UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Integración de programas de análisis de estructuras macromoleculares para biología estructural, modelos atómicos, y cribado virtual de fármacos

Autor: Martín Salinas Antón
Tutor: Carlos Oscar Sorzano Sánchez
Ponente: Roberto Marabini Ruiz

Mayo 2022

Resumen

En este Trabajo de Fin de Grado, se plantea realizar el desarrollo de dos protocolos del framework Scipion, consistiendo éstos en la integración de tres diferentes programas, desarrollados por el laboratorio de la Universidad de Purdue, Kiharalab, en dicha plataforma.

Para llevar a cabo la tarea mencionada, en primer lugar, se analizará cada uno de los programas por separado, se probarán, y corregirán errores o implementarán mejoras, con el fin de simplificar el proceso de su ejecución de cara a la integración en Scipion. Para cumplir este objetivo, habrá que trabajar en coordinación con los diferentes responsables de mantener ese software para poder integrar los cambios sugeridos en la rama principal de sus proyectos, lo que hará disponibles las mejoras para todos los usuarios, a la vez que simplificará el proceso de instalación y ejecución dentro de Scipion.

Tras haber modificado el software como sea necesario y conocer su funcionamiento, se procederá a desarrollar el propio protocolo que integre cada programa, para lo que también será necesario adquirir ciertos conocimientos sobre la plataforma Scipion, su flujo de ejecución, y su arquitectura.

Por último, tras haber integrado correctamente todos los programas, se deberá realizar una serie de pruebas que aseguren que el funcionamiento del software dentro del protocolo es equivalente al de su versión autónoma, y que se controlan todos los posibles casos de error dentro del protocolo con el fin de facilitar el uso del mismo.

Palabras clave

Scipion, tomografía electrónica, microscopía electrónica, procesamiento de imagen, gestor de flujo, biología estructural, integración, Emap2sec, MainMast.

Abstract

The goal of this project is to develop two protocols for the framework named Scipion. Those protocols will consist of the integration of three different programs, developed by Purdue University's lab, Kiharalab, in said platform.

To complete the above-mentioned task, each of the programs will be analyzed and tested individually, and some error corrections or improvements might be applied to them to ensure proper functioning when integrated into Scipion. To accomplish such a goal, coordinated work with the people responsible for maintaining the software will be needed, in order to merge the suggested changes into the main branch, so they are available to every user while allowing a hassle-free installation and execution process within Scipion.

After having modified the software as needed and properly knowing how it works, the protocol development process to integrate the programs will begin, which will require a certain amount of knowledge regarding Scipion, its workflow, and its architecture.

Lastly, after having successfully integrated every software, a series of tests must be carried out to guarantee that, from the same inputs, proper and equivalent results are produced in both the standalone and integrated versions. Also, every edge case must be controlled within the protocol to ensure its ease of use.

Keywords

Scipion, electron tomography, electron microscopy, image processing, workflow manager, structural biology, integration, Emap2sec, MainMast.

Agradecimientos

En primer lugar, me gustaría agradecerle tanto a Carlos, tutor de mi proyecto, como a Dani, compañero de laboratorio, su tiempo y esfuerzo dedicados a ayudarme a encaminar mi trabajo y resolverme todas las dudas que he tenido durante su desarrollo. Gracias también al resto de compañeros del laboratorio, por hacer que cada vez que fuese me sintiese muy acogido.

También le quiero dar gracias a mis compañeros de grado, en los que me he apoyado numerosas veces a la hora de tratar de entender el temario de una asignatura, buscar apuntes o ejemplos de exámenes antiguos, y que ayudaron a generar un ambiente colaborativo en el grupo de WhatsApp de clase, donde, si tenías una duda, casi siempre había alguien con la respuesta dispuesto a ayudarte.

Gracias a mis amigos más cercanos dentro del grado, los que lo completaron, los que lo dejaron por el camino, y los que aún siguen en él, no solo por lo dicho en el párrafo anterior, de lo que también eran parte, si no por los momentos que hemos vivido juntos, las tardes de estudio en la biblioteca reunidos en una mesa con una pizarra en víspera de algún examen, donde el que mejor preparado lo llevase, generalmente Marcos, se lo explicaba a todos los demás para poder aprobar todos, por no dejar a nadie detrás. Por las escapadas a la cafetería y al césped, las noches de juegos por Discord, y por otros tantos momentos.

Gracias a aquellos profesores que hacían del esfuerzo de aprobar una asignatura algo más llevadero, tratando de transmitir su entusiasmo por el tema, entendiendo que también somos humanos, y que entregar 10 minutos pasada la fecha de entrega cuando es el quinto proyecto que entregamos durante esa semana no es motivo de penalización, porque bastante teníamos con necesitar quedarnos de madrugada varios días a la semana para cumplir los plazos. Gracias por estar siempre disponibles para cualquier duda que nos pudiese surgir, y aportarnos todos los materiales necesarios para el buen aprovechamiento de las clases.

Y en último lugar, gracias a ti, por considerar este proyecto en el que tanto tiempo y esfuerzo he invertido, lo suficientemente interesante como para merecer al menos un vistazo.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

1 Introducción

1.1 Motivación

En el ámbito de la tomografía electrónica, se utiliza gran cantidad y variedad de software con el objetivo de procesar las imágenes producidas por un microscopio electrónico a un modelo interpretable por los seres humanos. Para resolver este problema, hay muchos programas de desarrolladores independientes, principalmente laboratorios, que realizan diferentes tipos de procesamiento, o partes del proceso completo. Estos programas no se comunican entre ellos y carecen de interfaz.

Scipion es un framework y gestor de flujo de trabajo, que integra estos programas y los comunica, aportando una interfaz, un sistema de instalación y pruebas automáticas, a su vez simplificando el proceso de uso de gran parte del software utilizando en este campo gracias a la capa de abstracción que provee sobre el uso de los programas en su estado autónomo, que funcionan por medio de comandos de terminal y siguiendo cada uno unas indicaciones específicas de uso [1].

El framework Scipion, completamente de código abierto, pretende simplificar los computacionalmente costosos procesos de procesamiento de imágenes que se llevan a cabo en la tomografía electrónica, siendo de gran utilidad en sectores tanto de investigación como industriales, y con aplicaciones en campos como la medicina, biotecnología, y farmacología.

1.2 Objetivos

Durante el desarrollo de este proyecto, se integrarán tres programas del laboratorio Kiharalab en el framework Scipion: Emap2sec, Emap2sec+, y MainMast, y se incluirán en el plugin scipion-em-kiharalab, en el que corresponde todo software procedente del laboratorio Kiharalab. Los dos programas Emap2sec y Emap2sec+, tienen como objetivo predecir las estructuras secundarias de una proteína, como láminas beta y hélices alfa, entre otras, por medio del uso de redes neuronales profundas, a partir del modelo 3D de una muestra. En el caso de Emap2sec+, además, siendo capaz de predecir estructuras de ADN y ARN [2]. Por otra parte, MainMast, construye un modelo 3D de una proteína a partir de un mapa de microscopía electrónica, más comúnmente llamados EM maps, de una resolución a nivel prácticamente atómico [3].

La tarea consistirá en tres fases claramente diferenciadas entre sí, siendo necesario seguirlas en orden para conseguir un correcto desempeño de las mismas.

Análisis de instalación y uso previos

En primer lugar, se deberán instalar cada uno de los programas de manera externa sin vincularlos a Scipion de ninguna forma, y probar su funcionamiento con varios casos de ejemplo, así como analizar el procedimiento de instalación y uso de la plataforma Scipion, completando algún tutorial ya documentado que permita comprender a grandes rasgos el uso de la plataforma. En caso de contener el software algún error, o de necesitar alguna adaptación por las necesidades que pueda generar la integración de los programas en Scipion, será necesario ponerse en contacto con los desarrolladores o, en su defecto, encargados de mantener ese software, para solicitar los cambios necesarios, o la validación de los mismos enviados en una Pull Request al repositorio correspondiente. También será necesario contactar con ellos para preguntar cualquier duda acerca del funcionamiento o instalación del software que no esté documentado o desactualizado, para posteriormente añadir la documentación necesaria con el fin de ayudar a futuros usuarios.

Integración

Tras terminar la fase de análisis previo, comenzará la fase de integración del software en un protocolo. Dentro de esta fase, el primer paso será desarrollar un instalador para el protocolo, que permita automatizar su instalación dentro de Scipion.

Una vez el instalador esté completo y el plugin se pueda instalar y desinstalar con facilidad, se procederá a desarrollar el propio protocolo, diseñando los elementos de la interfaz gráfica con los que contará, y su flujo completo de ejecución.

Testing

Por último, tras haber integrado por completo el protocolo y ser éste completamente funcional, se procederá a realizar una serie de pruebas que determinen si el software en cuestión produce la misma salida dada una misma entrada para las versiones standalone y en protocolo de Scipion, ya que Scipion debería ser una capa de abstracción sobre ese software, sin modificar en absoluto su funcionamiento. También se incluirán en el protocolo unos tests automáticos que cualquier usuario podrá ejecutar tras instalar el plugin que garanticen que la instalación se ha completado satisfactoriamente y los protocolos que contiene están listos para ser utilizados.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1:** Motivación, objetivos, y organización de la memoria.
- **Capítulo 2:** Estado del arte. Introducción a la biología estructural y desarrollo de algunos conceptos básicos.
- **Capítulo 3:** Diseño. Análisis del framework Scipion, y de la arquitectura de un plugin y un protocolo. Fase de análisis del software a integrar. Proceso de instalación y uso. Correcciones aplicadas con motivo de mejora de las funcionalidades y la documentación.
- **Capítulo 4:** Desarrollo. Proceso de desarrollo del protocolo, así como del instalador automático del plugin para todos sus protocolos.
- **Capítulo 5:** Integración, pruebas y resultados. Proceso de prueba del protocolo y de su equivalencia al software original, así como el desarrollo de los tests automáticos que se incluirán con el plugin.
- **Capítulo 6:** Conclusiones y trabajo futuro.
- **Referencias.**
- **Glosario.**
- **Anexos.**

2 Estado del arte

2.1 Introducción al procesado de imagen en el ámbito de la biología estructural

La biología estructural es la rama de la biología molecular que se centra en estudiar la estructura de las macromoléculas biológicas con el objetivo de entender su origen y la relación que estas estructuras tienen con la función biológica que desempeñan. Para poder llevar a cabo las tareas necesarias de investigación y obtención de datos a partir de una muestra, se usan diversas técnicas, como la criomicroscopía electrónica y la tomografía crioelectrónica, entre otras.

La criomicroscopía electrónica, (*cryo em*), es una forma de la microscopía electrónica en la que la muestra se encuentra a temperaturas criogénicas, con el fin de preservar la muestra.

Una proteína es una macromolécula compuesta por una cadena de aminoácidos plegadas de manera específica, con funciones tanto estructurales como de iniciar y detener reacciones químicas. La información necesaria para crear las proteínas se almacena en el ADN. Por otra parte, un aminoácido es una molécula en cuyos extremos tiene un grupo amino, (NH_2), y un grupo carboxilo, (COOH). Existen 20 aminoácidos diferentes.

La estructura de una proteína sigue cuatro niveles de organización, de nivel de complejidad menor a mayor, apreciables en la figura 2.1.

En primer lugar, al nivel de complejidad más básico, se encuentra una secuencia de aminoácidos, que pueden interactuar entre sí por medio de enlaces débiles, como los puentes de hidrógeno, dando lugar a las llamadas estructuras secundarias.

Dos ejemplos de estructuras secundarias son las láminas beta, también llamadas hojas plegadas, cuya forma es plana, y las hélices alfa, con forma helicoidal. Dos o más estructuras secundarias pueden también unirse, formando una estructura terciaria. De la misma forma, una combinación de estructuras terciarias forma la llamada estructura cuaternaria, siendo éste el nivel más alto de complejidad.

Extraído de https://es.wikipedia.org/wiki/Estructura_de_las_prote%C3%ADnas

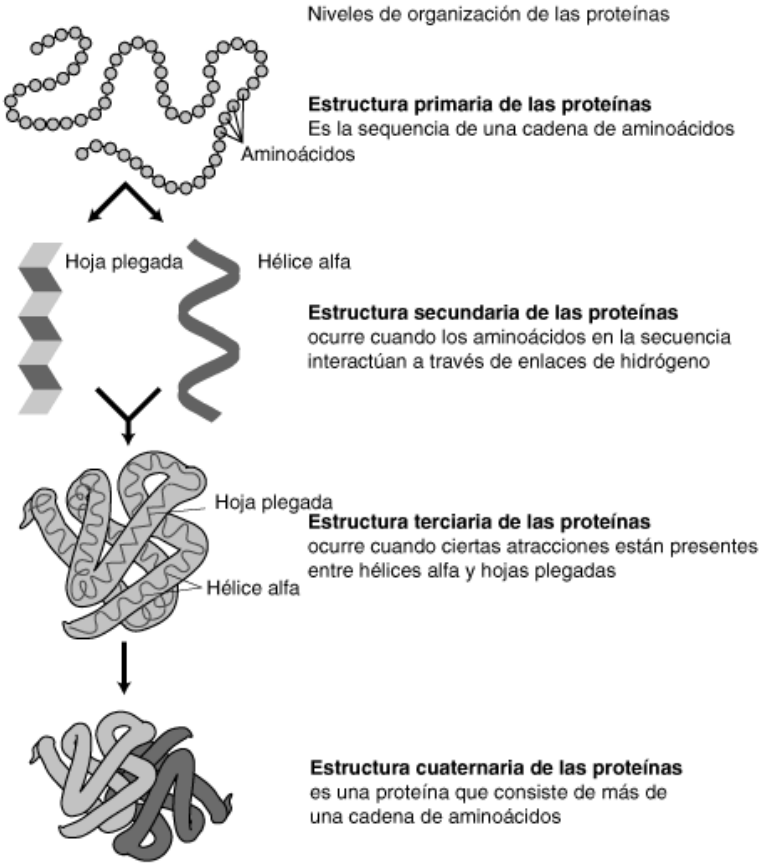

Figura 2.1: Niveles de organización de las proteínas.

