Universidad Zaragoza

Trabajo Fin de Grado

Implementación e Integración en OpenCASA de Módulos para Estudios de Acumulación y Recuento Celular

Implementation and Integration in OpenCASA of Modules to Analyze Cell Accumulation and Count

Autor

Jorge Yagüe Martínez

Directores

Jesús Alastruey Benedé

Rosaura Pérez Pe

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2019

AGRADECIMIENTOS

En primer lugar, me gustaría agradecer a Jesús Alastruey su oferta para realizar este trabajo bajo su dirección, ayudándome en cada paso del proyecto y resolviendo todas dudas que me han ido surgiendo. Me gustaría también agradecer a Rosaura Pérez, también directora del trabajo, el haberme introducido en un campo muy distinto al de la informática como es la biología molecular, y ayudándome en todas cuestiones que me han aparecido sobre él.

En segundo lugar, me gustaría agradecer a mi familia todo su apoyo moral en todos estos años de carrera, especialmente durante este último trabajo fin de grado, lo cual me ha permitido llegar hasta donde estoy.

Por último, quiero agradecer a todo el personal docente del grado de Ingeniería Informática de la Universidad de Zaragoza que me ha impartido clase durante mi estancia en la carrera, cuya profesionalidad y dedicación me ha hecho disfrutar todavía más de la especialidad de Computación.

Implementación e Integración en OpenCASA de Módulos para Estudios de Acumulación y Recuento Celular

RESUMEN

Es fundamental que los espermatozoides se dirijan al ovocito en el proceso de fertilización. En el caso de algunos animales como los invertebrados, ya ha sido demostrado que los espermatozoides son atraídos hacia el óvulo por un proceso de quimiotaxis. Sin embargo, en otras especies como los mamíferos, se han propuesto varios mecanismos de respuesta a un estímulo específico como termotaxis, reotaxis y quimiotaxis. Actualmente no se puede descartar que los espermatozoides se dirigen al ovocito debido a una combinación de dichos mecanismos. Por el momento no existen programas en el mercado para determinar si existe quimioatracción hacia una sustancia, por eso el grupo BIOFITER de la Universidad de Zaragoza desarrolló un módulo dentro del programa OpenCASA con el fin de analizar este comportamiento. Sin embargo, sería muy interesante poder analizar la acumulación de espermatozoides en contacto con los estímulos. Para ello, sería muy útil disponer de una herramienta que detectase este efecto de manera visual. También sería importante implementar una herramienta para realizar el recuento celular en una imagen captada con microscopio, lo cual evitaría a los investigadores el proceso de contar manualmente los espermatozoides.

En este trabajo se presenta la implementación de dos nuevos módulos del programa OpenCASA: módulo de acumulación celular y módulo de recuento celular. El módulo de acumulación celular se utiliza para proporcionar información visual de una muestra mediante un mapa de calor, así el usuario podrá analizar si los espermatozoides se acumulan cerca de la sustancia inyectada o al contrario no reaccionan a ella. El módulo de recuento celular sirve para estimar la concentración de una muestra en espermatozoides por mililitro, para este caso se podrá utilizar una cuadrícula de dimensiones conocidas que ayudará a calcular la concentración de manera más precisa.

Índice

Capítulo 1

Introducción

1.1. Motivación

Es fundamental que los espermatozoides se dirijan al ovocito en el proceso de fertilización. En el caso de los invertebrados marinos, está demostrado que los espermatozoides son atraídos hacia el óvulo por un fenómeno de quimiotaxis, es decir, se mueven en función de un gradiente de concentración en una determinada sustancia secretada por el óvulo. En dichas especies, ya se observó que, en presencia de un gradiente de sustancias liberadas por el óvulo, sus espermatozoides dejaban de moverse en círculos para adquirir un movimiento helicoidal, cuyo eje longitudinal apuntaba a la fuente del gradiente. En el caso de los mamíferos, se han propuesto hasta la fecha tres mecanismos diferentes: termotaxis [1] [2], reotaxis [3] y quimiotaxis [4], cada uno de los cuales es respuesta a un estímulo específico: gradiente de temperatura, flujo de fluido y gradiente de concentración, respectivamente. Actualmente no se puede descartar que el movimiento guiado hacia el ovocito se deba a una combinación de varios de esos mecanismos.

Para determinar si existe quimioatracción hacia una sustancia, no existen por el momento programas en el mercado para llevar a cabo esta evaluación. El grupo BIOFITER (Biología, Fisiología y Tecnologías de la Reproducción) de la Universidad de Zaragoza desarrolló para ello un módulo, dentro del programa OpenCASA [5], que detecta las coordenadas de las trayectorias de los espermatozoides y las dibuja normalizándolas a un mismo punto de referencia.

Sin embargo, en muchas ocasiones, resultaría muy interesante poder analizar la acumulación de espermatozoides en contacto directo con los estímulos. Esto se llevaría a cabo colocando la sustancia con una pipeta o en forma de microgotas directamente en la muestra donde se encuentran moviéndose los espermatozoides y de esta forma se acumularán cerca de la fuente del estímulo. Por ello sería muy útil disponer de una herramienta que detectase este efecto de acumulación. También se va a incluir una

función para realizar el recuento celular en una imagen de microscopio tomada con o sin cuadrícula, lo que evitaría a los investigadores el proceso de contar manualmente los espermatozoides para saber la concentración de la muestra.

1.2. Objetivos y Tareas

El objetivo de este trabajo es ampliar las funcionalidades del programa OpenCASA mediante el desarrollo e integración de dos nuevos módulos: módulo de estudio de la acumulación celular y módulo para medir la concentración celular en imágenes captadas con microscopio.

El módulo de estudio de la acumulación celular consiste en que el programa pueda proporcionar información visual de la acumulación estática o dinámica (imagen o vídeo) de las células en una muestra con una sustancia quimioatrayente. De este modo, el usuario podrá evaluar a simple vista si las células siguen el movimiento esperado y en qué magnitud mediante un mapa de calor superpuesto en la imagen o vídeo original.

El módulo de recuento celular para medir la concentración tiene como objetivo proporcionar el número de espermatozoides por mililitro que hay en una muestra determinada. Para ello, se usarán imágenes que cuentan con un dispositivo especial que presenta una cuadrícula definida, de la cual se conoce el lado de cada cuadro y la profundidad de la misma. A partir de esta información, el programa obtiene los espermatozoides de la imagen y cuenta los que están contenidos en dicha cuadrícula, de modo que finalmente se puede calcular la proporción de espermatozoides por mililitro contenidos en la muestra.

A la hora de implementar los módulos, hay que tener en cuenta que aunque OpenCASA se desarrolló inicialmente para analizar la funcionalidad espermática, puede utilizarse también de manera análoga para el estudio de cualquier otra célula.

La validación de estos nuevos módulos se ha realizado mediante estudios de simulación para el primer objetivo, y comparación con recuento a través de un observador en el caso del segundo.

1.3. Planificación

Durante la realización de este trabajo, desde Febrero hasta Septiembre del 2019 se han abordado diferentes tareas, entre las que destacan el estudio inicial del programa OpenCASA, la implementación de ambos módulos y su correspondiente validación.

La figura 1.1 muestra la distribución del trabajo realizado mediante un diagrama de Gantt, el cual incluye las horas dedicadas a cada parte del trabajo.

Figura 1.1: Diagrama de Gantt con la distribución de horas

1.4. Descripción del Contenido

En el capítulo dos se van a explicar las herramientas necesarias que se han utilizado como base de este trabajo, el programa de análisis de imagen ImageJ y el software original OpenCASA al que se le van a añadir los nuevos módulos.

Los capítulos 3 y 4 describen el diseño e implementación de los nuevos módulos de acumulación y recuento celular, desde el punto inicial del desarrollo hasta el estado final.

En el capítulo 5 se analizan los resultados obtenidos comparando con imágenes o vídeos de simulación ya precalculados para el módulo del acumulación celular y con mediciones a través de un observador para obtener la concentración en el caso del módulo de recuento celular.

Por último, en el capítulo 6 se muestran las conclusiones obtenidas al finalizar el trabajo, basándose en los apartados anteriores.

A modo de información adicional, se ha añadido un anexo que contiene el código nuevo desarrollado y el que ha sido modificado del programa original.

Capítulo 2

Herramientas

En este capítulo se presentan las herramientas que han sido utilizadas como punto inicial de este trabajo. Primero se describe el programa ImageJ, que es un software de código abierto del que se ha aprovechado su funcionalidad para integrar OpenCASA como un complemento (*plug-in*). En la segunda sección se habla del software OpenCASA, a partir de cuyo código base se van a implementar los nuevos módulos para posteriormente encapsularlos en el programa original.

2.1. ImageJ

ImageJ es una herramienta gratuita para procesar imágenes programada en Java creada en el National Institutes of Health. Es un programa diseñado de manera que proporciona extensibilidad gracias a la implementación de *plugins* en él. ImageJ también incluye un editor y un compilador Java que pueden ser usados para desarrollar *plugins* de escaneo personalizado, análisis o procesamiento. Los *plugins* escritos por usuarios permiten resolver muchos problemas de procesado y análisis de imágenes [6] [7], desde imágenes en vivo de las células en tres dimensiones, procesado de imágenes radiológicas, comparaciones de múltiples datos de sistema de imagen hasta sistemas automáticos de hematología.

En este caso se va a hacer uso de las bibliotecas disponibles de ImageJ para desarrollar las nuevas funcionalidades de la aplicación OpenCASA y posteriormente poder encapsularlo y añadirlo como *plugin* en ImageJ.

2.2. OpenCASA

En el campo de las técnicas de reproducción asistida, los sistemas CASA (Computer-Assisted Sperm Analysis) han demostrado su utilidad y potencial para evaluar la calidad del esperma, mejorando la predicción de la fertilidad potencial de

una dosis seminal. Aunque la mayoría de laboratorios y centros científicos usan sistemas comerciales, en los últimos años han surgido alternativas gratuitas y de código abierto que pueden reducir los costes que los grupos de investigación tienen que afrontar. Sin embargo, estas alternativas de código abierto no pueden analizar respuestas cinéticas del esperma ante diferentes estímulos, como la quimiotaxis, termotaxis o reotaxis. Además, los programas lanzados hasta ahora no suelen estar diseñados para fomentar la escalabilidad y la continuidad del desarrollo del software. El grupo BIOFITER ha desarrollado un sistema CASA de código abierto, llamado OpenCASA [5], que permite a los usuarios estudiar tres parámetros clásicos de funcionalidad espermática: motilidad, morfometría e integridad de la membrana (*viability*) y ofrece la posibilidad de analizar la respuesta de los espermatozoides a diferentes estímulos (útil para quimiotaxis, termotaxis o reotaxis) u otras células móviles como bacterias, usando un único programa.

Un repositorio en la plataforma GitHub, `https://github.com/calquezar/OpenCASA`, permite a los investigadores descargar el software y también contribuir en futuras mejoras. Este software propuesto usa una mínima infraestructura centralizada para permitir el continuo desarrollo de sus módulos por la comunidad de investigadores.

Capítulo 3

Módulo de Acumulación Celular

3.1. Procedimiento

La detección de figuras en imágenes requiere un preprocesado consistente en la conversión a escala de grises y en la ecualización de su histograma. De esta forma se mejora el contraste en caso de que la iluminación de la imagen no sea uniforme y se puede aplicar un *Threshold* para detectar los bordes de las figuras contenidas en ella. Las figuras 3.1 y 3.2 muestran la imagen original y la imagen con el histograma ecualizado, respectivamente. El programa OpenCASA utiliza por defecto el método de *Threshold* Otsu [8], que es el más conocido para detectar figuras en imágenes. Sin embargo, en este nuevo módulo no es válido porque las imágenes utilizadas contienen mayor cantidad de espermatozoides y de menor tamaño, ya que se usan menores aumentos en el microscopio para captar mayor amplitud de campo visual. Haciendo pruebas mediante todos los métodos de *Threshold* disponibles en ImageJ sobre estas imágenes, el que más espermatozoides detecta es el método RenyiEntropy [9].

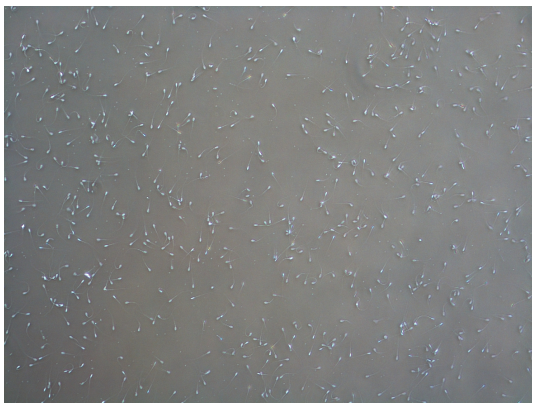

Figura 3.1: Imagen original

Figura 3.2: Imagen ecualizada

Como se puede observar en la figura 3.3, el método de *Threshold* Otsu no puede diferenciar los espermatozoides de la imagen al ser muy pequeños, mientras que en la figura 3.4 con Threshold RenyEntropy sí que se distinguen la mayoría de los

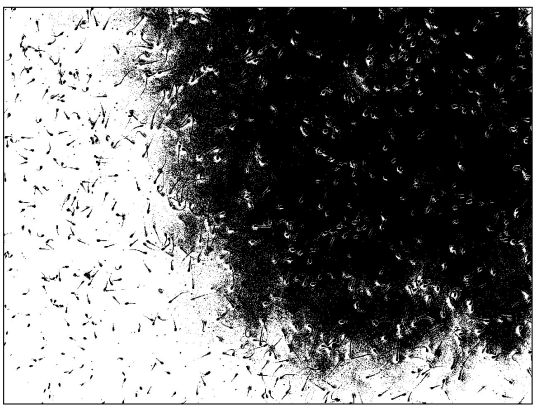

Figura 3.3: Threshold Otsu

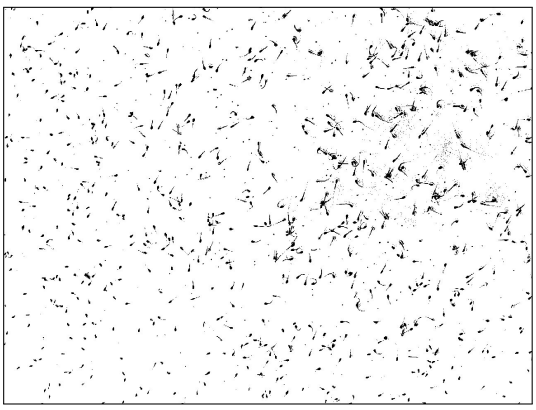

Figura 3.4: Threshold RenyiEntropy

espermatozoides.

A continuación se utiliza la función *Analize Particles* de ImageJ que detecta los espermatozoides de la imagen junto con sus posiciones. Para que todos los espermatozoides se detecten correctamente es necesario que el usuario introduzca en los ajustes de este módulo tanto el tamaño mínimo y máximo de los espermatozoides a detectar en micrómetros (μm), como la escala de la imagen en μm/píxel. De esta manera el programa puede detectar los espermatozoides basándose en su tamaño a escala real como se puede observar en la figura 3.5.

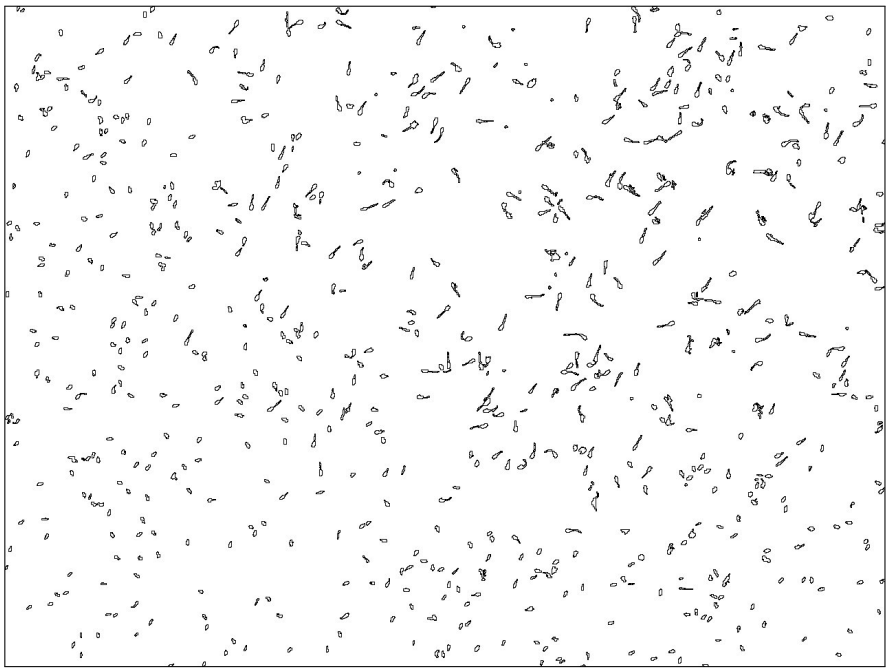

Figura 3.5: Espermatozoides detectados

Posteriormente se almacenan todos los espermatozoides por su posición en una estructura que permita luego su acceso a la hora de generar el mapa de calor. Para este propósito se ha utilizado la biblioteca de código abierto Java-ML [10], que contiene

una estructura para almacenar datos de tipo *k-d tree*. Este tipo de estructura permite guardar datos ordenados por varias (k) dimensiones, en este caso se utiliza para guardar los espermatozoides por su posición en píxeles (x, y). La principal ventaja de los *k-d tree* es que el tiempo en el peor caso para buscar espermatozoides en un radio definido es de orden $O(n^{1/2})$, y es la operación que más se va a realizar en este módulo.

Una vez que todos los espermatozoides se han almacenado en el *k-d tree*, se recorre la imagen original píxel a píxel buscando en cada uno de ellos los n espermatozoides más cercanos en un radio de distancia R definido por el usuario. Los resultados obtenidos se guardan en una matriz cuyas dimensiones son el ancho y el alto de la imagen original. Para las zonas laterales este radio R sobresale de la imagen, por tanto hay que ajustar la concentración en función de la proporción del círculo que queda dentro de la imagen. En el caso de las los bordes de la imagen, excluyendo las esquinas, el círculo solamente sobresale por un lado, como se observa en la figura 3.6. Por tanto, para obtener el área del círculo que queda dentro de la imagen se le resta el área del sector circular que sobresale de la imagen, y que se calcula mediante las ecuaciones 3.1 y 3.2. Conociendo el área del sector que queda fuera A_{sector} y el área real del círculo, se obtiene la constante de proporción $k = \frac{\pi R^2}{\pi R^2 - A_{sector}}$ por la que se multiplica el número de espermatozoides encontrados en ese radio R. Solamente es necesario calcular esta constante para todos puntos de un borde de la imagen, ya que por simetría se puede aplicar al resto de bordes, exceptuando las esquinas.

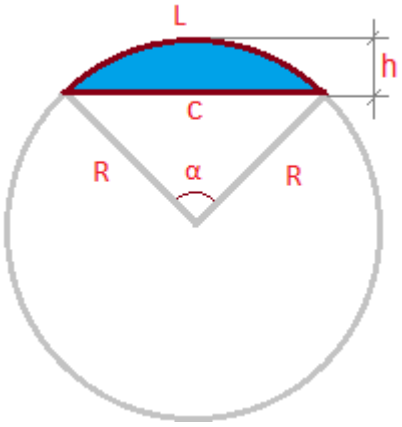

Figura 3.6: Sector circular

$$\alpha = 2 \arccos(1 - \frac{h}{R}) \tag{3.1}$$

$$A_{sector} = \frac{1}{2} R^2 (\alpha - \sin \alpha) \tag{3.2}$$

Para obtener el área real en las cuatro esquinas el procedimiento es más complicado, ya que el círculo sobresale por dos lados de la imagen como se ve en la figura 3.7.

