
00145         aSpermatozoon.id = "***";
00146         aSpermatozoon.x = sxRes[iPart];
00147         aSpermatozoon.y = syRes[iPart];
00148         aSpermatozoon.z = iFrame - 1;
00149         aSpermatozoon.bx = bxRes[iPart];
00150         aSpermatozoon.by = byRes[iPart];
00151         aSpermatozoon.width = widthRes[iPart];
00152         aSpermatozoon.height = heightRes[iPart];
00153         aSpermatozoon.total_area = areaRes[iPart];
00154         aSpermatozoon.total_perimeter = perimeterRes[iPart];
00155         aSpermatozoon.total_feret = feretRes[iPart];
00156         aSpermatozoon.total_minFeret = minFeretRes[iPart];
00157         spermatozoa[iFrame - 1].add(aSpermatozoon);
00158       }
00159     }
00160     return spermatozoa;
00161   }
00162   /*****************************************************/
00169   public SerializableList idenfityTracks(List[] spermatozoa, int nFrames) {
00170
00171     // int nFrames = imp.getStackSize();
00172     SerializableList theTracks = new SerializableList();
00173     int trackCount = 0;
00174     if(spermatozoa == null)
00175       return theTracks;
00176     for (int i = 0; i <= (nFrames - 1); i++) {
00177       IJ.showProgress((double) i / nFrames);
00178       IJ.showStatus("Calculating Tracks...");
00179       if(spermatozoa[i] == null)//no spermatozoa detected in frame i
00180         continue; //jump to next frame
00181       for (ListIterator j = spermatozoa[i].listIterator(); j.hasNext();) {
00182         Spermatozoon aSpermatozoon = (Spermatozoon) j.next();
00183         if (!aSpermatozoon.inTrack) {
00184           // This must be the beginning of a new track
00185           List aTrack = new ArrayList();
00186           trackCount++;
00187           aSpermatozoon.inTrack = true;
00188           aSpermatozoon.trackNr = trackCount;
00189           aTrack.add(aSpermatozoon);
00190           // *************************************************************
00191           // search in next frames for more Spermatozoa to be added to
00192           // track
00193           // *************************************************************
00194           boolean searchOn = true;
00195           Spermatozoon oldSpermatozoon = new Spermatozoon();
00196           Spermatozoon tmpSpermatozoon = new Spermatozoon();
00197           oldSpermatozoon.copy(aSpermatozoon);
00198           // *
00199           // * For each frame
00200           // *
00201           for (int iF = i + 1; iF <= (nFrames - 1); iF++) {
00202             boolean foundOne = false;
00203             Spermatozoon newSpermatozoon = new Spermatozoon();
00204             // *
00205             // * For each Spermatozoon in this frame
00206             // *
00207             for (ListIterator jF = spermatozoa[iF].listIterator(); jF.hasNext() && searchOn;) {
00208               Spermatozoon testSpermatozoon = (Spermatozoon) jF.next();
00209               float distance = testSpermatozoon.distance(oldSpermatozoon);
00210               // record a Spermatozoon when it is within the search
00211               // radius, and when it had not yet been claimed by another
00212               // track
00213               if ((distance < (Params.maxDisplacement /
      Params.micronPerPixel)) && !testSpermatozoon.inTrack) {
00214                 // if we had not found a Spermatozoon before, it is easy
00215                 if (!foundOne) {
00216                   tmpSpermatozoon = testSpermatozoon;
00217                   testSpermatozoon.inTrack = true;
00218                   testSpermatozoon.trackNr = trackCount;
00219                   newSpermatozoon.copy(testSpermatozoon);
00220                   foundOne = true;
00221                 } else {
00222                   // if we had one before, we'll take this one if it is
```

```
00223                   // closer. In any case, flag these Spermatozoa
00224                   testSpermatozoon.flag = true;
00225                   if (distance < newSpermatozoon.distance(oldSpermatozoon)) {
00226                     testSpermatozoon.inTrack = true;
00227                     testSpermatozoon.trackNr = trackCount;
00228                     newSpermatozoon.copy(testSpermatozoon);
00229                     tmpSpermatozoon.inTrack = false;
00230                     tmpSpermatozoon.trackNr = 0;
00231                     tmpSpermatozoon = testSpermatozoon;
00232                   } else {
00233                     newSpermatozoon.flag = true;
00234                   }
00235                 }
00236               } else if (distance < (Params.maxDisplacement /
      Params.micronPerPixel)) {
00237                 // this Spermatozoon is already in another track but
00238                 // could have been part of this one
00239                 // We have a number of choices here:
00240                 // 1. Sort out to which track this Spermatozoon really
00241                 // belongs (but how?)
00242                 // 2. Stop this track
00243                 // 3. Stop this track, and also delete the remainder of
00244                 // the other one
00245                 // 4. Stop this track and flag this Spermatozoon:
00246                 testSpermatozoon.flag = true;
00247               }
00248             }
00249             if (foundOne)
00250               aTrack.add(newSpermatozoon);
00251             else
00252               searchOn = false;
00253             oldSpermatozoon.copy(newSpermatozoon);
00254           }
00255           theTracks.add(aTrack);
00256         }
00257       }
00258     }
00259     return theTracks;
00260   }
00261
00262 }
```

Index