2.2 Problemas de la biología estructural respecto al procesado de imagen

Aunque la cantidad de software desarrollado en este ámbito es bastante notable, y hay algoritmos capaces de resolver casi cualquiera de los procesos necesarios para obtener un modelo 3D a partir de una muestra, muy diversos según el método de obtención de dicha muestra, estos programas han sido desarrollados por desarrolladores independientes, lo que conlleva que dichos programas no sean capaces de comunicarse entre sí, a la vez que carecen de interfaz, lo que dificulta mucho el uso para los profesionales de la biología, que carecen de los conocimientos informáticos necesarios para poder trabajar en estas condiciones con facilidad, y por tanto, ralentiza su trabajo.

Uno de los problemas más claros es la fiabilidad. La existencia de estos programas como algo completamente independiente del resto de software hace que la fiabilidad de las diferentes etapas sea mucho menor, ya que errores en una etapa concreta debido al software utilizado en esa etapa, desencadenará errores en las siguientes etapas, y no hay una forma sencilla de localizar el origen de esos errores para poder corregirlos.

Otro problema a tener en cuenta es la trazabilidad del software. Al no ser los diferentes programas capaces de comunicarse entre sí, se pierde la trazabilidad de las etapas que sigue la muestra digital a lo largo de su procesamiento, dificultándolo así.

3 Diseño. Análisis del framework Scipion y del software a integrar.

3.1 Introducción al framework Scipion

Como ya se ha explicado en apartados anteriores, Scipion es un framework y gestor de flujo, desarrollado por el CSIC con el objetivo de solucionar una serie de problemas en el campo de la tomografía electrónica, explicados anteriormente. La interfaz de Scipion, de la que se analizarán algunos componentes a continuación, contiene pocos elementos, con el objetivo de mantener su simplicidad y con ello su facilidad de uso.

Para lanzar Scipion tras su instalación, será suficiente con ejecutar el comando *scipion3* en la terminal. La ventana que aparecerá será la mostrada en la figura 3.1.

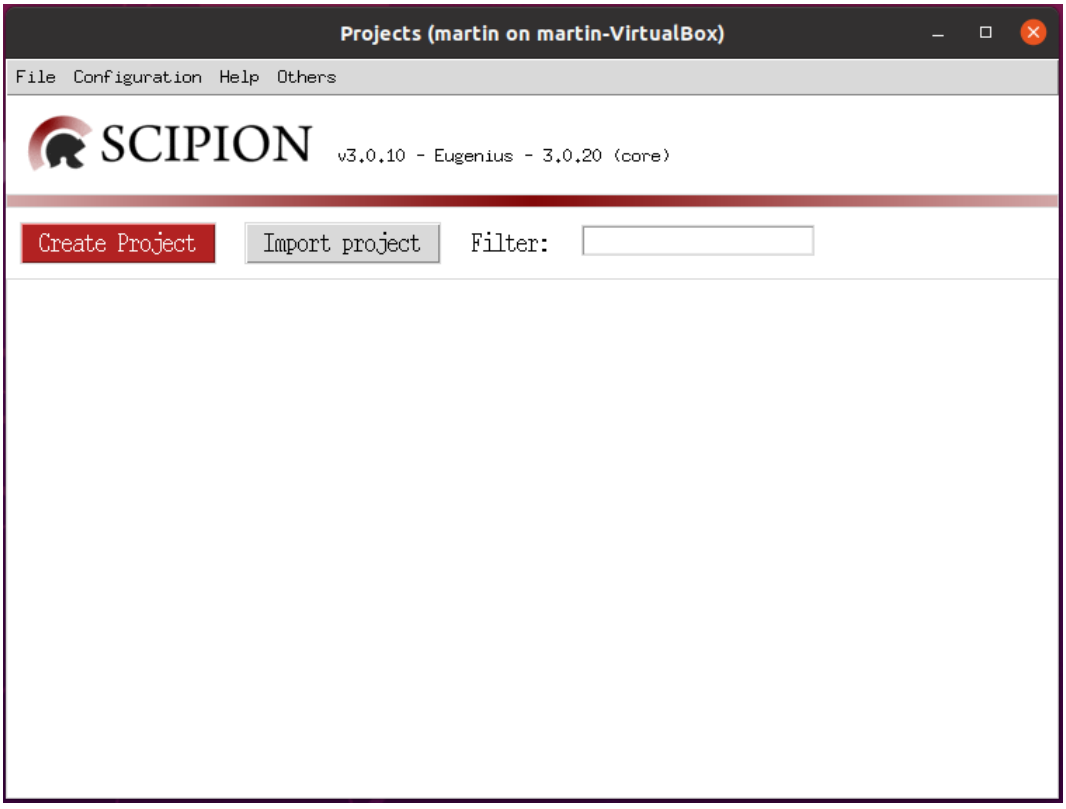

Figura 3.1: Ventana principal de Scipion.

En la ventana hay diferentes apartados que proveen al usuario de diferentes opciones. En primer lugar, están los botones de *Create Project* e *Import Project*. El primero permite crear un proyecto desde cero añadiendo un nombre, mientras que el segundo permite cargar un proyecto creado con anterioridad en otro equipo exportado como un archivo. La caja de texto con el nombre *Filter* permite filtrar entre los proyectos existentes según su nombre.

A continuación, en la parte superior de la ventana se encuentran las herramientas desplegables, repartidas en los siguientes apartados:

- File: Permite tanto abrir la carpeta de datos de Scipion, donde se guardan todos los archivos de protocolos de cada proyecto y los tests, como salir del framework.

- Configuration: Permite abrir los archivos de configuración de Scipion para varios aspectos, en concreto tres: General, Hosts, y Protocols.

- Help: Contiene un link a la página de ayuda de Scipion, a la vez que una ventana de información sobre la instalación de Scipion.

- Others: Contiene los elementos que no encajaban en ninguno de los apartados anteriores. Esta es una de las pestañas más importantes, ya que en ella se encuentra el *Plugin manager*, una interfaz desde la que instalar en Scipion todos los plugins necesarios para cumplir los objetivos de cualquier proyecto.

Tras crear y abrir un proyecto, la ventana que aparece es similar a la figura 3.2.

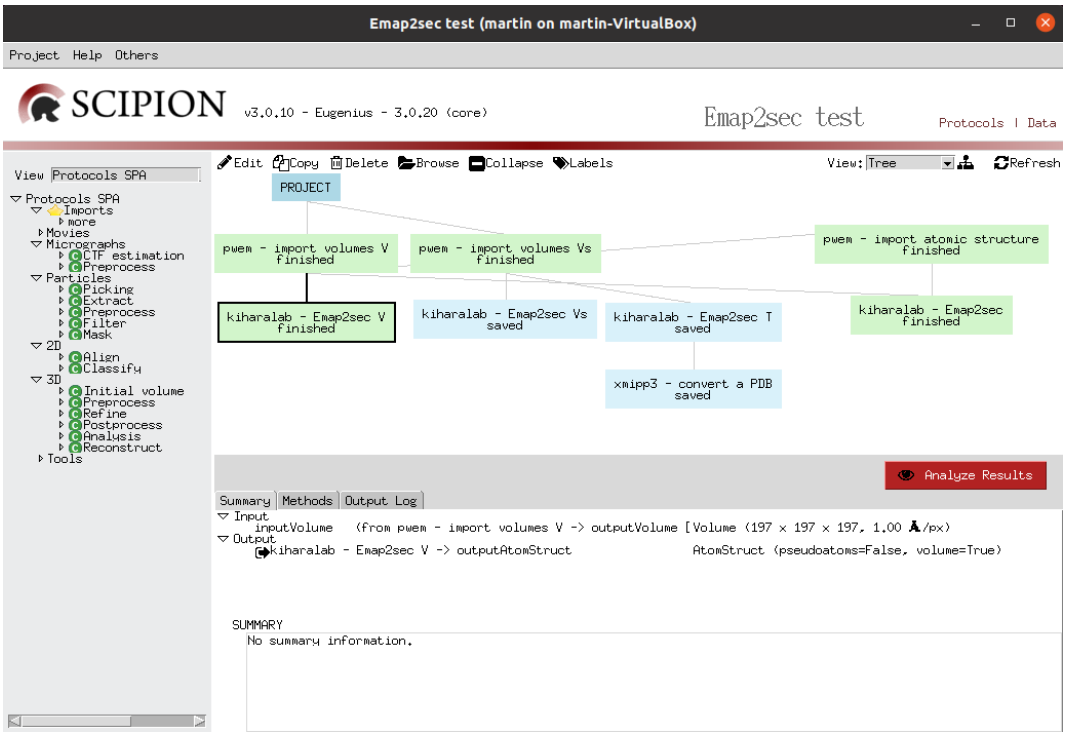

Figura 3.2: Ventana de proyecto.

En la ventana, se puede ver un espacio en blanco con una caja con la etiqueta *PROJECT*. Ahí es donde se irán colocando los protocolos cuando se vayan añadiendo, que tendrán forma de cajas, y lo harán conectándose entre sí de forma visual siempre que la

salida de uno se utilice como entrada en otro. A la izquierda, se encuentran los diferentes protocolos instalados en Scipion, agrupados en apartados según el tipo de problema que resuelven. Debajo del espacio donde se muestran como cajas los protocolos empleados en el proyecto, se encuentra, de haber algún protocolo actualmente seleccionado por el usuario, información acerca del mismo dividida en varias pestañas. En la pestaña *Summary* se puede encontrar el tipo de entrada y de salida del protocolo, más detallada si se ha ejecutado y existe información concreta acerca de ello. En la pestaña *Methods*, está el nombre de protocolo junto con la referencia a la publicación científica asociada a él, de haberla. Por último, en la pestaña *Output Log*, a su vez dividida entre *run.stdout*, *run.stderr*, y *schedule.log*, contiene la traza de ejecución del protocolo, posibles errores que surjan durante la misma, e información por terminal del planificador de tareas que coordina la ejecución de los protocolos.

Por otra parte, como se ha mencionado previamente, una herramienta importante de Scipion es su *Plugin manager*, lugar desde el que descargar e instalar los plugins de manera sencilla. Al abrirlo, la interfaz será algo parecida a la que se presenta en la figura 3.3.

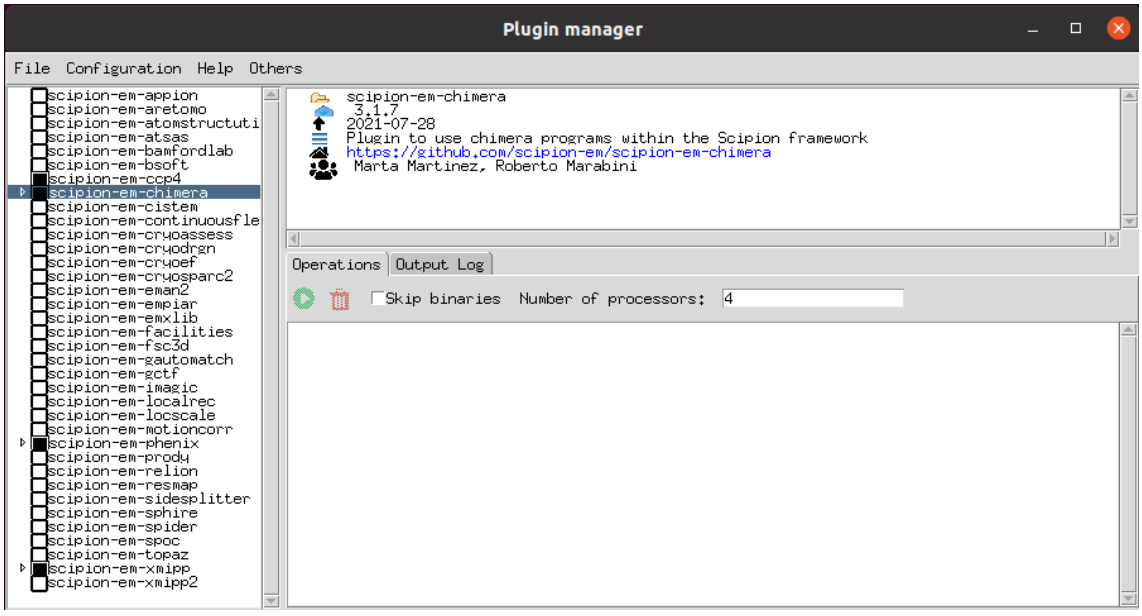

Figura 3.3: Ventana del *Plugin manager*.

A la izquierda se puede apreciar una lista de plugins disponibles que Scipion carga automáticamente desde los servidores, mostrándose como cajas rellenas los que ya se encuentren instalados. En el lado derecho de la ventana se encuentran dos paneles. El superior muestra la información del plugin seleccionado actualmente, mientras que el inferior está dividido en las pestañas *Operations*, con la cola de tareas pendientes o en ejecución, y *Output Log*, con la salida por terminal que van produciendo los comandos que se ejecutan durante las operaciones de la pestaña anterior. Para instalar un plugin, basta con marcar la caja, lo que añadirá la tarea de instalación a la cola. Para desinstalarlo, también basta con desmarcar un plugin que aparezca marcado. Finalmente, si se hace click en el

botón *Run*, con un icono circular verde con un triángulo en su interior, se ejecutan todas las operaciones de instalación y desinstalación añadidas a la cola.

3.1.1 Interfaz de los protocolos

Dentro de un proyecto abierto, se puede interactuar con cada uno de los protocolos, teniendo todos ellos una interfaz común, pero con campos diferentes según el protocolo. Un ejemplo de protocolo se muestra en la figura 3.4.

