Trabajo de Fin de Grado

Morfología de imágenes neuronales

Morphology of neuroimages

Adrián Hernández González

La Laguna, 9 de junio de 2015

D. Albano Fernández González, con N.I.F. 45.448.401-X profesor Titular de Universidad adscrito al Departamento de Nombre del Departamento de la Universidad de La Laguna, como tutor.

C E R T I F I C A

Que la presente memoria titulada:

"Morfología de imágenes neuronales."

ha sido realizada bajo su dirección por D. Adrián Hernández González, con N.I.F. 78.645.085-N.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 9 de junio de 2015.

Agradecimientos

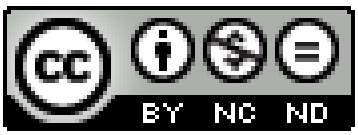
A D. Albano Fernández González por su labor en la dirección de este proyecto.

A los profesores de la Escuela Superior de Ingeniería y Tecnología de la Universidad de La Laguna.

A todas aquellas personas que, de alguna manera, me han ayudado y apoyado en la realización de este proyecto.

A mi familia, amigos y compañeros.

Licencia



Resumen

Los objetivos principales de este trabajo han sido: diseñar e implementar herramientas software que permitan el cálculo y representación de mapas de interconexión de áreas cerebrales a partir de imágenes de difusión tensorial.

El estándar de imágenes utilizado es el DICOM 3.0, puesto que permite la transmisión, tratamiento e impresión de imágenes biomédicas y que incluyen los metadatos necesarios para caracterizar el estudio realizado.

Las herramientas desarrolladas han sido creadas para su integración dentro de la aplicación ImageJ, en forma de plugin. Para la realización de los plugin se ha trabajado con librerías propias de IJ y JAVA para poder así combinar las funcionalidades propias de ImageJ y las de nueva creación que aportan las herramientas desarrolladas. Todo ha sido desarrollado enteramente en JAVA ayudándonos del entorno de desarrollo Eclipse y puede utilizarse en los sistemas operativos más habituales.

Palabras clave: mapas de interconexión, imágenes neuronales, DICOM, ImageJ, Java y plugin.

Abstract

The main aims of this dissertation have been: to design and implement software tools that will allow us to compute and represent interconnected regions of the brain from diffusion tensor images.

The image standard used is the DICOM 3.0 because it allows the transmission, processing and printing of biomedical imagery, and it also includes the corresponding metadata about the performed study.

The developed tools have been created to work within the ImageJ application as a plugin. For the development of the plugins we have worked with IJ and Java libraries so we could combine ImageJ's own functions and those newly created in the developed tools. The project has been developed entirely in JAVA, using the development environment Eclipse and it can run on most operating systems.

Keywords: interconnection maps, neural imaging, DICOM, ImageJ, Java and plugins.

Índice General

Índice de figuras

Capítulo 1. Introducción

En las últimas décadas se ha producido un relevante avance de las pruebas diagnósticas en medicina, mejorando tanto la detección de diversas patologías como la investigación médica. Una de las técnicas de observación más complejas y que permite obtener información de alta resolución del interior del cuerpo humano es la basada en imágenes por resonancia magnética nuclear (MRI – Magnetic Resonance Imaging). En concreto, la MRI de difusión (dMRI) permite evaluar los procesos de difusión de las moléculas, principalmente de agua, en los tejidos. Las imágenes de tensores de difusión (DTI – Diffusion Tensor Imaging) permiten evaluar la anisotropía de los tejidos a partir de la caracterización de la difusión tridimensional de las moléculas de agua en función de su localización espacial. Las diferentes capacidades de difusión medidas permiten estimar la estructura de los tejidos.

En los tejidos fibrosos la difusión del agua es favorecida en la dirección paralela a la orientación de las fibras, por el contrario, se ve desfavorecida en las direcciones perpendiculares a las mismas, presentando una clara anisotropía. En el sistema nervioso central, la anisotropía es más notable en las zonas de materia blanca (compuesta de fibras nerviosas mielinizadas) y la difusión presenta mayor isotropía en la materia gris (compuesta por los somas y cuerpos neuronales, que no poseen mielina) y en el líquido cefalorraquídeo (que baña el encéfalo y la medula espinal). La dirección de mayor difusividad se asume que es paralela a la dirección del tracto en regiones homogéneas de materia blanca. Esto es aprovechado por ciertos algoritmos de tractografía para detectar los principales tractos neuronales.

Para estimar el comportamiento espacial de la difusividad de las moléculas de agua, un gradiente del campo magnético es aplicado en diferentes direcciones, midiendo entonces las propiedades de los tejidos en todas esas direcciones o ángulos de observación. En las medidas básicas, los datos se obtienen en tres direcciones. Cuando aumenta el número de ángulos de

observación la cantidad de información a procesar aumenta considerablemente y, a menudo, las propiedades espaciales de cada uno de los voxels de la imagen son simplificadas, asumiendo que pueden modelarse como un elipsoide, cuyos ejes coinciden con los ejes principales de la distribución medida. Esta simplificación tiene algunas ventajas, como una mayor inmunidad al ruido de medida, pero también algunos inconvenientes, como la imposibilidad de detectar bifurcaciones en los tractos neuronales a partir de imágenes con una resolución angular elevada. En este proyecto se empleará la resolución angular completa proporcionada por las imágenes de difusión tensorial.

1.1 Objetivos del proyecto

El fin principal de este proyecto ha sido realizar el diseño e implementación de herramientas software que permitan el cálculo y representación de mapas de interconexión de áreas cerebrales a partir de imágenes de difusión tensorial.

Se pretende tener dos alternativas para llevar a cabo el cálculo y la generación de caminos. Una que se base en la búsqueda del camino donde exista mayor difusividad partiendo de un punto dado de una de las imágenes de nuestra secuencia de trabajo. Y otra que utilice técnicas de probabilidad basadas en el método de Monte Carlo, de forma que el salto de un voxel a otro es probabilístico y proporcional a la difusividad en cada una de las direcciones.

La representación de los resultados finales se hará mediante imágenes en 3D.

Para alcanzar el objetivo final del proyecto se han tenido que ir alcanzando otros objetivos secundarios pero no menos importantes como:

- Comprensión del estándar DICOM 3.0.
- Manejo de la aplicación ImageJ.
- Aprender a implementar, compilar y ejecutar plugins en ImageJ.
- Familiarizarse con librerías de Java e ImageJ.
- Adquirir conceptos básicos relacionados con la medicina y la biología, dada la naturaleza de las imágenes.

- Estudio de la representación 3D para poder representar los resultados finales y obtener así unas conclusiones más claras y precisas.

1.2 Marco del proyecto.

Para alcanzar los objetivos planteados, se fijaron una serie de tareas y objetivos intermedios, que se indican a continuación:

- En primer lugar, fijar la línea de trabajo y cerrar el planteamiento del diseño del proyecto y del software a desarrollar.
- En segundo lugar, revisión de las características de los formatos de las imágenes DTI, almacenadas en múltiples imágenes DICOM.
- En tercer lugar, estudio de la aplicación ImageJ y lenguaje Java y búsqueda de bibliografía y librerías correspondientes.
- En cuarto lugar, implementación de módulo de lectura de formatos.
- En quinto lugar, definición de los algoritmos de procesamiento a implementar y desarrollo de los mismos.
- Y en sexto lugar, evaluación de los resultados y estudio de sistemas de representación en 3D propios de ImageJ.

Como puede observarse, el trabajo ha sido realizado en diferentes etapas, la primera de las cuales consistió en determinar cuál iba a ser la línea de trabajo a seguir.

La siguiente etapa fue el estudio del estándar DICOM 3.0, base fundamental de este trabajo.

Luego se estudió tanto la aplicación ImageJ y sus funcionalidades como el lenguaje Java y sus tipos y estructuras de datos que se adecuaban mejor a los propósitos del proyecto.

Una vez conocido cómo está estructurado el estándar en el que están guardas las imágenes con las que trabajamos y las prestaciones que nos aportan tanto ImageJ como Java, empezamos con la implementación en Java de un primer plugin, para integrarlo dentro de ImageJ, que permitiera abrir imágenes DICOM.

Teniendo cada vez más afianzados los conocimientos sobre ImageJ y Java, en la quinta etapa, se realizó la implementación de los algoritmos de generación de caminos.

Por último y dada la necesidad de que nuestra herramienta tuviese una representación clara y precisa de los resultados finales, en la sexta etapa, se estudió la representación en 3D de las imágenes resultantes de los algoritmos, utilizando ImageJ.

Todo lo que se ha implementado ha sido utilizando el entorno de trabajo Eclipse para Java y ImageJ y el editor NotePad++, bajo plataforma Intel y sistema operativo Windows 8.1.

1.3 Estructura del proyecto

Esta memoria se encuentra dividida tal como sigue:

- Capítulo 1
  Introducción a la morfología de las imágenes neuronales y definición de los objetivos y alcance del proyecto.
- Capítulo 2
  Desarrollo de la base teórica del estándar DICOM, aplicación ImageJ y lenguaje Java.
- Capítulo 3
  Introducción y definición de la API ImageJ.
- Capítulo 4
  Presentación y estudio de los algoritmos desarrollados, atendiendo a su implantación y comportamiento.
- Capítulo 5
  Presentación y análisis de los resultados.
- Capítulo 6
  Conclusión y líneas de trabajo futuras de esta memoria redactada en lengua española.
- Capítulo 7
  Conclusión y líneas de trabajo futuras de esta memoria redactada en lengua inglesa.

Capítulo 2. Antecedentes

Los equipos de resonancia magnética proporcionan un software específico, que depende de cada marca y que permite realizar las operaciones básicas de pre-procesamiento de los datos obtenidos mediante la técnica de imágenes por resonancia magnética con difusión tensorial (DTI o ITD). Además, los especialistas en neurología utilizan software desarrollado por la misma marca o software general de presentación de imágenes neurológicas, que permiten evaluar sólo algunos aspectos de las mismas.

Sin embargo este tipo de software no suele permitir su mejora o ampliación, con el fin de que los especialistas puedan introducir algoritmos de procesamiento o análisis que les permita estudiar determinados aspectos de los tractos neuronales para alguna aplicación concreta. En el presente proyecto se ha desarrollado una herramienta software que permite realizar dichas tareas, incluyendo herramientas básicas de representación de los resultados de los algoritmos implementados.

Para la realización del proyecto se emplearon, como base para el manejo de imágenes en formato DICOM, las librerías de la aplicación ImageJ y se evaluó la posible aplicación de otras librerías específicas desarrolladas en lenguaje Java.

2.1 Resonancia Magnética

Las técnicas de Imagen por Resonancia Magnética (IMR) se han convertido en una de las modalidades de adquisición y representación de datos médicos más importante para la prevención, diagnóstico y control de desórdenes neurológicos como la esclerosis múltiple, la isquemia o el Alzheimer.

Para la obtención de las imágenes se utiliza el fenómeno de la resonancia magnética que es una prueba médica no invasiva en la que se utiliza un campo magnético y ondas electromagnéticas para obtener imágenes detalladas

de los órganos y las estructuras del cuerpo. En la resonancia magnética, al contrario que en otras pruebas médicas, como los rayos X, no se utiliza radiación perjudicial para los seres vivos.

La resonancia magnética permite distinguir entre la sustancia blanca, compuesta por fibras nerviosas cubiertas de mielina (que contiene sobre todo axones, encargados de transmitir los impulsos nerviosos entre las células); y sustancia gris, formada por dendritas y cuerpos neuronales que no poseen mielina y se relacionan con el procesamiento de información. De este modo, ambas sustancias se diferencian por la presencia o no de mielina, materia que rodea la fibra nerviosa y posibilita la transmisión veloz de los impulsos nerviosos entre las diferentes partes del cuerpo.

Los componentes más básicos de una unidad de resonancia magnética son un gran imán con forma de anillo que suele tener un túnel en el centro, aunque también existen maquinas abiertas, un radiotransmisor, una bobina receptora de radiofrecuencias y un ordenador. Los pacientes se ubican en una camilla que se desliza atravesando el imán en forma de anillo.

Figura 2.1. Unidad de resonancia magnética.

Durante la realización de la prueba, las ondas electromagnéticas ejercen su efecto sobre los núcleos de hidrógeno del agua en el cuerpo afectando a su posición, lo cual es detectado por la bobina, y el resultado de estas medidas es

enviado al ordenador. El ordenador realiza millones de cálculos que crean imágenes claras de cortes transversales del organismo. La intensidad de la señal es proporcional al campo magnético y se requieren campos magnéticos muy altos para conseguir imágenes de calidad.

Los datos obtenidos se pueden convertir en imágenes tridimensionales (3D) de la zona analizada. Esto ayuda a detectar de forma más clara, directa y nítida problemas en el organismo.

2.1.1 Resonancia magnética por difusión

En los años 90 surgió un nuevo fenómeno de IMR, el denominado Resonancia Magnética por Difusión (RMD o dMRI). A diferencia de la resonancia magnética tradicional, ésta permite estudiar sólo la sustancia blanca. Aportando algo novedoso, de gran utilidad y aplicabilidad en estudios, sobre todo, del cerebro. El fenómeno de IMR sirve de base para la técnica de Imagen por Resonancia Magnética de difusión.

La RMD se basa en la difusión de las moléculas de agua por el tejido cerebral, lo que permite identificar la estructura nerviosa del cerebro. La difusión libre del agua tiene lugar en todas las direcciones posibles. Se dice entonces que se está realizando una difusión isótropa. Si el agua difunde en un medio con barreras, la difusión deja de ser uniforme provocando anisotropía. En concreto, este tipo de resonancia magnética permite cuantificar el grado de anisotropía, la propiedad del tejido cerebral que depende de la direccionalidad de las moléculas de agua y de la integridad de las fibras de la sustancia blanca.

Muchas cosas pueden actuar como barrera: membranas celulares, axones, mielina, etc. pero en la sustancia blanca la principal barrera es la vaina de mielina de los axones. Éstos oponen una barrera a la difusión perpendicular y una ruta para la difusión paralela en el sentido de la orientación de las fibras.

