Universidad de Concepción

Facultad de Ingeniería

Departamento de Ingeniería Informática

Extensión y reestructuración de biblioteca Phybers.

Memoria de Título presentada para optar al título de Ingeniero Civil Informático

Autor: Alejandro Cofre Garcia.

Profesora Patrocinante: Cecilia Hernández R.

Profesora co-guía: Pamela Guevara A.

Concepción, Chile 2 de octubre de 2024

RESUMEN

El estudio del cerebro es fundamental para entender tanto las funciones normales como las patologías del sistema nervioso. La biblioteca Phybers de Python facilita el análisis de las fibras neuronales mediante técnicas avanzadas de procesamiento y visualización de datos de dMRI. Esta memoria de título se centra en la extensión y reestructuración de Phybers para mejorar su usabilidad y funcionalidad, haciendo énfasis en la integración de módulos en C/C++ a través de Cython, la reorganización del paquete mejorar la organización y compatibilidad multiplataforma, y la implementación de mejoras en las áreas de clustering, segmentación y visualización de fibras cerebrales.

En primer lugar, se integraron módulos de procesamiento de alto rendimiento escritos en C/C++. La utilización de Cython fue crucial para crear una interfaz eficiente entre Python y C/C++, manteniendo la simplicidad del código en Python sin sacrificar el rendimiento. Se reestructuró la biblioteca para adoptar una organización más modular, facilitando su mantenimiento y ampliación futura. Además, se aseguró la compatibilidad y la depuración en múltiples sistemas operativos, incluyendo Windows y Linux, lo que amplía el alcance de la biblioteca.

Se realizó la implementación correcta de estructura de paquete de Python. El módulo de visualización también fue mejorado para proporcionar una representación gráfica más intuitiva y detallada de los datos de tractografía, incluyendo la capacidad de personalizar la visualización y manipular las fibras en un entorno gráfico interactivo.

El trabajo realizado en esta memoria estandarizó y expandió las capacidades de Phybers, estableciendo una base sólida para futuras investigaciones utilizando esta biblioteca. Las mejoras implementadas facilitarán a los investigadores el estudio de la conectividad cerebral, ofreciendo herramientas más robustas y accesibles para el análisis de datos complejos. Los resultados obtenidos presentan una solución eficaz a los problemas identificados en la versión original de Phybers.

ÍNDICE

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Desde el siglo XVII, el estudio del cerebro ha marcado un campo importante en la ciencia [1]. La necesidad de descifrar el enigma que construyen sus interconexiones neuronales ha impulsado el desarrollo de tecnologías de vanguardia, entre las cuales la resonancia magnética de difusión (dMRI) destaca, ofreciendo una perspectiva única sobre las interconexiones estructurales de la materia blanca en el cerebro. Es una técnica no invasiva que permite capturar imágenes de la arquitectura cerebral e inferir una red de conexiones por medio del movimiento de moléculas de agua dentro de él [2], esta metodología aporta a la investigación en desafíos neurológicos como el entendimiento de discapacidades, trastornos y enfermedades [3].

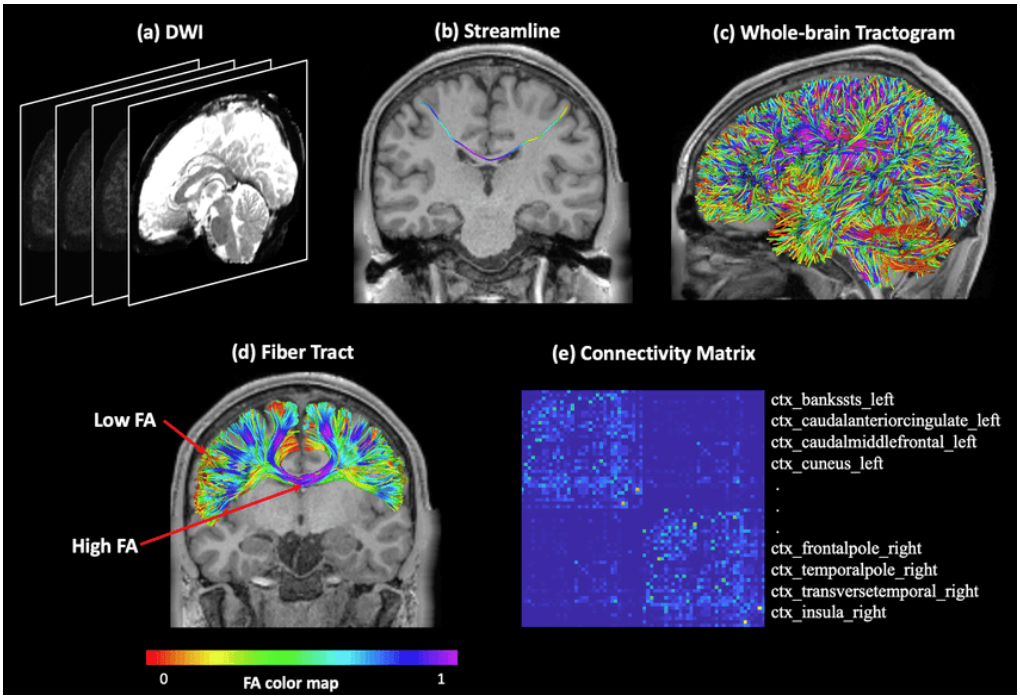


Figura 1: Ejemplo gráfico de tractografía

Actualmente la neurociencia abarca un espectro investigativo amplio que va desde el refinamiento del análisis de tractografía cerebral, crucial para el mapeo de redes neuronales[4], hasta aplicaciones específicas como el estudio del envejecimiento cerebral mediante imágenes por tensor de difusión [5]. Innovaciones como el marco teórico de tractoFormer y el análisis de materia blanca superficial [6] constituyen ejemplos destacados del análisis de datos neurológicos, para una tabla comparativa, referirse a anexos [A] y [B].

Adicionalmente, la integración de inteligencia artificial y aprendizaje automático en el análisis de datos obtenidos mediante dMRI ha inaugurado nuevas vías para el diagnóstico [8] y la

investigación. La habilidad de estas tecnologías para procesar volúmenes de datos extensos y detectar patrones sutiles podría revolucionar nuestra capacidad diagnóstica en fases tempranas de enfermedades y la personalización de tratamientos [9] . Es por esto que es sumamente importante obtener estos datos de forma rápida y confiable.

Las fibras cerebrales son conjuntos de datos de polilíneas 3D que representan las principales trayectorias de los fascículos de fibras de la materia blanca. Estas se calculan mediante tractografía a partir de datos de Resonancia Magnética por Difusión (dMRI). Estos datos son complejos y de gran tamaño, además de contener artefactos. Para analizarlos, se utilizan métodos de clustering que agrupan las fibras según su forma y posición, y algoritmos de segmentación que extraen fascículos conocidos usando información anatómica, como atlas de fascículos de fibras. Estos algoritmos permiten estudiar la conectividad cerebral en cerebros sanos y patológicos.

La biblioteca de Python Phybers se centra en el análisis de estas fibras cerebrales, encapsulando módulos para clustering, análisis de clústeres, manipulación de fibras y visualización de resultados. Este trabajo se enfocó en mejorar la usabilidad y expandir la funcionalidad de Phybers, asegurando su compatibilidad multiplataforma, integrando módulos en C/C++ mediante Cython, y optimizando la estructura del paquete. El estado final de Phybers presenta una herramienta más eficiente, accesible y robusta, lista para ser utilizada en investigaciones avanzadas de conectividad cerebral.

1.1 Objetivos

1.1.1 Objetivo General

El objetivo general de esta memoria de título es validar, extender y mejorar la biblioteca Phybers con el fin de obtener un paquete robusto y asequible.

1.1.2 Objetivos específicos

Los objetivos originales al comienzo de esta memoria de título se centran:

- **Análisis de la implementación actual de Phybers**: Evaluar y determinar puntos de mejora en la arquitectura y funcionalidad de la biblioteca.
- **Integración del código en C/C++**: Desarrollar e incorporar código de procesamiento de alto rendimiento escrito en C/C++ como módulos de Python utilizando Cython.
- **Mejoras en el módulo de visualización**: Implementar nuevas funcionalidades en la interfaz gráfica para una representación más detallada de los datos de tractografía.
- **Estandarización y simplificación del código**: Reestructurar la biblioteca para adoptar una organización más modular y estandarizada, facilitando su mantenimiento y expansión futura.