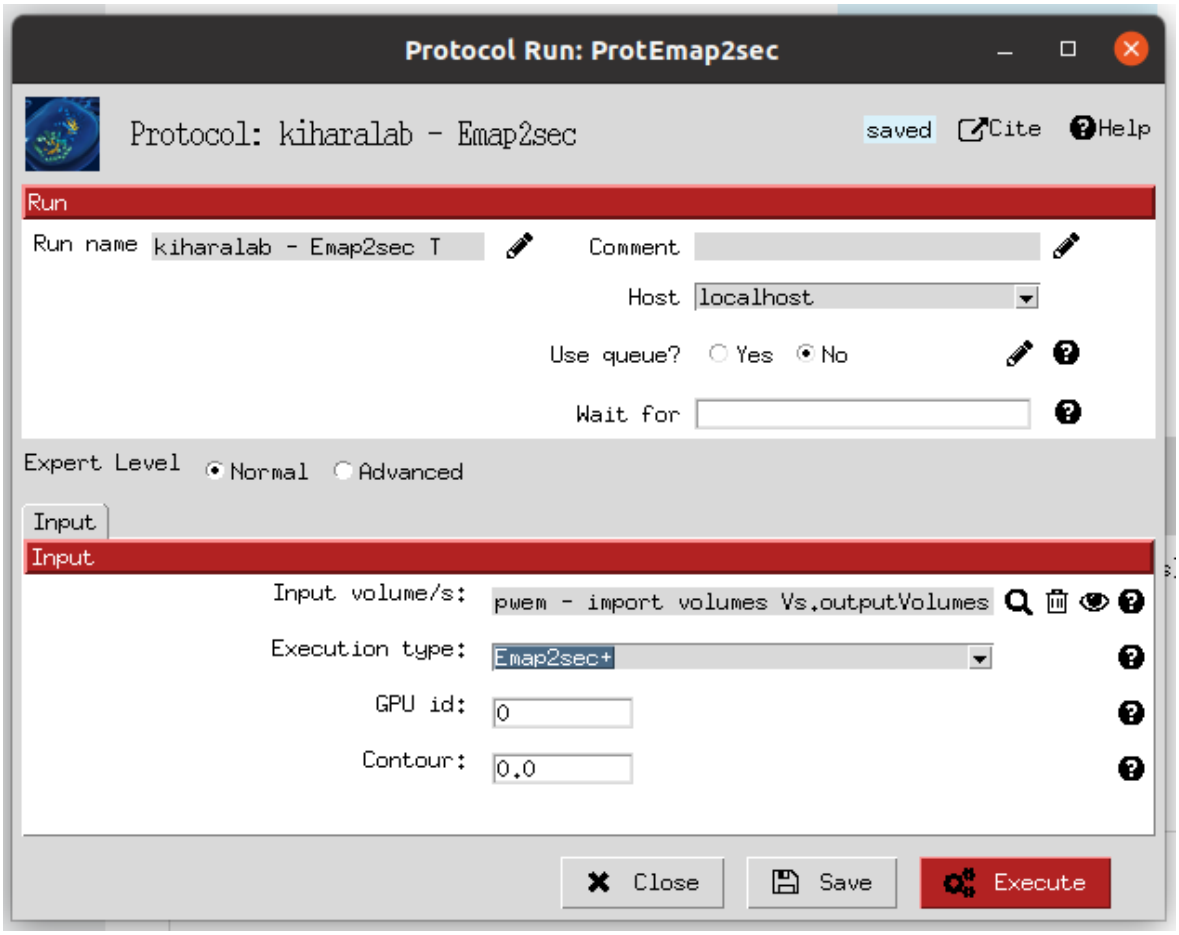

Figura 3.4: Interfaz de un protocolo.

Hay dos partes principales a analizar en esta interfaz. La parte superior es común a todos los protocolos, conteniendo campos como el nombre del mismo, un comentario opcional, dónde se ejecuta el protocolo, si se ejecuta en una cola, y si se espera a algún protocolo concreto a que termine para poder empezar la ejecución, especificando su identificador de protocolo. Por el contrario, la parte inferior contiene los campos específicos de cada protocolo. El programador define para cada uno su etiqueta, su tipo de dato, y sus posibles valores por defecto, entre otros. Todos los campos tienen a su lado un icono con una interrogación, que, de ser clicado, muestra en una pequeña ventana información de ayuda acerca de ese campo. Algunos campos tienen herramientas adicionales, como los archivos o directorios de entrada, o los tipos de datos definidos dentro de Scipion, que permiten explorar el sistema de archivos o ver los elementos del tipo especificado producidos

como output por otros protocolos. En el caso de los tipos de datos, se muestra también el icono de un ojo, lo que permite visualizar en una nueva ventana la muestra digital en el estado en el que esté en ese paso, ya sea en imágenes 2D o en un modelo 3D, según la situación.

Por último, en la zona inferior del protocolo se encuentran tres botones. El botón *Close* cierra la ventana del protocolo sin guardar los cambios, *Save* guarda los cambios sin cerrar la ventana, y *Execute* guarda los cambios, cierra la ventana, y añade el protocolo a la cola de ejecución, donde se ejecutará automáticamente en el momento que cumpla todos los requisitos necesarios para ello, como, por ejemplo, que estén disponibles todos los datos de entrada en el caso de que otro protocolo que los produzca aún se encuentre en ejecución.

Algunos campos, por motivos de limpieza de la interfaz y comodidad de uso, están ocultos por defecto, bajo el nivel *Expert Level: Advanced*, que se mostrarán si dicha opción es seleccionada, como se puede ver en la figura 3.5.

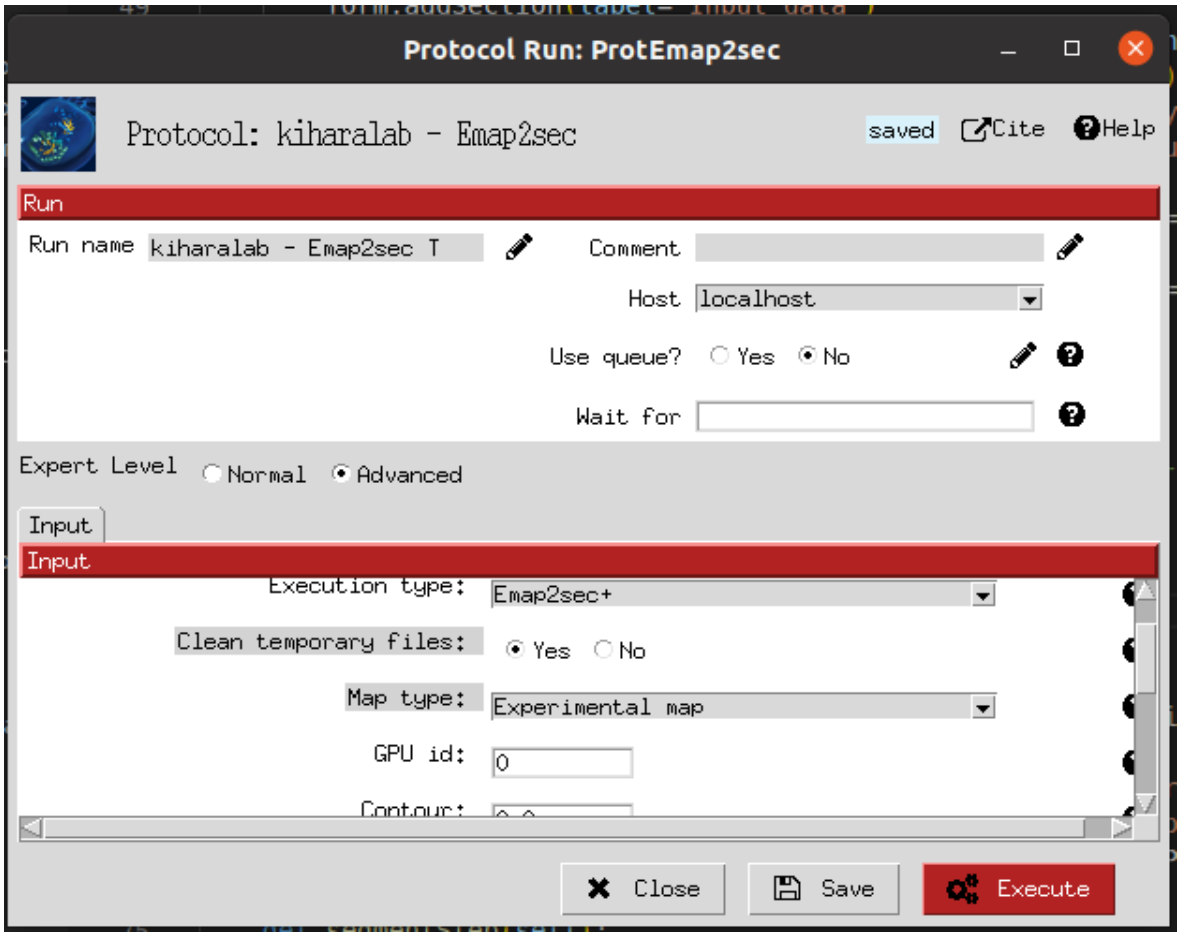

Figura 3.5: Interfaz de un protocolo en modo *Advanced*.

Además, Scipion permite separar los campos en pestañas diferentes si se cree conveniente, dejándolo siempre a elección del desarrollador, como se puede apreciar en la figura 3.6, perteneciente a otro protocolo.

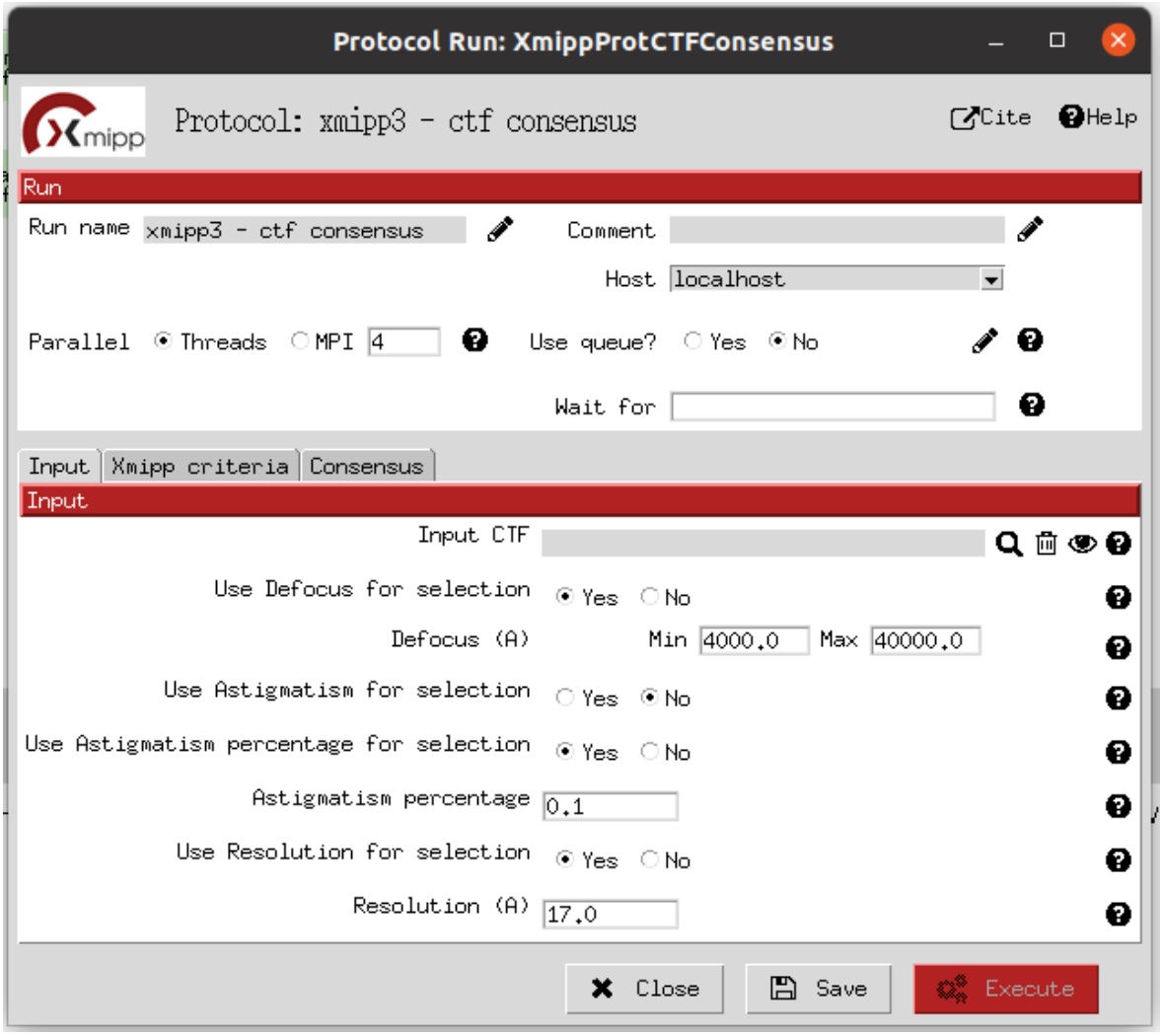

Figura 3.6: Interfaz de un protocolo con diferentes pestañas.

3.1.2 Arquitectura de los plugins

En Scipion, los plugins, y con ello también los protocolos, se desarrollan en Python. Todos los plugins siguen de la misma forma, una arquitectura de directorios específica, siendo algo similar a la figura 3.7.

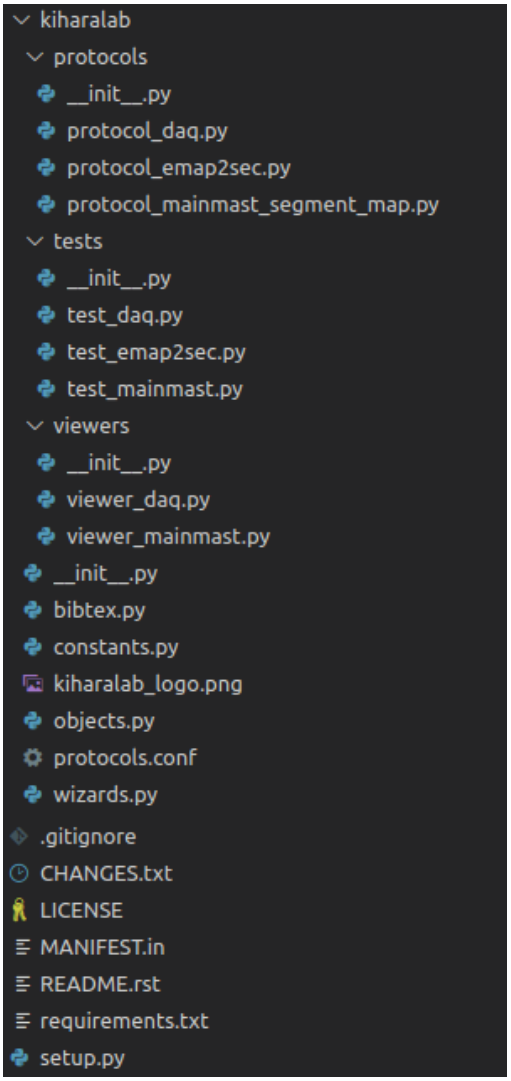

Figura 3.7: Árbol de directorios de un plugin.

