Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención en Ingeniería de Software

Desarrollo de una cadena de procesado con GRASP para gestionar datos de nefelómetro integrado IN102

Autor:

Mario Cobreros del Caz

Tutores:

Mario Corrales Astorgano

Juan Carlos Antuña-Sánchez

"The universe is, and we are"

Agradecimientos

Quiero dar las gracias a mis tutores, Mario y Juan Carlos, por haberme acompañado y guiado a lo largo de este proyecto.

También quiero agradecer mucho a mi familia y a las personas que tengo más cerca de mi. Por su paciencia, su apoyo incondicional y por estar ahí durante lo bueno y lo malo de estos últimos años. Sin ellos, no habría sido posible.

Y, por supuesto, gracias a mis compañeros de GRASP por su preocupación, consejos y por estar siempre dispuestos a echar una mano. Trabajar con vosotros ha hecho que este proyecto sea mucho más llevadero y enriquecedor.

Resumen

El monitoreo de la calidad del aire es fundamental para la protección de la salud pública y el medio ambiente, permitiendo identificar y cuantificar contaminantes atmosféricos perjudiciales como el material particulado ($PM_{2,5}$, PM_{10}). La nefelometría juega un papel significativo en este campo al proporcionar mediciones directas de la dispersión de la luz causada por partículas suspendidas en la atmósfera, ofreciendo datos en tiempo real que complementan los métodos gravimétricos tradicionales.

A pesar de su valor, el análisis de estos datos requiere algoritmos avanzados como GRASP (Generalized Retrieval of Aerosol and Surface Properties), que puede transformar mediciones de dispersión en información detallada sobre propiedades físicas y ópticas de los aerosoles. Sin embargo, los investigadores del campo de la física enfrentan desafíos técnicos como la complejidad del procesamiento, la transformación de los datos y la necesidad de disponer de interfaces intuitivas.

Buscando dar una respuesta a esta necesidad, este proyecto desarrolla una cadena de procesado que integra el nefelómetro IN102 con el algoritmo GRASP, creando una solución software completa que automatiza la adquisición, procesamiento y visualización de datos atmosféricos. El sistema implementa una arquitectura Modelo-Vista-Controlador utilizando Python, Qt para la interfaz gráfica, Docker para garantizar portabilidad multiplataforma, y Flask para implementar una API REST que permite el procesamiento tanto local como remoto.

La metodología iterativa e incremental, implementada en este proyecto, con participación activa de usuarios científicos internacionales, ha permitido desarrollar una herramienta que responde a necesidades reales: integración completa con hardware especializado, visualizaciones científicas avanzadas con Plotly, interfaz intuitiva y arquitectura modular extensible. El resultado es una solución efectiva que facilita la investigación en calidad del aire, permitiendo a los científicos centrarse en el análisis de resultados en lugar de en diversas complejidades técnicas.

Abstract

Air quality monitoring is crucial for safeguarding public health and the environment by identifying and quantifying harmful atmospheric pollutants such as particulate matter ($PM_{2,5}$, PM_{10}). Nephelometry plays a significant role by directly measuring light scattering from suspended atmospheric particles, providing real-time data that complements traditional gravimetric methods.

Despite its value, analyzing this data requires advanced algorithms like GRASP (Generalized Retrieval of Aerosol and Surface Properties), which can transform scattering measurements into detailed information about aerosol physical and optical properties. However, researchers in physics face technical challenges such as processing complexity, data transformation, and the need for intuitive interfaces.

Addressing this need, this project develops a processing chain that integrates the IN102 nephelometer with the GRASP algorithm, creating a comprehensive software solution that automates the acquisition, processing, and visualization of atmospheric data. The system implements a Model-View-Controller architecture using Python, Qt for the graphical interface, Docker for cross-platform portability, and Flask to implement a REST API enabling both local and remote processing.

The iterative and incremental methodology, implemented with active participation from international scientific users, has enabled the development of a tool that meets real-world needs: complete integration with specialized hardware, advanced scientific visualizations with Plotly, an intuitive interface, and an extensible modular architecture. The result is an effective solution that facilitates air quality research, allowing scientists to focus on results analysis rather than technical complexities.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1 Contexto

El monitoreo de la calidad del aire es fundamental para la protección de la salud pública y el medio ambiente. Permite identificar y cuantificar los niveles de contaminantes atmosféricos perjudiciales, como el material particulado ($PM_{2,5}$, PM_{10}), el ozono troposférico (O_3), los óxidos de nitrógeno (NO_x) y el dióxido de azufre (SO_2). Esta información es crucial para entender la exposición de la población a aire de mala calidad, que puede causar o agravar distintas enfermedades y afectar especialmente a grupos vulnerables como niños, ancianos y personas con patologías previas [1]. Además, los datos recogidos son fundamentales para evaluar la efectividad de las políticas públicas, desarrollar nuevas estrategias de mitigación, informar al público sobre los riesgos y permitir la toma de decisiones basadas en evidencia científica para mejorar la calidad de vida y proteger los ecosistemas.

La nefelometría juega un papel significativo en el estudio de la calidad del aire [2][3] al proporcionar una medida directa de la dispersión de la luz causada por las partículas suspendidas en la atmósfera. Esta técnica es particularmente valiosa porque la cantidad de luz dispersada está directamente relacionada con la concentración, tamaño y forma de las partículas, especialmente aquellas en el rango de tamaño del material particulado fino ($PM_{2,5}$) y grueso (PM_{10}), que son de gran preocupación para la salud pública y la visibilidad atmosférica. Los nefelómetros permiten obtener mediciones en tiempo real o casi real de la carga de partículas, lo que complementa a los métodos gravimétricos tradicionales (que requieren muestreo y análisis en laboratorio) y facilita el seguimiento de las variaciones rápidas en los niveles de contaminación, la identificación de fuentes de emisión y la evaluación de la eficacia de las medidas de control de la contaminación atmosférica.

Si bien la nefelometría ofrece mediciones valiosas y en tiempo real del coeficiente de dispersión total de la luz por partículas, que es un indicador clave de la carga de aerosoles y la visibilidad, a menudo se requiere un análisis más profundo para caracterizar completamente las propiedades físicas y ópticas de estas partículas. Aquí es donde entran en juego algoritmos avanzados como GRASP (Generalized Retrieval of Aerosol and Surface Properties). Aunque GRASP está diseñado principalmente para invertir mediciones más complejas de teledetección (como las radiancias y polarizaciones medidas por fotómetros solares o satélites), los principios físicos que utiliza para relacionar las propiedades ópticas de los aerosoles (como la dispersión) con sus propiedades microfísicas (distribución de tamaño, índice de refracción, forma) son fundamentales.

El algoritmo, aunque potente y versátil, requiere una infraestructura software que facilite su integración con instrumentos de medición como el nefelómetro IN102, la gestión y transformación de datos, y la visualización de resultados. Esta necesidad de infraestructura representa una oportunidad para aplicar principios de ingeniería de software en el desarrollo de soluciones que mejoren la accesibilidad y usabilidad de herramientas científicas complejas.

La integración de tecnologías modernas como Docker para la contenerización, Python para el desarrollo de drivers y procesamiento de datos, y Qt para interfaces gráficas, ofrece una oportunidad para crear una solución que permita aprovechar GRASP en entornos de investigación atmosférica.

1.2 Motivación

La motivación principal de este proyecto surge de la necesidad de mejorar la usabilidad del algoritmo GRASP, una herramienta fundamental en el campo de la investigación atmosférica, con el nefelómetro IN102 como instrumento de medición. A pesar de su potencial para transformar medidas de dispersión de luz en información valiosa sobre aerosoles atmosféricos, actualmente los investigadores se enfrentan a varios desafíos técnicos en su uso:

- La complejidad en el procesamiento de datos del nefelómetro integrado IN102, que requiere múltiples pasos y transformaciones.
- La necesidad de una interfaz más intuitiva que facilite el manejo de los datos y la visualización de resultados.
- La complejidad en la gestión de diferentes formatos de datos y su transformación para el procesamiento con GRASP.

Para abordar estos desafíos, se propone el desarrollo de una cadena de procesado que:

- Automatice y optimice el flujo de trabajo para el procesamiento de datos del nefelómetro.
- Implemente un driver dedicado para la conversión eficiente de datos al formato de entrada del algoritmo GRASP.
- Proporcione una interfaz gráfica intuitiva para la visualización y análisis de resultados.
- Garantice la reproducibilidad de los resultados y su uso en distintos sistemas operativos mediante contenedorización con Docker.

Este desarrollo tecnológico busca facilitar el trabajo de los investigadores mediante:

- La reducción de errores potenciales en la manipulación manual de datos.
- La automatización de procesos repetitivos de transformación de datos.
- La provisión de herramientas visuales para el análisis de resultados.
- La creación de un entorno de ejecución multiplataforma consistente y portable.

La implementación de esta solución software permitirá a los científicos centrarse en el análisis de los datos y la interpretación de resultados, en lugar de lidiar con las complejidades técnicas del procesamiento de datos y la ejecución del algoritmo. Además de proveer un producto cercano a tiempo real, que hasta el momento solo se generaba en un análisis posterior de los datos.

1.3 Objetivos

1.3.1. Objetivo principal

El foco de este trabajo de fin de grado, consiste en desarrollar una **cadena de procesado de datos de nefelometría**. El objetivo general puede definirse de la siguiente manera:

Diseñar e implementar un sistema automático y multiplataforma para el procesamiento de mediciones de scattering adquiridas a través de un nefelómetro integrado, utilizando el algoritmo GRASP para la obtención de productos relacionados con la calidad del aire y su visualización por parte de los usuarios.

1.3.2. Objetivos específicos

Para alcanzar el objetivo descrito, se establecen los siguientes objetivos específicos:

- Investigar y aplicar las tecnologías y herramientas necesarias para el proyecto, incluyendo:
  - Las especificaciones del nefelómetro IN102.
  - La estructura y requisitos de entrada y salida del algoritmo GRASP.
  - Los fundamentos y capacidades del framework Qt para Python.
  - Las funcionalidades y ventajas del uso de Docker.
- Implementar una cadena de procesado que incluya:
  - La conversión de los datos brutos del nefelómetro mediante un driver dedicado.
  - La integración y ejecución del algoritmo GRASP con los datos convertidos.
  - La transformación de los resultados procesados por GRASP en una representación gráfica.
- Diseñar y desarrollar una **interfaz gráfica interactiva e intuitiva** para la visualización y análisis de resultados por parte de un **usuario científico**.
- Configurar y optimizar un entorno basado en contenedores Docker que garantice la compatibilidad **multiplataforma**, la facilidad de despliegue y la reproducibilidad de los resultados.
- Validar el sistema mediante una **batería de pruebas**, además de pruebas con **usuarios científicos**, asegurando la precisión de los datos procesados y la eficiencia del sistema.
- Obtener **feedback** del usuario científico sobre la interfaz y su utilidad para su labor diaria con el fin de mejorar la experiencia de uso del sistema.

1.3.3. Objetivos académicos y personales

Además de los objetivos específicos, se establecen los siguientes objetivos personales y de formación académica:

- Profundizar en el uso del framework Qt para Python, desarrollando habilidades en la creación de interfaces gráficas interactivas.
- Dominar las tecnologías de contenedorización como Docker, explorando buenas prácticas y configuraciones que optimicen la portabilidad y despliegue del sistema.

- Familiarizarse con los principios de diseño modular y escalable, aplicados a proyectos científicos y técnicos.
- Ampliar el conocimiento en la integración de algoritmos científicos, como GRASP, en flujos de trabajo prácticos y funcionales.
- Realizar la memoria del TFG de una forma estructurada, concisa y explicativa, acorde a los requisitos establecidos por la universidad.
- Presentar los resultados del proyecto ante el tribunal, obteniendo una calificación acorde al esfuerzo y los conocimientos adquiridos durante su desarrollo.

1.4 Metodología

Para el desarrollo de este proyecto, se ha implementado una metodología **iterativa e incremental**, un enfoque que combina refinamiento progresivo con la entrega de funcionalidades incrementales. Esta decisión responde tanto a las características específicas del proyecto como a las tendencias actuales en desarrollo de software.

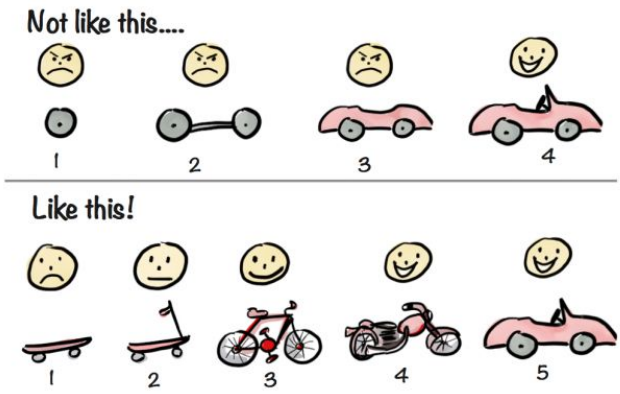


Figura 1.1: Metodología iterativa e incremental. Ilustración original de Henrik Kniberg [4].

La diferencia entre los enfoques iterativo, incremental y el combinado iterativo-incremental merece especial atención. Esta distinción, que ha inspirado el uso de esta metodología para este proyecto, fue estudiada por Jeff Patton en su artículo "Don't know what I want, but I know how to get it", donde ilustró los dos primeros enfoques usando el desarrollo de la Mona Lisa como ejemplo, tal y como se muestra en la figura 1.2 y 1.3.
Posteriormente, Steven Thomas en su artículo "Revisiting the Iterative Incremental Mona Lisa"[5] retomó este análisis y añadió una tercera perspectiva que combina ambos métodos tal y como se muestra en la figura 1.4.

Según Patton, citado por Thomas [5], el desarrollo iterativo "permite pasar de una idea vaga a su

Figura 1.2: Desarrollo iterativo. Ilustración original de Jeff Patton, adaptada por Steven Thomas [5].

realización. Iterar significa construir una versión preliminar, validarla y luego mejorar gradualmente su calidad".

Thomas señala que en este enfoque el alcance comienza de forma imprecisa y se va definiendo con el tiempo, pero el equipo aborda todo el alcance simultáneamente, lo que puede añadir riesgos considerables en la entrega de software.

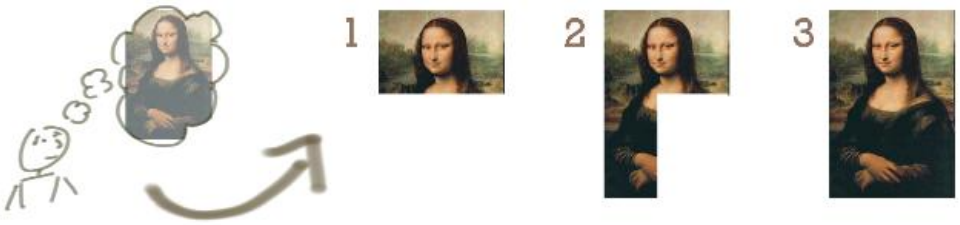


Figura 1.3: Desarrollo incremental. Ilustración original de Jeff Patton, adaptada por Steven Thomas [5].

Por otro lado, el desarrollo incremental, como recapitula Thomas [5] , "requiere una idea completamente formada que se construye poco a poco". Thomas destaca que, aunque "incremental" es el término más utilizado por los practicantes de Agile para describir lo que hacen, este enfoque puro sigue un estilo más cercano al desarrollo en cascada. El equipo conoce todo el alcance desde el inicio—solo posible con un análisis y diseño detallados previos—y luego comienza a construir por partes, en lo que en el mundo tradicional se conoce como entrega por fases.

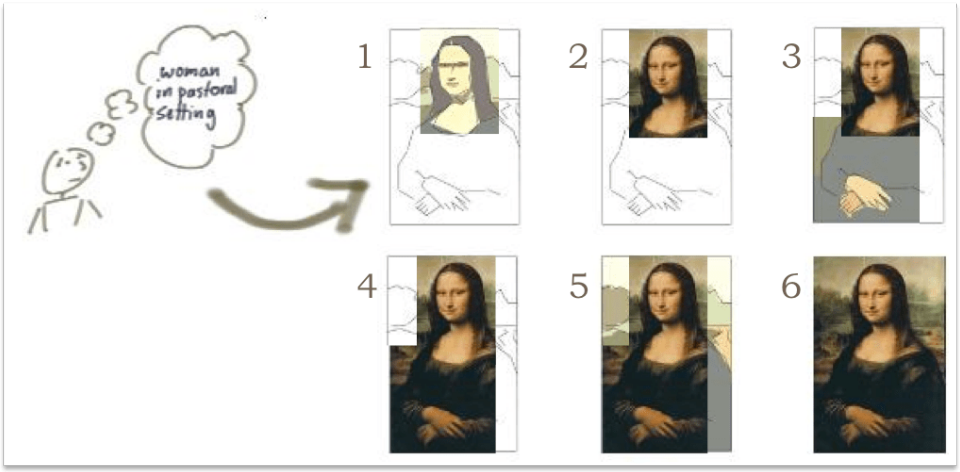


Figura 1.4: Desarrollo iterativo-incremental. Propuesta original de Steven Thomas, combinando los conceptos de Jeff Patton [5].

La aportación de Thomas [5] consiste en reconocer que, en realidad, la mayoría de los proyectos Agile combinan ambos enfoques y se convierten en iterativo-incrementales. Como él mismo explica: "Típicamente los equipos ágiles combinan ambos enfoques. Un equipo ágil realizará una iteración y producirá un incremento del producto. El incremento añade características completamente nuevas, basadas en historias de usuario, expandiendo el alcance de la funcionalidad—esto lo hace incremental. Pero cada incremento también refina la funcionalidad existente—lo que lo hace iterativo".

Esta combinación proporciona mayor potencia que cualquiera de los enfoques en aislamiento. Thomas enfatiza que el enfoque iterativo-incremental "da al propietario del producto, al equipo y a la empresa más opciones. Podemos detenernos en cualquier punto y descartar lo que hemos hecho o lanzarlo". Esta flexibilidad permite a los equipos construir menos pero también innovar constantemente, adaptándose a las prioridades cambiantes y permitiendo decisiones sobre la continuidad del proyecto en cualquier punto [5].

1.4.1. Fundamentos y justificación

El desarrollo iterativo e incremental constituye una aproximación al desarrollo de software que surgió como respuesta a las limitaciones del modelo tradicional en cascada [6]. Este enfoque integra dos principios fundamentales: la iteración, que permite refinar el producto mediante ciclos sucesivos, y el incremento, que facilita la construcción progresiva del sistema a través de módulos funcionales que aportan valor inmediato.

La elección de esta metodología se fundamenta en varios factores:

- **Restricciones temporales**: El desarrollo iterativo permite “entregar prototipos funcionales en

etapas tempranas del ciclo de vida del proyecto", un aspecto crucial considerando los plazos académicos establecidos. [7]

- **Incertidumbre en los requisitos**: Al tratarse de una aplicación orientada a la investigación científica, muchos requisitos evolucionan a medida que los usuarios interactúan con versiones preliminares del sistema, lo que hace especialmente valioso un enfoque que facilite la adaptación continua.

- **Complejidad de integración tecnológica**: La combinación de Docker, Python, QT y el algoritmo GRASP representa un desafío de integración que se beneficia de un desarrollo progresivo que permita abordar los riesgos técnicos de forma sistemática.

- **Naturaleza multidisciplinar del proyecto**: La colaboración entre expertos de diversas áreas (ciencias de la computación, física atmosférica, teledetección y ciencias ambientales) requiere un marco metodológico que facilite el intercambio constante de perspectivas y conocimientos.

1.4.2. Beneficios específicos para el proyecto

- **Visibilidad y validación temprana**: Este enfoque permite entregar software funcional desde etapas iniciales, facilitando la validación con usuarios científicos y la detección precoz de desviaciones conceptuales. [8]

- **Flexibilidad ante cambios**: La naturaleza iterativa permite incorporar modificaciones en los requisitos sin comprometer la integridad del proyecto, resultando en un producto final más alineado con las necesidades reales de los usuarios [9].

- **Aprendizaje continuo**: Cada iteración proporciona oportunidades de aprendizaje tanto sobre la tecnología como sobre las necesidades de los usuarios, lo que resulta especialmente valioso en contextos académicos y de investigación. [10]

- **Calidad general**: Las pruebas continuas y la retroalimentación temprana contribuyen a una mejor calidad global del software [7].

- **Fomento de la colaboración internacional**: Las reuniones periódicas estructuradas al final de cada iteración han permitido integrar las aportaciones de los colaboradores, enriqueciendo el proyecto con perspectivas diversas y conocimientos complementarios.

- **Integración de perspectivas multidisciplinarias**: El ciclo iterativo, con sus puntos de revisión sistemáticos, permite la incorporación efectiva de requisitos y consideraciones procedentes de diferentes disciplinas científicas.

1.4.3. Desafíos y estrategias de mitigación

- **Gestión de la complejidad creciente**: A medida que el sistema evoluciona, puede volverse más complejo [11]. La estrategia de mitigación incluye revisiones de código, modularización continua, refactorización sistemática y documentación actualizada.
- **Planificación adaptativa**: Se ha implementado una planificación por hitos con margen para ajustes, manteniendo siempre la fecha límite académica como referencia inamovible.

1.5 Estructura de la memoria

A continuación se describe la estructura de esta memoria, detallando brevemente el contenido de cada capítulo:

Capítulo 1 - Introducción: En este capítulo se presenta el contexto del proyecto, la motivación, los objetivos planteados y la metodología empleada.

Capítulo 2 - Estado del arte: Este capítulo presenta una descripción detallada del algoritmo GRASP, el estado actual del procesamiento de datos del nefelómetro, el análisis de soluciones existentes, las tendencias tecnológicas relevantes y las oportunidades de desarrollo identificadas.

Capítulo 3 - Planificación: Se detalla en este capítulo la planificación temporal seguida durante el desarrollo del proyecto, el plan de riesgos considerado y el presupuesto estimado.

Capítulo 4 - Descripción de las iteraciones: En este capítulo se describen las diferentes iteraciones realizadas durante el desarrollo del proyecto, desde la fase inicial hasta la fase final, detallando las actividades y logros de cada una.

Capítulo 5 - Análisis: En este capítulo se exponen los requisitos del sistema, los casos de uso, el modelo de dominio y el modelo de proceso definidos para el proyecto.

Capítulo 6 - Diseño: Este capítulo presenta el diseño adoptado para la aplicación, incluyendo arquitectura, patrones y decisiones de diseño relevantes.

Capítulo 7 - Implementación: Se comentan las tecnologías utilizadas, los detalles más relevantes de la implementación del sistema y la estructura final del proyecto.

Capítulo 8 - Pruebas: Se detallan en este capítulo las estrategias y tipos de pruebas realizadas para garantizar la calidad del software, incluyendo pruebas de integración, de sistema y de aceptación, junto con sus resultados.

Capítulo 9 - Conclusiones y líneas futuras: En este capítulo se presentan las conclusiones obtenidas tras la realización del proyecto, los objetivos cumplidos, las lecciones aprendidas y se proponen posibles líneas de trabajo futuro.

Adicionalmente, se incluyen varios apéndices con información complementaria como acrónimos, manuales de despliegue y uso, así como la descripción del contenido entregado.

Capítulo 2

Estado del arte

2.1 Descripción del algoritmo GRASP

GRASP (Generalized Retrieval of Aerosol and Surface Properties) [12][13], es un sofisticado algoritmo de inversión diseñado para determinar las propiedades ópticas y microfísicas de los aerosoles atmosféricos, así como las propiedades ópticas de la superficie terrestre, a partir de diversas mediciones de teledetección.

Este algoritmo se distingue por varias características clave que lo hacen una herramienta poderosa y ampliamente utilizada en la comunidad científica:

- **Versatilidad:** GRASP puede procesar una amplia gama de tipos de mediciones, tanto de forma independiente como combinada. Esto permite integrar información de diferentes instrumentos (como fotómetros solares, radiómetros satelitales, lidars, etc.) para obtener una caracterización más completa.
- **Flexibilidad:** Su diseño modular permite incorporar e intercambiar diferentes métodos numéricos, modelos físicos y librerías según las necesidades específicas del usuario o del tipo de datos a analizar.
- **Acceso Abierto:** GRASP es de código libre, abierto y gratuito, lo que fomenta su uso, modificación y desarrollo colaborativo por parte de la comunidad investigadora (disponible en `www.grasp-open.com`).

Fundamentalmente, GRASP se estructura en dos módulos principales (como se ilustra concep-

tualmente en la Figura 2.1):

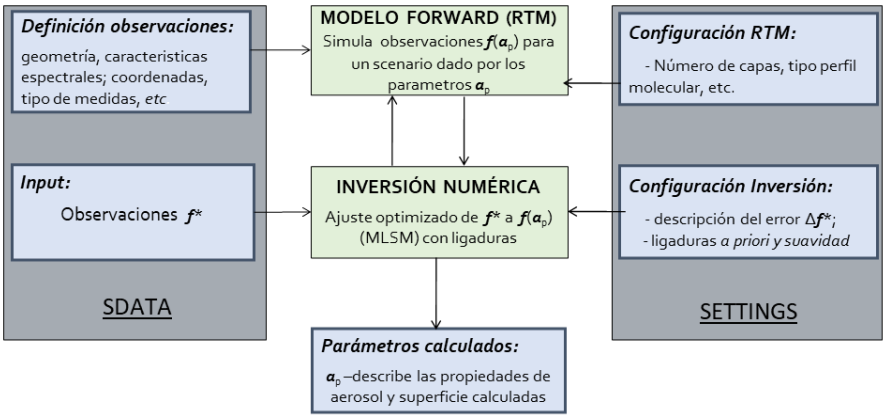


Figura 2.1: Estructura de GRASP [14].

1. **Módulo Forward:** Este componente simula las observaciones que un instrumento realizaría bajo un escenario atmosférico y de superficie específico, definido por el usuario mediante un conjunto de parámetros (vector de estado `ap`). Para ello, resuelve las ecuaciones de la teoría de la transferencia radiativa. Una ventaja importante es que este módulo puede utilizarse de forma independiente para simular diversas varibles (como el espesor óptico de aerosoles (AOD), radiancias del cielo, señales lidar, etc.) para cualquier longitud de onda y bajo cualquier escenario deseado (variando propiedades del aerosol, ángulo cenital solar (SZA), albedo de superficie, etc.), sin necesidad de ejecutar el proceso de inversión.

2. **Módulo de Inversión:** Este es el núcleo del algorítmo que ajusta iterativamente las propiedades del aerosol y/o la superficie (el vector de estado `ap`) hasta que las observaciones simuladas por el módulo directo coinciden de la mejor manera posible con las mediciones reales.

Esta estructura dual confiere a GRASP una gran potencia tanto para la simulación teórica de escenarios como para la extracción de información detallada a partir de observaciones reales.

2.2 Estado actual del procesamiento de datos del nefelómetro

El nefelómetro integrado IN102 genera datos en formato CSV que requieren una serie de transformaciones antes de poder ser procesados por GRASP. Actualmente, este proceso se realiza de forma manual o mediante scripts individuales, lo que puede resultar propenso a errores y poco eficiente. La conversión de estos datos a formato NetCDF, requerido por GRASP, representa un paso crítico que necesita ser automatizado y optimizado.

2.3 Análisis de soluciones existentes

En el ámbito del procesamiento de datos científicos, existen diversas herramientas y frameworks que abordan desafíos de procesamiento de datos, sin embargo, ninguno logra replicar la funcionalidad especializada que ofrece el algoritmo GRASP:

- **Frameworks de procesamiento científico:** Herramientas como SciPy y Pandas ofrecen capacidades robustas para el procesamiento de datos genéricos, pero carecen de los algoritmos específicos para la inversión de propiedades de aerosoles que GRASP implementa.
- **Interfaces gráficas:** Aplicaciones como Dash proporcionan capacidades de visualización científica, pero carecen de la robustez y versatilidad que ofrece Qt para el desarrollo de interfaces de usuario complejas con componentes interactivos avanzados.

