Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Ingeniería Informática

Trabajo Fin de Grado

Utilización de la herramienta GIMP (GNU Image Manipulation Program) y extensión de sus funcionalidades mediante scripts.

Autor: Nicolás Bravo Ruiz de la Prada

Tutor(a): José Crespo del Arco

Madrid, Junio de 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Ingeniería Informática
Título: Utilización de la herramienta GIMP (GNU Image Manipulation Program) y extensión de sus funcionalidades mediante scripts.
Junio 2025

Autor: Nicolás Bravo Ruiz de la Prada

Tutor:
José Crespo del Arco
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Este Trabajo de Fin de Grado consiste en la extensión de las funcionalidades del programa GIMP (GNU Image Manipulation Program) desarrollando scripts personalizados, con el objetivo de automatizar tareas repetitivas de procesamiento gráfico. GIMP es una potente herramienta de código abierto, la cual, además de poder ser usada interactivamente, permite la creación de scripts en distintos lenguajes para su extensión, por lo que posibilita su uso desde la terminal y su integración en flujos de trabajo automatizados.

El proyecto nace del interés por explorar las capacidades de GIMP para scripting en entornos donde es recurrente el tratamiento masivo de imágenes, como puede ser el análisis científico o técnico. Para ello, se ha trabajado con GIMP en su versión 3.0, instalada en un sistema Linux mediante Flatpak, lo que ha permitido acceder a un entorno aislado, actualizado y compatible con las últimas tecnologías.

El desarrollo del trabajo se ha orientado hacia la implementación de scripts usando principalmente el lenguaje propio de GIMP a través de Script-Fu, Scheme, aunque también se ha explorado la posibilidad de extensión con Python-Fu. Entre las funcionalidades que se han implementado se encuentran: la conversión de formatos de imagen, la binarización, la adición de texto, la expansión de una determinada selección dentro de una imagen y el resaltado de objetos con el mismo color. Algunas de ellas, dependiendo de su funcionalidad, pudiendo ser ejecutadas desde la terminal. Este enfoque permite aprovechar GIMP como una herramienta potente y versátil, más allá de su uso convencional en edición gráfica.

En conjunto, este trabajo demuestra que la herramienta GIMP, gracias a su capacidad de scripting, puede transformarse en una plataforma eficaz para la automatización de tareas gráficas, con aplicaciones que abarcan desde la edición básica hasta el análisis técnico o científico de imágenes y la aplicación de transformaciones.

Abstract

This Final Degree Project consists of extending the functionality of the program GIMP (GNU Image Manipulation Program) by developing custom scripts, with the goal of automating repetitive graphic processing tasks. GIMP is a powerful open-source that, in addition to being usable interactively, allows the creation of scripts in different languages for its extension, enabling its use from the terminal and its integration into automated workflows.

The project was born from the interest in exploring GIMP's scripting capabilities in enviroments where massive image processing is common, such as scientific or technical analysis. To this end, we worked with GIMP version 3.0, installed on a Linux system using Flatpak, which provided access to an isolated, updated environment compatible with the latest technologies.

The work has focused on implementing scripts primarily using GIMP's ownscripting language, Scheme, although the possibility of extending it with Python-Fu has also been explored. Implemented features include image format conversion, binarization, adding text, expanding a selection within an image, and highlighting objects with the same color. Some of these features, depending on their functionality, can be executed from the terminal. This approach allows GIMP to be leveraged as a powerful and versatile tool, beyond its conventional use in graphic editing.

Overall, this work demonstrates that GIMP, thanks to its scripting capabilities, can be transformed into an effective platform for automasing graphic tasks, with applications ranging from basic editing to technical or scientific image analysis.

Tabla de contenidos

1 Introducción

A lo largo de las últimas décadas, el procesamiento de imágenes se ha consolidado como una disciplina indispensable en numerosos campos, desde la medicina o la biología hasta la ingeniería, la astronomía o el arte digital. Según las tecnologías de captura y generación de imágenes han ido evolucionando, también lo han hecho las herramientas dedicadas al análisis, edición y automatización de estas mismas. En este contexto, el uso de scripts para la automatización de tareas repetitivas ha recibido una importancia bastante relevante, al ser capaces de reducir la carga de trabajo manual, minimizar los errores humanos, estandarizar los resultados y agilizar los procesos que requerían una inversión en tiempo y en esfuerzo.

Entre las herramientas disponibles, GIMP se presenta como una de las opciones de código abierto más robustas y versátiles [1]. Aunque normalmente se le ha vinculado al ámbito de la edición gráfica, su arquitectura extensible y su compatibilidad con lenguajes de scripting como Scheme y Python han favorecido su uso en áreas académicas y científicas [2]. Gracias a estas capacidades, GIMP permite automatizar operaciones sobre imágenes tanto de forma interactiva como desde una terminal de forma más automática, lo que convierte la herramienta en una solución eficiente y adaptable para investigadores, técnicos y estudiantes que necesitan procesar grandes volúmenes de imágenes con criterios precisos y repetibles.

El presente trabajo persigue la exploración, desarrollo y análisis de técnicas de scripting en GIMP para aprovechar su potencial como motor de automatización gráfica y, en particular, los objetivos que se plantean son el estudio en profundidad de la herramienta y sus mecanismos de extensión mediante scripts, la selección de las funcionalidades más relevantes para un entorno técnico, la implementación de dichas funcionalidades tanto en Scheme como en Python, y acompañar el código con ejemplos ilustrativos y documentación detallada que faciliten su comprensión y futura realización. Para ello, se ha optado por utilizar el lenguaje nativo de GIMP, Scheme, como principal lenguaje, gracias a su consistencia y estabilidad en su última versión, 3.0. Esta versión, instalada mediante el sistema Flatpak en un entorno Linux, proporciona una base técnica adecuada para el desarrollo de scripts reutilizables y ejecutables desde la terminal, lo que permite integrarlos fácilmente en flujos de trabajo automatizados [3].

A lo largo del proyecto se implementan distintos scripts enfocados en tareas comunes relacionadas con el procesamiento digital, como la conversión de formatos, la binarización, la inserción de texto o la generación de máscaras. Estos scripts, aparte de buscar la demostración de la posibilidad de extensión de funcionalidades en GIMP mediante el scripting, también ofrecen soluciones prácticas a necesidades reales de automatización. De la misma manera, el proyecto propone una reflexión sobre las capacidades del programa como plataforma extensible, analizando que tipo de tareas y perfiles de usuario se benefician más de estas, ya sea para agilizar tareas simples y repetitivas como para aquellos que necesitan un control detallado sobre las funcionalidades mediante la programación.

El documento se estructura en ocho capítulos que guían al lector desde los fundamentos hasta las conclusiones finales. Tras esta introducción, en la que se contextualiza el trabajo y se exponen sus objetivos, se revisa de forma crítica los trabajos previos, la historia de scripting en GIMP y sus lenguajes de extensión. A continuación se expone el desarrollo llevado a cabo, describiendo el entorno experimental, comparando las versiones del programa, justificando la elección del lenguaje utilizado, presentando la arquitectura modular de los scripts y detallando cada funcionalidad implementada sin olvidar su respectiva automatización en la terminal y las incidencias resueltas. Más adelante se muestran las pruebas realizadas y los resultados obtenidos de cada script, apoyados en ejemplos ilustrativos. Después se profundiza en las diversas aplicaciones prácticas que demuestran la utilidad real de las herramientas desarrolladas. Hacia el final del documento se recogen las conclusiones y se proponen futuras líneas de mejora, seguidas de una reflexión sobre el impacto técnico y social del proyecto. El trabajo se cierra con la bibliografía consultada, que garantiza la trazabilidad académica de todas las fuentes utilizadas.