Se espera que la difusión anisótropa se incremente en áreas donde se presenten axones maduros altamente ordenados. En las afecciones en las que se distorsiona la mielina o la estructura de los axones, como en traumatismos

físicos, tumores e inflamaciones se reduce la anisotropía, ya que las barreras a la difusión quedan afectadas por destrucción o desorganización.

2.1.2 Tractografía

Una tractografía es un procedimiento que se usa para trazar los tractos neuronales. Utiliza la ya mencionada técnica de imagen por resonancia magnética de difusión y análisis de imágenes asistido por ordenador. El resultado se presenta en imágenes 2D y/o 3D.

En concreto la tractografía trabaja con unas imágenes especiales que se denominan imágenes de difusión tensorial (ITD) que permiten cuantificar el grado de anisotropía de los protones de agua en los tejidos.

Como se ha mencionado, la anisotropía es la propiedad del tejido cerebral normal que depende de la direccionalidad de las moléculas del agua y de la integridad de las fibras de sustancia blanca.

La anisotropía se mide mirando la capacidad de difusión de cada uno de los puntos de la imagen, que es almacenada en los voxeles de las matrices tridimensionales que se obtienen mediante la realización de la RMD. El conjunto de matrices tridimensionales de cortes verticales del cerebro se obtiene gracias a que durante la RMD se va cambiando el ángulo del campo magnético y se toman un número de imágenes, luego, se repite el proceso con un nuevo ángulo. Dicho proceso se puede repetir n veces dependiendo del aparato con el que se realice la prueba. La secuencia de imágenes con la que hemos trabajado durante este TFG consta de 34 conjuntos de 27 imágenes cada uno, en total, suman 918 imágenes.

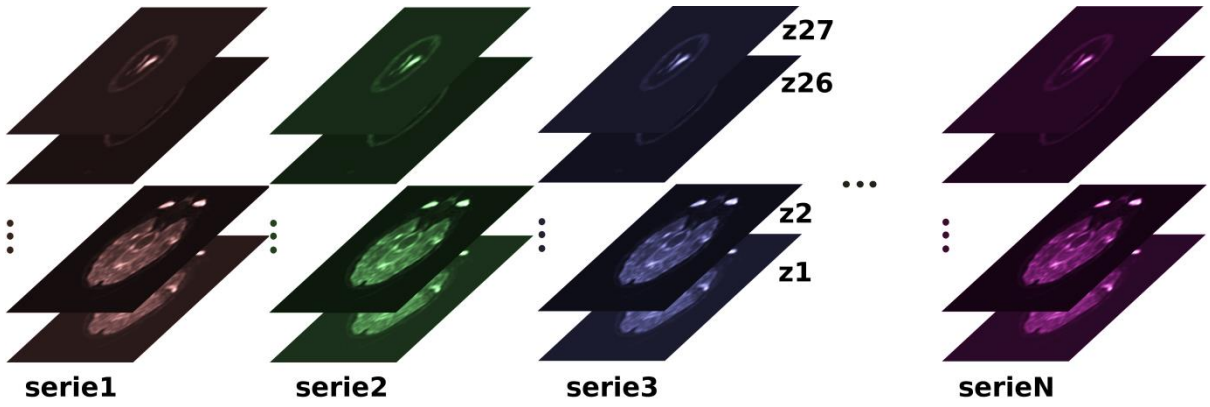


Figura 2.2. Esquema de la secuencia de imágenes que genera una RMD.

La tractografía ayuda a representar la complicada red tridimensional formada por conexiones a corta distancia entre las diferentes áreas corticales y subcorticales del cerebro. Antes de la aparición de este procedimiento el estudio de este tipo de tractos se había realizado mediante técnicas biológicas y de histoquímica en individuos fallecidos, sin embargo, las pruebas existentes aplicables sobre individuos vivos no permiten identificar los tractos cerebrales. Esta dificultad explica lo pobre de su descripción en los atlas de neuroanatomía y lo escasamente comprendido de sus funciones.

Se ha demostrado que el uso de tractografías puede llegar a ser de gran interés y ha generado expectativas sobre su utilidad diagnóstica y pronóstico en el accidente cerebrovascular, la esclerosis, así como también en ciertas enfermedades mentales, y particularmente en el estudio de los tumores cerebrales.

2.2 DICOM.

La aparición y uso de las distintas técnicas de imagen por resonancia junto con la creciente tendencia de tener un mundo digitalizado, ha llevado tanto a organizaciones médicas como a fabricantes de aparatos médicos de diagnosis a buscar una manera de estandarizar, sobre todo, el diagnóstico por imagen, estableciendo un formato único para los datos y las comunicaciones médicas, generalizando los protocolos. Uno de los estándares más exitosos hasta la fecha es DICOM (siglas de Digital Imaging and Communications in Medicine).

DICOM es un estándar de comunicación entre sistemas de información utilizado a nivel mundial para el intercambio de pruebas médicas, pensado para su manejo, visualización, almacenamiento, impresión y transmisión. Su aparición supuso acabar con los problemas de interoperabilidad entre distintos tipos de dispositivos. Incluye la definición de un formato de fichero y de un protocolo de comunicación de red. El protocolo de comunicación es un protocolo de aplicación que usa TCP/IP para la comunicación entre sistemas. Los ficheros DICOM pueden intercambiarse entre dos entidades que tengan capacidad de recibir imágenes y datos de pacientes en formato DICOM.

Una imagen médica por sí misma no aporta suficiente información. Para que sea correctamente interpretada es necesario que vaya acompañada de datos del paciente y del proceso de adquisición, incluyendo datos de la máquina y de las circunstancias de cómo se realizó la prueba.

2.2.1 Formato de fichero

Los ficheros DICOM constan de una cabecera mixta con campos estandarizados y campos de libre elección, y un cuerpo con la imagen en sí. Cada campo va etiquetado en la cabecera del fichero DICOM. Es en estos campos donde se almacena la información sobre el paciente (identificación y datos demográficos), el estudio donde se encuadra la toma de la imagen, la serie a la que pertenece la imagen e información sobre la propia máquina.

La imagen y su información asociada se almacenan en objetos. Los objetos están compuestos por entidades de información (hay entidades de paciente, de estudio, de serie, de equipo, de imagen...) que a su vez se componen de uno o varios módulos, que a su vez contienen varios atributos. Un atributo se define con etiqueta, nombre, palabra clave, la representación del valor (tipo de datos y el formato) y multiplicidad del valor (especifica el número de valores que pueden ser codificados en ese campo).

En la siguiente figura se puede apreciar una síntesis de los metadatos más importantes y que hemos usado durante la implementación de la herramienta que nos ocupa.

```
XFV.06.06.2013-0008-0001-00001.dcm
.
.
.
0008,0020  Study Date: 20130606
0008,0021  Series Date: 20130606
0008,0022  Acquisition Date: 20130606
0008,0023  Image Date: 20130606
0008,0030  Study Time: 182939
0008,0031  Series Time: 191540
0008,0032  Acquisition Time: 191540
0008,0033  Image Time: 191540
0008,0050  Accession Number:
0008,0060  Modality: MR
0008,0070  Manufacturer: GE MEDICAL SYSTEMS
0008,0080  Institution Name: IMETISA
.
.
.
0010,0010  Patient's Name: XFV.06.06.2013
0010,0020  Patient ID: geservice
0010,0030  Patient's Birth Date:
0010,0040  Patient's Sex: F
0010,1010  Patient's Age: 032Y
0010,1030  Patient's Weight: 80
0010,21B0  Additional Patient History:
.
.
.
0019,10BB  ---: 0.000000
0019,10BC  ---: 0.000000
0019,10BD  ---: 0.000000
.
.
.
0020,0013  Image Number: 1
.
.
.
0020,1002  Images in Acquisition: 918
.
.
.
Calibration function: y = a+bx
  a: -32768.000000
  b: 1.000000
  Unit: "Gray Value"
.
.
.
```

Figura 2.3. Ejemplo de los metadatos de una imagen DICOM.

2.3 ImageJ

ImageJ es un programa de procesamiento de imágenes de dominio público escrito en Java inspirado por el NIH Image para Macintosh. Se puede ejecutar, como una applet en línea o como una aplicación descargable, en cualquier ordenador con una máquina virtual de Java 1.1 o posterior. Las distribuciones descargables están disponibles para Windows, Mac OS, Mac OS X y Linux.

Con este programa se pueden mostrar, editar, analizar, procesar, guardar e imprimir imágenes de 8 bits, 16 bits y 32 bits. Puede leer muchos formatos de imagen, incluyendo TIFF, GIF, JPEG, BMP, DICOM y FITS, entre otros. Es capaz de manejar secuencias de imágenes estructurándolas en forma de pila que luego compartirán una misma ventana en la que se permite navegar por todas las imágenes de la serie (como se puede apreciar en la figura 1.2). Es una aplicación multiproceso, por lo que las operaciones que consumen mucho tiempo, como la lectura de archivos de imágenes, se puede realizar en paralelo con otras operaciones. Se puede abrir un gran número de imágenes al mismo tiempo, que será limitado solamente por la memoria disponible.

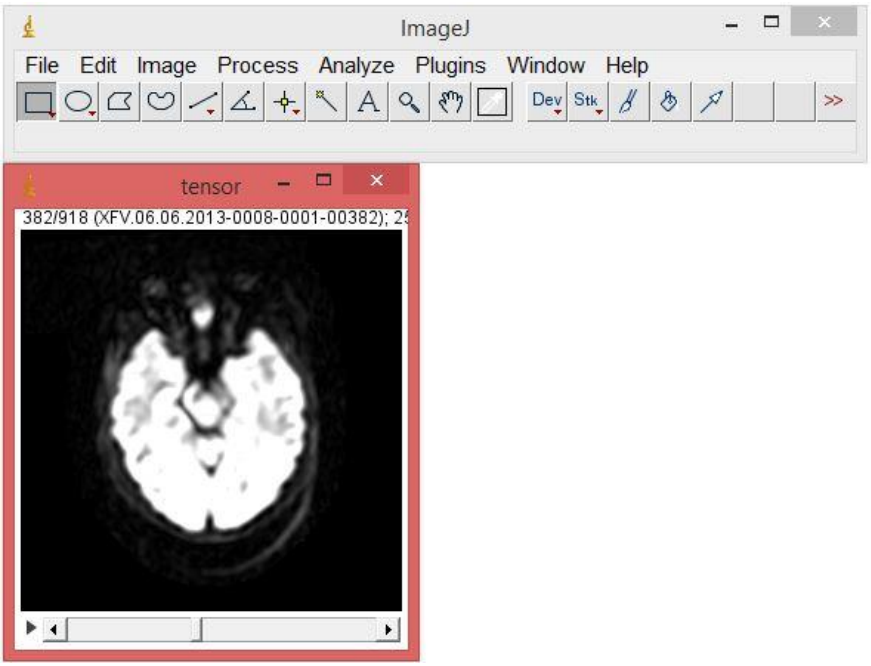


Figura 2.4.Ejemplo de secuencia abierta en ImageJ.

Entre las características más destacadas encontramos que puede calcular el área y las estadísticas de valor de píxel de regiones de interés definidas por el usuario, medir distancias y ángulos, crear histogramas de densidad y gráficos

de línea de perfil. Además, es compatible con las funciones estándar de procesamiento de imágenes tales como operaciones lógicas y aritméticas entre imágenes, manipulación de contraste, convolución, análisis de Fourier, suavizado, detección de bordes y filtrado de mediana, etc. Incluye, además, diferentes herramientas para realizar operaciones geométricas.

ImageJ fue diseñado con una arquitectura abierta que proporciona extensibilidad a través de plugins de Java y macros. Se pueden desarrollar plugins de adquisición de imágenes personalizado, de análisis y de procesamiento. Para ello, puede utilizarse el editor incluido en ImageJ y un compilador Java. Los plugins escritos por el usuario permiten resolver muchos problemas de procesamiento y análisis de imágenes que no son abordados por las funcionalidades que vienen por defecto instaladas en ImageJ. La posibilidad de extender con plugins y la existencia de entorno de desarrollo integrado en ImageJ la han convertido en una plataforma popular para la enseñanza del procesamiento de imagen tanto en entornos académicos como profesionales.

2.3.1 Plugins

Los plugins son más potentes que las macros puesto que éstas sólo pueden ejecutar una serie de comandos propios de ImageJ que hayan sido "grabados" previamente. Un plugin es código escrito en Java que sirve para ampliar la lista de funcionalidades nativas de la aplicación. Al ser clases de Java podemos usar todas las características del lenguaje, acceder a la API de ImageJ y a otras API estándar y externas a ImageJ. Esto nos abre un campo amplio de posibilidades a realizar.

El uso más común para los plugins es el de crear filtros para el análisis o procesamiento de imágenes o stacks/pilas. También para leer y escribir sobre formatos no soportados por la aplicación.

Existen dos diferentes tipos de plugins:

- PlugIn: no requiere de una imagen abierta para iniciar el plugin.
- PlugInFilter: requiere una imagen abierta y activa, el plugin se ejecutar sobre dicha imagen.

Durante el presente trabajo hemos desarrollado uno del tipo PlugIn. Para empezar a desarrolar un plugin de este tipo sólo se necesita indicar que la clase de Java implementa la interfaz PlugIn y se debe añadir un método llamado "run".

```
void run(java.lang.String arg)
```

Este método ejecuta el plugin, así que lo que se implemente aquí es lo que el plugin en realidad hace. "arg" es un String que se le pasa como argumento de entrada al plugin, y puede estar vacío.

Si fuese necesario, se pueden añadir más métodos a la clase aparte de este método "run" y conseguir así extender las tareas que podemos realizar con un mismo plugin.

2.4 Java

Java surgió en 1991 cuando un grupo de ingenieros de Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada. Se ejecuta sobre una "maquina hipotética o virtual" denominada Java Virtual Machine (JVM). Es la JVM quien interpreta el código neutro convirtiéndolo a código particular de la CPU utilizada. Cualquier aplicación que se desarrolle se apoya en un gran número de clases preexistentes.

La ejecución de programas en Java tiene muchas posibilidades: ejecución como aplicación independiente (Stand-alone Application), ejecución como applet, ejecución como servlet, etc...

Java permite fácilmente el desarrollo tanto de arquitecturas cliente-servidor como de aplicaciones distribuidas, consistentes en crear aplicaciones capaces de conectarse a otros ordenadores y ejecutar tareas en varios de ellos simultáneamente.