- **Compatibilidad multiplataforma**: Asegurar la compatibilidad y la depuración del paquete en Windows y Linux, para ampliar su accesibilidad.
- **Implementación adecuada del proceso de compilación, empaquetado y distribución**: Desarrollar un proceso robusto y simplificado para la compilación, empaquetado y distribución del paquete, utilizando herramientas estándar como setup.py, MANIFEST.in y pyproject.toml.

1.2 Metodología

Para llevar a cabo esta memoria, se adoptó una metodología ágil, caracterizada por la organización de reuniones breves pero frecuentes. Estas sesiones estuvieron enfocadas en establecer objetivos adaptativos, resolver problemas emergentes y compartir avances, en colaboración directa con la estudiante de postgrado Liset González y las profesoras Cecilia Paola Hernández Rivas y Pamela Beatriz Guevara Alvez, quienes cumplieron roles de stakeholders. Ellas participaron activamente en la revisión de las propuestas y en la validación del alineamiento de las soluciones con los objetivos del proyecto.

El desarrollo del trabajo se estructuró en las siguientes fases:

1. **Investigación**: Se identificaron las secciones que necesitaban modificaciones y se realizó un análisis detallado del código existente.
2. **Diseño:** Se propuso una nueva arquitectura que ofrece una mejora estructural para el paquete.
3. **Implementación :** Se procedió con la puesta en práctica del diseño propuesto, aplicando las modificaciones adecuadas.
4. **Validación:** Se realizó deploy con profesores, comprobando que las instalaciones funcionarán correctamente.

Considerar que los puntos 1, 2 y 3 se repitieron por cada mejora estructural sobre la base original y extensión prevista.

1.3 Limitaciones

La necesidad de una reestructuración del código fue más extensa de lo inicialmente planificado, presentando un desafío adicional. Las restricciones de tiempo incrementaron al tener que modificar aspectos fundamentales de la biblioteca, lo que complicó el proceso. Además, la diversidad de los códigos en C dentro de la biblioteca requería una estandarización que habría consumido significativos recursos; sin embargo, esta estandarización no se llevó a cabo, y adaptar el trabajo a cada código individual resultó menos laborioso que una estandarización completa. Adicionalmente se enfrentaron dificultades con casos de prueba cuyos conjuntos de datos eran demasiado extensos y carecían de resultados de referencia para su comparación y evaluación adecuada.

1.4 Resultados esperados

- **Fácil Instalación**: Facilitar la instalación de la biblioteca en cualquier dispositivo sin requerir conocimientos avanzados previos de Python.
- **Acceso Mejorado:** Proporcionar acceso directo y simplificado a las extensiones dentro de la biblioteca, disponible en linux y windows.
- **Ampliación de Opciones para Investigadores:** Enriquecer el repertorio de herramientas disponibles para los investigadores, ampliando así las posibilidades de análisis y estudio.

2 MARCO TEÓRICO

Esta sección entrega la definiciones de conceptos relacionados al trabajo realizado, incluyendo tecnologías y definiciones.

2.1 PYTHON

Python es un popular lenguaje de programación de alto nivel, interpretado y de propósito general, conocido por su legibilidad y eficiencia. Python soporta múltiples paradigmas de programación, incluyendo programación orientada a objetos, imperativa y funcional, lo que lo hace versátil para una amplia variedad de aplicaciones. A continuación se muestran contenidos de python fundamentales para el entendimiento del trabajo realizado.

2.1.1 Cython

Cython es un compilador estático optimizado tanto para el lenguaje de programación Python como para el lenguaje de programación extendido Cython (basado en Pyrex). Facilita la escritura de extensiones en C para Python de tal forma como si se tratase del propio Python. [10] Se utiliza en las extensiones de Phybers que utilizan código en C.

El código de Cython se escribe en archivos con extensión .pyx, es compilado a código en CPython con extensión .c y éste a su vez es compilado a un módulo importable de python con extensión .pyd.

2.1.2 Paquetes de Python

En python, un script es definido como un archivo de texto que posee la extensión "*.py*". Este script, susceptible de ser importado desde otros scripts o a través de una interfaz interactiva, se categoriza como un módulo. Un paquete, por su parte, constituye una categoría especial de módulo capaz de albergar otros módulos dentro de sí. La estructura más comúnmente adoptada para un paquete comprende una carpeta que contiene varios módulos y un archivo especial denominado *"__init__.py"*. Este archivo puede ser vacío o incluir código que se ejecuta durante la inicialización del paquete.

Hay tres métodos que por los que un módulo puede ser importado:

- La búsqueda en el directorio activo de un archivo que corresponda al nombre del módulo con extensión *" .py"*.
- La búsqueda en las ubicaciones que el intérprete de Python tiene registradas para localizar módulos.
- La utilización de la sintaxis "*from paquete import modulo"* para importar módulos de manera relativa dentro del mismo paquete.

El primer método es especialmente útil para scripts que no están destinados a ser distribuidos o reutilizados, mientras que el segundo método constituye el principal mecanismo para la administración de paquetes externos que los usuarios instalan. El tercer método se emplea de manera interna en paquetes diseñados para la distribución, y es considerado un enfoque fundamental en el desarrollo de software modular, como en el caso de Phybers. Las importaciones relativas eliminan posibles ambigüedades en la importación de paquetes con nombres comunes y facilitan la exploración interactiva del contenido de un paquete.

2.1.3 Distribución

La distribución de paquetes en Python es un proceso que facilita el uso del software. Una estructuración y distribución adecuadas provee que los módulos y paquetes sean accesibles tanto para la comunidad de desarrolladores como para los usuarios finales, contribuyendo significativamente a la eficacia del paquete en distintos sistemas y configuraciones.

- **Manifest.in**

Archivo que contiene los comandos que permiten describir y manipular una lista de archivos [11], es esencial para especificar qué archivos no codificados en Python, como documentación, datos y shaders, deben incluirse en el paquete.

- **Pyproject.toml**

Introducido en las versiones más recientes de Python, este archivo de configuración define los requerimientos de construcción del paquete, incluyendo las herramientas y versiones necesarias para compilar el paquete desde la fuente [12]. Este archivo estandariza y simplifica el proceso de empaquetamiento y distribución, proporcionando una descripción más robusta en comparación con el tradicional setup.py.

- **Extensiones binarias**

Las Extensiones Binarias son módulos escritos en lenguajes como C o C++ y compilados para plataformas específicas (archivos .so para Unix, .pyd para Windows). Permiten que operaciones computacionalmente intensivas se realicen mucho más rápidamente que si se ejecutaran en Python puro, mejorando significativamente el rendimiento del paquete.

- **Código Fuente**

El Código Fuente incluido en los paquetes permite a los usuarios finales y desarrolladores revisar y modificar el comportamiento del software según sus necesidades.

- **Cwrapper**

Cwrapper es una clase template para envolver recursos de C en RAII (Resource acquisition is initialization) [13], se utiliza cuando integran módulos de C/C++ mediante Cython, actuando como un puente entre el código de Python y las funciones nativas de C/C++. Estos envoltorios

son esenciales para la correcta ejecución de funciones compiladas, proporcionando una interfaz segura y eficiente entre los diferentes lenguajes de programación.

2.1.4 Pistas de tipo

Python posee tipo dinámico, lo cual permite que las entradas y salidas de funciones admitan una amplia variedad de tipos. No obstante, Python soporta la posibilidad de utilizar pistas de tipo. Estas permiten especificar el tipo de dato esperado por cada argumento de una función, así como el tipo de dato que se espera retornar. De manera similar, es posible indicar el tipo de datos esperado para atributos y variables.

Estas pistas de tipo no deben influir en la ejecución del código pero, pueden ser objeto de verificación durante la ejecución si así se configura. El principal propósito de las pistas de tipo se manifiesta durante la fase de edición del código. Estas facilitan la verificación de la correctitud del código y proporcionan datos valiosos para la función de autocompletar.

Además, la herramientas Cython aprovecha estas pistas de tipo para compilar código en C/C++ de manera más eficiente en comparación con la ejecución de código interpretado tradicionalmente en Python. Cython extiende la funcionalidad del lenguaje Python, permitiendo la integración con código en C/C++ mediante la adición de sintaxis necesaria en archivos con extensión .pyx. En Cython, es posible combinar las pistas de tipo con un sistema de tipeo estático para optimizar aún más la ejecución del código.

2.1.5 Docstrings

Los docstrings son un elemento de sintaxis para la documentación interna del código. Estos se definen mediante cadenas de texto encerradas en triples apóstrofes o comillas y se colocan inmediatamente después de la definición de una función o método, tras la declaración de una clase, o al inicio de un módulo. Los docstrings sirven para dar una descripción detallada del objeto que documentan, facilitando la mejor comprensión de su propósito.