Hemos visto cómo se calcula el área de los sectores en la sección anterior, de modo que ahora hay que calcular el área de la intersección para obtener el área real según las ecuaciones 3.3 y 3.4.

$$A_{interseccion} = A_{sector1} \bigcap A_{sector2} \tag{3.3}$$

$$A_{real} = A_{total} - A_{sector1} - A_{sector2} + A_{interseccion} \tag{3.4}$$

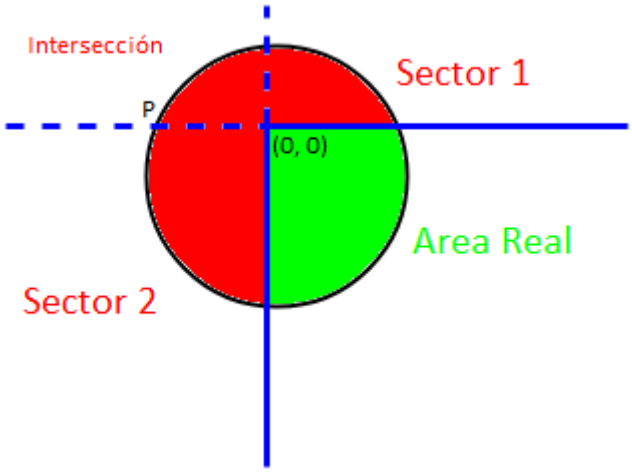

Figura 3.7: Esquina circular

Para calcular el área de la intersección se va a utilizar la función de la circunferencia 3.5, donde (a, b) son las coordenadas de su centro. Usando los límites de la imagen como origen de los ejes x e y se calcula la integral positiva de la zona que queda fuera entre los puntos P y 0. La integral resultante que nos indica el área de la intersección entre los dos sectores se muestra en las ecuaciones 3.6 y 3.7. En el caso de que la intersección entre ambos vectores sea vacía, su área será negativa y por tanto no se sumará al área real. Solamente es necesario calcular este ajuste para los puntos de una esquina, ya que por simetría el resultado es el mismo para las otras tres esquinas de la imagen.

$$(x-a)^2 + (y-b)^2 = R^2 \Rightarrow y = \sqrt{R^2 - (x-a)^2} + b \tag{3.5}$$

$$A_{interseccion} = \int_P^0 \sqrt{R^2 - (x-a)^2} + b \; dx \tag{3.6}$$

$$-\frac{(P-a)\sqrt{R^2 - P^2 + 2aP - a^2} + a\sqrt{R^2 - a^2} + r^2(\arcsin\frac{P-a}{r} + \arcsin\frac{a}{r}) + 2bP}{2} \tag{3.7}$$

Una vez calculada la matriz completa con todos resultados de acumulación en cada celda representado cada píxel, se tiene que obtener un vector que represente los colores que se van a utilizar para crear el mapa de calor. Este vector tiene tantas componentes como número de acumulación máxima se haya obtenido en el cálculo de la matriz, ya que este valor representará la zona donde se han detectado más espermatozoides y tiene que ser un color más cálido. En principio se pensó en calcular los colores de manera manual siendo azul las zonas donde la concentración es 0, rojo las zonas donde la acumulación es la máxima y verde para la acumulación intermedia. Los demás colores se extrapolarían a partir de estos tres valores.

Finalmente se decidió utilizar una LUT (Look Up Table) de ImageJ llamada *Jet* con el diseño que se muestra en la figura 3.8, que también representa los colores desde el frío al cálido con valores entre 0 y 255. De este modo, para obtener el vector que representa la escala de calor se consulta esta LUT, interpolando los valores mínimo y máximo encontrados entre 0 y 255 para calcular el color adecuado.

Figura 3.8: Jet LUT

Por último, teniendo la matriz con la acumulación en cada píxel, y la escala en forma de vector con el color correspondiente para cada valor de acumulación encontrado, falta crear una nueva imagen RGB con las mismas dimensiones que la original y pintar píxel a píxel el color correspondiente basándose en la matriz de acumulación y la escala de colores calculada, obteniendo resultados como los mostrados en la figura 3.9.

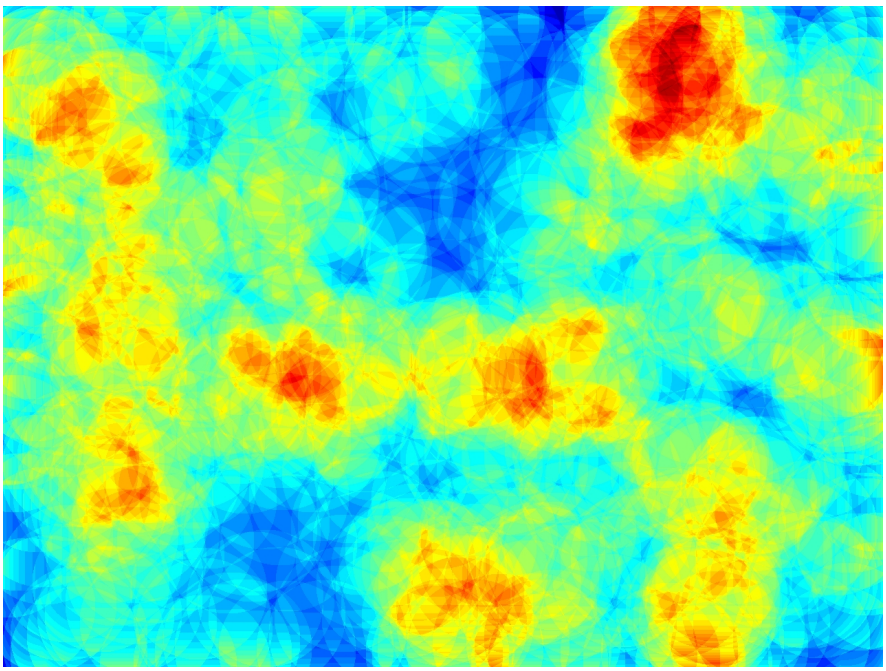

Figura 3.9: Mapa de calor

3.2. Mapa de calor en imagen

El usuario puede interactuar con el mapa de calor creado a partir de una imagen mediante unos controles que permiten modificar diversos parámetros.

En la primera versión de la aplicación se añadió un *Slider* que representa el radio de búsqueda en cada píxel, de modo que al modificar su valor se realiza todo el procedimiento de cálculo de la matriz de acumulación con el nuevo radio y se obtiene una nueva escala de color. Al ser la primera versión el cálculo de la escala de calor se hacía de manera manual con colores entre el azul y el rojo como se observa en la figura 3.10.

En la segunda versión se implementó la escala de color basada en la LUT Jet, de manera que los colores fueran más representativos y no se tuvieran que hacer los cálculos de forma manual. Además, se añadió un nuevo *Slider* que controla la transparencia de la imagen original, así se podría validar al momento si de verdad el mapa de calor realizado coincide con la posición de los espermatozoides, cambiando el valor de la transparencia se muestra en tiempo real la imagen original superpuesta. Para que el usuario pueda interpretar los resultados se añadió un campo de texto que indica la máxima acumulación encontrada en la imagen, de modo que se puede saber a que valor pertenece exactamente el color más cálido de la imagen. Además la imagen que se genera puede ser guardada en formato JPG con su tamaño original gracias a un botón situado en la esquina inferior, conservando los parámetros elegidos tanto de radio como de transparencia. Se puede ver un ejemplo de esta versión en la figura 3.11.

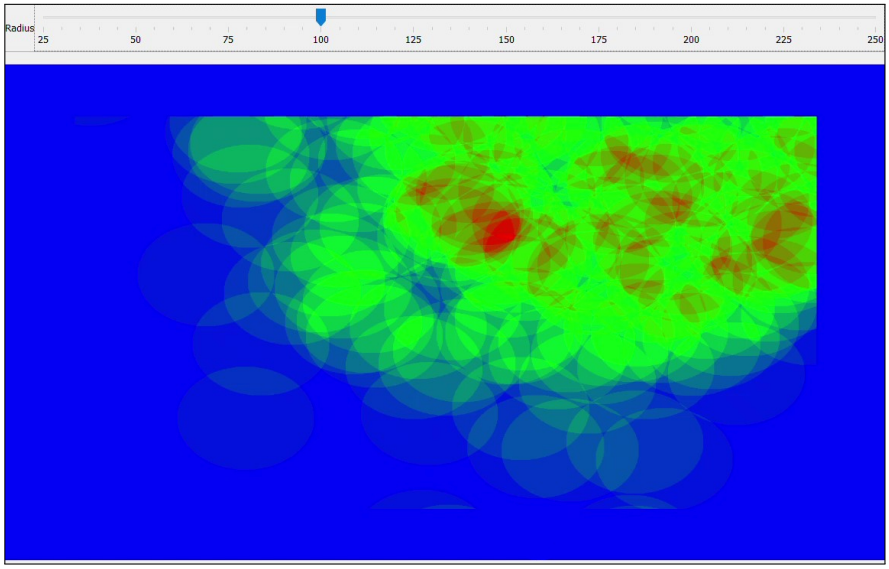

Figura 3.10: Mapa de calor Versión 1

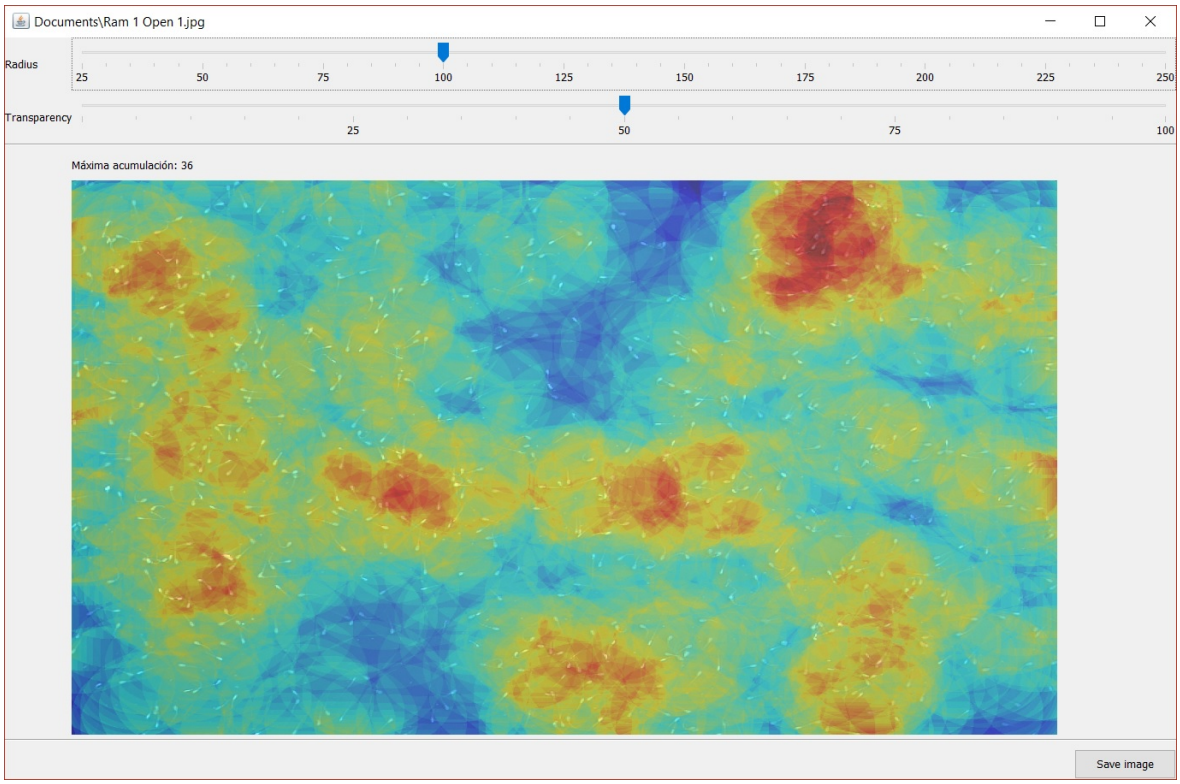

Figura 3.11: Mapa de calor Versión 2

En la versión final se han incorporado pequeños detalles como la escala de la imagen en μm/píxel debajo del radio, de manera que el usuario puede ver en qué escala se ha tomado la imagen que está analizando y saber la correspondencia entre el radio del *Slider* y el radio en μm. También se ha modificado la etiqueta de texto que mostraba la máxima acumulación encontrada para añadir las unidades de las que se trata basándose en el radio y la escala utilizada, así el usuario puede saber la máxima acumulación

encontrada en el radio real (μm). Por último, se incluyó una barra lateral en la parte derecha de la imagen que indica la escala de calor que se está utilizando, en este caso la LUT Jet, y a su lado se representa el porcentaje de espermatozoides en relación al total de los detectados correspondiente a cada color.

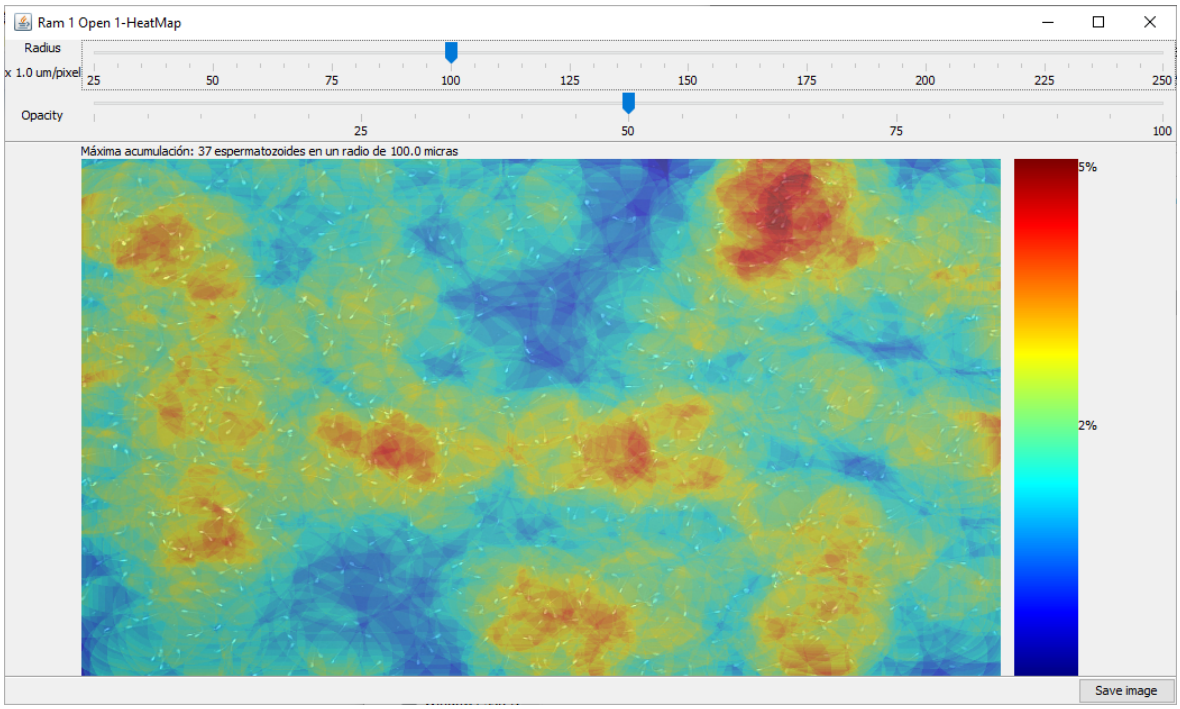

Figura 3.12: Mapa de calor Versión 3

3.3. Mapa de calor en vídeo

La manera de calcular el mapa de calor para un vídeo es análoga a calcular el mapa en una imagen: hay que aplicar el mismo procedimiento descrito anteriormente a cada fotograma del vídeo. Como es un proceso más costoso, especialmente en vídeos con muchos fotogramas, se han incorporado varios parámetros para agilizarlo.

Interpolación de mapa de calor entre varios fotogramas: se calcula un mapa de calor por cada f fotogramas, y para los fotogramas intermedios se interpola el mapa de calor obtenido en el fotograma inicial y final. De este modo el procedimiento de calcular el mapa de calor en un vídeo se acelera en tiempo en un orden de magnitud $O(\frac{F}{f})$, siendo F el total de fotogramas del vídeo y f el rango de fotogramas a interpolar.

Ventana de píxeles: en vez de calcular los espermatozoides más cercanos para cada píxel de la imagen, el usuario puede elegir calcular la concentración para cada ventana de lado W píxeles, siendo W un número natural impar mayor que 1. De este modo se reduce el número de veces que se buscan los N espermatozoides más cercanos en la imagen, ya que para cada ventana se calcula una vez y se rellenan los píxeles

contenidos con el mismo color. El coste del proceso se reduce en un factor de W^2. Como consecuencia de esta aceleración se pierde calidad del vídeo ya que se están pintando ventanas de W píxeles cada vez. Comparando las imágenes en las figuras 3.14 y 3.15 puede apreciarse la diferencia de realizar los cálculos con ventanas de tamaño 1 y 5.

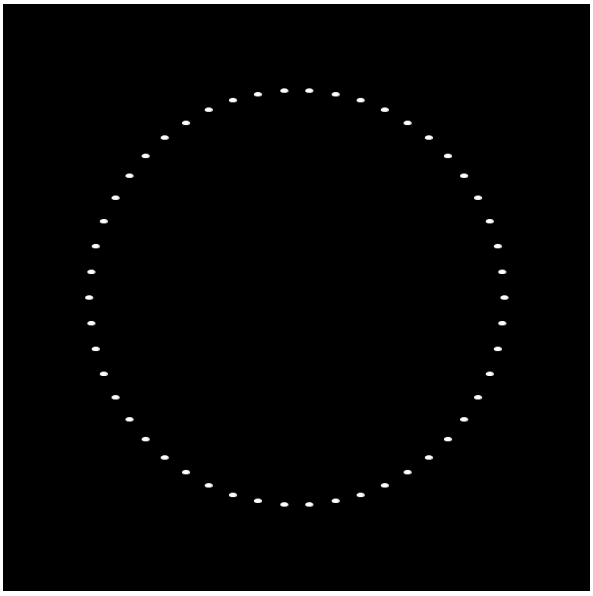

Figura 3.13: Vídeo original

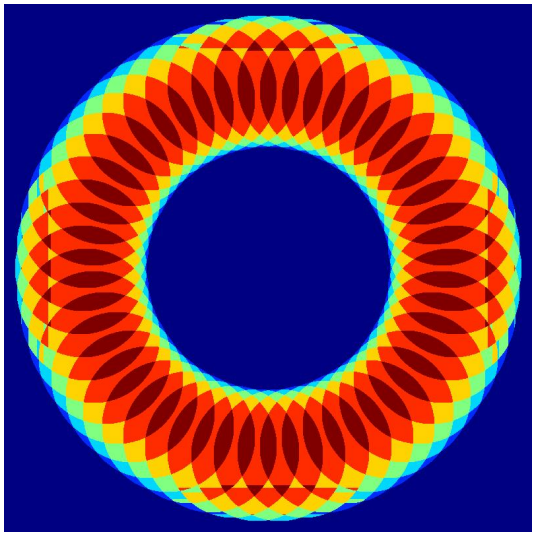

Figura 3.14: Vídeo original con $W = 1$

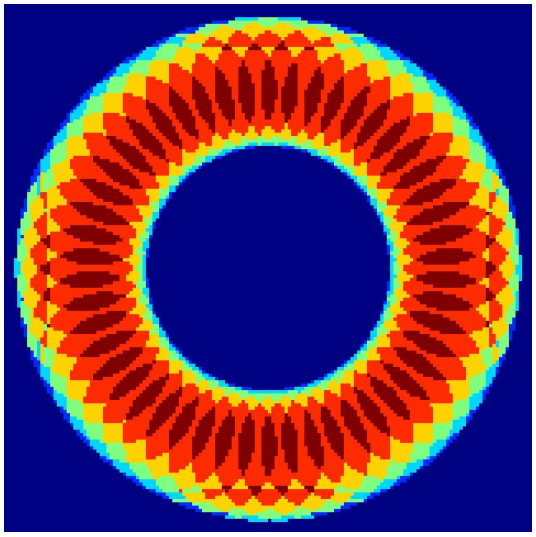

Figura 3.15: Vídeo original con $W = 5$

La tabla 3.1 muestra una comparativa del tiempo del cálculo del mapa de calor en un vídeo de 500 fotogramas con distintos parámetros combinados. Se puede observar la gran reducción de tiempo cuando aumentan los fotogramas a interpolar o cuando se incrementa el tamaño de la ventana.

Tabla 3.1: Comparación parámetros mapa de calor en vídeo

Frames to interpolate	**Window size (pixels)**	**Time (s)**
1	1	233
1	3	37
1	5	22
5	1	49
5	3	12
5	5	9
10	1	28
10	3	10
10	5	8

De la misma forma que para el análisis de imágenes, también se puede ajustar el radio de búsqueda y la opacidad del vídeo original desde los ajustes de la aplicación antes de procesar el vídeo.