En primer lugar, se pueden distinguir algunos archivos en el nivel principal. Los principales son *README.str*, para mostrar información importante sobre el plugin, sobre todo pensado para GitHub, *requirements.txt*, donde se indican otros plugins necesarios para ejecutar el plugin actual, o *setup.py*, donde se encuentra algo de información acerca del plugin. Además, se encuentra también una carpeta con el nombre del plugin, en cuyo interior se encuentra el resto de los elementos necesarios para el correcto funcionamiento del mismo: el instalador, los protocolos, los tests, y los *viewers*, de ser necesarios, además de algunos elementos de la configuración del plugin.

El instalador se encuentra en el archivo *__init__.py*, que a su vez utiliza variables definidas en el archivo *constants.py*. El procedimiento de instalación de los protocolos de cada plugin puede ser muy diferente, ya que el software utilizado por los protocolos es muy heterogéneo, existiendo protocolos que realizan llamadas tanto a scripts de Python propios, como a programas externos escritos en lenguajes como C/C++, Java, MATLAB, o Perl, entre otros. Sin embargo, hay algunos elementos comunes a todo instalador, concretamente dos funciones que deben estar definidas en todo instalador, y que se ejecutan siempre en el proceso de instalación. La primera se llama *_defineVariables*, y define algunas variables de Scipion y las almacena en el archivo de configuración *scipion.conf*. El nombre o valor de dichas variables se deja a elección del desarrollador. La segunda se llama *defineBinaries*, y es la que se encarga de realizar la instalación como tal. Su procedimiento es muy variado en diferentes protocolos, como se explica anteriormente, por lo que muchos desarrolladores optan por escribir funciones separadas para cada protocolo para posteriormente llamarlas dentro de esta función. No obstante, una instalación de cualquier software en Scipion, puede reducirse a la ejecución de una serie de comandos de terminal, cuyo contenido variará de protocolo a protocolo. El único elemento común a todas las instalaciones dentro de esta función es que, el último paso de la misma siempre debe ser llamar a la función *addPackage*, que ejecuta los comandos definidos previamente y crea el protocolo con el nombre deseado.

Adicionalmente, existen varios directorios. Uno de ellos, tiene como nombre *protocols*, donde se encuentra un archivo *__init__.py* que importa los protocolos, además de un archivo por cada protocolo, que será analizado próximamente. El siguiente directorio se llama *viewers*, y su función es albergar los viewers de los protocolos que necesiten uno. Un viewer es un fragmento de código que llama con determinados argumentos a un software de visualización de la muestra digital, que suele emplear otros plugins de Scipion como *chimera*, *xmipp*, o *pymol*. Por último, se encuentra el directorio *tests*, que contiene un archivo por cada protocolo con tests que realizan ejecuciones con casos de ejemplo y comprueban que los resultados sean correctos, con el fin de confirmar que la instalación se ha realizado correctamente y el plugin instalado está listo para su uso.

3.1.3 Arquitectura de los protocolos

En el interior del directorio *protocols*, se encuentran los protocolos definidos para el plugin. Éstos también siguen una estructura relativamente similar, salvo por el contenido de los comandos para su ejecución, que varían en gran medida dada la alta heterogeneidad del software de Scipion. Respecto a los elementos estructurales que son similares en todos los protocolos, se pueden encontrar tres.

En primer lugar, en el interior de la clase del protocolo, se definen algunas variables que Scipion utiliza para aportar información acerca del mismo. Ejemplos de este tipo de variables son *_label*, que define el nombre del protocolo, y *_possibleOutputs*, que

define las posibles salidas del protocolo sin que necesite haberse ejecutado previamente, de forma que otros protocolos puedan enlazarse a él y comenzar su ejecución automáticamente tan pronto como haya terminado la de su predecesor, lo que es especialmente útil si se necesita encadenar ejecuciones largas, ya que evita la necesidad de intervención humana a cada paso de la cadena.

A continuación, existe la función *_defineParams*, en la que se definen los campos de formulario que tendrá el protocolo, a través de los cuales se recolecta la información necesaria para generar los comandos de ejecución del software. Para ello, Scipion contiene una gran variedad de tipos de campos, además de diferentes argumentos opcionales que ayudan a desarrollar un formulario a medida de las necesidades del protocolo, como condiciones para mostrar uno o varios campos, grupos visuales de campos, o etiquetas de ayuda para mostrar información adicional, entre otros.

Por último, llegado el momento de la ejecución del protocolo, Scipion realiza una llamada a la función *_insertAllSteps*, definida a continuación de *_defineParams*. En ella, se introducen las funciones que van a componer la ejecución del protocolo, habitualmente divididas en tres fases: procesamiento de la entrada, ejecución principal del protocolo, y procesamiento de la salida. El procesamiento de la entrada es un paso opcional, ya que no todos los protocolos necesitan adaptar la entrada recibida, sobre todo si procede de otro protocolo que ya la ha convertido en un tipo de dato de Scipion. La ejecución principal del protocolo se encarga de, en primer lugar, recoger los valores del formulario, para posteriormente generar las cadenas de texto que conforman los comandos a emplear, y finalmente llaman a la función de ejecución del protocolo definida en el archivo *__init__.py* donde también se encuentra el instalador. Finalmente, el paso de procesamiento de la salida se encarga de importar el archivo o archivos generados por el software ejecutado en un tipo de dato reconocido por Scipion, para que posteriormente pueda ser empleado como entrada por otros protocolos.

3.2 Análisis del software a integrar

En este proyecto se ha propuesto analizar cada uno de los programas que posteriormente se integrarán como protocolos de Scipion, con el objetivo de entender su funcionamiento para poder desarrollar el protocolo adecuadamente, tanto en la creación del formulario, como en la recolección de sus valores y en la generación de los comandos de ejecución, como en el posterior procesado de la salida. Adicional y opcionalmente, se implementarán mejoras o correcciones al repositorio original de dichos programas de ser posible, con el fin de simplificar el proceso de ejecución y adaptar el software lo mejor posible a su uso dentro de Scipion sin perjudicar su usabilidad fuera de este framework.

En este proyecto, se integrarán tres programas en el mismo plugin, pero, dado que uno de ellos ya se había integrado con anterioridad en un plugin diferente y se ha probado su funcionamiento, únicamente será necesario analizar y opcionalmente modificar dos de los programas, y el tercero solo necesitará una migración de plugin.

3.2.1 Emap2sec

El primer software por analizar será Emap2sec. Este programa carece de interfaz, y funciona por medio de comandos de terminal. Su función es recibir como entrada un mapa de microscopía electrónica perteneciente a una macromolécula, o *EM-Map*, y, por medio de redes neuronales profundas, predecir las estructuras secundarias de la misma. Un mapa de microscopía electrónica es un conjunto de vóxeles, equivalente tridimensional del píxel, que forman modelo 3D de la muestra en el que no se diferencian las estructuras que la conforman.

Para completar esa tarea, es necesario ejecutar una serie de comandos que serán analizados a continuación. Cabe mencionar, que todos los comandos descritos a continuación deberán ejecutarse desde el directorio raíz del proyecto.

3.2.1.1 Generación de archivo trimmap

El primer paso es utilizar la entrada, que será un archivo con extensión *.mrc*, y generar a partir de él un archivo *trimmap*, un tipo de archivo que contiene valores normalizados de densidad de electrones en vóxeles. Para completar este paso, el comando que hay que ejecutar es el siguiente: ./**map2train_src/bin/map2train {volume file} -c {contour level} > {output trimmap file}**. En este comando, el primer parámetro es el EM-Map de entrada, -c se refiere al nivel de la isosuperficie para el cual se generan valores de densidad, cuyo valor debe ser especificado por el autor del archivo de volumen, y el resultado del mismo se vuelca en el archivo indicado. Hay otros parámetros opcionales que pueden ser empleados, cada uno con diferentes funciones, todos ellos explicados en la documentación del proyecto.

3.2.1.2 Generación opcional de archivo stride

El segundo paso es opcional, y se usa con el fin de realizar un benchmark sobre los modelos utilizados para procesar una muestra. Para cumplir este objetivo, se debe contar con un archivo de extensión *.pdb* correspondiente a la solución correcta de la muestra a analizar. Además, se debe emplear un software externo llamado *stride*, cuya guía de instalación se adjunta en los prerrequisitos del proyecto. Para ejecutar este paso, el comando a ejecutar será: **./stride -f{protein stride} {pdb file}**. El primer parámetro será el archivo de salida que producirá este comando, mientras que el segundo hace referencia al archivo *.pdb* solución.

3.2.1.3 Generación del dataset de entrada

Tras completar el primer, y opcionalmente, el segundo paso, se utiliza el archivo *trimmap* y, de existir, el archivo *stride*, para generar un archivo de tipo *dataset*, que será el que utilice la red neuronal para realizar su predicción. El comando para ejecutar este paso es **python data_generate/dataset.py {trimmap file} {stride file} {dataset filename} {protein id}**. En este comando, el primer argumento de entrada es el archivo *trimmap* generado previamente, el segundo, de naturaleza opcional, hace referencia al archivo *stride*, de existir, el tercero se trata del nombre de archivo que deberá tener la salida, y el último un identificador único de la proteína en cuestión, que, de ser una muestra digital ya existente, se encontrará en el banco de datos de donde se descargó inicialmente.

3.2.1.4 Ejecución de la red neuronal

Disponiendo del archivo *dataset*, el siguiente paso consiste en emplear la red neuronal profunda para realizar las predicciones deseadas sobre la muestra digital. El script de Python que maneja la ejecución recibe como entrada un archivo de texto, en cuyo contenido se encuentra, por cada línea del mismo, la ubicación de un archivo *dataset*, en el caso de que se quieran procesar varias muestras en lote. Para ello se ejecutan dos comandos. El primero genera el archivo de texto con las ubicaciones, mientras que el segundo, llama finalmente al script. Hay dos opciones para el primer comando, siendo cada una el caso en el que se tiene una o más muestras, respectivamente. Para la primera opción, el comando necesario será **echo {dataset file} > {dataset location filename}**, donde el primer argumento será el archivo *dataset*, y el último el nombre del archivo de ubicaciones. Para la segunda opción, el comando será **echo $'{dataset file 1}\n{dataset file 2}' > {dataset location filename}**, y se diferencia del comando anterior en que en este caso se puede introducir un número indefinido de archivos *dataset*, separados cada uno del carácter *\n*, con el fin de separarlos por un salto de línea en el archivo de destino. Tras ejecutar cualquiera de estos dos comandos, se llama al script con el comando **python emap2sec/Emap2sec.py {dataset location file}**, donde el único argumento de entrada hace referencia al archivo de ubicaciones de los *dataset*. Para ejecutar este paso, el script emplea unos modelos entrenados previamente cuya descarga se facilita desde la misma

guía de uso del software. Es importante que el directorio de los modelos se encuentre en la carpeta raíz del proyecto y tenga la estructura de directorios con la que figura en su lugar de descarga.

3.2.1.5 Generación de archivo solución

Finalmente, se empleará la salida producida por el paso anterior para producir el archivo *.pdb* que se genera como resultado del uso de este software. La ejecución anterior genera dos archivos de salida por cada *dataset* de entrada, que siempre se encontrarán en el directorio raíz del proyecto, y siempre se llamarán *outputP1_N* y *outputP2_N*, donde N será un número entero que corresponde al índice del archivo *dataset* en la lista de archivos. De estos dos archivos, el único que es de utilidad para el próximo paso es *outputP2_N*, y el comando necesario para generar el archivo solución será **Visual/Visual.pl {trimmap file} output_P2_N -p > {output filename}**, donde el primer argumento se corresponde al archivo *trimmap* generado en el primer paso, y el último es el nombre del archivo solución generado como salida.

3.2.1.6 Observaciones del software

Tras analizar este software y probarlo en sus diferentes modos de uso, se han encontrado algunos inconvenientes que serán corregidos.

El primero de ellos es la escasez de documentación de calidad, ya que, aunque este programa dispone de una documentación, esta no es exhaustiva y resulta insuficiente para comprender en profundidad el flujo de ejecución del software.

El segundo, y más importante que el problema anterior, es la imposibilidad de completar la ejecución si no se ha optado por la utilización del archivo *stride*, debido a que, al ejecutar el comando que genera el archivo *dataset*, éste devuelve un error al no encontrar el archivo *stride* entre sus argumentos de entrada.

El tercero es la inexistencia de un archivo *requirements.txt* donde se definan los paquetes de Python que conforman las dependencias del proyecto.

Finalmente, la imposibilidad de escoger la ubicación o nombre de los archivos de salida de la red neuronal puede resultar un problema en algunos entornos en los que se ejecute en paralelo más de una vez el mismo protocolo con diferentes muestras, ya que al tener estas salidas siempre el mismo nombre y ubicarse en el mismo lugar, se sobrescribirá la más antigua con la más nueva, provocando errores en el flujo de trabajo que no se podrán detectar, al asumirse que la salida que se obtiene para una muestra es la correspondiente a esa muestra.

3.2.1.7 Correcciones aplicadas

En este proyecto, como parte de la fase de diseño, se ha propuesto corregir cada uno de los inconvenientes encontrados en cada uno de los programas analizados, y las correspondientes a este software se detallan a continuación.

Se ha creado una *Pull Request* con los cambios necesarios a la documentación para facilitar su uso a cualquier futuro usuario de Emap2sec, que, si bien no es un cambio que afecte de ninguna forma al protocolo de Scipion, mejora el uso del software en su versión *standalone*.

Para solucionar el segundo problema, ha sido necesaria la colaboración del responsable de mantener el repositorio, al que, tras informarle del problema encontrado, ha desarrollado un segundo archivo *dataset_wo_stride.py*, que se emplea en lugar de *dataset.py* en el tercer paso del flujo de ejecución, no necesitando éste el argumento correspondiente al archivo *stride*. También se han añadido los cambios correspondientes a la documentación.

Con el objetivo de solventar el tercer problema, se han averiguado todas las dependencias del proyecto realizando pruebas de ejecución y resolviendo los errores de dependencias que iban apareciendo en el proceso, y se ha creado una nueva *Pull Request* donde se añade al repositorio un archivo *requirements.txt* cuyo contenido son todas las dependencias encontradas en las versiones más recientes con las que funciona correctamente la ejecución.