2.4 Tendencias tecnológicas relevantes

Las tendencias actuales en el desarrollo de software científico apuntan hacia:

- **Contenedorización de aplicaciones científicas:** El uso creciente de Docker y tecnologías similares para garantizar la reproducibilidad de experimentos y análisis.
- **Interfaces gráficas modernas:** La evolución hacia frameworks como Qt que permiten el desarrollo de interfaces multiplataforma con capacidades avanzadas de visualización.
- **Automatización de procesos:** La tendencia hacia la creación de pipelines automatizados que reduzcan la intervención manual y los errores asociados.
- **Integración de formatos científicos:** El uso creciente de formatos estandarizados como NetCDF para el almacenamiento y procesamiento de datos científicos.

2.5 Limitaciones de las soluciones actuales

Las soluciones existentes presentan limitaciones específicas para el caso de uso del nefelómetro y GRASP:

- **Especificidad:** Las herramientas genéricas requieren considerable adaptación para manejar los formatos y requisitos específicos del nefelómetro IN102.

- **Curva de aprendizaje:** Las soluciones existentes suelen requerir conocimientos técnicos avanzados, limitando su accesibilidad para nuevos usuarios.
- **Integración:** La falta de soluciones que integren todos los componentes necesarios (preprocesamiento, ejecución de GRASP y visualización) en un único sistema.

2.6 Métodos tradicionales vs. nuevas tecnologías

Los métodos actuales para la medición de material particulado (PMx) se basan principalmente en dos técnicas [15]:

- **Métodos gravimétricos:** Considerados el estándar de referencia, se basan en la recolección de aerosoles en filtros que deben ser pesados en condiciones controladas de laboratorio. Si bien proporcionan mediciones precisas, presentan limitaciones significativas:
  - Requieren el análisis posterior en laboratorio, retrasando la obtención de resultados hasta 15-30 días.
  - Tienen una limitada resolución temporal.
  - Implican altos costes de material y mantenimiento.
- **Atenuación beta:** Aunque ofrece mejor resolución temporal, esta técnica:
  - Requiere el uso de material radiactivo.
  - Necesita procedimientos estrictos de seguridad y certificación.
  - Presenta limitaciones en condiciones de baja concentración de aerosoles.

La solución propuesta, basada en medidas de dispersión de luz y el algoritmo GRASP, permitiría superar estas limitaciones al ofrecer:

- Mediciones en tiempo real con alta resolución temporal.
- Procesamiento inmediato de datos sin necesidad de análisis en laboratorio.
- Menor coste de mantenimiento y operación.
- Mayor sensibilidad en condiciones de baja concentración de aerosoles.

2.7 Oportunidad de desarrollo

El análisis del estado actual revela la necesidad de una solución integrada que:

- Combine las mejores prácticas de desarrollo moderno con las necesidades específicas del procesamiento de datos de aerosoles.
- Proporcione una experiencia de usuario optimizada para investigadores no técnicos.
- Garantice la reproducibilidad y portabilidad de los análisis mediante tecnologías de contenedorización.
- Facilite la automatización del flujo completo de procesamiento de datos.

Capítulo 3

Planificación

3.1 Organización temporal

La planificación temporal del proyecto se ha estructurado en fases e iteraciones siguiendo un modelo de desarrollo iterativo e incremental. Este enfoque ha permitido dividir el trabajo en bloques manejables, facilitando la adaptación a los requisitos cambiantes y la validación continua con los usuarios científicos.

El proyecto ha tenido una duración total de aproximadamente 4 meses, con una dedicación de 300 horas distribuidas entre las diferentes fases. Cada iteración ha contado con objetivos específicos que han permitido evaluar el progreso de manera efectiva y realizar ajustes cuando ha sido necesario.

La tabla 3.1 presenta las fases y las iteraciones del proyecto junto a las tareas realizadas en cada una de ellas, así como el tiempo dedicado en días y horas. Por su parte, la figura 3.1 muestra el diagrama de Gantt que representa de una forma más visual la organización temporal del proyecto.

Fase / Iteración	**Fecha Inicio**	**Fecha Fin**	**Días**	**Horas**
Fase inicial: Análisis y planificación	8/01/2025	15/01/2025	8	25
Iteración 1: Base de la aplicación - Configuración del entorno de desarrollo - Estructura básica del proyecto - Interfaz mínima viable con QT	16/01/2025	23/01/2025	8	30
Iteración 2: Integración con GRASP - Configuración del entorno Docker - Integración del driver del nefelómetro - Transferencia de archivos y procesamiento local en el entorno Docker - Validación con usuario científico	24/01/2025	13/02/2025	21	60
Iteración 3: Procesamiento remoto - Integración de procesamiento local/remoto - Desarrollo de API REST con Flask - Implementación de procesamiento individual con tokens de sesión	14/02/2025	28/02/2025	15	50
Iteración 4: Visualización de resultados - Diseño de interfaz de visualización - Implementación con Plotly, HTML y CSS - Ajustes basados en feedback científico - Optimización de representaciones gráficas	01/03/2025	17/03/2025	17	50
Iteración 5: Integración final - Unificación de componentes - Implementación de flujo de trabajo completo - Conexión directa con el nefelómetro - Pruebas de sistema completo	18/03/2025	12/04/2025	26	60
Fase final: Documentación y entrega - Memoria final - Preparación de la presentación - Pruebas finales y ajustes	13/04/2025	29/04/2025	17	50
			Total horas:	**300**

Tabla 3.1: Organización temporal detallada del proyecto

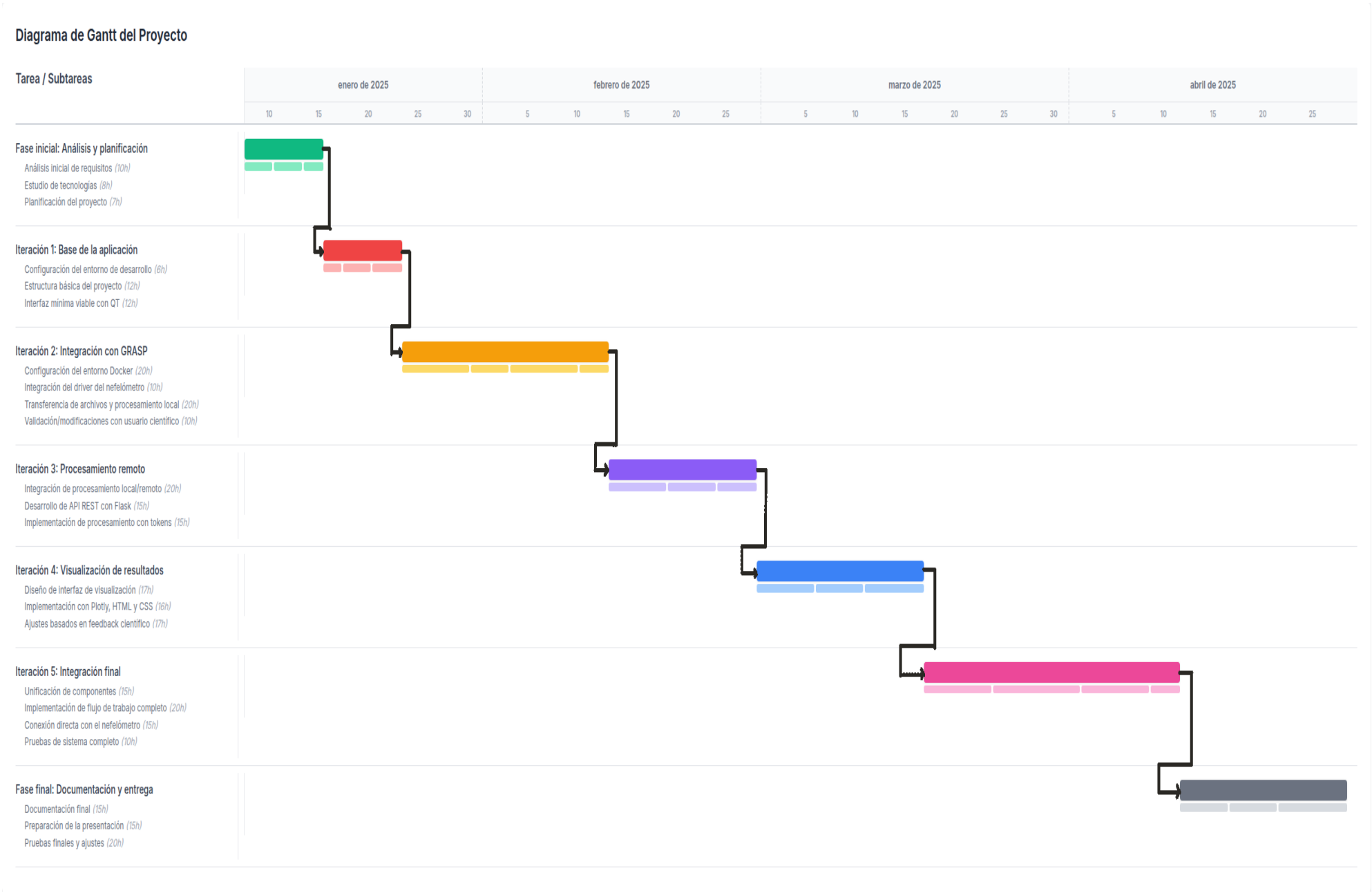


Figura 3.1: Diagrama de Gantt con la planificación temporal del proyecto. Elaboración Propia.

3.2 Plan de riesgos

En esta sección se analizan los distintos riesgos identificados para el proyecto y se presenta el plan de contigencia para gestionarlos adecuadamente.

Según el PM-BOK, un riesgo se define como "un evento o condición incierta que, si ocurre, tiene un efecto positivo o negativo sobre los objetivos del proyecto" [16]. Partiendo de esta definición, se ha realizado un análisis de los posibles riesgos que podrían afectar al desarrollo de este trabajo.

Se han conseguido identificar aproximadamente 15 riesgos potenciales que podrían afectar al desarrollo, los cuales están documentados detalladamente desde la tabla 3.3 hasta la tabla 3.16.
Esta documentación resultará útil para realizar un seguimiento continuo y mantener bajo control los riesgos durante todas las fases del proyecto.

Para la evaluación de estos riesgos, se ha utilizado la matriz que aparece en la tabla 3.2, que relaciona el nivel de impacto con la probabilidad de ocurrencia, permitiendo así determinar el grado de exposición a cada riesgo.

	Impacto		
Probabilidad	**Bajo**	**Medio**	**Alto**
Baja	Bajo	Bajo	Medio
Media	Bajo	Medio	Alto
Alta	Medio	Alto	Alto

Tabla 3.2: Matriz de Riesgos

A continuación se presentan las tablas detalladas de cada uno de los riesgos identificados, incluyendo su evaluación, impacto y las medidas de mitigación propuestas:

REGISTRO DE RIESGO			
ID Riesgo	RSK001	**Título**	Problemas de integración con GRASP
Categoría	Tecnología	**Estado**	Abierto
Descripción del riesgo			
Fallos en la lógica de negocio e inconsistencias funcionales.			
Amenaza			
Comportamiento inesperado del sistema y posibles errores críticos.			
Mitigación recomendada para el riesgo			
Pruebas iterativas y consultas al tutor.			
Acciones correctivas			
Revisión exhaustiva del código y documentación de los problemas encontrados.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Alto	Alto
Postmitigación	Media	Medio	Medio

Tabla 3.3: Registro de Riesgo - Problemas de integración con GRASP

REGISTRO DE RIESGO			
ID Riesgo	RSK002	**Título**	Falta de experiencia con QT
Categoría	Tecnología	**Estado**	Abierto
Descripción del riesgo			
Retrasos en el desarrollo del frontend e implementación incorrecta de componentes.			
Amenaza			
Implementación incorrecta de componentes y retrasos en el desarrollo.			
Mitigación recomendada para el riesgo			
Tutoriales iniciales y soporte online.			
Acciones correctivas			
Consultar documentación oficial y ejemplos de código.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Medio	Medio
Postmitigación	Baja	Medio	Bajo

Tabla 3.4: Registro de Riesgo - Falta de experiencia con QT

REGISTRO DE RIESGO			
ID Riesgo	RSK003	**Título**	Problemas con Docker en producción
Categoría	Tecnología	**Estado**	Abierto
Descripción del riesgo			
Imposibilidad de desplegar servicios o errores críticos en producción.			
Amenaza			
Fallos en el despliegue que impidan el funcionamiento del sistema.			
Mitigación recomendada para el riesgo			
Pruebas en entornos controlados.			
Acciones correctivas			
Mantener configuraciones de respaldo y documentar proceso de despliegue.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Baja	Alto	Medio
Postmitigación	Baja	Medio	Bajo

Tabla 3.5: Registro de Riesgo - Problemas con Docker en producción

REGISTRO DE RIESGO			
ID Riesgo	RSK004	**Título**	Retrasos por falta de tiempo
Categoría	Gestión	**Estado**	Abierto
Descripción del riesgo			
Posibles retrasos en el desarrollo del proyecto debido a una mala gestión del tiempo.			
Amenaza			
Incumplimiento de plazos y entrega incompleta del proyecto.			
Mitigación recomendada para el riesgo			
Planificación ajustada, constante revisión y establecimiento de márgenes temporales en caso de imprevistos.			
Acciones correctivas			
Priorizar tareas críticas y ajustar el alcance si es necesario.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Alto	Alto
Postmitigación	Media	Medio	Medio

Tabla 3.6: Registro de Riesgo - Retrasos por falta de tiempo

REGISTRO DE RIESGO			
ID Riesgo	RSK005	**Título**	Problemas con dependencias de librerías
Categoría	Tecnología	**Estado**	Abierto
Descripción del riesgo			
Incompatibilidades entre versiones de librerías y problemas de dependencias.			
Amenaza			
Fallos en tiempo de ejecución debido a versiones incompatibles.			
Mitigación recomendada para el riesgo			
Uso de versiones específicas y pruebas continuas.			
Acciones correctivas			
Mantener un registro de versiones compatibles y funcionales.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Medio	Medio
Postmitigación	Baja	Medio	Bajo

Tabla 3.7: Registro de Riesgo - Problemas con dependencias de librerías

REGISTRO DE RIESGO			
ID Riesgo	RSK006	**Título**	Errores en los datos de prueba
Categoría	Calidad	**Estado**	Abierto
Descripción del riesgo			
Datos de prueba incorrectos o insuficientes para validar la funcionalidad.			
Amenaza			
Resultados de pruebas inválidos o procesamiento incorrecto de datos.			
Mitigación recomendada para el riesgo			
Verificación manual de datos y uso de datos reales cuando sea posible.			
Acciones correctivas			
Crear conjuntos de datos de prueba verificados.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Medio	Medio
Postmitigación	Baja	Medio	Bajo

Tabla 3.8: Registro de Riesgo - Errores en los datos de prueba

REGISTRO DE RIESGO			
ID Riesgo	RSK007	**Título**	Errores en la planificación de recursos
Categoría	Gestión	**Estado**	Abierto
Descripción del riesgo			
Recursos insuficientes para completar las tareas a tiempo.			
Amenaza			
Retrasos significativos en el desarrollo del proyecto.			
Mitigación recomendada para el riesgo			
Revisión periódica de recursos y necesidades.			
Acciones correctivas			
Ajustar la asignación de recursos y priorizar tareas críticas.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Alto	Alto
Postmitigación	Media	Medio	Medio

Tabla 3.9: Registro de Riesgo - Errores en la planificación de recursos

REGISTRO DE RIESGO			
ID Riesgo	RSK008	**Título**	Errores en la estimación de tiempo
Categoría	Gestión	**Estado**	Abierto
Descripción del riesgo			
Imposibilidad de completar las tareas a tiempo con sus correspondientes retrasos.			
Amenaza			
Incumplimiento de plazos y entrega incompleta.			
Mitigación recomendada para el riesgo			
Revisión periódica del cronograma para hacer los ajustes necesarios.			
Acciones correctivas			
Reajustar plazos y priorizar funcionalidades críticas.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Alto	Alto
Postmitigación	Media	Medio	Medio

Tabla 3.10: Registro de Riesgo - Errores en la estimación de tiempo

REGISTRO DE RIESGO			
ID Riesgo	RSK009	**Título**	Problemas con herramientas de desarrollo
Categoría	Tecnología	**Estado**	Abierto
Descripción del riesgo			
Retrasos debido a configuración defectuosa o herramientas no compatibles.			
Amenaza			
Pérdida de tiempo en configuración y resolución de problemas técnicos.			
Mitigación recomendada para el riesgo			
Evaluar herramientas antes de usarlas.			
Acciones correctivas			
Mantener configuraciones alternativas.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Baja	Medio	Medio
Postmitigación	Baja	Bajo	Bajo

Tabla 3.11: Registro de Riesgo - Problemas con herramientas de desarrollo

REGISTRO DE RIESGO			
ID Riesgo	RSK010	**Título**	Realización de tareas adicionales al TFG
Categoría	Gestión	**Estado**	Abierto
Descripción del riesgo			
Se tiene que parar el transcurso del TFG debido a la asignación de tareas ajenas al TFG.			
Amenaza			
Retrasos significativos en el desarrollo del TFG.			
Mitigación recomendada para el riesgo			
Planificar las tareas adicionales de la empresa y reestructurar el plan de proyecto cuanto antes si es necesario.			
Acciones correctivas			
Negociar con supervisores y ajustar cronograma del TFG.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Alta	Medio	Alto
Postmitigación	Media	Medio	Medio

Tabla 3.12: Registro de Riesgo - Realización de tareas adicionales al TFG

REGISTRO DE RIESGO			
ID Riesgo	RSK011	**Título**	Falta de familiarización con tecnologías
Categoría	Tecnología	**Estado**	Abierto
Descripción del riesgo			
Se requiere más tiempo en formación y aprendizaje, lo que genera retrasos.			
Amenaza			
Retrasos en el desarrollo y posibles errores de implementación.			
Mitigación recomendada para el riesgo			
Aprovechar los conocimientos y experiencias del resto de compañeros para coger soltura con las tecnologías lo antes posible.			
Acciones correctivas			
Dedicar tiempo extra a formación y buscar recursos de aprendizaje adicionales.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Medio	Medio
Postmitigación	Baja	Medio	Bajo

Tabla 3.13: Registro de Riesgo - Falta de familiarización con tecnologías

REGISTRO DE RIESGO			
ID Riesgo	RSK012	**Título**	Problemas de salud
Categoría	Personal	**Estado**	Abierto
Descripción del riesgo			
Los tiempos y fechas del proyecto se ven afectados si el desarrollador necesita reposo.			
Amenaza			
Retrasos en el desarrollo y posible incumplimiento de plazos.			
Mitigación recomendada para el riesgo			
Hacer una planificación del proyecto añadiendo holgura a las actividades.			
Acciones correctivas			
Ajustar cronograma y priorizar tareas críticas al reincorporarse.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Medio	Medio
Postmitigación	Media	Bajo	Bajo

Tabla 3.14: Registro de Riesgo - Problemas de salud

REGISTRO DE RIESGO			
ID Riesgo	RSK013	**Título**	Problemas de comunicación
Categoría	Gestión	**Estado**	Abierto
Descripción del riesgo			
Retrasos en la resolución de dudas o falta de claridad en los objetivos.			
Amenaza			
Malentendidos y decisiones incorrectas en el desarrollo.			
Mitigación recomendada para el riesgo			
Establecer reuniones regulares y un canal de comunicación claro.			
Acciones correctivas			
Programar reuniones adicionales para aclarar malentendidos cuando sea necesario.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Medio	Medio
Postmitigación	Baja	Medio	Bajo

Tabla 3.15: Registro de Riesgo - Problemas de comunicación

REGISTRO DE RIESGO			
ID Riesgo	RSK015	**Título**	Cambios en los requisitos
Categoría	Gestión	**Estado**	Abierto
Descripción del riesgo			
Necesidad de ajustar los objetivos del proyecto, impactando en los plazos.			
Amenaza			
Retrasos y posible necesidad de rehacer trabajo ya realizado.			
Mitigación recomendada para el riesgo			
Documentar requisitos al inicio y validar cada cambio con un cronograma.			
Acciones correctivas			
Evaluar impacto de cambios y negociar ajustes en plazos si es necesario.			
Valores de probabilidad e impacto			
	Probabilidad	**Impacto**	**Exposición al riesgo**
Premitigación	Media	Alto	Alto
Postmitigación	Media	Medio	Medio

Tabla 3.16: Registro de Riesgo - Cambios en los requisitos

3.3 Plan de Presupuestos

En este apartado se detalla la planificación económica del proyecto. La estimación presupuestaria incluye los costes de incorporación de un empleado (específicamente, un Ingeniero de Software Junior) y los costes de los recursos materiales usados en el desarrollo del proyecto.

3.3.1. Coste Humano

El puesto requerido para este proyecto ha sido de Ingeniero de Software Junior, el cual tiene un sueldo medio de `24000€` euros brutos al año [17]. Con un salario anual de `24000€` en Castilla y León, se aplican `2517.60€` euros de retención por IRPF (`10.49 %`) y `1524.00€` euros de cuotas a la Seguridad Social. Es decir, el salario neto será de `19958.40€` euros al año (descontando IRPF y Seguridad Social), que se traducen en `1663.20€` euros al mes.

Dado que en España la jornada laboral es de 40 horas semanales o 160 horas mensuales, el sueldo por hora se calcula como:

$$\frac{\texttt{1663.20€}}{160} = \texttt{10.40€}\ €/\text{hora} \tag{3.1}$$

El proyecto ha durado `300` horas, por lo que el coste humano es de `3120€` euros.

3.3.2. Hardware, Software y Otros

Respecto al hardware usado para desarrollar el proyecto, se ha utilizado un ordenador portátil personal HP ENVY 13-ba1000 cuyo precio es de 1100 euros. La vida útil media de un portátil es aproximadamente 4 años (48 meses)[18]. El coste mensual es de:

$$\frac{\texttt{1100}}{48} = \texttt{22.92}\ €/\text{mes} \tag{3.2}$$

Durante el proyecto (4 meses), el coste amortizado del ordenador ha sido de 91.68 euros.

En cuanto al software, no se ha utilizado ninguna licencia de pago, puesto que se ha utilizado software de código abierto.

3.3.3. Equipamiento Específico

Para el desarrollo y pruebas del proyecto se ha utilizado un nefelómetro IN102, fabricado por la compañía AirPhoton. Este instrumento tiene un coste aproximado de 33000 euros y una vida útil estimada de 10 años (120 meses). Durante el periodo de desarrollo del proyecto (4 meses), el coste amortizado del nefelómetro se calcula como:

$$\frac{33000}{120} = 275 \text{ €/mes} \times 4 \text{ meses} = 1100 \text{ euros} \tag{3.3}$$

Figura 3.2: Nefelómetro IN102 de AirPhoton [19]

3.3.4. Presupuesto Total

Concepto	Precio (sin amortizar)	Precio parcial	Horas	Precio total
Salario del desarrollador	—	10.40 euros/hora	300h	3120 euros
Equipo portatil	1100 euros	22.92 euros/mes	4 meses	91.68 euros
Nefelómetro IN102	33000 euros	275 euros/mes	4 meses	1100 euros
			TOTAL	**4311.68 euros**

Tabla 3.17: Resumen del presupuesto del proyecto

3.4 Comparación entre planificación inicial y ejecución real

La ejecución del proyecto ha seguido en gran medida la planificación temporal inicial, aunque se han producido algunas desviaciones y ajustes. A continuación, se presenta una comparación de las principales diferencias, aunque estas se explican con más detalle en el capitulo 4.

3.4.1. Desviaciones temporales por fase

- **Fase Inicial (25h planificadas, 25h reales):** El tiempo total coincidió con lo previsto, aunque con diferencias en las tareas:
  - El análisis de requisitos requirió 2 horas adicionales (12h vs 10h planificadas)
  - El estudio de tecnologías se completó en menos tiempo (4h vs 8h planificadas)
  - La planificación necesitó 2 horas extra (9h vs 7h planificadas)
- **Iteración 1 (30h planificadas, 30h reales):** Se alcanzó el objetivo temporal, con ajustes en las tareas:
  - La configuración del entorno fue más rápida (4h vs 6h planificadas)
  - La estructura básica requirió más tiempo (14h vs 12h planificadas)
  - La interfaz mínima se ajustó al tiempo previsto (12h)
- **Iteración 2 (60h planificadas, 60h reales):** El tiempo total se cumplió, con diferencias notables:
  - La configuración de Docker requirió 5 horas adicionales (25h vs 20h planificadas)
  - La integración del driver fue más rápida (8h vs 10h planificadas)
  - La transferencia de archivos necesitó 2 horas extra (22h vs 20h planificadas)
  - La validación se completó en menos tiempo (5h vs 10h planificadas)
- **Iteración 3 (50h planificadas, 50h reales):** Se logró el tiempo objetivo, con ajustes internos:
  - La integración del procesamiento requirió 2 horas adicionales (22h vs 20h planificadas)
  - El desarrollo de la API REST necesitó 3 horas extra (18h vs 15h planificadas)
  - La implementación de tokens se completó antes (10h vs 15h planificadas)
- **Iteración 4 (50h planificadas, 50h reales):** Se cumplió exactamente con la planificación en todas las tareas.

- **Iteración 5 (60h planificadas, 60h reales):** El tiempo total se alcanzó, con diferencias en las tareas:
  - La unificación de componentes requirió 3 horas adicionales (18h vs 15h planificadas)
  - El flujo de trabajo necesitó 2 horas extra (22h vs 20h planificadas)
  - La conexión con el nefelómetro fue más rápida (12h vs 15h planificadas)
  - Las pruebas se completaron antes (8h vs 10h planificadas)
- **Fase Final (50h planificadas, 50h reales):** Se cumplió el objetivo temporal, con ajustes en las tareas:
  - La documentación requirió 3 horas adicionales (18h vs 15h planificadas)
  - La preparación de la presentación fue más rápida (12h vs 15h planificadas)
  - Las pruebas finales se ajustaron al tiempo previsto (20h)

3.4.2. Análisis de riesgos materializados

Durante el desarrollo del proyecto se materializaron tres de los riesgos identificados inicialmente:

1. **RSK015 - Cambios en los requisitos:** Se produjo un cambio significativo durante la iteración 2, cuando surgió la necesidad de implementar un procesamiento remoto. Este cambio se gestionó adecuadamente gracias a la metodología iterativa adoptada, permitiendo incorporar la nueva funcionalidad sin impactar negativamente en los plazos finales.
2. **RSK005 - Problemas con dependencias de librerías:** Durante la iteración 2, surgieron dificultades con las dependencias en la configuración del entorno Docker, lo que provocó un retraso de 5 horas en esta tarea. Sin embargo, este retraso se compensó con otras tareas que requirieron menos tiempo del previsto.
3. **RSK006 - Errores en los datos de prueba:** Se manifestó durante la iteración 3, afectando principalmente a la integración del procesamiento local/remoto. Este riesgo se gestionó mediante la implementación de validaciones adicionales y la creación de conjuntos de datos de prueba más robustos.

3.4.3. Conclusiones de la comparativa

El análisis de la ejecución del proyecto frente a su planificación inicial revela varios aspectos positivos:

- La estimación global del proyecto fue precisa, manteniéndose las 300 horas planificadas.
- La metodología iterativa permitió absorber las desviaciones sin afectar al plazo final.
- Los riesgos materializados se gestionaron adecuadamente gracias a las medidas de mitigación previstas.
- Las variaciones en los tiempos de las tareas individuales se compensaron dentro de cada iteración.