Con el objetivo de monitorizar la cantidad de espermatozoides en una zona del vídeo a través del tiempo, se ha añadido la opción de dibujar un círculo en el vídeo procesado para que el programa muestre una tabla con información sobre esa zona para cada fotograma del vídeo. De este modo si el usuario desea saber cuántos espermatozoides se llegan a acumular en la parte de la muestra donde ha sido inyectada la sustancia quimioatrayente, o cuánto han tardado en llegar, simplemente tiene que seleccionar dicha zona con el ratón mientras pulsa la tecla *Shift* para crear un círculo con el radio deseado. A continuación y de manera automática, el programa muestra una ventana con la siguiente información de la zona seleccionada en relación a cada fotograma del vídeo: radio real del círculo en micras, espermatozoides totales y espermatozoides relativos respecto a todos los de la muestra. También se muestra la misma información para dos círculos adicionales concéntricos con el doble y el triple de radio que el círculo original. Así se puede observar cómo varía la cantidad de espermatozoides conforme aumenta el radio del círculo seleccionado. En la tabla 3.2 se puede observar un breve resumen de la información mostrada por el programa al seleccionar una zona para analizar. Cada fila representa un fotograma del vídeo, en la tabla de ejemplo no se muestran todo los fotogramas pero el programa sí que los mostraría. Esta información permite generar gráficas para examinar el movimiento de los espermatozoides hacia la sustancia quimioatrayente inyectada en relación con el tiempo.

Tabla 3.2: Ejemplo de la información mostrada al analizar vídeo

Frame	**Abs**	**Rel**	**Abs x2**	**Rel x2**	**Abs x3**	**Rel x3**
1	3	3 %	5	5 %	17	17 %
50	2	2 %	9	11 %	17	21 %
100	2	2 %	7	9 %	18	24 %
150	1	1 %	6	9 %	12	19 %
200	0	0 %	5	9 %	14	25 %
250	0	0 %	2	4 %	8	18 %
300	0	0 %	2	5 %	6	15 %
350	0	0 %	0	0 %	7	17 %
400	0	0 %	0	0 %	3	8 %
450	0	0 %	0	0 %	4	13 %
499	0	0 %	2	7 %	3	11 %

Capítulo 4

Módulo de Recuento Celular

4.1. Cámara Makler

La cuadrícula de la cámara de recuento Makler® [11] mide 1 x 1 mm y tiene una profundidad de 0.01 mm, por lo que el volumen total de una cámara Makler es 10^{-5} ml. Se divide en 100 cuadros de 0.1 x 0.1 mm cuyo volumen es $V_{cuadro} = 10^{-7}$ ml. En la figura 4.1 se observa el aspecto de la cámara y cómo se vería una muestra de los espermatozoides en el microscopio.

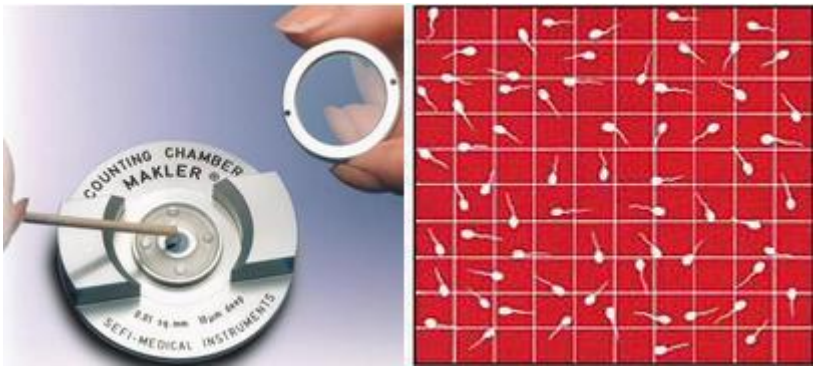

Figura 4.1: Cámara Makler®

Si n es el número total de espermatozoides contados en c cuadros, la concentración se puede calcular a partir de la ecuación 4.1. Habitualmente se cuentan los espermatozoides contenidos en $c = 10$ cuadros, por lo que este recuento n es directamente el valor de la concentración medido en Mespermatozoides/ml.

$$Concentracion = \frac{n}{c \cdot V_{cuadro}} = \frac{n}{c \cdot 10^{-7}\ ml} = \frac{10 \cdot n}{c}\ Mespermatozoides/ml \qquad (4.1)$$

4.2. Recuento celular

Este módulo se ha creado para facilitar la tarea de recuento celular, ya que el proceso de contar los espermatozoides en cada cuadro bajo el microscopio es costoso y propenso a errores, especialmente si hay un elevado número de muestras. El programa ejerce la función del observador y detecta de manera automática tantos cuadros como sea posible y los espermatozoides contenidos en ellos y finalmente, mediante la ecuación 4.1, muestra al usuario la concentración estimada.

Al igual que en el módulo anterior, en primer lugar se realiza un preprocesado de imagen: conversión a escala de grises y ecualización de histograma. A continuación se detectan los espermatozoides mediante el método de *Threshold* RenyiEntropy. Dado que en este caso hay zonas de los bordes de los cuadros que se identifican como espermatozoides, es necesario hacer un post-filtrado para eliminar las células detectadas cuyo porcentaje de diferencia entre ancho y alto sea mayor que 75 % (ecuación 4.2).

$$\%\ diferencia = 100\frac{|ancho - alto|}{max(ancho, alto)} \tag{4.2}$$

Finalmente se detectan los cuadros. En primer lugar se utiliza la función de ImageJ *Find edges*, que al aplicar el filtro Sobel [12] detecta los cambios de gradiente horizontales y verticales y así remarca mejor los bordes de cada cuadro. Finalmente se aplica el Threshold Otsu [8], que es el algoritmo que más cuadros detecta. Las siguientes figuras muestran la imagen de la cámara Makler original y ecualizada (figuras 4.2 y 4.3) junto con los resultados obtenidos al aplicar los dos métodos Threshold utilizados para detectar cuadros y espermatozoides (figuras 4.4 y 4.6) y los contornos detectados (figuras 4.5 y 4.7).

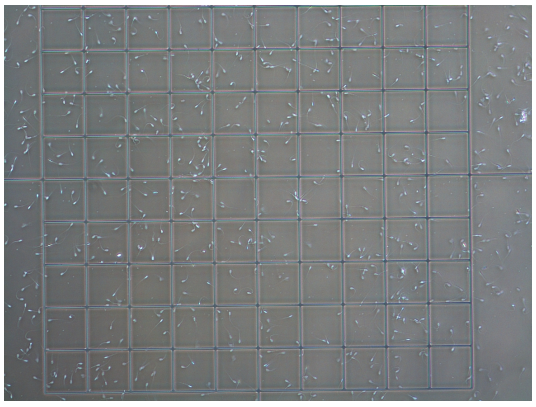

Figura 4.2: Imagen cámara Makler

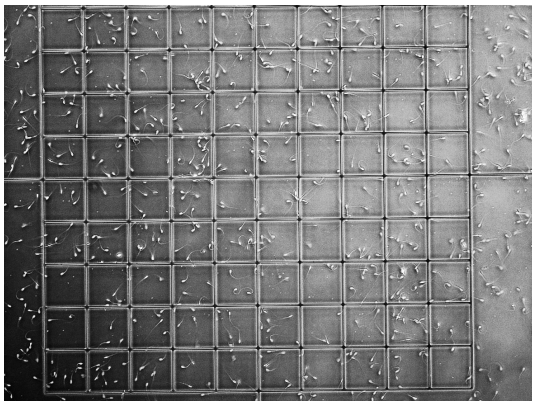

Figura 4.3: Imagen ecualizada

Los espermatozoides detectados se almacenan como en el módulo anterior en un *k-d tree* que permite realizar búsquedas eficientes de los espermatozoides contenidos en

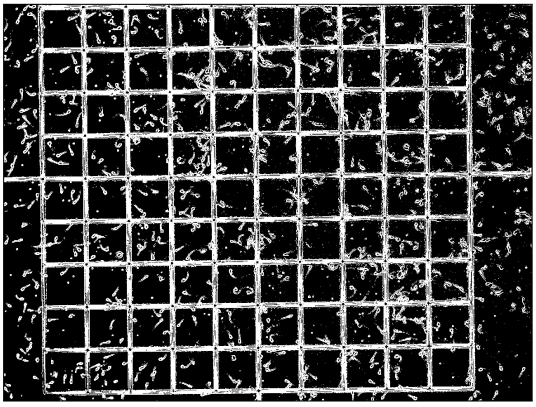

Figura 4.4: Threshold Otsu

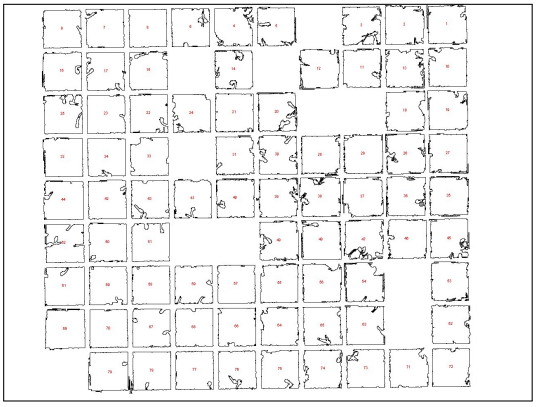

Figura 4.5: Cuadros detectados

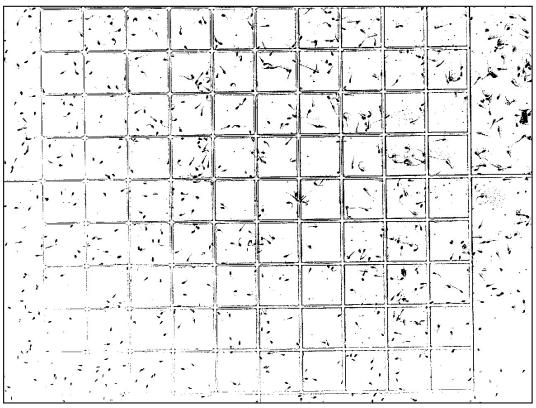

Figura 4.6: Threshold RenyiEntropy

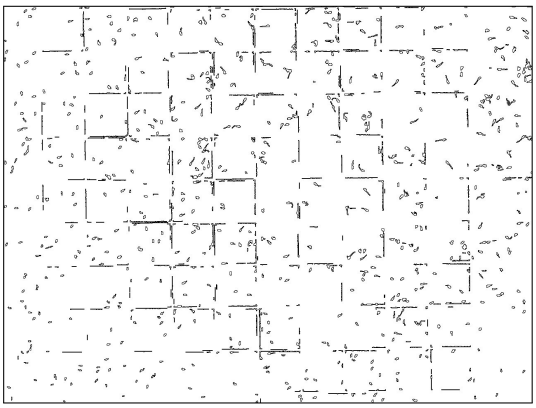

Figura 4.7: Espermatozoides detectados

cada cuadro. Para cada cuadro detectado, se guarda su ubicación en la imagen y sus dimensiones, y todos ellos en una lista enlazada.

Una vez almacenados espermatozoides y cuadros, para cada cuadro se busca en el *k-d tree* los espermatozoides que se encuentran en su interior, basándose en su ubicación y dimensiones. Por último, se obtiene la concentración de la muestra medida usando la ecuación 4.1. En el caso de utilizar una cámara de recuento distinta a la Makler, se pueden cambiar tanto las dimensiones de los cuadros como la profundidad de los mismos desde la ventana de ajustes de la aplicación.

Un *JFrame* de Java muestra de manera resaltada los cuadros que ha detectado la aplicación, así como el número de espermatozoides contenidos en cada uno. Además, una etiqueta informa al usuario de la concentración estimada en Mespermatozoides/ml. En el caso de que la muestra no tenga cuadrícula, la concentración se calcula basándose en las dimensiones (ancho x alto) de la imagen en escala real, la profundidad de la cámara, y el número total de espermatozoides contados. Existen muestras que tienen que ser diluidas antes de poder ser analizadas ya que su alta concentración de espermatozoides dificultaría e incluso imposibilitaría su reconocimiento en la imagen. Por ello se incluye un cuadro de texto donde el usuario puede introducir la dilución de

la muestra actual, que afectaría a la concentración real según la ecuación 4.3, donde *Dilución* es un número real mayor o igual que 1, y cuyo valor 1 se refiere a una muestra no diluida. La figura 4.8 muestra un ejemplo de recuento obtenido a partir de una imagen real.

$$Concentración_{Calculada} = Concentración_{Real} \frac{1}{Dilución} \tag{4.3}$$

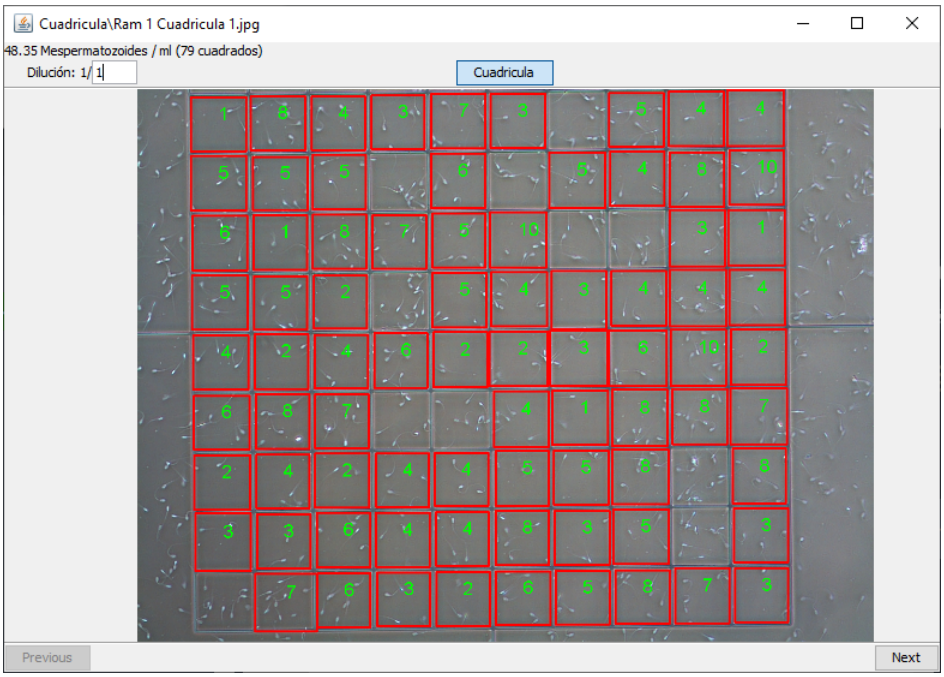

Figura 4.8: Ventana de recuento celular

En una primera versión se pensó en solamente analizar las imágenes de una en una, pero posteriormente y basándose en el funcionamiento de otros módulos se incluyó la posibilidad de analizar un directorio completo, de modo que el usuario pudiera navegar a través de las imágenes contenidas en dicho directorio gracias a dos botones y observar los resultados de concentración obtenidos en cada una de ellas. Además ha sido añadido un botón que permite mostrar u ocultar el número de espermatozoides y los cuadros dibujados sobre la imagen original.

Capítulo 5

Resultados y Validación

5.1. Módulo de Acumulación Celular

Con el objetivo de validar este módulo se eligieron varias muestras de espermatozoides y se compararon con sus respectivos mapas de calor. Así se comprobó que efectivamente las zonas con colores más cálidos se corresponden con las partes de la muestra donde hay más espermatozoides.

En primer lugar se validaron las muestras que tienen zonas donde hay más espermatozoides acumulados. Para ello se utilizó la imagen mostrada en la figura 5.1, de la cual se obtuvo el mapa de calor con un radio de 75 píxeles y marcaron las zonas donde se puede apreciar a simple vista mayor acumulación de espermatozoides. En el mapa de calor mostrado en la figura 5.2 se puede observar que las zonas con colores más cálidos concuerdan con las zonas marcadas en la imagen original.

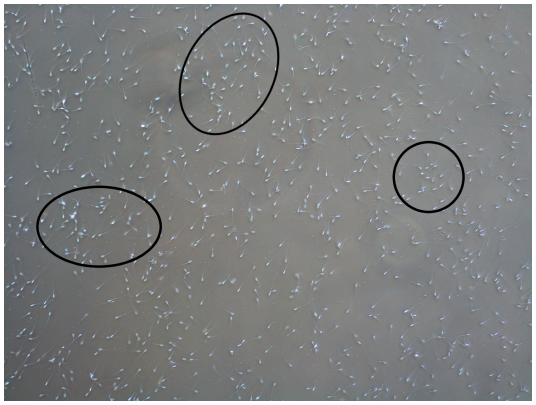

Figura 5.1: Muestra 1

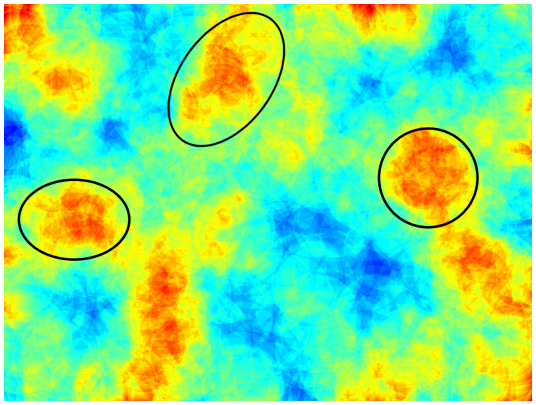

Figura 5.2: Mapa de calor de la muestra 1

En segundo lugar se analizó de manera análoga una muestra con más zonas donde la acumulación de espermatozoides es baja (figura 5.3) y se marcaron las zonas donde se observa a simple vista esos huecos en la imagen. El mapa de calor se calculó con un radio de 200 píxeles, se muestra en la imagen de la figura 5.4. Se puede observar que las zonas marcadas en la imagen original concuerdan con las zonas con colores más fríos

del mapa de calor.

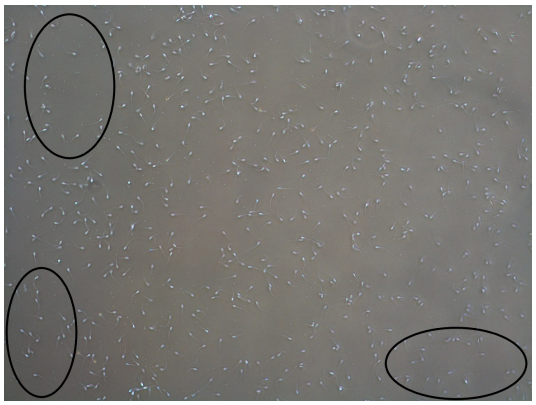

Figura 5.3: Muestra 2

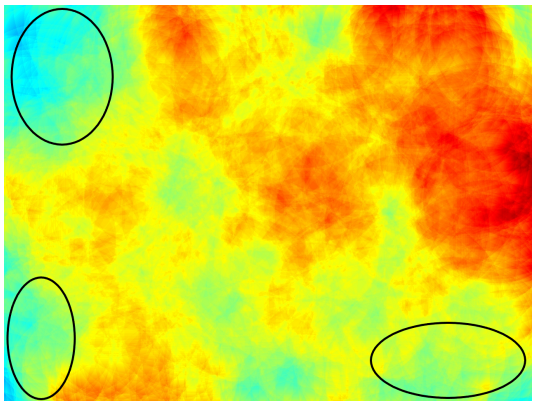

Figura 5.4: Mapa de calor de la muestra 2

5.2. Módulo de Recuento Celular

La validación del módulo de recuento celular se ha efectuado comparando las concentraciones obtenidas por cada versión de OpenCASA con las obtenidas a partir de recuentos manuales efectuados por una investigadora del grupo BIOFITER. Se han analizado 11 muestras captadas por una cámara Makler.

La tabla 5.1 recoge los datos correspondientes a los recuentos realizados por la primera versión del módulo OpenCASA (la imagen no se ecualiza y no se le aplica el filtro de Sobel) y los recuentos manuales. Para cada muestra considerada, se recogen los cuadros detectados por OpenCASA, la concentración obtenida por OpenCASA, la concentración calculada a partir del recuento manual de espermatozoides en los mismos cuadros detectados por OpenCASA, y la diferencia relativa entre ambas estimaciones. La unidad de medida de las concentraciones es millones de espermatozoides por mililitro (Mesp/ml). La diferencia relativa se calcula de la siguiente manera:

$$Diferencia = 100 \cdot \frac{OpenCASA - Manual}{Manual} \tag{5.1}$$

El número medio de cuadros detectados es de 25 sobre un máximo de 100. La mayoría están localizados en la misma zona de la imagen. Esto puede ser debido a que una iluminación no uniforme afecta a la detección de cuadros. Los valores de concentración obtenidos por OpenCASA son menores que los estimados a partir del recuento manual para todas las muestras. El valor medio de las diferencias relativas[1] es 45.8 %.

[1]El valor medio de las diferencias relativas es el resultado de promediar los valores absolutos de las diferencias relativas.

Tabla 5.1: Comparación de concentraciones obtenidas por OpenCASA (versión 1) y por el recuento manual.