Esta documentación es accesible durante el desarrollo del código y a través de interfaces interactivas como IPython, donde los usuarios pueden obtener ayuda específica sobre los objetos consultando el docstring correspondiente mediante comandos como *“help(objeto)”* o *“objeto?*”. Los docstrings son fundamentales para tener claridad sobre el código, también permiten su conversión a otros formatos de documentación, como páginas web o documentos PDF, esto facilita la ampliación sin comprometer la legibilidad del código fuente.

2.1.6 Setup.py

Un paquete de Python debe incluir información y código para ser correctamente identificado e instalado en un ambiente determinado. Uno de los mecanismos para conseguir esto es la inclusión de un script llamado setup.py en la raíz del paquete. [14]

En este script se puede:

- Especificar datos, como autores, licencias, versión, descripción del paquete y más.
- Presentar distintos módulos de extensión que deben ser compilados para usar el paquete, junto con información necesaria para compilarlos.
- Exponer puntos de entrada para interactuar con otros programas o ofrecer interfaces de línea de comando o interfaces gráficas como ejecutables.
- Declarar paquetes requeridos para usar el paquete.
- Marcar archivos presentes en el paquete que no son códigos, pero deben ser incluidos para el funcionamiento del paquete.

2.2 Conceptos relacionados

2.2.1 Clustering

El clustering en Phybers es una técnica avanzada utilizada para agrupar fibras neuronales según su similitud geométrica [15] y espacial, lo que facilita el análisis de la conectividad cerebral. Los principales algoritmos de clustering implementados en Phybers son:

- **K-Means**: Este algoritmo agrupa las fibras neuronales en k clústeres definidos por la distancia mínima entre las fibras y los centroides del clúster. Se itera hasta que los centroides se estabilizan y no hay cambios significativos en la asignación de fibras a clústeres.
- **Clustering Jerárquico (HClust)** : Este método construye una jerarquía de clústeres mediante un enfoque aglomerativo o divisivo. En el enfoque aglomerativo, cada fibra comienza como un clúster individual, y estos se combinan iterativamente en clústeres más grandes. En el enfoque divisivo, se parte de un único clúster que se divide sucesivamente en clústeres más pequeños. Phybers utiliza un enfoque basado en distancias euclidianas para determinar las uniones o divisiones óptimas.
- **Map Clustering**: Este método utiliza un mapa de similitud para agrupar las fibras basándose en las características geométricas y topológicas. Se generan mapas de similitud que representan la proximidad entre fibras, y se aplican técnicas de reducción de dimensionalidad para identificar grupos densos de fibras.
- **FFClust (FFClust)**: Utiliza técnicas de clustering rápido para agrupar fibras neuronales basadas en similitudes geométricas y espaciales. Implementa el algoritmo k-means y optimizaciones específicas para manejar grandes conjuntos de datos de tractografía.

Estos algoritmos están optimizados para manejar datos de tractografía cerebral de alta dimensionalidad y complejidad, permitiendo una agrupación precisa que es crucial para estudios de conectividad cerebral.

2.2.2 Visualización

La visualización en Phybers permite representar gráficamente las trayectorias de las fibras neuronales, facilitando su análisis e interpretación. Las características clave del módulo de visualización incluyen:

- **Interacción en Tiempo Real:** Permite a los usuarios manipular y explorar las fibras en un entorno gráfico interactivo, ajustando la vista, el enfoque y el color de las fibras.
- **Personalización de la Visualización:** Ofrece herramientas para cambiar colores, enfocar la visualización en fibras específicas y visualizar los límites de los objetos. También permite la manipulación de fibras en tiempo real, lo que facilita una comprensión intuitiva de las complejas redes de materia blanca.
- **Integración de Datos Multimodales:** El módulo puede superponer datos de tractografía con imágenes de resonancia magnética, proporcionando un contexto anatómico detallado que mejora la interpretación de las fibras.

2.2.3 Segmentación

La segmentación en Phybers se centra en identificar y aislar conjuntos de fibras que representan tractos o fascículos específicos dentro de la materia blanca. Los algoritmos de segmentación utilizan criterios avanzados basados en similitud geométrica y proximidad espacial, así como información anatómica de atlas cerebrales:

- **Segmentación Basada en Atlas**: Utiliza atlas predefinidos para identificar y extraer fascículos específicos. Esto es esencial para estudios detallados de estructuras específicas de materia blanca y para comparaciones entre sujetos.
- **Segmentación Automática**: Algoritmos que identifican segmentos de fibras basados en características geométricas sin necesidad de una intervención manual, mejorando la precisión y eficiencia del proceso.
- **ROI (Region of Interest)**: Permite a los usuarios definir regiones específicas en las que se enfocará la segmentación, facilitando estudios dirigidos a áreas particulares del cerebro.

Estos métodos mejoran la precisión y eficiencia de la segmentación, lo que es fundamental para estudios de conectividad cerebral en condiciones tanto normales como patológicas.

2.3 Introducción a Phybers

Phybers es una biblioteca de Python diseñada para el análisis de tractografía cerebral, integrando módulos en C/C++ para mejorar el rendimiento. Facilita la manipulación, clustering, segmentación y visualización de fibras neuronales, utilizando datos obtenidos mediante resonancia magnética por difusión (dMRI). Phybers es compatible con sistemas operativos Windows y Ubuntu, y se distribuye como software de código abierto, lo que permite su uso y extensión en investigaciones neurocientíficas.

2.3.1 Fibras cerebrales y bundles

Las fibras cerebrales son estructuras cruciales para comprender la conectividad en el tejido cerebral, extendiéndose a través de la materia gris. Están compuestas por neuronas interconectadas y se visualizan mediante una lista de puntos en un espacio tridimensional. Estos datos se almacenan en un formato denominado "bundles". Este formato organiza una o más fibras cerebrales en un archivo con extensión .bundles, que describe las características generales de las fibras, y un archivo adicional con extensión .bundlesdata, que contiene los datos específicos de las fibras. Cada fibra se registra indicando el número de puntos que la componen, seguido por las coordenadas x, y, z de cada punto de forma secuencial.

Debido a la estructura de este formato, es necesario procesar las fibras de manera individual, ya que para determinar los datos de inicio y fin de una fibra específica es esencial conocer el final de los datos de la fibra anterior. Un aspecto destacable de estos datos es que cada fibra puede variar en el número de puntos que contiene.

2.3.2 Estructura de datos bundle en C/C++

Para el procesamiento de estas fibras durante la ejecución de programas, se utiliza una representación en estructuras en C/C++ definida como sigue:

```
struct bundle {
    int32_t nfibers;
    int32_t* npoints;
    float** points;
 };
```

Figura 2 : Representación de las fibras durante su procesamiento en módulos en C/C++

Esta estructura de datos requiere un total de $2 + nfibers$ asignaciones de memoria dinámica para cada bundle, adecuándose a la variabilidad en el número de puntos por fibra.

2.3.3 Módulos de procesamiento de fibras

Phybers incluye varios módulos especializados en el procesamiento de estos datos, entre los que se encuentran: ffclust, hclust, segment, deform, intersection, postprocessing y sampling. Cada uno de estos módulos cumple funciones específicas en el manejo, análisis y modificación de las fibras cerebrales.

2.3.4 Módulo de visualización

El componente de visualización de Phybers, conocido como FiberVis, facilita la visualización de mallados que representan la corteza cerebral, bundles para las fibras cerebrales y MRI, que proporcionan un contexto visual basado en imágenes en serie del cráneo del paciente. Este

módulo es fundamental para la interpretación visual de los datos estructurales y funcionales del cerebro.

3 TRABAJO REALIZADO

El desarrollo del proyecto se centró en la expansión y optimización de Phybers. Además de la estabilización de las funcionalidades básicas del paquete, asegurando una base funcional.

Para mejorar la integración entre los módulos nativos de C/C++ y Python, se implementaron adaptadores en Cython, lo que permitió una llamada eficiente de funciones como si fueran aplicaciones externas, manteniendo un alto rendimiento del código nativo. Esta sección detalla los módulos en C/C++ en los que se trabajó, incluyendo FFClust y HClust para el clustering de fibras cerebrales, y Segmentation para segmentar conjuntos de fibras, entre otros.