Capítulo 4

Descripción de las iteraciones

4.0 Fase Inicial: Análisis y planificación (8/01/2025 - 15/01/2025)

A continuación se presenta el desglose detallado de las actividades realizadas en esta fase inicial, junto con los tiempos dedicados a cada tarea:

Tarea	**T. Estimado**	**T. Invertido**	**Estado**
Análisis inicial de requisitos	10h	12h	Completada
Estudio de tecnologías	8h	4h	Completada
Planificación del proyecto	7h	9h	Completada
TOTAL	25h	25h	**100 %**

Tabla 4.1: Resumen temporal de la fase inicial

La fase inicial del proyecto constituyó un período de preparación y análisis. Se llevó a cabo una investigación detallada de las necesidades del proyecto, incluyendo reuniones con los usuarios científicos del equipo al igual que con el equipo de desarrollo para comprender como especificar mejor los requerimientos. La planificación se estructuró cuidadosamente, estableciendo hitos claros y definiendo la metodología de desarrollo incremental e iterativa que se seguiría durante todo el proyecto.

4.1 Iteración 1: Base de la aplicación (16/01/2025 - 23/01/2025)

El siguiente cuadro resume el esfuerzo invertido en las distintas tareas de esta primera iteración:

Tarea	**T. Estimado**	**T. Invertido**	**Estado**
Configuración del entorno de desarrollo	6h	4h	Completada
Estructura básica del proyecto	12h	14h	Completada
Interfaz mínima viable con QT	12h	12h	Completada
TOTAL	30h	30h	**100 %**

Tabla 4.2: Resumen temporal de la primera iteración

La primera iteración se enfocó en establecer las bases técnicas del proyecto. La configuración del entorno de desarrollo requirió menos tiempo del esperado debido al trabajo previo realizado durante las prácticas de empresa. La estructura básica del proyecto se retrasó debido a una mala planificación de las tareas mientras que la interfaz mínima se implementó según lo planificado.

Un hito importante de esta iteración fue la reunión inicial mantenida con John Hall, COO de AirPhoton, en la que se discutieron las ideas preliminares para el desarrollo del software. Durante este encuentro, se estableció un diálogo sobre la visión del proyecto y las posibles necesidades específicas que la empresa tendría para la aplicación.

Durante esta iteración se estableció un entorno de desarrollo que incluye todas las herramientas y tecnologias necesarias para el proyecto. La estructura básica se diseñó siguiendo un planteamiento modular que facilitará el mantenimiento y la escalabilidad del software. La interfaz mínima implementada demostró la viabilidad del enfoque elegido para la interacción con el usuario.

Al finalizar la iteración, se contaba con un proyecto correctamente estructurado y una interfaz básica funcional que permitía la navegación y la interacción mínima con la aplicación.

4.2 Iteración 2: Integración con GRASP (24/01/2025 - 13/02/2025)

Las actividades desarrolladas durante esta iteración, junto con su distribución temporal, se detallan en el siguiente resumen:

Esta iteración se centró principalmente en la configuración del entorno Docker y el desarrollo del sistema de procesamiento de archivos. La configuración del entorno Docker requirió considerablemente más tiempo del previsto debido a la complejidad de la instalación y configuración del algoritmo GRASP para el procesamiento de datos en un entorno Docker.

Tarea	T. Estimado	T. Invertido	Estado
Configuración del entorno Docker	20h	25h	Completada
Integración del driver del nefelómetro	10h	8h	Completada
Transferencia de archivos y procesamiento local	20h	22h	Completada
Validación/modificaciones con usuario científico	10h	5h	Completada
TOTAL	60h	60h	**100 %**

Tabla 4.3: Resumen temporal de la segunda iteración

En cuanto al driver del nefelómetro, se implementó una versión previamente funcional que permite la transformación de datos brutos obtenidos del nefelómetro a un formato compatible con el algoritmo GRASP, en este caso el formato NetCDF (.nc). Las pruebas se realizaron con éxito, centrándose, en esta iteración, en la validez de los datos obtenidos tras el procesamiento.

Durante esta iteración se mantuvieron varias reuniones con el equipo de AirPhoton para discutir el progreso del software. En una de estas reuniones, surgió la idea de implementar un procesamiento remoto desde la propia aplicación, hosteando un servidor dedicado al procesamiento de datos en un sistema externo. Esta propuesta llevó a la planificación de una funcionalidad basada en una API REST con Flask, que se implementaría en la siguiente iteración.

La iteración concluyó con un entorno Docker completamente funcional y un sistema de transferencia de archivos, que juntos proporcionan la infraestructura necesaria para el procesamiento de datos. Esta base permite el desarrollo de funcionalidades avanzadas en las siguientes fases del proyecto, centrándose en la experiencia del usuario científico y en la ampliación de las capacidades del sistema.

4.3 Iteración 3: Procesamiento remoto (14/02/2025 - 28/02/2025)

La distribución del esfuerzo durante esta tercera iteración se refleja en el siguiente desglose temporal:

Tarea	T. Estimado	T. Invertido	Estado
Integración de procesamiento local/remoto	20h	22h	Completada
Desarrollo de API REST con Flask	15h	18h	Completada
Implementación de procesamiento con tokens	15h	10h	Completada
TOTAL	50h	50h	**100 %**

Tabla 4.4: Resumen temporal de la tercera iteración

La tercera iteración se enfocó en el desarrollo del sistema de procesamiento remoto. Cabe destacar que esta iteración no estaba contemplada en la idea inicial del proyecto, sino que surgió como resultado de las reuniones mantenidas con AirPhoton, donde se identificó esta funcionalidad como un añadido que podría implementarse sin retrasar gravemente el proyecto. La integración del procesamiento local

y remoto requirió más tiempo del estimado debido a la necesidad de reescribir varias partes del código tras la detección de diversos bugs durante las fases de prueba. El proceso de depuración fue complejo, requiriendo varias horas de revisión y corrección. De igual forma, el desarrollo de la API REST también superó el tiempo planificado, no solo por la implementación, sino también por la aparición de errores inesperados que obligaron a refactorizar secciones significativas del código. Sin embargo, la implementación del sistema de tokens se completó en menos tiempo del esperado.

Al finalizar la iteración, el sistema contaba con la posibilidad de una arquitectura distribuida funcional que permite el procesamiento local o remoto de los datos preprocesados del nefelómetro. La API REST proporciona una interfaz para la comunicación entre componentes, y el sistema de tokens garantiza el procesamiento individualizado y asincrono del sistema en caso de ser usado por múltiples usuarios.

4.4 Iteración 4: Visualización de resultados (01/03/2025 - 17/03/2025)

El esfuerzo dedicado a las distintas actividades de esta cuarta iteración queda reflejado en el siguiente cuadro:

Tarea	**T. Estimado**	**T. Invertido**	**Estado**
Diseño de interfaz de visualización	17h	17h	Completada
Implementación con Plotly, HTML y CSS	16h	16h	Completada
Ajustes basados en feedback científico	17h	17h	Completada
TOTAL	50h	50h	**100 %**

Tabla 4.5: Resumen temporal de la cuarta iteración

La cuarta iteración se centró en el desarrollo de la interfaz de visualización de datos. El diseño de la interfaz se ajustó al tiempo estimado, permitiendo crear visualizaciones específicas para los datos procesados por el sistema. La implementación con Plotly se completó según lo planificado gracias a la extensa documentación disponible en su web [20], mientras que los ajustes para una mejor representación y visualización de los datos se realizaron dentro del marco temporal establecido.

Durante esta fase se implementó un sistema completo de visualización utilizando Plotly, HTML y CSS. Las gráficas se diseñaron específicamente para representar los datos procesados del nefelómetro de manera clara y relevante. Se mantuvieron reuniones periódicas con un colaborador experto en el uso del nefelómetro y la interpretación de datos, cuyas sugerencias fueron fundamentales para adaptar las visualizaciones a las necesidades reales de análisis científico. Esta colaboración permitió incorporar elementos específicos en la interfaz de usuario y en las gráficas que facilitan la identificación de patrones relevantes y la interpretación correcta de las mediciones.

Al concluir la iteración, el sistema contaba con una interfaz de visualización adaptada a las necesidades específicas de los usuarios finales, incorporando las preferencias y recomendaciones de un usuario científico real.

4.5 Iteración 5: Integración final (18/03/2025 - 12/04/2025)

La siguiente tabla muestra la inversión temporal realizada en cada una de las actividades de esta quinta iteración:

Tarea	**T. Estimado**	**T. Invertido**	**Estado**
Unificación de componentes	15h	18h	Completada
Implementación de flujo de trabajo completo	20h	22h	Completada
Conexión directa con el nefelómetro	15h	12h	Completada
Pruebas de sistema completo	10h	8h	Completada
TOTAL	60h	60h	**100 %**

Tabla 4.6: Resumen temporal de la quinta iteración

La quinta iteración se dedicó a la integración final de todos los componentes del sistema. La unificación de componentes y la implementación del flujo de trabajo completo requirieron más tiempo del estimado debido a la necesidad de asegurar una integración perfecta entre todos los módulos, la cual no fue sencilla debido a las extensas configuraciones necesarias para la automatización de los componentes individuales. Sin embargo, la conexión directa con el nefelómetro y las pruebas del sistema se completaron antes de lo previsto gracias al correcto funcionamiento de los componentes individuales.

Durante esta fase se logró la integración exitosa de todos los módulos desarrollados en las iteraciones anteriores. El flujo de trabajo completo se implementó permitiendo una experiencia fluida desde la adquisición de datos hasta su visualización. La conexión directa con el nefelómetro se optimizó para garantizar una comunicación estable y fiable, añadiendo un sistema de logs y notificaciones para detectar posibles fallos en la conexión.

La iteración concluyó con un sistema completamente integrado y funcional. Las pruebas exhaustivas confirmaron la estabilidad del sistema y su capacidad para manejar el flujo completo de trabajo científico, desde la captura de datos hasta su análisis y visualización, cumpliendo con todos los requisitos establecidos inicialmente.

4.6 Fase Final: Documentación y entrega (13/04/2025 - 29/04/2025)

El esfuerzo dedicado a las actividades de cierre del proyecto se resume en el siguiente cuadro:

Tarea	**T. Estimado**	**T. Invertido**	**Estado**
Documentación final	15h	18h	Completada
Preparación de la presentación	15h	12h	Completada
Pruebas finales y ajustes	20h	20h	Completada
TOTAL	50h	50h	**100 %**

Tabla 4.7: Resumen temporal de la fase final

La fase final del proyecto se centró en las pruebas exhaustivas del sistema y la documentación para la entrega. Las pruebas finales y ajustes constituyeron una parte significativa del esfuerzo, dedicando tiempo a verificar el funcionamiento correcto de todos los componentes integrados y resolver los últimos bugs detectados. La documentación técnica requirió más tiempo del previsto, aunque se priorizaron las pruebas para garantizar la calidad del producto final.

Durante esta fase se realizaron pruebas intensivas del sistema en diferentes escenarios de uso, verificando su robustez, rendimiento y usabilidad en condiciones reales. Se identificaron y corrigieron varios problemas relacionados con la integración de componentes y la visualización de datos. Paralelamente, se completó la documentación del proyecto y la memoria del trabajo de fin de grado.

Un hito importante de esta fase fue la reunión mantenida con el CCO de AirPhoton para evaluar el estado final de la aplicación. Durante este encuentro se discutieron posibles mejoras y se planteó la utilización real de la aplicación en una próxima conferencia científica, como demostración de las capacidades tanto del instrumento como del software desarrollado. La fase final concluyó con un proyecto completamente validado y documentado, listo para su entrega y con perspectivas concretas de aplicación en entornos científicos reales.

Capítulo 5

Análisis

Este capítulo describe el análisis detallado del sistema, comenzando con la identificación de los requisitos funcionales y no funcionales, seguido por los casos de uso y el modelo de dominio.

5.1 Requisitos

El análisis de requisitos constituye una fase crucial en el desarrollo del sistema, pues define las funcionalidades que debe proporcionar el software y establece las restricciones bajo las cuales debe operar.

5.1.1. Requisitos Funcionales

Los requisitos funcionales describen las capacidades específicas que el sistema debe ofrecer:

1. **Procesamiento de datos del nefelómetro:**
   - **RF1.1:** El sistema debe permitir la recogida de datos desde el nefelómetro integrado IN102.
   - **RF1.2:** El sistema debe transformar los datos del nefelómetro en un formato compatible con el algoritmo GRASP.
2. **Ejecución del algoritmo GRASP:**
   - **RF2.1:** El sistema debe ejecutar el algoritmo GRASP con los datos procesados del nefelómetro.

- **RF2.2:** El sistema debe gestionar adecuadamente los errores en la ejecución del algoritmo.

3. **Visualización de resultados:**

   - **RF3.1:** El sistema debe presentar los resultados del procesamiento en formatos visuales apropiados (gráficos, tablas).
   - **RF3.2:** El sistema debe proporcionar herramientas para el visualizado interactivo de los resultados.
   - **RF3.3:** El sistema debe permitir la exportación de resultados en diversos formatos (CSV, imágenes).

4. **Cadena de procesado:**

   - **RF4.1:** El sistema debe implementar una cadena de procesado automatizada desde la ingesta de datos hasta la visualización de resultados.
   - **RF4.2:** El sistema debe permitir la ejecución de pasos individuales de la cadena de procesado.
   - **RF4.3:** El sistema debe proporcionar mecanismos para monitorizar el estado de cada etapa del procesamiento.

5. **Gestión de preferencias:**

   - **RF5.1:** El sistema debe guardar las preferencias y configuraciones personalizadas por usuario.

5.1.2. Requisitos No Funcionales

Los requisitos no funcionales especifican criterios que definen las cualidades y restricciones del sistema, más allá de su comportamiento funcional.

1. **Usabilidad:**

   - **RNF1.1:** La interfaz gráfica debe ser fácil de usar, con un manejo claro de errores, adaptándose a las necesidades específicas de los usuarios científicos.
   - **RNF1.2:** El sistema debe proporcionar retroalimentación clara sobre las operaciones realizadas.
   - **RNF1.3:** La interfaz debe estar diseñada siguiendo principios de experiencia de usuario.

2. **Rendimiento:**

- **RNF2.1:** El sistema debe procesar los datos del nefelómetro y ejecutar el algoritmo GRASP en un tiempo razonable según el tamaño de los datos (inferior a 1 minuto).
- **RNF2.2:** La visualización de resultados debe ser fluida y responsiva.

3. **Portabilidad:**

- **RNF3.1:** El sistema debe ser compatible con diferentes plataformas mediante el uso de contenedores Docker.
- **RNF3.2:** El despliegue debe ser simple y requerir una configuración mínima en el sistema host.

4. **Mantenibilidad:**

- **RNF4.1:** El código debe estar bien documentado para facilitar su mantenimiento.
- **RNF4.2:** El diseño debe ser modular para permitir futuras ampliaciones de funcionalidad.

5.1.3. Requisitos de Información

La tabla 5.1 presenta los requisitos de información identificados para el sistema. Estos requisitos definen los datos que el sistema necesita almacenar para su correcto funcionamiento.

ID	Nombre	Descripción
RI01	Datos del nefelómetro	El sistema debe almacenar los datos brutos obtenidos del nefelómetro IN102 a través de la conexión serial.
RI02	Resultados del algoritmo	El sistema debe almacenar los resultados generados por el algoritmo GRASP, incluyendo propiedades calculadas de interés para el usuario.
RI03	Resultados para visualización	El sistema debe almacenar un archivo recortado de los resultados del algoritmo GRASP con las propiedades utilizadas para visualización.
RI04	Configuración del usuario	El sistema debe almacenar preferencias del usuario relacionadas con el instrumento (baudrate, puerto) y procesamiento (directorios, URL para procesamiento remoto).
RI05	Registro de errores (logs)	El sistema debe mantener un registro de errores durante la ejecución, incluyendo tipo, timestamp, contexto y descripción.
RI06	Historial de procesamiento	El sistema debe almacenar un registro de operaciones realizadas.

Tabla 5.1: Requisitos de información

5.2 Casos de Uso

Los casos de uso describen las interacciones entre los usuarios y el sistema para alcanzar objetivos específicos. Cada caso de uso está relacionado con uno o más requisitos funcionales y no funcionales, lo que permite trazar cómo se implementan los requisitos de forma concreta.

5.2.1. Actores del Sistema

Los principales actores que interactuarán con el sistema son:

- **Usuario Científico:** Investigador o técnico que utiliza el sistema para procesar y analizar datos obtenidos a través de un nefelómetro.
- **Nefelómetro IN102:** Instrumento de medida que proporciona datos al sistema.
- **API GRASP:** Representa el servidor Flask que contiene la API que ejecuta el algoritmo GRASP.

5.2.2. Diagrama General de Casos de Uso

El diagrama de casos de uso proporciona una visión integral de la interacción entre los actores identificados y las funcionalidades del sistema. Esta representación visual permite comprender el alcance funcional del sistema y cómo los actores interactúan con el sistema. Como se puede observar en la Figura 5.1, el diagrama muestra los siete casos de uso principales del sistema y su relación con los tres actores identificados.

Figura 5.1: Diagrama general de casos de uso. Elaboración Propia.

5.2.3. Listado de Casos de Uso

A continuación, se presenta el listado completo de casos de uso identificados para el sistema, agrupados por su funcionalidad principal:

ID	Nombre del Caso de Uso	Requisitos Relacionados
CU-01	Cargar datos del nefelómetro	RF1.1, RNF1.1, RNF3.1, RI01, RI05, RI06
CU-02	Preprocesar datos para GRASP	RF1.2, RNF2.1, RI01, RI05, RI06
CU-03	Ejecutar algoritmo GRASP	RF2.1, RF2.2, RNF2.1, RNF2.2, RI02, RI05, RI06
CU-04	Visualizar resultados	RF3.1, RF3.2, RF3.3, RNF1.1, RNF2.2, RI02, RI03, RI05, RI06
CU-05	Ejecutar cadena de procesado completa	RF4.1, RF4.3, RNF2.1, RNF2.2, RI01, RI02, RI03, RI05, RI06
CU-06	Ejecutar paso individual de procesado	RF4.2, RF4.3, RNF1.2, RI01, RI02, RI03, RI05, RI06
CU-07	Gestionar preferencias de usuario	RF5.1, RNF1.1, RI04, RI05

Tabla 5.2: Casos de uso y requisitos relacionados

5.2.4. Descripción Detallada de Casos de Uso

A continuación se detallan todos los casos de uso del sistema en orden:

Título	CU-01. Cargar datos del nefelómetro
Actor	Usuario Científico, Nefelómetro IN102
Descripción	El actor carga datos del nefelómetro IN102 al sistema para su posterior procesamiento
Precondiciones	PRE-1. El nefelómetro IN102 debe estar conectado al ordenador.
Postcondiciones	POST-1. Los datos quedan cargados en el sistema. POST-2. El sistema valida el formato de los datos.
Flujo normal	1. El actor introduce los parametros de conexión al nefelómetro. 2. El actor inicia la conexión con el nefelómetro. 3. El sistema inicia la conexión serial con el nefelómetro. 4. El nefelómetro envía los datos al sistema. 5. El sistema carga y valida los datos. 6. El sistema confirma la carga exitosa y mantiene la conexión con el nefelómetro. 7. El sistema muestra los datos cargados y recibe periodicamente datos del nefelómetro.
Excepciones	3.1. Si la conexión con el dispositivo falla, el sistema notifica al usuario y finaliza la conexión. 5.1. Si los datos no son válidos, el sistema notifica el problema.
Requisitos relacionados	RF1.1, RNF1.1, RNF3.1, RI01, RI05, RI06

Tabla 5.3: CU-01. Cargar datos del nefelómetro

Título	CU-02. Preprocesar datos para GRASP
Actor	Usuario Científico
Descripción	El actor procesa los datos cargados del nefelómetro para prepararlos para el algoritmo GRASP
Precondiciones	PRE-1. Los datos del nefelómetro deben estar en el sistema.
Postcondiciones	POST-1. Los datos son procesados y listos para el algoritmo GRASP.
Flujo normal	1. El actor inicia el preprocesamiento de datos para GRASP. 2. El sistema permite seleccionar el archivo de datos. 3. El actor selecciona el archivo de datos a procesar e inicia la ejecución. 4. El sistema procesa los datos e indica el progreso. 5. El sistema notifica la finalización del preprocesamiento.
Excepciones	3.1. Si no hay datos cargados, el sistema notifica al actor. 4.1. Si el preprocesamiento falla, el sistema proporciona información sobre el error.
Requisitos relacionados	RF1.2, RNF2.1, RI01, RI05, RI06

Tabla 5.4: CU-02. Preprocesar datos para GRASP

Título	CU-03. Ejecutar algoritmo GRASP
Actor	Usuario Científico, API GRASP
Descripción	El actor ejecuta el algoritmo GRASP con los datos previamente procesados del nefelómetro
Precondiciones	PRE-1. Los datos deben estar preprocesados y listos para GRASP.
Postcondiciones	POST-1. El algoritmo GRASP se ejecuta correctamente. POST-2. Los resultados del algoritmo se almacenan en el sistema.
Flujo normal	1. El actor inicia la ejecución del algoritmo GRASP. 2. El sistema ofrece opciones para procesamiento local o remoto. 3. El actor selecciona procesamiento local. 4. El sistema solicita especificar directorios de entrada y salida. 5. El actor proporciona la ruta del directorio de entrada. 6. El actor proporciona la ruta del directorio de salida. 7. El actor inicia la ejecución del algoritmo. 8. El sistema se comunica con la API GRASP localmente. 9. La API GRASP ejecuta el algoritmo e indica el progreso. 10. El sistema notifica la finalización y muestra los resultados.
Flujos alternativos	3.1. El actor selecciona procesamiento remoto: 3.1.1. El sistema solicita la URL del servidor. 3.1.2. El actor proporciona la URL del servidor remoto. 3.1.3. El actor proporciona los directorios de entrada y salida como en los pasos 5 y 6. 3.1.4. El actor inicia la ejecución del algoritmo. 3.1.5. El sistema se comunica con la API GRASP remota para procesar los datos. 3.1.6. La API GRASP recibe los datos, ejecuta el algoritmo y devuelve los resultados. 3.1.7. El sistema notifica la finalización y muestra los resultados.
Excepciones	5.1. Si el directorio de entrada especificado no existe o no es válido, el sistema notifica el error. 6.1. Si el directorio de salida especificado no es válido, el sistema notifica el error. 8.1. Si la comunicación con la API GRASP local falla, el sistema proporciona detalles del error. 9.1. Si la ejecución del algoritmo falla, la API GRASP devuelve el error y el sistema lo notifica al actor. 3.1.5.1. Si la conexión con la API GRASP remota falla, el sistema notifica al actor. 3.1.6.1. Si la API GRASP remota devuelve un error, el sistema muestra el mensaje correspondiente al usuario.
Requisitos relacionados	RF2.1, RF2.2, RNF2.1, RNF2.2, RI02, RI05, RI06

Tabla 5.5: CU-03. Ejecutar algoritmo GRASP

Título	CU-04. Visualizar resultados
Actor	Usuario Científico
Descripción	El actor visualiza los resultados del procesamiento con el algoritmo GRASP
Precondiciones	PRE-1. El algoritmo GRASP debe haberse ejecutado correctamente.
Postcondiciones	POST-1. Los resultados se visualizan en el sistema.
Flujo normal	1. El actor accede a la visualización de resultados. 2. El sistema muestra los conjuntos de resultados disponibles. 3. El actor selecciona un conjunto de resultados. 4. El sistema genera y muestra gráficos interactivos correspondientes. 5. El actor puede interactuar con los gráficos (zoom, pan, selección de datos, mostrar/ocultar series).
Excepciones	4.1. Si la visualización no se puede generar, el sistema notifica el error.
Requisitos relacionados	RF3.1, RF3.2, RF3.3, RNF1.1, RNF2.2, RI02, RI03, RI05, RI06

Tabla 5.6: CU-04. Visualizar resultados

Título	CU-05. Ejecutar cadena de procesado completa
Actor	Usuario Científico
Descripción	El actor ejecuta la cadena completa de procesado, desde la carga de datos hasta la visualización de resultados
Precondiciones	PRE-1. Los datos del nefelómetro deben estar disponibles. PRE-2. Los parámetros de ejecución deben estar configurados.
Postcondiciones	POST-1. La cadena completa de procesado se ejecuta correctamente. POST-2. Los resultados finales quedan disponibles para su visualización.
Flujo normal	1. El actor inicia la ejecución de la cadena completa. 2. El sistema presenta las opciones de configuración disponibles. 3. El actor selecciona los datos de entrada y confirma los parámetros. 4. El sistema inicia el procesamiento secuencial: 4.1. Carga de datos. 4.2. Preprocesamiento para GRASP. 4.3. Ejecución del algoritmo GRASP. 4.4. Generación de visualizaciones. 5. El sistema indica el progreso de cada etapa. 6. El sistema notifica la finalización y presenta un resumen del proceso.
Excepciones	4.1. Si alguna etapa falla, el sistema detiene el proceso y proporciona información del error.
Requisitos relacionados	RF4.1, RF4.3, RNF2.1, RNF2.2, RI01, RI02, RI03, RI05, RI06

Tabla 5.7: CU-05. Ejecutar cadena de procesado completa

Título	CU-06. Ejecutar paso individual de procesado
Actor	Usuario Científico
Descripción	El actor ejecuta un paso específico del procesado de forma individual
Precondiciones	PRE-1. Deben cumplirse las precondiciones específicas del paso a ejecutar.
Postcondiciones	POST-1. El paso individual se ejecuta correctamente. POST-2. Los resultados del paso quedan disponibles en el sistema.
Flujo normal	1. El actor inicia la ejecución de un paso individual. 2. El sistema presenta las opciones de configuración para ese paso. 3. El actor configura los parámetros e inicia la ejecución. 4. El sistema ejecuta el paso e indica el progreso. 5. El sistema notifica la finalización y presenta los resultados.
Excepciones	4.1. Si no se cumplen las precondiciones, el sistema notifica al actor. 4.2. Si la ejecución falla, el sistema informa del error.
Requisitos relacionados	RF4.2, RF4.3, RNF1.2, RI01, RI02, RI03, RI05, RI06

Tabla 5.8: CU-06. Ejecutar paso individual de procesado

Título	CU-07. Gestionar preferencias de usuario
Actor	Usuario Científico
Descripción	El actor configura y gestiona sus preferencias relacionadas con el instrumento y el procesamiento de datos
Precondiciones	Ninguna
Postcondiciones	POST-1. Las preferencias del usuario quedan guardadas. POST-2. El sistema aplica las preferencias configuradas inmediatamente.
Flujo normal	1. El actor accede a la sección de preferencias. 2. El sistema presenta las opciones disponibles. 3. El actor modifica parámetros específicos. 4. El sistema valida cada cambio. 5. El sistema guarda cada preferencia modificada . 6. El sistema aplica los cambios a la sesión actual.
Excepciones	4.1. Si alguna configuración no es válida, el sistema notifica al actor. 5.1. Si el archivo de configuración no es accesible, el sistema notifica el error.
Requisitos relacionados	RF5.1, RNF1.1, RI04, RI05

Tabla 5.9: CU-07. Gestionar preferencias de usuario

5.2.5. Diagramas de Actividad

Los diagramas de actividad proporcionan una representación del comportamiento del sistema, permitiendo comprender claramente el flujo de ejecución de cada caso de uso y la interacción entre los actores y el sistema.