Muestra	**Cuadros detectados**	**OpenCASA** (Mesp/ml)	**Manual** (Mesp/ml)	**Diferencia** (%)
1	26	35.0	50.0	-30.0
2	23	36.0	44.3	-18.8
3	23	38.2	51.3	-25.6
4	11	27.7	41.8	-34.8
5	12	29.8	45.0	-35.2
6	19	32.5	61.0	-46.7
7	37	11.4	41.6	-72.5
8	35	14.3	36.3	-60.6
9	33	19.7	39.4	-50.0
10	25	14.29	68.5	-79.1
11	27	27.41	55.9	-51.0

En la figura 5.5 se representan los valores de las estimaciones de concentración obtenidas por OpenCASA y por el recuento manual (*MANUAL N*). La línea marcada como *MANUAL 10* representa el recuento estándar de 10 cuadros que suele realizarse en la cámara Makler (*MANUAL N*).

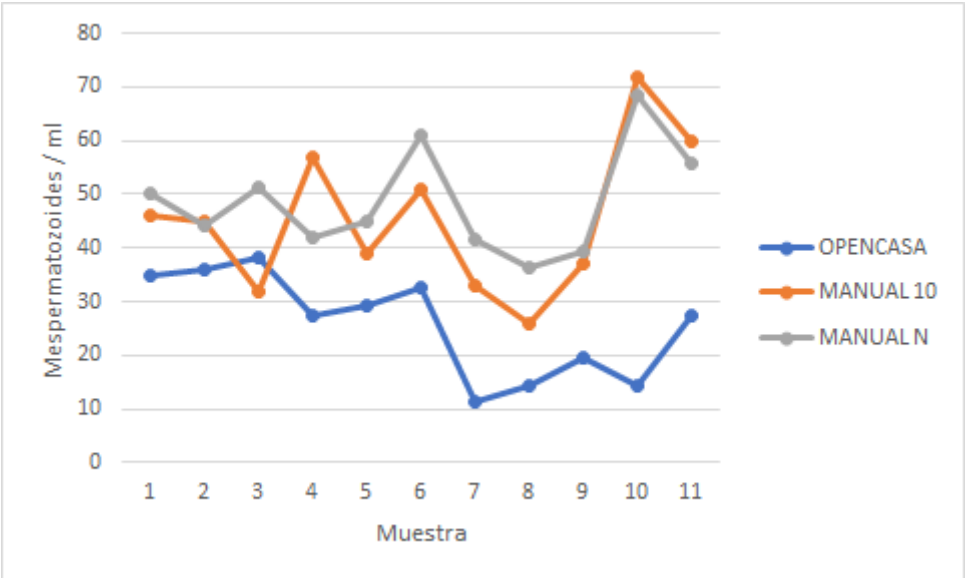

Figura 5.5: Comparación de las concentraciones obtenidas por OpenCASA (versión 2) y por dos recuentos manuales.

La tabla 5.2 corresponde a la segunda versión del módulo OpenCASA, donde la imagen está ecualizada y se ha aplicado el filtro Sobel. Para cada muestra, se tienen las concentraciones estimadas y los cuadros utilizados por OpenCASA y por el recuento manual, y la diferencia relativa entre ambas estimaciones. El número medio de cuadros detectados es de 79 sobre un máximo de 100. El preprocesado de imagen ha mejorado

notablemente la detección de los cuadros. En cuanto a las concentraciones, el valor medio de las diferencias ha disminuido al 33.5 %. Tanto en la tabla 5.2 como en la gráfica de la figura 5.6 se observa que la concentración detectada por OpenCASA supera la de los recuentos manuales en todos las muestras. Esto se debe a que el programa está detectando como espermatozoides ciertas zonas de los bordes de la cuadrícula.

Tabla 5.2: Comparación de concentraciones obtenidas por OpenCASA (versión 2) y por el recuento manual.

Muestra	**OpenCASA**		**Manual**		**Diferencia**
	Concentración (Mesp/ml)	**Cuadros detectados**	**Concentración** (Mesp/ml)	**Cuadros considerados**	(%)
1	56,7	79	50.0	26	13,4
2	58,9	82	44.3	23	32,8
3	61,8	82	51.3	23	20,5
4	74,5	74	41.8	11	78,1
5	71,7	78	45.0	12	59,3
6	77,4	76	61.0	19	26,8
7	50,6	87	41.6	37	21,5
8	51,4	88	36.3	35	41,6
9	43,7	87	39.4	33	10,9
10	80.2	63	68.5	25	17,0
11	81.8	68	55.9	27	46,2

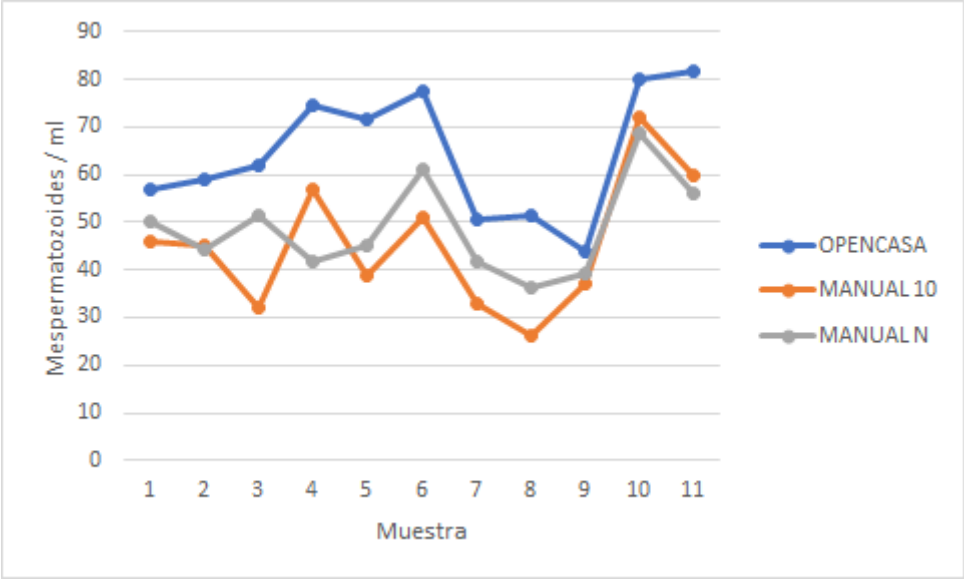

Figura 5.6: Comparación de las concentraciones obtenidas por OpenCASA (versión 2) y por dos recuentos manuales

La tabla 5.3 muestra los resultados obtenidos con la tercera versión del módulo OpenCASA, en la que se realiza una fase de post-filtrado que evita que los bordes de los cuadros se detecten como espermatozoides. La estructura de la tabla es la misma

que la anterior. El valor medio de las diferencias ha sido reducido de forma significativa hasta el 14.6 %. Se observa una gran variabilidad en las diferencias: la menor es del 1.3 % y la mayor del 41.8 %.

Tabla 5.3: Comparación de concentraciones obtenidas por OpenCASA (versión 3) y por el recuento manual.

Muestra	**OpenCASA**		**Manual**		**Diferencia**
	Concentración (Mesp/ml)	**Cuadros detectados**	**Concentración** (Mesp/ml)	**Cuadros considerados**	(%)
1	48,4	79	50.0	26	-3,3
2	48,9	82	44.3	23	10,3
3	52,0	82	51.3	23	1,3
4	62,6	74	41.8	11	49,7
5	61,9	78	45.0	12	37,6
6	65,1	76	61.0	19	6,8
7	42,4	87	41.6	37	1,9
8	42,8	88	36.3	35	18,1
9	38,2	87	39.4	33	-3,1
10	72,4	63	68.5	25	5,7
11	69,0	68	55.9	27	23,3

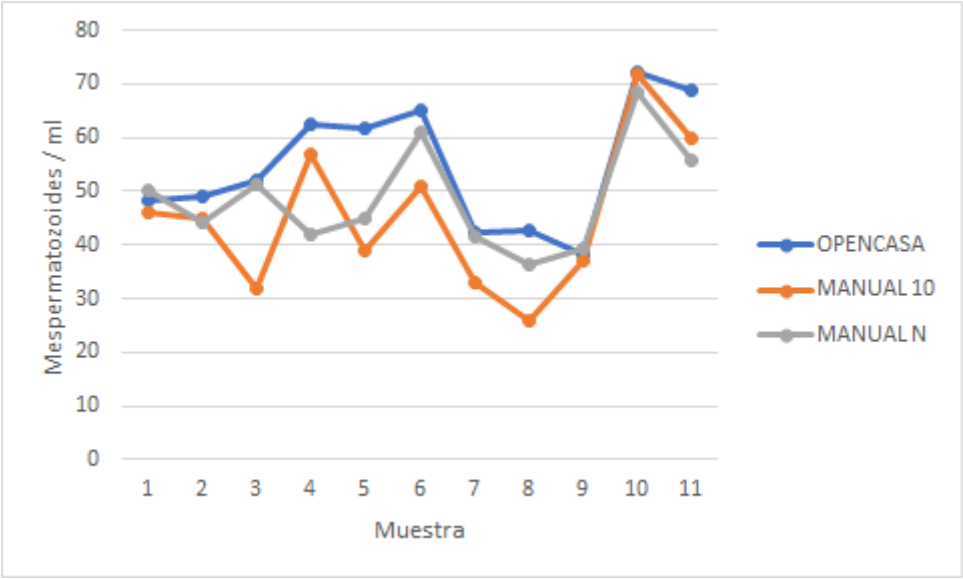

Figura 5.7: Comparación de las concentraciones obtenidas por OpenCASA (versión 3) y por dos recuentos manuales

Se ha creado un gráfico de Bland-Altman [13] (figura 5.8) para observar la correlación entre el recuento manual y la tercera versión de OpenCASA. Se pueden ver dos líneas verdes que representan los límites de concordancia, que son equivalentes a $Mean \pm 1{,}96SD$, siendo SD la desviación estándar del *bias* de cada muestra. Todos

los valores del gráfico están situados entre estos límites y la variabilidad es consistente en torno a la línea de la media y no se detectan anomalías.

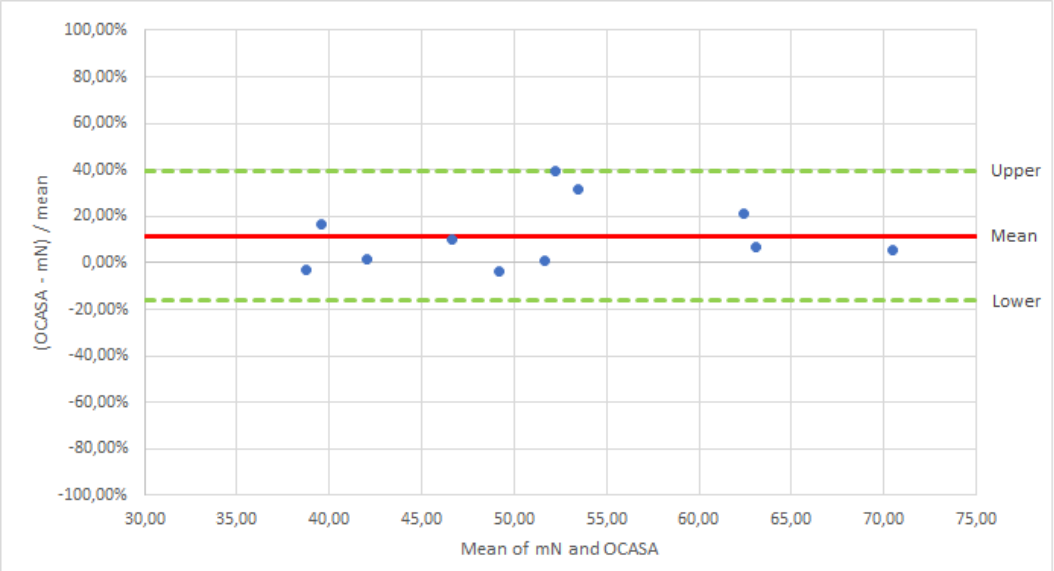

Figura 5.8: Test de Bland-Altman

La tabla 5.4 muestra un resumen de las tres versiones que contiene la diferencia media relativa, la raíz de la desviación cuadrática media (RDCM) y el test de Bland-Altman [13] para cuantificar la diferencia del recuento por observador y OpenCASA.

Tabla 5.4: Resumen de las diferencias de concentración entre las tres versiones OpenCASA y el recuento manual considerando los cuadros detectados por la versión 1: diferencia relativa media, RDCM (raíz de la desviación cuadrática media) y Bland-Altman Bias.

	Diferencia (%)	**RDCM**	**Bland-Altman Bias (%)**
Versión 1	45.8	25.7	-63.45
Versión 2	33.5	17.9	27.48
Versión 3	14.6	9.5	11.58

Con el objetivo de validar con más detalle el programa, los investigadores del grupo BIOFITER han realizado un recuento completo de la cuadrícula para dos muestras. De esta manera se puede realizar una comparación detallada del recuento obtenido cuadro a cuadro por el módulo de recuento celular. En las figuras 5.9 y 5.10 se observa la diferencia de espermatozoides contados para cada una de las muestras entre los observadores y el programa. Los cuadros que no contienen ningún número, en fondo blanco, no han sido detectados por el programa. Destacar que el recuento coincide en 19 y 29 cuadros de las muestras 1 y 2, respectivamente.

Al realizar el recuento manual sobre la cuadrícula entera, cambia la estimación de la concentración respecto al recuento realizado anteriormente sobre un número reducido de cuadros (25 en media), y con ellas, las diferencias con las concentraciones calculadas. En concreto, la estimación para la muestra 1 pasa de 50.0 a 45.7 Mesp/ml, con lo que la diferencia respecto OpenCASA cambia del -3.3 % al 5.9 %. La estimación de la muestra 2 cambia de 44.3 a 44.6, y la diferencia respecto OpenCASA pasa del 10.3 % al 8.9 %.

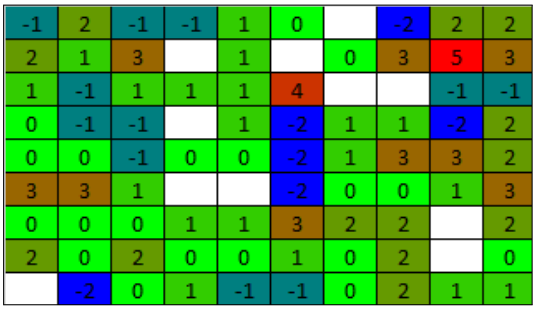

Figura 5.9: Diferencias de los recuentos cuadro a cuadro para la muestra 1

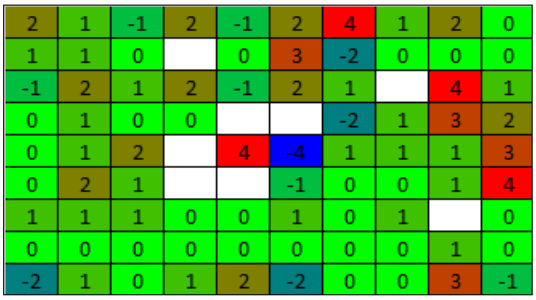

Figura 5.10: Diferencias de los recuentos cuadro a cuadro para la muestra 2

Capítulo 6

Conclusiones

En este trabajo de final de grado se han ampliado las funcionalidades del programa OpenCASA con la implementación de dos nuevos módulos. El módulo de acumulación celular permite al usuario analizar de una forma más visual mediante el uso de un mapa de calor la respuesta de los espermatozoides ante una sustancia quimioatrayente o quimiorrepelente. El módulo de recuento celular estima la concentración medida en espermatozoides/ml a partir de una muestra con o sin cuadrícula.

Ambos módulos han sido validados por separado, lo que ha permitido detectar errores que se han corregido. El módulo de acumulación ha sido validado mediante la observación de las zonas de la muestra con más/menos espermatozoides acumulados y su correspondencia en el mapa de calor con las zonas más cálidas/frías. En la validación del módulo de recuento celular se ha contado con la colaboración de una investigadora del grupo BIOFITER que ha realizado varios recuentos manuales que han sido comparados con las estimaciones de OpenCASA. Para una serie de 11 muestras, la diferencia entre el recuento manual de un número reducido de cuadros y el realizado por OpenCASA es del 14.6 %. Aunque pueda parecer un valor elevado, hay que tener en cuenta que la aplicación añade precisión al conteo, debido a que el número de cuadros contados es casi la totalidad de la cuadrícula frente a 10 cuadros del conteo manual. En dos muestras el observador ha contado los espermatozoides de todos cuadros de la cuadrícula y el error medio de ambas con respecto a OpenCASA es reducido al 7.4 %. Además con el test de Bland-Altman se ha comprobado que ambos métodos están muy relacionados obteniendo una diferencia en porcentaje o *bias* del -11.58 %.

Se está escribiendo un artículo para la revista *Computational and Structural Biotechnology Journal*, publicación científica de la editorial Elsevier que cubre temas de biología computacional y estructural, y cuyo factor de impacto es de 4.418 (2017). En este artículo se describirán los nuevos módulos de OpenCASA presentados en este trabajo. También van a estar disponibles los nuevos módulos en el repositorio de la plataforma Github de OpenCASA, `https://github.com/calquezar/OpenCASA`, de

esta manera se permite a los investigadores descargar el software y también contribuir para posibles futuras mejoras.

En cuanto a conocimientos adquiridos, cabe destacar la introducción de conceptos de biología molecular tales como el análisis de la motilidad espermática o el cálculo de la concentración celular mediante el uso de la cámara Makler®. También he aprendido a implementar plugins en la aplicación de código abierto ImageJ, así como el uso de sus librerías internas para desarrollar los nuevos módulos de OpenCASA.

También ha sido interesante la labor de coordinación con científicos que trabajan en un campo distinto del informático como es el campo de la biología molecular. En el caso de este trabajo y los módulos que se han implementado, se facilita la labor de los investigadores a la hora de analizar la respuesta de los espermatozoides a una sustancia quimioatrayente. Además con el módulo de recuento, se ha mejorado la manera de estimar la concentración celular, pasando del trabajo de un observador que contaba de manera manual los espermatozoides de 10 cuadros, a un programa que de manera instantánea estima la concentración de una muestra contando todos cuadros posibles, lo que mejora la precisión del cálculo.

Mi valoración personal es bastante positiva, ya que me ha gustado mucho trabajar en este tema y me ha resultado muy interesante saber como la informática puede aportar en diversos campos de investigación como en este caso la biología molecular. Pienso que los módulos implementados pueden suponer una mejora para los investigadores de este campo, ya que es una alternativa gratuita frente a los programas comerciales de elevado precio, además al ser de código abierto, cualquiera puede editar el programa a su gusto añadiendo posibles mejoras o modificando su funcionamiento. Por otra parte, el artículo que se va a redactar para la revista *Computational and Structural Biotechnology Journal* puede ser para mí una oportunidad para ganar reconocimiento de cara a mi futuro laboral.

Bibliografía

[1] Anat Bahat and Michael Eisenbach. Sperm thermotaxis. *Molecular and cellular endocrinology*, 252(1-2):115–119, 2006.

[2] Sergii Boryshpolets, Serafín Pérez-Cerezales, and Michael Eisenbach. Behavioral mechanism of human sperm in thermotaxis: a role for hyperactivation. *Human Reproduction*, 30(4):884–892, 2015.

[3] Kiyoshi Miki and David E Clapham. Rheotaxis guides mammalian sperm. *Current Biology*, 23(6):443–452, 2013.

[4] Leah Armon and Michael Eisenbach. Behavioral mechanism during human sperm chemotaxis: involvement of hyperactivation. *PloS one*, 6(12):e28359, 2011.

[5] Carlos Alquézar-Baeta, Silvia Gimeno-Martos, Sara Miguel-Jiménez, Pilar Santolaria, Jesús Yániz, Inmaculada Palacín, Adriana Casao, José Álvaro Cebrián-Pérez, Teresa Muiño-Blanco, and Rosaura Pérez-Pé. OpenCASA: A new open-source and scalable tool for sperm quality analysis. *PLoS computational biology*, 15(1):e1006691, 2019.

[6] Johannes Schindelin, Curtis T Rueden, Mark C Hiner, and Kevin W Eliceiri. The ImageJ ecosystem: An open platform for biomedical image analysis. *Molecular reproduction and development*, 82(7-8):518–529, 2015.

[7] Caroline A Schneider, Wayne S Rasband, and Kevin W Eliceiri. NIH Image to ImageJ: 25 years of image analysis. *Nature methods*, 9(7):671, 2012.

[8] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[9] Jagat Narain Kapur, Prasanna K Sahoo, and Andrew KC Wong. A new method for gray-level picture thresholding using the entropy of the histogram. *Computer vision, graphics, and image processing*, 29(3):273–285, 1985.

[10] Thomas Abeel, Yves Van de Peer, and Yvan Saeys. Java-ML: A machine learning library. *Journal of Machine Learning Research*, 10(Apr):931–934, 2009.

[11] W Cardona-Maya, J Berdugo, and A Cadavid. Comparación de la concentración espermática usando la cámara de Makler y la cámara de Neubauer. *Actas urológicas españolas*, 32(4):443–445, 2008.