Capítulo 3.
API ImageJ

Una API (Application Programming Interface) es un conjunto de reglas (código) y especificaciones que las aplicaciones pueden seguir para comunicarse entre ellas, sirviendo de interfaz entre programas diferentes, de la misma manera en que la interfaz de usuario facilita la interacción humano-software.

Las APIs pueden servir para comunicarse con el sistema operativo, con bases de datos o con protocolos de comunicaciones. En los últimos años, las APIs más usadas están íntimamente relacionadas con las redes sociales o con otras plataformas online como Google Maps o WordPress. El uso de estas APIs está muy extendido, puesto que, facilitan las comunicaciones entre aplicaciones y páginas web con las propias redes sociales, convirtiendo el social media marketing en algo más sencillo, más rastreable y, por tanto, más rentable.

Las APIs son valiosas, ante todo, porque permiten hacer uso de funciones ya existentes en otro software (o de la infraestructura ya existente en otras plataformas) para no estar constantemente trabajando sobre lo mismo, reutilizando así código que se sabe que está probado y que funciona correctamente. En el caso de herramientas propietarias (es decir, que no sean de código abierto), son un modo de hacer saber a los programadores de otras aplicaciones cómo incorporar una funcionalidad concreta sin por ello tener que proporcionar información acerca de cómo se realiza internamente el proceso.

Durante el desarrollo de la herramienta software que forma parte de este proyecto se han utilizado una serie de APIs de ImageJ y Java que nos han servido para simplificar nuestro código y facilitar las comunicaciones entre nuestras clases propias de Java y las aplicaciones ImageJ.

A continuación, se indicarán los paquetes utilizados, describiendo para qué sirven y cuáles son las clases que se han utilizado en cada paquete.

3.1 Paquete ij

Este paquete es el más básico y esencial que posee ImageJ. Dentro hay clases que sirven tanto para gestionar las comunicaciones entre la aplicación y el usuario como para dar forma en estructuras de datos específicos a las imágenes que abrimos o creamos.

De la variedad de clases que se pueden encontrar en este paquete sólo hemos utilizado estas cuatro:

- ImageJ: es la clase principal de ImageJ. Esta clase contiene el punto de acceso principal al programa, y la ventana principal de ImageJ.
- ImagePlus: los objetos de esta clase sirven para la representación de una imagen en ImageJ. Esta clase depende de otra que veremos más adelante, ImageProcessor.
- ImageStack: son los objetos conocidos como pilas, que almacenan una matriz extensible de imágenes.
- WindowManager: esta clase gestiona la lista de ventanas abiertas en ImageJ.

3.2 Paquete ij.gui

Este paquete permite crear nuevas ventanas, ventanas emergentes y cuadros de dialogo diferentes a la venta principal y el menú del programa.

Las clases utilizadas de este paquete son:

- ImageCanvas: esta clase se utiliza para crear un lienzo que sirva para mostrar imágenes en una ventana.
- StackWindow: esta clase hereda de ImageWindow y sirve para mostrar una pila de imágenes.

En este proyecto, hemos utilizado estas clases como superclases de unas clases creadas por nosotros mismos y poder así manipular los comportamientos de éstas.

3.3 Paquete ij.plugin

La mayoría de los comandos de menú ImageJ se implementan como plugins, por lo que se pueden encontrar en el paquete de ij.plugin y sus subpaquetes. Además debemos utilizar este paquete cuando queremos implementar un plugin por nosotros mismos.

En este caso, no hemos utilizado las clases que están incluidas en este paquete sino la interfaz PlugIn que sirve para indicar que el plugin que vamos a desarrollar no requiere de una imagen de entrada.

3.4 Paquete ij.process

Este paquete lo hemos utilizado para trabajar con la imágenes desde el nivel más bajo y próximo a la imagen en sí que existe. En concreto lo usamos para utilizar la clase ImageProcessor que sirve de soporte para los objetos ImagePlus.

Capítulo 4.
Algoritmos implementados

Durante el tiempo que hemos estado trabajando en este TFG hemos realizado muchas tareas de estudio previas a la realización de trabajos más relacionados con la implementación y creación de los algoritmos que más adelante describiremos.

4.1 Tareas previas

Unas de las tareas previas que más nos ayudó fue el desarrollado de un plugin que nos permitiera abrir una secuencia de imágenes en formato DICOM. El objetivo era familiarizarnos con ImageJ y la creación de nuevos plugins.

Puesto que éramos bastante noveles en esto de la creación de plugins para ImageJ optamos por tomar como referencia la clase ImportDICOM ya implementada e incluida dentro de ImageJ. Esta clase contenía demasiadas sentencias que fueron eliminadas para simplificar la apertura de una secuencia de imágenes en formato DICOM. Esta tarea de simplificación añadida a la interpretación del código nos ayudó a acercarnos más a los tipos y estructuras de datos propios de ImageJ y con los que íbamos a trabajar más adelante.

Finalmente y una vez aprendida la implementación y funcionamiento de plugins en Imagej decicimos descartar el plugin desarrollado y utilizar la opción ya implícita en ImageJ que nos permitía abrir una secuencia de imágenes.

A continuación, pasaremos a describir una serie de métodos secundarios, que no por ello menos importantes, puesto que sirvieron para luego poder implementar los algoritmos principales de este trabajo de los que hablaremos en próximos puntos.

4.2 Métodos secundarios

Como comentamos en el punto 2.3.1 el plugin para ImageJ que hemos creado debe tener un método *run*. En este método hemos creado e instanciado todas las estructuras de datos que vamos a necesitar en los distintos algoritmos de búsqueda de áreas neuronales interconectadas.

Las variables más destacables que se inicializan en este método *run* son:

- Dim_secuencia: almacena el tamaño de la secuencia con la que estamos trabajando. Dicho valor se extrae directamente de la imagen mediante métodos propios de la API de ImageJ.
- Dim_serie: almacena el número de imágenes que forman una serie. Este valor se calcula mediante un método (contarSerie) del que hablaremos más adelante.
- Width: en esta variable guardamos el ancho que extraemos de la imagen mediante un método propio de los objetos tipo ImagePlus.
- Height: en esta otra almacenamos el alto mediante un método similar.

Además almacenamos una serie de arrays que son de vital importancia para el funcionamiento del resto del plugin:

- Matrix: es una matriz *short* de cuatro dimensiones (la primera igual al ancho de la imagen, la segunda corresponde al alto de la imagen, la tercera es igual al número de imágenes que hay por serie, y la cuarta y última igual al número de series que tiene la secuencia con la que vayamos a trabajar) que nos sirve para almacenar la información de los pixeles de cada imagen.
- valoresAngulos: es una matriz *double* de dos dimensiones (cantidad de series de la secuencia por tres) que nos sirve para almacenar los cosenos de las componentes (x, y, z) que extraemos de las etiquetas de la imagen.
- Valoresgrados: es una matriz *double* de dos dimensiones (cantidad de series de la secuencia por dos (grados theta y phi)) que nos sirve para almacenar los grados que calculamos a través de operaciones trigonométricas y los valores de la matriz anterior.
- SuperMatrix: es una matriz *int* de tres dimensiones (la primera igual al ancho de la imagen, la segunda corresponde al alto de la imagen y la tercera es igual al número de imágenes que hay por serie, es decir, la

altura de la imagen tridimensional) que nos sirve para almacenar el número de veces que se visita cada pixel dentro del grupo de imágenes que abarca la serie en la que nos encontremos trabajando.

- SuperMatrixRuta: es una matriz *int* de cuatro dimensiones (la primera igual al ancho de la imagen, la segunda corresponde al alto de la imagen, la tercera es igual al número de imágenes que hay por serie y la cuarta es igual a tres (x,y,z)) que nos sirve para almacenar el punto (x',y',z') al que saltaremos habiendo calculado ese salto desde un punto (x,y,z) por el que ha pasado un algoritmo anteriormente y poder así agilizar el cálculo de futuros caminos que pasen por ese mismo punto.
- VectorMayor: es un *ArrayList* (vector dinámico en Java) de tipo DatosSerie (es una clase que nos hemos creado para poder almacenar en una misma casilla del ArrayList dos valores *short*) que nos sirve para almacenar los valores que tiene un mismo pixel (x,y,z) en todas las series, guardando para cada uno su valor y la serie.
- VectorVoxel: es un vector *short* de tamaño igual al número de series que conforman la secuencia.
- Variances: se ha calculado una matriz tridimensional que mide la variabilidad de la difusión de las moléculas de agua en las diferentes direcciones medidas. Se ha calculado la varianza de los distintos valores para cada voxel, que es proporcional a la anisotropía en dicho elemento de la imagen.

Una vez tenemos las variables mencionadas inicializadas y los arrays instanciados pasamos a rellenarlos haciendo uso del código escrito en *run* y de otros métodos.

Para rellenar la matriz valoresAngulos recorremos de principio a fin toda la secuencia pero avanzaremos 27 imágenes en cada salto para así solo visitar la primera imagen de cada serie y extraer de ella los cosenos de cada componente de dirección del vector.

Estando ya posicionado en la primera imagen de cada serie llamamos al método de apoyo, obtenerAngulos, con el que rellenamos la matriz valoresAngulos. El código de este método es muy fácil de extraer de los metadatos de la imagen ya que la información de los cosenos de cada componente está colocada en etiquetas diferentes, simplemente tenemos que

obtener la información de la imagen y almacenarla en la matriz valoresAngulos en el lugar de la serie con que estemos trabajando.

Ahora que ya tenemos los cosenos almacenados para cada serie y puesto que no vamos a trabajar con cosenos sino con grados debemos convertirlos. Para realizar esta conversión implementamos un método llamado calcularGrados que aplica formulas trigonométricas sobre los cosenos y obtiene los grados llamados *theta* y *phi* que nos servirán para más adelante decidir cuál es el movimiento que realizamos.

Por último, encontramos la llamada al método construirMatrix. Este método se encarga de modelar la imagen en una estructura de datos típica de Java. Lo que hacemos es almacenar la capacidad de difusión de cada voxel de la imagen en la matriz de cuatro dimensiones llamada matrix, para cada uno de los ángulos de observación. En este método, y al contrario de lo que hicimos antes, debemos recorrer todas las imágenes de la secuencia e ir tomando el valor de cada pixel aunque deberemos tener en cuenta que el orden de los índices para acceder a la imagen es (columna, fila) pero que para la matriz será (fila, columna) puesto que supone un ahorro en el tiempo de acceso a memoria.

Por ejemplo, fijándonos en la figura 1.3., si queremos almacenar el valor del pixel marcado en la tabla "1" debemos acceder a él mediante (2,3) si lo hacemos en la imagen pero mediante (3,2) si accedemos a la matriz, véase la tabla "2".

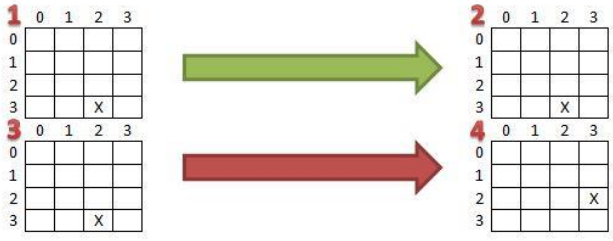


Figura 4.1. Esquema explicativo de acceso a la imagen y la matriz.

En cambio, si accediésemos tanto a la imagen como a la matriz usando el mismo orden en los índices obtendríamos algo distinto y que no se adapta a la realidad, véase las tablas "3" y "4" de la imagen de laFigura 4.1.

Aclarada la forma en la que accedemos a la imagen y a la matriz, debemos especificar que los valores que se leen de la imagen son convertidos a los

valores medidos mediante una recta de calibración proporcionada en los metadatos de la imagen.

Todos los métodos descritos anteriormente forman las bases para poder lanzar los algoritmos de búsqueda de tractos neuronales que conforman este trabajo y que definiremos en las siguientes secciones.

4.3 Búsqueda saltando al mayor pixel partiendo de uno dado

El objetivo de este algoritmo es que tomando un pixel inicial de la imagen (se obtiene haciendo clic sobre un punto de la imagen) se pueda calcular el camino que siguen las fibras neuronales fijándonos en la capacidad de difusión de cada pixel y poder así dibujar un tracto en la imagen.

Para poder llevar a cabo este algoritmo necesitamos la matriz de 4 dimensiones, "matrix", en donde hemos guardado la información de cada uno de los pixeles de la imagen y la matriz donde tenemos la información de los grados para cada serie, "Valoresgrados".

Una vez tenemos el punto seleccionado (x,y,z) desde donde queremos iniciar a calcular un tracto, se debe comprobar si el pixel tiene una capacidad de difusión mayor a un umbral que se ha fijado para determinar si estamos en un punto del cerebro o fuera de él. Una vez hecha la comprobación, el algoritmo entra en un bucle en el que se harán una serie de tareas repetitivas para ir saltando de un pixel a otro hasta que alcancemos un pixel con valor menor al umbral establecido, un pixel fuera de la materia blanca (que presenta una anisotropía, o varianza menor que otro umbral establecido) o se hayan producido más de 200 saltos. Los umbrales, tanto para los valores de la difusividad, como para la anisotropía, se han establecido a partir de los histogramas de las imágenes.

Los dos primeros pasos que se realizan dentro del bucle son pintar en negro el pixel actual y marcarlo como visitado usando una matriz tridimensional booleana.

El siguiente paso que debe hacer el algoritmo es decidir a qué punto salta, para ello comprueba la capacidad de difusión de ese mismo punto (x, y, z) en las demás series y tomará la serie donde esa difusión sea mayor.

Para determinar el pixel de mayor valor de difusión utilizamos el ArrayList "VectorMayor", en él se van introduciendo cada uno de los valores de difusión de ese pixel para las distintas series junto con el número de la serie. Luego, el vector es ordenado mediante el método de ordenación Quicksort. Estas dos tareas de crear el vector y ordenarlo se hacen en dos métodos de apoyo.

De vuelta al método principal, el siguiente paso es extraer el último elemento del vector, puesto que está ordenado ascendentemente. Sólo haremos uso del número de la serie y no podrá ser la serie 0, puesto que, los ángulos de esta serie son igual a cero, es decir, no nos moveríamos. Ahora se mira qué pasa con cada una de las coordenadas del pixel actual. Para estas comprobaciones se hace uso nuevamente de tres métodos de apoyo.