A continuación, se presentan los diagramas de actividad correspondientes a cada uno de los casos de uso detallados previamente. Cada diagrama representa gráficamente el flujo de trabajo específico, incluyendo las acciones realizadas por los actores, las respuestas del sistema y bifurcaciones que pueden ocurrir durante su ejecución.

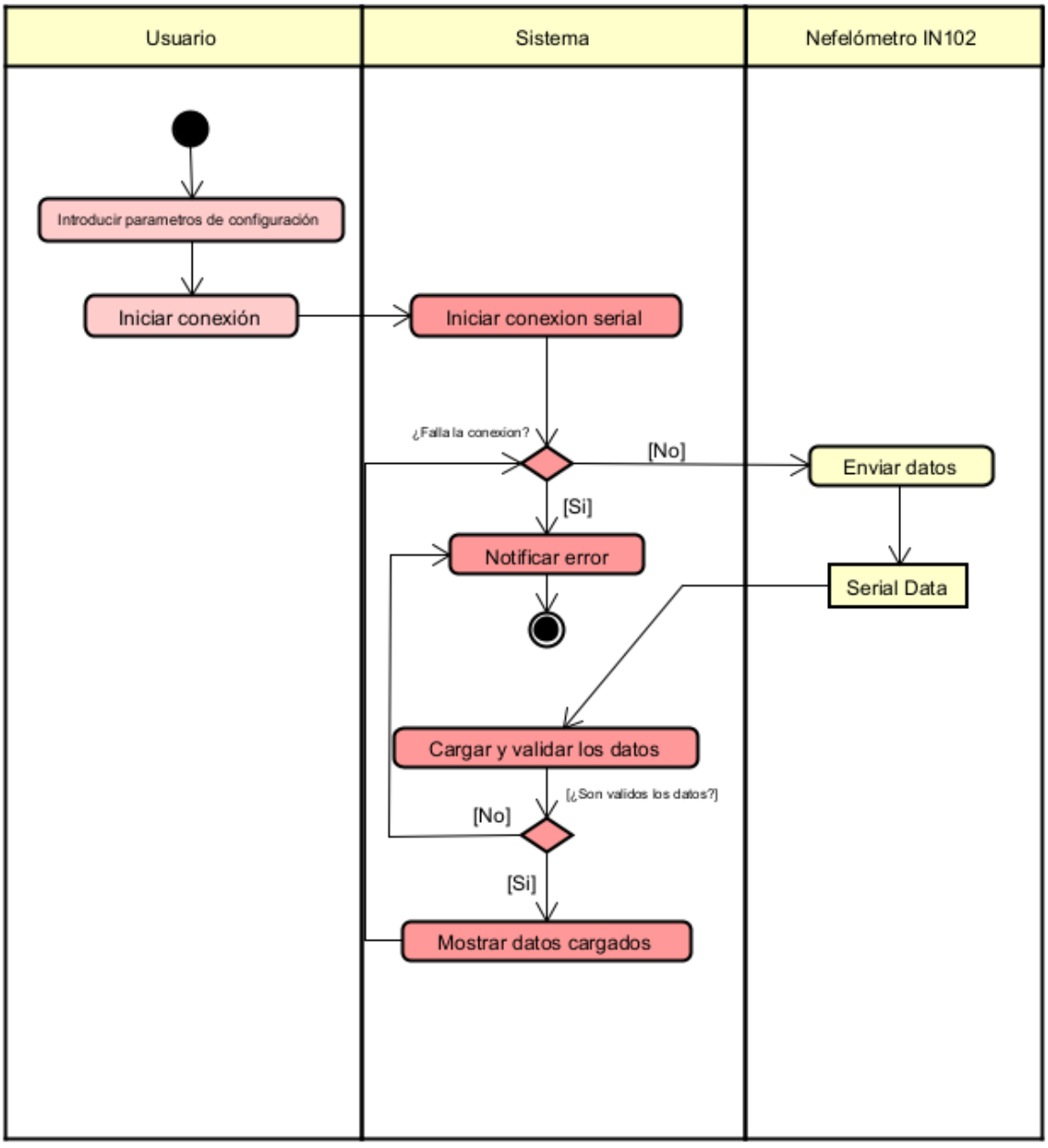


Figura 5.2: Diagrama de actividad: Cargar datos del nefelómetro (CU-01). Elaboración Propia.

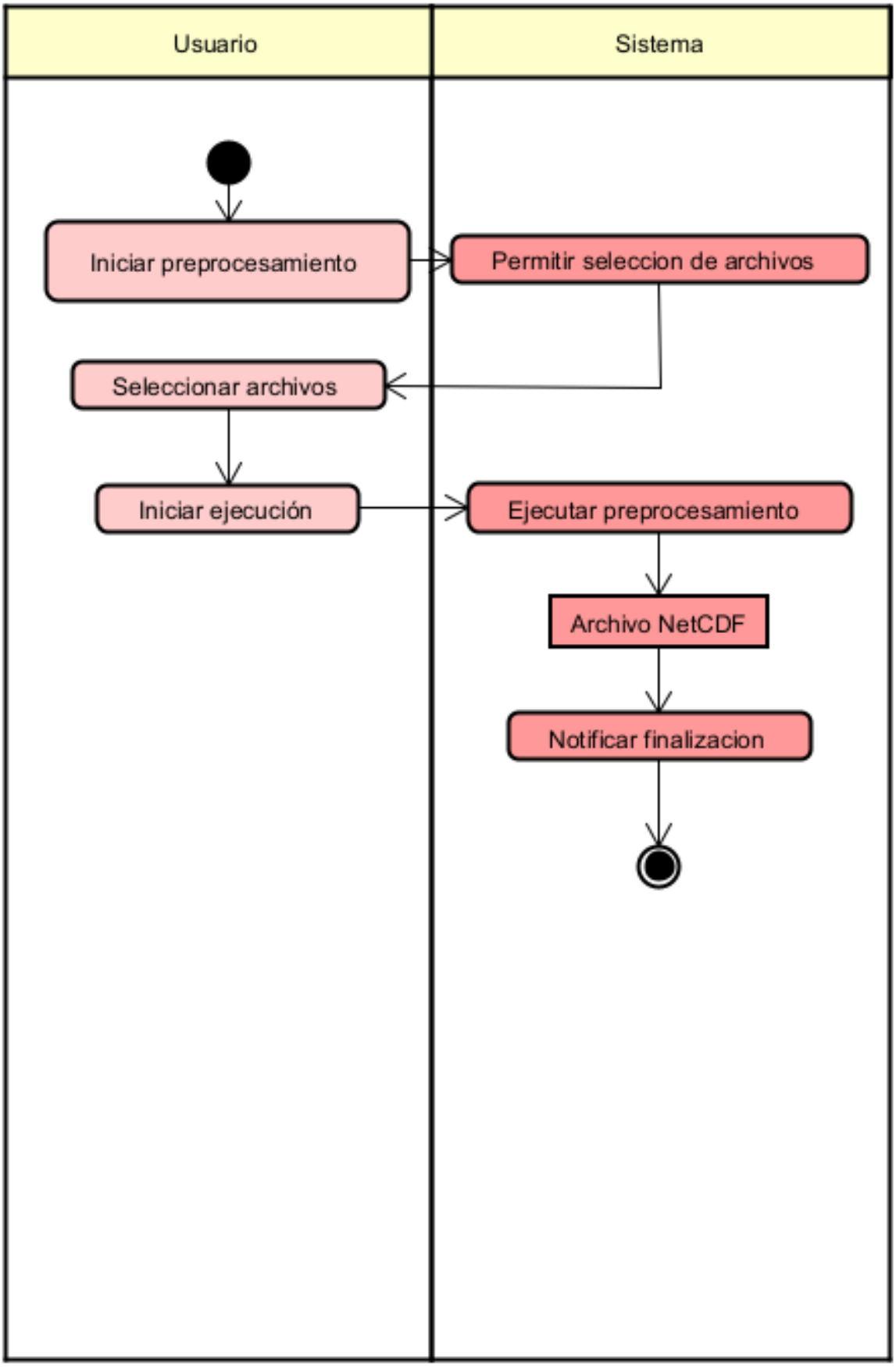


Figura 5.3: Diagrama de actividad: Preprocesar datos para GRASP (CU-02). Elaboración Propia.

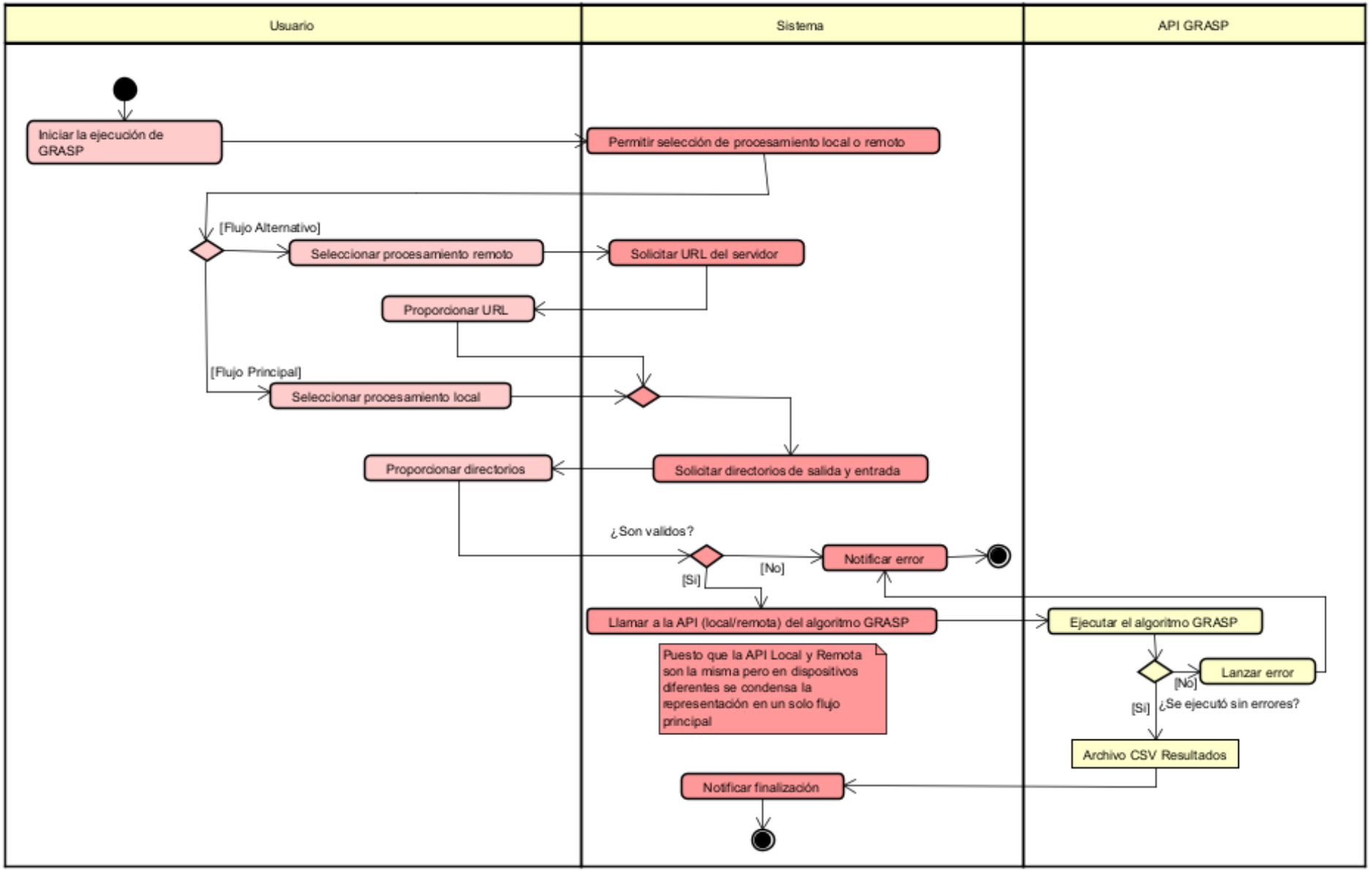


Figura 5.4: Diagrama de actividad: Ejecutar algoritmo GRASP (CU-03). Elaboración Propia.

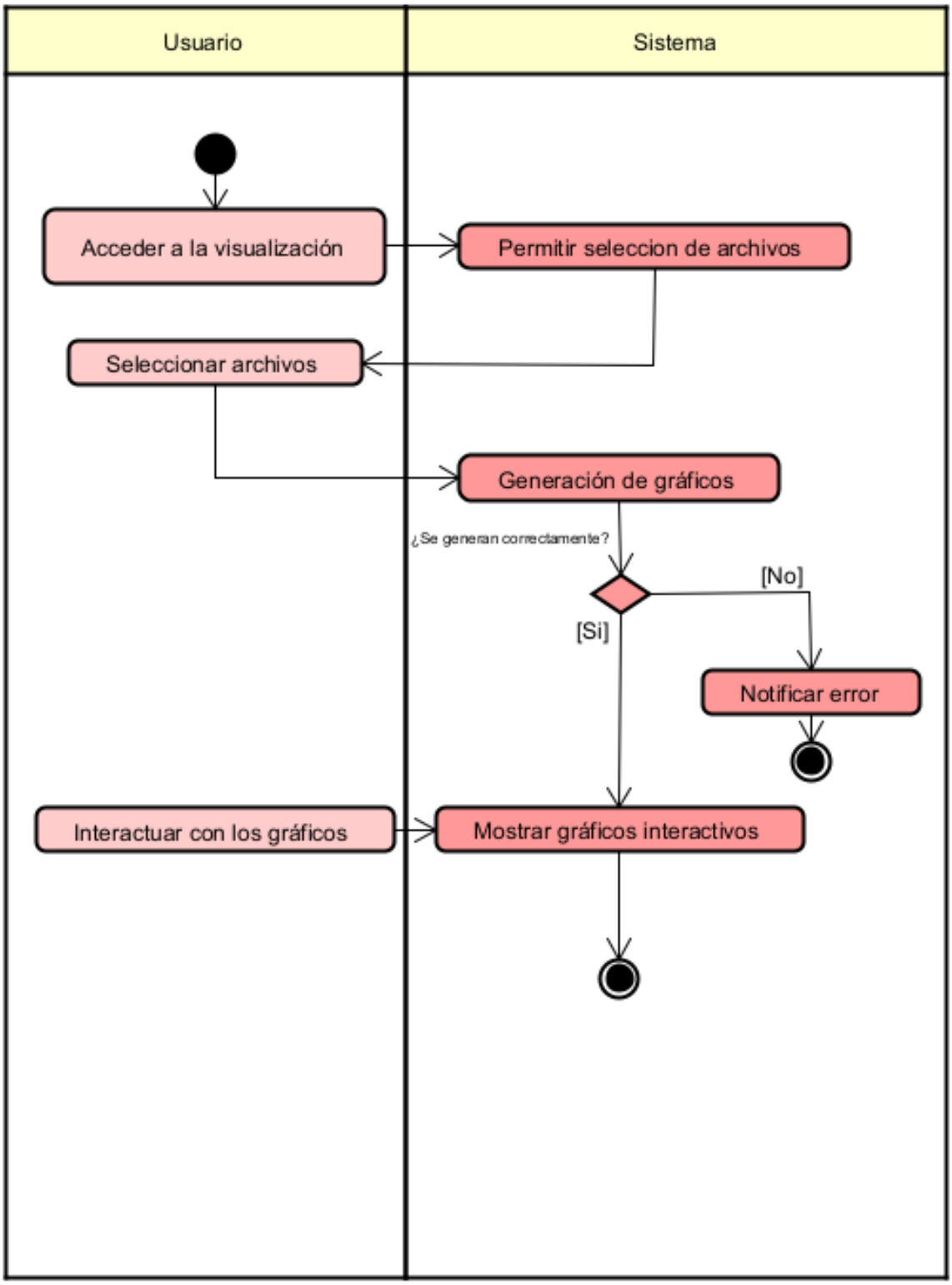


Figura 5.5: Diagrama de actividad: Visualizar resultados (CU-04). Elaboración Propia.

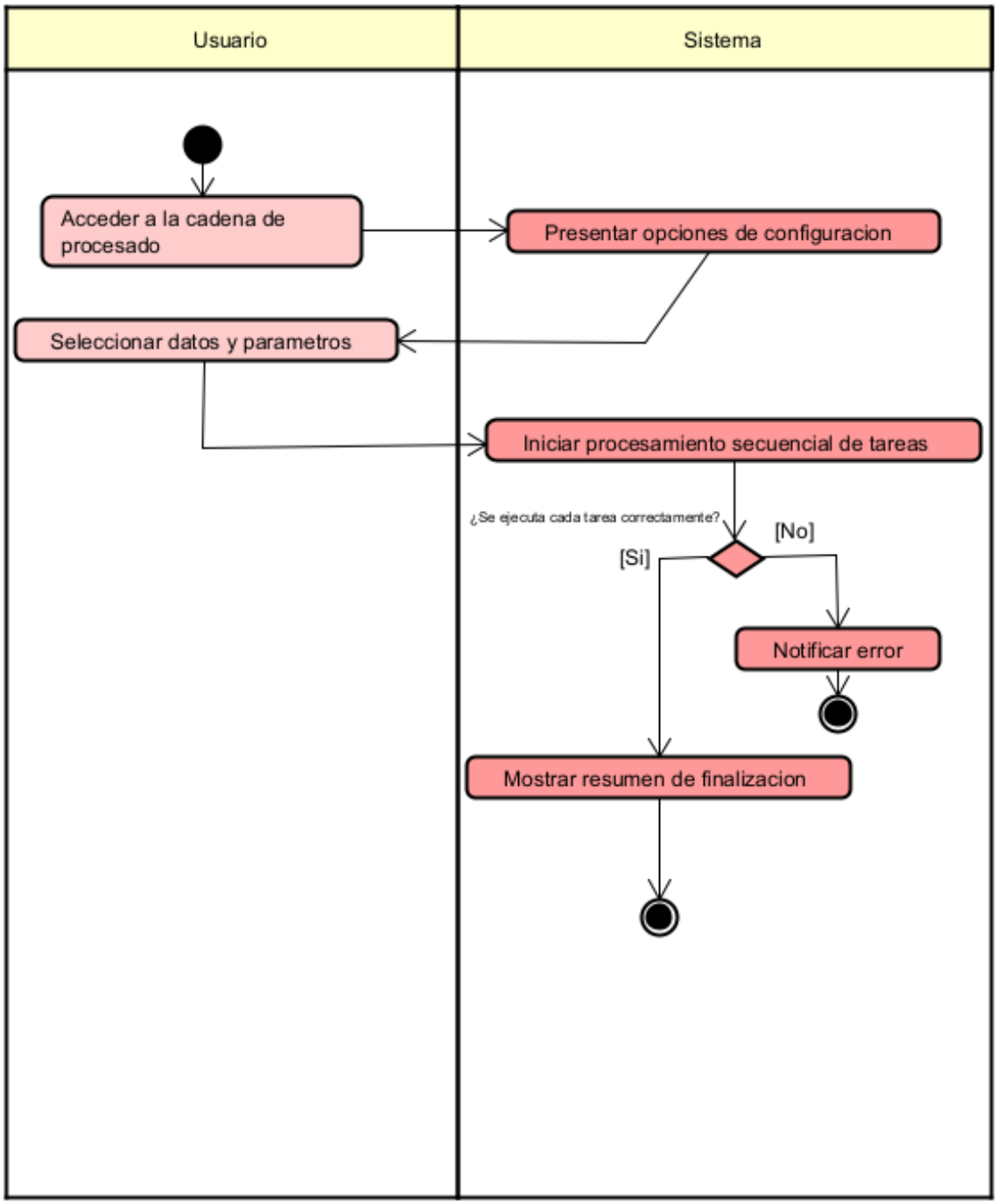


Figura 5.6: Diagrama de actividad: Ejecutar cadena de procesado completa (CU-05). Elaboración Propia.

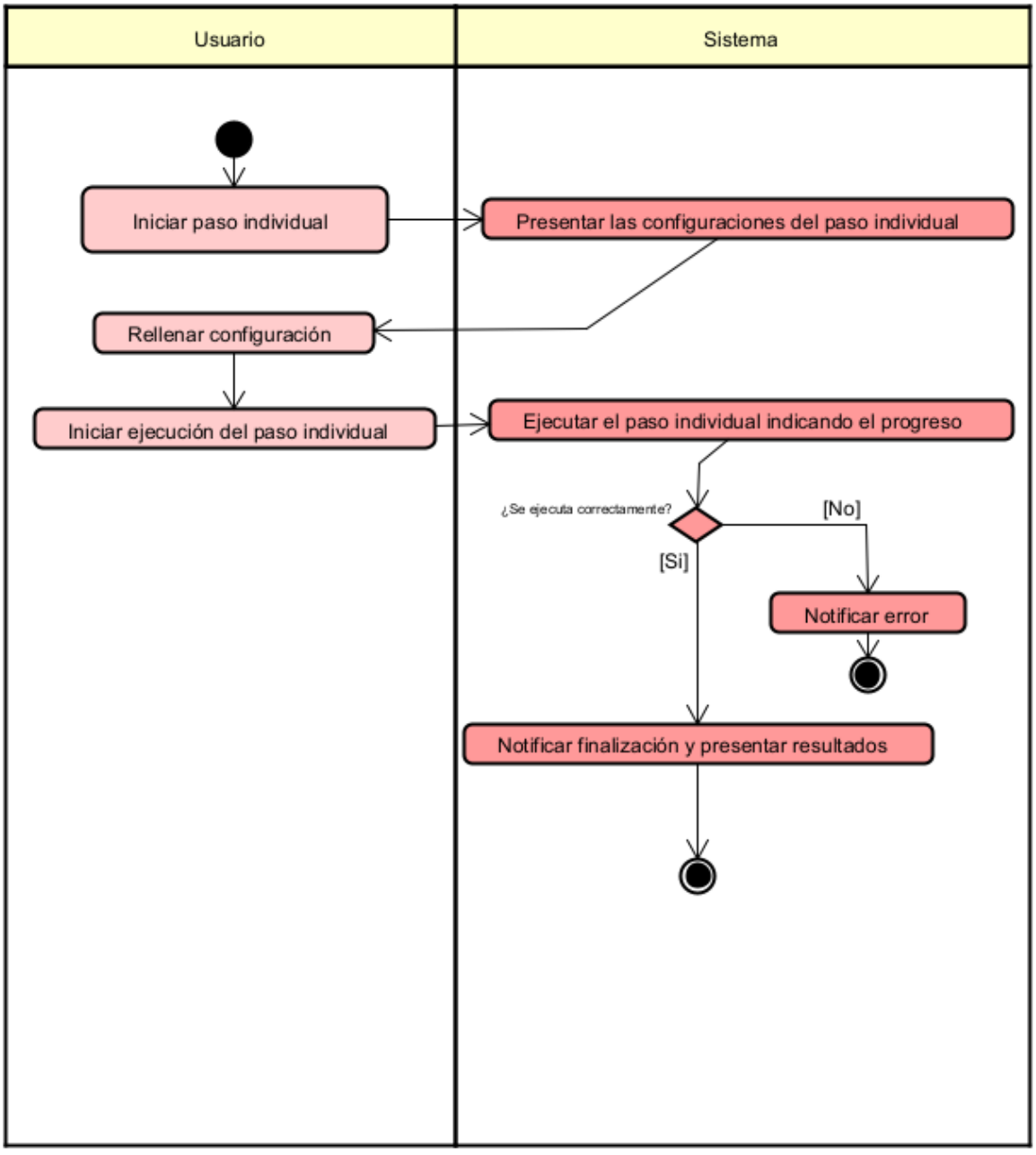


Figura 5.7: Diagrama de actividad: Ejecutar paso individual de procesado (CU-06). Elaboración Propia.

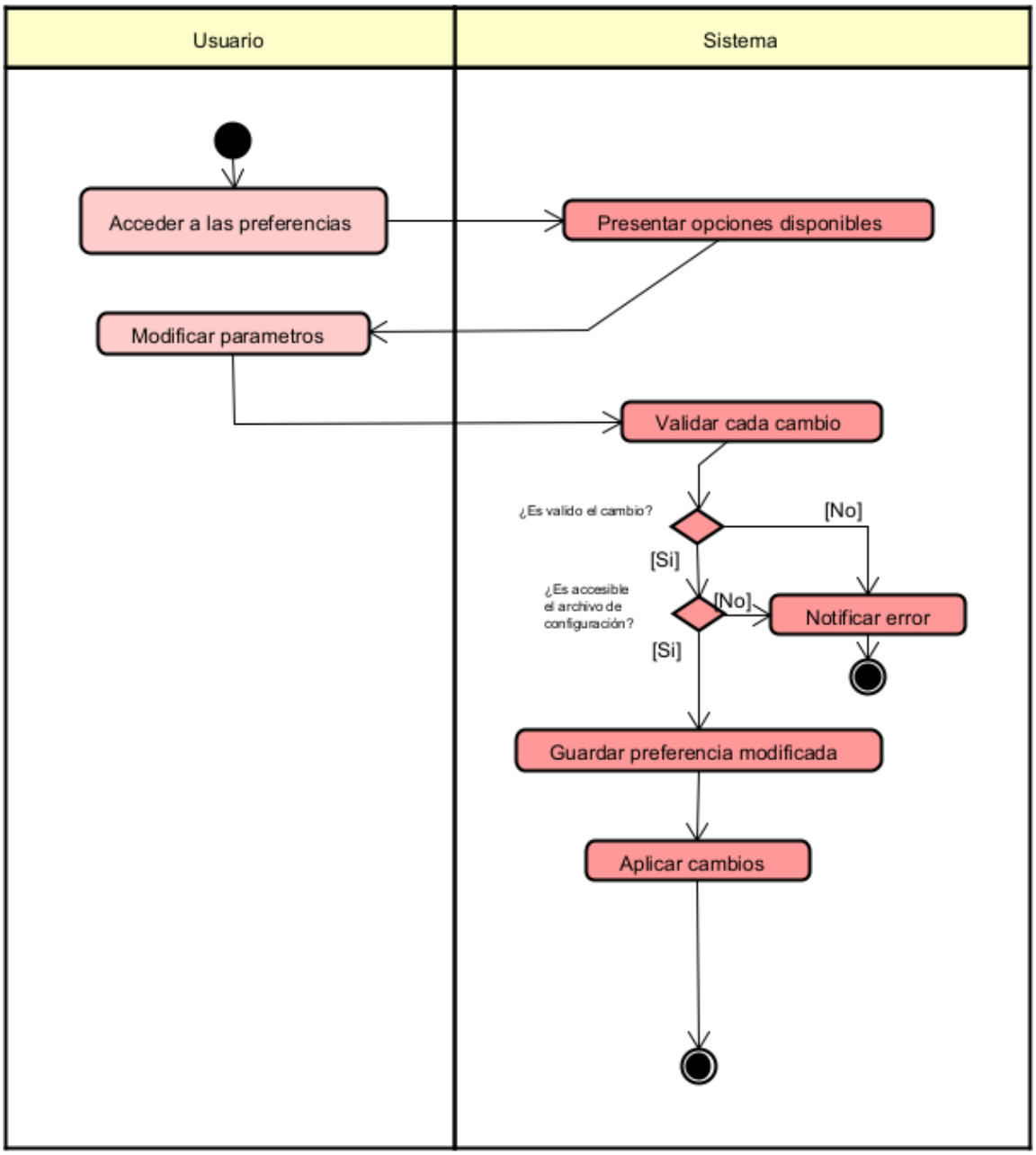


Figura 5.8: Diagrama de actividad: Gestionar preferencias de usuario (CU-07). Elaboración Propia.