Con todo ello, este trabajo busca profundizar en las posibilidades de scripting como mecanismo de extensión funcional que ofrece GIMP, y también valorar críticamente su papel dentro de los entornos actuales, destacando su aplicabilidad en ámbitos donde la automatización resulta esencial.

2 Trabajos Previos

El uso de scripts en GIMP ha sido objeto de múltiples iniciativas, publicaciones técnicas y proyectos independientes que han buscado extender el alcance del programa más allá de su uso tradicional como editor gráfico. En sus primeras versiones, GIMP incorporaba Script-Fu como subsistema nativo para la ejecución de scripts escritos en una variante del lenguaje Scheme [4]. Esta elección respondía a la intención de ofrecer a los usuarios una herramienta compacta, funcional y eficiente, con capacidad de definir nuevos filtros, automatizar procesos por lotes y en algunos casos, generar interfaces básicas de usuario. No obstante, a pesar de su potencia, Scheme ha sido considerado tradicionalmente poco accesible debido a su sintaxis y a su paradigma funcional, lo que ha complicado su adopción fuera de entornos especializados.

Como respuesta a estas limitaciones, GIMP integró soporte para Python a través del módulo Python-Fu. Esta incorporación supuso una mejora significativa en términos de accesibilidad y versatilidad, ya que permitió a los usuarios crear scripts con una sintaxis más familiar y trabajar de forma más directa con estructuras de datos complejas [5]. Además, facilitó el uso de bibliotecas externas del ecosistema de Python. Esta evolución fue especialmente relevante en contextos científicos, donde Python ya se había consolidado como lenguaje principal para análisis de datos, inteligencia artificial y visualización. A partir de entonces, GIMP comenzó a integrarse en flujos de trabajo automatizados en áreas como la microscopía, la geología o la ingeniería biomédica, donde el procesamiento masivo de imágenes es una necesidad habitual [6].

Otro factor clave en la evolución del scripting en GIMP ha sido el desarrollo progresivo de su documentación. La comunidad de usuarios y desarrolladores ha contribuido con tutoriales, ejemplos prácticos y quías detalladas que explican cómo interactuar con la API, registrar nuevos procedimientos y manipular imágenes directamente desde scripts [7]. Con el lanzamiento de GIMP 3.0, esta documentación se ha actualizado para adaptarse a los nuevos estándares del programa [8]. Esta modernización ha mejorado el rendimiento general, la compatibilidad con pantallas de alta resolución y el soporte para lenguajes de programación contemporáneos.

Además, existen varias investigaciones y trabajos académicos que han utilizado el scripting en GIMP como herramienta de apoyo para el análisis automatizado de imágenes. En ámbitos científicos se ha demostrado que los scripts permiten realizar tareas como la generación de máscaras, el cálculo de histogramas, la separación de canales o la aplicación de umbrales con parámetros configurables [9]. Todo ello contribuye a reducir significativamente el tiempo de trabajo manual sin comprometer la precisión de los resultados.

En resumen, los trabajos previos muestran cómo el scripting en GIMP ha evolucionado desde un enfoque funcional básico hacia un ecosistema más versátil y accesible, especialmente gracias a la integración de Python. Esta transición ha abierto la herramienta a nuevos perfiles de usuarios y ha facilitado su integración con otros lenguajes, bibliotecas y herramientas, consolidando a GIMP como una opción potente y viable en el ámbito del procesamiento automatizado de imágenes.

3 Desarrollo

En este capítulo se describe el proceso técnico utilizado para la realización del proyecto. Se detalla el entorno de desarrollo en el que se ha configurado específicamente para trabajar con GIMP 3.0, así como los lenguajes y las herramientas empleadas. También se explica el diseño general de los scripts desarrollados, las funcionalidades implementadas y su ejecución tanto desde la interfaz gráfica como desde la terminal.

Asimismo, se recogen las decisiones tomadas en función de las limitaciones y características del entorno, y el enfoque utilizado para garantizar estabilidad, reutilización y automatización de los procedimientos. Finalmente, este capítulo contiene una revisión de los principales problemas encontrados y las soluciones aplicadas a lo largo del proceso de desarrollo.

3.1 Entorno de desarrollo

El entorno de desarrollo para este proyecto se ha configurado sobre una máquina virtual con el sistema operativo Ubuntu 24. Esta elección se debe a su estabilidad, la amplia compatibilidad con herramientas de código abierto y la facilidad para gestionar entornos aislados de desarrollo. Inicialmente, se planteó sobre la versión 2.10 de GIMP, ya que es una versión que hoy en día sigue siendo ampliamente respaldada por la comunidad, pero el entorno de scripting de esta versión está basado en Python 2.7, lo que imposibilita utilizar bibliotecas modernas de código abierto como pueden ser OpenCV o NumPy, siendo estas dependientes de Python 3. Es por esta razón que se decidió utilizar la versión más reciente y avanzada de la herramienta, GIMP 3.0. Actualmente su instalación solo puede ser dada mediante Flatpak, la cual permite disponer de una versión compatible con Python 3, y que además se garantice un entorno aislado, actualizado y alineado con los desarrollos más actuales de GIMP.

Algunas de las ventajas de la instalación de GIMP mediante Flatpak son la garantía del entorno aislado del sistema base, la evasión de conflictos de dependencias o las actualizaciones automáticas con cada versión oficial. No obstante, también introduce algunas limitaciones como restricciones de acceso a determinadas rutas del sistema de archivos o dificultades para utilizar bibliotecas externas desde Python-Fu [10]. Estas limitaciones han condicionado parte del desarrollo, como se detalla en apartados posteriores.

Además, para la creación y edición de los scripts se ha utilizado el editor de texto Geany, una herramienta ligera y altamente configurable que ha facilitado la escritura de scripts de forma estructurada y eficiente.

Asimismo, durante todo el desarrollo se empleó la herramienta Git como sistema de control de versiones y un repositorio en GitHub para salvaguardar los scripts. La secuencia de commits proporcionó una trazabilidad precisa de la evolución del código, de modo que comparar versiones o revertir cambios resultó sencillo.

En resumen, el entorno de desarrollo contiene herramientas actualizadas, configuraciones personalizadas para facilitar la automatización y un enfoque orientado a la portabilidad y estabilidad del sistema, lo cual garantiza las condiciones necesarias para el desarrollo y prueba de scripts.

3.1.1 Comparativa entre GIMP 2.10 y GIMP 3.0

Durante el desarrollo de este trabajo se valoraron las dos versiones más relevantes de GIMP en el momento actual, GIMP 2.10 y GIMP 3.0. Aunque la versión 2.10 sigue siendo ampliamente utilizada y cuenta con una gran cantidad de documentación y comunidad activa, la elección final del entorno de desarrollo se decantó por la versión 3.0, considerando una serie de aspectos comparativos que se explicarán en este apartado.

El factor más determinante de la elección fue la compatibilidad con las versiones de Python. Gimp 2.10 continúa basando su sistema de scripting en Python 2.7, el cual ha quedado oficialmente obsoleto y sin soporte desde enero de 2020 [11]. Esta limitación es bastante relevante sobre todo cuando se pretende integrar bibliotecas modernas para procesamiento de imágenes como pueden ser OpenCV o NumPy, que requieren de Python 3 para su funcionamiento. Por el contrario, GIMP 3.0 ofrece una compatibilidad completa con esta versión de Python, lo que permite utilizar sin restricciones las bibliotecas actualizadas de análisis de imágenes.