Básicamente, lo que se realiza dentro de estos métodos es mirar los valores de los ángulos *theta* y *phi* para la serie extraída y ver cómo se modifica cada componente, las posibilidades son que se quede igual, que se le sume uno o que se le reste uno.

- Para la componente X, deberemos fijarnos en los valores del ángulo *phi* para la serie seleccinada:
  - Si el valor de *phi* es menor igual que -112.5 o mayor igual que 112.5, entonces se le resta uno.
  - Si el valor de *phi* está entre -67.5 y 67.5, entonces se le suma uno.
  - Para otros valores de *phi*, el valor permanece intacto.
- Para la componente Y deberemos fijarnos de nuevo en el ángulo *phi*:
  - Si el valor de *phi* está entre -22.5 y -157.5, entonces se le resta uno.
  - Si el valor de *phi* está entre 22.5 y 157.5, entonces se le suma uno.
  - Para otros valores de *phi*, el valor permanece intacto.
- Para la componente Z deberemos fijarnos en los valores del ángulo *theta*:
  - Si el valor de *theta* es mayor igual que 135, entonces se le resta uno.
  - Si el valor de *theta* está entre 0 y 45, entonces se le suma uno.
  - Para otros valores de *theta*, el valor permanece intacto.

Cuando ya se sabe qué pasa con cada componente, se comprueba si ese salto es posible o no. Existen tres casos en los que el salto se podría considerar no posible:

- Cuando suponga saltar a un pixel de la imagen ya visitado, en este caso y siempre y cuando queden aún elementos en el "VectorMayor", se extrae la siguiente serie mayor y se repite todo el proceso de comprobación de las tres componentes para obtener un nuevo posible punto al que saltar.
- Cuando signifique realizar un cambio de dirección de más de 35º. Si no tuviéramos esto en cuenta los caminos que se generarían serían un poco gruesos ya que no siguen una trayectoria suave sino que se retuercen, debido a un cambio brusco de dirección. Eso es probablemente debido al ruido de medida de la difusividad en los diferentes ángulos y hace que el camino seguido tenga una componente de aleatoriedad. Se extrae el siguiente valor del vector ordenado, como en el caso anterior.
- Cuando provoque salirse de los límites de la imagen, en este caso se para la búsqueda del tracto.

Cuando el salto sí es bueno, se convierte en definitivo y si las condiciones del bucle lo permiten, se empieza el proceso otra vez para ese nuevo punto.

El resultado final de este algoritmo se ve de dos formas, una es, como ya hemos dicho, poner a negro sobre las imágenes originales aquellos pixeles por los que vamos pasando y la otra es usar una matriz contador marcando todos aquellos pixeles que se van visitando, al final esa matriz tridimensional se vuelca en una nueva secuencia de imágenes estando todo en negro menos los puntos visitados que serán de un color más claro. Esta segunda forma, de la que hablaremos más adelante, permite representar mucho mejor el resultado de este algoritmo en 3D.

4.4 Búsqueda saltando al mayor pixel partiendo de todos

En este segundo algoritmo tomamos como base la filosofía del anterior pero ahora no se trazará un único tracto partiendo de un pixel seleccionado en la imagen sino que partiendo de cada uno de los puntos de la imagen se trazará uno.

El punto de partida de este método como ya hemos dicho no va a ser un punto que seleccionemos nosotros así que hemos optado por crear un método

de apoyo, llamado creaSuperMatrix, que recorra la secuencia centrándose en las primeras 27 imágenes que corresponden a la primera serie. Para cada pixel (x,y,z) se efectuará una llamada al algoritmo principal.

El funcionamiento del algoritmo principal que nos ocupa en este punto toma como base el código del descrito en el punto anterior pero añade nuevas tareas para poder obtener el resultado que queremos. El resultado visual de este algoritmo no es dibujar sobre la misma imagen los caminos que se van generando sino que el algoritmo nos devolverá una matriz contador tridimensional, llamada "SuperMatrix", que almacena en su interior el número de veces que se visita cada pixel.

Puesto que en este segundo algoritmo el número de caminos que se van a generar es muchísimo mayor que en el primero, se ha optado por tener una matriz de cuatro dimensiones de apoyo, llamada "SuperMatrixRuta", que nos ayude a ir guardando para cada pixel a donde se salta y tener un tiempo de cómputo y de acceso a memoria más bajo. Entonces, estando en un determinado pixel habrá que diferenciar entre si nunca hemos pasado por él o sí. Para pixeles nunca antes visitados se deberán llevar a cabo las tareas de búsqueda descritas en el punto anterior, en cambio, en el caso de que se haya visitado alguna vez, se usará la información almacenada en "SuperMatrixRuta" para realizar el salto.

Dada las novedades introducidas en este algoritmo con respecto al primero se deben tener tres cosas en cuenta:

- La matriz de pixeles visitados se sigue utilizando en este algoritmo pero se debe "resetear" al comienzo de la generación de cada nuevo tracto.
- La matriz "SuperMatrix" no se debe "resetear" nunca, sus casillas irán acumulando el número de veces que se pasa por cada pixel durante la ejecución completa del algoritmo.
- La matriz "SuperMatrixRuta" tampoco se debe "resetear" puesto que cuanta más información contenga más ahorro obtendremos.

El último paso será volcar la matriz "SuperMatrix" en una nueva secuencia de imágenes en la que veremos en negro los pixeles nunca visitados y en un tono grisáceo o blanco a los pixeles que en menor o mayor medida han sido visitados. Estos últimos, representarán los tractos.

4.5 Búsqueda saltando mediante probabilidad

Para este algoritmo se eligió utilizar, el método Monte Carlo, una filosofía totalmente diferente a la de los anteriores.

El método Monte Carlo es un método estadístico numérico que permite resolver problemas físicos y matemáticos mediante la simulación de variables aleatorias. Fue bautizado así por su clara analogía con los juegos de ruleta de los casinos, el más célebre de los cuales es el de Monte Carlo.

La importancia actual de este método se basa en la existencia de problemas que tienen difícil solución por métodos exclusivamente analíticos o numéricos, pero que dependen de factores aleatorios o se pueden asociar a un modelo probabilístico artificial. Debido al avance en el diseño de los ordenadores, cálculos de este tipo que en otros tiempos hubieran sido inalcanzables, hoy en día se presentan como asequibles para la resolución de ciertos problemas. La base es la generación de números aleatorios de los que nos serviremos para calcular probabilidades. Para la generación de números aleatorios hemos hecho uso de la función *random* incluida en la clase Math que la máquina virtual Java trae por defecto como generador. Las pruebas realizadas verifican su calidad a la hora de calcular números aleatorios sin tendencia aparente a la repetición ordenada.

Como ya hicimos en el primer algoritmo, para disparar el cálculo de caminos deberemos hacer clic en el pixel desde el que empezarán todos los caminos y comprobar que la capacidad de difusión es mayor que el umbral establecido. En caso afirmativo, se empezarán a generar los caminos partiendo siempre del mismo punto. El número de caminos a generar ha de ser bastante grande para obtener con más claridad cuáles son los caminos favoritos.

Al igual que en los dos algoritmos anteriores usaremos matrices, vectores y métodos de apoyo. Lo primero que se debe hacer cuando empezamos a calcular un camino es usar las matrices "visitados" y "SuperMatrix", la primera matriz para marcar qué pixeles están visitados y cuáles no y la segunda para contabilizar el número de veces que se pasa por un pixel. A continuación, se pasa a obtener el conjunto estadístico que usaremos, que estará formado por todos los valores del pixel en el resto de series de la secuencia. Para esto, hacemos uso de un método de apoyo que, por un lado, nos genera un ArrayList que contiene los 34 valores posibles ordenados por

serie, y por otro, nos devuelve la suma de todos los valores. Usando este valor, llega la hora de generar un número aleatorio que este entre uno y dicho número, luego, se debe determinar en qué serie está contenido ese valor. Una vez tengamos la serie pasamos a realizar el salto mediante los métodos que ya usamos en los algoritmos anteriores, que comprobaban qué pasaba con cada componente (x, y, z) del pixel actual. Dependiendo de la respuesta conjunta que dan estos tres métodos podemos actuar de tres formas:

- Si el salto se produce a un pixel ya visitado durante la generación del camino actual, se repetirá el proceso de generar un número y ver en qué serie está contenido para luego preguntar qué pasa con cada componente un máximo de 2000 veces.
- Si el salto se produce a un pixel fuera de la imagen se para la generación de este camino y se inicia uno nuevo.
- Si, por el contrario, el salto ha sido a un pixel "bueno", se toma el valor de ese pixel y si tanto su valor como la varianza son mayores a los umbrales establecidos se marca como visitado, se contabiliza que se ha pasado por él y se empieza a calcular un nuevo salto y se estará saltando hasta un máximo de 2000 saltos.

Una vez se termine el trazado de un camino se deberá volver a iniciar otro, partiendo del pixel inicial en el que hicimos clic y reseteando la matriz de "visitados".

Cuando se hayan calculado el número de caminos deseado se pasará a volcar la matriz "SuperMatrix" en una nueva secuencia de imágenes en la que veremos en negro los pixeles nunca visitados y en un tono grisáceo o blanco a los pixeles que en menor o mayor medida han sido visitados. Los más blancos serán los favoritos.

4.6 Dificultades encontradas

Durante la implementación de los algoritmos que hemos comentado en las secciones anteriores, nos hemos encontrado con una serie de dificultades típicas a la hora de trabajar con estructuras de datos matriciales, un alto número de accesos a memoria y un tema tan desconocido como creación de mapas de interconexiones neuronales.

Una vez metidos en materia y sabiendo ya qué estructuras de datos íbamos a utilizar para modelar nuestras imágenes, nos encontramos con el primero de nuestros problemas, la diferencia que se aprecia dependiendo del orden de los índices a la hora de acceder a los pixeles de una imagen o a las casillas de la matriz que almacena los valores de éstos. Lo que comentamos ya en la sección 4.2 tuvimos que seguir teniéndolo presente cada vez que en un mismo algoritmo se producían accesos a la imagen y a la matriz. Un ejemplo claro lo podemos ver en el primer algoritmo cuando debemos marcar a negro cada uno de los pixeles que visitamos y a la vez obtener de la matriz el valor de ese pixel.

La siguiente dificultad con la que nos encontramos no estuvo tan relacionada con la implementación del código sino con la ambigüedad de la documentación que se encontraba acerca de los valores de los cosenos de las componentes (x,y,z) que se obtienen de la imagen y si se debían transformar o no dependiendo de la orientación con la que están dispuestas las imágenes en la secuencia, es decir, si el orden de las imágenes es de pies a cabeza o viceversa. Finalmente, tras consultar la documentación del fabricante del aparato con el que se tomó la prueba, observar bien las imágenes y ver que verdaderamente las imágenes van desde la zona más baja del cerebro a la más alta y realizar diferentes pruebas para a ver cómo variaba el resultado final dependiendo de si hacíamos o no las transformaciones indicadas se decidió que sólo se debía transformar el coseno de la componente z, multiplicándolo por "-1".

Durante el testeo de los algoritmos nos dimos cuenta que en ciertos puntos en los que se recorrían matrices estábamos dejando atrás la buena práctica de empezar a recorrerlas desde la primera dimensión hasta la última y que algunas matrices *double* podían ser convertidas a *short*. Ambas cosas provocaban un retraso considerable en la finalización de los algoritmos, debido al gran número de accesos a memoria y el tamaño de las matrices. Usábamos, en realidad, todo lo contrario puesto que era más cómodo, por ejemplo, a la hora de extraer una imagen de la pila de imágenes y volcarla en la matriz. Cogíamos todos los pixeles de una imagen y luego los de otra y así hasta tenerlas todas pero era más rápido extraer el mismo pixel de todas las imágenes y luego cambiar a otro y así hasta tenerlos todos.

Unas de las últimas dificultades destacable que solucionamos y que supuso un gran avance, en el tema de reducir el tiempo de ejecución, sobre todo, del segundo algoritmo, fue reducir la matriz "nvisitados" de cuatro dimensiones a tres y pasarla como parámetro de entrada hacia dicho algoritmo. La reducción del tamaño de la matriz vino dada por no existir la necesidad de tener una matriz tan grande para representar los pixeles que estaban visitados durante la búsqueda del camino, puesto que, si iniciamos la ejecución desde un pixel de una serie determinada, no puede darse el caso de que saltemos a un pixel de otra serie. Por lo tanto, solo podemos saltar entre los 65536 (las imágenes son de 256x256) pixeles de cada una de las 27 imágenes que conforman la serie a la que pertenece el pixel desde el que inicias el algoritmo.

El otro cambio que hicimos con respecto a la utilización de la matriz "nvisitados", estuvo relacionado con la manera que teníamos de instanciarla y "resetearla". Inicialmente la matriz la teníamos colocada al inicio del algoritmo como una estructura más que se instanciaba al entrar en él, consiguiendo así, que para cada llamada la matriz se "reseteará". Sin embargo, creándola en el método de apoyo, creaSuperMatrix y pasándola como parámetro de entrada hacia el segundo algoritmo, y además, creando un método auxiliar para "resetearla" (poner todas sus casillas a false) se logró reducir, nuevamente, el tiempo de ejecución de este algoritmo, puesto que, cuesta más borrar e instanciar constantemente una matriz que pasarla como parámetro y "resetearla manualmente".

Capítulo 5.
Resultados

De los diferentes algoritmos descritos en el capítulo anterior, hemos obtenidos diversos resultados que son propensos a interpretación, sin embargo, creemos que los algoritmos que mejores resultados ha dado son el primero y el tercero. Durante este capítulo vamos a intentar ver y explicar de una manera gráfica cuales han sido esos resultados.

Antes de empezar a comentar esos resultados, nos gustaría presentar unas imágenes que también aportan sentido a todo lo que hemos implementado. Aprovechando el cálculo de la varianza de los distintos valores para cada voxel, que teníamos almacenada en una matriz de tres dimensiones (256(ancho de la imagen)x256(alto de la imagen)x27(número de capas)) creamos una nueva secuencia de imágenes, donde se vieran representados los valores de dicha matriz de varianzas. En las siguientes figuras podemos ver algunas capturas de esa secuencia creada, se ha jugado con la orientación para que se pudieran apreciar los resultados desde distintos ángulos.