Capítulo 6

Diseño

6.1 Diseño

Tras completar el análisis inicial, se procede al diseño de la aplicación, una etapa más detallada y cercana a la implementación que facilita el desarrollo mediante código. En este capítulo se documenta el diseño de paquetes de la aplicación, las clases principales, el diagrama de despliegue y los patrones de diseño implementados.

6.1.1. Arquitectura del sistema

La arquitectura del sistema se ha diseñado siguiendo el patrón Modelo-Vista-Controlador (MVC), que divide la aplicación en tres componentes interconectados. Este patrón facilita la separación de responsabilidades y mejora la modularidad y mantenibilidad del código [21].

- **Modelo:** Representado por las clases que gestionan los datos de la aplicación, incluyendo configuraciones, ajustes de dispositivos y lógica de procesamiento de datos.
- **Vista:** Implementada mediante interfaces de usuario generadas por Qt Designer y sus contrapartes compiladas en python, que definen los elementos visuales y la estructura de presentación.
- **Controlador:** Representado por las clases 'interface' de la aplicación que gestionan la lógica de control, conectando las interacciones del usuario con el modelo y actualizando la vista cuando sea necesario.

El flujo de datos en este patrón MVC se ve claramente en los módulos de la aplicación: los datos

del nefelómetro (Modelo) son procesados por distintos hilos, la interfaz (Vista) muestra los resultados, y las clases controladoras actúan como intermediarias coordinando ambos componentes.

Estructura de paquetes

Para la estructuración del código, se ha realizado la siguiente distribución de paquetes. La figura 6.1 muestra el diagrama de paquetes general de la aplicación.

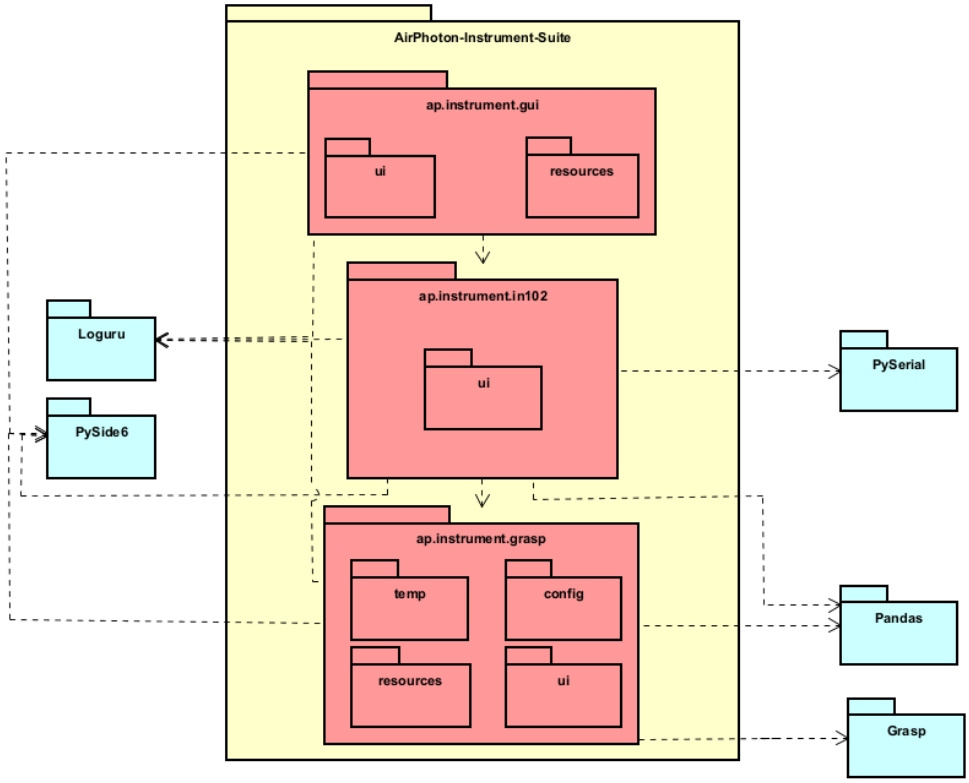


Figura 6.1: Diagrama de paquetes general de la aplicación. Elaboración Propia.

La aplicación se divide en tres paquetes principales:

- **ap-instrument-gui**: Proporciona la interfaz principal de la aplicación donde se integran los demás módulos. Contiene componentes de vista (ventanas y diálogos principales), controladores para la gestión general de la aplicación y modelos para la configuración global.
- **ap-instrument-in102**: Módulo específico para el instrumento IN102 que se integra en la interfaz principal. Implementa su propia estructura MVC con vistas específicas del instrumento, controladores para la interacción con el hardware y modelos para la gestión de datos del nefelómetro.
- **ap-instrument-grasp**: Módulo para el procesamiento mediante algoritmo GRASP que se integra en la interfaz principal. Al igual que el módulo IN102, implementa su propia estructura MVC.

Paquete ap-instrument-grasp

Para ilustrar en detalle la estructura interna de los paquetes, se presenta el paquete ap-instrument-grasp como ejemplo representativo. La figura 6.2 muestra su diagrama de paquetes.

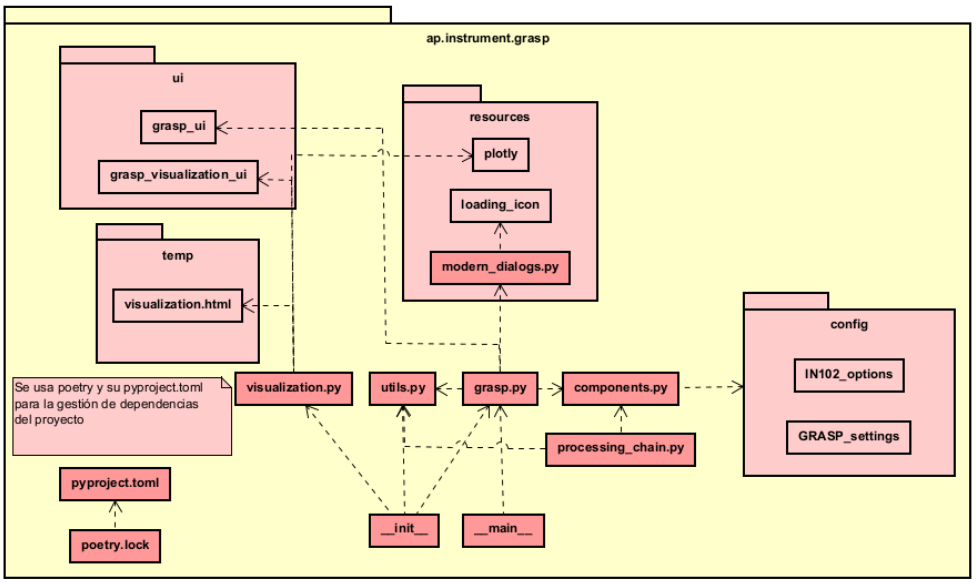


Figura 6.2: Diagrama de paquetes del módulo ap-instrument-grasp. Elaboración Propia.

Este paquete se estructura de la siguiente manera:

- **ui**: Contiene los archivos de interfaz de usuario generados por Qt Designer (.ui) y sus versiones compiladas a Python, que representan la vista en el patrón MVC.
- **temp**: Directorio donde se almacenan temporalmente las visualizaciones de datos generadas.
- **resources**: Contiene el código de plotly.js necesario para la visualización interactiva, un icono de carga y archivos de diálogos gráficos personalizados para notificaciones de finalización.
- **config**: Almacena los archivos de configuración tanto para el driver como para el algoritmo GRASP, definiendo parámetros y opciones para el procesamiento.
- **components.py**: Implementa las clases del modelo que gestionan la funcionalidad principal, como la interacción con el contenedor Docker de GRASP y el procesamiento de datos.
- **grasp.py**: Contiene la clase GraspInterface que actúa como controlador, conectando la interfaz de usuario (grasp_ui) con la lógica de negocio y coordinando las operaciones del módulo.
- **poetry.lock** y **pyproject.toml**: Archivos para la gestión de dependencias mediante Poetry, que garantizan un entorno de desarrollo consistente y facilitan la instalación de las bibliotecas necesarias.

Esta organización modular facilita la extensión y mantenimiento del código, permitiendo que cada componente se desarrolle y pruebe de manera independiente mientras mantiene una clara separación de responsabilidades según el patrón MVC.

6.2 Patrones de Diseño

Los patrones de diseño son soluciones probadas a problemas comunes que proporcionan una estructura y organización adecuadas para crear sistemas escalables, mantenibles y reutilizables. A continuación, se analizan los patrones principales implementados en el proyecto.

6.2.1. Patrones de diseño

Patrón Observador (Observer)

El patrón Observer define una dependencia uno-a-muchos entre objetos, de modo que cuando un objeto cambia su estado, todos sus dependientes son notificados y actualizados automáticamente. Esto permite un acoplamiento bajo entre componentes que interactúan entre sí [22].

En la aplicación, este patrón se implementa a través del mecanismo Signal-Slot de Qt:

- Las clases principales implementan slots que observan señales emitidas por otros componentes.
- Los elementos del sistema emiten señales que son observadas por métodos específicos.
- Los hilos de trabajo implementan señales para notificar sobre el estado y resultado de sus procesos.
- Los componentes de procesamiento de datos emiten señales cuando hay nueva información disponible para ser mostrada en la interfaz gráfica.

Este patrón facilita la comunicación entre componentes desacoplados, especialmente útil para operaciones asíncronas como la calibración y el procesamiento de datos, permitiendo que la interfaz de usuario permanezca receptiva mientras se realizan tareas intensivas en segundo plano.

Patrón Decorador (Decorator)

El patrón Decorador permite añadir responsabilidades a objetos individuales dinámicamente, sin afectar el comportamiento de otros objetos de la misma clase. Proporciona una alternativa flexible a la herencia para extender funcionalidad [23].

En la aplicación AirPhoton, el patrón Decorator se implementa principalmente mediante el uso de los decoradores de Python, en concreto los decoradores de Qt:

- Los métodos que manejan eventos en la interfaz de usuario están decorados con `@Slot`, añadiendo funcionalidad para conectarse al sistema de señales de Qt.
- Los métodos que procesan señales de la interfaz de usuario utilizan decoradores para integrarse con el sistema de señales y slots.

Este patrón permite que la infraestructura de señales y slots de Qt se integre sin problemas con las clases Python, añadiendo comportamiento para manejar eventos sin modificar el código base de las clases.

Patrón Composite

El patrón Composite permite componer objetos en estructuras de árbol para representar jerarquías. Permite tratar objetos individuales y composiciones de objetos de manera uniforme [24].

En esta aplicación, el patrón Composite se implementa a través de la jerarquía de widgets de Qt:

- Las interfaces de la aplicación crean estructuras jerárquicas de widgets que contienen widgets hijos, todos tratados uniformemente a través de layouts.
- Las ventanas principales contienen múltiples páginas que son tratadas uniformemente, aunque cada una puede contener diferentes jerarquías de widgets.
- Los componentes de visualización contienen elementos gráficos compuestos para mostrar resultados complejos.

Este patrón facilita la creación de interfaces de usuario complejas que pueden ser manipuladas de manera uniforme, permitiendo que las operaciones se propaguen a través de la jerarquía de

componentes.

6.2.2. Diagrama de despliegue

El diagrama de despliegue muestra la distribución física de los componentes software en la infraestructura hardware necesaria para su ejecución. En esta sección se presentan los diagramas de despliegue que ilustran los dos escenarios principales de funcionamiento de la aplicación: procesamiento local y procesamiento remoto.

Escenario de procesamiento local

En el escenario de procesamiento local, todos los componentes se ejecutan en la misma máquina del usuario, como se muestra en la figura 6.3.

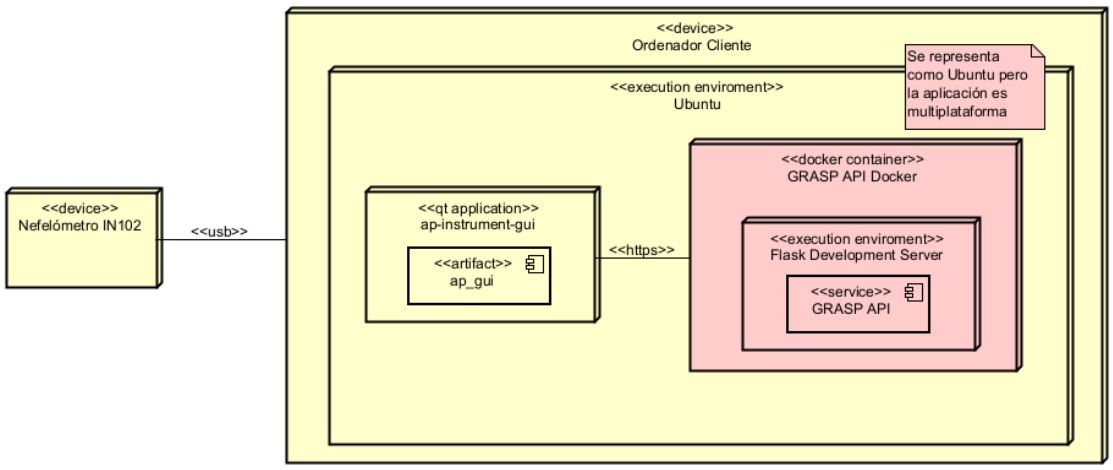


Figura 6.3: Diagrama de despliegue para procesamiento local. Elaboración Propia.

En este escenario:

- **Ordenador del usuario**: Aloja la aplicación principal.
- **Nefelómetro IN102**: Instrumento físico conectado al ordenador del usuario mediante puerto USB que proporciona los datos de medición.
- **Contenedor Docker GRASP**: Ejecuta el algoritmo GRASP en un servidor docker en la misma máquina, aislando sus dependencias del resto del sistema.

Escenario de procesamiento remoto

En el escenario de procesamiento remoto, el algoritmo GRASP se ejecuta en un servidor dedicado, como se muestra en la figura 6.4.

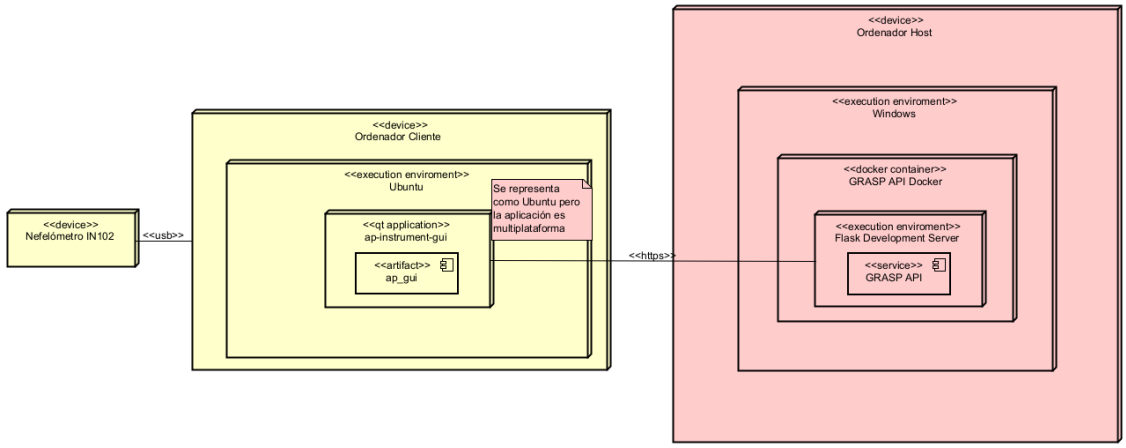


Figura 6.4: Diagrama de despliegue para procesamiento remoto. Elaboración Propia.

En este escenario:

- **Ordenador del usuario**: Aloja la aplicación principal.
- **Nefelómetro IN102**: Instrumento físico conectado al ordenador del usuario mediante puerto USB que proporciona los datos de medición.
- **Ordenador/Servidor GRASP**: Un servidor dedicado que ejecuta el algoritmo GRASP en un contenedor Docker, exponiendo una API REST para recibir datos y devolver resultados. Este servidor puede estar dimensionado para manejar cargas de trabajo significativas.

Capítulo 7

Implementación

7.1 Tecnologías utilizadas

A continuación se describen las principales tecnologías, frameworks, bibliotecas y herramientas empleadas en el desarrollo del sistema, organizadas según las tres etapas principales del flujo de trabajo: preprocesado de datos, procesamiento, y visualización, además de las tecnologías transversales utilizadas en todo el proyecto.

7.1.1. Tecnologías transversales

Python

Python [25] es el lenguaje de programación principal utilizado en el desarrollo de todas las etapas del sistema, debido a su amplia adopción en la comunidad científica y su ecosistema de bibliotecas para procesamiento de datos. Entre sus ventajas para este proyecto destacan:

1. Sintaxis clara y legible que facilita el desarrollo y mantenimiento del código.
2. Gran soporte para procesamiento numérico y científico.
3. Gran cantidad de bibliotecas especializadas para análisis de datos.
4. Facilidad para integración con otros sistemas y tecnologías.

La implementación ha aprovechado Python como lenguaje base para desarrollar tanto el driver

del nefelómetro, como la interfaz gráfica y los servicios de procesamiento, permitiendo una integración fluida entre todos los componentes del sistema.

Poetry

Poetry [26] es la herramienta de gestión de dependencias y entornos virtuales utilizada en el proyecto. Sus características incluyen:

1. Gestión declarativa de dependencias con resolución automática de conflictos.
2. Creación y gestión de entornos virtuales aislados.
3. Control preciso de versiones de dependencias para garantizar la reproducibilidad.

En este proyecto, Poetry ha sido fundamental para gestionar las numerosas dependencias de los diferentes componentes del sistema, asegurando la compatibilidad entre bibliotecas y facilitando la instalación del software en diferentes entornos virtuales.

Git

Git [27] es el sistema de control de versiones seleccionado para gestionar el desarrollo del proyecto. Sus características principales incluyen:

1. Control de cambios y versiones del código fuente.
2. Facilidad para la colaboración entre desarrolladores.
3. Gestión eficiente de ramas para funcionalidades y correcciones.
4. Soporte para pipelines y pruebas automatizadas.

Durante el desarrollo, Git ha sido fundamental para mantener un desarrollo organizado y documentado, facilitando la implementación iterativa e incremental del sistema.

Cursor

Cursor [28] es el entorno de desarrollo integrado (IDE) utilizado durante la implementación del proyecto. Sus principales características incluyen:

1. Integración nativa con modelos de inteligencia artificial para asistencia en la programación.
2. Interfaz moderna y personalizable basada en VSCode [29] que mejora la productividad del desarrollador.
3. Funcionalidades de autocompletado y sugerencias de código inteligentes.
4. Integración con Git para facilitar el control de versiones.

La utilización de Cursor ha permitido agilizar el desarrollo, especialmente en la implementación de componentes repetitivos y mecánicos, como en la resolución de problemas técnicos, gracias a su capacidad para proporcionar sugerencias contextuales basadas en el código existente.

YAML y ruamel.yaml

YAML es el formato de serialización de datos utilizado para gestionar las configuraciones del sistema. La biblioteca ruamel.yaml [30] ha sido empleada específicamente para manipular estos archivos de configuración. Sus principales características incluyen:

1. Formato legible por humanos que facilita la edición manual de configuraciones.
2. Preservación de comentarios y estructura al modificar archivos existentes.
3. Soporte para tipos de datos complejos y referencias.
4. Gestión robusta de errores de sintaxis y validación.

En el contexto del proyecto, ruamel.yaml ha sido fundamental para cargar y guardar configuraciones de diversos componentes: parámetros del sistema, configuración del driver del nefelómetro IN102 y ajustes del algoritmo GRASP. Esta tecnología ha permitido implementar un sistema donde los usuarios pueden modificar los parámetros de funcionamiento sin necesidad de modificar el código fuente.

Loguru

Loguru [31] es la biblioteca de registro (logging) utilizada en todos los módulos del sistema. Sus características principales incluyen:

1. Sintaxis simple y expresiva para la generación de registros.

2. Soporte para múltiples niveles de registro (debug, info, warning, error, critical).
3. Capacidad para formatear mensajes con información contextual como timestamps, nombres de archivos y números de línea.
4. Rotación automática de archivos de registro y gestión eficiente del almacenamiento.
5. Integración fluida con el ecosistema Python y compatibilidad con aplicaciones asíncronas.

La implementación de Loguru ha permitido un seguimiento detallado de la ejecución del sistema, facilitando la depuración durante el desarrollo y proporcionando información útil para el diagnóstico de problemas.

Docker

Docker [32] es la plataforma de contenedorización elegida para el despliegue del algoritmo GRASP y la API REST. Sus ventajas principales son:

1. Entorno aislado y reproducible para la ejecución de algoritmos GRASP.
2. Gestión eficiente de dependencias complejas.
3. Facilidad para distribuir y desplegar el servicio de procesamiento.
4. Consistencia entre entornos de desarrollo y producción.

La utilización de Docker consiste en encapsular el entorno de ejecución de GRASP junto con la API REST, garantizando su correcta ejecución independientemente del sistema operativo del usuario.

7.1.2. Tecnologías para el preprocesado de datos

Driver personalizado

El driver personalizado desarrollado previamente para la comunicación con el nefelómetro IN102 constituye una pieza fundamental del sistema.

La función principal del driver es la conversión de los datos brutos del nefelómetro, obtenidos originalmente en formato CSV, al formato NetCDF (.nc) [33].

NetCDF (Network Common Data Form) es un formato de archivo diseñado específicamente para el almacenamiento de datos científicos multidimensionales, que ofrece ventajas significativas como la independencia de plataforma, la capacidad de almacenar metadatos junto con los datos, y una estructura optimizada para el acceso eficiente a grandes volúmenes de información.

7.1.3. Tecnologías para el procesamiento de datos

GRASP

GRASP [14] es el algoritmo especializado utilizado para el procesamiento de datos.

Como núcleo del sistema, GRASP se encarga de transformar las mediciones del nefelómetro (una vez estas han sido convertidas a formato NetCDF) en información científicamente relevante sobre la calidad del aire.

Flask

Flask [34] es el framework web ligero seleccionado para implementar la API REST del sistema. Sus características más relevantes incluyen:

1. Framework minimalista y flexible para APIs REST.

2. Fácil integración con el ecosistema Python.

3. Gestión eficiente de solicitudes de procesamiento.

4. Soporte para procesamiento asíncrono.

Dentro de la arquitectura del sistema, Flask proporciona la capa de servicios web que permite la comunicación entre la interfaz de usuario y el procesamiento de datos tanto en modo local como en modo remoto.

7.1.4. Tecnologías para la visualización de datos

Qt

Qt [35] es el framework de desarrollo de interfaces gráficas elegido para el proyecto. Sus características principales son:

1. Componentes de interfaz multiplataforma con aspecto nativo.
2. Herramientas de diseño visual que aceleran el desarrollo.
3. Capacidad para integrar gráficos y visualizaciones complejas.
4. Sistema de señales y slots para una interacción fluida.
5. Capacidad de threading para la ejecución de tareas en paralelo.

La implementación con Qt ha permitido desarrollar una interfaz de usuario intuitiva y profesional que permite a los científicos interactuar eficientemente con el sistema. Además, la capacidad de threading integrada en la propia librería Qt ha permitido mejorar la respuesta del sistema frente a eventos de usuario, lanzando tareas como el procesamiento o la visualización de resultados en segundo plano, permitiendo una interacción más fluida y una mejor experiencia de usuario.

Plotly

Plotly [36] es la biblioteca de visualización de datos utilizada para el proyecto. Sus características destacadas son:

1. Gráficos interactivos y responsivos de alta calidad.
2. Capacidad para exportar las gráficas en formato png.
3. Personalización avanzada con HTML, CSS y JavaScript de representaciones gráficas.

Para la visualización de resultados, Plotly ha sido fundamental en la creación de visualizaciones interactivas de los resultados del procesamiento, permitiendo a los usuarios analizar y explorar los datos de forma comoda y precisa.

7.2 Detalles de implementación

En esta sección se describen los aspectos técnicos más relevantes de la implementación del sistema, incluyendo los principales componentes desarrollados y las decisiones técnicas más importantes.

7.2.1. Desarrollo de la interfaz gráfica

El desarrollo de la interfaz gráfica no siguió un diseño completamente predefinido. Siguiendo la metodología incremental iterativa planteada, fue necesario generar funcionalidades operativas lo antes posible para comenzar con el ciclo de retroalimentación.

Las interfaces se fueron construyendo y actualizando con QtDesigner sobre la marcha, en función de los requisitos que iban surgiendo durante cada iteración. Esta aproximación permitió una evolución gradual de la interfaz, adaptándose a las necesidades reales de los usuarios.

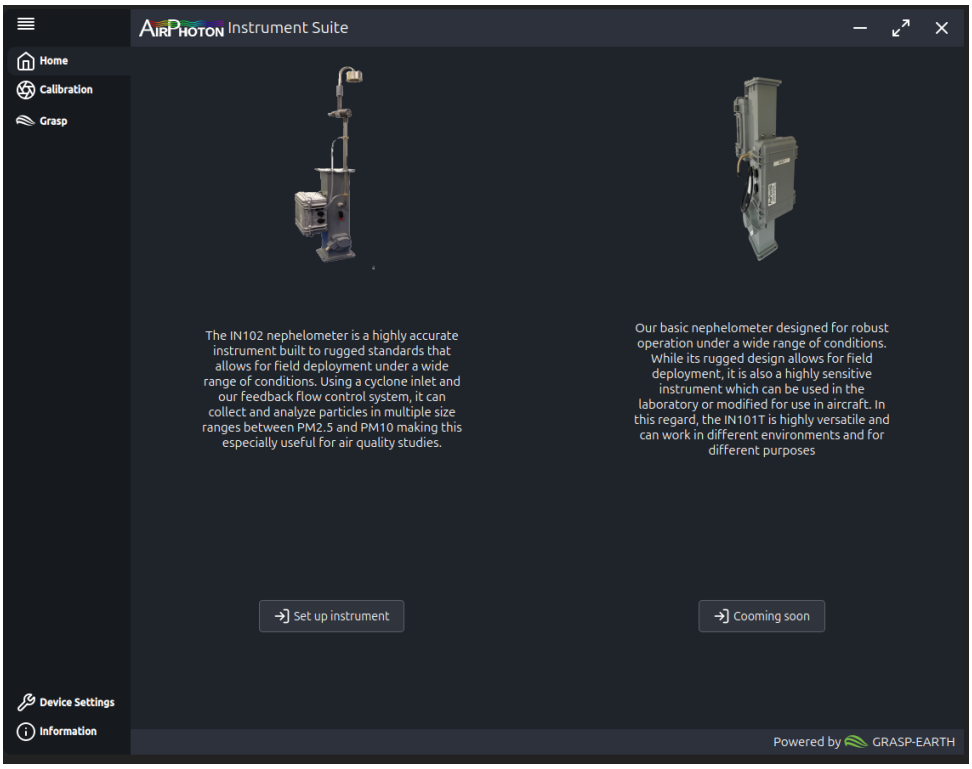
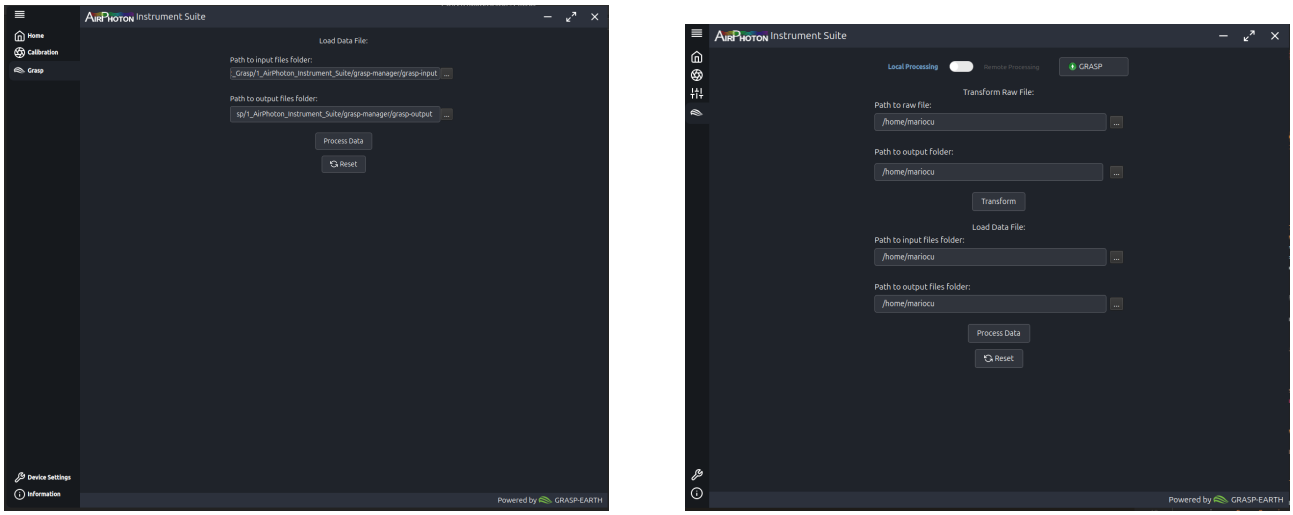


Figura 7.1: Pantalla principal de la aplicación. Elaboración Propia.