[12] Irwin Sobel and Gary Feldman. A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in*, pages 271–272, 1968.

[13] J Martin Bland and Douglas G Altman. Measuring agreement in method comparison studies. *Statistical methods in medical research*, 8(2):135–160, 1999.

Lista de Figuras

Lista de Tablas

Anexos

Anexo A

Código

A.1. MainWindow.java

```
package gui;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.HeadlessException;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.image.IndexColorModel;
import java.io.IOException;
import java.net.URL;
import java.util.Scanner;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

import analysis.Accumulation;
import analysis.Chemotaxis;
import analysis.Motility;
import data.AccumulationParams;
import data.CellCountParams;
import data.ChemotaxisParams;
import data.MorphometryParams;
import data.MotilityParams;
import data.Params;
import data.PersistentRandomWalker;
import data.Simulation;
import data.ViabilityParams;
import ij.IJ;
```

```java
import ij.gui.GenericDialog;
import ij.plugin.LutLoader;
import ij.process.LUT;

/**
 * This window shows all functional modules available.
 *
 * @author Carlos Alquezar and Jorge Yagüe
 */
public class MainWindow extends JFrame {

   /**
    *
    */
   private static final long serialVersionUID = 1L;
   /**
    * @brief Self reference used in action listeners to show and hide main
↪   window.
    */
   private MainWindow mw;

   /**
    * @brief Constructor. The main graphical user interface is created.
    * @param title - String that is used as window's title
    */
   public MainWindow(String title) throws HeadlessException {
      super(title);
      createGUI();
      this.setPreferredSize(new Dimension(600, 400));
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      this.pack();
      this.setVisible(true);
      this.setLocationRelativeTo(null);
      mw = this;

      MotilityParams.resetParams();
      ChemotaxisParams.resetParams();
      ViabilityParams.resetParams();
      MorphometryParams.resetParams();
      AccumulationParams.resetParams();
      CellCountParams.resetParams();
   }

   /**
    * @brief This method add to the given JPanel a button with the specified
    *        parameters
    * @param label      - String that is shown as button's label
    * @param gridx      - relative layout's x location
    * @param gridy      - relative layout's y location
    * @param background - Background color
    * @param iconPath   - Path relative to the icon's image
```

```
     * @param panel      - panel where the button is going to be added
     */
    private void addButton(final String label, int gridx, int gridy, Color
    ↪  background, String iconPath, JPanel panel) {

        GridBagConstraints c = new GridBagConstraints();
        c.weightx = 0.5;
        c.fill = GridBagConstraints.HORIZONTAL;
        c.ipady = 0;
        c.gridx = gridx;
        c.gridy = gridy;
        JButton btn = new JButton(label);
        btn.setBackground(background);
        try {
            // Image img =
            // ImageIO.read(getClass().getResource("/resources/motility.png"));
            Image img = ImageIO.read(getClass().getResource(iconPath));
            btn.setIcon(new ImageIcon(img));
        } catch (Exception ex) {
            // IJ.handleException(ex);
            // System.out.println(ex);
        }
        // Add action listener
        btn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                if (label.equals("Chemotaxis")) {
                    Chemotaxis ch = new Chemotaxis();
                    try {
                        ChemotaxisParams.setGlobalParams();
                        // ChemotaxisParams.printParams();
                        // Params.printParams();
                        mw.setVisible(false);
                        ch.selectAnalysis();// this method has to be run outside
                        ↪  the
                                        // execute() method because it
                                        // is a GUI method and has to be run on the
                                        ↪  EDT
                        ch.execute();
                        mw.setVisible(true);
                    } catch (Exception e1) {
                        IJ.handleException(e1);
                    }
                } else if (label.equals("Motility")) {
                    Motility mot = new Motility();
                    try {
                        MotilityParams.setGlobalParams();
                        // MotilityParams.printParams();
                        // Params.printParams();
                        mw.setVisible(false);
                        mot.selectAnalysis();
                        mot.execute();
```

```
            mw.setVisible(true);
        } catch (Exception e1) {
            IJ.handleException(e1);
        }
    } else if (label.equals("Viability")) {
        ViabilityParams.setGlobalParams();
        // ViabilityParams.printParams();
        // Params.printParams();
        mw.setVisible(false);
        ViabilityWindow viabilityW = new ViabilityWindow();
        viabilityW.addWindowListener(new
        ↪  java.awt.event.WindowAdapter() {
            @Override
            public void windowClosing(java.awt.event.WindowEvent
            ↪  windowEvent) {
                if (mw != null)
                    mw.setVisible(true);
            }
        });
        int out = viabilityW.run();
        if (out < 0) {
            mw.setVisible(true);
        }
    } else if (label.equals("Morphometry")) {
        MorphometryParams.setGlobalParams();
        // MorphometryParams.printParams();
        // Params.printParams();
        mw.setVisible(false);
        MorphWindow morphW = new MorphWindow();
        morphW.addWindowListener(new java.awt.event.WindowAdapter() {
            @Override
            public void windowClosing(java.awt.event.WindowEvent
            ↪  windowEvent) {
                if (mw != null)
                    mw.setVisible(true);
            }
        });
        int out = morphW.run();
        if (out < 0) {
            mw.setVisible(true);
        }
    } else if (label.equals("Accumulation")) {
        LUT lut = getLut(getClass().getResource("/Jet.lut"));
        Accumulation ac = new Accumulation(lut);
        try {
            AccumulationParams.setGlobalParams();
            mw.setVisible(false);
            ac.selectAnalysis();
            ac.execute();
            mw.setVisible(true);
        } catch (Exception e1) {
```

```
                IJ.handleException(e1);
            }
        } else if (label.equals("Cell Count")) {
            CellCountParams.setGlobalParams();
            mw.setVisible(false);
            CellCountWindow ccW = new CellCountWindow();
            ccW.addWindowListener(new java.awt.event.WindowAdapter() {
                @Override
                public void windowClosing(java.awt.event.WindowEvent
                ↪  windowEvent) {
                    if (mw != null) {
                        mw.setVisible(true);
                    }
                }
            });
            int out = ccW.run();
            if (out < 0) {
                mw.setVisible(true);
            }
        } else if (label.equals("Simulation")) {
            simulate();
        } else if (label.equals("Settings")) {
            SettingsWindow sw = new SettingsWindow("Settings");
            sw.addWindowListener(new java.awt.event.WindowAdapter() {
                @Override
                public void windowClosing(java.awt.event.WindowEvent
                ↪  windowEvent) {
                    if (mw != null)
                        mw.setVisible(true);
                }
            });
            mw.setVisible(false);
            // sw.run();
        }
    }
});
panel.add(btn, c);
}

/**
 * @brief This method creates the main user interface.
 */
private void createGUI() {
    String parentDir = "";
    // String parentDir = "/resources";
    JPanel panel = new JPanel(new GridBagLayout());
    addButton("Motility", 0, 0, new Color(255, 255, 255), parentDir +
    ↪  "/motility.png", panel);
    addButton("Chemotaxis", 1, 0, new Color(255, 255, 255), parentDir +
    ↪  "/chemotaxis.png", panel);
```

```
        addButton("Viability", 0, 1, new Color(255, 255, 255), parentDir +
        ↪  "/viability.png", panel);
        addButton("Morphometry", 1, 1, new Color(255, 255, 255), parentDir +
        ↪  "/morphometry.png", panel);
        addButton("Accumulation", 0, 2, new Color(255, 255, 255), parentDir +
        ↪  "/accumulation.png", panel);
        addButton("Cell Count", 1, 2, new Color(255, 255, 255), parentDir +
        ↪  "/mackler.png", panel);
        addButton("Simulation", 0, 3, new Color(255, 255, 255), parentDir +
        ↪  "/settings.png", panel);
        addButton("Settings", 1, 3, new Color(255, 204, 153), parentDir +
        ↪  "/settings.png", panel);
        this.setContentPane(panel);
    }

    /**
     * @brief Shows a Generic Dialog to ask user which simulation parameters
↪   have to
     *        be used for the simulation.
     */
    private void simulate() {
        GenericDialog gd = new GenericDialog("Set Simulation parameters");
        gd.addNumericField("Beta", 0, 2);
        gd.addNumericField("Responsiveness (%)", 50, 2);
        gd.addNumericField("Length of the simulation (frames)", 500, 0);
        gd.showDialog();
        if (gd.wasCanceled())
            return;
        double beta = gd.getNextNumber();
        double responsiveness = gd.getNextNumber() / 100; // value must be
        ↪  between
                                                  // [0,1]
        int length = (int) gd.getNextNumber();
        Simulation sim = new PersistentRandomWalker(beta, responsiveness,
        ↪  length);
        try {
            sim.run();
        } catch (Exception e1) {
            IJ.handleException(e1);
            // e1.printStackTrace();
        }
    }

    /**
     *
     * @param url
     * @return
     */
    private LUT getLut(URL url) {
        byte r[] = new byte[256], g[] = new byte[256], b[] = new byte[256];
        try {
```

```
            Scanner sc = new Scanner(url.openStream());

            while (sc.hasNextInt()) {
               int i = sc.nextInt();
               r[i] = (byte) sc.nextInt();
               g[i] = (byte) sc.nextInt();
               b[i] = (byte) sc.nextInt();
            }

            sc.close();
         } catch (Exception e) {
            IJ.handleException(e);
         }
         return new LUT(r, g, b);
      }

}
```

A.2. Accumulation.java

```
package analysis;

import java.awt.Color;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.Arrays;
import java.util.List;

import javax.swing.SwingWorker;

import data.Cell;
import data.Params;
import functions.ComputerVision;
import functions.FileManager;
import functions.Utils;
import functions.VideoRecognition;
import gui.AccumulationWindow;
import ij.IJ;
import ij.ImagePlus;
import ij.ImageStack;
import ij.gui.ImageCanvas;
import ij.gui.ImageRoi;
import ij.gui.ImageWindow;
import ij.gui.Roi;
import ij.measure.ResultsTable;
import ij.process.ImageProcessor;
import ij.process.LUT;
import net.sf.javaml.core.kdtree.KDTree;
```

```java
/**
 * This class implements all the functions related to accumulation analysis.
 *
 * @author Jorge Yagüe
 */
public class Accumulation extends SwingWorker<Boolean, String> {

   /**
    * Constructor. Initialize LUT with file named "Jet.lut"
    */
   public Accumulation(LUT lut) {
      try {
         Accumulation.lut = lut;
      } catch (Exception ex) {
         IJ.handleException(ex);
      }
   }

   private static LUT lut = null;

   private enum TypeOfAnalysis {
      IMAGE, VIDEO, NONE
   }

   private TypeOfAnalysis analysis = TypeOfAnalysis.NONE;

   private static int MAX = 0;

   /**
    * This method counts spermatozoa located in the circle with center (x,y)
↪   and
    * radius r, which are contained in tree
    *
    * @param x
    * @param y
    * @param r
    * @param tree
    * @return número de esperamtozoides
    */
   private static int contar(int x, int y, int r, KDTree tree) {
      Object[] nearest0 = tree.range(new double[] { x - r, y - r }, new
↪      double[] { x + r - 1, y + r - 1 });
      if (nearest0.length > 0) {
         int n = nearest0.length;
         // Se filtran las que están fuera del círculo de radio r
         Cell[] nearest = Arrays.copyOf(nearest0, n, Cell[].class);
         int dist2 = r * r;
         for (Cell c : nearest) {
            int dist = ((int) c.x - x) * ((int) c.x - x) + ((int) c.y - y) *
↪              ((int) c.y - y);
```

```
            n -= dist < dist2 ? 0 : 1;
         }
         return n;
      } else {
         return 0;
      }
   }

   /**
    * This method returns the area of the circumference sector with radius r
↪   and
    * height h
    *
    * @param r
    * @param h
    * @return
    */
   private static double areaSector(int r, int h) {
      double alpha = 2 * Math.acos(1 - (double) (r - h) / r);
      return 0.5 * r * r * (alpha - Math.sin(alpha));
   }

   /**
    * This method returns the integral which is defined by circumference
↪   function
    * between "from" to 0
    *
    * @param r    circumference radius
    * @param a    circumference center
    * @param b    circumference center
    * @param from
    * @return
    */
   private static double areaInt(int r, int a, int b, double from) {
      // (x - a)^2 + (y - b)^2 = r^2
      // y = sqrt(r^2 - (x - a)^2) + b
      return -((from - a) * Math.sqrt(r * r - from * from + 2 * a * from - a
       ↪  * a) + a * Math.sqrt(r * r - a * a)
            + r * r * (Math.asin((from - a) / r) + Math.asin(a / r)) + 2 * b
             ↪  * from) / 2;
   }

   /**
    * This method returns a matrix[width][height] with the accumulation in
↪   pixel
    * window W with radius r
    *
    * @param width
    * @param height
    * @param r
```

```
     * @param tree
     * @param debug
     * @return
     */
    public static int[][] getAccumulation(int width, int height, int r, int
    ↪  W, KDTree tree, boolean debug) {
        if (debug)
            IJ.showStatus("Analizing accumulation...");
        int AC[][] = new int[width][height];
        MAX = 0;

        double A = r * r * Math.PI;
        int l = W / 2;
        boolean div1 = r % W == 0, div2;
        int t = 0, T = width * height;
        // Esquinas
        for (int i = l; i < r; i = div1 || i + W < r ? i + W : i + l + 1) {
            if (debug)
                IJ.showProgress(t, T);
            for (int j = l; j < r; j = div1 || j + W < r ? j + W : j + l + 1) {
                t += W * W * 4;
                double Aizq = areaSector(r, i);
                double Asup = areaSector(r, j);
                double intersec = areaInt(r, -i, -j, i - r);
                /**
                 * Each sector area is substracted and then intersecction is
↪  added if it exists
                 */
                double At = A - Aizq - Asup;
                if (intersec > 0) { // There is and intersection
                    At += intersec;
                }
                double k = A / At;
                // Up left corner
                AC[i][j] = (int) Math.round(k * contar(i, j, r, tree));
                MAX = AC[i][j] > MAX ? AC[i][j] : MAX;

                // Up right corner
                AC[width - 1 - i][j] = (int) Math.round(k * contar(width - 1 -
                ↪  i, j, r, tree));
                MAX = AC[width - 1 - i][j] > MAX ? AC[width - 1 - i][j] : MAX;

                // Bottom left corner
                AC[i][height - 1 - j] = (int) Math.round(k * contar(i, height -
                ↪  1 - j, r, tree));
                MAX = AC[i][height - 1 - j] > MAX ? AC[i][height - 1 - j] : MAX;

                // bottom right corner
                AC[width - 1 - i][height - 1 - j] = (int) Math
                        .round(k * contar(width - 1 - i, height - 1 - j, r,
                        ↪  tree));
```

```
            MAX = AC[width - 1 - i][height - 1 - j] > MAX ? AC[width - 1 -
             ↪  i][height - 1 - j] : MAX;

            // Fill the pixel window
            for (int i2 = 0; i2 < W && i - l + i2 < r; i2++) {
               for (int j2 = 0; j2 < W && j - l + j2 < r; j2++) {
                  AC[i - l + i2][j - l + j2] = AC[i][j];
                  AC[width - 1 - (i - l + i2)][j - l + j2] = AC[width - 1 -
                   ↪  i][j];
                  AC[i - l + i2][height - 1 - (j - l + j2)] = AC[i][height -
                   ↪  1 - j];
                  AC[width - 1 - (i - l + i2)][height - 1 - (j - l + j2)] =
                   ↪  AC[width - 1 - i][height - 1 - j];
               }
            }
         }
      }

      // Lateral zones
      for (int i = l; i < r; i = div1 || r + W < r ? i + W : i + l + 1) {
         if (debug)
            IJ.showProgress(t, T);
         double A2 = A - areaSector(r, i);
         double k = A / A2;
         div2 = (height - 2 * r) % W == 0;
         for (int j = r + l; j < height - r; j = div2 || j + W < height - r
          ↪  ? j + W : j + l + 1) {
            t += W * W * 2;
            // Left zone
            AC[i][j] = (int) Math.round(k * contar(i, j, r, tree));
            MAX = AC[i][j] > MAX ? AC[i][j] : MAX;

            // Right zone
            AC[width - 1 - i][j] = (int) Math.round(k * contar(width - 1 -
             ↪  i, j, r, tree));
            MAX = AC[width - 1 - i][j] > MAX ? AC[width - 1 - i][j] : MAX;

            // Fill the pixel window
            for (int i2 = 0; i2 < W && i - l + i2 < r; i2++) {
               for (int j2 = 0; j2 < W && j - l + j2 < height - r; j2++) {
                  AC[i - l + i2][j - l + j2] = AC[i][j];
                  AC[width - 1 - (i - l + i2)][j - l + j2] = AC[width - 1 -
                   ↪  i][j];
               }
            }
         }
         div2 = (width - 2 * r) % W == 0;
         for (int j = r + l; j < width - r; j = div2 || j + W < width - r ?
          ↪  j + W : j + l + 1) {
            {
               t += W * W * 2;
```

```
            // Up zone
            AC[j][i] = (int) Math.round(k * contar(j, i, r, tree));
            MAX = AC[j][i] > MAX ? AC[j][i] : MAX;

            // Bottom zone
            AC[j][height - 1 - i] = (int) Math.round(k * contar(j, height
             ↪  - 1 - i, r, tree));
            MAX = AC[j][height - 1 - i] > MAX ? AC[j][height - 1 - i] :
             ↪  MAX;

            for (int i2 = 0; i2 < W && i - l + i2 < r; i2++) {
               for (int j2 = 0; j2 < W && i - l - j2 < width - r; j2++) {
                  AC[j - l + j2][i - l + i2] = AC[j][i];
                  AC[j - l + j2][height - 1 - (i - l + i2)] =
                   ↪  AC[j][height - 1 - i];
               }
            }

         }

      }
   }

   div1 = (width - 2 * r) % W == 0;
   div2 = (height - 2 * r) % W == 0;
   // Central zone
   for (int i = r + l; i < width - r; i = div1 || i + W < width - r ? i +
    ↪  W : i + l + 1) {
      if (debug)
         IJ.showProgress(t, T);
      for (int j = r + l; j < height - r; j = div2 || j + W < height - r
       ↪  ? j + W : j + l + 1) {
         t += W * W;
         AC[i][j] = contar(i, j, r, tree);
         MAX = AC[i][j] > MAX ? AC[i][j] : MAX;

         for (int i2 = 0; i2 < W && i - l + i2 < width - r; i2++) {
            for (int j2 = 0; j2 < W && j - l - j2 < height - r; j2++) {
               AC[i - l + i2][j - l + j2] = AC[i][j];
            }
         }
      }

   }

   if (debug)
      IJ.showProgress(2);
   return AC;
}

/**
```

```
    * This method returns an array with the same components as the maximum
    * accumulation found, it use the LUT to set the max and min, and the
↪   rest
    * colors are interpolated
    *
    * @return
    */
   public static Color[] getScale() {
      Color conc[] = new Color[MAX + 1];

      double k = 255.0 / MAX;
      for (int i = 0; i <= MAX; i++) {
         int index = (int) (k * i);
         conc[i] = new Color(lut.getRGB(index));
      }

      return conc;
   }

   /**
    * This method analyzes a spermatozoa image and shows the heat map in a
↪   separate
    * window
    */
   private void analyseImage() {
      FileManager fm = new FileManager();
      List<ImagePlus> images = fm.loadImageFile();
      ImagePlus impOrig = images.get(0);
      impOrig.setTitle(fm.getFilename(impOrig.getTitle()));

      ComputerVision cv = new ComputerVision();
      VideoRecognition vr = new VideoRecognition();
      ImagePlus imp = impOrig.duplicate();
      cv.convertToGrayscale(imp);
      cv.equalize(imp);
      cv.autoThresholdImagePlus(imp, "RenyiEntropy");
      List[] spermatozoa = vr.detectCells(imp);
      if (spermatozoa != null && spermatozoa[0].size() > 0) { // Se han
       ↪   detectado espermatozoides en la imagen
         Utils u = new Utils();
         KDTree tree = u.getKDTree(spermatozoa)[0];
         AccumulationWindow aw = new AccumulationWindow(tree, impOrig,
          ↪   spermatozoa[0].size());
      } else {
         IJ.error("Accumulation", "No se han detectado espermatozoides");
      }
   }

   /**
    * This method analyzes a spermatozoa video and shows the heat map in
↪   each frame
```

```
 */
private void analyseVideo() {
   FileManager fm = new FileManager();
   String file = fm.selectFile();
   ImagePlus impOrig = fm.getAVI(file), imp = impOrig.duplicate();
   impOrig.setTitle(fm.getFilename(file));
   ComputerVision cv = new ComputerVision();
   cv.convertToGrayscale(imp);
   cv.thresholdStack(imp);
   VideoRecognition vr = new VideoRecognition();
   final List[] spermatozoa = vr.detectCells(imp);
   Utils u = new Utils();
   final KDTree[] trees = u.getKDTree(spermatozoa);
   final ImagePlus impDraw = drawHeatMap(trees, impOrig);
   impDraw.show();
   IJ.setTool("oval");
   ImageWindow impW = impDraw.getWindow();
   ImageCanvas impC = impW.getCanvas();
   impC.addMouseListener(new MouseAdapter() {

      @Override
      public void mouseReleased(MouseEvent arg0) {
         Roi roi = impDraw.getRoi();
         if (roi == null)
            return;
         int x = (int) (roi.getXBase() + roi.getFloatWidth() / 2);
         int y = (int) (roi.getYBase() + roi.getFloatWidth() / 2);
         int r = (int) (roi.getFloatWidth() / 2);
         ResultsTable rt = new ResultsTable();
         rt.reset();
         for (int i = 0; i < trees.length; i++) {
            rt.incrementCounter();
            // First circumference with radius r
            int n = contar(x, y, r, trees[i]);
            rt.addValue("Spermatozoa  Abs.", n);
            rt.addValue("Spermatozoa Rel.", 100 * n /
             ↪  spermatozoa[i].size() + "%");

            // Second circumference with radius r*2
            n = contar(x, y, r * 2, trees[i]);
            rt.addValue("Spermatozoa Abs. x2", n);
            rt.addValue("Spermatozoa Rel. x2", 100 * n /
             ↪  spermatozoa[i].size() + "%");

            // Third circumference with radius r*3
            n = contar(x, y, r * 3, trees[i]);
            rt.addValue("Spermatozoa Abs. x3", n);
            rt.addValue("Spermatozoa Rel. x3", 100 * n /
             ↪  spermatozoa[i].size() + "%");

         }
```

```
            rt.show("Spermatozoa per Frame (" + (r * Params.micronPerPixel)
            ↪  + " um)");

        }
      });
  }

  /**
   * This method draws a heat map over impOrig video with spermatozoa
↪  contained in
   * tree
   *
   * @param tree
   * @param impOrig
   * @return
   */
  private ImagePlus drawHeatMap(KDTree[] tree, ImagePlus impOrig) {
     ImagePlus impDraw = impOrig.duplicate();
     if (impDraw.getType() != ImagePlus.COLOR_RGB) { // Si no es RGB
        ComputerVision cv = new ComputerVision();
        cv.convertToRGB(impDraw);
     }
     impDraw.setTitle(impOrig.getTitle() + "-HeatMap");
     ImageStack stack = impDraw.getStack(), stackOrig = impOrig.getStack();
     int nFrames = impDraw.getStackSize();
     int width = impDraw.getWidth(), height = impDraw.getHeight();
     int AC1[][] = getAccumulation(width, height, Params.radius,
     ↪  Params.window, tree[0], false);
     for (int i = 1; i <= nFrames; i += Params.frameInt) {
        IJ.showStatus("Drawing image... (" + i + ", " + nFrames + ")");
        IJ.showProgress(i, nFrames);

        Color conc[] = getScale();
        ImageProcessor ip = stack.getProcessor(i);
        drawImage(ip, AC1, conc);
        ImageRoi roi = new ImageRoi(0, 0, stackOrig.getProcessor(i));
        roi.setOpacity(Params.opacity / 100.0);
        ip.drawRoi(roi);

        if (i + Params.frameInt < nFrames) {
           int AC2[][] = getAccumulation(width, height, Params.radius,
           ↪  Params.window, tree[i + Params.frameInt],
                 false);
           for (int j = 1; j < Params.frameInt; j++) {
              double p = (double) j / Params.frameInt;
              int AC[][] = interpolar(AC1, AC2, p);
              conc = getScale();
              ip = stack.getProcessor(i + j);
              drawImage(ip, AC, conc);
              roi = new ImageRoi(0, 0, stackOrig.getProcessor(i + j));
              roi.setOpacity(Params.opacity / 100.0);
```

```
                ip.drawRoi(roi);
            }
            AC1 = AC2;
        } else {
            for (int j = 1; j < Params.frameInt && j + i <= nFrames; j++) {
                ip = stack.getProcessor(i + j);
                drawImage(ip, AC1, conc);
                roi = new ImageRoi(0, 0, stackOrig.getProcessor(i + j));
                roi.setOpacity(Params.opacity / 100.0);
                ip.drawRoi(roi);
            }
        }
    }
    IJ.showProgress(2);
    return impDraw;
}

/**
 * This method returns the interpolated matrix between AC1 and AC2 based
↪  on the
 * distance p
 *
 * @param AC1
 * @param AC2
 * @param p
 * @return
 */
private int[][] interpolar(int AC1[][], int AC2[][], double p) {
    int height = AC1.length;
    int width = AC1[0].length;
    int AC[][] = new int[width][height];

    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {
            AC[i][j] = (int) Math.round((1 - p) * AC1[i][j] + p *
            ↪  AC2[i][j]);
            MAX = AC[i][j] > MAX ? AC[i][j] : MAX;
        }
    }

    return AC;
}

/**
 * This method draws on the image ip the heat map using accumulation in
↪  AC and
 * the scale conc
 *
 * @param ip
 * @param AC
 * @param conc
```

```
     */
    public static void drawImage(ImageProcessor ip, int AC[][], Color conc[])
    ↪  {
       for (int i = 0; i < ip.getWidth(); i++) {
          for (int j = 0; j < ip.getHeight(); j++) {
             ip.setColor(conc[AC[i][j]]);
             ip.drawPixel(i, j);
          }
       }
    }

    /**
     * This method is inherit from SwingWorker class and it is the starting
↪   point
     * after the execute() method is called.
     */
    @Override
    public Boolean doInBackground() {
       switch (analysis) {
       case IMAGE:
          analyseImage();
          break;
       case VIDEO:
          analyseVideo();
          break;
       default:
          break;
       }
       return null;
    }

    /**
     * This method is executed at the end of the worker thread in the Event
↪   Dispatch
     * Thread.
     */
    @Override
    protected void done() {
    }

    /**
     * This method opens a set of dialogs to ask the user which analysis has
↪   to be
     * carried on.
     */
    public void selectAnalysis() {
       // Ask if user wants to analyze a file or experiment
       Object[] options = { "Image", "Video" };
       String question = "What do you want to analyze?";
       String title = "Choose one analysis...";
       final int IMAGE = 0;
```

```
        int VIDEO = 1;
        Utils utils = new Utils();
        int sourceSelection = utils.analysisSelectionDialog(options, question,
        ↪  title);
        if (sourceSelection < 0) {
            return;
        } else if (sourceSelection == IMAGE) { // Image
            analysis = TypeOfAnalysis.IMAGE;

        } else if (sourceSelection == VIDEO) { // Video
            analysis = TypeOfAnalysis.VIDEO;
        }
    }

    /**
     * This method returns the class attribute MAX
     *
     * @return
     */
    public static int getMax() {
        return MAX;
    }

}
```