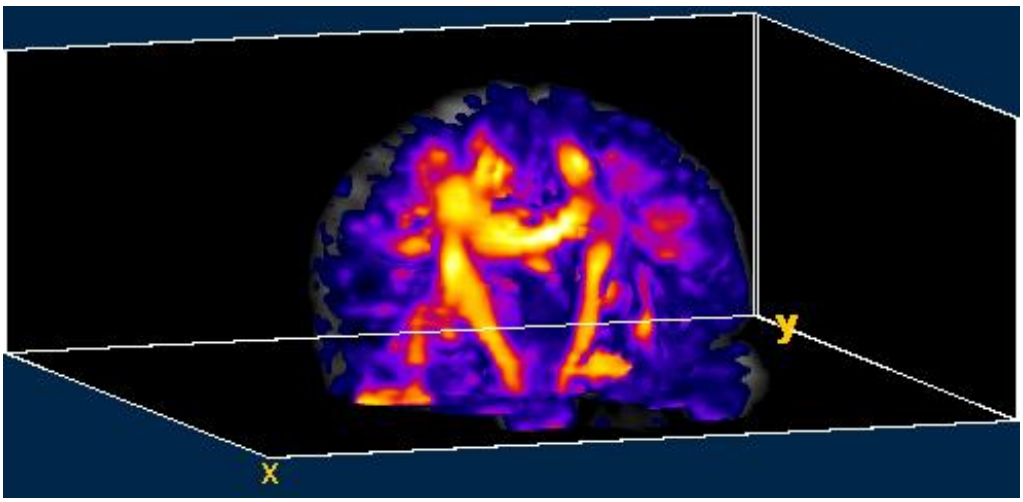


Figura 5.1. Primer ejemplo imagen tridimensional de varianzas.

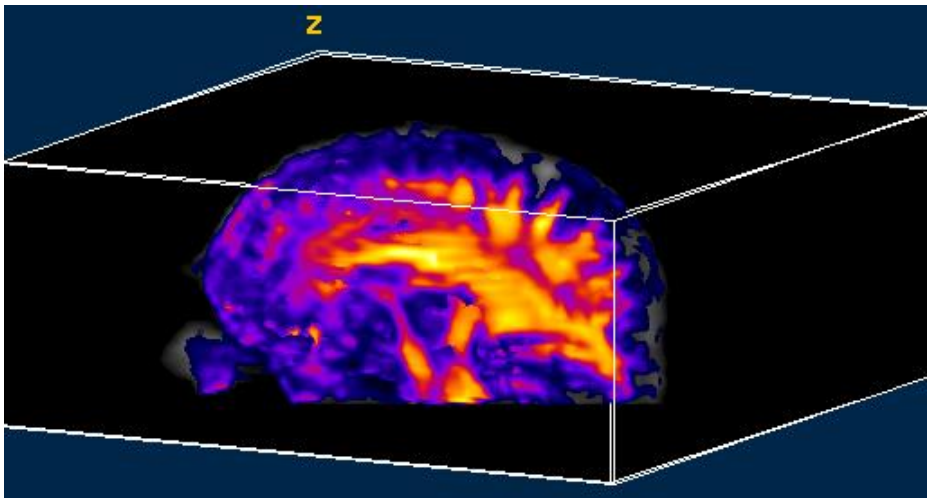


Figura 5.2. Ejemplo 2.

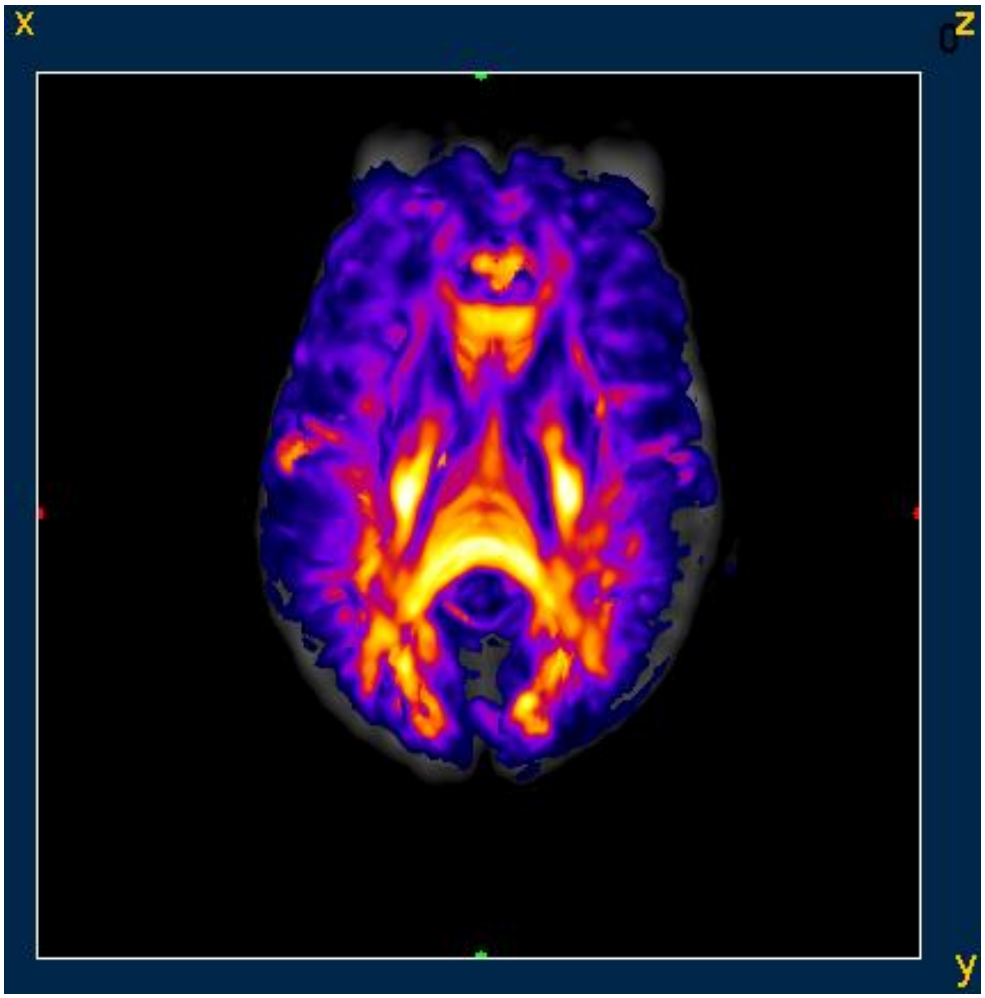


Figura 5.3. Ejemplo 3.

La secuencia fue creada con imágenes en escala de calor (colores cálidos), donde las zonas más amarillas son aquellas en las que existe mayor anisotropía. Esta nueva secuencia, ayuda mucho más que la original a detectar cuales son las zonas donde existe mayor anisotropía.

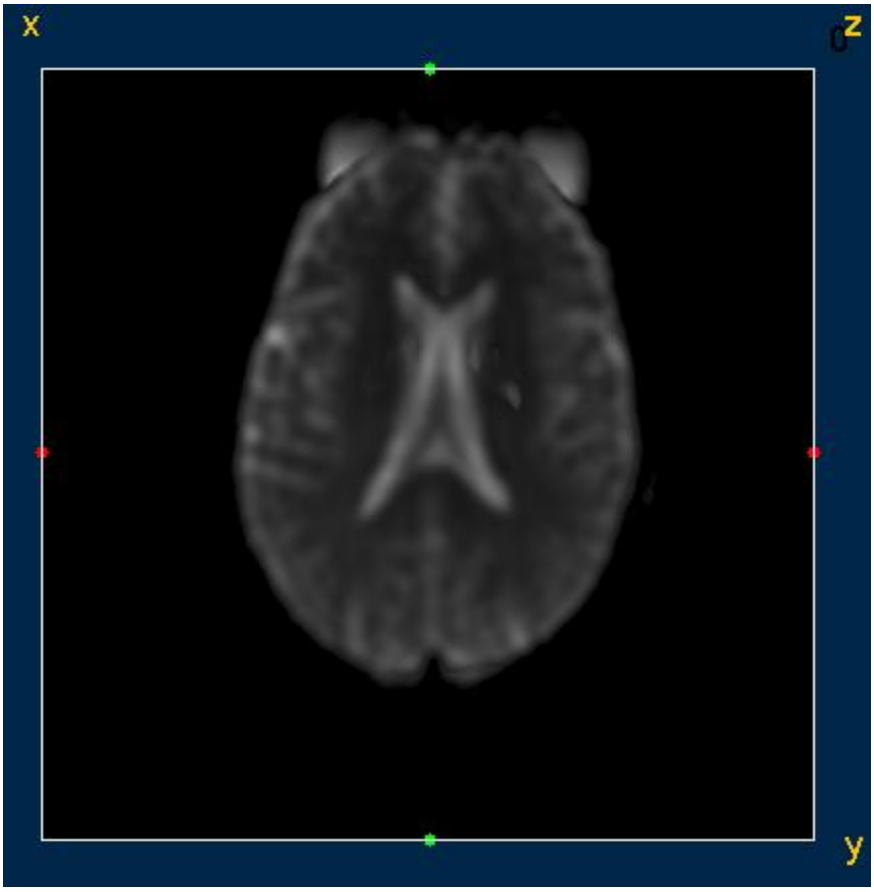


Figura 5.4. Captura de la secuencia original.

Para todas estas imágenes y las que se seguirán presentando a lo largo del capítulo se ha utilizado una funcionalidad propia de ImageJ que permite transformar las secuencias resultantes de cada algoritmo en cubos 3D.

5.1 Resultados del método "Busqueda saltando al mayor pixel"

En este punto nos gustaría hablar sobre los diferentes resultados que hemos obtenido con este algoritmo. En primer lugar vamos a mostrar una serie de imágenes, que muestran los resultados obtenidos sin tener en cuenta una de las condiciones que antes describimos para determinar un salto como no posible. Dicha condición es la que evita que se den cambios bruscos de

dirección en el camino que se va trazando, en las siguientes figuras podemos observar como el camino obtenido a partir de un punto inicial elegido al azar no está del todo definido, es muy grueso..

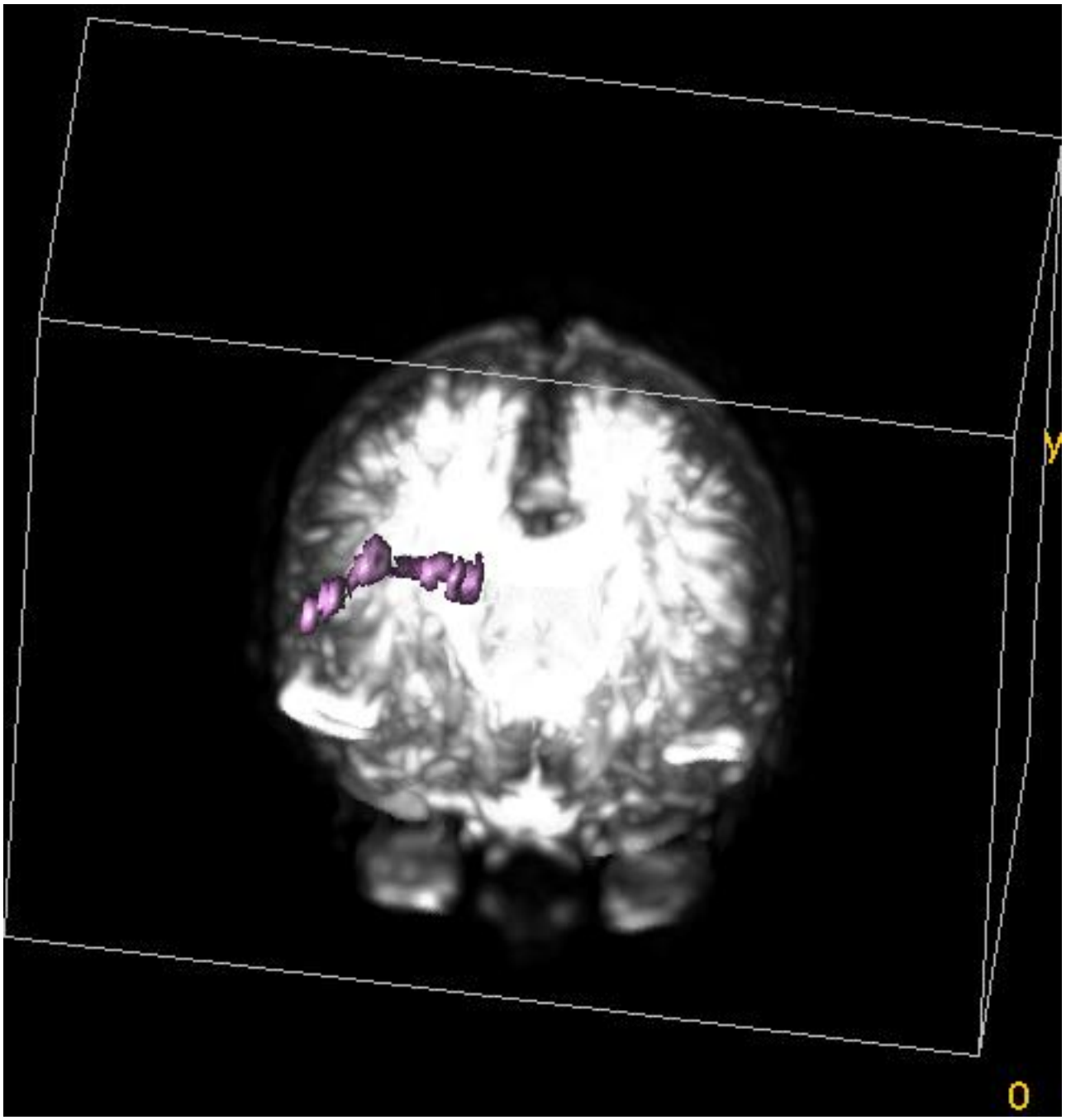


Figura 5.1.1.