La evolución de los componentes de la interfaz fue notable durante el desarrollo, especialmente en los módulos de procesamiento. La Figura 7.2 muestra esta evolución, desde los primeros prototipos hasta la versión final.

La configuración del instrumento también fue un aspecto importante de la interfaz, permitiendo al usuario establecer los parámetros de conexión de forma intuitiva, como se muestra en la Figura 7.3.



(a) Prototipo inicial de los componentes (b) Versión final de los componentes

Figura 7.2: Evolución de la interfaz de procesamiento GRASP. Elaboración Propia.

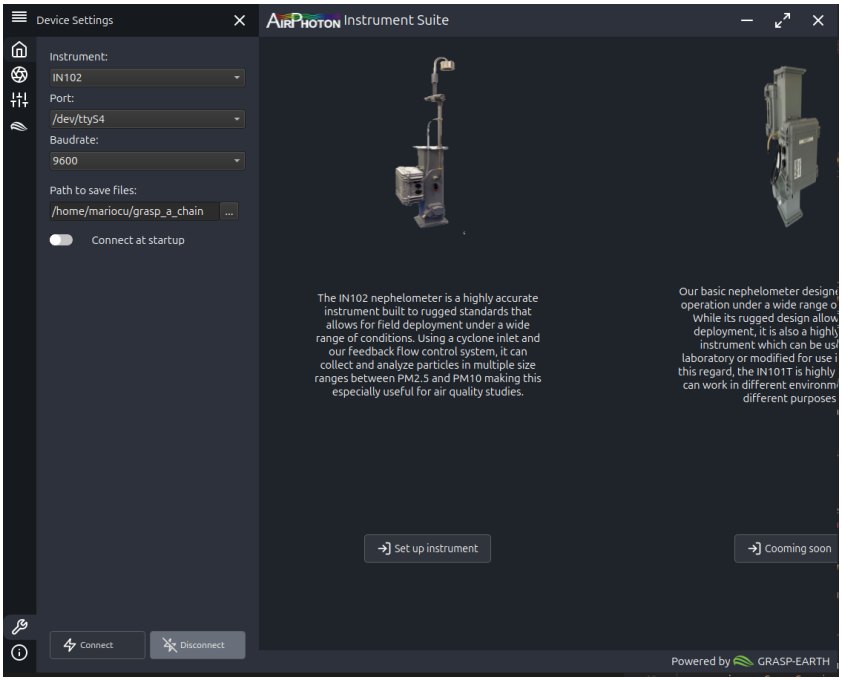
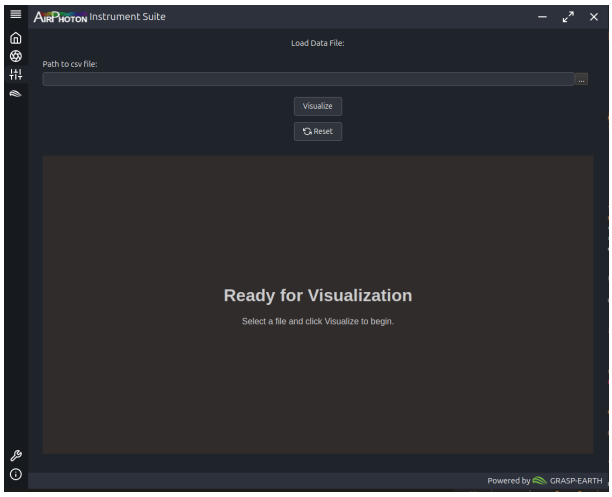


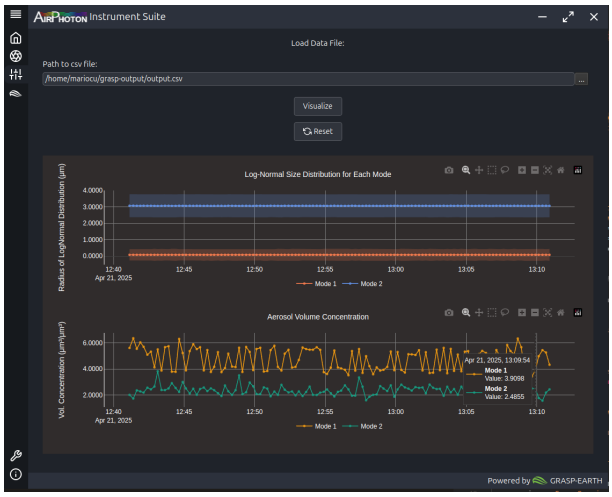
Figura 7.3: Configuración de conexión al instrumento. Elaboración Propia.

7.2.2. Visualización de resultados

Un componente clave de la aplicación es la visualización de los datos obtenidos. Se diseñaron interfaces específicas para presentar los resultados de forma clara y útil para los científicos. Esta visualización fue implementada utilizando Plotly, generando un HTML enriquecido con CSS y JavaScript que proporciona una experiencia visual e interactiva. Estos elementos gráficos fueron integrados en la aplicación mediante el componente QWebEngineView de Qt, permitiendo a los usuarios interactuar con las visualizaciones directamente dentro de la interfaz principal sin necesidad de aplicaciones externas.



(a) Interfaz de visualización sin datos cargados. Elaboración Propia.



(b) Interfaz de visualización con datos procesados. Elaboración Propia.

Figura 7.4: Sistema de visualización de datos. Elaboración Propia.

El resultado final de la cadena completa de procesado, incluyendo la adquisición, procesamiento y visualización de los datos, se muestra en la Figura 7.5.

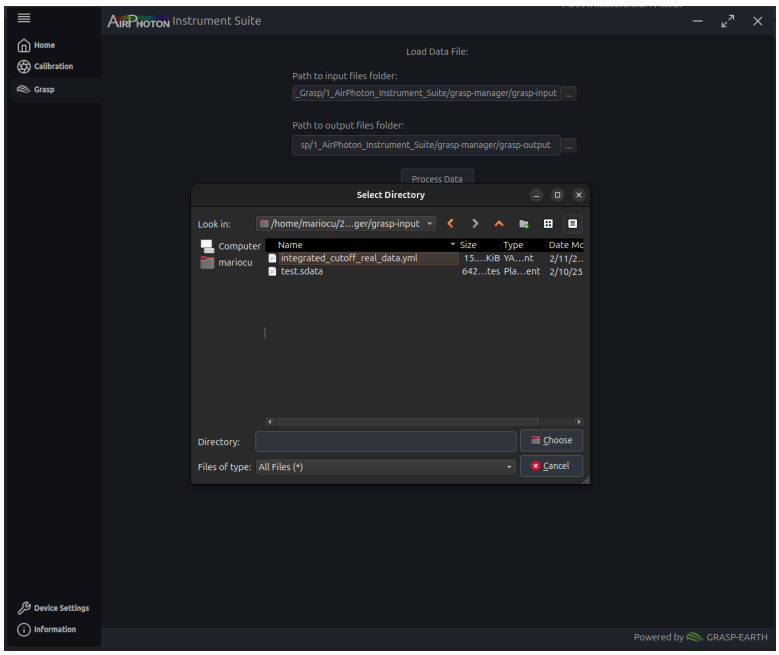
Figura 7.5: Aplicación en funcionamiento con la cadena de procesado completa. Elaboración Propia.

7.2.3. Implementación del procesamiento en segundo plano

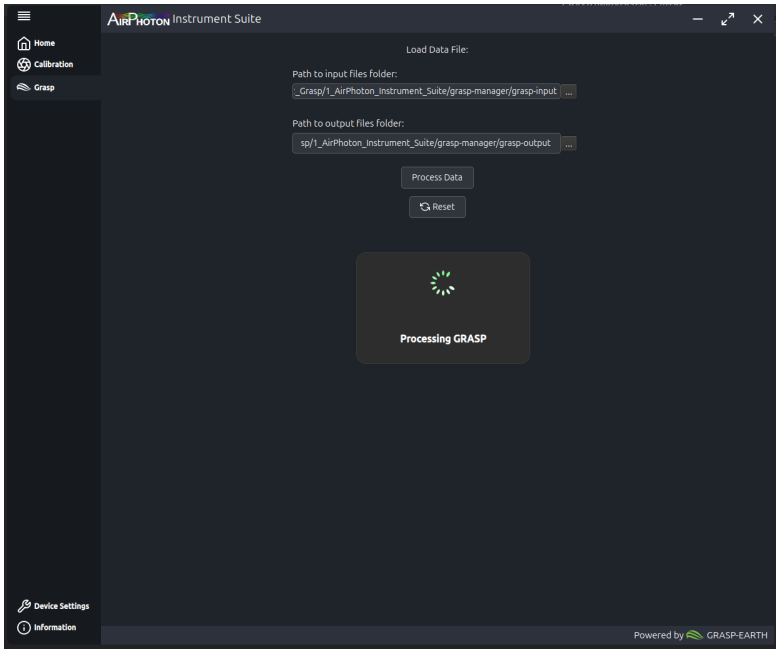
Un aspecto crucial de la implementación fue evitar la ralentización de la aplicación durante operaciones de larga duración. Para lograr esto, se implementó un sistema basado en `QThread` que permite ejecutar diferentes componentes del sistema en hilos separados:

- **Preprocesamiento**: Las operaciones de lectura, validación y preparación de datos del instrumento se ejecutan en un hilo independiente.
- **Procesamiento**: La ejecución del algoritmo GRASP, que puede durar varios minutos, se realiza en un hilo separado.
- **Visualización**: La generación de gráficas e imágenes también se realiza en segundo plano para mantener la interfaz reactiva.

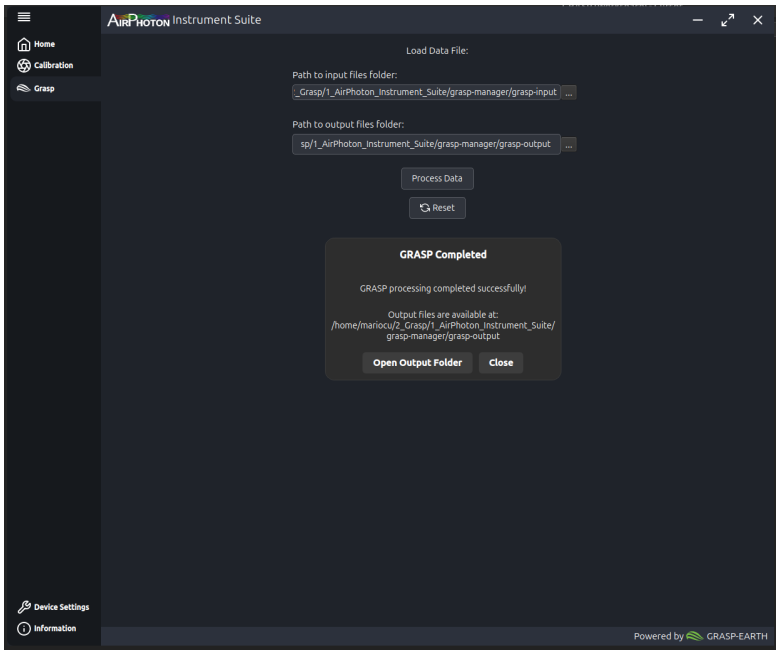
Este enfoque multihilo garantiza que la interfaz de usuario permanezca receptiva incluso durante operaciones intensivas, mejorando la experiencia del usuario y evitando bloqueos. La Figura 7.6 muestra el flujo de procesamiento, desde la selección de archivos hasta la notificación de finalización.



(a) Selección de archivos



(b) Hilo en segundo plano



(c) Notificación de finalización

7.2.4. Implementación del servidor GRASP API

Tras la reunión de seguimiento durante la iteración 2 (ver sección 3.2), surgió la necesidad de implementar un enfoque más flexible para la ejecución del algoritmo GRASP. Esto llevó al desarrollo de una solución basada en Docker con las siguientes características:

- **Contenedor Docker**: Se desarrolló una imagen Docker que incluye todas las dependencias necesarias para ejecutar el algoritmo GRASP, junto con un servidor Flask para proporcionar una API REST.
- **API REST**: Se implementó un servicio web REST que permite la ejecución remota del algoritmo GRASP, recibiendo datos de entrada y devolviendo resultados procesados.
- **Sistema de identificación UUID**: Se implementó un sistema de identificación basado en UUID para gestionar procesamiento asíncrono e individualizado, permitiendo múltiples solicitudes concurrentes sin conflictos.
- **Despliegue simplificado**: Para resolver el problema de la construcción manual del contenedor por parte de los usuarios, se publicó la imagen en un repositorio privado de artefactos de la empresa. Se añadió un diálogo de inicio de sesión y descarga en la aplicación para facilitar la obtención de la imagen.

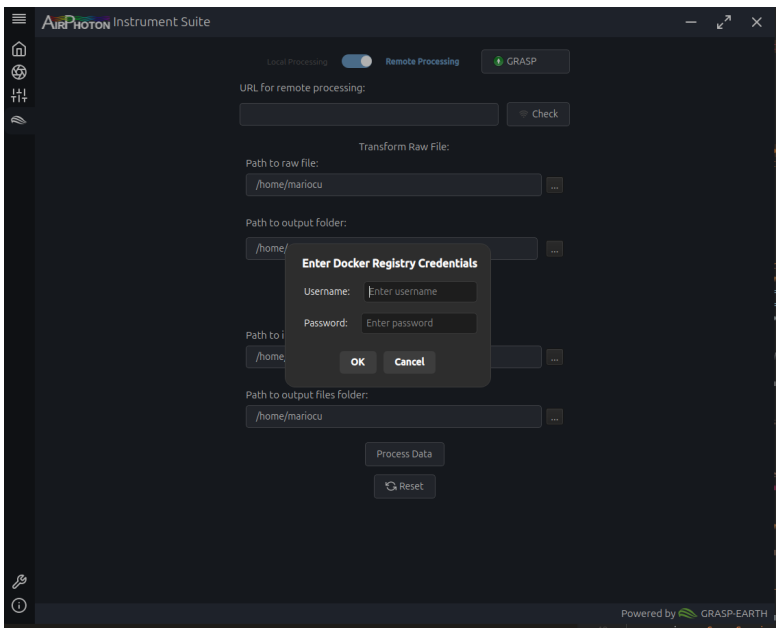


Figura 7.7: Interfaz de inicio de sesión para descarga de la imagen Docker. Elaboración Propia.

Esta implementación proporciona flexibilidad para ejecutar el algoritmo GRASP tanto localmente como en servidores remotos, según las necesidades del usuario.

7.2.5. Conexión con el nefelómetro IN102

Durante el desarrollo y pruebas del sistema, se estableció una conexión directa con el nefelómetro IN102 a través de USB, como se muestra en la Figura 7.8.

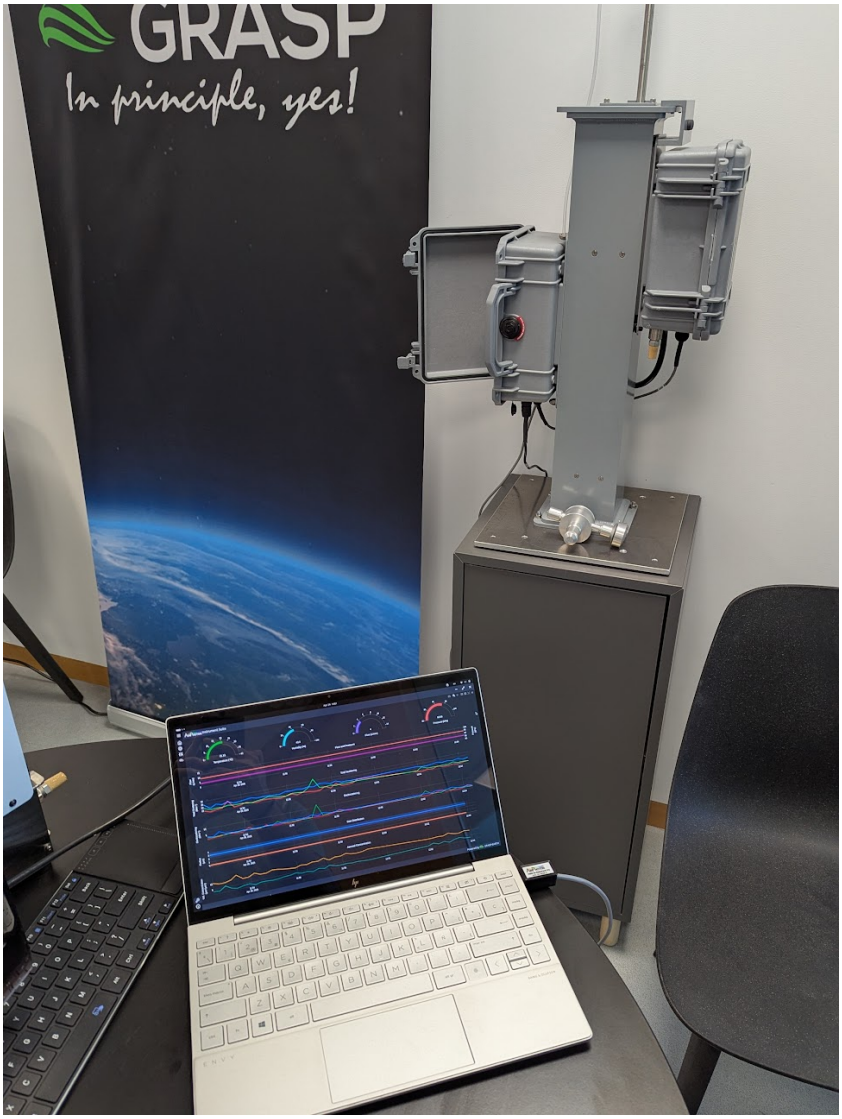


Figura 7.8: Pruebas de desarrollo con el nefelómetro IN102 conectado por USB. Elaboración Propia.

Esta configuración permitió desarrollar y probar en tiempo real la cadena de procesado, verificando la correcta adquisición de datos desde el instrumento, su procesamiento y la generación de resultados.

7.2.6. Procesos de desarrollo

El proyecto se desarrolló en un entorno profesional, internacional e interdisciplinar, siguiendo prácticas de desarrollo de software reconocidas:

- **Reuniones semanales**: Durante todo el desarrollo, se contó con el apoyo y asesoramiento de un equipo de desarrolladores internacional, que se reunía semanalmente los jueves para compartir progresos e información relevante.
- **Control de versiones**: Se utilizó Git como sistema de control de versiones, siguiendo un flujo de trabajo basado en ramas (Feature Branches) para implementar las distintas funcionalidades.
- **Principio de los seis ojos**: Antes de integrar cualquier funcionalidad en la rama principal, se realizaba un proceso de revisión de código por parte de dos compañeros del equipo de desarrollo, siguiendo el principio de los seis ojos. Este principio extiende el concepto tradicional de cuatro ojos (dos personas revisando un proceso) a tres personas, proporcionando un nivel adicional de garantía de calidad y detección de posibles problemas.

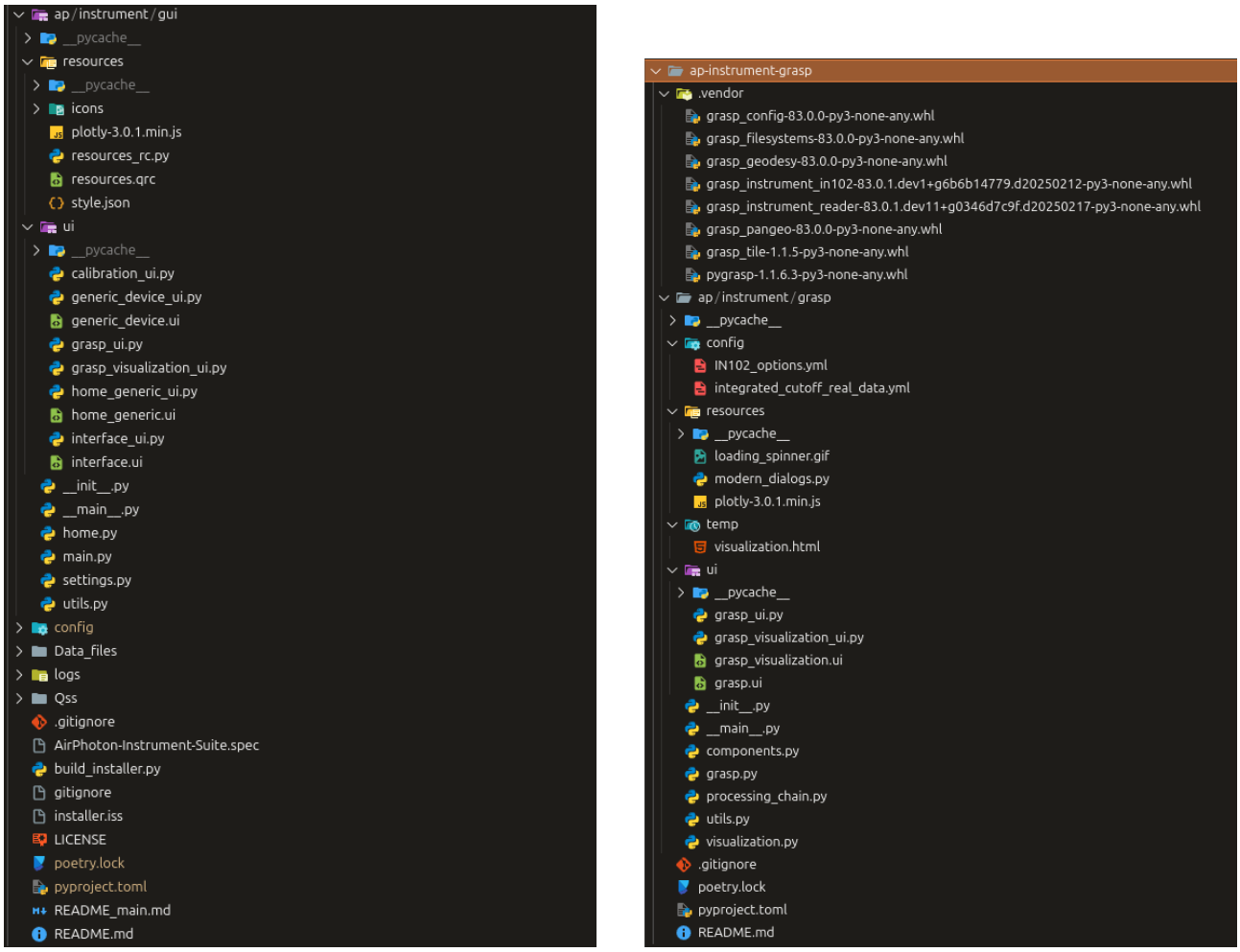
Este enfoque de desarrollo garantizó la calidad del código, facilitó la colaboración entre los desarrolladores y proporcionó un entorno de aprendizaje muy relevante para el proyecto.

7.3 Estructura final del proyecto

En esta sección se detalla la estructura final del proyecto, incluyendo tanto la aplicación principal como el servidor Flask que implementa la API GRASP para el procesamiento de datos.

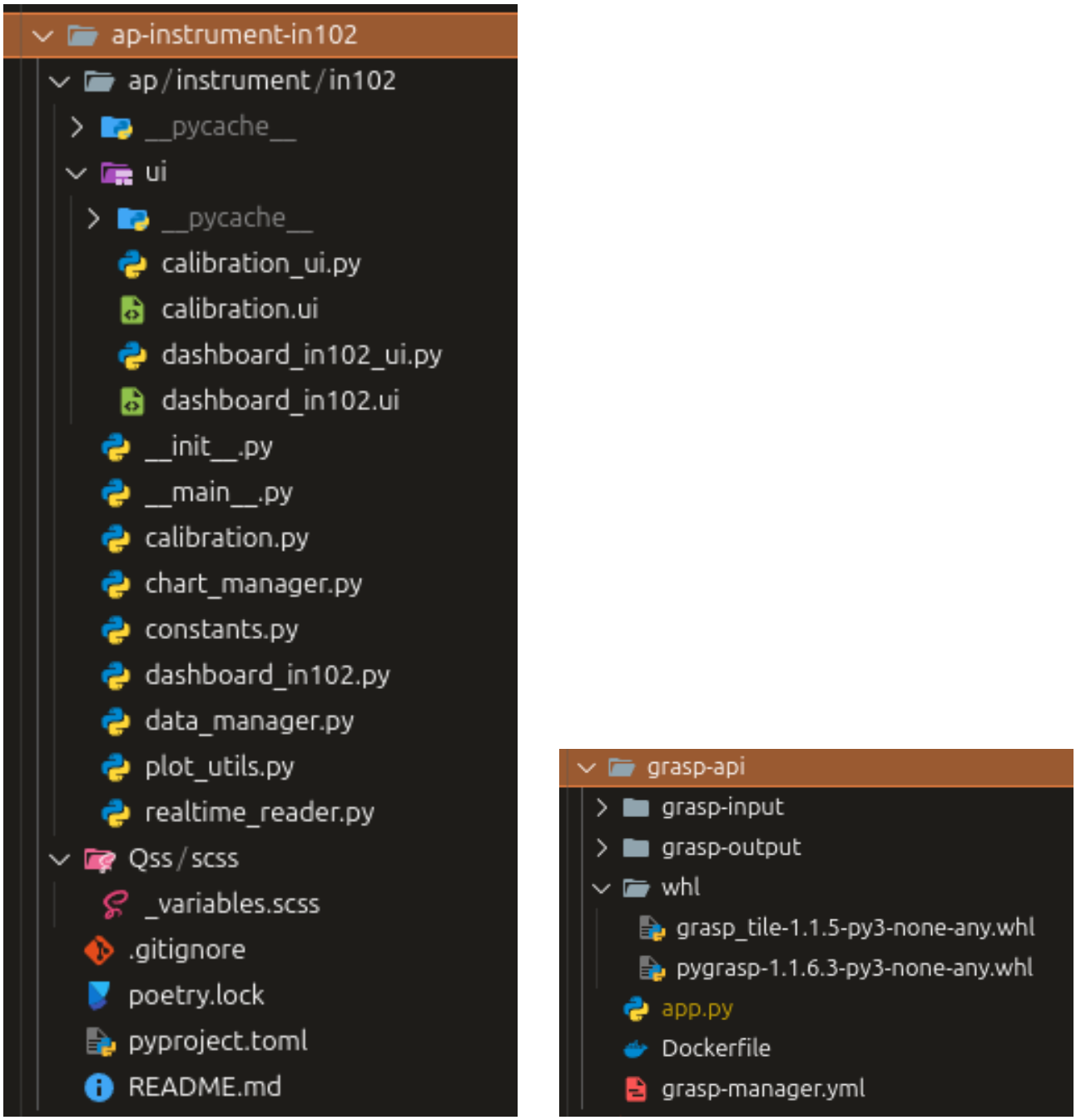
La aplicación principal está organizada en tres paquetes principales que implementan una arquitectura modular y extensible. La estructura de cada paquete refleja el patrón MVC explicado en el capítulo de diseño, con una clara separación de responsabilidades.