Como solución al último problema, se ha creado una nueva *Pull Request* que añade dos cambios al script de Python encargado de llamar a la red neuronal. El primero es un parámetro opcional correspondiente al prefijo al nombre de los archivos de salida, de forma que se puede añadir una ruta personalizada, cualquier identificador que diferencie las distintas instancias del mismo, o ambas cosas a la vez. El segundo modifica el sufijo de los mismos archivos, sustituyendo el índice del archivo en la lista de ubicaciones por el nombre del archivo *dataset* correspondiente a esa salida, simplificando la tarea de relacionar una salida con su entrada. Los cambios realizados por esta *Pull Request* se han reflejado también en la documentación.

Como aclaración respecto a los cambios realizados, se añade que todos los cambios incluidos en las *Pull Request* realizadas, han sido fusionados con la rama principal del proyecto, lo que permitirá que todos los usuarios disfruten de estas correcciones y mejoras.

3.2.2 Emap2sec+

El segundo programa que será analizado es una versión basada en GPU y con funcionalidades extra de Emap2sec, llamada Emap2sec+. Este software es capaz de realizar las mismas tareas que Emap2sec, con el añadido de ser capaz de predecir ADN/ARN, y la condición de necesitar una GPU para su ejecución.

La ejecución de este software se lleva a cabo por medio de un único comando con gran diversidad de argumentos para diferentes modos de uso, todos ellos explicados en una documentación exhaustiva y clara. Al igual que con Emap2sec, la ejecución debe iniciarse desde el directorio raíz del proyecto.

3.2.2.1 Observaciones del software

Tras analizar este programa y probarlo en sus diferentes modos de uso, se han encontrado dos inconvenientes, similares a dos también presentes en Emap2sec.

El primero es que este software, genera como salida un árbol de directorios con nombres correspondientes a ciertos modos de ejecución, en los que guardará el archivo de salida si el modo de ejecución es el correspondiente al de ese directorio. Esos directorios se generan en la raíz del proyecto y tienen siempre el mismo nombre, al igual que los archivos generados en ellos. Este diseño puede causar los errores explicados para Emap2sec en entornos con múltiples instancias funcionando concurrentemente.

El segundo hace referencia al archivo *requirements.txt*, que, si bien existe en este caso, tiene algún error en algunas versiones de paquetes, y su nombre es *requirement.txt*, conllevando que el instalador automático no lo encuentre al no buscarlo por este nombre.

3.2.2.2 Correcciones aplicadas

Como solución al primer problema, se ha creado una *Pull Request* que añade un parámetro nuevo opcional al comando, llamado *--output_folder*, empleado para definir una ruta personalizada para el árbol de directorios de salida, que, si bien van a mantener los mismos nombres que antes de realizar el cambio, al estar en ubicaciones diferentes ya no causarán ninguna interferencia con otras instancias simultáneas.

Como solución al segundo problema, se ha creado otra *Pull Request* donde se soluciona el error en las versiones de algunos paquetes, y se renombra el archivo al nombre estándar para archivos de dependencias de Python.

Al igual que con Emap2sec, las *Pull Request* han sido fusionadas con la rama principal, haciendo disponible sus mejoras para todos los usuarios.

3.2.3 MainMast

A pesar de no ser necesario un análisis y pruebas de este software en su versión standalone, se ha decidido incluir una breve explicación de la función que cumple MainMast. Este programa, recibe como entrada una estructura cuaternaria, así como el tipo de simetría de la misma. A menudo, las estructuras cuaternarias están compuestas por diferentes estructuras terciarias que se repiten y unen entre sí, pudiendo mostrar estas cierta simetría. Dada esa entrada, MainMast diferencia las estructuras terciarias repetidas que componen la cuaternaria de entrada, y, opcionalmente, las separan en un archivo diferente cada una.

4 Desarrollo.

4.1 Desarrollo de un instalador automático

El primer paso del desarrollo consiste en crear un instalador automático para el plugin, que contenga todos sus protocolos. Para ello, será necesario modificar los archivos *constants.py* e *__init__.py* en el directorio con nombre del plugin dentro del repositorio del mismo. Cada programa por instalar requiere unos comandos concretos, que se detallarán a continuación.

El plugin en el que se van a integrar los protocolos propuestos en este proyecto contiene inicialmente otro protocolo, correspondiente al software *DAQScore*. Analizando los procesos de instalación de todos los programas que se quieren integrar, añadidos al protocolo ya existente, se ha determinado que, una buena forma de afrontar el instalador consistirá en generalizar todo lo posible el código a ejecutar en *__init__.py*, y que sea el contenido de las variables en *constants.py* el que determine el flujo de ejecución de la instalación. Para ello, se ha propuesto desarrollar una serie de pasos que seguirá de forma genérica el instalador, siempre guiados por una serie de variables definidas con anterioridad, cuyo nombre y propósito se definen a continuación.

4.1.1 Variables de instalación

Como norma general, todas las variables de cada grupo tendrán un sufijo idéntico y su prefijo coincidirá con el nombre del protocolo o repositorio al que harán referencia, lo que permitirá al instalador automático, sabiendo dicho prefijo, encontrar las variables correspondientes a cada protocolo y repositorio, permitiendo la generalización del proceso para su posible uso en otros plugins en un futuro sin apenas realizar modificaciones en el código.

En primer lugar, se define el nombre genérico del plugin, junto con su URL de GitHub. La primera variable permitirá al instalador localizar otras variables con el nombre del plugin de forma automática y extrapolable a otros plugins, mientras que la segunda se combinará con las variables que definan el fragmento de URL correspondiente a un repositorio a descargar, formando así la ruta completa.

A continuación, en de igual manera que con la mayoría de los plugins, se han definido las variables responsables de definir los números de versión de cada uno de los protocolos y repositorios empleados. De esta forma, se puede mantener una trazabilidad sobre el código del proyecto, y permitirá a los usuarios utilizar simultáneamente dos versiones de un mismo protocolo si así lo desean.

El siguiente conjunto de variables definidas hace referencia a las versiones de Python que cada repositorio necesita para funcionar correctamente. No todos los programas emplean código en Python, por lo que no todos los programas necesitarán una variable para definir dicha versión.

Tras terminar de definir las variables de la versión de Python, se encuentran dos grupos de variables similares, el primero siendo responsable del nombre que tendrá el directorio raíz del repositorio tras clonarlo, mientras que el segundo definirá el fragmento de URL correspondiente a dicho repositorio. La razón de dividir esta información en dos grupos de variables separados no es otra que la existencia de repositorios con nombres de directorio deseados diferentes al indicado en la URL de GitHub del mismo. De esta forma, al clonar un repositorio, se puede indicar su enlace completo a la vez que el nombre de directorio deseado en un único comando.

Las dos variables siguientes, son las responsables de definir los protocolos que se instalarán, así como de los repositorios que contendrá cada uno de ellos. Este enfoque permite a cada protocolo disponer en su interior de un número indeterminado de repositorios diferentes, pudiendo tener requisitos de instalación también muy dispares entre sí. En este caso, esa funcionalidad va a emplearse para fusionar Emap2sec y Emap2sec+ en un mismo protocolo. Estos dos programas toman como entrada el mismo tipo de archivo y generan como resultado un archivo también del mismo tipo en ambos casos, por lo que se ha decidido integrar ambos softwares en un mismo protocolo, y presentar al usuario la opción de utilizar uno u otro como un desplegable. Esta decisión permite adicionalmente que el usuario escoja el software a emplear entre estos dos basado principalmente en el hardware sobre el que se ejecuta, según si quiere llevar a cabo una ejecución basada en CPU o GPU.

A continuación, se puede apreciar otro grupo de variables, en este caso encargadas de definir una lista de dependencias para cada repositorio, que se le entregará a Scipion antes de comenzar el proceso de instalación para que compruebe si se encuentran instaladas en el sistema, y de no ser así, advertir al usuario y abortar el proceso de instalación para evitar errores en el mismo.

El siguiente grupo de variables se trata de los archivos extra de cada repositorio. Algunos programas requieren la descarga de archivos extra para funcionar correctamente, como es el caso de Emap2sec y Emap2sec+, que emplean ambos unos modelos entrenados previamente que deben descargarse de una dirección diferente al repositorio de GitHub. Como además de necesitar la URL de descarga de cada archivo, éstos pueden necesitar encontrarse en lugares específicos del árbol de directorios del proyecto, se ha definido la variable como una lista de tuplas, donde el primer elemento de la tupla será el enlace de descarga, y el segundo la ubicación relativa a la raíz del proyecto, que creará la ruta indicada de no existir previamente.

Como último grupo de variables de la propia instalación, se encuentran los comandos extra que necesitará ejecutar cada uno de los repositorios para terminar su proceso de instalación. Los ejemplos más comunes incluyen la descompresión de archivos descargados, la compilación de cierto código, o la adición de permisos de ejecución al directorio del proyecto.

Por último, se pueden apreciar algunas variables definidas para cada protocolo, que, al margen de la instalación, sirven para almacenar opciones en las entradas del formulario del protocolo.

4.1.2 Proceso de instalación

El proceso que sigue el instalador, aunque, dada la heterogeneidad del software de distintos protocolos de Scipion, no es todavía lo suficientemente general como para reutilizar el mismo código para absolutamente todos los protocolos, se espera que sea fácilmente adaptable a las necesidades aún no cubiertas por la implementación actual, que de momento cubre las necesidades de los protocolos del plugin actual, y las de todos los plugins con necesidades similares. Dicho proceso viene guiado por los valores de las variables de las que se ha hablado en el punto anterior, y su flujo de ejecución es el siguiente.

En primer lugar, se escriben en el archivo de configuración de Scipion las variables que determinan la ubicación de instalación de cada protocolo. Esto se lleva a cabo en la función *_defineVariables*, existente en todos los instaladores de plugins de Scipion. Este paso se realiza una vez por cada protocolo a instalar.

A continuación, se ejecuta la segunda función común, *defineBinaries*, en donde se llama a la función personalizada *addProtocolPackage*, donde se generan los comandos de instalación según las necesidades de cada protocolo y finalmente se llama a la última función común, *addPackage*. El proceso de generación de dichos comandos sigue la lógica detallada a continuación. Cada uno de los siguientes pasos se realizará una vez por cada repositorio de cada protocolo definido.

El instalador comprueba que exista la variable que define el fragmento de URL de GitHub necesaria para clonar el repositorio, y, de existir, clona el repositorio dentro del directorio del protocolo, yo lo nombra con el valor de la variable definida para esa función.

El siguiente paso de la instalación es comprobar si el repositorio necesita descargar algún archivo extra, y de ser así, los descarga en la ruta indicada, relativa a la raíz del repositorio si se ha clonado, y a la del protocolo si no se ha clonado nada.

En el mismo lugar que la descarga de los archivos, se ejecutan a continuación los comandos especificados en la variable de ese grupo correspondiente al protocolo actual, de existir dicha variable.

Como último paso, y también de manera opcional, se crea el entorno de *Conda* para ese software en la versión de Python especificada y se instalan los paquetes pertenecientes a su lista de dependencias definida en el archivo *requirements.txt*.

4.2 Desarrollo de los protocolos

Una vez terminado el instalador automático, el siguiente paso propuesto en este Trabajo de Fin de Grado, consiste en el desarrollo de los protocolos indicados, creando desde cero el protocolo en el que se encontrarán Emap2sec y Emap2sec+, y migrando y adaptando el correspondiente a MainMast.

4.2.1 Emap2sec

En primer lugar, se va a desarrollar el protocolo que integre los programas Emap2sec y Emap2sec+. Dicho protocolo se llamará como el primero de los dos programas, al ser el nombre más general de los dos.

4.2.1.1 Parámetros del formulario

Los primeros dos parámetros que se van a incluir en el formulario del protocolo son, en primer lugar, el objeto de entrada que recibirá el programa seleccionado, y a continuación el desplegable que permite seleccionar entre Emap2sec y Emap2sec+. El elemento recibido como entrada podrá ser un objeto de tipo *Volume* o *SetOfVolumes*, que son dos tipos de datos admitidos por Scipion para representar archivos de volumen. A partir del programa escogido, el resto de los campos del formulario serán diferentes, ya que cada uno de los dos programas recibe diferentes argumentos.

4.2.1.1.1 Emap2sec

En el caso de haber seleccionado Emap2sec, en el modo normal se podrá apreciar únicamente un campo nuevo obligatorio, englobado en un grupo. Un grupo es un elemento únicamente visual, con una etiqueta, que rodea uno o más campos con el fin de agrupar aquellos pertenecientes a una llamada concreta de las múltiples que puede realizar un programa. En el caso de este protocolo, es grupo tendrá como etiqueta "Trimmap generation", ya que engloba a todos los argumentos que recibirá la llamada al ejecutable que cumple con este paso, descrito en el apartado 3.2.1.1. El aspecto del formulario será el de la figura 4.1.