Además, se realizaron adaptaciones para garantizar la compatibilidad del código en múltiples sistemas operativos, y se reestructuró la arquitectura del paquete para mejorar su estructura y funcionamiento dentro del ecosistema de Python, facilitando futuras integraciones y mantenimiento.

3.1 MÓDULOS DE CYTHON

Inicialmente, se planificó la integración del módulo de parcelación dentro de la biblioteca Phybers para expandir sus capacidades en el análisis de datos neurocientíficos. Sin embargo, con el propósito de asegurar un estado funcional base del paquete, la incorporación de dicho módulo fue deliberadamente aplazada. Esta decisión estratégica permitió centrar los esfuerzos en la estabilización y optimización de la funcionalidad básica del paquete, alineándose principalmente con los objetivos 1 y 3 del proyecto y, en una menor medida, con el objetivo 4. Esta aproximación pragmática aseguró que las funcionalidades esenciales fueran robustas y operativas antes de añadir complejidad adicional al sistema.

Para facilitar la interacción entre los módulos nativos de C/C++ y Python, se adoptó un enfoque de implementación inicial de adaptadores en Cython de manera rudimentaria. Este enfoque consistió en desempaquetar los argumentos de las funciones principales para permitir su llamada desde Python de manera análoga a como se invocarían si fueran aplicaciones externas independientes. Este método permitió una integración eficaz y flexible de funciones escritas en C/C++ dentro del ecosistema de Python, manteniendo un rendimiento cercano al del código nativo mientras se aprovechaba la simplicidad y legibilidad de Python.

A continuación, se presenta una lista detallada de los módulos implementados en Cython dentro de la biblioteca Phybers, destacando sus funciones y aplicaciones específicas:

3.1.1 Módulos

- FFClust y HClust:

Diseñados para el clustering de fibras cerebrales, utilizando algoritmos como k-means para agrupar fibras basándose en propiedades geométricas y de proximidad espacial. Facilitan el análisis de la conectividad cerebral y la identificación de patrones comunes en datos de tractografía.

- Segmentation:

Permite la segmentación de conjuntos de fibras utilizando atlas cerebrales predefinidos para identificar y aislar fascículos específicos. Es crucial para estudios detallados de la estructura de la materia blanca y comparaciones entre sujetos sanos y aquellos con afecciones neurológicas.

- Deform:

Permite la transformación de fibras a diferentes espacios anatómicos usando campos de deformación, ajustando las fibras a variaciones individuales en la anatomía cerebral. Importante para estudios longitudinales y de seguimiento en neurociencia.

- Fiber Distance y Slice Fiber:

Calculan distancias entre fibras y cambia el número de puntos de fibras usando interpolación respectivamente, permitiendo una evaluación detallada de las relaciones espaciales entre fibras. Utilizados en el análisis cuantitativo de la conectividad y en la verificación de la consistencia de los resultados de clustering y segmentación.

- PostProcessing:

Analiza y filtra los resultados obtenidos de los procesos de clustering y segmentación para mejorar la calidad y precisión de los datos de salida. Es esencial para asegurar la validez y confiabilidad de los resultados en investigaciones neurológicas.

- Fibervis_ext:

Extensión dedicada a la visualización avanzada de fibras cerebrales, ofreciendo herramientas gráficas para representar de manera efectiva las complejidades de las redes neuronales. Proporciona una interfaz visual intuitiva para la exploración y análisis de datos estructurales y funcionales del cerebro.

3.1.2 Modificaciones a código C para funcionar en windows

El desarrollo para múltiples sistemas operativos requiere adaptaciones específicas para manejar las diferencias inherentes entre plataformas. Para el caso de la biblioteca Phybers, que inicialmente fue desarrollada para ambientes Linux, se enfrentaron desafíos al trasladar su funcionamiento a Windows debido a la dependencia de ciertas bibliotecas y cabeceras que son exclusivas de sistemas Unix, como dirent.h.

Para superar las limitaciones impuestas por la falta de la cabecera dirent.h en Windows, se adoptó la biblioteca win_dirent.h. Esta biblioteca es un wrapper que emula la funcionalidad de dirent.h, permitiendo que los códigos que dependen de esta última puedan ejecutarse en Windows sin necesidad de reescribir el código fundamental. Esto implicó los siguientes pasos:

- Inclusion de win_dirent.h

Se añadió la biblioteca win_dirent.h al conjunto de cabeceras incluidas en el proyecto.

- Modificación del código existente

En las secciones del código donde originalmente se utilizaba dirent.h, se realizaron modificaciones para que, en su lugar, se hiciera uso de win_dirent.h cuando el código se compilara para Windows. Esta adaptación específica asegura que el código mantenga su funcionalidad original sin errores de compilación.

- Pruebas de compatibilidad

Se llevaron a cabo pruebas de uso para verificar que las modificaciones no introdujera nuevos errores y que el comportamiento del software en Windows replicara el esperado en Unix.

3.2 IMPLEMENTACIÓN CORRECTA DE ESTRUCTURA DE PAQUETE DE PYTHON

El paquete original presentaba un uso predominante de importación de módulos desde el mismo directorio y, en algunos casos, modificar los directorios que el intérprete de Python utiliza para localizar paquetes antes de realizar importaciones absolutas de sus módulos. Este enfoque resultó en la instalación de subpaquetes como si fueran paquetes independientes, creando un entorno con paquetes desconectados y sin un paquete raíz común. Para resolver esto, se modificaron todas las importaciones dentro del paquete Phybers para que usaran importaciones relativas de los módulos necesarios.

Cada ramificación del paquete se aseguró de contener un archivo __init__.py, y se realizaron ajustes donde se requerían versiones específicas de __init__.py para cumplir con las necesidades de importación del paquete. Esta estructura facilita la organización y el reconocimiento de los componentes del paquete por parte del intérprete de Python.

Adicionalmente, el uso excesivo de la sintaxis from paquete import * había oscurecido el origen de múltiples funciones y clases utilizadas en el código. Esto condujo a un proceso de prueba y error para determinar qué módulos necesitaban ser importados y qué elementos específicos se debían incluir desde esos módulos. Para clarificar y optimizar la importación de módulos, los scripts __init__.py en los módulos del paquete Phybers se actualizaron para incluir las

importaciones deseadas. Esto asegura que la importación del paquete Phybers propague adecuadamente la ejecución e importación de los módulos requeridos.

Debido a los nombres coincidentes entre los subpaquetes y las funciones principales del módulo de clustering, fue necesario oscurecer los módulos ffclust y hclust para permitir el acceso a las funciones homónimas a través de una sintaxis más abreviada.

A continuación se presenta un ejemplo de lo mencionado:

```
# Forma original de importar funcion ffclust:
>>> from phybers.clustering.ffclust.mainFFClust import ffclust

# Para mejorar la legibilidad se desea poder usar esta forma
# Forma modificada para importar funcion ffclust:
>>> from phybers.clustering import ffclust
```

Figura 3: Ejemplo de cambio de sintaxis

Pese a que se realizó la modificación de la importación y que se puede utilizar la forma original, no se recomienda debido a que no está asegurado su funcionamiento correcto.

3.2.1 Árbol del paquete, incluyendo funciones.

```
└── Clustering
    └── Ffclust
        ├── c_wrappers
        │   └── seg
        ├── clustering
        │   ├── cluster_kmeans
        │   ├── parallel_points_clustering
        │   ├── map_clustering
        │   ├── split_fibers
        │   ├── clusters_to_clustermap
        │   ├── get_groups
        │   ├── small_clusters_reasignment
        │   ├── MapClustering
        │   ├── joinable_clusters
        │   ├── clique_join
        │   └── parallel_group_join_clique
        ├── FiberTools
        │   ├── getBundleSize
        │   ├── fiber_lens
        │   └── Stadistics_txt_toxlsx
```

Figura 4: Representación de arquitectura en árbol jerárquico

3.3 Depuración de módulos

Una vez que se logró la correcta importación de los distintos módulos, se inició la depuración de sus funciones principales. Este proceso resultó ser más largo de lo anticipado, principalmente debido a la insuficiente gestión de errores en las funciones de C/C++. A diferencia de Python, donde es posible revisar el estado del intérprete tras un error no manejado para identificar rápidamente el problema, los errores en el código compilado de C/C++ suelen limitarse a mensajes genéricos como "segmentation fault", los cuales carecen de contexto específico.

Los errores identificados incluyeron problemas en el traspaso de datos entre Python y C, manejo inadecuado de referencias a archivos por línea de comando, problemas de terminación

de líneas, longitud incorrecta de líneas, y manejo deficiente de strings y archivos. También se detectaron errores relacionados con el símbolo de directorio.