A.3. AccumulationWindow.java

```
package gui;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Image;
import java.awt.Insets;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
```

```
import javax.swing.JSeparator;
import javax.swing.JSlider;
import javax.swing.SwingConstants;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import analysis.Accumulation;
import data.Params;
import ij.IJ;
import ij.ImagePlus;
import ij.gui.ImageRoi;
import ij.io.FileSaver;
import ij.process.ImageProcessor;
import net.sf.javaml.core.kdtree.KDTree;

/**
 * This class implements all the functions related to accumulation analysis
 ↪  on
 * image.
 *
 * @author Jorge Yagüe
 *
 */
public class AccumulationWindow extends JFrame implements MouseListener,
 ↪  ChangeListener {

   private static final long serialVersionUID = 1L;
   private JLabel imgLabel;
   /** Radius slider */
   private JSlider sldRadius;
   /** Opacity slider */
   private JSlider sldOpac;
   /** K-d tree which contains spermatozoa */
   private KDTree tree;
   private boolean isProcessing = false;
   /** Label where maximun accumulation is shown */
   private JLabel acText;
   private JButton saveBtn;
   private JLabel label1;
   private JLabel label2;
   /** Processed image */
   private ImagePlus imp;
   /** Original image */
   private ImagePlus impOrig;
   /** Image's width and height on JFrame */
   private int width;
   private int height;
   /** Total spermatozoa in image */
   private int total;

   /**
```

```
     * Constructor
     *
     * @param tree
     * @param orig
     * @param total
     */
    public AccumulationWindow(KDTree tree, ImagePlus orig, int total) {
        super(orig.getTitle() + "-HeatMap");
        this.total = total;
        acText = new JLabel();
        saveBtn = new JButton("Save image");
        this.tree = tree;
        this.impOrig = orig;
        this.imp = orig.duplicate();
        imp.setTitle(this.getTitle());
        imgLabel = new JLabel();

        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        double w = screenSize.getWidth();
        double h = screenSize.getHeight();
        this.width = (int) (w * 0.6);
        this.height = (int) (h * 0.6);

        sldRadius = new JSlider(JSlider.HORIZONTAL, 25, 250, 100);
        sldRadius.setMinorTickSpacing(5);
        sldRadius.setMajorTickSpacing(25);
        sldRadius.setPaintTicks(true);
        sldRadius.setPaintLabels(true);
        sldRadius.setLabelTable(sldRadius.createStandardLabels(25));
        sldRadius.addMouseListener(this);
        sldRadius.setVisible(true);

        sldOpac = new JSlider(JSlider.HORIZONTAL, 0, 100, 50);
        sldOpac.setMinorTickSpacing(5);
        sldOpac.setMajorTickSpacing(25);
        sldOpac.setPaintTicks(true);
        sldOpac.setPaintLabels(true);
        sldOpac.setLabelTable(sldRadius.createStandardLabels(25));
        sldOpac.setVisible(true);
        sldOpac.addChangeListener(this);

        JPanel panel = new JPanel(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        c.anchor = GridBagConstraints.CENTER;
        JLabel label = new JLabel("Radius");
        panel.add(label, c);
```

```
c.gridy = 1;
label = new JLabel("x " + Params.micronPerPixel + " um/pixel");
panel.add(label, c);

c.gridx = 1;
c.gridy = 0;
c.gridwidth = 4;
c.gridheight = 2;
c.weightx = 1;
c.fill = GridBagConstraints.HORIZONTAL;
panel.add(sldRadius, c);

c.gridx = 0;
c.gridy = 2;
c.gridwidth = 1;
c.gridheight = 2;
c.weightx = 0;
c.weightx = 0;
c.fill = GridBagConstraints.NONE;
label = new JLabel("Opacity");
panel.add(label, c);

c.gridx = 1;
c.gridwidth = 4;
c.gridheight = 2;
c.weightx = 1;
c.fill = GridBagConstraints.HORIZONTAL;
panel.add(sldOpac, c);

c.gridx = 0;
c.gridy += 2;
c.gridwidth = 4;
c.gridheight = 1;
panel.add(new JSeparator(SwingConstants.HORIZONTAL), c);

c.gridx = 1;
c.gridy += 1;
c.gridwidth = 1;
c.gridheight = 1;
c.weightx = 0;
panel.add(acText, c);

c.gridx = 1;
c.gridy += 1;
c.gridwidth = 1;
c.gridheight = 3;
c.fill = GridBagConstraints.BOTH;
c.anchor = GridBagConstraints.NORTHWEST;
c.weightx = c.weighty = 1;
panel.add(imgLabel, c);
```

```
ImagePlus scale = null;
try {
   Image img = ImageIO.read(getClass().getResource("/Jet.png"));
   scale = new ImagePlus("Scale", img);
} catch (IOException e1) {
   IJ.handleException(e1);
}
int scalewidth = scale.getWidth() * this.height / scale.getHeight();
ImageProcessor ip = scale.getProcessor();
ip.setInterpolationMethod(ImageProcessor.BILINEAR);
ip = ip.resize(scalewidth, this.height);
scale.setProcessor(ip);
c.gridx = 2;
c.gridwidth = 1;
c.gridheight = 3;
c.weightx = 0;
c.insets = new Insets(0, 12, 0, 0);
JLabel scaLabel = new JLabel();
scaLabel.setIcon(new ImageIcon(scale.getImage()));
scaLabel.repaint();
panel.add(scaLabel, c);

label1 = new JLabel();
c.gridx = 3;
c.gridheight = 1;
c.gridwidth = 1;
c.fill = GridBagConstraints.HORIZONTAL;
c.anchor = GridBagConstraints.NORTHWEST;
c.insets = new Insets(0, 0, 0, 0);
panel.add(label1, c);

label2 = new JLabel();
c.gridy += 1;
panel.add(label2, c);

c.gridx = 0;
c.gridy += 2;
c.gridwidth = 4;
c.gridheight = 1;
c.weightx = 1;
c.anchor = GridBagConstraints.CENTER;
panel.add(new JSeparator(SwingConstants.HORIZONTAL), c);

c.gridx = 3;
c.gridy += 1;
c.gridwidth = 1;
c.gridheight = 1;
panel.add(saveBtn, c);

saveBtn.addActionListener(new ActionListener() {
```

```
            public void actionPerformed(ActionEvent e) {
                ImageProcessor ip = impOrig.getProcessor();
                ImageRoi roi = new ImageRoi(0, 0, ip);
                roi.setOpacity(sldOpac.getValue() / 100.0);
                imp.getProcessor().drawRoi(roi);
                FileSaver f = new FileSaver(imp);
                FileSaver.setJpegQuality(100);
                if (f.saveAsJpeg())
                    saveBtn.setEnabled(false);
            }
        });

        processImage();

        this.setContentPane(panel);
        this.pack();
        this.setVisible(true);
    }

    @Override
    public void mouseClicked(MouseEvent e) {
    }

    @Override
    public void mousePressed(MouseEvent e) {
    }

    /**
     * When radius slider changes, image is processed
     */
    @Override
    public void mouseReleased(MouseEvent e) {
        Object auxWho = e.getSource();
        if (auxWho == sldRadius) {
            refreshImage();
        }
    }

    @Override
    public void mouseEntered(MouseEvent e) {
    }

    @Override
    public void mouseExited(MouseEvent e) {
    }

    /**
     * Refresh image
     */
    private void refreshImage() {
        if (!isProcessing) {
```

```
            isProcessing = true;
            Thread t1 = new Thread(new Runnable() {
                public void run() {
                    processImage();
                    isProcessing = false;
                }
            });
            t1.start();
        }
    }

    /**
     * Process image and update imp attribute
     */
    private void processImage() {
        int AC[][] = Accumulation.getAccumulation(impOrig.getWidth(),
        ↪  impOrig.getHeight(), sldRadius.getValue(), 1,
              tree, true);
        Color conc[] = Accumulation.getScale();
        Accumulation.drawImage(imp.getProcessor(), AC, conc);
        label1.setText(100 * Accumulation.getMax() / total + "%");
        label2.setText((100 * Accumulation.getMax() / 2) / total + "%");
        showImage();
    }

    /**
     * Paint image on screen
     */
    private void showImage() {
        ImagePlus impDraw = imp.duplicate();

        ImageProcessor ip = impDraw.getProcessor();
        ip.setInterpolationMethod(ImageProcessor.BILINEAR);
        ip = ip.resize(this.width, this.height);
        impDraw.setProcessor(ip);

        ImagePlus impOrig2 = impOrig.duplicate();
        ip = impOrig2.getProcessor();
        ip.setInterpolationMethod(ImageProcessor.BILINEAR);
        ip = ip.resize(this.width, this.height);
        impOrig2.setProcessor(ip);

        ImageRoi roi = new ImageRoi(0, 0, ip);
        roi.setOpacity(sldOpac.getValue() / 100.0);
        impDraw.getProcessor().drawRoi(roi);

        imgLabel.setIcon(new ImageIcon(impDraw.getImage()));
        imgLabel.repaint();

        acText.setText("Max. Accumulation: " + Accumulation.getMax() + "
        ↪  spermatozoa in a "
```

```
            + sldRadius.getValue() * Params.micronPerPixel + " um radius");
    }

    /**
     * When opacity slider changes, image is updated automatically
     */
    @Override
    public void stateChanged(ChangeEvent e) {
        Object auxWho = e.getSource();
        if (auxWho == sldOpac) {
            showImage();
        }

    }

}
```