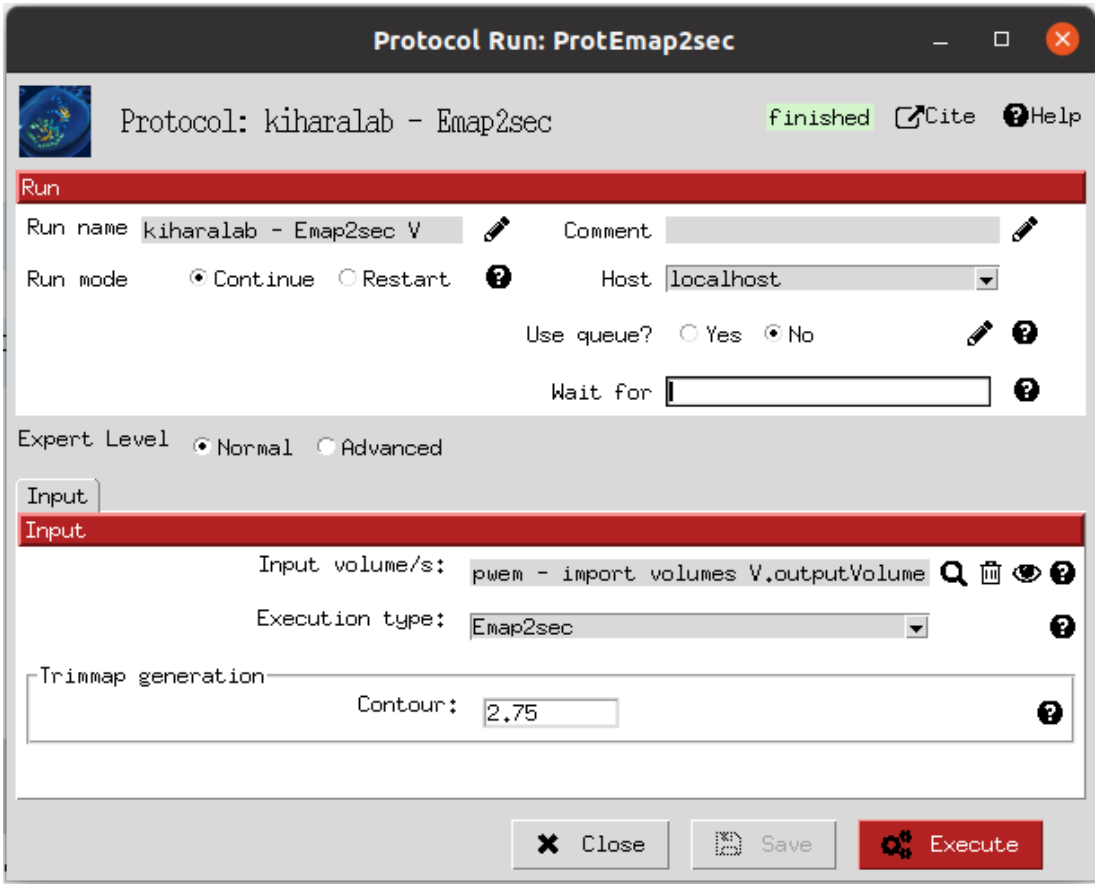

Figura 4.1: Formulario Emap2sec en el modo *Normal*.

El resto de los parámetros se encuentran ocultos por defecto, ya que tienen todos valores por defecto recomendados para la mayoría de los casos, pero pueden mostrarse y modificarse marcando la opción "Advanced" en el campo "Expert Level". El aspecto del formulario tras modificar esa opción será el mostrado por la figura 4.2.

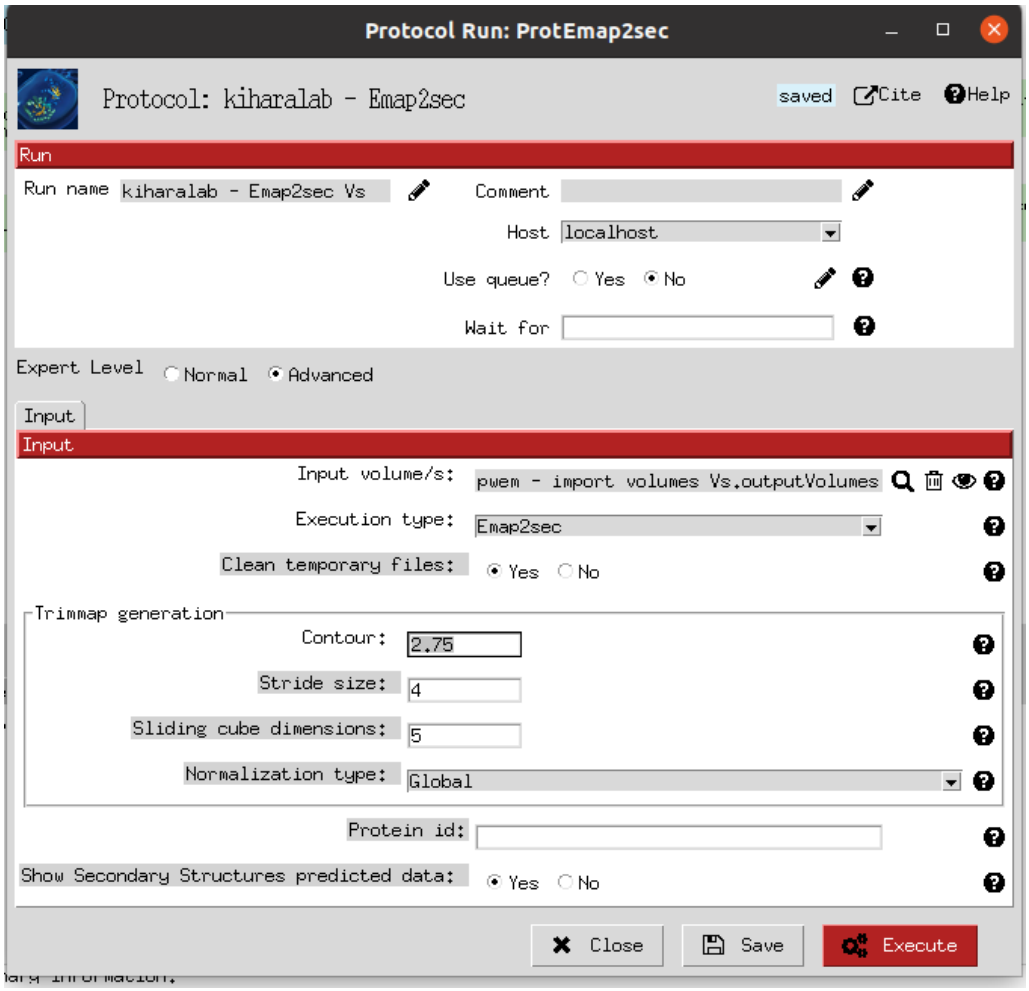

Figura 4.2: Formulario Emap2sec en el modo *Advanced*.

Como se puede comprobar, con el fin de ahorrar espacio en disco, se incluye además la opción de eliminar todos los archivos generados por la ejecución salvo los que contienen los resultados, que se importan como un objeto de Scipion, concretamente de tipo *AtomStruct* en el caso de haber recibido uno de *Volume* como entrada, o *SetOfAtomStructs*, con un objeto de tipo *AtomStruct* en su interior por cada uno de los *Volume* pertenecientes al set de entrada, si se ha recibido uno de tipo *SetOfVolumes*.

4.2.1.1.2 Emap2sec+

De la misma forma, si se selecciona como programa a emplear Emap2sec+, se presenta por defecto un conjunto formado por dos campos obligatorios, correspondientes a la id de la GPU que se va a emplear para ejecutar el software, y el nivel de contorno que se aplicará a la muestra digital. El valor del nivel de contorno deberá coincidir con el recomendado por los autores de dicha muestra, información frecuentemente disponible en los repositorios de muestras digitales, como EMDB. Por defecto, se deja el valor a 0.0, lo que indica a Emap2sec+ que no hay un nivel de contorno disponible, por lo que deberá emplearse el modelo generalista para esa ejecución. De la misma forma, se ha añadido 0 como valor por defecto de la GPU a emplear, ya que, de existir al menos una GPU disponible para el sistema operativo sobre el que se esté ejecutando Scipion, siempre existirá una con el id 0. Al ejecutar Emap2sec+ una única llamada a un script de Python no será necesario englobar ningún parámetro en un grupo. La figura 4.3 muestra el aspecto del formulario en esta situación.

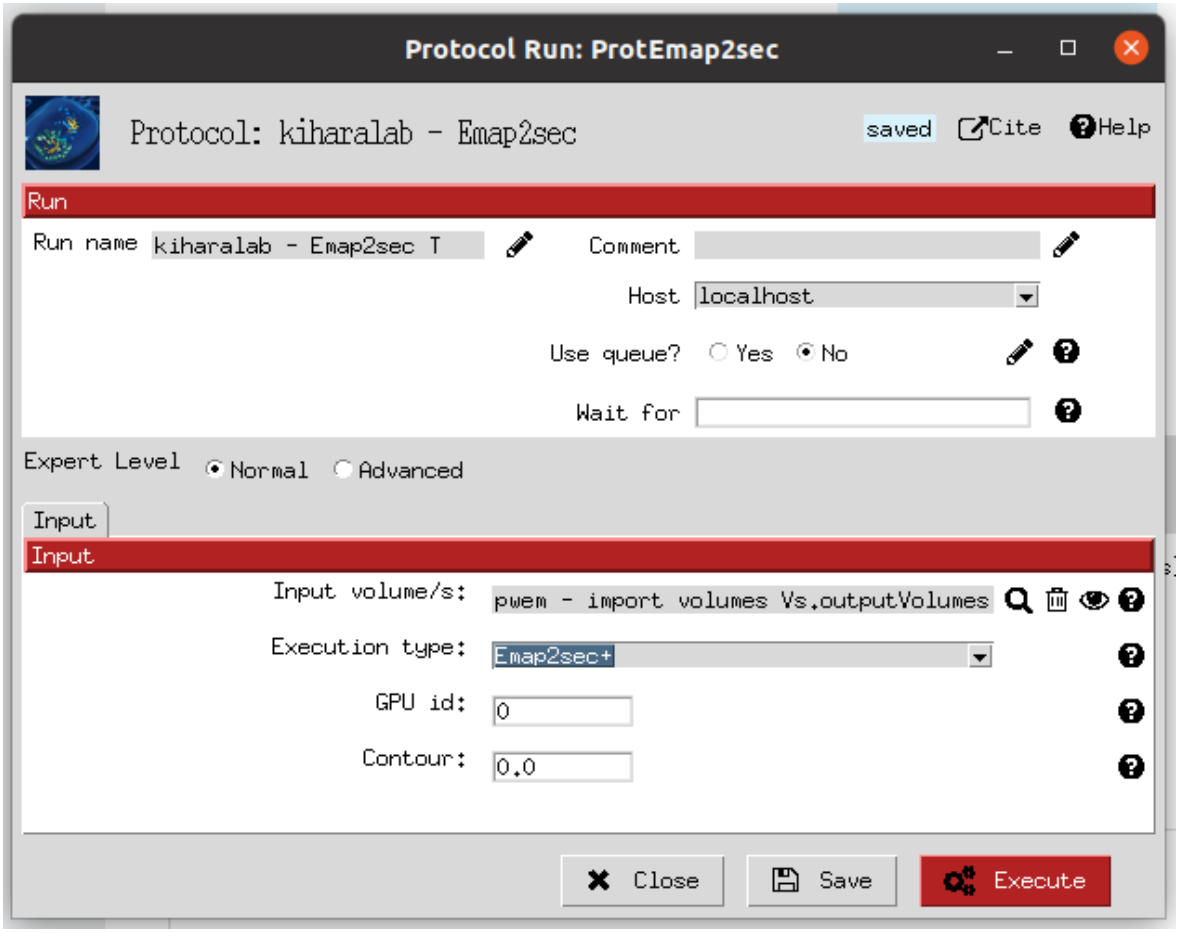

Figura 4.3: Formulario Emap2sec+ en el modo *Normal*.

Al igual que con Emap2sec, los demás parámetros se encuentran ocultos tras la opción de uso avanzado, que, de seleccionarse, modifica el aspecto del formulario tal y como se aprecia en las figuras 4.4 y 4.5.

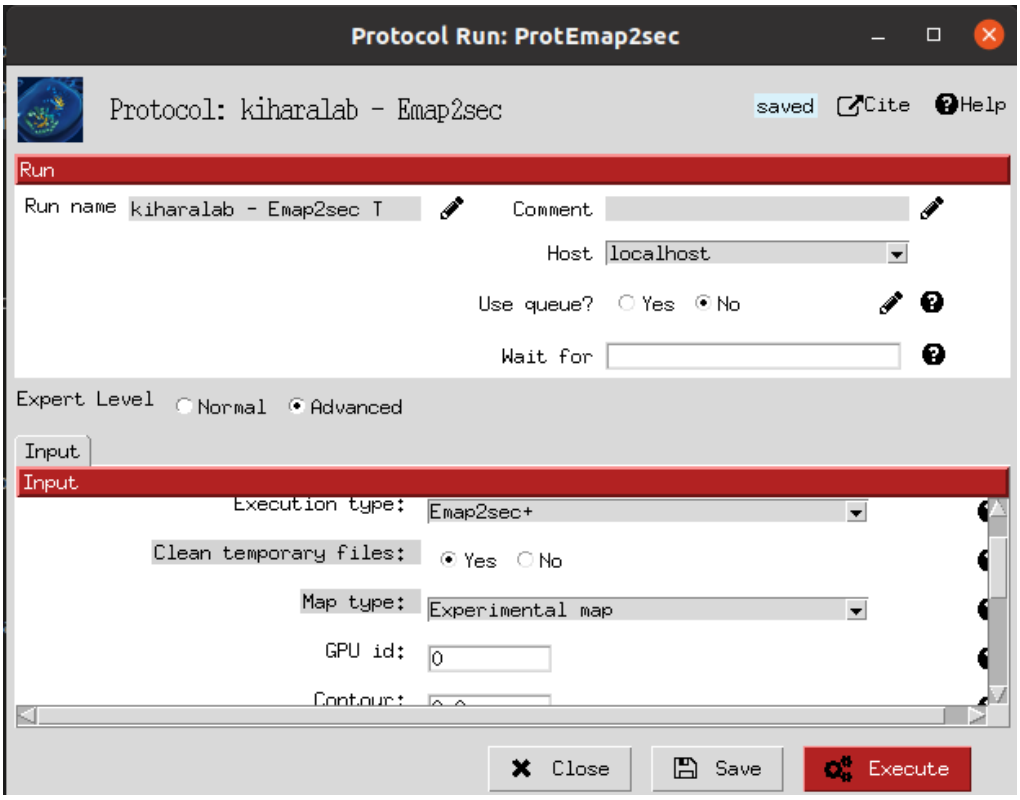

Figura 4.4: Formulario Emap2sec+ en el modo *Advanced*. Parte 1 de 2.