Una de las grandes diferencias entre las dos versiones de este programa son los cambios que han añadido en la arquitectura de scripting, ya que GIMP 3.0 introduce un cambio estructural en la arquitectura de plugins de Python. Mientras que GIMP 2.10 utiliza el módulo gimpfu, ampliamente documentado y relativamente sencillo de utilizar, GIMP 3.0 ha migrado su sistema para estar basado en GObject y gi.repository. Este nuevo modelo ofrece una integración más moderna y robusta, pero requiere de un conocimiento técnico adicional sobre el manejo de GObject's, las estructuras del modelo GTK y el nuevo sistema de procedimientos, lo que hace incrementar la curva de aprendizaje para desarrollar con Python-Fu en esta nueva versión [12].

Otro aspecto relevante de la versión 3.0 es el sistema de instalación y ejecución, debido a que actualmente, esta versión es distribuida principalmente con Flatpak, lo que proporciona un entorno de ejecución completamente aislado, evitando conflictos de dependencias, permitiendo actualizaciones automáticas y asegurando la compatibilidad con las versiones más recientes del programa. Esta estrategia de distribución es especialmente útil para mantener entornos de desarrollo reproducibles y actualizados. Por su parte, Gimp 2.10 sigue ofreciendo la instalación clásica integrada en el sistema operativo, lo que facilita en algunos casos la interacción directa con bibliotecas de terceros instaladas a nivel de sistema, pero con mayor riesgo de incompatibilidad de las dependencias.

En cuanto a la madurez de la documentación y de la comunidad de estas versiones, GIMP 2.10 continúa siendo más estable, con una gran cantidad de documentación oficial, ejemplos de código, tutoriales y foros de ayuda tanto para Script-Fu como para Python-Fu. En cambio, GIMP 3.0, al tratarse de una versión mucho más reciente, todavía presenta una documentación no tan completa y una comunidad reducida en lo que respecta al desarrollo de scripts y plugins. Este hecho ha supuesto un reto adicional durante el desarrollo del proyecto, especialmente en las fases de aprendizaje, de resolución de errores y de búsqueda de referencias actualizadas.

Además de los cambios mencionados, otro de los principales avances técnicos de la versión 3.0 ha sido la integración completa de GEGL (Generic Graphics Library) como núcleo de su motor de procesamiento. Aunque esta biblioteca ya estaba parcialmente integrada en versiones anteriores, en GIMP 3.0 pasa a ser el motor de procesamiento por defecto sobre el que se ejecutan la mayoría de

las operaciones internas, ofreciendo un sistema no destructivo, con mayor precisión, un soporte ampliado para imágenes de alta profundidad de color y una mejor gestión de operaciones en coma flotante [13]. Sin embargo, esta transición al motor GEGL no es directamente visible en el desarrollo de scripts en Scheme, ya que los procedimientos accesibles desde el entorno de scripting siguen escritos bajo la nomenclatura tradicional "gimp-", manteniendo la compatibilidad con los scripts existentes y simplificando el desarrollo. Aunque muchos de los procedimientos continúan estando disponibles y funcionando de forma transparente, internamente muchas de estas operaciones delegan ahora en GEGL para realizar el procesamiento real. De este modo, procedimientos como "gimp-drawable-desaturate" o "gimp-drawable-threshold", que han sido empleados durante el desarrollo de este proyecto, continúan funcionando igual desde el punto de vista del scripting, aunque su ejecución aprovecha las mejoras introducidas por el motor GEGL a nivel interno.

En resumen, GIMP 2.10 sigue siendo la opción más madura para quienes priorizan estabilidad y abundancia de documentación, sin embargo, sus limitaciones antes mencionadas lo convierten en una solución con fecha de caducidad para proyectos que requieran bibliotecas modernas o flujos de trabajo de alta precisión. Gimp 3.0, por otro lado, pese a contar aún con menor cobertura documental y exigir un esfuerzo adicional de adaptación a su nueva arquitectura basada en GObject, Python 3 y un motor GEGL plenamente integrado, ofrece un entorno preparado para el futuro, compatible con tecnologías actuales, con un motor gráfico no destructivo y con un sistema de distribución aislado que asegura actualizaciones continuas. Por todos estos motivos, la elección de GIMP 3.0 es la más adecuada para un proyecto orientado a la extensión y la automatización, donde la escalabilidad, la compatibilidad con bibliotecas externas y la proyección a largo plazo son criterios decisivos.

3.2 Lenguajes

Este proyecto se ha desarrollado utilizando principalmente dos lenguajes de scripting compatibles con GIMP, Scheme y Python. Ambos lenguajes han sido considerados desde la fase inicial por las ventajas que ofrece cada uno, pero en la práctica ha sido necesario priorizar un lenguaje sobre otro debido a las limitaciones encontradas en el entorno.

3.2.1 Scheme (Script-Fu)

Este lenguaje, utilizado a través del sistema Script-Fu, constituye la tecnología de scripting principal empleada en este trabajo. Scheme es un lenguaje funcional, compacto y eficiente, que GIMP ha adoptado a lo largo de su historia como su sistema de scripting nativo. Además permite definir procedimientos personalizados integrados directamente en el menú de GIMP, automatizando operaciones sobre imágenes desde la interfaz gráfica o desde la línea de comandos en la terminal.

Una de las principales razones para priorizar el desarrollo de las funcionalidades en Scheme ha sido la plena integración de este lenguaje con GIMP 3.0 bajo Flatpak, ya que no necesita de configuraciones adicionales ni

dependencias externas. Su funcionamiento ha demostrado ser estable y fiable, lo que ha permitido desarrollar los scripts de tal manera que sean reutilizables. Asimismo, Script-Fu facilita el registro de procedimientos, la definición de parámetros interactivos, el control de capas o canales de una imagen, todo ello sin abandonar el entorno propio de GIMP.

3.2.2 Python (Python-Fu)

El lenguaje de Python también ha sido considerado desde las fases iniciales del proyecto por razones como su popularidad, legibilidad o potencia, así como por la posibilidad que ofrece de integrar bibliotecas externas como OpenCV o NumPy. Sin embargo, durante el desarrollo de las funcionalidades se identificaron varias limitaciones técnicas en el uso de Python-Fu en GIMP 3.0 bajo Flatpak. Una complicación importante ha sido la limitada comunidad que se encuentra en torno a esta nueva versión de GIMP, ya que todavía es una versión muy reciente y la cantidad de documentación, foros, tutoriales y ejemplos es aún escasa en comparación con versiones anteriores, como puede ser GIMP 2.10. Esta falta de soporte es un impedimento a la hora de resolver errores, buscar buenas prácticas o encontrar soluciones a problemas comunes, por lo cual aumenta significativamente la curva de aprendizaje.

Además, merece la pena detenerse en OpenCV (Open Source Computer Vision Library), una biblioteca de visión artificial y procesamiento de imágenes de código abierto que cuenta con enlaces nativos para C++ y, mediante bindings, para este lenguaje. OpenCV ofrece un amplio repertorio de algoritmos optimizados como filtrados, transformaciones geométricas, seguimiento de objetos o deep learning, lo que la convierte en un complemento idóneo para potenciar scripts. La posibilidad de recurrir a OpenCV amplía considerablemente el alcance de los procesados que se pueden automatizar, justificando su consideración durante las fases de diseño del proyecto.

En resumen, aunque Python sigue siendo una herramienta potente y flexible, la utilización de este lenguaje en el contexto específico de GIMP 3.0 instalado vía Flatpak ha implicado varios retos técnicos y formativos que han condicionado el desarrollo del proyecto.

3.2.3 Comparativa entre Scheme y Python

La elección entre estos dos lenguajes para el desarrollo de scripts en GIMP responde, ante todo, a la distinta filosofía de ambos lenguajes. Scheme, heredero directo de Lisp, se asienta sobre una sintaxis prefija extraordinariamente regular y un paradigma funcional, este minimalismo obliga a pensar cada procedimiento como una composición de transformaciones puras, algo muy acorde con la lógica declarativa de las operaciones gráficas de GIMP. Python, por su parte, ofrece una sintaxis cercana al pseudocódigo que casi cualquier programador asimila de inmediato y un estilo multiparadigma que facilita estructurar proyectos complejos sin necesidad de adoptar de entrada patrones puramente funcionales.