A.4. CellCountWindow.java

```
package gui;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSeparator;
import javax.swing.JTextField;
import javax.swing.SwingConstants;

import data.Cell;
import data.Params;
import data.Square;
import functions.ComputerVision;
import functions.FileManager;
```

```java
import functions.Utils;
import functions.VideoRecognition;
import ij.ImagePlus;
import ij.process.ImageProcessor;
import net.sf.javaml.core.kdtree.KDTree;

/**
 * This class implements all the functions related to cell count analysis.
 *
 * @author Jorge Yagüe
 *
 */
public class CellCountWindow extends JFrame {

   private static final long serialVersionUID = 1L;

   private enum TypeOfAnalysis {
      DIRECTORY, FILE, NONE
   }

   private TypeOfAnalysis analysis = TypeOfAnalysis.NONE;

   /** Image list to process */
   private List<ImagePlus> images;
   /** Label where the current image is shown */
   private JLabel imgLabel;
   /** Label where the cell count is shown */
   private JLabel concLabel;
   private JButton nextBtn;
   private JButton prevBtn;
   /** Button to hide/show the squares */
   private JButton cuadricula;
   /** TextField to write the dilution of the current image */
   private JTextField diluc;
   private int imgIndex;
   /** Cell count of each image on list */
   private double[] conc;
   /** Image list which have been processed */
   private ArrayList<ImagePlus> impDraw;
   /** Number of squares detected in each image */
   private int[] squares;

   /**
    * Constructor
    */
   public CellCountWindow() {
      imgLabel = new JLabel();
      concLabel = new JLabel();
      nextBtn = new JButton("Next");
      prevBtn = new JButton("Previous");
      cuadricula = new JButton("Cuadricula");
```

```
        diluc = new JTextField("1", 4);
        imgIndex = 0;
        conc = null;
        squares = null;
    }

    /**
     * This method creates and shows the window.
     */
    private void showWindow() {
        JPanel panel = new JPanel(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;

        c.gridy = 0;
        c.gridx = 0;
        c.gridheight = 1;
        c.gridwidth = 3;
        c.weightx = 1;
        panel.add(concLabel, c);

        JLabel label = new JLabel("Dilution: 1/");
        label.setHorizontalAlignment(SwingConstants.RIGHT);
        c.gridy += 1;
        c.gridwidth = 1;
        c.anchor = GridBagConstraints.CENTER;
        panel.add(label, c);

        c.gridx = 1;
        c.weightx = 1;
        diluc.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                double dis = Double.parseDouble(((JTextField)
                ↪  e.getSource()).getText());
                setConc(dis);
            }
        });
        panel.add(diluc, c);

        c.gridx = 2;
        c.fill = GridBagConstraints.NONE;
        cuadricula.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                cuadricula.setSelected(!cuadricula.isSelected());
                setImage();
            }
        });
        cuadricula.setSelected(true);
        panel.add(cuadricula, c);

        c.gridy += 1;
```

```
c.ipadx = 0;
c.gridx = 0;
c.gridwidth = 4;
c.gridheight = 1;
c.weightx = 1;
c.fill = GridBagConstraints.HORIZONTAL;
panel.add(new JSeparator(SwingConstants.HORIZONTAL), c);

c.gridx = 2;
c.gridy += 1;
c.gridwidth = 1;
c.gridheight = 1;
c.fill = GridBagConstraints.BOTH;
c.anchor = GridBagConstraints.NORTHWEST;
c.weighty = 1;
panel.add(imgLabel, c);
processImage();

c.gridy += 1;
c.gridx = 0;
c.gridwidth = 4;
c.gridheight = 1;
c.weighty = 0;
c.anchor = GridBagConstraints.NORTH;
c.fill = GridBagConstraints.HORIZONTAL;
panel.add(new JSeparator(SwingConstants.HORIZONTAL), c);

prevBtn.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      if (imgIndex > 0) {
         nextBtn.setEnabled(true);
         imgIndex--;
         setImage();
      } else if (imgIndex == 0) {
         prevBtn.setEnabled(false);
      }
   }
});
c.gridy += 1;
c.gridwidth = 1;
c.gridheight = 1;
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 1;
c.weighty = 1;
c.anchor = GridBagConstraints.CENTER;
prevBtn.setEnabled(false);
panel.add(prevBtn, c);

nextBtn.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      if (imgIndex < (images.size() - 1)) {
```

```
                prevBtn.setEnabled(true);
                if (impDraw.size() > ++imgIndex) {
                    // Se evita procesar una imagen dos veces
                    setImage();
                } else {
                    processImage();
                }
            } else if (imgIndex == (images.size() - 1)) {
                nextBtn.setEnabled(false);
            }
        }
    });
    c.gridx = 3;
    panel.add(nextBtn, c);

    this.setContentPane(panel);
    this.pack();
    this.setVisible(true);
}

public int run() {
    int out = selectAnalysis();
    if (out < 0)
        return out;
    switch (analysis) {
    case FILE:
        out = analyseFile();
        break;
    case DIRECTORY:
        out = analyseDirectory();
        break;
    default:
        out = -2;
        break;
    }
    return out;
}

/**
 * This method opens a set of dialogs to ask the user which analysis has
↪  to be
 * carried on.
 */
public int selectAnalysis() {
    // Ask if user wants to analyze a file or directory
    Object[] options = { "File", "Directory" };
    String question = "What do you want to analyze?";
    String title = "Choose one analysis...";
    final int FILE = 0;
    final int DIR = 1;
    Utils utils = new Utils();
```

```
        int sourceSelection = utils.analysisSelectionDialog(options, question,
        ↪  title);
        if (sourceSelection < 0) {
            analysis = TypeOfAnalysis.NONE;
            return -1;
        } else if (sourceSelection == FILE) {
            analysis = TypeOfAnalysis.FILE;
        } else if (sourceSelection == DIR) {
            analysis = TypeOfAnalysis.DIRECTORY;
        }
        return 0;
    }

    private int analyseFile() {
        FileManager fm = new FileManager();
        List<ImagePlus> images = fm.loadImageFile();
        if (images != null) {
            setImages(images);
            showWindow();
            return 0;
        } else {
            return -1;
        }
    }

    private int analyseDirectory() {
        FileManager fm = new FileManager();
        List<ImagePlus> images = fm.loadImageDirectory();
        if (images != null) {
            setImages(images);
            showWindow();
            return 0;
        } else {
            return -1;
        }
    }

    /**
     * This method sets the images attribute with the given list of
↪    ImagePlus.
     *
     * @param i
     */
    public void setImages(List<ImagePlus> i) {
        images = i;
        impDraw = new ArrayList<ImagePlus>(i.size());
        conc = new double[i.size()];
        squares = new int[i.size()];
    }

    /**
```

```
 * Process the first image and show it on screen
 */
private void processImage() {
   ComputerVision cv = new ComputerVision();
   VideoRecognition vr = new VideoRecognition();
   ImagePlus impOrig = images.get(imgIndex);
   this.setTitle(impOrig.getTitle());
   ImagePlus imp = impOrig.duplicate();
   ImagePlus impDraw = imp.duplicate();
   cv.convertToGrayscale(imp);
   cv.equalize(imp);
   imp.getProcessor().findEdges();
   cv.autoThresholdImagePlus(imp, "Otsu");
   imp.getProcessor().invertLut();
   List<Square> squares = vr.detectSquares(imp)[0];
   this.squares[imgIndex] = squares.size();
   imp = impOrig.duplicate();
   cv.convertToGrayscale(imp);
   cv.equalize(imp);
   cv.autoThresholdImagePlus(imp, "RenyiEntropy");
   List[] spermatozoa = squares.size() == 0 ? vr.detectCells(imp) :
    ↪  vr.detectCellsSquares(imp);
   double acum = spermatozoa[0].size();
   if (squares != null && !squares.isEmpty()) {
      if (spermatozoa != null && !spermatozoa[0].isEmpty()) {
         ImageProcessor ip = impDraw.getProcessor();
         Utils u = new Utils();
         KDTree tree = u.getKDTree(spermatozoa)[0];
         List<Cell> inside = new LinkedList<Cell>();
         ip.setLineWidth(4);
         ip.setFont(new Font("SansSerif", Font.PLAIN, 32));
         for (Square s : squares) {
            ip.setColor(Color.red);
            ip.drawRect((int) s.x, (int) s.y, (int) s.width, (int)
             ↪  s.height);
            Object[] inside0 = tree.range(new double[] { s.x, s.y },
                  new double[] { s.x + s.width, s.y + s.height });
            Cell[] insideAux = Arrays.copyOf(inside0, inside0.length,
             ↪  Cell[].class);
            inside.addAll(Arrays.asList(insideAux));
            ip.setColor(Color.green);
            ip.moveTo((int) (s.x + s.width / 2), (int) (s.y + s.height /
             ↪  2));
            ip.drawString("" + insideAux.length);
         }
         // Square volume (ml)
         double volume = Params.sideS * Params.sideS * Params.depthC /
          ↪  1e12;
         // Concentration (Million spermatozoa / ml)
         acum = inside.size() / (squares.size() * volume * 1e6);
      }
```

```
        } else {
            double volume = Params.micronPerPixel * impDraw.getWidth() *
             ↪  Params.micronPerPixel * impDraw.getHeight()
                  * Params.depthC / 1e12;
            acum = acum / (volume * 1e6);
        }
        conc[imgIndex] = acum;
        this.impDraw.add(impDraw);
        setImage();
    }

    /**
     * This method sets the first image on the list and show it on screen.
     */
    public void setImage() {
        setConc(Double.parseDouble(diluc.getText()));
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        double w = screenSize.getWidth();
        int targetWidth = (int) (w * 0.45);
        ImagePlus imp = cuadricula.isSelected() ? impDraw.get(imgIndex) :
         ↪  images.get(imgIndex);
        ImageProcessor ip = imp.getProcessor();
        ip.setInterpolationMethod(ImageProcessor.BILINEAR);
        ip = ip.resize(targetWidth);
        imp.setProcessor(ip);
        imgLabel.setIcon(new ImageIcon(imp.getImage()));
        imgLabel.repaint();
    }

    /**
     * Update cell count with new disolution d
     *
     * @param d
     */
    private void setConc(double d) {
        double aux = Math.round(100 * conc[imgIndex] * d) / 100.0;
        concLabel.setText(aux + " Mespermatozoa / ml (" + squares[imgIndex] +
         ↪  " squares)");
    }

}
```

A.5. SettingsWindow.java

```
package gui;

import java.awt.Dimension;
import java.awt.GridBagConstraints;
```

```
import java.awt.GridBagLayout;
import java.awt.HeadlessException;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTabbedPane;

/**
 * @author Carlos Alquezar and Jorge Yagüe
 *
 */
public class SettingsWindow extends JFrame {

   JButton cancelBtn;
   JButton saveBtn;
   SettingsWindow sw; // Self reference used in action listeners
   MotilitySettings ms;
   ChemotaxisSettings cs;
   MorphometrySettings mphs;
   ViabilitySettings vs;
   AccumulationSettings ac;
   CellCountSettings ccs;

   public SettingsWindow(String title) throws HeadlessException {
      super(title);
      sw = this;
      ms = new MotilitySettings();
      cs = new ChemotaxisSettings();
      mphs = new MorphometrySettings();
      vs = new ViabilitySettings();
      ac = new AccumulationSettings();
      ccs = new CellCountSettings();
      createGUI();
      this.setVisible(true);
      // this.setLocationRelativeTo(null);
      Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
      int w = (int) screenSize.getWidth();
      int h = (int) screenSize.getHeight();
      this.setMinimumSize(new Dimension(w / 2, h));
   }

   private JTabbedPane addTabPane() {
      JTabbedPane tabbedPane = new JTabbedPane();
      tabbedPane.addTab("Motility Module", ms.createGUI());
      tabbedPane.addTab("Chemotaxis Module", cs.createGUI());
      tabbedPane.addTab("Morphometry Module", mphs.createGUI());
      tabbedPane.addTab("Viability Module", vs.createGUI());
```

```
        tabbedPane.addTab("Accumulation Module", ac.createGUI());
        tabbedPane.addTab("Cell Count Module", ccs.createGUI());
        return tabbedPane;
    }

    private void createButtons() {
        saveBtn = new JButton("Save");
        // Add action listener
        saveBtn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                setParameters();
                sw.dispatchEvent(new WindowEvent(sw,
                ↪  WindowEvent.WINDOW_CLOSING));
            }
        });
        cancelBtn = new JButton("Cancel");
        // Add action listener
        cancelBtn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                sw.dispatchEvent(new WindowEvent(sw,
                ↪  WindowEvent.WINDOW_CLOSING));
            }
        });
    }

    private void createGUI() {
        this.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.fill = GridBagConstraints.HORIZONTAL;
        ///////////////////////
        c.gridx = 1;
        c.gridy = 0;
        c.ipadx = 2;
        c.gridheight = 10;
        c.gridwidth = 10;
        JTabbedPane tabbedPane = addTabPane();
        this.add(tabbedPane, c);
        ///////////////////////
        c.gridheight = 1;
        c.gridwidth = 1;
        createButtons();
        c.gridx = 0;
        c.gridy = 10;
        this.add(cancelBtn, c);
        c.gridx = 11;
        this.add(saveBtn, c);
    }

    public void setParameters() {
        ms.setParameters();
        cs.setParameters();
```

```
        mphs.setParameters();
        vs.setParameters();
        ac.setParameters();
        ccs.setParameters();
    }
}
```

A.6. AccumulationSettings.java

```
package gui;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

import data.AccumulationParams;

/**
 * This class implements all the settings related to accumulation analysis
 *
 * @author Jorge Yagüe
 *
 */
public class AccumulationSettings extends JPanel {

    JTextField maxSizeTF = new JTextField("" + AccumulationParams.maxSize,
    ↪  4);
    JTextField micronPerPixelTF = new JTextField("" +
    ↪  AccumulationParams.micronPerPixel, 4);
    JTextField minSizeTF = new JTextField("" + AccumulationParams.minSize,
    ↪  4);
    JTextField frameRateTF = new JTextField("" +
    ↪  AccumulationParams.frameRate, 4);
    JTextField frameIntTF = new JTextField("" + AccumulationParams.frameInt,
    ↪  4);
    JTextField radiusTF = new JTextField("" + AccumulationParams.radius, 4);
    JTextField opacTF = new JTextField("" + AccumulationParams.opacity, 4);
    JTextField windowTF = new JTextField("" + AccumulationParams.window, 4);
    JTextField firstFrameTF = new JTextField("", 4);
    JTextField lastFrameTF = new JTextField("", 4);
    JTextField dateTF = new JTextField(AccumulationParams.date, 8);
    JTextField genericTF = new JTextField(AccumulationParams.genericField,
    ↪  8);
    JTextField maleTF = new JTextField(AccumulationParams.male, 8);

    public AccumulationSettings() {
```

```
        super();
        firstFrameTF.setText("" + AccumulationParams.firstFrame);
        if (AccumulationParams.lastFrame < 0)
            lastFrameTF.setText("-1");
        else
            lastFrameTF.setText("" + AccumulationParams.lastFrame);
    }

    public JPanel createGUI() {
        this.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.gridx = 1;
        c.gridy = 0;
        ///////////////
        JLabel label = new JLabel("------------------------------ ");
        this.add(label, c);
        c.gridy += 1;
        label = new JLabel("-- Persistent parameters -- ");
        c.gridx = 1;
        this.add(label, c);
        c.gridy += 1;
        label = new JLabel("------------------------------ ");
        c.gridx = 1;
        this.add(label, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Microns per Pixel: ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(micronPerPixelTF, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Minimum cell size (um^2): ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(minSizeTF, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Maximum cell size (um^2): ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(maxSizeTF, c);
        ///////////////
        c.gridy += 1;
        c.gridx = 1;
        label = new JLabel("-------------");
        this.add(label, c);
        c.gridy += 1;
```

```
label = new JLabel("Video");
this.add(label, c);
c.gridy += 1;
label = new JLabel("-------------");
this.add(label, c);
///////////////
c.gridy += 1;
label = new JLabel("Frame Rate (frames/s): ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(frameRateTF, c);
//////////
c.gridy += 1;
label = new JLabel("Frames to interpolate: ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(frameIntTF, c);
//////////
c.gridy += 1;
label = new JLabel("Radius to analyze (pixels): ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(radiusTF, c);
//////////
c.gridy += 1;
label = new JLabel("Opacity (%): ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(opacTF, c);
//////////
c.gridy += 1;
label = new JLabel("Window size (pixels): ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(windowTF, c);
//////////
c.gridy += 1;
label = new JLabel("------------------------------- ");
c.gridx = 1;
this.add(label, c);
c.gridy += 1;
label = new JLabel("-- Not persistent parameters -- ");
c.gridx = 1;
this.add(label, c);
c.gridy += 1;
label = new JLabel("------------------------------- ");
```

```
        c.gridx = 1;
        this.add(label, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("First frame (seconds): ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(firstFrameTF, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Last frame (seconds): ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(lastFrameTF, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Male: ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(maleTF, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Date: ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(dateTF, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Generic: ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(genericTF, c);

        return this;
    }

    public void setParameters() {
        AccumulationParams.micronPerPixel =
         ↪  Double.parseDouble(micronPerPixelTF.getText());
        AccumulationParams.male = maleTF.getText();
        AccumulationParams.date = dateTF.getText();
        AccumulationParams.genericField = genericTF.getText();
        AccumulationParams.minSize = Float.parseFloat(minSizeTF.getText());
        AccumulationParams.maxSize = Float.parseFloat(maxSizeTF.getText());
        AccumulationParams.frameRate =
         ↪  Float.parseFloat(frameRateTF.getText());
```

```
        AccumulationParams.frameInt = Integer.parseInt(frameIntTF.getText());
        AccumulationParams.radius = Integer.parseInt(radiusTF.getText());
        AccumulationParams.opacity = Integer.parseInt(opacTF.getText());
        AccumulationParams.window = Integer.parseInt(windowTF.getText());
        AccumulationParams.firstFrame =
        ↪  Float.parseFloat(firstFrameTF.getText());
        AccumulationParams.lastFrame =
        ↪  Float.parseFloat(lastFrameTF.getText());
        AccumulationParams.saveParams();
    }

}
```

A.7. CellCountSettings.java

```
package gui;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

import data.CellCountParams;

/**
 * This class implements all the settings related to cell count analysis
 *
 * @author Jorge Yagüe
 *
 */
public class CellCountSettings extends JPanel {

    JTextField maxSizeTF = new JTextField("" + CellCountParams.maxSize, 4);
    JTextField micronPerPixelTF = new JTextField("" +
    ↪  CellCountParams.micronPerPixel, 4);
    JTextField minSizeTF = new JTextField("" + CellCountParams.minSize, 4);
    JTextField sideSTF = new JTextField("" + CellCountParams.sideS, 4);
    JTextField depthCTF = new JTextField("" + CellCountParams.depthC, 4);
    JTextField dateTF = new JTextField(CellCountParams.date, 8);
    JTextField genericTF = new JTextField(CellCountParams.genericField, 8);
    JTextField maleTF = new JTextField(CellCountParams.male, 8);

    public CellCountSettings() {
        super();
    }

    public JPanel createGUI() {
```

```
this.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
c.gridx = 1;
c.gridy = 0;
///////////////
JLabel label = new JLabel("------------------------------- ");
this.add(label, c);
c.gridy += 1;
label = new JLabel("-- Persistent parameters -- ");
c.gridx = 1;
this.add(label, c);
c.gridy += 1;
label = new JLabel("------------------------------- ");
c.gridx = 1;
this.add(label, c);
///////////////
c.gridy += 1;
label = new JLabel("Microns per Pixel: ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(micronPerPixelTF, c);
///////////////
c.gridy += 1;
label = new JLabel("Minimum cell size (um^2): ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(minSizeTF, c);
///////////////
c.gridy += 1;
label = new JLabel("Maximum cell size (um^2): ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(maxSizeTF, c);
///////////////
c.gridy += 1;
label = new JLabel("Square side length (um): ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(sideSTF, c);
///////////////
c.gridy += 1;
label = new JLabel("Camera depth (um): ");
c.gridx = 1;
this.add(label, c);
c.gridx = 2;
this.add(depthCTF, c);
///////////////
```

```
        ///////////////
        c.gridy += 1;
        label = new JLabel("------------------------------- ");
        c.gridx = 1;
        this.add(label, c);
        c.gridy += 1;
        label = new JLabel("-- Not persistent parameters -- ");
        c.gridx = 1;
        this.add(label, c);
        c.gridy += 1;
        label = new JLabel("------------------------------- ");
        c.gridx = 1;
        this.add(label, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Male: ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(maleTF, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Date: ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(dateTF, c);
        ///////////////
        c.gridy += 1;
        label = new JLabel("Generic: ");
        c.gridx = 1;
        this.add(label, c);
        c.gridx = 2;
        this.add(genericTF, c);

        return this;
    }

    public void setParameters() {
        CellCountParams.micronPerPixel =
        ↪  Double.parseDouble(micronPerPixelTF.getText());
        CellCountParams.male = maleTF.getText();
        CellCountParams.date = dateTF.getText();
        CellCountParams.genericField = genericTF.getText();
        CellCountParams.minSize = Float.parseFloat(minSizeTF.getText());
        CellCountParams.maxSize = Float.parseFloat(maxSizeTF.getText());
        CellCountParams.sideS = Float.parseFloat(sideSTF.getText());
        CellCountParams.depthC = Float.parseFloat(depthCTF.getText());
        CellCountParams.saveParams();
    }
```

```
}
```

A.8. Square.java

```
package data;

import java.io.Serializable;

/**
 * @author Jorge Yagüe
 *
 */
public class Square implements Serializable {

   @Override
   public String toString() {
      return "Square [x=" + x + ", y=" + y + ", width=" + width + ",
       ↪  height=" + height + "]";
   }

   /**
    *
    */
   private static final long serialVersionUID = 1L;
   /** Up left corner */
   public float x;
   /**   */
   public float y;
   /**   */
   public float width;
   /**   */
   public float height;
   /**   */
   public float area;
   /** ID */
   public String id = "*";

   /**
    * @param source - Square to be copied
    */
   public void copy(Square source) {
      this.id = source.id;
      this.x = source.x;
      this.y = source.y;
      this.area = source.area;
      this.width = source.width;
      this.height = source.height;
   }
```

```
}
```

A.9. Utils.java

```
package functions;

import java.util.List;
import java.util.ListIterator;

import javax.swing.JOptionPane;

import data.Cell;
import ij.IJ;
import net.sf.javaml.core.kdtree.KDTree;

/**
 * @author Carlos Alquezar and Jorge Yagüe
 *
 */
public class Utils {

   /**
    *
    * @param options
    * @param question
    * @param title
    * @return
    */
   public int analysisSelectionDialog(Object[] options, String question,
    ↪  String title) {
      int n = JOptionPane.showOptionDialog(null, question, title,
       ↪  JOptionPane.YES_NO_OPTION,
            JOptionPane.QUESTION_MESSAGE, null, // do not use a custom Icon
            options, // the titles of buttons
            options[0]); // default button title
      return n;
   }

   /**
    * @param orig
    * @return
    */
   public int[] convertLongArrayToInt(long[] orig) {
      int[] arrayInt = new int[orig.length];
      for (int i = 0; i < orig.length; i++)
         arrayInt[i] = (int) orig[i];
      return arrayInt;
```

```
}

/*******************************************************/
/**
 * @param id
 * @param spermatozoa
 * @return
 */
public Cell getCell(String id, List spermatozoa) {
   Cell cell = null;
   for (ListIterator j = spermatozoa.listIterator(); j.hasNext();) {
      Cell candidate = (Cell) j.next();
      if (candidate.id.equals(id) && id != "***") {
         cell = candidate;
         break;
      }
   }
   return cell;
}

public int getTrackNr(List track) {
   Cell cell = (Cell) track.get(0);
   return cell.trackNr;
}

/*******************************************************/
/**
 * @param theTracks 2D-ArrayList with all the tracks
 * @return String with the results in tsv format (tab separated values)
 */
public String printXYCoords(List theTracks) {
   int nTracks = theTracks.size();
   // strings to print out all of the data gathered, point by point
   String xyPts = " ";
   // initialize variables
   int trackNr = 0;
   int displayTrackNr = 0;
   int line = 1;
   String output = "Line" + "\tTrack" + "\tRelative_Frame" + "\tX" +
    ↪  "\tY";
   // loop through all sperm tracks
   for (ListIterator iT = theTracks.listIterator(); iT.hasNext();) {
      int frame = 0;
      trackNr++;
      IJ.showProgress((double) trackNr / nTracks);
      IJ.showStatus("Analyzing Tracks...");
      List bTrack = (List) iT.next();
      // keeps track of the current track
      displayTrackNr++;
      ListIterator jT = bTrack.listIterator();
      Cell oldCell = (Cell) jT.next();
```

```
            Cell firstCell = new Cell();
            firstCell.copy(oldCell);
            // For each instant (Cell) in the track
            String outputline = "";
            for (; jT.hasNext();) {
               Cell newCell = (Cell) jT.next();
               xyPts = "\t" + displayTrackNr + "\t" + frame + "\t" + (int)
                ↪  newCell.x + "\t" + (int) newCell.y;
               frame++;
               oldCell = newCell;
               outputline += "\n" + line + xyPts;
               line++;
            }
            output += outputline;
         }
         IJ.showProgress(2); // To remove progresBar
         return output;
      }

      /**
       *
       * @param cells
       * @return KDTree array containing each cells component
       */
      public KDTree[] getKDTree(List[] cells) {
         KDTree res[] = new KDTree[cells.length];
         for (int i = 0; i < cells.length; i++) {
            res[i] = new KDTree(2);
            if (cells[i] != null) {
               for (ListIterator j = cells[i].listIterator(); j.hasNext();) {
                  Cell c = (Cell) j.next();
                  res[i].insert(new double[] { (int) c.x, (int) c.y }, c);
               }
            }
         }
         return res;
      }

   }
```