Las Figuras 7.9 y 7.10 muestran la estructura de los distintos paquetes del sistema, ademas de la estructura de la API GRASP desarrollada.



(a) Estructura del paquete ap-instrument-gui (b) Estructura del paquete ap-instrument-grasp

Figura 7.9: Estructura de los paquetes GUI y GRASP. Elaboración propia.



(a) Estructura del paquete ap-instrument-in102

(b) Estructura del servidor GRASP API

Figura 7.10: Estructura del paquete IN102 y el servidor API. Elaboración propia.

7.3.1. Comunicación entre componentes

La comunicación entre los diferentes componentes del sistema se ha implementado siguiendo estas estrategias:

- **Entre paquetes de la aplicación principal**: Mediante la importación directa de módulos y el uso del sistema de señales y slots de Qt para la comunicación asíncrona.
- **Entre la aplicación y el servidor GRASP API**: Mediante peticiones HTTP utilizando la biblioteca `requests` de Python, siguiendo un enfoque RESTful para las operaciones principales:
  - POST /process para enviar datos a procesar, el resultado de la ejecución se obtiene directamente en la respuesta de la petición para acelerar el proceso, aunque se puede obtener a través de una petición GET.
  - GET /output_files para obtener los archivos de salida del procesamiento.
  - GET /output/{file_name} para obtener un archivo de salida del procesamiento.
  - GET /metrics para obtener las métricas del estado del servidor.

Esta arquitectura modular y la comunicación definida entre componentes facilitan el mantenimiento del sistema y permiten su evolución futura, siguiendo los principios de diseño establecidos en el capítulo anterior.

Capítulo 8

Pruebas

8.1 Introducción

En el ámbito del desarrollo software, las pruebas representan un elemento crítico para garantizar la calidad del producto final. Este proceso estructurado permite ejecutar el software bajo condiciones controladas para detectar posibles fallos, comportamientos inesperados o inconsistencias, facilitando su corrección para futuras mejoras.

Tras un análisis de las características y alcance de este proyecto, se ha determinado que la estrategia más eficiente para verificar la calidad del sistema de procesamiento de datos del nefelómetro se centra en los siguientes niveles de prueba:

- **Pruebas de integración:** Se enfocan en verificar la correcta interacción entre los distintos módulos del sistema cuando operan de manera conjunta. Estas pruebas verifican la adecuada comunicación y el intercambio de datos entre componentes.
- **Pruebas de sistema:** Evalúan el comportamiento integral del software cuando todos sus componentes están ensamblados. Permiten verificar el cumplimiento de los requisitos y especificaciones a nivel del sistema completo.
- **Pruebas de aceptación:** Se ejecutan con la participación directa de usuarios finales para confirmar que el sistema satisface sus necesidades reales y cumple con los criterios establecidos.

Es importante destacar que las pruebas se han llevado a cabo en los tres principales sistemas operativos: Windows, Linux y MacOS. Esta estrategia de verificación multiplataforma ha sido esencial

para garantizar la capacidad del sistema para funcionar correctamente en diversos entornos, lo cual representa una característica crítica del proyecto dada la diversidad de plataformas utilizadas por los usuarios potenciales en el ámbito científico y de investigación.

Este capítulo documenta los escenarios de prueba más significativos implementados en cada uno de los niveles mencionados, haciendo especial hincapié en las pruebas de sistema y aceptación.

Además, durante el ciclo de desarrollo iterativo, se han realizado verificaciones constantes después de implementar cada funcionalidad relevante, adaptando los métodos de prueba según las características particulares de cada etapa.

8.2 Pruebas de integración

A lo largo del proceso de desarrollo, se implementaron diversas pruebas de integración para examinar las interacciones entre los componentes críticos del sistema. Este esfuerzo se centró en verificar el correcto funcionamiento conjunto de las distintas etapas de la cadena de procesado: adquisición de datos del nefelómetro a tiempo real, transformación de estos datos (preprocesado) utilizando el driver propietario, procesamiento con GRASP y la posterior visualización de los resultados.

8.2.1. Integración de la adquisición de datos a tiempo real con el preprocesado

Para asegurar la correcta integración entre el módulo de adquisición de datos a tiempo real del nefelómetro IN102 y el sistema de preprocesado, se diseñaron pruebas específicas que verifican:

- La captura y transferencia efectiva de datos en tiempo real desde el dispositivo hacia los componentes de preprocesado.
- La sincronización adecuada entre la adquisición continua de datos y su transformación de formato .csv a .nc.
- La correcta aplicación del driver propietario en la conversión de los datos.
- El manejo apropiado de excepciones en la cadena de adquisición y preprocesado.
- La preservación de la integridad de los datos durante todo el proceso de transformación.

Estas pruebas se ejecutaron utilizando entornos simulados que reproducen la interacción completa entre el hardware y los componentes de software, permitiendo verificar el comportamiento del

sistema.

8.2.2. Integración del preprocesamiento con el algoritmo GRASP

La interacción entre el driver del nefelómetro IN102 y el algoritmo GRASP fue sometida a pruebas de integración que verifican:

- La compatibilidad completa de los formatos NetCDF generados con los requerimientos de entrada del algoritmo GRASP.
- El manejo adecuado de los diversos escenarios de datos y su impacto en el procesamiento posterior.
- La correcta generación de archivos CSV como resultado del procesamiento GRASP.

Estas verificaciones se realizaron utilizando conjuntos de datos reales y sintéticos, permitiendo evaluar el comportamiento de la fase de transformación .nc a .csv mediante el procesamiento con GRASP.

8.2.3. Integración de los resultados de GRASP con el módulo de visualización

La integración entre el componente que gestiona los resultados CSV del algoritmo GRASP y el módulo de visualización fue analizada mediante pruebas que evalúan:

- La correcta interpretación y presentación visual de los datos CSV procesados.
- La consistencia entre los resultados numéricos y su representación gráfica.
- La sincronización entre la disponibilidad de resultados y su visualización en tiempo real.

Para estas pruebas se emplearon revisiones visuales junto a usuarios experimentados que permitieron verificar esta fase de la cadena de procesado, desde la generación de resultados CSV hasta su representación visual final.

8.2.4. Integración del flujo de trabajo completo

El sistema en su conjunto fue sometido a pruebas de integración end-to-end que verifican la cadena completa de procesado:

- El correcto funcionamiento del flujo integrado: adquisición a tiempo real → preprocesado (driver) de .csv a .nc → procesado GRASP de .nc a .csv → visualización.
- La respuesta del sistema ante diferentes condiciones de adquisición de datos.
- El manejo apropiado de estados intermedios y la comunicación entre componentes.
- El correcto manejo de excepciones en la cadena de procesado.

Este conjunto de pruebas de integración ha sido esencial para garantizar que los componentes del sistema funcionan correctamente en conjunto, proporcionando un flujo de trabajo coherente y fiable a lo largo de toda la cadena de procesado. Debido al elevado número de pruebas realizadas, este documento recoge únicamente los aspectos más relevantes de la estrategia implementada durante el desarrollo.

8.3 Pruebas de sistema

Con el objetivo de validar el funcionamiento integral del software, se diseñaron y ejecutaron pruebas de sistema para cada uno de los casos de uso identificados en la fase de análisis. Estas pruebas permitieron evaluar el comportamiento coordinado de todos los componentes del sistema en escenarios que reflejan situaciones reales de uso.

Para cada caso de uso principal, se elaboró un escenario de prueba específico que valida su funcionamiento completo. A continuación, se presentan los casos de prueba.

Identificador	TS01
Precondiciones	El sistema está iniciado correctamente y el nefelómetro está conectado al puerto configurado.
Datos de Entrada	Configuración del puerto serial (puerto, baudrate, directorio de guardado de datos).
Resultado Esperado	El sistema establece conexión con el nefelómetro y recibe datos correctamente.
Escenario	1. El usuario accede a la sección de configuración del nefelómetro. 2. Introduce o selecciona los parámetros de conexión adecuados. 3. Presiona el botón para establecer la conexión. 4. El sistema intenta establecer conexión con el dispositivo. 5. El sistema muestra el panel grafico indicando que se ha establecido la conexión y comienza a recibir datos.
Trazabilidad	Prueba del caso de uso CU-01. Cargar datos del nefelómetro.
Ejecución	2024-04-22
Defectos encontrados	El sistema no muestra el puerto en la sección de configuración si el nefelómetro se ha conectado despues de iniciar el programa.

Tabla 8.1: Caso de prueba TS01

Identificador	TS02
Precondiciones	Datos del nefelómetro disponibles en formato CSV.
Datos de Entrada	Ruta del archivo CSV con datos del nefelómetro.
Resultado Esperado	Los datos se procesan correctamente y se genera un archivo en formato NetCDF compatible con GRASP.
Escenario	1. El usuario selecciona la pestaña manual de procesado. 2. Utiliza el explorador de archivos para seleccionar el archivo CSV. 3. Confirma la selección e inicia el procesamiento. 4. El sistema muestra una barra de progreso durante el procesamiento. 5. Al finalizar, el sistema muestra confirmación y la ubicación del archivo NetCDF generado.
Trazabilidad	Prueba del caso de uso CU-02. Preprocesar datos para GRASP.
Ejecución	2024-04-22
Defectos encontrados	Ninguno.

Tabla 8.2: Caso de prueba TS02

Identificador	TS03
Precondiciones	Tener disponible archivo NetCDF con datos preprocesados y archivo .yml de settings de GRASP.
Datos de Entrada	Ruta del archivo NetCDF con datos preprocesados y ruta del archivo .yml de settings de GRASP.
Resultado Esperado	El algoritmo GRASP se ejecuta correctamente y genera resultados en el directorio especificado.
Escenario	1. El usuario selecciona la opción de ejecutar algoritmo GRASP. 2. Selecciona procesamiento local. 3. Especifica el directorio de entrada mediante el explorador de archivos. 4. Especifica el directorio de salida mediante el explorador de archivos. 5. Inicia la ejecución. 6. El sistema muestra progreso durante la ejecución del algoritmo. 7. Al finalizar, el sistema notifica la finalización y resume los resultados generados.
Trazabilidad	Prueba del caso de uso CU-03. Ejecutar algoritmo GRASP.
Ejecución	2024-04-22
Defectos encontrados	Ninguno.

Tabla 8.3: Caso de prueba TS03

Identificador	TS04
Precondiciones	Resultados del algoritmo GRASP disponibles en un directorio.
Datos de Entrada	Ruta del directorio con resultados del procesamiento GRASP.
Resultado Esperado	Los resultados se visualizan correctamente con gráficos interactivos.
Escenario	1. El usuario accede a la sección de visualización de resultados. 2. Selecciona el fichero .csv de resultados mediante el explorador de archivos. 3. El sistema carga y procesa los archivos de resultados. 4. Se muestran diversos gráficos interactivos con los datos procesados. 5. El usuario interactúa con los gráficos (zoom, pan, etc.).
Trazabilidad	Prueba del caso de uso CU-04. Visualizar resultados.
Ejecución	2024-04-22
Defectos encontrados	Ninguno.

Tabla 8.4: Caso de prueba TS04

Identificador	TS05
Precondiciones	El nefelómetro está conectado y hay datos disponibles para procesar.
Datos de Entrada	Configuración del nefelómetro.
Resultado Esperado	La cadena completa de procesado se ejecuta correctamente desde la adquisición de datos hasta la visualización de resultados.
Escenario	1. Se ejecuta el caso de prueba TS01 para establecer conexión con el nefelómetro. 2. El sistema muestra los datos principales del nefelómetro en el panel gráfico. 3. Automáticamente se ejecuta el driver para convertir los datos CSV a formato NetCDF. 4. Se ejecuta el algoritmo GRASP con los datos preprocesados. 5. El sistema recorta el CSV obtenido para crear un fichero de salida con los datos procesados necesarios para la visualización. 6. Se añaden los datos procesados en el propio panel gráfico. 7. En intervalos temporales de 1 minuto, el sistema vuelve al paso 3, continuando con la cadena de procesado.
Trazabilidad	Prueba del caso de uso CU-05. Ejecutar cadena de procesado completa.
Ejecución	2024-04-22
Defectos encontrados	Ninguno.

Tabla 8.5: Caso de prueba TS05

Identificador	TS06
Precondiciones	Datos o resultados intermedios disponibles en el sistema.
Datos de Entrada	Selección del paso a ejecutar y parámetros específicos para ese paso.
Resultado Esperado	El paso individual se ejecuta correc tamente.
Escenario	1. El usuario selecciona el paso a ejecutar. 2. Configura los parámetros específicos para el paso seleccionado. 3. Inicia la ejecución del paso. 4. El sistema ejecuta solo el paso seleccionado y muestra el progreso. 5. Al finalizar, el sistema notifica la finalización y muestra los resultados de ese paso.
Trazabilidad	Prueba del caso de uso CU-06. Ejecutar paso individual de procesado.
Ejecución	2024-04-22
Defectos encontrados	Ninguno.

Tabla 8.6: Caso de prueba TS06

Identificador	TS07
Precondiciones	El sistema está correctamente iniciado.
Datos de Entrada	Parámetros de configuración personalizados (puerto COM, baudrate, rutas predeterminadas, URL para procesamiento remoto).
Resultado Esperado	Las preferencias se guardan correctamente y se aplican en la sesión actual y futuras.
Escenario	1. El usuario accede a la sección de preferencias. 2. Modifica varios parámetros en diferentes pestañas (configuración de la aplicación, rutas predeterminadas, configuración de procesamiento). 3. Cambia a otra sección de la aplicación. 4. Regresa a la sección de preferencias para verificar que los cambios persisten. 5. Reinicia la aplicación y verifica que los cambios se han guardado correctamente.
Trazabilidad	Prueba del caso de uso CU-07. Gestionar preferencias de usuario.
Ejecución	2024-04-22
Defectos encontrados	Ninguno.

Tabla 8.7: Caso de prueba TS07

8.4 Pruebas de aceptación

En el ciclo de desarrollo de software, las pruebas de aceptación representan una etapa determinante que permite verificar si el sistema cumple con los requisitos establecidos y está preparado para su despliegue en entornos productivos. Este tipo de evaluación confirma que las funcionalidades implementadas operan correctamente y, lo más importante, que el software responde eficazmente a las necesidades concretas de los usuarios finales.

En el marco de este proyecto, se organizaron varias sesiones formales de pruebas de aceptación con diferentes actores clave durante el mes de abril de 2024.

8.4.1. Estructura de las evaluaciones

Se realizaron tres sesiones de pruebas de aceptación siguiendo un protocolo estructurado:

1. **Reunión internacional online (3 de abril de 2024):** Sesión con John Hall, COO de Airphoton, donde:
   - Se realizó una demostración completa compartiendo pantalla.
   - Se presentaron las distintas funcionalidades del sistema.
   - Se mostró en detalle su funcionamiento operativo y capacidades de visualización.
2. **Sesión presencial técnica (10 de abril de 2024):** Encuentro con el Doctor Fernando Rejano, especialista del equipo GRASP de Valladolid, enfocado en:
   - Evaluación del manejo específico de datos provenientes del nefelómetro IN102.
   - Análisis de la precisión y utilidad del procesamiento de datos.
   - Validación técnica de los resultados por parte de un experto en la materia.
   - Análisis de la interfaz de usuario y la experiencia del usuario por parte de un usuario acostumbrado a estos procesamientos previamente realizados manualmente.
3. **Evaluación final online (24 de abril de 2024):** Prueba con Richard Kleidman, CCO de Airphoton, donde:
   - Se realizó una demostración similar a la primera sesión.
   - Se evaluaron las mejoras implementadas tras la retroalimentación previa.
   - La sesión concluyó con una valoración muy positiva del resultado final.

- Se propuso presentar la aplicación como producto inicial a los usuarios finales en un webinario previsto para mayo.

Durante todas las sesiones, se siguió un esquema que incluía: presentación general del sistema, demostración práctica, evaluación interactiva y retroalimentación detallada.

8.4.2. Valoración y conclusiones

Las evaluaciones concluyeron de manera altamente satisfactoria, con valoraciones positivas por parte de todos los participantes. Entre los aspectos destacados del sistema figuran:

1. **Integración efectiva del flujo de trabajo:** Los usuarios valoraron especialmente cómo el sistema unifica el proceso completo desde la captura de datos hasta la visualización de resultados, reduciendo significativamente la anterior complejidad operativa.
2. **Diseño accesible de la interfaz:** Se destacó particularmente la claridad de la interfaz gráfica, que permite realizar operaciones técnicamente complejas de manera sencilla.
3. **Capacidades avanzadas de visualización:** Las funcionalidades de visualización interactiva recibieron una valoración muy positiva, ya que facilitan notablemente la interpretación y análisis de los resultados generados por el algoritmo GRASP.

El interés mostrado por los ejecutivos de Airphoton, materializado en la propuesta de presentar la aplicación en un webinario dirigido a usuarios finales, confirma el potencial y la calidad del producto desarrollado. Todo ello, sin olvidar las posibles mejoras que se podrían realizar en futuras versiones.

La retroalimentación obtenida durante estas pruebas de aceptación ha sido documentada y será considerada para el plan de desarrollo de futuras versiones del producto, con el objetivo de mejorar progresivamente su funcionalidad y adaptarlo de manera más precisa a las necesidades de los usuarios.

Capítulo 9

Conclusiones y lineas futuras

9.1 Conclusiones del Proyecto

Este proyecto ha tenido como objetivo principal el desarrollo de una aplicación para la captura, procesamiento y visualización de datos de nefelómetros, proporcionando a los investigadores científicos una herramienta integral para su trabajo diario. A continuación, se presentan las principales conclusiones obtenidas durante el desarrollo y finalización del proyecto.

9.1.1. Logros Principales

El desarrollo de este trabajo ha permitido alcanzar varios logros significativos:

- **Integración completa con hardware especializado**: Se ha logrado una comunicación robusta con el nefelómetro, permitiendo la captura de datos en tiempo real, estableciendo un puente entre la aplicación y la instrumentación.
- **Procesamiento local/remoto**: La implementación de la posibilidad de procesamiento tanto local como remoto ofrece una flexibilidad importante a los usuarios, permitiéndoles trabajar en diversas condiciones.
- **Visualizaciones científicas avanzadas**: Las capacidades de visualización implementadas permiten a los investigadores explorar y analizar los datos de manera interactiva, facilitando la identificación de patrones y la interpretación de los resultados.
- **Interfaz intuitiva para usuarios científicos**: El diseño centrado en el usuario ha resultado

en una interfaz que minimiza la curva de aprendizaje y maximiza la productividad en entornos de investigación, con soporte multilingüe para adaptarse a la comunidad científica internacional.

- **Arquitectura extensible**: La arquitectura modular desarrollada permite la incorporación futura de nuevas funcionalidades y la adaptación a diferentes dispositivos.

9.1.2. Objetivos Cumplidos

Revisando los objetivos planteados al inicio del proyecto, podemos concluir que:

- Se ha implementado exitosamente la conexión y comunicación con el nefelómetro, permitiendo la configuración y captura de datos.
- El procesamiento de datos tanto en modo local como remoto funciona correctamente, ofreciendo resultados consistentes entre ambos modos.
- Las visualizaciones científicas desarrolladas cumplen con los requisitos de los usuarios y proporcionan las herramientas necesarias para el análisis de los datos.
- La interfaz de usuario ha sido validada por diferentes desarrolladores e investigadores científicos de múltiples disciplinas y países, confirmando su usabilidad y adecuación a los flujos de trabajo reales en diversos entornos de investigación.
- El sistema ha demostrado ser robusto y fiable en las pruebas realizadas, con una gestión adecuada de errores y excepciones.

9.1.3. Aprendizaje Personal

Este proyecto ha supuesto un importante proceso de aprendizaje personal en varios aspectos, destacando la naturaleza multidisciplinar e internacional del desarrollo:

- Profundización en el desarrollo de aplicaciones gráficas y la integración con hardware especializado.
- Mejora de habilidades en la gestión de proyectos complejos y la planificación de iteraciones en entornos colaborativos internacionales.
- Experiencia en la colaboración con usuarios finales de diversos orígenes científicos y la adaptación del desarrollo a sus necesidades especializadas.

- Ampliación de conocimientos en tecnologías específicas ampliamente utilizadas en la situación tecnológica actual.

- Desarrollo de competencias interculturales y lingüísticas a través de la comunicación científica en inglés, facilitando el intercambio efectivo de ideas con los interesados y la integración de sus diversas perspectivas en el diseño e implementación del sistema.

9.1.4. Conclusión General

El proyecto ha culminado con éxito, entregando una aplicación multiplataforma que responde a una necesidad real en la comunidad de ciencias ambientales y teledetección. La herramienta desarrollada no solo cumple con los requisitos funcionales especificados, sino que proporciona una base sólida para futuras mejoras y ampliaciones.

El enfoque multidisciplinar, integrando ciencias de la computación y física atmosférica, ha sido vital para crear una solución integral que aborda las necesidades del análisis de datos del nefelómetro. Esta integración de diversos dominios de conocimiento ha enriquecido el proyecto, resultando en una herramienta que habla el lenguaje de los usuarios científicos mientras aprovecha los principios de ingeniería de software.

La estrecha colaboración con usuarios finales de diversos orígenes científicos ha sido clave para el éxito del proyecto, permitiendo ajustar el desarrollo a sus necesidades reales y garantizando la utilidad práctica de la aplicación en diversos entornos de trabajo.

El enfoque metodológico iterativo e incremental adoptado ha demostrado ser adecuado para este tipo de proyecto, permitiendo reuniones periódicas con los usuarios interesados y una adaptación continua a los desafíos encontrados y a las nuevas necesidades identificadas durante el desarrollo.

9.2 Líneas futuras

Aunque el proyecto ha alcanzado sus objetivos principales, existen numerosas oportunidades para expandir y mejorar la aplicación en el futuro. A continuación, se presentan las líneas de trabajo futuro más prometedoras.

9.2.1. Mejoras técnicas

- **Ampliación de la compatibilidad**: Extender el soporte a otros modelos de nefelómetros y dispositivos similares, creando interfaces genéricas para facilitar la integración.
- **Mejoras en la comunicación con el dispositivo**: Perfeccionar y ampliar los protocolos de comunicación con el nefelómetro para aumentar la velocidad y fiabilidad de la transferencia de datos.

9.2.2. Nuevas funcionalidades

- **Análisis comparativo**: Desarrollar herramientas para comparar automáticamente diferentes conjuntos de datos, facilitando estudios temporales y espaciales.
- **Validación y control de calidad**: Desarrollar mecanismos automáticos para la validación de datos y detección de mediciones anómalas o erróneas.

9.2.3. Mejoras en la interfaz de usuario

- **Personalización de visualizaciones**: Permitir a los usuarios personalizar completamente las visualizaciones científicas según sus necesidades específicas de análisis.
- **Módulo de ayuda interactiva**: Desarrollar un sistema de ayuda contextual que guíe a los usuarios en tiempo real durante el uso de la aplicación.

9.2.4. Distribución y comunidad

- **Documentación multilingüe**: Ampliar la documentación en varios idiomas para facilitar su adopción internacional.
- **Programa de mejora continua**: Establecer un mecanismo para recopilar sistemáticamente feedback de los usuarios e incorporarlo en futuras versiones.

Estas líneas futuras representan oportunidades significativas para expandir el alcance y la utilidad de la aplicación desarrollada. Su implementación permitiría no solo mejorar las capacidades actuales del sistema, sino también adaptarlo a un entorno en constante evolución, respondiendo a las necesidades cambiantes de la comunidad investigadora.

Apéndices

Capítulo A

Acrónimos

- **API**: Application Programming Interface (Interfaz de Programación de Aplicaciones).
- **CCO**: Chief Communications Officer (Director de Comunicaciones).
- **COM**: Component Object Model (Modelo de Objetos Componentes).
- **COO**: Chief Operating Officer (Director de Operaciones).
- **CSS**: Cascading Style Sheets (Hojas de Estilo en Cascada).
- **CSV**: Comma-Separated Values (Valores Separados por Comas).
- **CU**: Caso de Uso.
- **GRASP**: Generalized Retrieval of Atmosphere and Surface Properties
- **GUI**: Graphical User Interface (Interfaz Gráfica de Usuario).
- **HP**: Hewlett-Packard.
- **HTML**: HyperText Markup Language (Lenguaje de Marcado de Hipertexto).
- **HTTP**: HyperText Transfer Protocol (Protocolo de Transferencia de Hipertexto).
- **IDE**: Integrated Development Environment (Entorno de Desarrollo Integrado).
- **IRPF**: Impuesto sobre la Renta de las Personas Físicas.
- **MVC**: Model-View-Controller (Modelo-Vista-Controlador).
- **REST**: Representational State Transfer (Transferencia de Estado Representacional).
- **SSH**: Secure Shell (Intérprete de Órdenes Seguro).

- **TFG**: Trabajo de Fin de Grado.
- **URL**: Uniform Resource Locator (Localizador Uniforme de Recursos).
- **USB**: Universal Serial Bus (Bus Universal en Serie).
- **UTF**: Unicode Transformation Format (Formato de Transformación Unicode).
- **UUID**: Universally Unique Identifier (Identificador Único Universal).
- **YAML**: YAML Ain't Markup Language (YAML No es un Lenguaje de Marcado).

Capítulo B

Manual de despliegue

Este apéndice proporciona instrucciones detalladas para la instalación y despliegue de la aplicación, tanto en un entorno de desarrollo como en un entorno de usuario final.

B.1 Instalación para usuarios finales

Para facilitar la instalación de la aplicación en sistemas Windows, se ha desarrollado un instalador que automatiza todo el proceso. Este instalador se encarga de la configuración de todas las dependencias necesarias, permitiendo a los usuarios comenzar a utilizar la aplicación inmediatamente sin necesidad de conocimientos técnicos avanzados.

B.2 Requisitos previos para desarrollo

La aplicación es compatible con versiones de Python **3.10 a 3.13**, pero para el desarrollo se recomienda usar Python 3.10. Además, se recomienda el uso de un entorno virtual con `conda` para garantizar una configuración aislada y evitar problemas de compatibilidad. Para esto, se puede instalar Miniconda o Anaconda y crear un entorno específico para el proyecto.

También se requiere:

- **Git** para clonar el repositorio
- **Docker** para ejecutar la API GRASP

- **Cuenta en code.grasp-open** con una clave SSH previamente configurada

B.3 Proceso de instalación para desarrollo

B.3.1. Clonar el repositorio

Abrir un Terminal en la carpeta donde se desee guardar el repositorio ap-instrument-gui. Asegurarse de tener configurada una clave SSH en la cuenta de code.grasp-open y ejecutar:

```
git clone git@code.grasp-open.com:airphoton/software/
    airphoton-instrument-suite/ap-instrument-gui.git
```

B.4 Ejecutar el software

1. **Una vez clonado el repositorio**, navegar a la carpeta creada en el proceso de clonado. Para verificar que se está en la carpeta correcta, comprobar la existencia del archivo pyproject.toml.