Estas diferencias conceptuales también están reflejadas en la integración con la API de GIMP. Script-Fu, el intérprete nativo de Scheme, expone de forma directa la mayoría de los procedimientos internos mediante la tradicional

nomenclatura “gimp-”, genera automáticamente los cuadros de diálogo de parámetros y registra un script con un par de líneas. Python-Fu, en GIMP 3.0, ha abandonado el módulo gimpfu para apoyarse en GObject y gi.repository, necesita definir una clase que hereda de Gimp.PlugIn para escribir un plugin, maneja objetos GTK para los cuadros de diálogo y precisa de comprender el sistema de señales de GObject [14], todo esto amplía enormemente el potencial, por ejemplo, para crear interfaces gráficas ricas, pero incrementa la curva de aprendizaje y el volumen de código imprescindible para un prototipo sencillo, como se mostrará en capítulos posteriores de este proyecto. Aun así, la fortaleza del ecosistema Python resulta decisiva cuando se necesitan algoritmos avanzados de visión artificial o análisis numérico. Bibliotecas como OpenCV o NumPy pueden importarse sin restricciones en GIMP 3.0 gracias a la compatibilidad antes mencionada con Python 3. Scheme carece de un repositorio de paquetes comparable, ya que depende casi por completo de las primitivas que ofrece la propia API de GIMP, lo que simplifica las dependencias y garantiza portabilidad, pero limita las posibilidades cuando el proyecto exige técnicas de procesamiento de imagen o aprendizaje automático más sofisticadas.

En términos de rendimiento, los scripts en Scheme se ejecutan dentro del mismo proceso de GIMP con una sobrecarga mínima de arranque, gracias a su naturaleza funcional y la ausencia de librerías externas se favorece el consumo mínimo de memoria. Por otro lado, Python introduce un intérprete más pesado y, si se cargan módulos de cálculo intensivo, el consumo puede crecer, aunque esta desventaja se compensa cuando se aprovechan implementaciones en C/C++ altamente optimizadas, como el núcleo de OpenCV, que superan con creces la velocidad de las internas de GIMP para ciertas tareas [15].

Finalmente, la sostenibilidad a largo plazo inclina la balanza de forma matizada. Scheme cuenta con una API estable y una retrocompatibilidad casi absoluta, es previsible que un script escrito hoy funcionará en futuras versiones mientras siga existiendo Script-Fu. Python, pese a su popularidad, está aún consolidando su integración en GIMP y carece de tanta documentación y ejemplos como Script-Fu, lo que obliga a asumir cierta inestabilidad a corto plazo. Aun así, con su sintaxis moderna y el respaldo masivo de la comunidad científica lo convierten en la opción con mayor proyección para extensiones de gran envergadura.

Por todo ello, en este proyecto se optó por Scheme como lenguaje principal, ya que ofrece estabilidad inmediata y apenas requiere dependencias externas dentro del contenedor Flatpak de GIMP 3.0. No obstante, se reconoce el potencial de Python para futuras ampliaciones que demanden algoritmos más complejos, interfaces personalizadas o interacción con ecosistemas de ciencia de datos. A medida que la implementación de Python-Fu madure y la comunidad aporte ejemplos sólidos, ambos lenguajes convivirán con naturalidad, reservando el uso de Scheme para tareas rápidas y ligeras, mientras que Python se utilizará para desarrollos que precisen la potencia de su vasto ecosistema.

3.3 Diseño de los scripts

El desarrollo de los scripts en este proyecto aparte de haber estado orientado a resolver tareas específicas de procesamiento de imágenes, también están

dedicados a establecer un diseño que sea coherente, modular y reutilizable, que facilite el que sean integrados en distintos flujos de trabajo. Dado que principalmente uno de los objetivos del trabajo es automatizar operaciones gráficas tanto en contextos interactivos como no interactivos, se ha prestado especial atención a que los scripts fueran compatibles con ambos modos de ejecución, siempre en la medida de lo posible dependiendo de la funcionalidad, ya sea utilizándolos desde la interfaz gráfica que ofrece GIMP como desde la terminal de Linux.

Para todo ello, ha sido fundamental seguir una serie de principios de diseños enfocados en la modularidad, la reutilización y la parametrización, consiguiendo así que cada script cumpla una función bien definida y si fuese necesario, que puedan ser adaptados con facilidad a nuevas tareas o entornos. Se ha procurado que los parámetros necesarios para el funcionamiento de cada script puedan ser definidos por el usuario en el momento de su respectiva llamada, como puede ser las rutas de entrada o salida, gracias a esto es más viable una ejecución automatizada y flexible sin requerir interacción posterior.

Los scripts han sido implementados en su gran mayoría utilizando Script-Fu, el sistema de scripting de GIMP basado en el lenguaje Scheme. Estos scripts se pueden estructurar en tres partes principales: la definición del script, su lógica funcional y el registro del script. En la definición del script se especifica el nombre del procedimiento, los parámetros de entrada requeridos (como pueden ser rutas de archivos, valores numéricos o cadenas de texto) y los tipos de datos esperados. La lógica funcional es la parte del script en donde se describen las operaciones que el script realiza sobre la imagen o la capa activa, algunos ejemplos de ellas podrían ser duplicar capas, convertir a escala de grises, aplicar umbrales o guardar el resultado en una ubicación determinada. El registro se realiza mediante la función “`script-fu-register`”, el cual tiene una estructura determinada que se mostrará a continuación. Al registrar el procedimiento, el script queda integrado en el entorno de GIMP, visible bajo una categoría específica del menú si así se desea, esto permite una fácil ejecución de la funcionalidad desde la interfaz gráfica del programa.

Un ejemplo básico y detallado de esta estructura podría ser la siguiente, donde cada bloque representa un componente esencial de cualquier script en Script-Fu:

```
;Definición del procedimiento
(define (nombre-del-procedimiento parámetro-entrada-1 parámetro-entrada-2)
;Lógica funcional
(gimp-message “Aquí van las operaciones del script.”)
;Registro del procedimiento
(script-fu-register
       “nombre-del-procedimiento”
       “Nombre visible en interfaz”
       “Descripción del script.”
       “Autor”
       “Licencia o institución”
       “Fecha (año)”
       “Parámetros visibles en el cuadro de diálogo”
```

```
)
(script-fu-menu-register “nombre-del-procedimiento” “ruta-en-interfaz”)
```

Además de esta estructura formal, los scripts han sido diseñados para evitar cualquier tipo de interacción durante su ejecución, esto es fundamental para garantizar su correcto funcionamiento desde la terminal. Una vez definidos los parámetros al lanzar el script desde la terminal, el procesamiento se lleva a cabo de forma completamente automatizada, lo que los hace aptos para escenarios de tratamiento masivo de imágenes.

También se ha cuidado especialmente la organización y legibilidad del código fuente, incorporando comentarios explicativos y manteniendo una separación clara entre funciones. Esta práctica no es solo para facilitar su comprensión y mantenimiento, sino que además permite extender fácilmente los scripts a nuevos casos de uso o adaptarlos a proyectos futuros.

En conjunto, el diseño adoptado ha permitido crear scripts funcionales, robustos y flexibles, plenamente alineados con los objetivos de automatización planteados en el proyecto y con los principios de desarrollo sostenible en entornos de procesamiento técnico o científico.

3.4 Funcionalidades implementadas

A lo largo del desarrollo del proyecto se han implementado diversas funcionalidades mediante scripts personalizados con el objetivo de automatizar tareas comunes de procesamiento de imágenes. Estas funcionalidades han sido seleccionadas por su aplicabilidad en contextos científicos, técnicos o educativos, y responden a necesidades frecuentes como la conversión de formatos, la preparación de imágenes para análisis o la generación de elementos visuales sobre las imágenes.