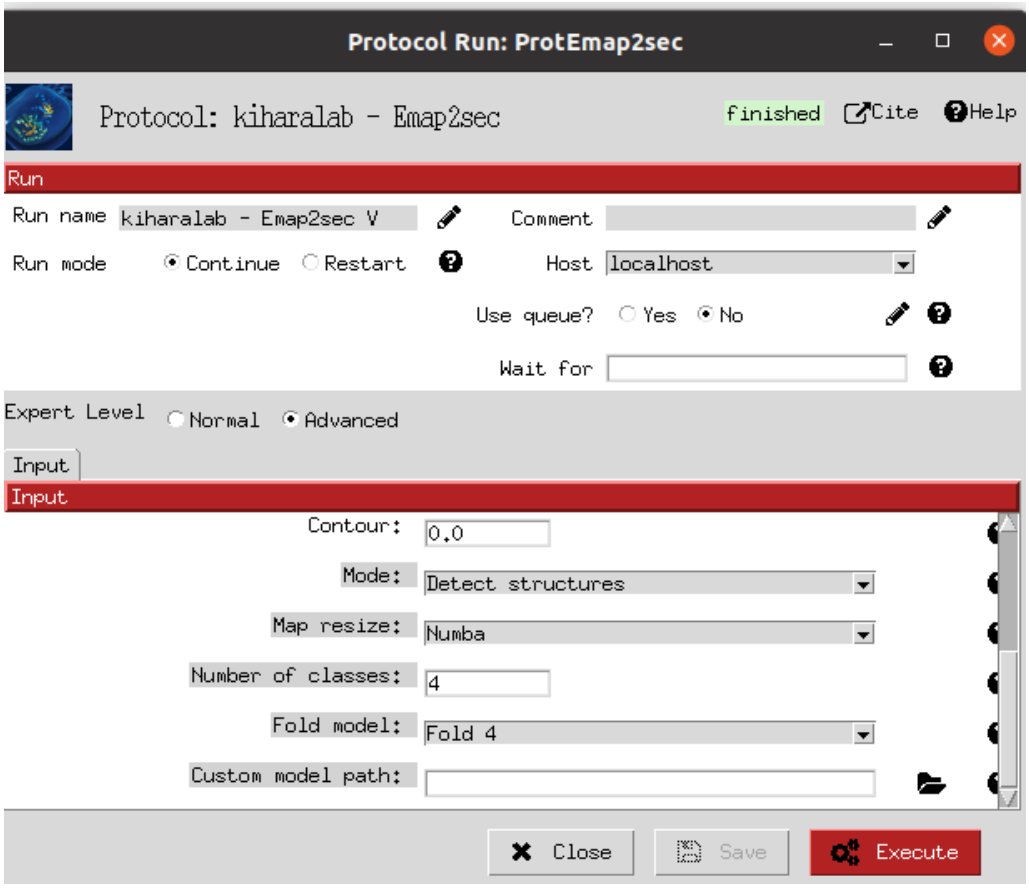

Figura 4.5: Formulario Emap2sec+ en el modo *Advanced*. Parte 2 de 2.

4.2.1.2 *Ejecución del protocolo*

Una vez el formulario se encuentra completo, en el archivo *__init__.py* en el que se desarrolló previamente el instalador, también deberá incluirse una función para llevar a cabo la ejecución del protocolo instalado. Dicha función será llamada desde otra función ubicada en el archivo del protocolo, la cual le proporcionará toda la información necesaria para realizar las llamadas necesarias. Por ello, el desarrollo de la etapa de ejecución consistirá en tres pasos:

1. Desarrollo de las funciones del protocolo que, en función de los valores de los campos del formulario, generen las cadenas de texto necesarias para realizar las llamadas al software.

2. Desarrollo de la función en *__init__.py* que se encarga de ejecutar el protocolo con la información recibida del paso anterior.

3. Recolección de los archivos que contienen los resultados de la ejecución y su importación como tipos de datos de Scipion.

4.2.1.2.1 *Uso de los valores del formulario*

La fase de recolección de los datos del formulario y la generación de las cadenas de texto necesarias para la ejecución del protocolo viene definida dentro de la función *mainExecutionStep*, que realiza llamadas a diferentes funciones de procesamiento de los formularios según si se ha escogido una ejecución basada en Emap2sec o en Emap2sec+, y recoge sus resultados como argumentos que luego transmite a la función correspondiente de ejecución del programa seleccionado.

En el caso de Emap2sec, emplea una función para cada una de las llamadas a un programa externo, coincidiendo a su vez con cada uno de los pasos de este software, explicados en el apartado 3.2.1. Por otra parte, Emap2sec+ genera una única llamada a un script de Python por cada archivo de entrada, necesitando para ello llamar a una sola función para generar sus argumentos. Ambas ejecuciones soportan la introducción de conjuntos de volúmenes, representados en Scipion por el objeto *SetOfVolumes*, implicando una ejecución en lote de los volúmenes individuales que lo componen.

4.2.1.2.2 *Ejecución de los comandos con los argumentos generados*

Una vez se han generado las cadenas de texto con los argumentos necesarios para la ejecución, mainExecutionStep llama a la función que corresponda según el software seleccionado. Por ello, la ejecución a partir de ese punto sigue un hilo distinto hasta la recogida de los resultados.

Cuando se ejecuta Emap2sec, se recibe como argumento una lista de listas. Cada lista que conforma la lista general está a su vez compuesta por cadenas de texto, cada una correspondiente a los argumentos de una llamada a ejecutable o script de Python, salvo la última, que contiene los elementos que serán eliminados. En primer lugar, se ejecuta cada paso previo al uso de la red neuronal una vez por cada volumen que contenga la entrada, para, posteriormente, realizar una única llamada al script que utiliza dicha red, que ya es capaz de gestionar en la misma llamada los múltiples archivos correspondientes a cada entrada en una misma ejecución. Finalmente, se ejecuta también una vez por cada volumen de entrada el comando que genera la solución a partir de los resultados del paso anterior.

Por otra parte, al ejecutar Emap2sec+, se recibe una lista con dos listas en su interior. La primera contiene, por cada volumen de entrada, una cadena de texto con los argumentos necesarios para la ejecución del programa, que solo puede procesar una entrada a la vez. La última, al igual que con Emap2sec, contiene los archivos y directorios que deberán ser eliminados al finalizar la ejecución del software si se ha seleccionado la opción de limpiar los archivos residuales, con el fin de optimizar el uso del disco. Debido a la incapacidad de Emap2sec+ de procesar varias entradas en lote, el protocolo cubre esa funcionalidad en su lugar realizando en un bucle la llamada al programa por cada una de las entradas.

4.2.1.2.3 Procesamiento de los resultados

Una vez ha terminado la ejecución del software elegido, termina con él la de la función *mainExecutionStep*, comenzando a su vez la de *createOutputStep*. En esta función, se comprueba el tipo de entrada recibida, y se genera, empleando los archivos generados como resultado por Emap2sec o Emap2sec+, el objeto de salida correspondiente que devolverá el protocolo y que podrá ser empleado como entrada por otros protocolos.

4.2.2 MainMast

Tras integrar el protocolo Emap2sec, en este Trabajo de Fin de Grado, se ha propuesto también migrar el protocolo MainMast de su plugin actual, *scipion-em-mainmast*, al plugin con el que se trabaja durante este proyecto, al ser el software de este protocolo propiedad del mismo laboratorio que los programas que conforman todos los protocolos de este plugin.

La migración del protocolo es un proceso significativamente más corto que la creación desde cero, y consta de los siguientes pasos:

1. Análisis de las dependencias y comandos necesarios para completar la instalación del protocolo, información definida en el archivo *__init__.py* de *scipion-em-mainmast*.

2. Definición de las variables de instalación necesarias para completar los pasos extraídos del paso anterior, en el archivo *__init__.py* del plugin actual, *scipion-em-kiharalab*.

3. Copia de los archivos del protocolo y el viewer, y la función de su ejecución de *__init__.py*.

4. En el caso específico de *scipion-em-mainmast*, no hay unos tests desarrollados, por lo que se incluirá el desarrollo de los mismos como parte de la integración total del protocolo.

4.2.2.1 *Adaptación de la instalación*

Un análisis al código presente en el archivo *__init__.py* de *scipion-em-mainmast* revela que el proceso de instalación de MainMast es más sencillo y corto que el de Emap2sec, y el instalador ya desarrollado en el apartado 4.1 está preparado para, tras definirse las variables necesarias y modificar otras en *constants.py*, añadir MainMast al proceso automático de instalación. Se ha decidido modificar uno de los comandos empleados en el plugin original, encargado de la descarga de los binarios. En *scipion-em-mainmast*, se descarga una versión específica del software, por medio del comando *wget*, y se ha modificado por el correspondiente *git clone*, con el objetivo de permitir a los usuarios recibir por defecto la última versión de la rama principal de proyecto, pudiendo añadir versiones anteriores por separado si se desea más adelante. Adicionalmente, al emplear el comando *git clone*, se clona el proyecto como un directorio sin comprimir, por lo que no es necesario descomprimirlo a continuación, y se puede clonar con el nombre de directorio deseado sin necesidad de renombrarlo a continuación, como sucede con la descarga basada en el comando *wget*.

Tras el análisis realizado del software y los cambios propuestos a su proceso de instalación, se procede a definir las variables necesarias para completar la instalación de MainMast, siguiendo el siguiente orden:

1. Clonación del repositorio.

2. Compilación y generación de los ejecutables, y limpieza de archivos residuales.

3. Descompresión de los archivos de ejemplo incluidos.

4.2.2.2 *Adaptación del archivo del protocolo y su función de ejecución*

Con el proceso de instalación completado, el siguiente paso consiste en copiar el archivo del protocolo a su nueva ubicación, así como las funciones de ejecución del protocolo dentro del archivo *__init__.py*. Tras finalizar dicho proceso, será necesario adaptar el contenido de los mismos a la nueva ubicación, principalmente los *imports* que realiza Python, añadiendo además la variable *_possibleOutputs*, con el fin de indicar las posibles salidas del protocolo sin necesitar haber completado su ejecución, característica no disponible en *scipion-em-mainmast*, al ser su desarrollo previo a la actualización de Scipion que introdujo esta funcionalidad, y no haber sido actualizada posteriormente en consecuencia.

5 Integración, pruebas y resultados

5.1 Tests de integración y pruebas

Tras haber completado tanto el análisis como desarrollo de todos los protocolos propuestos en este Trabajo de Fin de Grado, a continuación, se ha previsto el desarrollo de los tests de integración de cada protocolo, pretendiendo en ellos completar una serie de ejecuciones de prueba con diferentes argumentos, a fin de comprobar que la instalación se ha realizado correctamente y el protocolo testado está listo para ser utilizado. Por ello, será necesario desarrollar dos tests, siendo el primero destinado al protocolo Emap2sec, el restante a MainMast.

En Scipion, para obtener los archivos de entrada necesarios en la ejecución de un protocolo cualquiera, existe una clase definida llamada *Dataset*, que contiene a su vez una función encargada de obtener de un servidor de Scipion proveedor de estos sets de datos, los archivos correspondientes a un *dataset* definido con un nombre específico, para lo que únicamente hay que entregar el nombre del set que se quiere obtener. Si un protocolo necesita emplear uno o varios archivos muy específicos para sus pruebas de integración, se pueden subir esos archivos a un dataset existente o incluso crear uno nuevo. Sin embargo, con el fin de evitar aumentar el espacio ocupado por los distintos sets en el servidor subiendo a los mismos los archivos de ejemplo provistos por los diferentes programas empleados en los protocolos creados durante el desarrollo de este proyecto, se van a desarrollar los tests empleando archivos de entrada ya existentes en el *dataset* con nombre *model-building-tutorial*, simplificando de esta forma también el proceso de desarrollo de los mismos.

5.1.1 Emap2sec

Para testar el funcionamiento del protocolo Emap2sec, se va a emplear un archivo de volumen del *dataset* mencionado previamente, llamado *emd_6838.mrc*. Tras buscar su id en el repositorio de muestras EMDB, se ha obtenido su valor de contorno, que deberá ser 5,4.

El set de tests consistirá en dos ejecuciones para el software Emap2sec, y tres para Emap2sec+.

En el caso de Emap2sec, cada una de las ejecuciones se utilizará para introducir en el protocolo un tipo de entrada diferente, correspondiendo la primera prueba a un objeto de tipo *Volume*, y la segunda a uno del tipo *SetOfVolumes*, conteniendo este, dos objetos de tipo *Volume*. Como en el *dataset* hay un único archivo de volumen válido para la ejecución de este protocolo, para el test del set de volúmenes se va a duplicar el archivo, simulando la existencia de un segundo volumen diferente del primero. El objetivo del test no es otro que comprobar que la ejecución puede completarse con ambos tipos de entrada, y que los objetos devueltos como salida coinciden con el tipo esperado y contienen resultados válidos.

Para Emap2sec+, de forma similar, se realiza un test para los dos modos de ejecución cuyo uso se estima más frecuente, y uno para un tercer modo escogido con un set como entrada en lugar de un único volumen. El motivo de esta elección es la cantidad de tiempo necesario para completar una ejecución del software. Emap2sec+ tiene un tiempo de ejecución considerablemente prolongado, pudiendo durar más de una hora en completarse, por lo que hacer una ejecución por cada posible cambio en cada uno de los parámetros disponibles es completamente inviable, ya que estos tests automáticos están diseñados para poder ejecutarse automáticamente tras terminar el proceso de instalación, teniendo que cumplir un cierto equilibrio entre probar la instalación con alguna ejecución de prueba, evitando consumir demasiado tiempo en el proceso, ya que la ejecución de estos test podría estar bloqueando el flujo de trabajo del usuario.

5.1.2 MainMast

En el caso de MainMast, al solo estar compuesto por un programa, el proceso de prueba es mucho más sencillo, por lo que se realizarán dos tests que prueben su correcto funcionamiento, basados en si se ha seleccionado o no la opción del formulario “Combine masks”, ya que la salida del protocolo cambiará de tipo de dato en base a esa selección. Para ello, se va a emplear el mismo archivo de entrada que en los tests del protocolo Emap2sec, y, al igual que para el primer programa de dicho protocolo, tras iniciar su ejecución, se comprobará que el resultado coincide con el tipo de dato esperado y que su contenido es válido.

Durante el desarrollo de los tests, se ha descubierto un fallo del software en sistemas con una cantidad de memoria RAM aparentemente insuficiente para la ejecución del programa, lo que lleva a una salida prematura del mismo sin ninguna clase de aviso, lo que genera posteriormente errores más difíciles de diagnosticar. Entrando más en detalle, al ejecutar un comando que genera varios archivos con un nombre siguiendo el patrón “region*.mrc”, donde “*” puede ser un número entero, dichos archivos no aparecían, y más adelante en la ejecución, se realizaban una serie de operaciones, entre ellas la definición de una variable, iterando sobre la lista de archivos cuyo nombre cumplía el patrón descrito, significando esto que variable nunca llegase a definirse. Tras intentar el protocolo utilizar esa variable poco después, se generaba una excepción indicando que la variable se empleaba antes de haber sido definida, un error que aportaría muy poca información útil a un usuario promedio del protocolo, obligándole a mirar en el código para encontrar su origen.