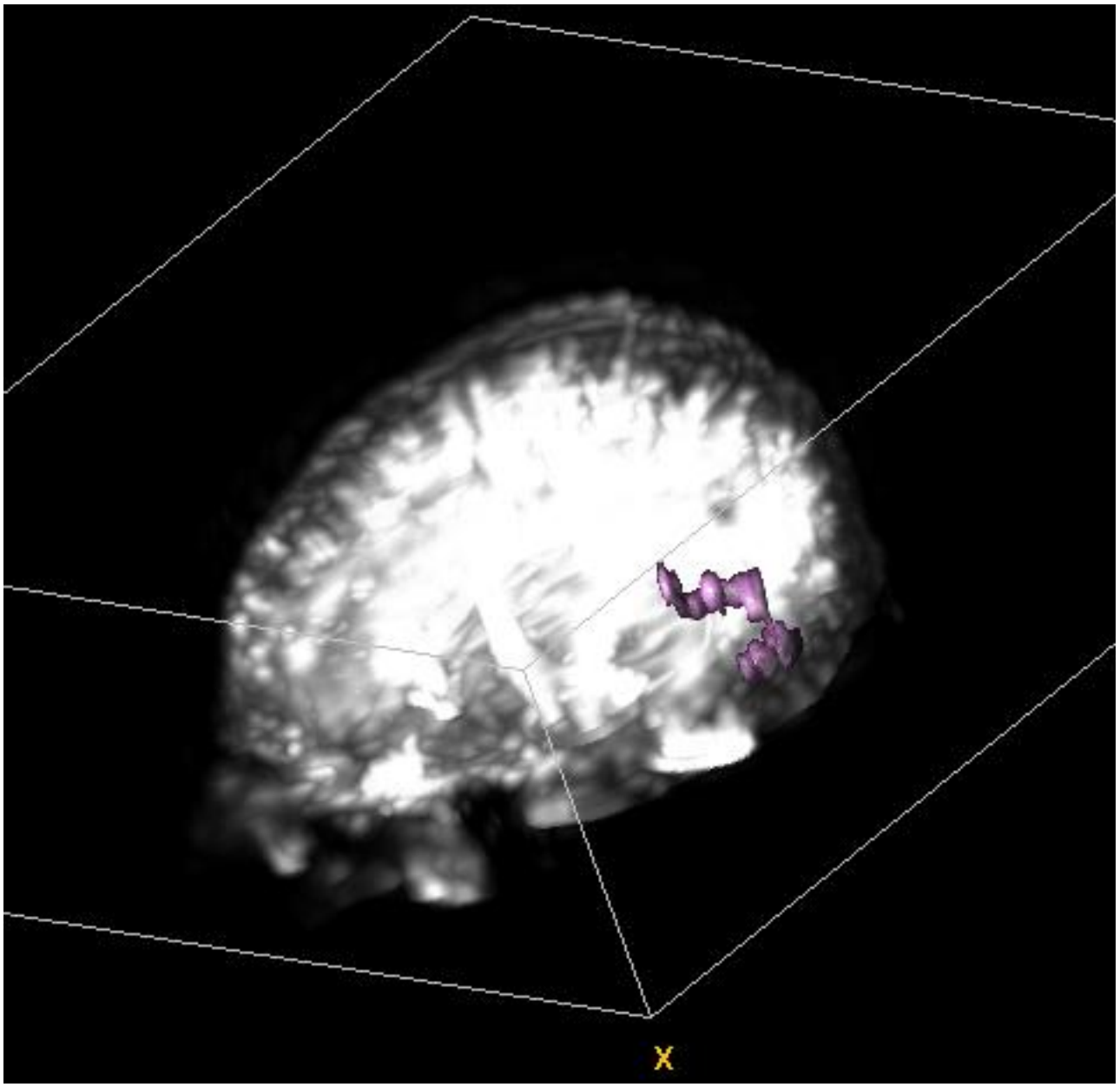


Figura 5.1.2.

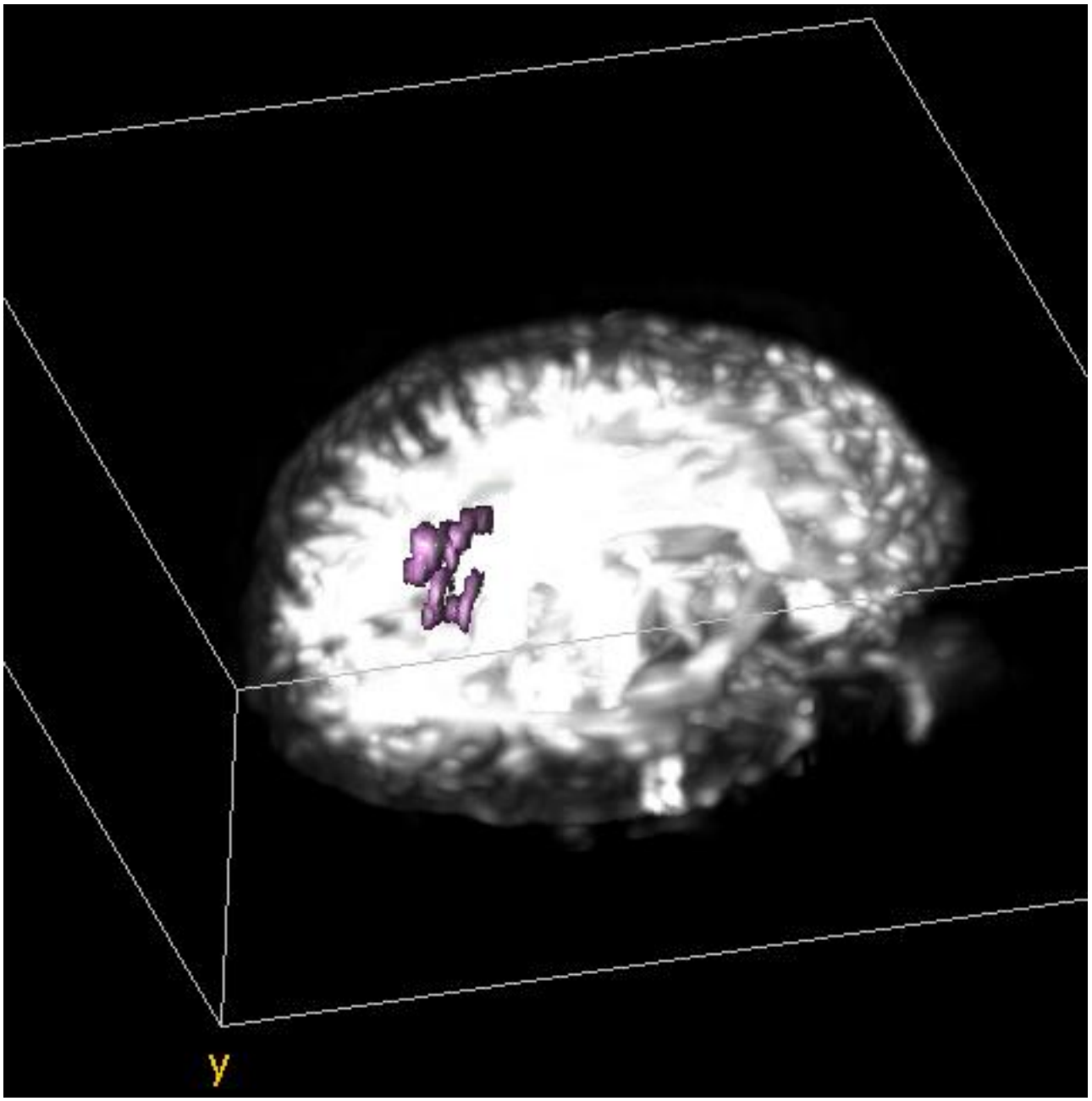


Figura 5.1.3.

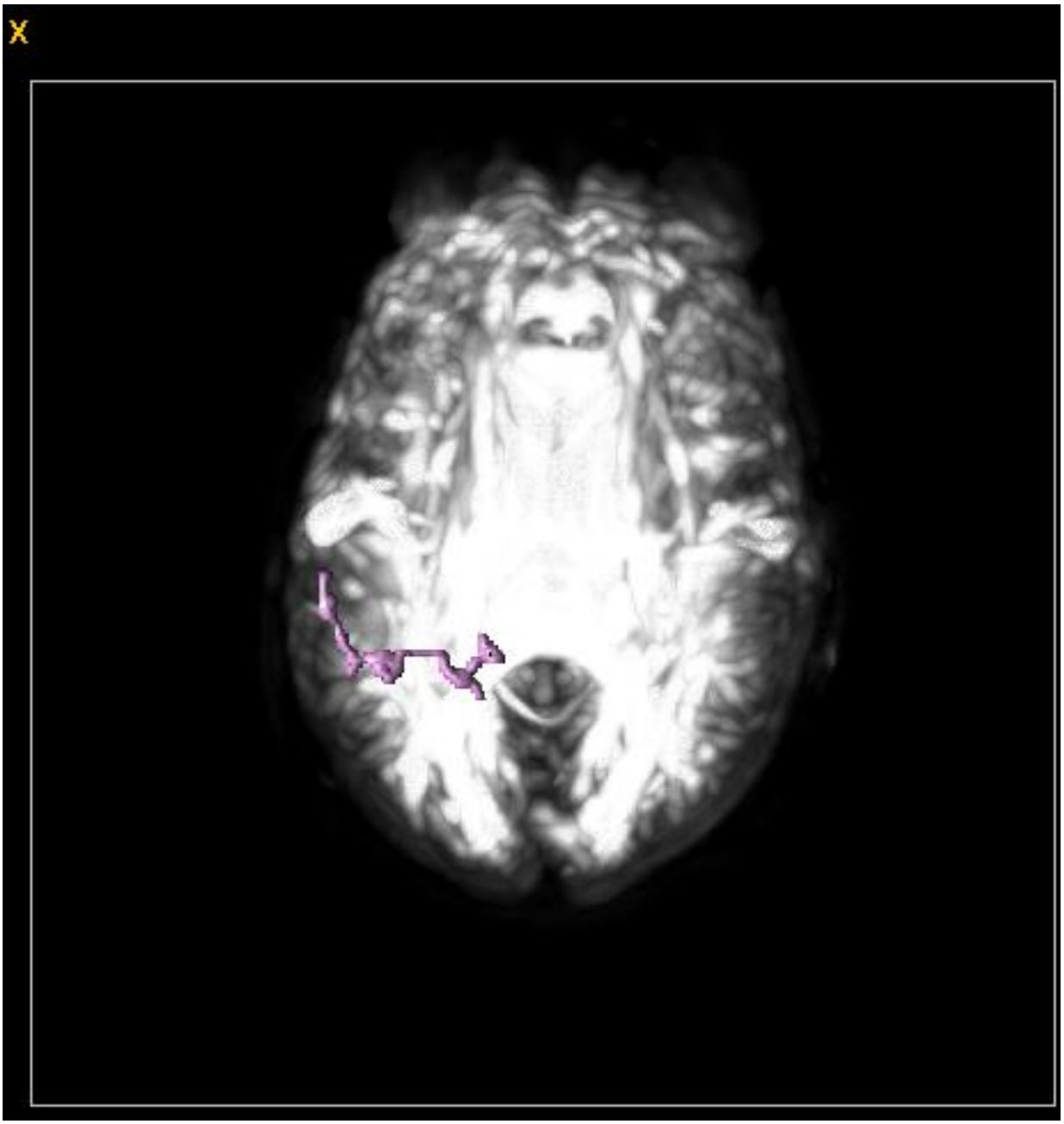


Figura 5.1.4.

Una vez vistos estos resultados, tomamos la decisión de cambiar el código e incluir esa condición para que en los saltos se evitara realizar cambios bruscos de dirección del camino que se iba trazando. En las siguientes figuras podemos observar los resultados tras añadir esa modificación, suavizándose los caminos.

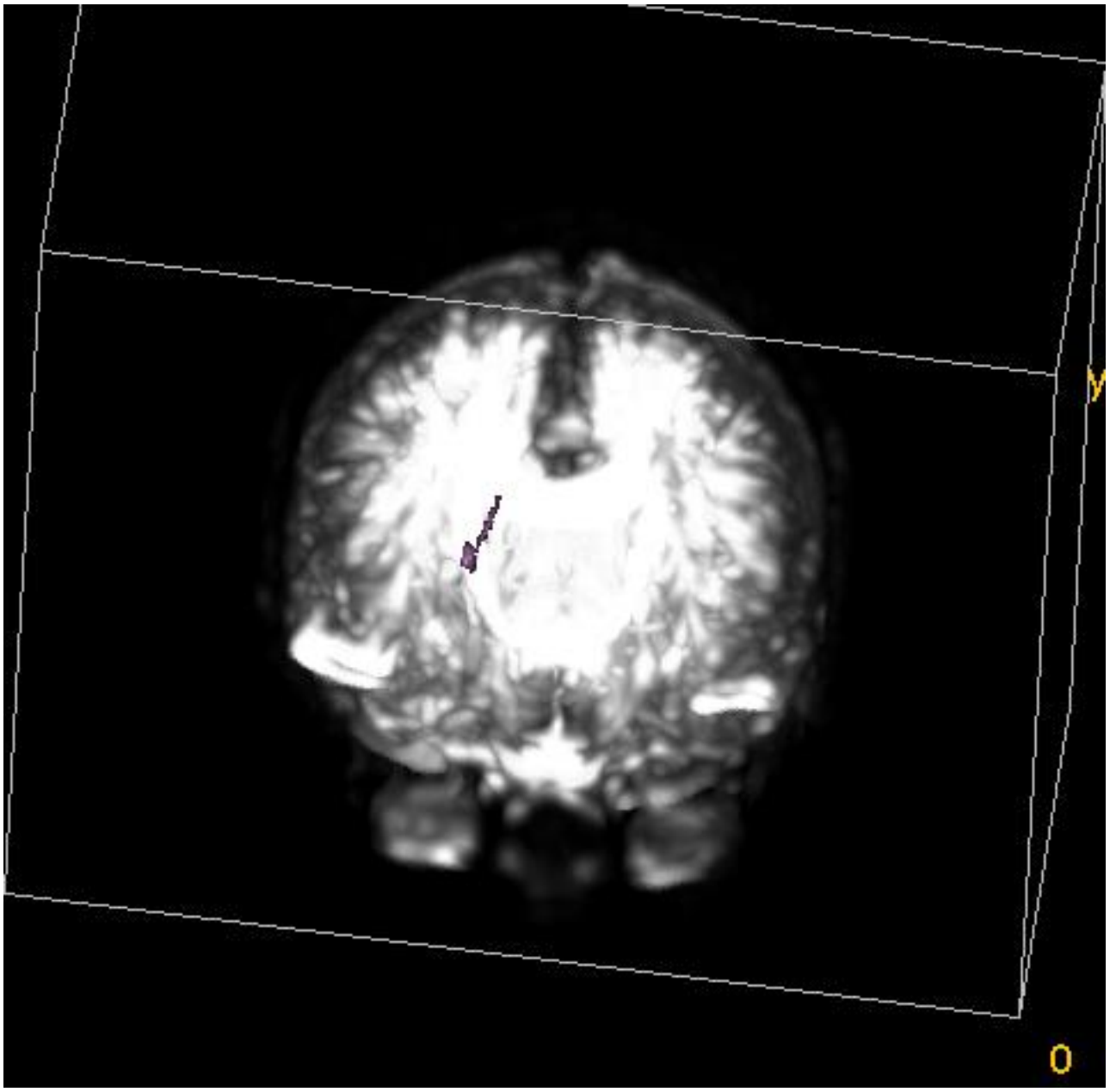


Figura 5.1.5.