Todas estas funcionalidades han sido implementadas mayoritariamente en Scheme, utilizando Script-Fu, y han sido diseñadas para funcionar tanto de forma interactiva desde la interfaz que ofrece GIMP como en modo no interactivo desde la terminal de Linux. Esto ha permitido validar su integración con el entorno gráfico del programa y, al mismo tiempo, su utilidad en flujos de trabajo automatizados.

Las funcionalidades implementadas son las siguientes:

- **Conversión de formato de imagen**: este script permite convertir imágenes entre distintos formatos, como “.jpg”, “.png” o “.tiff” entre otras. Este script trabaja de forma automatizada por lo que lo hace ideal para normalizar grandes lotes de archivos.

- **Binarización**: esta funcionalidad consiste en transformar una imagen a escala de grises seguido de una binarización en función de un umbral escogido por el usuario, lo que lo hace útil para tareas de segmentación o análisis visual.

- **Adición de texto**: este procedimiento permite insertar texto sobre la imagen activa con varias opciones de inserción, lo que lo hace práctico para anotar, numerar o clasificar imágenes dentro de un conjunto de trabajo.

- **Expansión de selección**: implementación de una funcionalidad que se basa en permitir al usuario ampliar una selección activa en la imagen y aplicar sobre ella un tratamiento visual para mejorar la calidad gráfica de la misma, permitiendo así flexibilidad para mejorar la precisión y la calidad de las imágenes.

- **Resaltado de regiones**: este script analiza una zona escogida, a partir del color de un píxel seleccionado, añadiendo a una nueva máscara todos los pixeles que comparten la misma tonalidad. Este script es práctico para la identificación visual de objetos o patrones homogéneos dentro de la imagen.

Todas estas funcionalidades demuestran la capacidad de GIMP para actuar como una herramienta flexible y automatizable, capaz de adaptarse a flujos de trabajo personalizados. Además, resuelven tareas específicas, y también sirven como base para una posible extensión o integración futura en procesos aún más complejos.

3.4.1 Conversión de formato de imagen

Una de las principales funcionalidades implementadas en este proyecto ha sido la conversión automática de imágenes entre distintos formatos. Este tipo de tarea es habitual en flujos de trabajo donde es necesario homogeneizar extensiones de archivos o preparar imágenes para su uso en otras herramientas que solo aceptan determinados formatos, ya que permite estandarizar las extensiones de imágenes, consiguiendo entre otras cosas, mejorar su portabilidad o preparar los archivos para su uso en flujos de análisis o de documentación.

Para esta funcionalidad se han desarrollado dos scripts, uno en Scheme utilizando Script-Fu, integrado en GIMP 3.0, y otro en Python haciendo uso de la biblioteca OpenCV, diseñado para ejecutarse desde la terminal de manera autónoma. A continuación, se explica más detalladamente cada uno de los scripts.

3.4.1.1 ChangeFormat.scm (Scheme)

```
; Lista de formatos aceptados
(define valid-formats '("jpg" "jpeg" "png" "bmp" "tiff" "webp" "xcf"))


(define (script-fu-convert-image-format path dir name exitformat)
  (let* (
         ; Obtener el formato real a partir del indice seleccionado
         (format (list-ref valid-formats exitformat))
```

```
        ; Verificar que el formato está en la lista
        (format-valid? (member format valid-formats))

        ; Cargar imagen
        (image (car (gimp-file-load RUN-NONINTERACTIVE path path)))

        ; Extraer nombre original
        (path-parts (list->vector (strbreakup path "/")))
        (filename (vector-ref path-parts (- (vector-length path-parts) 1)))
        (name-parts (list->vector (strbreakup filename ".")))
        (name-no-ext (vector-ref name-parts 0))

        ; Usar name si no está vacio
        (final-name (if (string=? name "")
                        name-no-ext
                        name))

        (output-filename (string-append dir "/" final-name "." format))
       )

    ; Verificar formato
    (if (not format-valid?)
        (begin
          (gimp-message (string-append "Error: formato no válido: " format))
          (error "Formato de salida no admitido.")))

    ; Guardar imagen
    (gimp-file-save RUN-NONINTERACTIVE image output-filename output-filename)

    ; Cerrar imagen
    (gimp-image-delete image)
    (gimp-message (string-append "Imagen guardada como: " output-filename))
  )
)

(script-fu-register
 "script-fu-convert-image-format"
 "Convert Image Format"
```

```
 "Convierte una imagen a otro formato."
 "Nicolas Bravo"
 "Nicolas Bravo"
 "2025"
 ""
 SF-FILENAME "Imagen de entrada" ""
 SF-DIRNAME  "Carpeta de salida" ""
 SF-STRING   "Nombre de salida" ""
 SF-OPTION   "Formato de salida" '("jpg" "jpeg" "png" "bmp" "tiff" "webp"
"xcf")
)
(script-fu-menu-register "script-fu-convert-image-format"
"<Image>/Filters/Script-Fu")
```

Este script ha sido diseñado para que pueda ejecutarse tanto desde la interfaz de GIMP, como desde la terminal de manera no interactiva. Está desarrollado en el lenguaje de Scheme, mediante el sistema Script-Fu.

```
(define valid-formats '("jpg" "jpeg" "png" "bmp" "tiff" "webp" "xcf"))
```

El script comienza creando una lista con los formatos admitidos, entre ellos se encuentran los más comúnmente usados en ámbitos científicos, para uso web o documental.

```
(define (script-fu-convert-image-format path dir name exitformat)
```

Se define la función con el nombre “`script-fu-convert-image-format`”, y recibe cuatro parámetros: la ruta del archivo de entrada, “`path`”, el directorio de salida, “`dir`”, el nombre de salida de la imagen, “`name`”, y el índice numérico que representa el formato escogido para la nueva imagen, “`exitformat`”.

```
(format (list-ref valid-formats exitformat))
```

Se traduce el índice de la lista de formatos recibido por su correspondiente cadena de texto, utilizando “`list-ref`”.

```
(format-valid? (member format valid-formats))
```

Se verifica que el formato escogido está entre los válidos.

```
(image (car (gimp-file-load RUN-NONINTERACTIVE path path)))
```

Se carga la imagen original desde el disco utilizando la ruta dada, además el parámetro “`RUN-NONINTERACTIVE`” permite ejecutar el script desde la terminal sin necesidad de abrir el programa. La función devuelve el identificador interno de la imagen dentro de GIMP.

```
(path-parts (list->vector (strbreakup path "/")))
```

```
(filename (vector-ref path-parts (- (vector-length path-parts) 1)))
(name-parts (list->vector (strbreakup filename ".")))
(name-no-ext (vector-ref name-parts 0))
```

El nombre original se extrae, por si fuese necesario reutilizarlo si el usuario no introdujo un nuevo nombre para la imagen. Para ello se realiza un tratamiento completo de la ruta del fichero, separándola por el carácter “/”, obteniendo el nombre del archivo y finalmente separando el nombre por “.” para eliminar su extensión.

```
(final-name (if (string=? name "")
                          name-no-ext
                          name))
```

Se determina el nombre final de la imagen. Primero comprueba si el usuario introdujo un nuevo nombre, conservando el nombre en ese caso y utilizando el nombre proporcionado en el caso contrario.

```
(output-filename (string-append dir "/" final-name "." format))
```

Se construye la ruta de salida concatenando el directorio destino dado, el nombre final y la nueva extensión.

```
(if (not format-valid?)
        (begin
          (gimp-message (string-append "Error: formato no válido: " format))
          (error "Formato de salida no admitido.")))
```

Antes de guardar la imagen, se comprueba nuevamente que el formato es válido. En el caso de que no lo sea, el usuario será informado con un mensaje en la consola de GIMP y se abortará la ejecución.

```
(gimp-file-save RUN-NONINTERACTIVE image output-filename output-filename)
```

La imagen se guarda en la ruta de salida anteriormente construida.

```
(gimp-image-delete image)
```

Se libera de la memoria la imagen cargada.

```
(gimp-message (string-append "Imagen guardada como: " output-filename))
```

Por último, se informa al usuario del resultado exitoso de la operación.