La primera prueba se realizó sobre un sistema operativo Ubuntu 20.4 con 4GB de RAM asignadas, cuyo resultado de forma consistente era la aparición de este error, sin importar si se reiniciaba el sistema y se lanzaba la ejecución con Scipion como único software en primer plano, y ningún otro programa en segundo plano más allá de los procesos del propio sistema operativo.

Sin embargo, para la segunda prueba, la única modificación realizada fue la asignación de 8GB de RAM en lugar de las 4GB iniciales, medida tras la cual el error no apareció en ningún momento.

Como se estima que la cantidad de memoria RAM requerida por MainMast para poder funcionar adecuadamente varía en función del archivo que se esté procesando, se ha decidido, tras llamar al comando que genera los archivos descritos, comprobar la existencia de los mismos, y de no encontrarse, terminar la ejecución del protocolo generando una excepción indicando la inexistencia de los archivos solicitados, añadiendo que la causa del fallo podría deberse a la falta de memoria dinámica a la que puede acceder el proceso.

5.2 Resultados

Tras completar todos los apartados propuestos durante el desarrollo del Trabajo de fin de Grado, se ha conseguido crear los protocolos propuestos, con un funcionamiento satisfactorio de acuerdo con lo esperado de los programas que los componen, permitiendo su uso dentro del framework Scipion a cualquiera de los del mismo, facilitando así la tarea de gran cantidad de investigadores en el área de la Biología Molecular en múltiples laboratorios repartidos por diversos países del planeta.

Respecto a la ejecución de los protocolos, se ha probado cada uno de los protocolos con un caso de ejemplo y se han analizado sus resultados, viniendo definidos en las figuras 5.1 y 5.2.

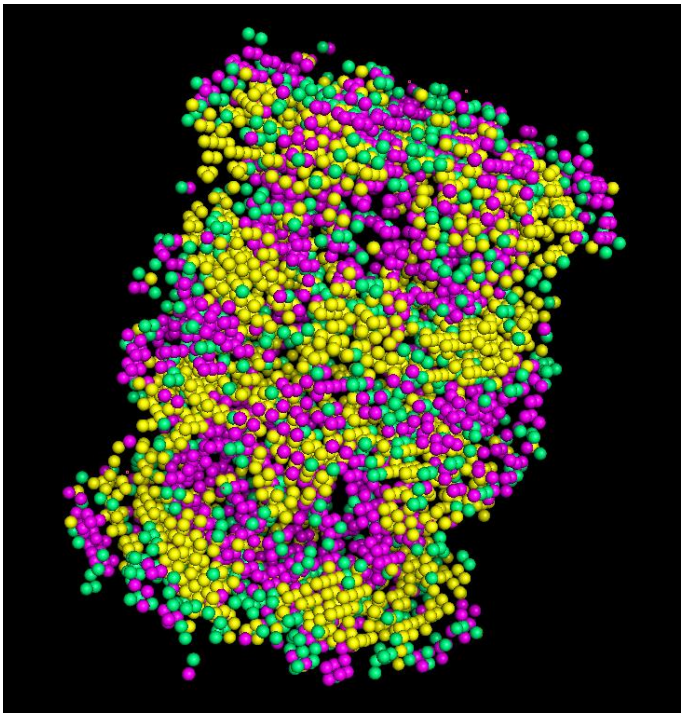

Figura 5.1: Resultado de Emap2sec para el objeto con EMID 1733 visualizado en Pymol.

En la figura 5.1, se puede apreciar el resultado de una ejecución, visualizado en el software Pymol. En él, se han coloreado de manera diferente las estructuras secundarias predichas por el protocolo Emap2sec. La razón de que la visualización sea en forma de esferas es debido a que el resultado de Emap2sec se da en forma de rejilla, por lo que cada esfera representa un vóxel del mapa original.

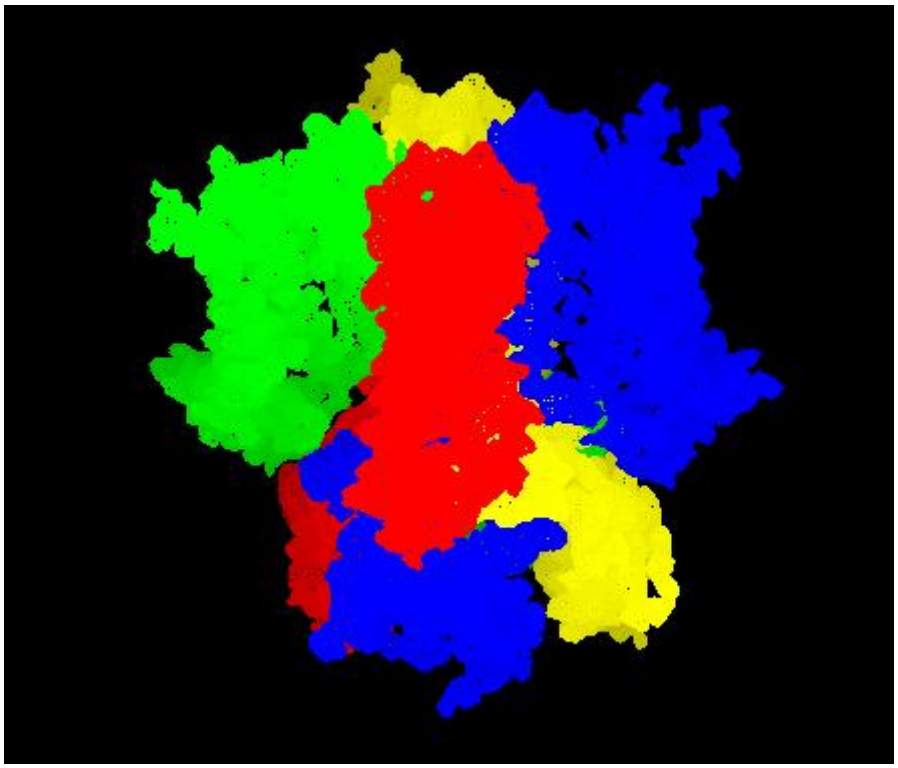

Figura 5.2: Resultado de MainMast para el objeto con EMID 0093 visualizado en Pymol.

En la figura 5.2, se muestra el resultado de una ejecución de MainMast, donde, dada la estructura cuaternaria, correspondiente al modelo completo, se muestra de cada color una de las estructuras terciarias ubicadas como parte de la misma.

Como aporte adicional, durante el desarrollo del proyecto se han encontrado algunos errores en Scipion, que han sido reportados a sus desarrolladores con el fin de que sean subsanados. También se han sugerido algunas mejoras de la funcionalidad que serán tenidas en cuenta por los desarrolladores del framework para mejorar la experiencia de usuario del mismo.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

A lo largo de este Trabajo de Fin de Grado, se ha seguido el proceso de desarrollo de varios protocolos del framework Scipion.

En primer lugar, se ha analizado la arquitectura y funcionamiento del framework para el que se iban a desarrollar los protocolos propuestos, a fin de comprender mejor su funcionamiento y simplificar su uso y, en definitiva, el desarrollo que se tenía como objetivo del proyecto.

A continuación, como parte también del esfuerzo de análisis previo, se han explorado los programas que debían conformar los protocolos a desarrollar, realizando a los mismos las modificaciones necesarias para su correcto funcionamiento tanto de manera individual como integrados en un protocolo.

Posteriormente se ha realizado el desarrollo del instalador necesario para el proceso automático de instalación del plugin, así como el de los propios protocolos, y se ha probado que su funcionamiento es el esperado y equivalente al de su versión *standalone*.

Por último, se han desarrollado los tests de integración necesarios para comprobar de forma automática la correcta instalación de los protocolos, solucionando en el proceso errores en el software que se hayan encontrado en esta fase bajo las circunstancias de los tests.

6.2 Trabajo futuro

Existe una gran variedad de posibilidades referentes al trabajo a futuro, ya que Scipion es un framework siendo desarrollado activamente, con una base de usuarios en constante crecimiento y unas utilidades mayores a medida que pasa el tiempo, pero algunas de las propuestas se describen a continuación.

- Ofrecer mantenimiento al plugin, o en su defecto, a los protocolos desarrollados, ante cualquier problema que puedan tener los usuarios del mismo, o ante las propuestas de mejora de las funcionalidades que se puedan recibir.

- Realizar, una vez se conoce el proceso completo de desarrollo de un protocolo, la integración de más programas que puedan ser útiles en el entorno del framework y que carezcan de protocolo todavía, ampliando con ello la variedad de algoritmos de procesamiento de muestras digitales disponibles en Scipion.

- Al ser Scipion un framework completamente de código abierto, se puede colaborar en su desarrollo de forma activa, y, si bien se han propuesto mejoras y correcciones de errores durante el desarrollo del Trabajo de Fin de Grado, puede resultar interesante aportar al framework directamente como desarrollador en lugar de como mero usuario.

Referencias

[1] J.M. de la Rosa-Trevín, A. Quintana, L. del Cano, A. Zaldívar, I. Foche, J. Gutiérrez, J. Gómez-Blanco, J. Burguet-Castell, J. Cuenca-Alba, V. Abrishami, J. Vargas, J. Otón, G. Sharov, J.L. Vilas, J. Navas, P. Conesa, M. Kazemi, R. Marabini, C.O.S. Sorzano, J.M. Carazo, "Scipion: A software framework toward integration, reproducibility and validation in 3D electron microscopy", Journal of Structural Biology, July 2016, pp. 93-99.

[2] Xiao Wang, Eman Alnabati, Tunde W Aderinwale, Sai Raghavendra Maddhuri Venkata Subramaniya, Genki Terashi & Daisuke Kihara, "Detecting protein and DNA/RNA structures in cryo-EM maps of intermediate resolution using deep learning", Nature Communications 12, 2302 (2021).

[3] Terashi, G., Kagaya, Y. & Kihara, D, "MAINMASTseg: Automated Map Segmentation Method for Cryo-EM Density Maps with Symmetry", J Chem Inf Model 60, 2634-2643 (2020).

Glosario

GPU	Graphics Processing Unit
CPU	Central Processing Unit
ADN	Ácido Desoxirribonucleico
ARN	Ácido Ribonucleico
URL	Uniform Resource Locator
CSIC	Consejo Superior de Investigaciones Científicas

Definiciones

Microscopio electrónico
Un microscopio que emplea electrones en lugar de fotones o luz visible para formar imágenes de objetos diminutos.

Framework
Puede traducirse como entorno de trabajo. Conjunto estandarizado de conceptos, criterios, y prácticas que resuelven un problema en particular, diseñado para resolver problemas similares.

Código abierto
Modelo de desarrollo de software basado en la colaboración abierta.

Biotecnología
Rama de las ciencias biológicas que estudia toda aplicación tecnológica que emplee organismos vivos o sistemas biológicos o sus derivados.

Farmacología
Rama de las ciencias farmacéuticas que estudia las sustancias químicas que interactúan con los seres vivos.

Pull Request
Es la petición realizada por el propietario de un fork de un proyecto al propietario del proyecto original para que introduzca en este los cambios realizados en el fork.

Fork
Copia de un repositorio, que permite a quien lo crea experimentar libremente con el código del proyecto copiado y realizar modificaciones sin necesitar permisos de acceso al proyecto original.

Plugin
Aplicación que se añade a un programa informático ampliando su funcionalidad.

Python
Lenguaje de programación de alto nivel interpretado diseñado con la legibilidad del código en mente.

Git
Software de control de versiones diseñado para mejorar la eficiencia, confiabilidad, y compatibilidad de versiones de aplicaciones cuando tienen gran cantidad de archivos de código fuente.

GitHub
Forja para alojar proyectos empleando el software Git.

Red neuronal
Grupo interconectado de nodos (neuronas) que transmiten señales entre sí.

Red neuronal profunda
Tipo de red neuronal que contiene varias capas ocultas entre las capas de entrada y salida.

Benchmark
Prueba de rendimiento de un sistema o componente de un sistema.

Script
Secuencia de comandos que conforman un programa relativamente sencillo.

Dataset
Colección de datos frecuentemente tabulados.

Standalone
Aplicado a un software, significa que puede instalarse y/o ejecutarse sin requisitos adicionales.

Conda
Gestor de paquetes y sistema de gestión de entornos de código abierto.

Anexos

A Manual de instalación

El proceso de instalación del software desarrollado durante este proyecto se divide en dos pasos, estando cada uno de ellos ya desarrollados en profundidad en sus respectivas guías.

En primer lugar, será necesaria la instalación del framework Scipion, cuya guía completa de instalación puede encontrarse en el siguiente enlace: https://scipion-em.github.io/docs/docs/scipion-modes/how-to-install.html.

Con el framework y sus dependencias correctamente instaladas, sería necesario instalar el protocolo, y eso puede hacerse, de estar empaquetado el plugin y publicado como paquete en los repositorios adecuados, desde el *Plugin manager* en la interfaz de Scipion. Alternativamente, siempre existe la posibilidad de clonar el repositorio de manera independiente e instalarlo por medio de comandos de terminal.

Para instalar *scipion-em-kiharalab* de esa manera, el primer paso será clonar el repositorio, que puede hacerse por medio del comando *git clone https://github.com/scipion-em/scipion-em-kiharalab*, cuya rama más estable siempre es *master*, y la que suele contener los cambios más recientes *devel*. Es posible que existan otras ramas con desarrollos en proceso que, si bien pueden ser utilizadas para la instalación, no se recomienda al usuario promedio, ya que pueden tener problemas de inestabilidad.

Una vez está clonado el repositorio, se puede instalar en scipion con el comando *scipion3 installp -p <ruta_al_repositorio_del_plugin> --devel*, lo que instalará el plugin con todos sus protocolos y los binarios de éstos que se instalen de manera automática con sus protocolos, según haya sido definido por su desarrollador. Si los binarios de un protocolo no vienen incluidos por defecto, éstos pueden instalarse con el comando *scipion3 installb <nombre_de_binarios>*, donde el nombre los binarios puede verse en la lista de binarios, tanto instalados como sin instalar, disponible con el comando *scipion3 installb*.