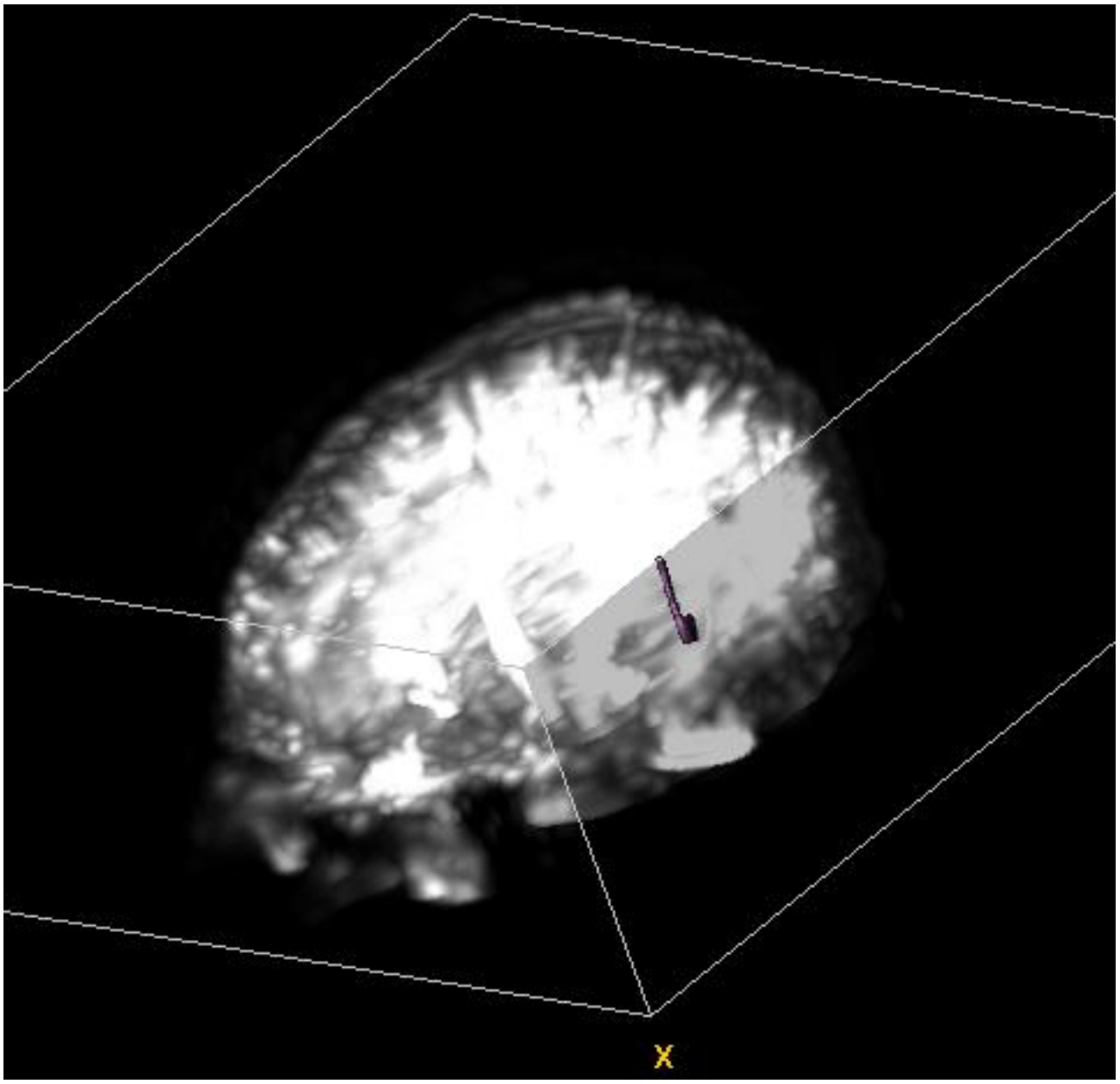


Figura 5.1.6.

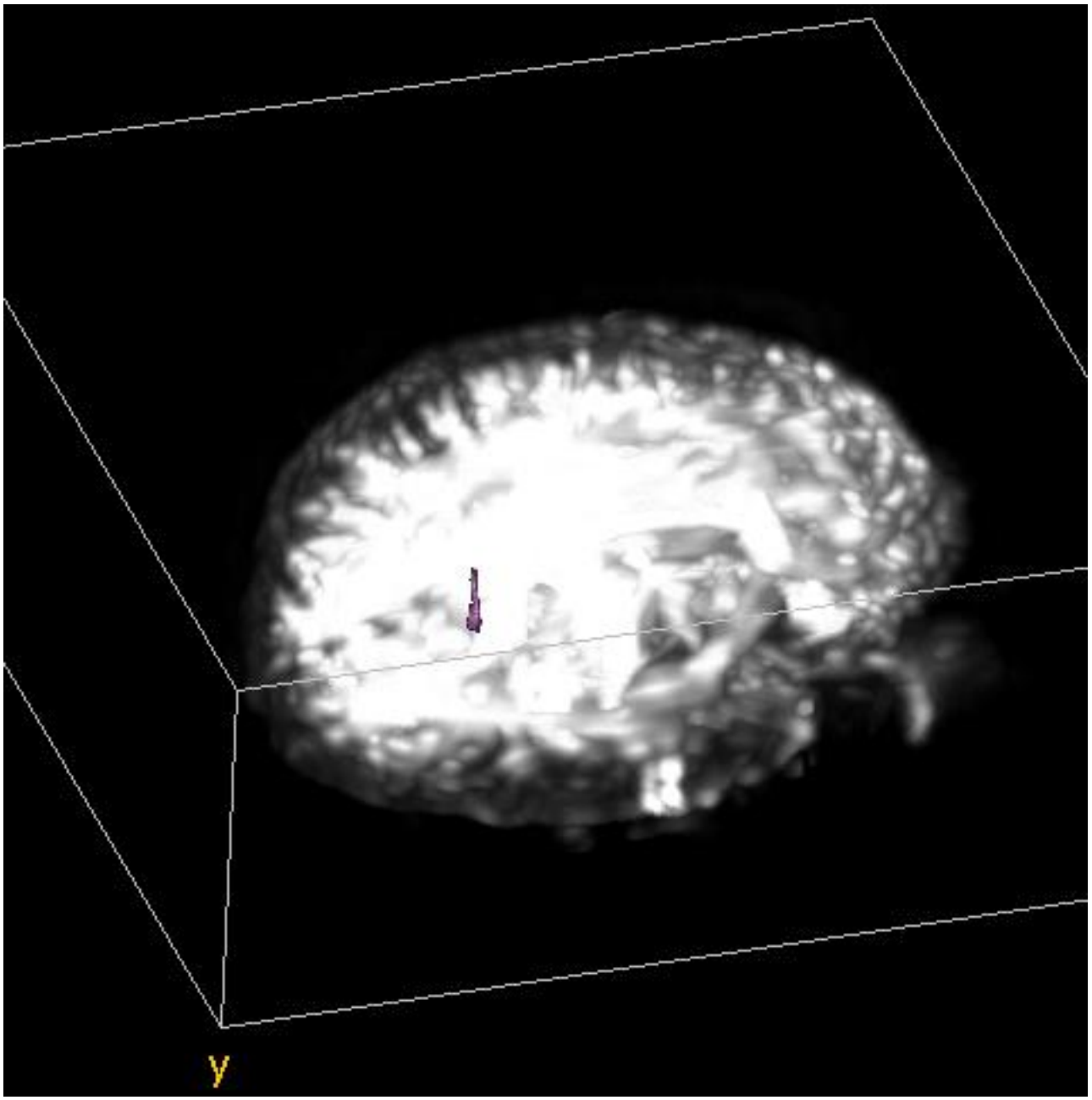


Figura 5.1.7

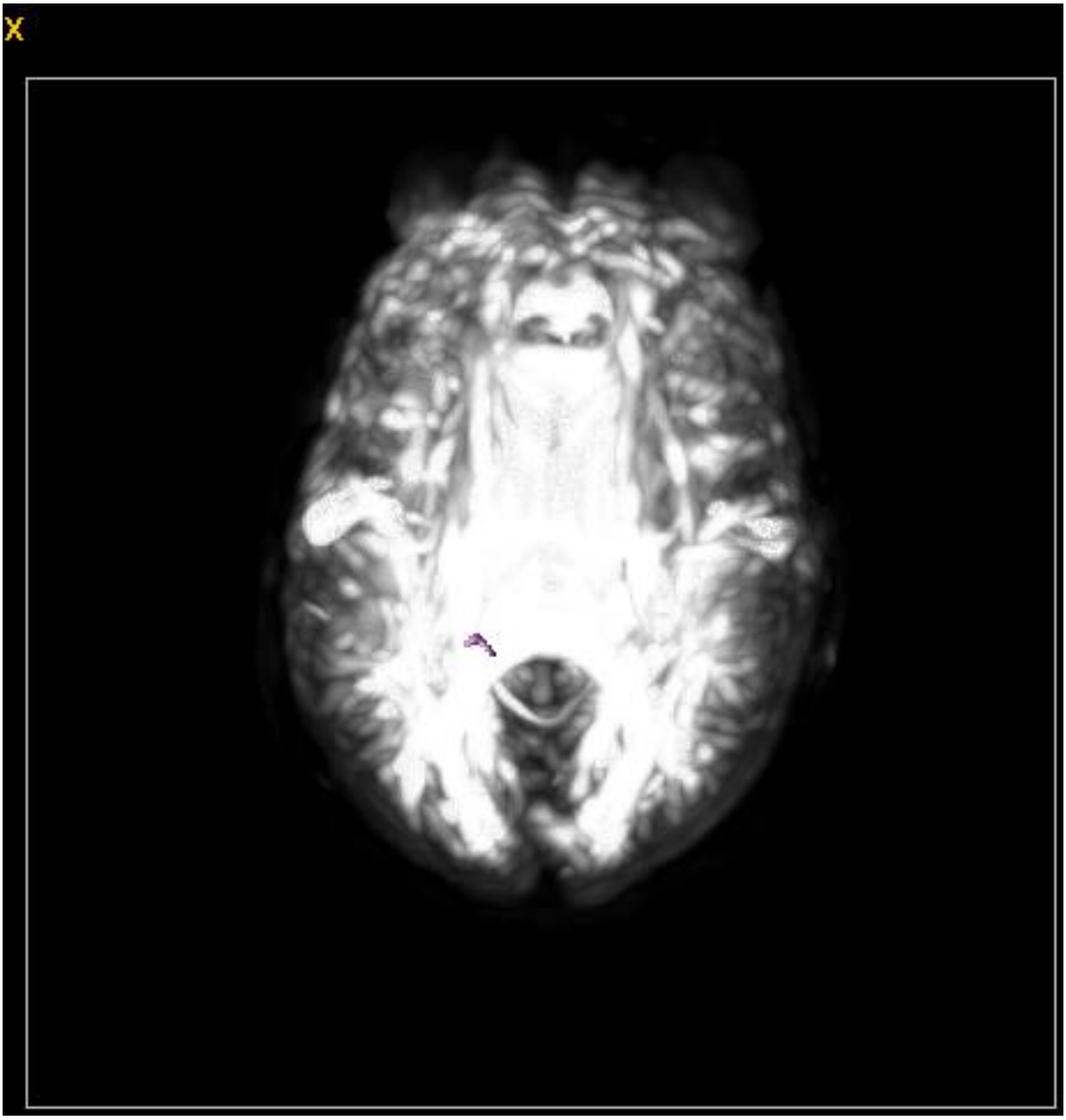


Figura 5.1.8

5.2 Resultados del método Monte Carlo

En los resultados obtenidos por este algoritmo, es donde más se puede apreciar la creación de un mapa de zonas interconectadas. Se ha decidido hacer múltiples pruebas, partiendo desde puntos distintos del cerebro y poder conseguir así que se dibujen las zonas en las que existen conexiones, en las imágenes esas zonas se han representado con diferentes colores.

El siguiente conjunto de imágenes son vistas superiores de las probabilidades de difusión (con Monte Carlo) para distintos puntos de inicio. Como las zonas obtenidas se iban solapando, hemos utilizado un editor de imágenes para convertir cada uno de los cubos 3D que nos generaba ImageJ en una capa de una imagen final que aglutinara todas las zonas pintadas.