3.4.1.2 change_format.py (Python y OpenCV)

```
import cv2
import argparse
```

```
import os


def main():
     parser = argparse.ArgumentParser(description="Cambiar el formato de una imagen usando OpenCV.")
     parser.add_argument('--image', type=str, required=True, help='Ruta de la imagen original.')
    parser.add_argument('--format', type=str, required=True, help='Nuevo formato (png, jpg, bmp, etc.).')
    parser.add_argument('--exit_path', type=str, default='', help='Ruta opcional de la imagen de salida.')


    args = parser.parse_args()


    # Cargar la imagen
    img = cv2.imread(args.image)
    if img is None:
        print(f"No se pudo cargar la imagen: {args.image}")
        return


    # Determinar ruta de salida
    if args.exit_path:
        root, ext = os.path.splitext(args.exit_path)
        if not ext:
            path = f"{args.exit_path}.{args.format}"
        else:
            path = args.exit_path
    else:
        path = f"{args.image.rsplit('.', 1)[0]}.{args.format}"


    # Guardar imagen en nuevo formato
    result = cv2.imwrite(path, img)
    if result:
        print(f"Imagen guardada en: {path}")
    else:
        print(f"No se pudo guardar la imagen en el formato {args.format}")


if __name__ == "__main__":
    main()
```

Además del script desarrollado en Scheme integrado en GIMP, se ha implementado una solución alternativa empleando Python y la biblioteca de

procesamiento de imágenes OpenCV. Esta implementación resulta útil para entornos en los que no se requiera el uso de GIMP, ofreciendo una herramienta ligera, flexible y fácil de integrar en flujos de trabajo automatizados.

```
import cv2
import argparse
import os
```

Primero se importan las librerías necesarias, “cv2” es el módulo principal de la biblioteca OpenCV, “argparse” permite gestionar de forma sencilla los argumentos que el usuario proporciona al ejecutar el script desde la terminal y “os” otorga la capacidad de realizar operaciones de manipulación de rutas de archivos de forma y multiplataforma.

```
def main():
```

El cuerpo de la función se organiza dentro de la función “main”.

```
parser = argparse.ArgumentParser(description="Cambiar el formato de una imagen usando OpenCV.")
```

Se inicializa un objeto “ArgumentParser” para definir los parámetros que el script aceptará al ser ejecutado. Además, se añade una pequeña descripción que el usuario puede ver si ejecuta el comando “–help”.

```
parser.add_argument('--image', type=str, required=True, help='Ruta de la imagen original.')
parser.add_argument('--format', type=str, required=True, help='Nuevo formato (png, jpg, bmp, etc.).')
parser.add_argument('--exit_path', type=str, default='', help='Ruta opcional de la imagen de salida.')
```

Se declaran los tres argumentos de entrada, siendo obligatorios “--image”, la ruta de la imagen origen, y “--format”, el nuevo formato deseado. También se declara el argumento opcional “--exit_path” con la ruta de salida que si no es proporcionada por el usuario se genera automáticamente por el script.

```
args = parser.parse_args()
```

Con esta instrucción se almacenan los valores dados por el usuario para los argumentos en el objeto “args”.

```
img = cv2.imread(args.image)
if img is None:
     print(f"No se pudo cargar la imagen: {args.image}")
     return
```

La imagen es cargada, y si la ruta no es válida o la imagen no se puede abrir, se informa al usuario con un mensaje y se finaliza la ejecución del script.

```
if args.exit_path:
    root, ext = os.path.splitext(args.exit_path)
    if not ext:
        path = f"{args.exit_path}.{args.format}"
    else:
        path = args.exit_path
else:
    path = f"{args.image.rsplit('.', 1)[0]}.{args.format}"
```

Si el usuario ha proporcionado la ruta de salida, el script separa el nombre y la extensión, a continuación, si falta la extensión, la añade automáticamente, y si la extensión existe, usa la ruta tal cual. Por otra parte, si el usuario no indica la ruta de salida, se toma el nombre y ruta del archivo original, se le quita su extensión y se le añade la nueva, garantizando así que siempre se obtenga una ruta válida antes de guardar la imagen con el formato deseado.

```
result = cv2.imwrite(path, img)
if result:
    print(f"Imagen guardada en: {path}")
else:
    print(f"No se pudo guardar la imagen en el formato {args.format}")
```

Se guarda la imagen con el nuevo formato y si no hay errores, se le informa al usuario con un mensaje de confirmación en donde se indica la ruta establecida, en caso contrario también se le avisa al usuario de que no fue posible guardar el archivo.

```
if __name__ == "__main__":
    main()
```

Por último, se añade este bloque para asegurar que la función “`main()`” se ejecute solo cuando el script es ejecutado directamente desde la terminal. Esto permite su reutilización como módulo en otros programas Python si fuese necesario.

3.4.2 Binarización

La binarización es una operación interesante en el procesamiento de imágenes científicas, debido a que permite segmentar una imagen en dos niveles de intensidad (blanco y negro) a partir de un umbral escogido. Esto es especialmente útil en tareas como la detección de formas o estructuras en imágenes, o la preparación de datos para su posterior análisis.

Para esta funcionalidad también se han desarrollado dos scripts, uno en Scheme integrado en GIMP y otro en Python utilizando la biblioteca OpenCV, pudiendo ser utilizados ambos desde la terminal.

3.4.2.1 Binarize.scm (Scheme)

```
; Funcion auxiliar que verifica si una cadena str termina en el sufijo suffix
(define (string-suffix? suffix str)
  (let ((slen (string-length suffix))
        (len (string-length str)))
    (and (<= slen len)
         (string=? (substring str (- len slen) len) suffix))))

; Funcion auxiliar que obtiene el nombre del archivo sin la extension a partir de una ruta completa
(define (get-filename-without-extension path)
  (let* ((parts (strbreakup path "/"))
         (filename (list-ref parts (- (length parts) 1)))
         (name-parts (strbreakup filename "."))
         (base (if (> (length name-parts) 1)
                   (list-ref name-parts 0)
                   filename)))
    base))

(define (script-fu-binarize path dir umbral name)

  ; Asegurarse de que name es una cadena valida
  (set! name (if (string? name) name ""))

  (let* (
         ; Carga la imagen desde disco
         (image (car (gimp-file-load RUN-NONINTERACTIVE path path)))

         ; Obtiene las capas de la imagen
         (layers (gimp-image-get-layers image))

         ; Selecciona la primera capa o aplana la imagen si no hay ninguna
         (layer (if (and (vector? layers) (> (vector-length layers) 0))
                    (aref layers 0)
                    (begin
```

```
                    (car (gimp-image-flatten image)))))

      ; Crea una copia de la capa para aplicar el filtro
      (copy (car (gimp-layer-copy layer TRUE)))

      ; Extrae el nombre base del archivo de entrada
      (basename (get-filename-without-extension path))

      ; Elimina una barra final en el directorio si la hay
      (output-dir (if (string-suffix? "/" dir)
                      (substring dir 0 (- (string-length dir) 1))
                      dir))

      ; Construye el nombre del archivo de salida
      (filename (if (string=? name "")
                    (string-append basename "_binarized.png")
                    (if (string-suffix? ".png" name)
                        name
                        (string-append name ".png"))))

      ; Ruta completa de salida
      (output-name (string-append output-dir "/" filename))
     )

  ; Inserta la capa copiada en la imagen
  (gimp-image-insert-layer image copy 0 -1)

  ; Convierte a escala de grises
  (gimp-drawable-desaturate copy DESATURATE-LIGHTNESS)

  ; Aplica umbral de binarizacion segun el valor dado
  (gimp-drawable-threshold copy HISTOGRAM-VALUE (/ umbral 255.0) 1.0)

  ; Guarda la imagen resultante
  (gimp-file-save RUN-NONINTERACTIVE image output-name #f)

  ; Elimina la imagen de la memoria
  (gimp-image-delete image)
)
```

```
)


(script-fu-register
 "script-fu-binarize"
 "Binarize"
 "Carga una imagen, aplica binarizacion y guarda el resultado."
 "Nicolas Bravo"
 "Nicolas Bravo"
 "2025"
 ""
 SF-FILENAME "Archivo de imagen" ""
 SF-DIRNAME  "Directorio de salida" ""
 SF-ADJUSTMENT "Umbral" '(128 0 255 1 10 0 0)
 SF-STRING "Nombre de salida" ""
)


(script-fu-menu-register "script-fu-binarize" "<Image>/Filters/Script-Fu")
```