Para resolver estos problemas, se aseguró que las llamadas dentro de Python funcionaran de manera idéntica a las ejecutadas por línea de comando, lo que permitió atribuir los fallos directamente al código en C. Además, se corrigieron errores de manejo de punteros y se ajustaron las prácticas de gestión de memoria, abordando tanto la falta de liberación de memoria como la liberación prematura de la misma.

Después de asegurar que la funcionalidad del paquete estuviera mayormente libre de errores fatales, se procedió a realizar una verificación más extensiva de los módulos. Durante esta fase, se identificaron instancias de código incorrecto o funciones no implementadas, como la falta de 'dirent.h'. Finalmente, se completó el procedimiento de instalación del paquete, evaluando su comportamiento en la instalación desde el código fuente en sistemas Windows y Linux, así como el desempeño de las versiones compiladas del paquete. Este proceso exhaustivo permitió que el paquete funcionará adecuadamente en diferentes plataformas y configuraciones.

3.4 Compilación y distribución

En el archivo setup.py, se especifican los argumentos necesarios para los compiladores gcc y msvc, además de los módulos que deben ser compilados junto con sus respectivos códigos fuente. También se realiza una verificación del estado de los archivos fuente en Cython y C/C++, examinando la existencia del archivo y su tiempo de modificación, lo que incluye una revisión de las propiedades del archivo y los metadatos. Esto permite determinar si es necesario recompilar el código en Cython.

Adicionalmente, los archivos que deben incluirse en el paquete para su distribución se enumeran en MANIFEST.in. Esto incluye atlases utilizados en la segmentación y el código compilado de Cython. Las herramientas requeridas para la compilación se declaran en pyproject.toml.

Este enfoque facilita la obtención de una versión local funcional del paquete simplemente descargando el código fuente y ejecutando el comando python setup.py build_ext –inplace. Para generar versiones destinadas a la distribución, se utiliza el módulo build con el comando python -m build, y los paquetes resultantes pueden ser subidos a PyPI mediante el módulo twine y el comando python -m twine upload .\dist*.

3.5 Revisión de estilo

Se procedió a la estandarización de nombres de argumentos (por ejemplo, nombres de el/los archivos que una función perteneciente a los módulos necesitará, dirección en la que se escribe

el resultado, etc), funciones principales y otros. Sin embargo, esta estandarización se aplicó de forma parcial debido a que crear un consenso no era prioridad para este trabajo.

El código todavía presenta mezclas de estándares en nombres de funciones, módulos, clases, argumentos y variables.

Se reemplazaron todas las instancias de indentación que usaban tabulación en vez de cuatro espacios.

3.6 Mejora de módulo de visualización

El módulo de visualización se mejoró para aceptar argumentos desde la línea de comando, permitiendo especificar archivos compatibles que se abrirán automáticamente al iniciar el módulo. La manipulación original de la cámara en el módulo se basaba en tres parámetros: distancia de la cámara, centro de foco y rotación de la cámara. Este último parámetro modificaba de manera no intuitiva la rotación de la cámara en el eje de visualización, donde una rotación de 180° resultaba en la misma imagen, pero invertida. Para solucionar esto, la rotación de la cámara ahora emplea coordenadas esféricas, y es necesario mantener presionada la rueda del ratón mientras se mueve horizontalmente para ajustar la rotación en dicho eje.

Se eliminaron varias opciones de prueba del menú de visualización, y se añadió una nueva para alternar entre bounding boxes visibles e invisibles, una función que antes solo estaba disponible mediante una combinación de teclas. Además, se incorporó una opción en la lista de objetos de visualización para mover la cámara hacia el objeto seleccionado. También se corrigió un error en el cambio de color de los mallados y se implementó este sistema de cambio de color en las fibras.

En la ventana de configuración para la segmentación por ROIs, se modificó el campo de lógica de intersección para verificar la validez de la lógica no sólo al presionar “Enter”, sino también al modificar cualquier parte de la lógica. Adicionalmente, ahora se proporcionan explicaciones de los errores en la lógica y se restaura la última lógica válida.

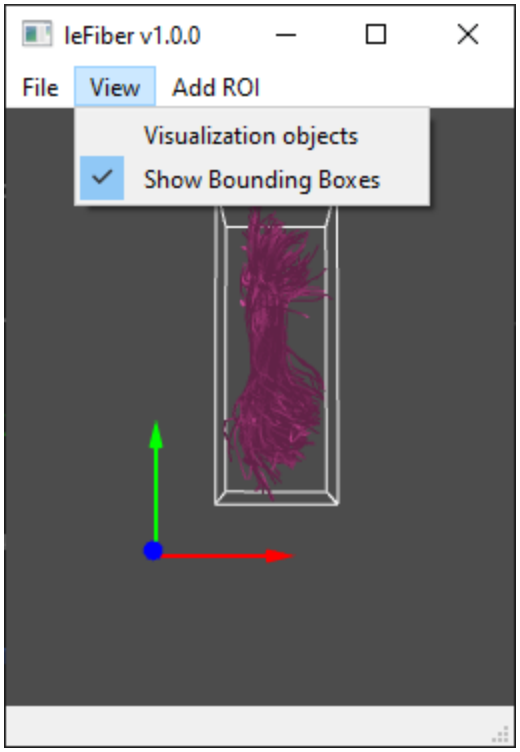


Figura 4 Opción para mostrar los límites de los objetos.

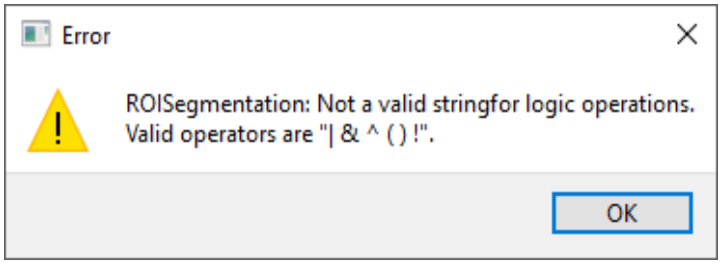


Figura 5: Error explicando caracteres permitidos.

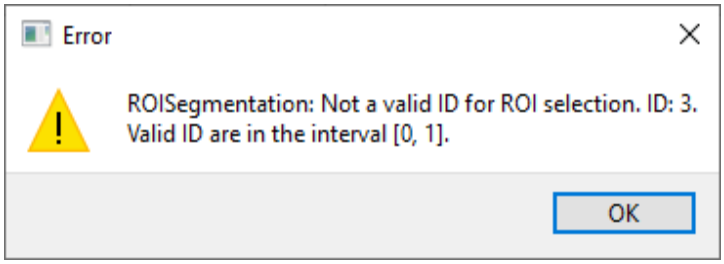


Figura 6: Error explicando números de ID permitidos.

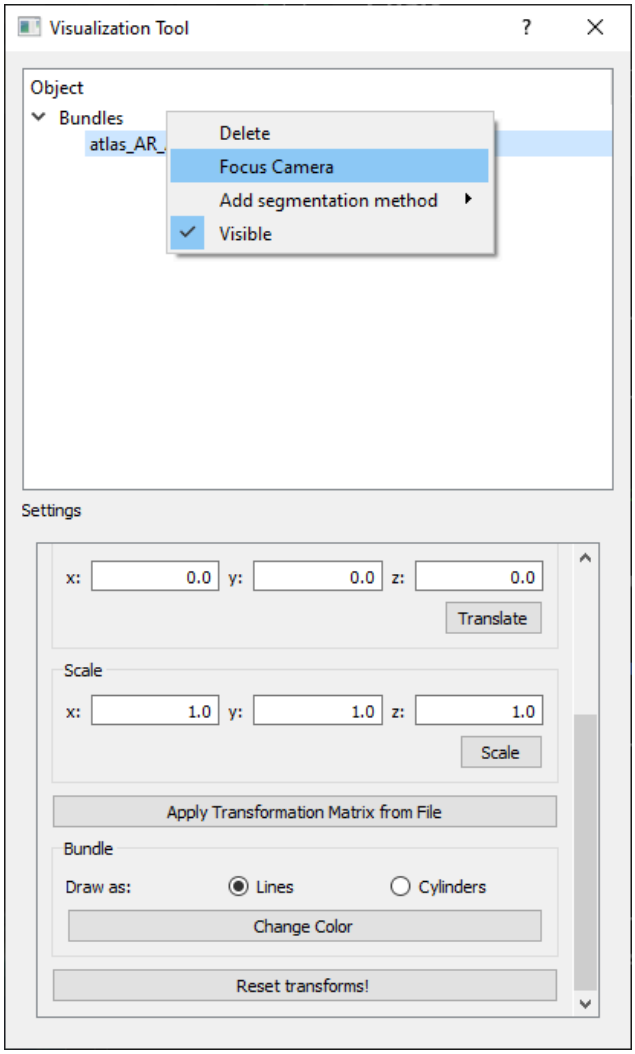


Figura 7: Opción en el menú contextual para enfocar la visualización en el objeto seleccionado.