A.10. ComputerVision.java

```
package functions;

import data.Cell;
import ij.IJ;
import ij.ImagePlus;
import ij.ImageStack;
```

```
import ij.plugin.ChannelSplitter;
import ij.plugin.ContrastEnhancer;
import ij.process.AutoThresholder;
import ij.process.BinaryProcessor;
import ij.process.ByteProcessor;
import ij.process.ImageConverter;
import ij.process.ImageProcessor;
import ij.process.ImageStatistics;

/**
 * @author Carlos Alquezar and Jorge Yagüe
 *
 */
public class ComputerVision {

   /*****************************************************/

   /**
    * @param imp
    * @return
    */
   public double autoThresholdImagePlus(ImagePlus imp) {
      return autoThresholdImagePlus(imp, "Otsu"); // Otsu as a default
                                          // thresholding method
   }

   /*****************************************************/
   /**
    * @param imp
    * @param thresholdMethod
    * @return
    */
   public double autoThresholdImagePlus(ImagePlus imp, String
    ↪  thresholdMethod) {
      ImageProcessor ip = imp.getProcessor();
      double lowerThreshold = 0;
      ImageStatistics st = ip.getStatistics();
      long[] histlong = st.getHistogram();
      Utils utils = new Utils();
      int histogram[] = utils.convertLongArrayToInt(histlong);
      AutoThresholder at = new AutoThresholder();
      lowerThreshold = (double) at.getThreshold(thresholdMethod, histogram);
      // Upper threshold set to maximum
      double upperThreshold = 255;
      // Threshold image processor
      thresholdImageProcessor(ip, lowerThreshold, upperThreshold);
      return lowerThreshold;
   }

   /*****************************************************/
   /**
```

```
     * @param imp ImagePlus
     *
     *            This functions converts imp to grayscale.
     */
    public void convertToGrayscale(ImagePlus imp) {
        ImageConverter ic = new ImageConverter(imp);
        ic.convertToGray8();
    }

    /*****************************************************/
    /**
     * @param imp ImagePlus
     *
     *            This functions converts imp to grayscale.
     */
    public void convertToRGB(ImagePlus imp) {
        ImageConverter ic = new ImageConverter(imp);
        ic.convertToRGB();
    }

    /*****************************************************/
    /**
     * @param impColor
     * @return
     */
    public ImagePlus getBlueChannel(ImagePlus impColor) {
        ImagePlus[] images = ChannelSplitter.split(impColor);
        return images[2];
    }

    /*****************************************************/
    /**
     * @param impColor
     * @return
     */
    public ImagePlus getGreenChannel(ImagePlus impColor) {
        ImagePlus[] images = ChannelSplitter.split(impColor);
        return images[1];
    }

    /*****************************************************/
    /**
     * @param part
     * @param impGray
     * @param impTh
     * @return
     */
    public float getMeanGrayValue(Cell part, ImagePlus impGray, ImagePlus
    ↪  impTh) {

        ImageProcessor ipTh = impTh.getProcessor();
```

```
    ImageProcessor ipGray = impGray.getProcessor();
    int bx = (int) part.bx;
    int by = (int) part.by;
    int width = (int) part.width;
    int height = (int) part.height;
    float totalGray = 0;
    float totalPixels = 0;
    for (int x = bx; x < (width + bx); x++) {
        IJ.showStatus("scanning pixels...");
        for (int y = by; y < (height + by); y++) {
            int pixel = ipTh.get(x, y);
            if (pixel == 0) {
                totalGray += (float) ipGray.get(x, y);
                totalPixels++;
            }
        }
    }
    return totalGray / totalPixels;
}

/*******************************************************/
/**
 * @param impColor
 * @return
 */
public ImagePlus getRedChannel(ImagePlus impColor) {
    ImagePlus[] images = ChannelSplitter.split(impColor);
    return images[0];
}

/*******************************************************
 * /**
 *
 * @param imp
 */
public void outlineThresholdImage(ImagePlus imp) {
    convertToGrayscale(imp);
    ImageProcessor ip = imp.getProcessor();
    BinaryProcessor bp = new BinaryProcessor((ByteProcessor) ip);
    bp.outline();
}

/*******************************************************/
/**
 * @param imp
 * @param lowerThreshold
 */
public void thresholdImagePlus(ImagePlus imp, double lowerThreshold) {
    ImageProcessor ip = imp.getProcessor();
    // Upper threshold set to maximum
    double upperThreshold = 255;
```

```
        // Threshold image processor
        thresholdImageProcessor(ip, lowerThreshold, upperThreshold);
    }

    /*****************************************************/
    /**
     * @param ip
     * @param lowerThreshold
     * @param upperThreshold
     */
    public void thresholdImageProcessor(ImageProcessor ip, double
    ↪  lowerThreshold, double upperThreshold) {
        // Make binary
        int[] lut = new int[256];
        for (int j = 0; j < 256; j++) {
            if (j >= lowerThreshold && j <= upperThreshold)
                lut[j] = (byte) 0;
            else
                lut[j] = (byte) 255;
        }
        ip.applyTable(lut);
    }

    /**
     * Equalize imp histogram
     *
     * @param imp
     */
    public void equalize(ImagePlus imp) {
        ContrastEnhancer c = new ContrastEnhancer();
        c.equalize(imp);
    }

    /*****************************************************/
    /**
     * @param ImagePlus This function makes binary 'imp' applying an
↪   statistical
     *                  threshold
     */
    public void thresholdStack(ImagePlus imp) {

        ImageStack stack = imp.getStack();
        ImageProcessor ip = stack.getProcessor(1);
        ImageStatistics st = ip.getStatistics();
        double mean = st.mean;
        double std = st.stdDev;
        // Set threshold as mean + 2 x standard deviation
        double lowerThreshold = mean + 2 * std; // std factor: candidate to be
        ↪  a
                                        // parameter of the plugin
        double upperThreshold = 255;
```

```
        // Make binary
        int[] lut = new int[256];
        for (int j = 0; j < 256; j++) {
            if (j >= lowerThreshold && j <= upperThreshold)
                lut[j] = 0;
            else
                lut[j] = (byte) 255;
        }
        int nFrames = imp.getStackSize();
        for (int iFrame = 1; iFrame <= nFrames; iFrame++) {
            ip = stack.getProcessor(iFrame);
            ip.applyTable(lut);
        }
    }
}
```

A.11. VideoRecognition.java

```
package functions;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.ListIterator;

import data.Cell;
import data.Params;
import data.SerializableList;
import data.Square;
import ij.IJ;
import ij.ImagePlus;
import ij.ImageStack;
import ij.measure.Measurements;
import ij.measure.ResultsTable;
import ij.plugin.filter.ParticleAnalyzer;

/**
 * @author Carlos Alquezar y Jorge Yagüe
 *
 */
public class VideoRecognition implements Measurements {

    public VideoRecognition() {
    }

    /**
     * @param ImagePlus imp
```

```
     * @return
     */
    public SerializableList analyzeVideo(ImagePlus imp) {
        if (imp == null)
            return new SerializableList();
        // System.out.println("converToGrayScale...");
        ComputerVision cv = new ComputerVision();
        cv.convertToGrayscale(imp);
        // ************************************************************
        // * Automatic Thresholding
        // ************************************************************
        // System.out.println("thresholdStack...");
        cv.thresholdStack(imp);
        // ************************************************************
        // * Record particle positions for each frame in an ArrayList
        // ************************************************************
        // System.out.println("detectSpermatozoa...");
        List[] theCells = detectCells(imp);
        // ************************************************************
        // * Now assemble tracks out of the spermatozoa lists
        // * Also record to which track a particle belongs in ArrayLists
        // ************************************************************
        // System.out.println("identifyTracks...");
        SerializableList theTracks = idenfityTracks(theCells,
         ↪  imp.getStackSize());
        // Filtering tracks by length
        SignalProcessing sp = new SignalProcessing();
        theTracks = sp.filterTracksByLength(theTracks);
        // IJ.saveString(Utils.printXYCoords(theTracks),"");
        return theTracks;
    }

    /**
     * @param imp ImagePlus with cells and squares
     * @return 2D-ArrayList with all cells detected for each frame
     */
    public List[] detectCellsSquares(ImagePlus imp) {
        int nFrames = imp.getStackSize();
        ImageStack stack = imp.getStack();
        int options = 0;
        int measurements = MEAN + CENTROID + RECT + AREA + PERIMETER + FERET;
        // Initialize results table
        ResultsTable rt = new ResultsTable();
        rt.reset();
        int minSize = (int) (Params.minSize * Math.pow((1 /
         ↪  Params.micronPerPixel), 2));
        int maxSize = (int) (Params.maxSize * Math.pow((1 /
         ↪  Params.micronPerPixel), 2));
        // create storage for Spermatozoa positions
        List[] spermatozoa = new ArrayList[nFrames];
        // ************************************************************
```

```
    // * Record spermatozoa positions for each frame in an ArrayList
    // ************************************************************/
    for (int iFrame = 1; iFrame <= nFrames; iFrame++) {
       IJ.showProgress((double) iFrame / nFrames);
       IJ.showStatus("Identifying spermatozoa per frame...");
       spermatozoa[iFrame - 1] = new ArrayList();
       rt.reset();
       ParticleAnalyzer pa = new ParticleAnalyzer(options, measurements,
        ↪  rt, minSize, maxSize);
       pa.analyze(imp, stack.getProcessor(iFrame));
       float[] sxRes = rt.getColumn(ResultsTable.X_CENTROID);
       float[] syRes = rt.getColumn(ResultsTable.Y_CENTROID);
       float[] bxRes = rt.getColumn(ResultsTable.ROI_X);
       float[] byRes = rt.getColumn(ResultsTable.ROI_Y);
       float[] widthRes = rt.getColumn(ResultsTable.ROI_WIDTH);
       float[] heightRes = rt.getColumn(ResultsTable.ROI_HEIGHT);
       float[] areaRes = rt.getColumn(ResultsTable.AREA);
       float[] perimeterRes = rt.getColumn(ResultsTable.PERIMETER);
       float[] feretRes = rt.getColumn(ResultsTable.FERET);
       float[] minFeretRes = rt.getColumn(ResultsTable.MIN_FERET);
       if (sxRes == null) // Nothing detected
          continue;// jump to next frame
       for (int iPart = 0; iPart < sxRes.length; iPart++) {
          int dif = (int) (100
                * Math.abs((widthRes[iPart] - heightRes[iPart]) /
                 ↪  Math.max(widthRes[iPart], heightRes[iPart])));
          if (dif < 75) { // Filter each particle with height and width
           ↪  difference
             Cell aCell = new Cell();
             aCell.id = "***";
             aCell.x = sxRes[iPart];
             aCell.y = syRes[iPart];
             aCell.z = iFrame - 1;
             aCell.bx = bxRes[iPart];
             aCell.by = byRes[iPart];
             aCell.width = widthRes[iPart];
             aCell.height = heightRes[iPart];
             aCell.total_area = areaRes[iPart];
             aCell.total_perimeter = perimeterRes[iPart];
             aCell.total_feret = feretRes[iPart];
             aCell.total_minFeret = minFeretRes[iPart];
             spermatozoa[iFrame - 1].add(aCell);
          }
       }
    }
    IJ.showProgress(2); // To remove progresBar
    return spermatozoa;
}

/*****************************************************/
/**
```

```
     * @param imp ImagePlus
     * @return 2D-ArrayList with all cells detected for each frame
     */
    public List[] detectCells(ImagePlus imp) {
        int nFrames = imp.getStackSize();
        ImageStack stack = imp.getStack();
        int options = 0; // set all PA options false
        int measurements = MEAN + CENTROID + RECT + AREA + PERIMETER + FERET;
        // Initialize results table
        ResultsTable rt = new ResultsTable();
        rt.reset();
        int minSize = (int) (Params.minSize * Math.pow((1 /
         ↪  Params.micronPerPixel), 2));
        int maxSize = (int) (Params.maxSize * Math.pow((1 /
         ↪  Params.micronPerPixel), 2));
        // create storage for Spermatozoa positions
        List[] spermatozoa = new ArrayList[nFrames];
        // ************************************************************
        // * Record spermatozoa positions for each frame in an ArrayList
        // ************************************************************/
        for (int iFrame = 1; iFrame <= nFrames; iFrame++) {
            IJ.showProgress((double) iFrame / nFrames);
            IJ.showStatus("Identifying spermatozoa per frame...");
            spermatozoa[iFrame - 1] = new ArrayList();
            rt.reset();
            ParticleAnalyzer pa = new ParticleAnalyzer(options, measurements,
             ↪  rt, minSize, maxSize);
            pa.analyze(imp, stack.getProcessor(iFrame));
            float[] sxRes = rt.getColumn(ResultsTable.X_CENTROID);
            float[] syRes = rt.getColumn(ResultsTable.Y_CENTROID);
            float[] bxRes = rt.getColumn(ResultsTable.ROI_X);
            float[] byRes = rt.getColumn(ResultsTable.ROI_Y);
            float[] widthRes = rt.getColumn(ResultsTable.ROI_WIDTH);
            float[] heightRes = rt.getColumn(ResultsTable.ROI_HEIGHT);
            float[] areaRes = rt.getColumn(ResultsTable.AREA);
            float[] perimeterRes = rt.getColumn(ResultsTable.PERIMETER);
            float[] feretRes = rt.getColumn(ResultsTable.FERET);
            float[] minFeretRes = rt.getColumn(ResultsTable.MIN_FERET);
            if (sxRes == null) // Nothing detected
                continue;// jump to next frame
            for (int iPart = 0; iPart < sxRes.length; iPart++) {
                Cell aCell = new Cell();
                aCell.id = "***";
                aCell.x = sxRes[iPart];
                aCell.y = syRes[iPart];
                aCell.z = iFrame - 1;
                aCell.bx = bxRes[iPart];
                aCell.by = byRes[iPart];
                aCell.width = widthRes[iPart];
                aCell.height = heightRes[iPart];
                aCell.total_area = areaRes[iPart];
```

```
            aCell.total_perimeter = perimeterRes[iPart];
            aCell.total_feret = feretRes[iPart];
            aCell.total_minFeret = minFeretRes[iPart];
            spermatozoa[iFrame - 1].add(aCell);
        }
    }
    IJ.showProgress(2); // To remove progresBar
    return spermatozoa;
}

/**
 * @param imp ImagePlus
 * @return 2D-ArrayList with all squares detected for each frame
 */
public List[] detectSquares(ImagePlus imp) {

    int nFrames = imp.getStackSize();
    ImageStack stack = imp.getStack();
    int options = ParticleAnalyzer.INCLUDE_HOLES;
    int measurements = RECT + AREA;
    // Initialize results table
    ResultsTable rt = new ResultsTable();
    rt.reset();
    float sideR = (float) (Params.sideS / Params.micronPerPixel);
    float areaS = sideR * sideR;
    int minSize = (int) (areaS * 0.75);
    int maxSize = (int) (areaS * 1.25);
    // create storage for Square positions
    List[] square = new ArrayList[nFrames];
    // ************************************************************
    // * Record square positions for each frame in an ArrayList
    // ************************************************************/
    for (int iFrame = 1; iFrame <= nFrames; iFrame++) {
        IJ.showProgress((double) iFrame / nFrames);
        IJ.showStatus("Identifying square per frame...");
        square[iFrame - 1] = new ArrayList();
        rt.reset();
        ParticleAnalyzer pa = new ParticleAnalyzer(options, measurements,
        ↪  rt, minSize, maxSize);
        pa.analyze(imp, stack.getProcessor(iFrame));
        float[] bxRes = rt.getColumn(ResultsTable.ROI_X);
        float[] byRes = rt.getColumn(ResultsTable.ROI_Y);
        float[] widthRes = rt.getColumn(ResultsTable.ROI_WIDTH);
        float[] heightRes = rt.getColumn(ResultsTable.ROI_HEIGHT);
        float[] areaRes = rt.getColumn(ResultsTable.AREA);
        if (bxRes == null) // Nothing detected
            continue;// jump to next frame
        for (int i = 0; i < bxRes.length; i++) {
            // Filter each particle with square side length
            if ((widthRes[i] <= 1.25 * sideR && widthRes[i] >= 0.75 * sideR)
            ↪  && heightRes[i] <= 1.25 * sideR
```

```
                && heightRes[i] >= .75 * sideR) {
            Square aSquare = new Square();
            aSquare.id = "***";
            aSquare.x = bxRes[i];
            aSquare.y = byRes[i];
            aSquare.width = widthRes[i];
            aSquare.height = heightRes[i];
            aSquare.area = areaRes[i];
            square[iFrame - 1].add(aSquare);
         }
      }
   }
   IJ.showProgress(2); // To remove progresBar
   return square;
}

/*****************************************************/
/**
 * @param cells   2D-ArrayList with all cells detected for each frame
 * @param nFrames
 * @return 2D-ArrayList with all tracks detected
 */
public SerializableList idenfityTracks(List[] cells, int nFrames) {

   // int nFrames = imp.getStackSize();
   SerializableList theTracks = new SerializableList();
   int trackCount = 0;
   if (cells == null)
      return theTracks;
   for (int i = 0; i <= (nFrames - 1); i++) {
      IJ.showProgress((double) i / nFrames);
      IJ.showStatus("Calculating Tracks...");
      if (cells[i] == null)// no spermatozoa detected in frame i
         continue; // jump to next frame
      for (ListIterator j = cells[i].listIterator(); j.hasNext();) {
         Cell aCell = (Cell) j.next();
         if (!aCell.inTrack) {
            // This must be the beginning of a new track
            List aTrack = new ArrayList();
            trackCount++;
            aCell.inTrack = true;
            aCell.trackNr = trackCount;
            aTrack.add(aCell);
            //
             ↪  ************************************************************
            // search in next frames for more Spermatozoa to be added to
            // track
            //
             ↪  ************************************************************
            boolean searchOn = true;
            Cell oldCell = new Cell();
```

```
Cell tmpCell = new Cell();
oldCell.copy(aCell);
// *
// * For each frame
// *
for (int iF = i + 1; iF <= (nFrames - 1); iF++) {
   boolean foundOne = false;
   Cell newCell = new Cell();
   // *
   // * For each Cell in this frame
   // *
   for (ListIterator jF = cells[iF].listIterator();
    ↪  jF.hasNext() && searchOn;) {
      Cell testCell = (Cell) jF.next();
      float distance = testCell.distance(oldCell);
      // record a Cell when it is within the search
      // radius, and when it had not yet been claimed by
       ↪  another
      // track
      if ((distance < (Params.maxDisplacement /
       ↪  Params.micronPerPixel)) && !testCell.inTrack) {
         // if we had not found a Cell before, it is easy
         if (!foundOne) {
            tmpCell = testCell;
            testCell.inTrack = true;
            testCell.trackNr = trackCount;
            newCell.copy(testCell);
            foundOne = true;
         } else {
            // if we had one before, we'll take this one if
             ↪  it is
            // closer. In any case, flag these Spermatozoa
            testCell.flag = true;
            if (distance < newCell.distance(oldCell)) {
               testCell.inTrack = true;
               testCell.trackNr = trackCount;
               newCell.copy(testCell);
               tmpCell.inTrack = false;
               tmpCell.trackNr = 0;
               tmpCell = testCell;
            } else {
               newCell.flag = true;
            }
         }
      } else if (distance < (Params.maxDisplacement /
       ↪  Params.micronPerPixel)) {
         // this Cell is already in another track but
         // could have been part of this one
         // We have a number of choices here:
         // 1. Sort out to which track this Cell really
         // belongs (but how?)
```

```
                        // 2. Stop this track
                        // 3. Stop this track, and also delete the remainder
                        ↪  of
                        // the other one
                        // 4. Stop this track and flag this Cell:
                        testCell.flag = true;
                    }
                }
                if (foundOne)
                    aTrack.add(newCell);
                else
                    searchOn = false;
                oldCell.copy(newCell);
            }
            theTracks.add(aTrack);
        }
    }
}
IJ.showProgress(2); // To remove progresBar
return theTracks;
}

}
```