De la misma manera que el anterior script en Scheme, esta funcionalidad también está diseñada para poder ser utilizada tanto desde la interfaz gráfica que ofrece GIMP, como desde la terminal de manera no interactiva.

Primero se crean dos funciones auxiliares.

```
(define (string-suffix? suffix str)
  (let ((slen (string-length suffix))
        (len (string-length str)))
    (and (<= slen len)
         (string=? (substring str (- len slen) len) suffix))))
```

Esta función auxiliar comprueba si una cadena termina con un sufijo dado.

```
(define (get-filename-without-extension path)
  (let* ((parts (strbreakup path "/"))
         (filename (list-ref parts (- (length parts) 1)))
         (name-parts (strbreakup filename "."))
         (base (if (> (length name-parts) 1)
                   (list-ref name-parts 0)
                   filename)))
    base))
```

Con esta se extrae el nombre original a partir de una ruta proporcionada, primero se separa la ruta por el carácter “/” y después se obtiene el nombre separándolo de su extensión con el carácter “.”.

```
(define (script-fu-binarize path dir umbral name)
```

El procedimiento se define con el nombre “`script-fu-binarize`” y contiene cuatro parámetros: la ruta de la imagen a binarizar, “`path`”, el directorio de salida, “`dir`”, el valor del umbral que se aplicará siendo desde 0 hasta 255, “`umbral`”, y el nombre final que tendrá la imagen, “`name`”.

```
(set! name (if (string? name) name ""))
```

Con esta línea de código se comprueba que el nombre otorgado sea una cadena de texto.

```
(image (car (gimp-file-load RUN-NONINTERACTIVE path path)))
```

La imagen se carga en modo no interactivo para permitir su ejecución desde terminal.

```
(layers (gimp-image-get-layers image))
(layer (if (and (vector? layers) (> (vector-length layers) 0))
                    (aref layers 0)
                    (begin
                      (car (gimp-image-flatten image)))))
```

A continuación se obtiene la capa sobre la que se trabajará, primero se comprueba si no tiene capas y si es así aplana la imagen para hallar una única capa válida, en caso contrario se selecciona la primera.

```
(copy (car (gimp-layer-copy layer TRUE)))
```

Se crea una copia de la capa para que los procesos que se apliquen no afecten a la imagen original.

```
(basename (get-filename-without-extension path))
```

Se obtiene el nombre de la imagen original, sin extensión, gracias a la función auxiliar creada anteriormente.

```
(output-dir (if (string-suffix? "/" dir)
                                (substring dir 0 (- (string-length dir) 1))
                                dir))
```

Si la ruta de salida acaba en “/”, se elimina este último carácter para evitar errores que pudiesen darse al crear la nueva ruta de salida por una posible duplicidad de este carácter.

```
(filename (if (string=? name "")
              (string-append basename "_binarized.png")
              (if (string-suffix? ".png" name)
```

```
                    name
                    (string-append name ".png"))))
```

Este fragmento corresponde a la construcción del nombre de la imagen resultante. En el caso de que el usuario no proporcione nombre, o este no sea una cadena de texto, se usa el nombre original al que se le añade "_binarized.png". Por el contrario, si el usuario sí que proporcionó un nombre válido, se utiliza directamente, y si este ya contiene la extensión ".png" se mantiene sin duplicarla.

```
(output-name (string-append output-dir "/" filename))
```

La ruta completa de salida se construye añadiendo el nuevo nombre.

```
(gimp-image-insert-layer image copy 0 -1)
```

Se inserta la capa copiada dentro de la imagen para trabajar sobre ella.

```
(gimp-drawable-desaturate copy DESATURATE-LIGHTNESS)
```

Convierte la imagen a escala de grises ya que es necesario antes de aplicar la binarización porque esta solo funciona sobre imágenes con esta tonalidad.

```
(gimp-drawable-threshold copy HISTOGRAM-VALUE (/ umbral 255.0) 1.0)
```

Se aplica la binarización, para ello se divide el umbral proporcionado por 255 para transformarlo en el rango 0.0 – 1.0, que es el requerido por GIMP, también se añade el comando "HISTOGRAM-VALUE" para indicar que el umbral trabaja sobre los valores de luminosidad. Tras la binarización los píxeles quedan convertidos únicamente en blanco o negro según superen el umbral o no.

```
(gimp-file-save RUN-NONINTERACTIVE image output-name #f)
```

Guarda la imagen en la ruta de salida.

```
(gimp-image-delete image)
```

Borra la imagen de la memoria para liberar recursos.

3.4.2.2 binarize_py.py (Python y OpenCV)

```
import cv2
import argparse
import os


def string_suffix(suffix, string):
    return string.endswith(suffix)
```

```
def get_filename_without_extension(path):
    return os.path.splitext(os.path.basename(path))[0]


def main():
    parser = argparse.ArgumentParser(description="Binarizar una imagen usando OpenCV.")
    parser.add_argument('--image', type=str, required=True, help='Ruta de la imagen original.')
    parser.add_argument('--directory', type=str, required=True, help='Directorio de salida.')
    parser.add_argument('--threshold', type=int, default=128, help='Umbral para la binarización (0-255).')
    parser.add_argument('--name', type=str, default='', help='Nombre opcional para la imagen de salida.')

    args = parser.parse_args()

    # Cargar la imagen en escala de grises
    img = cv2.imread(args.image, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"No se pudo cargar la imagen: {args.image}")
        return

    # Aplicar binarización
    _, binarized = cv2.threshold(img, args.threshold, 255, cv2.THRESH_BINARY)

    # Preparar nombre del archivo de salida
    og_name = get_filename_without_extension(args.image)
    exit_name = args.name if args.name else f"{og_name}_binarized.png"
    if not string_suffix(".png", exit_name):
        exit_name += ".png"

    # Asegurar que el directorio no termina con barra
    exit_dir = args.directory.rstrip('/')

    # Ruta completa de salida
    exit_path = os.path.join(exit_dir, exit_name)

    # Guardar imagen binarizada
    cv2.imwrite(exit_path, binarized)
    print(f"Imagen binarizada guardada en: {exit_path}")
```

```
if __name__ == "__main__":
    main()
```

Este script ha sido desarrollado en Python utilizando la librería OpenCV, está orientado a realizar la binarización de imágenes de forma automática y no interactiva, pero sin depender de GIMP. Su ejecución se realiza desde la terminal, lo que permite integrarlo fácilmente en flujos de trabajo automatizados. A continuación se describe de manera detallada su funcionamiento interno.

```
import cv2
import argparse
import os
```