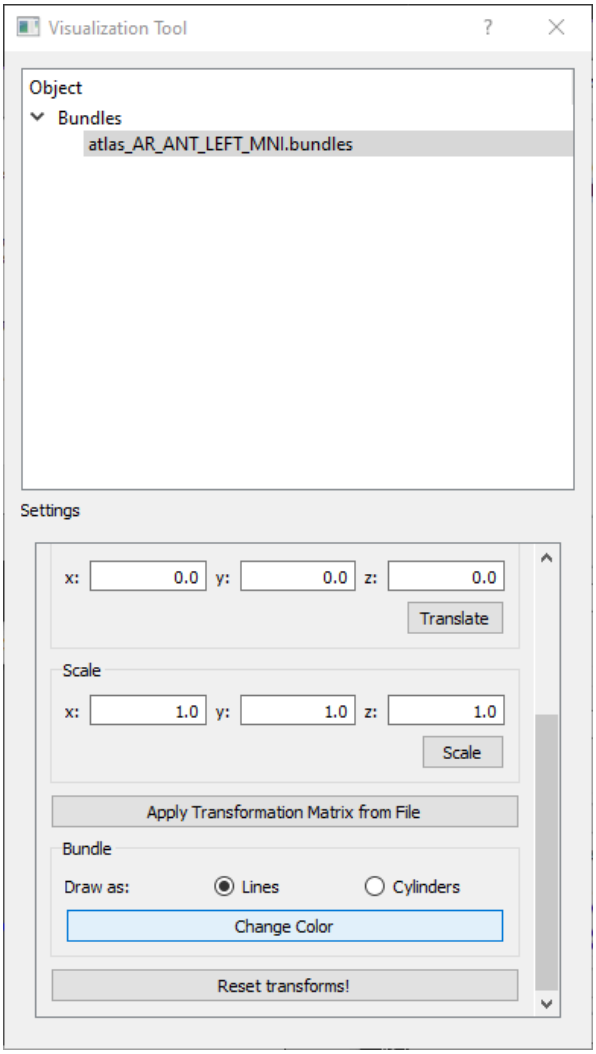


Figura 8: Botón para cambiar el color de las fibras.

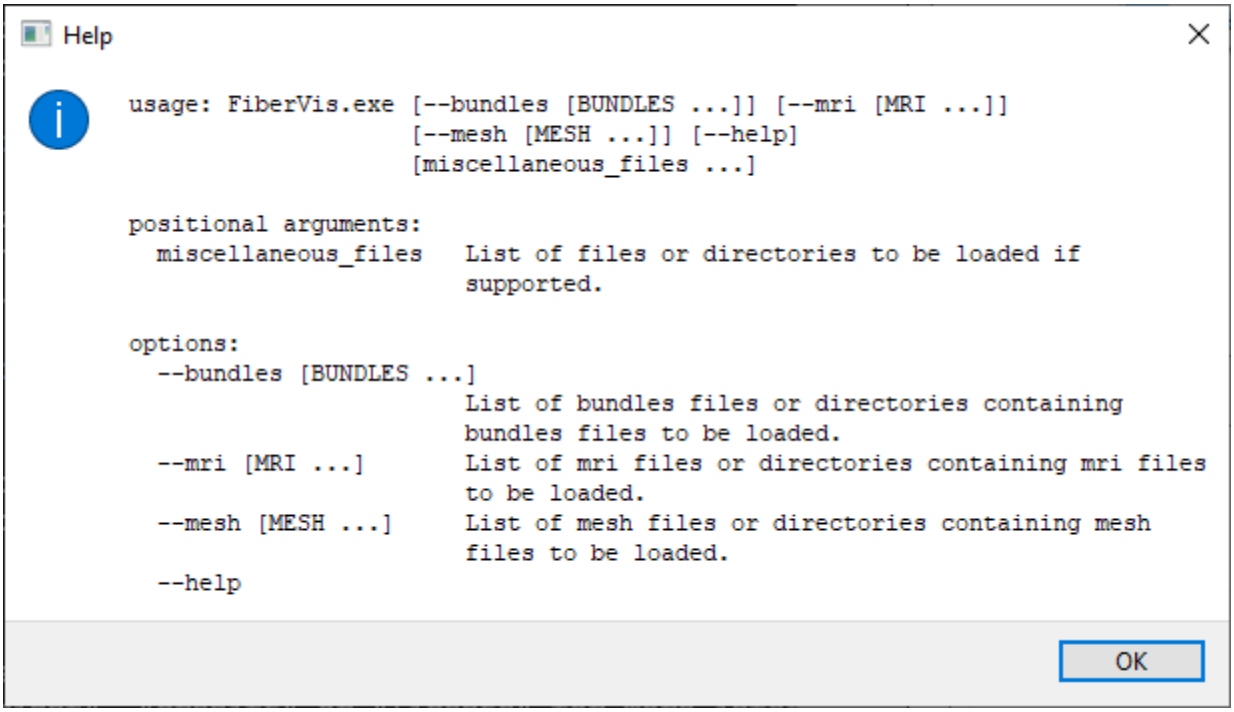


Figura 9: Mensaje de ayuda para ejecutar FiberVis con archivos a visualizar como argumentos.

4 ESTADO FINAL

En este momento el paquete puede ser instalado en su formato source en Windows y Linux, si los compiladores apropiados están presentes y configurados. Para Windows y Linux estos serían respectivamente Microsoft C++ Build Tools y gcc en formato compilado (actualmente solo se ha subido una versión compilada de Windows con Python 3.10, versiones compiladas para Linux tienen menos prioridad, ya que la mayoría de las distribuciones tienen las herramientas necesarias para compilar instaladas por defecto).

El módulo de visualización está funcionando en Windows y Linux, este es además instalado como aplicación gráfica, accesible por el ejecutable FiberVis.

Planes para la estandarización de docstrings y pistas de tipo están en progreso, con miras a eventualmente adoptar apropiadamente un generador de documentación (considerando no generar la documentación solo una vez, si no que establecer un procedimiento claro para actualizar y regenerar la documentación) en formato accesible.

La mayoría de los módulos todavía dependen de versiones locales de funciones comunes. Para el caso de la función de “resampling”, que toma fibras con números variables de puntos y las manipula para obtener fibras de un único número de puntos, esta funcionalidad está implementada en “slicefibers.c”, con cada módulo requiriendo “resampling” teniendo su propia versión de este archivo.

Ahora solo existe una única versión de slicefibers.c en el módulo “phybers.utils.sampling“ que expone la función slice_fibers, que es importada por los módulos que previamente tenían su propia versión.

En síntesis, los aspectos más importantes que fueron incorporados al trabajo respecto a la versión original son:

1. Reestructuración del paquete:
    a. Uso de importaciones relativas.
    b. Inclusión de archivos __init__.py.
    c. Eliminación de from paquete import *.
2. Mejoras en la compatibilidad multiplataforma:
    a. Adaptación del código para funcionar en Windows.
    b. Pruebas de compatibilidad entre Unix y Windows.
3. Integración de módulos en Cython:
    a. Implementación de adaptadores en Cython.
    b. Desempaquetado de argumentos de funciones principales.
4. Depuración exhaustiva de módulos:

    a. Resolución de problemas en el traspaso de datos.
    b. Ajustes en la gestión de memoria.
5. Optimización de la visualización:
    a. Mejora del módulo de visualización.
    b. Implementación de coordenadas esféricas para la rotación de la cámara.
6. Estandarización del código:
    a. Reemplazo de indentación por cuatro espacios.
    b. Estandarización parcial de nombres de argumentos y funciones.
7. Compilación y distribución:
    a. Inclusión de argumentos necesarios para compiladores en setup.py.
    b. Enumeración de archivos en MANIFEST.in.
    c. Declaración de herramientas requeridas en pyproject.toml.
8. Centralización de funciones comunes:
    a. Unificación de la función de “resampling” en un único módulo.