2. **Instalar poetry**: Si no se tiene poetry instalado, ejecutar el siguiente comando:

   ```
   pip install poetry
   ```

   Si eso no funciona, se puede intentar instalarlo a través del gestor de paquetes:

   ```
   # En Ubuntu/Debian
   apt install python3-poetry

   # En MacOS
   brew install poetry
   brew install python-tk # En MacOS se debe instalar python-tk localmente
   ```

3. **Ejecutar el proyecto**: Configurar el software con los siguientes comandos:

   ```
   poetry install
   ```

Si el comando poetry install falla, es posible que poetry no tenga las dependencias necesarias para instalarlo. En ese caso, intentar:

```
# En Ubuntu/Debian
sudo apt install pkg-config
sudo apt install libcairo2-dev

# En MacOS
brew install pkg-config
brew install cairo
```

Luego intentar ejecutar el comando poetry install nuevamente. Si finaliza sin errores, ejecutar:

```
poetry run ap_gui
```

Esto debería lanzar la aplicación sin ningún error.

En caso de que este último comando devuelva un error como este:

```
Traceback (most recent call last):
File "/path/to/proyect/ap-instrument-gui/ap/instrument/gui/main.py",
line 5, in <module>
    from ap.instrument.gui.interface_ui import Ui_MainWindow
ModuleNotFoundError: No module named 'ap.instrument.gui'
```

Se puede añadir el proyecto al PYTHONPATH usando:

```
export \hyperref[acro:pythonpath]{PYTHONPATH}=$PYTHONPATH:/path/to/project/
```

Y ejecutar el comando poetry run nuevamente.

B.5 Configuración del instrumento

Si se desea conectar un instrumento, es posible encontrar un error como este:

```
Error reading from serial port /dev/ttyUSB0: [Errno 13] could not open port /dev/ttyUSB0:
[Errno 13] Permission denied: '/dev/ttyUSB0'
```

Si esto ocurre, se debe abrir el puerto especificado para permitir que la aplicación comience a recibir datos del instrumento. Esto se puede hacer a través del terminal:

Cambiar permisos temporalmente con `chmod`:

```
~$ sudo chmod 666 /dev/ttyUSB0
```

Uso del nefelómetro in102: Al utilizar el nefelómetro in102, asegurarse de seleccionar el puerto y la tasa de baudios correctos:

1. **Selección de puerto:** Elegir el puerto serie correcto. Si hay varios puertos disponibles, es probable que sea el último de la lista.
2. **Baudrate:** Establecer el baudrate a **9600** para una comunicación de datos precisa.

Esta configuración permitirá recibir datos del nefelómetro a través de la aplicación.

Capítulo C

Manual de uso

Este manual proporciona instrucciones para el uso de la aplicación, detallando la funcionalidad de cada módulo y los pasos necesarios para operar correctamente el sistema.

C.1 Primeros pasos

Una vez iniciada la aplicación, se mostrará la interfaz principal mostrada en la Figura C.1. La interfaz muestra una barra lateral con varias funcionalidades de módulos y botones de configuración del instrumento.

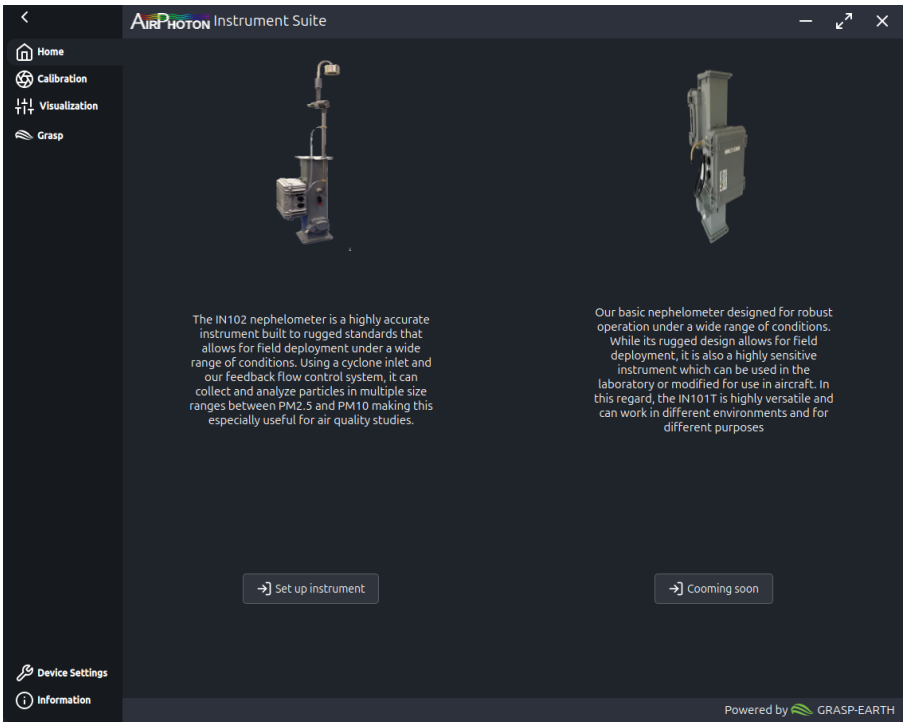


Figura C.1: Interfaz de inicio de la aplicación con barra de navegación lateral. Elaboración Propia.

C.2 Configuración del Módulo GRASP

C.2.1. Configuración de la imagen Docker

Si no se dispone de una imagen Docker con la API GRASP instalada, el usuario puede descargar la imagen desde el repositorio privado de grasp con una cuenta con acceso:

1. Navegar a la sección GRASP usando la barra lateral
2. Hacer clic en el botón de descarga como se muestra en la Figura C.2
3. Iniciar sesión en el servicio Docker (Figura 7.7)
4. Descargar la imagen de la API GRASP

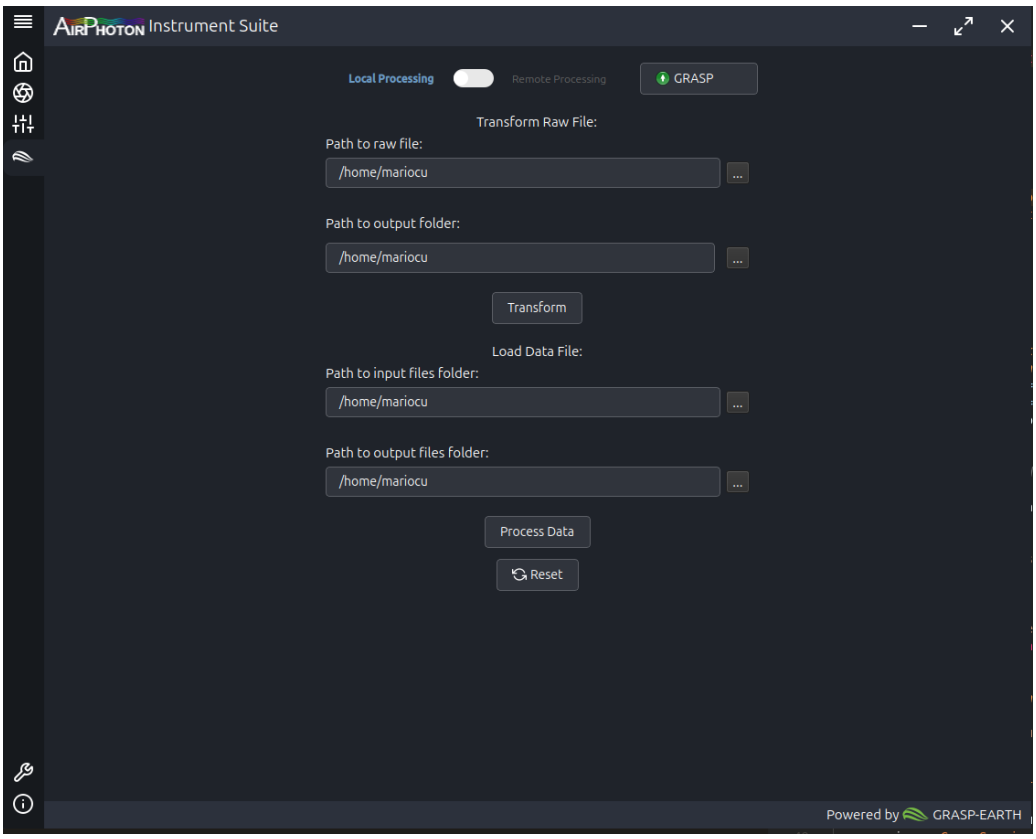


Figura C.2: Interfaz del módulo GRASP. Elaboración Propia.

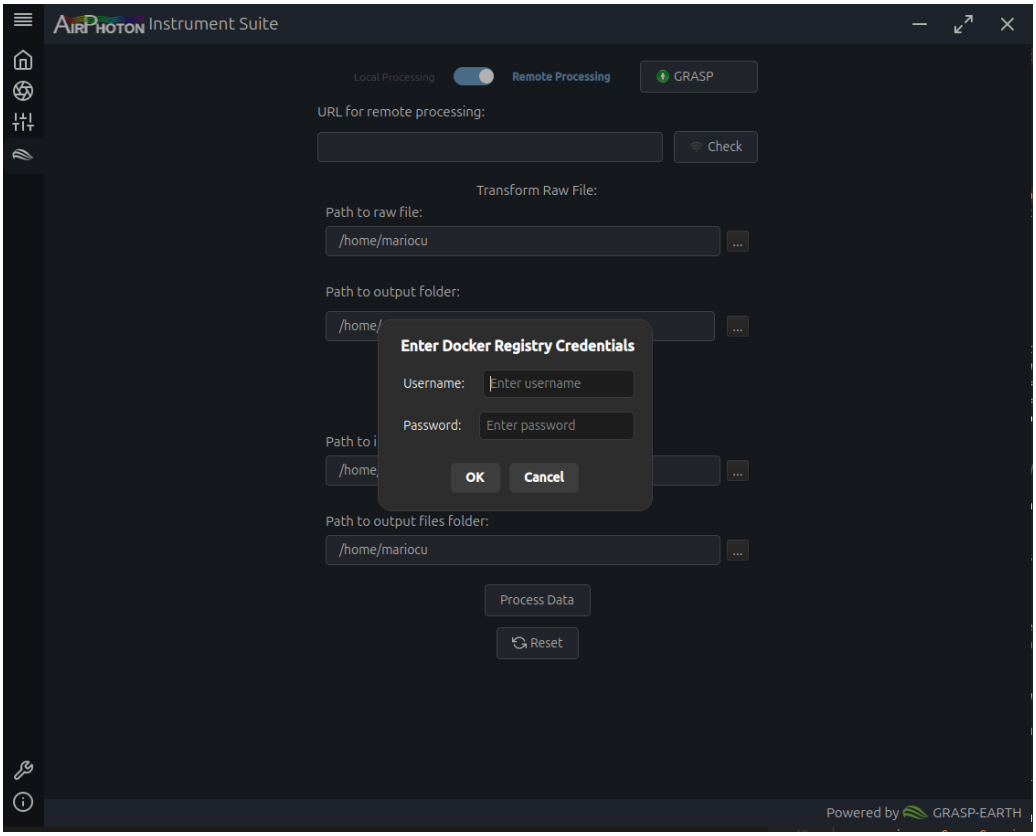


Figura C.3: Interfaz de inicio de sesión de Docker. Elaboración Propia.

C.2.2. Pasos de procesamiento individual

Una vez obtenida la imagen, puede realizar los siguientes pasos de procesamiento individuales:

- **Preprocesamiento**: Utilice el driver para convertir los datos brutos del nefelómetro al formato requerido por GRASP
- **Procesamiento con GRASP**: Este paso puede realizarse de dos formas:
  - **Local**: Ejecuta el algoritmo GRASP en su máquina local utilizando la imagen Docker
  - **Remoto**: Se debe añadir una URL que aloje la API GRASP y hacer clic en el botón Check para verificar el estado del servidor. Un diálogo confirmará si el servidor está listo para procesar datos

C.3 Visualización de datos

Para visualizar datos procesados:

1. Navegue a la pestaña Visualización usando la barra lateral
2. Seleccione el archivo introduciendo la ruta o utilizando el explorador de archivos como se muestra en las Figuras C.4 y C.5

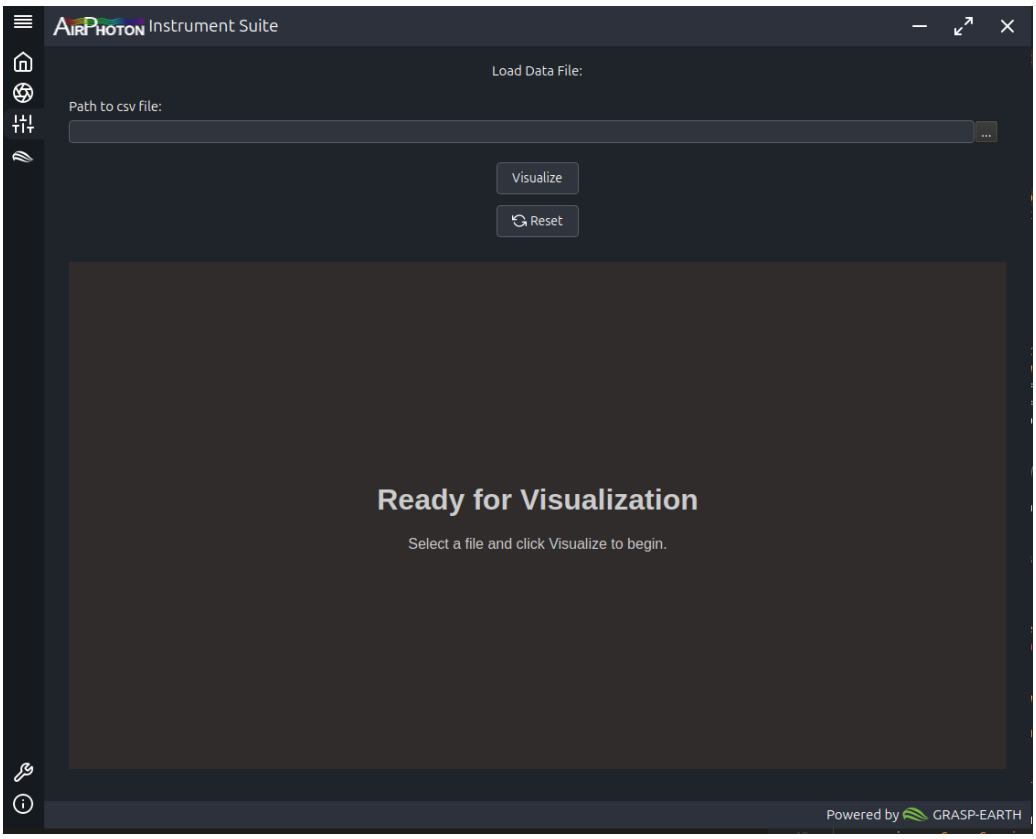


Figura C.4: Interfaz del módulo de visualización. Elaboración Propia.

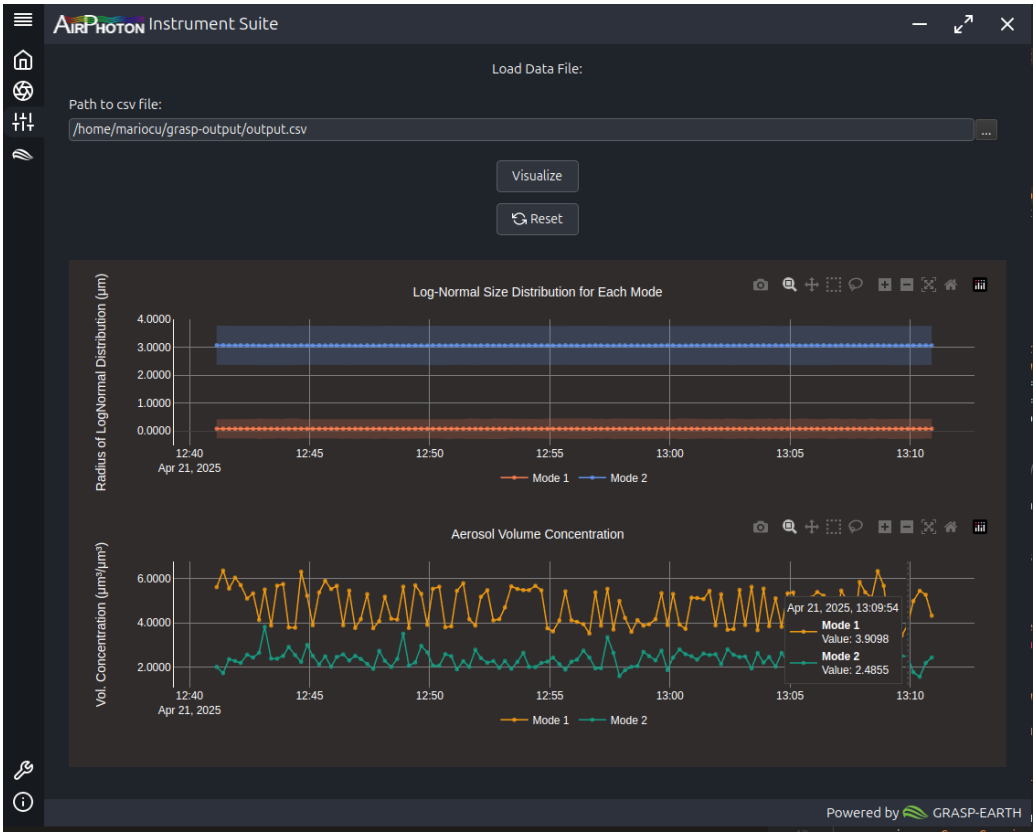


Figura C.5: Interfaz de visualización de datos. Elaboración Propia.

C.4 Cadena de procesamiento automatizada

Para ejecutar la cadena de procesado automatizada:

1. Acceder al menú de Device Settings haciendo clic en el botón Set up instrument en la pantalla de inicio o seleccionándolo desde la barra lateral
2. Configurar el instrumento con los parámetros apropiados en el menú de configuración y hacer clic en el botón Connect (Figura C.6)
3. El sistema iniciará la cadena de procesamiento y mostrará una visualización de los datos entrantes
4. La visualización se actualizará periódicamente con los datos recibidos y los resultados del procesamiento, proporcionando un panel de control similar al mostrado en la Figura 7.5
5. Para detener la cadena de procesamiento, hacer clic en el botón de desconexión
6. Todos los datos se almacenarán en la carpeta seleccionada, y se podrá seguir utilizando la aplicación con normalidad

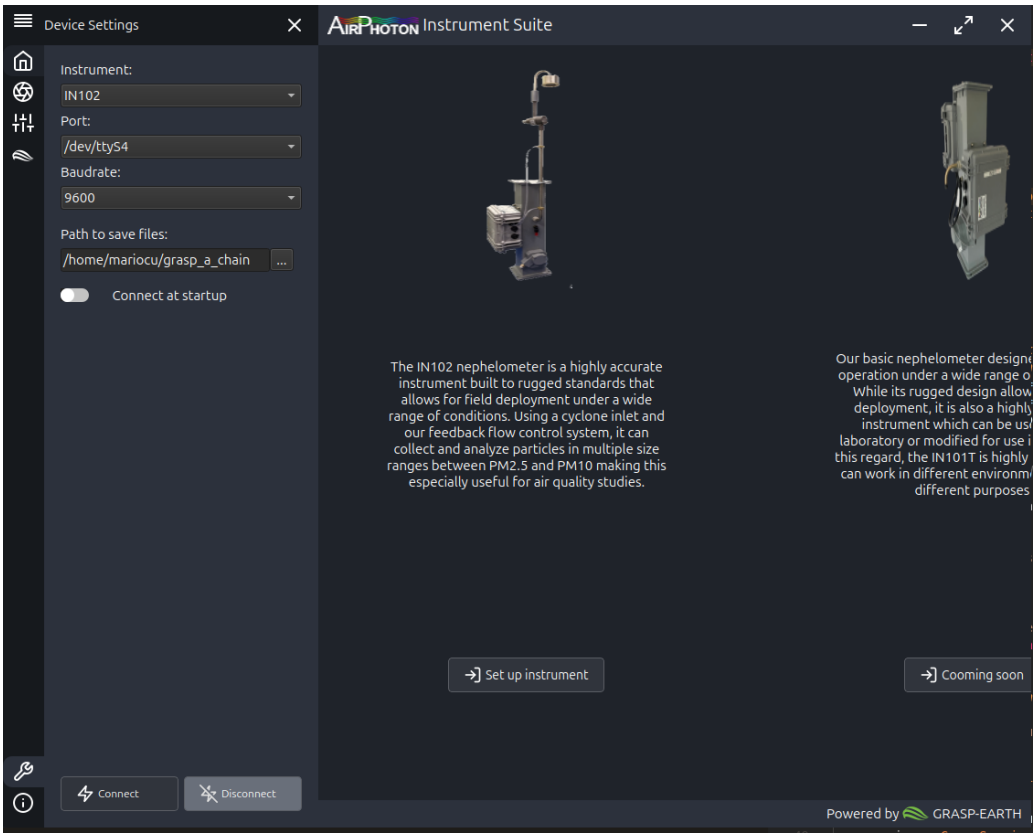


Figura C.6: Interfaz de configuración del dispositivo. Elaboración Propia.

Figura C.7: Panel de control de la aplicación durante el funcionamiento. Elaboración Propia.

Capítulo D

Contenido del TFG

El código fuente del proyecto se encuentra en el repositorio privado de la empresa. Sin embargo, para esta memoria se ha realizado una copia del código final en el repositorio de la Escuela de Ingeniería Informática de la Universidad de Valladolid. Los repositorios relevantes son:

- `https://gitlab.inf.uva.es/marcobr/ap-instrument-gui`
- `https://gitlab.inf.uva.es/marcobr/ap-instrument-grasp`
- `https://gitlab.inf.uva.es/marcobr/ap-instrument-in102`
- `https://gitlab.inf.uva.es/marcobr/grasp-api`

Bibliografía

[1] M. Hussain, P. Madl y A. Khan. «Lung deposition predictions of airborne particles and the emergence of contemporary diseases Part-I». En: *The Health* 2 (2011), págs. 51-59.

[2] Dimitra Karali, Glykeria Loupa y Spyridon Rapsomanikis. «Nephelometer sensitivities for the determination of PM2. 5 mass concentration in ambient and indoor air». En: *Aerosol and Air Quality Research* 21.1 (2021). Publisher: Springer, pág. 200159.

[3] Arun D Shendrikar y William K Steinmetz. «Integrating nephelometer measurements for the airborne fine particulate matter (PM2. 5) mass concentrations». En: *Atmospheric Environment* 37.9-10 (2003). Publisher: Elsevier, págs. 1383-1392.

[4] *Ow.ly - image uploaded by @henrikkniberg*. URL: http://ow.ly/i/659WS.

[5] Steven Thomas. *Revisiting the Iterative Incremental Mona Lisa*. en-US. Dic. de 2012. URL: https://itsadeliverything.com/revisiting-the-iterative-incremental-mona-lisa (visitado 21-03-2025).

[6] *Desarrollo iterativo y creciente*. es. Page Version ID: 165415821. Feb. de 2025. URL: https://es.wikipedia.org/w/index.php?title=Desarrollo_iterativo_y_creciente&oldid=165415821.

[7] *The Pros and Cons of Iterative Software Development*. Mar. de 2017. URL: https://www.one-beyond.com/pros-cons-iterative-software-development/ (visitado 20-03-2025).

[8] NimbleWork. *Iterative And Incremental Development - Agile Methodology*. 2023. URL: https://www.nimblework.com/agile/iterative-and-incremental-development/ (visitado 20-03-2023).

[9] GeeksforGeeks. *Advantages and Disadvantages of Incremental Process Model*. 2023. URL: https://www.geeksforgeeks.org/advanategs-and-disadvanategs-of-incremental-process-model/ (visitado 20-03-2023).

[10] Sphere Partners. *Iterative vs. Incremental Development*. 2023. URL: https://www.sphereinc.com/blogs/iterative-vs-incremental-development/ (visitado 20-03-2023).

[11] *Iterative vs. Incremental Development - Software Engineering Perspectives*. 2023. URL: https://builtin.com/software-engineering-perspectives/iterative-vs-incremental (visitado 20-03-2025).

[12] Oleg Dubovik et al. «GRASP: a versatile algorithm for characterizing the atmosphere». En: *Spie Newsroom* 25.10.1117 (2014). Publisher: SPIE-Intl Soc Optical Eng, págs. 2-1201408.

[13] Oleg Dubovik et al. «A comprehensive description of multi-term LSM for applying multiple a priori constraints in problems of atmospheric remote sensing: GRASP algorithm, concept, and applications». En: *Frontiers in Remote Sensing* 2 (2021). Publisher: Frontiers Media SA, pág. 706851.

[14] *GRASP OPEN*. en-US. URL: https://www.grasp-open.com/.

[15] W. R. Espinosa, J. Vanderlei Martins y R. Levy. «Retrievals of aerosol size distribution, spherical fraction, and complex refractive index from airborne in situ angular light scattering and absorption measurements». En: *Journal of Geophysical Research: Atmospheres* 124 (2019), págs. 7997-8024.

[16] *Gestión de RIESGOS PMBOK 6 | SoyPM*. es. Oct. de 2020. URL: https://www.soypm.website/area-de-conocimiento/gestion-de-riesgos/.

[17] *Sueldo: Junior Software Engineer en España 2025*. es. URL: https://www.glassdoor.es/Sueldos/junior-software-engineer-sueldo-SRCH_KO0,24.htm.

[18] *Average Computer Lifespan: How Long Do PCs Last? | HP® Tech Takes*. en-us. URL: https://www.hp.com/us-en/shop/tech-takes/average-computer-lifespan.

[19] *Airphoton IN102 Airphoton Size Scanning Nephelometer*. en. URL: https://www.environmental-expert.com/products/airphoton-model-in102-airphoton-size-scanning-nephelometer-496623.

[20] *Plotly*. URL: https://plotly.com/api/.

[21] *Modelo–vista–controlador*. es. Page Version ID: 165920571. Mar. de 2025. URL: https://es.wikipedia.org/w/index.php?title=Modelo%E2%80%93vista%E2%80%93controlador&oldid=165920571.

[22] *Observer*. en. URL: https://refactoring.guru/design-patterns/observer.

[23] *Decorator*. en. URL: https://refactoring.guru/design-patterns/decorator.

[24] *Composite*. en. URL: https://refactoring.guru/design-patterns/composite.

[25] *Welcome to Python.org*. en. Mar. de 2025. URL: https://www.python.org/.

[26] *Poetry - Python dependency management and packaging made easy*. URL: https://python-poetry.org/.

[27] *Git*. URL: https://git-scm.com/.

[28] *Cursor - The AI Code Editor*. en. URL: https://www.cursor.com/.

[29] *Visual Studio Code - Code Editing. Redefined*. en. URL: https://code.visualstudio.com/.

[30] *yaml*. URL: https://yaml.dev/doc/ruamel.yaml/.

[31] *loguru*. URL: https://loguru.readthedocs.io/en/stable/.

[32] *Docker: Accelerated Container Application Development*. en-US. Mayo de 2022. URL: https://www.docker.com/.

[33] *NetCDF*. es. Page Version ID: 158263318. Feb. de 2024. URL: https://es.wikipedia.org/w/index.php?title=NetCDF&oldid=158263318.

[34] *Welcome to Flask — Flask Documentation (3.1.x)*. URL: https://flask.palletsprojects.com/en/stable/.

[35] *Qt | Development Framework for Cross-platform Applications*. en. URL: https://www.qt.io/product/framework.

[36] *Data Apps for Production | Plotly*. en. URL: https://plotly.com/.