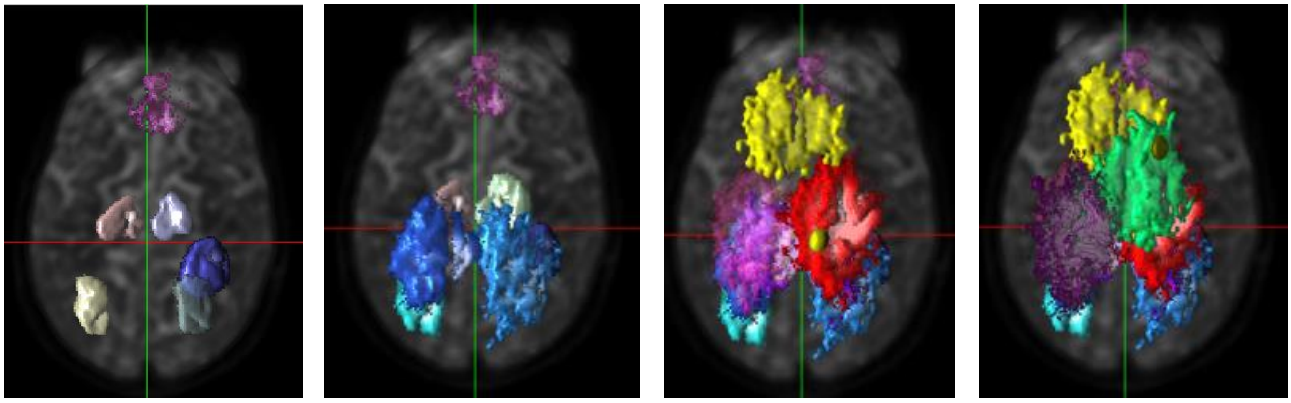


Figura 5.2.1.

En las siguientes figuras podemos ver otros resultados desde puntos de observación distintos.

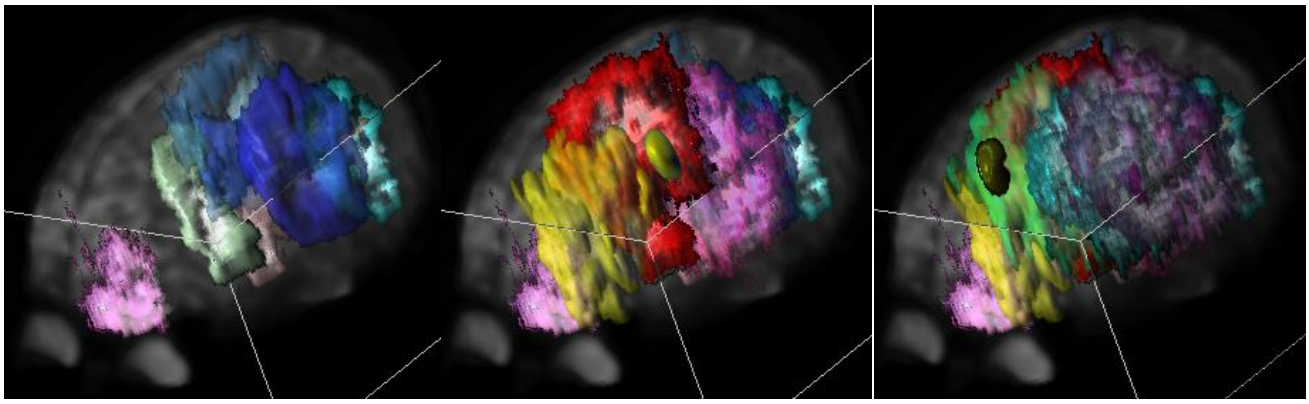


Figura 5.2.2

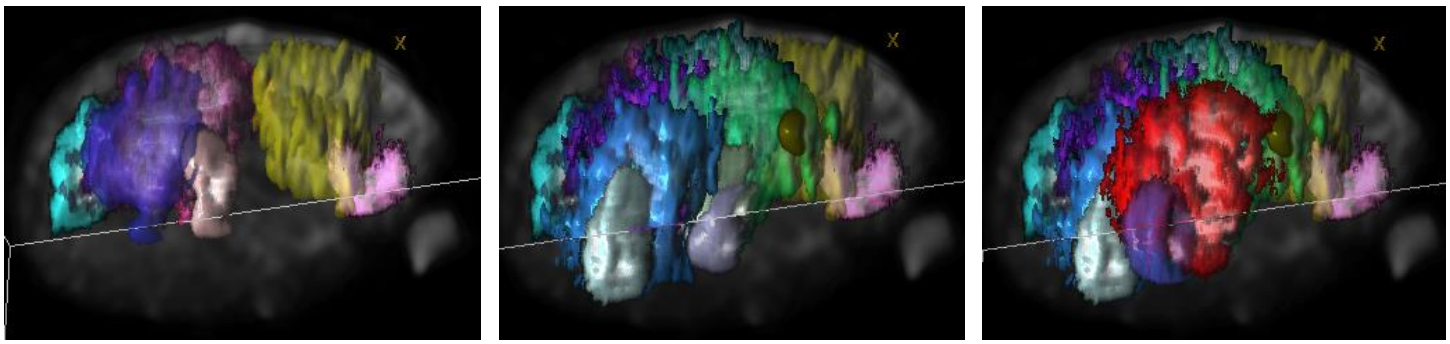


Figura 5.2.3.

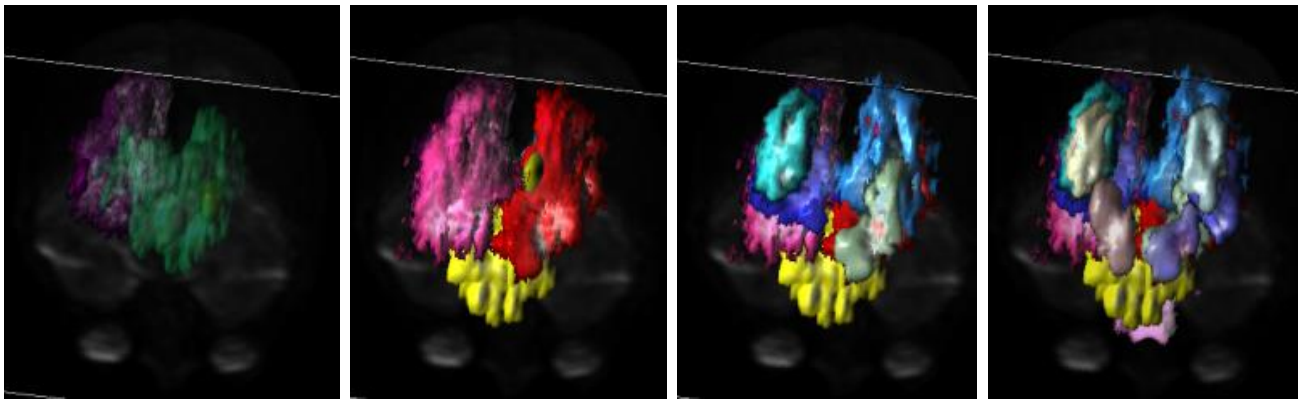


Figura 5.2.4.

En estas imágenes podemos observar la capacidad de este método para generar, dado un punto de inicio, una zona de interconexión neuronal, proporcionando la probabilidad de seguimiento de las diferentes rutas.

Capítulo 6.
Conclusiones y líneas futuras

Durante el desarrollo de este TFG se ha conseguido implementar una herramienta experimental para generar mapas de interconexión de áreas cerebrales a partir de imágenes de difusión tensorial. Para conseguir esta meta hemos tenido que adquirir conocimientos médicos, hasta ahora desconocidos, y descubrir y empezar a trabajar con aplicaciones como ImageJ que ayudan a comprender y a procesar imágenes médicas.

Una vez implementados y testeados todos los algoritmos que hemos desarrollado a lo largo de este TFG y examinando los resultados obtenidos podemos comprobar que el algoritmo basado en el método Monte Carlo es el más efectivo en la generación de zonas de interconexión neuronal, como se puede apreciar en las figuras del capítulo anterior, y posiblemente el que más posibilidades ofrezca para estudios clínicos. Sin embargo la posibilidad de trazar un solo trayecto a partir de un punto dado puede ser también de utilidad en otra clase de situaciones.

Como trabajo futuro se considera poder tratar un número mayor de imágenes, ese incremento en el tamaño de la secuencia significaría que hay más ángulos de observación, y por lo tanto, se podría medir mejor la anisotropía de los tejidos. Otra de las líneas de trabajo futuras podría ser intentar encontrar una forma de poder reducir el "ruido" en las imágenes ya sea a la hora de la captura o del procesamiento de las imágenes. Más centrados en crear nuevos algoritmos, se podría considerar la posibilidad de extraer los caminos favoritos de la secuencia obtenida en el tercer algoritmo y realizar tractografías a partir de regiones de interconexión.

Capítulo 7.
Conclusions and future work

During the development of this TFG we have been able to implement an experimental tool to generate maps of interconnection of brain areas from diffusion tensor imaging. To get to this goal we had to acquire medical knowledge, previously unknown, and discover and start working with applications such as ImageJ that helped us understand and process medical images.

Once we implemented and tested all the algorithms that we developed over this TFG and did a close examination of the results, we concluded that the Monte Carlo based algorithm is the most effective when generating areas of neuronal interconnection, as it is seen in the figures in the previous chapter. And it is possibly, the one who has the most potential in clinical studies. However, the possibility of establishing a single trajectory from a specific point could also be useful in other circumstances.

As a future development, we are considering the possibility to work with a greater number of images. This increment in the size of the sequence will mean more viewing angles, and as a result we could better measure the anisotropy of tissues. Another future research might be to try to find a way to reduce "noise" in images, either at the time of capture or at image processing. More focused on creating new algorithms, we could consider extracting the favorite ways of the sequence obtained in the third algorithm and make tractography from interconnection regions.

Capítulo 8. Presupuesto

En este capítulo se especifica un presupuesto que indica cuánto costaría realizar este Trabajo de Fin de Grado si se tratase de un trabajo encargado por un cliente.

8.1 Introducción y coste por hora

Se definirá una tabla con la lista de actividades desglosadas por cada tarea realizada en este Trabajo de Fin de Grado. Otra columna indicará la duración en horas que se han empleado para dicha actividad junto con el precio por hora calculado.

Tras realizar una consulta sobre el precio por hora de un analista programador *freelance* en España, el precio por hora que se considerará en este precupuesto es de 26€/hora.

8.2 Tareas previas y algoritmos secundarios

Actividad	Duración	Precio
Estudio de la aplicación ImageJ y lenguaje Java	15	390
Implementación de módulo de lectura de formatos	30	780
Implementación de la estructura para el almacenamiento y gestión de imágenes.	30	780
	Subtotal	1950

Tabla 8.2.1. Tabla de actividades, duración y precios del tercer y cuarto objetivo.

8.3 Desarrollo de los algoritmos principales

Actividad	Duración	Precio
Implementación del método "Búsqueda saltando al mayor pixel partiendo de uno dado"	45	1170
Implementación del método "Búsqueda saltando al mayor pixel partiendo de todos"	60	1560
Implementación del método "Búsqueda saltando mediante probabilidad"	24	624
	Subtotal	3354

Tabla 8.3.1. Tabla de actividades, duración y precios del quinto objetivo.

8.4 Preparación y análisis de resultados

Actividad	Duración	Precio
Obtención de las imágenes 3D con el plugin "Volume viewer"	30	780
Edición de las imágenes en GIMP	75	1950
Analisis de las imágenes finales	15	390
	Subtotal	3120

Tabla 8.4.1. Tabla de actividades, duración y precios del sexto objetivo.

8.5 Coste y duración total.

Actividad	Duración	Precio
Tareas previas y algoritmos secundarios	75	1950
Desarrollo de los algoritmos principales	129	3354
Preparación y análisis de resultados	120	3120
Subtotal	324	8424

Tabla 8.5.1. Precio y duración total.

Bibliografía

Articulos

[1] Alexander, A. L., Lee, J. E., Lazar, M., & Field, A. S. (2007). "Diffusion Tensor Imaging of the Brain". Neurotherapeutics : The Journal of the American Society for Experimental NeuroTherapeutics, 4(3), 316–329. doi:10.1016/j.nurt.2007.05.011

[2] Graessner, J., (2011) "Frequently Asked Questions: Diffusion-Weighted Imaging (DWI)", MAGNETOM World, ISMRM Edition - Issue 46, 1-2011.

[3] Mori, Susumu & van Zijl, Peter C. M. (2002). "Fiber tracking: principles and strategies – a technical review", NMR in Biomedicine, 15, 468-480. doi: 10.1002/nbm.781

Páginas web

[1] La Resonancia Magnética de Difusión revela la estructura nerviosa del cerebro. http://www.dicyt.com/noticias/la-resonancia-magnetica-de-difusion-revela-la-estructura-nerviosa-del-cerebro

[2] Clínica Ruber: Resonancia Magnética. http://www.ruber.es/ruber/servicios-medicos-privados/index.html?params=133,0,194,0,resonancia_magnetica

[3] Kidshealth: Resonancia Magnética. http://kidshealth.org/parent/en_espanol/medicos/mri_esp.html

[4] Imagen por resonancia magnética. http://es.wikipedia.org/wiki/Imagen_por_resonancia_magn%C3%A9tica

[5] Nuclear magnetic resonance. http://en.wikipedia.org/wiki/Nuclear_magnetic_resonance#History

[6] Tractografía. http://es.wikipedia.org/wiki/Tractograf%C3%ADa

[7] Tractography. http://en.wikipedia.org/wiki/Tractography

[8] Diffusion MRI. http://en.wikipedia.org/wiki/Diffusion_MRI#Diffusion_tensor_imaging

[9] DICOM. http://medical.nema.org/standard.html

[10] ¿Qué es el formato DICOM? Las claves del estándar en imágenes médicas. https://clinic-cloud.com/formato-dicom-que-es-estandar-imagenes-medicas/#

[11] ¿Qué es DICOM? http://www.c2ctsis.com/archives/231
[12] Diffusion Tensor Imaging. http://emedicine.medscape.com/article/345561-overview#aw2aab6b2
[13] ImageJ, Home page. http://rsb.info.nih.gov/ij/index.html
[14] Writing ImageJ Plugins—A Tutorial. http://www.gm.fh-koeln.de/~konen/WPF-BV/tutorial-ImageJ_V1.71.pdf
[15] Introducción a ImageJ. https://campusvirtual.univalle.edu.co/moodle/pluginfile.php/78612/mod_resource/content/0/JLopez_-_Intro_ImageJ.pdf
[16] Managing and Viewing DICOM Images with ImageJ.
[17] http://santec.tudor.lu/_media/project/dicom/tudor_dicom_ijconf2008_paper.pdf