4.1 Objetivos no realizados

Considerando la propuesta inicial, los objetivos que no fueron realizados:

- Implementación del módulo de parcelación, este objetivo no se alcanzó debido a limitaciones de tiempo.
- Documentación, organización y mejora en legibilidad del código original. Aunque ciertas discrepancias de estilo fueron removidas, no se llegó a un consenso para el nombramiento de funciones, parámetros u otros. En el caso del código en “C/C++”, se intentó eliminar el código no usado, pero todavía se puede hacer más para mejorar la legibilidad.

5 TRABAJO FUTURO

Aunque el módulo de visualización es parcialmente funcional en las plataformas evaluadas, se considera que hay varias funcionalidades que no han sido debidamente probadas. Esto requiere de un periodo de prueba extenso debido a que las pruebas son lentas de conducir.

Más allá de funcionalidad básica, este módulo podría mejorarse con adiciones y modificaciones como:

- Mejor control de cámara e interacción con el mouse.
- Adición de modos de acceder al visualizador, como abrir y cargar contenido de manera programática.
- Acceso más intuitivo a funcionalidades, como son la generación de segmentación con ROIs que requieren el ingreso manual de operaciones de set para su uso.
- Mejor personalización de la visualización, como modificar el color de objetos o cambiar gráficos de fondo.

Aunque es posible usar datos conocidos para chequear la funcionalidad de los módulos, podría ser útil implementar un generador de datos artificiales para la evaluación automática del rendimiento y eficacia de los algoritmos. Esto requeriría el uso de datos ligeros a partir de los que se puedan generar una cantidad mayor de datos sintéticos o el estudio a profundidad del comportamiento de los datos reales para generar datos no necesariamente comparables con un cerebro real, pero con complejidad similar.

Otra posible mejora al funcionamiento interno de Phybers es la implementación de un tipo extendido de Fibras o clase “cdef” en cython para tener una representación accesible directamente desde Python de las fibras y clúster que también sea internamente compatible con las estructuras en C de estos. Lo cual haría posible el traspaso de datos entre el código C y el código Python sin necesidad de conversión o copiado.

Adicionalmente, un beneficio de lo anterior es una mejor organización del código, implementando varias de las funciones de lectura, escritura u otros como métodos de esta clase.

6 CONCLUSIONES

Este proyecto muestra el mejoramiento de la biblioteca Phybers, destinada a la investigación neurocientífica. A través de la integración efectiva de módulos en Cython y la adaptación para compatibilidad con sistemas operativos Windows y Linux, se ha reforzado la infraestructura técnica de la biblioteca para hacerla más accesible y funcional para los investigadores.

El trabajo realizado se ha enfocado en la estabilización de la base funcional del paquete antes de incorporar nuevas características, haciendo que las funcionalidades existentes fueran robustas. La reestructuración de la importación de módulos y la adecuación del código a las normativas de Python han mejorado significativamente la arquitectura del paquete, lo que facilita su uso y extensión futura.

La implementación de módulos en la biblioteca para la incorporación de módulos existentes de FFClust, HClust, y Segmentación, junto con mejoras en la visualización y el manejo de datos, ha potenciado la capacidad de Phybers para procesar y analizar complejas redes neuronales. Además, la depuración exhaustiva y la prueba de los módulos han garantizado que el paquete funcione correctamente en diversas plataformas, superando los desafíos presentados por las diferencias en los sistemas operativos y entornos de desarrollo. La capacidad de instalar y ejecutar el paquete tanto en entornos de código fuente como compilados ofrece a los usuarios una flexibilidad significativa, adaptándose a sus necesidades específicas.

Mientras que algunos objetivos como la implementación del módulo de parcelación aún están pendientes, el progreso realizado establece una base sólida para futuros desarrollos. La planificación para la estandarización de docstrings y pistas de tipo sigue en curso, lo que promete mejorar aún más la legibilidad y mantenibilidad del código.

BIBLIOGRAFÍA

[1]Fidel Ramón Romero, Armando Mansilla Olivares, Amelia Rivera Cruz. Las Neurociencias. Historia from NEUROFISIOLOGIA Para estudiantes de Medicina

[2] Bihan, D. L. (2014). Diffusion MRI: what water tells us about the brain. *EMBO Molecular Medicine*, *6*(5), 569-573. from https://doi.org/10.1002/emmm.201404055\

[3] Mueller BA, Lim KO, Hemmy L, Camchong J. Diffusion MRI and its Role in Neuropsychology. Neuropsychol Rev. 2015 Sep;25(3):250-71. doi: 10.1007/s11065-015-9291-z. Epub 2015 Aug 9. PMID: 26255305; PMCID: PMC4807614.

[4] National Institute of Child Health and Human Development (NICHD). (s. f.). *NICHD en español*. Retrieved August 9, 2024 *¿Por qué los científicos deberían estudiar neurociencia?* (2019, 17 octubre). from https://espanol.nichd.nih.gov/. https://espanol.nichd.nih.gov/salud/temas/neuro/informacion/estudiar

[5] Zhang, F., Xue, T., Cai, W., Rathi, Y., Westin, C., & O'Donnell, L. J. (2022, 5 julio). *TractoFormer: A Novel Fiber-level Whole Brain Tractography Analysis Framework Using Spectral Embedding and Vision Transformers*. arXiv.org.

[6] Xue, E., Zhang, F., Zhang, C., Chen, Y., Song, Y., Golby, A. J., Makris, N., Rathi, Y., Cai, W., & O'Donnell, L. J. (2023, 23 January) *Superficial White Matter Analysis: An Efficient Point-cloud-based Deep Learning Framework with Supervised Contrastive Learning for Consistent Tractography Parcellation across Populations and dMRI Acquisitions*. (s. f.). Ar5iv.

[7] Rodríguez, L. L. G., Osorio, I., G, A. C., Larzabal, H. H., Román, C., Poupon, C., Mangin, J., Hernández, C., & Guevara, P. (2024). Phybers: a package for brain tractography analysis. Frontiers In Neuroscience, 18.

[8] Antonio, S. I. M., & De Ingeniería Informática, U. D. (2014, 1 febrero). *Métodos para la Clasificación Automática de Imágenes de Resonancia Magnética del Cerebro*. Universidad Autónoma de Madrid.

[9] Slezak, D. F., Dorr, F., Varela, F. J., Alessandro, L., Bruno, V., & Farez, M. (2017). Inteligencia artificial y neurología: la revolución al acecho. Documento de posición. *Neurología Argentina*, *9*(2), 134-136.

[10] *Cython: C-Extensions for Python*. (s. f.). from https://cython.org/

[11] *Controlling files in the distribution - setuptools 69.5.1.post20240502 documentation*. (s. f.). from https://setuptools.pypa.io/en/latest/userguide/miscellaneous.html

[12] *Writing your pyproject.toml - Python Packaging User Guide*. (s. f.). from https://packaging.python.org/en/latest/guides/writing-pyproject-toml/

[13] Dobragab. (s. f.). *GitHub - dobragab/CWrapper: Template class for wrapping C resources in RAII*. GitHub. from https://github.com/dobragab/CWrapper

[14] GeeksforGeeks. (2022, 9 diciembre). *What is setup.py in Python?* GeeksforGeeks. https://www.geeksforgeeks.org/what-is-setup-py-in-python/

[15] Hall, P. (2023, August 16). *clustering in machine learning*. Enterprise AI. from https://www.techtarget.com/searchenterpriseai/definition/clustering-in-machine-learning

FIGURAS

[1] Figura 1: Zhang, F. (2021, abril). *ResearchGate*. https://www.researchgate.net/publication/351095296_Quantitative_mapping_of_the_brain's_structural_connectivity_using_diffusion_MRI_tractography_a_review

ANEXOS

[A] Tabla 1 de comparación de softwares usados para el estudio de dMRIs proveniente del paquete phybers [7].

Software	Lenguaje	OS	Distribution license	dMRI format	Tractography format
BrainSUITE	C++, Matlab	Multi.	Open-source	DICOM, NIfTI, Analyze	TRK
Camino	Java	Linux MacOs	Open-source	DICOM, NIfTI	VTK
Diffusion toolkit	C++	Multi.	Open-source	DICOM, NIfTI, Analyze	TRK
ExploreDTI	Matlab	Multi.	Non-commercial package	DICOM, NIfTI Analyze, Matlab formats	MAT
FSL	C++/Unix	Linux MacOs Windows	Non-commercial package	NIfTI	NIfTI
MRtrix	C++, OpenGL	Linux MacOs	open-source	DICOM, Analize NIfTI, MGH MRtrix formats	TCK
FreeSurfer	C/C++, Python, Matlab	Linux	Open-source	DICOM, Analyze NIfTI, MINC	-
DSI Studio	C++	Multi.	Open-source	DICOM, NIfTI	TRK
Dipy	Pyhton, Cython	Multi.	Open-source	Analyze, NIfTI, DICOM	TCK, TRK
DiffusionKit	C/C++	Windows Linux	Freely available	DICOM, NIfTI	TRK
SlicerDMRI	C++, Python	Multi.	Open-source	DICOM, Analyze, NIfTI, nrrd/nhdr	VTK

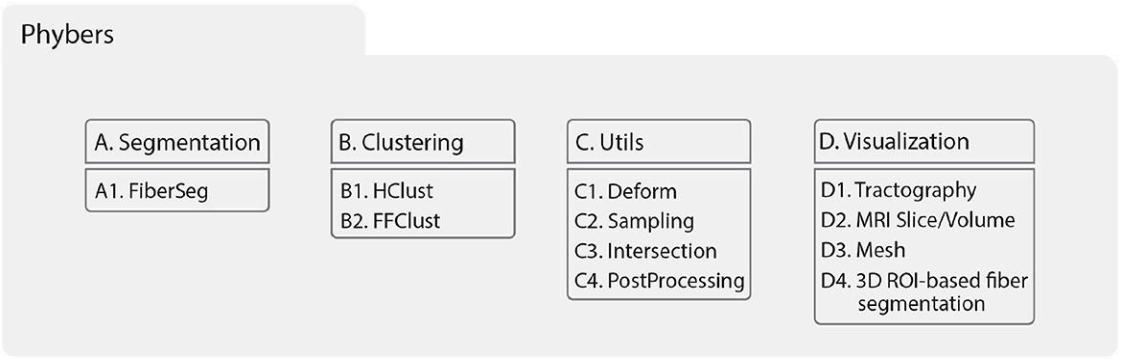
The order of the columns is as follows: Software, Lenguaje, OS, Distribution License, dMRI Format, and Tractography Format. OpenGL, Open graphics library; Multi., Multiplatform; NIfTI, Neuroimaging informatics technology initiative; DICOM, Digital imaging and communications in Medicine; MGH, FreeSurfer format; MINC, FreeSurfer format; VTK, Visualization ToolKit; TRK, Track File; Analyze, Image data format; TCK, Tracks file format; and MAT, Matlab file.

[B] Tabla 2 de comparación de softwares usados para el estudio de dMRIs proveniente del paquete phybers [7].

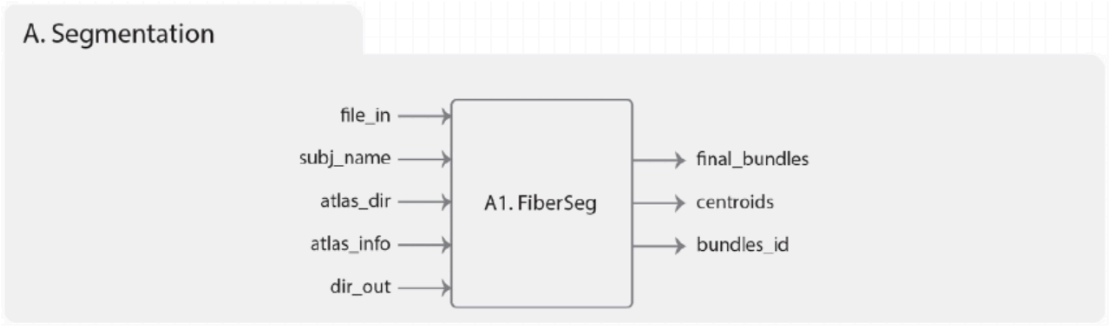
Software	DW Model Reconst.	Fiber Track.	Fiber Clustering	Bundle Segment.	Visual-ization	Fiber Measur.
BrainSUITE	DTI	Det.	-	-	Slice/Volume, DW model, tractography	-
Camino	DTI/ multifiber HARDI, QBall, PASMRI	Det. Prob.	-	-	Slice/Volume, DW model, tractography	-
Diffusion toolkit	DTI, DSI, QBI	Det.	-	-	Uses TrackVis	-
ExploreDTI	DTI, QBI, CSD	Det. Prob.	-	Using ROIs	Slice/Volume, DW model, tractography	Mean length
FSL	DTI	Prob.	-	-	Slice/Volume, Meshes	-
MRtrix	DTI, Single-tissue CSD, Multi-tissue CSD	Det. Prob.	-	-	Slice/Volume, DW model, tractography	-
FreeSurfer	TRACULA	Prob.	Anato-micCuts	TRACULA	Slice/Volume, Meshes	-
DSI Studio	DTI, DSI, QBI	Det.	-	Using ROIs	Slice/Volume, DW model, Tractography, Meshes	Count, mean length
Dipy	DTI, DSI, QBI, CSD	Det. Prob.	Quick-Bundles	Reco-Bundles	Slice/Volume, DW model, Tractography, Meshes	Count, mean length
DiffusionKit	DTI, CSD, dec. -based SPFI	Det.	-	-	Slice/Volume, DW model, Tractography	-
SlicerDMRI	DTI, Multi-tensor UKF	Det.	-	Using ROIs	Slice/Volume, DW model, Tractography	Points numbers, count, mean length

The order of the columns is as follows: Software, DW (diffusion-weighted) Model Reconstruction, Fiber Tracking, Fiber Clustering, Bundle Segmentation, Visualization, and Fiber Measures. DTI, Diffusion tensor imaging; HARDI, High angular resolution diffusion Imaging; DSI, Diffusion spectrum imaging; QBI, Q-Ball imaging; CSD, Constrained spherical deconvolution; SPFI, Spherical polar Fourier imaging; TRACULA, TRActs Constrained by UnderLying Anatomy; Det., Deterministic; Prob., Probabilistic; ROIs, Regions of interest; QuickBundles, (Garyfallidis et al., 2012); AnatomicCuts, (Siless et al., 2018) RecoBundles (Garyfallidis et al., 2018); DW, Diffusion weighted; and UKF, unscented Kalman filter (Malcolm et al., 2010).

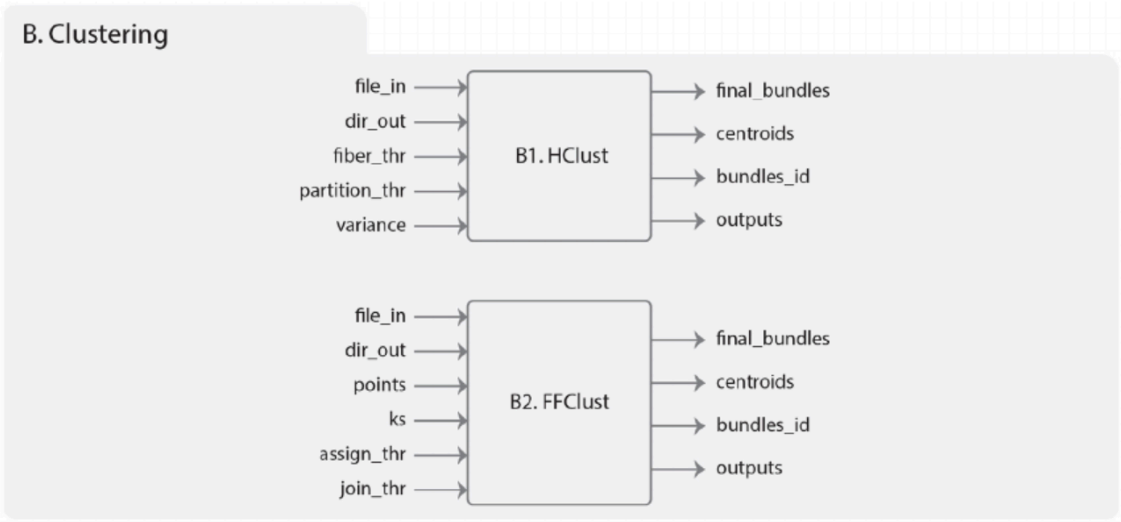
[C] Estructura jerárquica de la biblioteca separada en módulos de segmentación, clustering, utils y visualización provenientes del paper de phybers [7].



[D] Diagrama de representación del módulo de segmentación del paper de phybers [7].



[E] Diagrama de representación del módulo de clustering del paper de phybers [7].



[F] Diagrama de representación del módulo de Utils del paper de phybers [7].