De la misma manera que en el anterior script de Python, este también importa la librería de OpenCV , la librería que permite la gestión de parámetros desde la línea de comandos y la librería que otorga la capacidad de realizar operaciones de manipulación de rutas de archivos de forma y multiplataforma.

```
def string_suffix(suffix, string):
    return string.endswith(suffix)


def get_filename_without_extension(path):
    return os.path.splitext(os.path.basename(path))[0]
```

Al igual que el script en Scheme, este también utiliza las mismas dos funciones auxiliares para comprobar si una cadena termina con un sufijo y para obtener el nombre de la imagen sin ruta ni extensión.

```
def main():
    parser = argparse.ArgumentParser(description="Binarizar una imagen usando OpenCV.")
```

También el cuerpo de este script está organizado dentro de la función “`main`” y tiene inicializado un objeto “`ArgumentParser`” para definir los parámetros que el script aceptará al ser ejecutado.

```
parser.add_argument('--image', type=str, required=True, help='Ruta de la imagen original.')
parser.add_argument('--directory', type=str, required=True, help='Directorio de salida.')
parser.add_argument('--threshold', type=int, default=128, help='Umbral para la binarización (0-255).')
parser.add_argument('--name', type=str, default='', help='Nombre opcional para la imagen de salida.')
args = parser.parse_args()
```

Los parámetros para este procedimiento son "`--image`" que es obligatorio y contiene la ruta de la imagen original, "`--directory`" que también es obligatorio e incluye el directorio de salida, "`--threshold`" que representa el umbral que se utilizará para la binarización y por defecto toma el valor 128 si no se especifica, y "`--name`" que almacena el nuevo nombre para la imagen, siendo generado automáticamente si no se especifica. Por último se procesan los argumentos recibidos y se guardan en "`args`", desde donde serán accesibles durante el resto de la función.

```
img = cv2.imread(args.image, cv2.IMREAD_GRAYSCALE)
if img is None:
    print(f"No se pudo cargar la imagen: {args.image}")
    return
```

Se carga la imagen directamente en escala de grises con el parámetro "`cv2.IMREAD_GRAYSCALE`", mostrando un error y terminando la ejecución en el caso de que hubiese algún error en la carga ya sea por una ruta mal escrita o por un archivo no existente.

```
_, binarized = cv2.threshold(img, args.threshold, 255, cv2.THRESH_BINARY)
```

Se aplica la binarización a la imagen con el umbral proporcionado por el usuario, el parámetro "`cv2.THRES_BINARY`" sirve para especificar que es una binarización pura, transformando los pixeles en blanco o negro dependiendo de del umbral.

```
og_name = get_filename_without_extension(args.image)
exit_name = args.name if args.name else f"{og_name}_binarized.png"
if not string_suffix(".png", exit_name):
    exit_name += ".png"
```

Se obtiene el nombre original de la imagen utilizando la función auxiliar antes declarada y posteriormente se define el nombre de salida utilizando el nombre dado por el usuario o en el caso de que el usuario no especifique un nombre se reutilice el nombre original añadiendo "_binarized.png". Además, si el nombre no termina con ".png" se agrega automáticamente para asegurar la coherencia con el formato de salida.

```
exit_dir = args.directory.rstrip('/')
exit_path = os.path.join(exit_dir, exit_name)
```

Para la construcción de la ruta de salida se asegura que la ruta proporcionada no acaba en el carácter "/" para evitar duplicidad con este símbolo. Después se une la ruta con el nombre final.

```
cv2.imwrite(exit_path, binarized)
print(f"Imagen binarizada guardada en: {exit_path}")
```

La imagen binarizada se guarda en la ruta antes construida y se le informa al usuario mediante un mensaje de confirmación en el cual se especifica la ruta exacta en donde ha sido almacenada.

```
if __name__ == "__main__":
    main()
```

Por último, se garantiza que la función "`main`" solo se ejecutará cuando el script se invoque directamente desde la terminal.

3.4.3 Adición de texto

Otra de las funcionalidades desarrolladas en el proyecto ha sido la adición automatizada de texto sobre imágenes, ya que permite insertar anotaciones, etiquetas o marcas. Este procedimiento es especialmente útil en tareas de documentación técnica, clasificación de muestras, generación de informes gráficos o para la preparación de imágenes para presentaciones científicas. Como en las funcionalidades anteriores se han desarrollado un script completamente integrado en GIMP, un script externo en Python acompañado de la biblioteca OpenCV y adicionalmente una implementación Python-Fu integrada en GIMP utilizando el nuevo sistema basado en GObject que requiere esta última versión del programa.

3.4.3.1 AddText.scm (Scheme)

```
(define (script-fu-add-text img drawable text pos-x-prcnt pos-y-prcnt font size
opacity color)
  (let* (
         ; Obtener el ancho y alto de la imagen
         (width (car (gimp-image-get-width img)))
         (height (car (gimp-image-get-height img)))

         ; Calcular posicion X e Y en pixeles a partir del porcentaje dado
         (pos-x (truncate (* (/ pos-x-prcnt 100) width)))
         (pos-y (truncate (* (/ pos-y-prcnt 100) height)))
        )

    ; Establecer el color para el texto
    (gimp-context-set-foreground color)

    (let* (
           ; Crear una nueva capa de texto con los parametros dados
           (text-layer (car (gimp-text-layer-new img text font size PIXELS)))
```

```
          ; Obtener las dimensiones del texto creado
          (text-width (car (gimp-drawable-get-width text-layer)))
          (text-height (car (gimp-drawable-get-height text-layer)))

          ; Calcular las coordenadas finales para centrar el texto
          (final-x (- pos-x (/ text-width 2)))
          (final-y (- pos-y (/ text-height 2)))
         )

     ; Insertar la capa de texto en la imagen
     (gimp-image-insert-layer img text-layer 0 -1)

     ; Establecer la posición de la capa de texto
     (gimp-layer-set-offsets text-layer final-x final-y)

     ; Establecer la opacidad de la capa de texto
     (gimp-layer-set-opacity text-layer opacity)

     ; Actualizar la interfaz de GIMP para reflejar los cambios
     (gimp-displays-flush))))

(script-fu-register
 "script-fu-add-text"
 "Add Text"
 "Inserta una capa de texto sobre la imagen."
 "Nicolas Bravo"
 "Nicolas Bravo"
 "2025"
 "RGB*, GRAY*"
 SF-IMAGE      "Imagen"              0
 SF-DRAWABLE   "Capa activa"         0
 SF-STRING     "Texto"               "Texto"
 SF-ADJUSTMENT "Posición X (%)"      '(50 0 100 1 10 0 1)
 SF-ADJUSTMENT "Posición Y (%)"      '(50 0 100 1 10 0 1)
 SF-FONT       "Fuente"              "Sans"
 SF-ADJUSTMENT "Tamaño fuente"       '(30 1 500 1 10 0 1)
 SF-ADJUSTMENT "Opacidad (%)"        '(100 0 100 1 10 0 1)
 SF-COLOR      "Color del texto"     '(0 0 0))
```

```
(script-fu-menu-register "script-fu-add-text" "<Image>/Filters/Script-Fu")
```

El script se desarrolló en Scheme con la intención de automatizar la inserción de texto en GIMP, permite posicionar el texto en función de coordenadas, controlar el tamaño, el color y la opacidad de la anotación.

```
(define (script-fu-add-text img drawable text pos-x-prcnt pos-y-prcnt font size
opacity color)
```

Primero se define la funcionalidad con el nombre "`script-fu-add-text`" y recibe como argumentos la imagen, "`img`", la capa activa, "`drawable`", el texto que se desea insertar, "`text`", las posiciones "x" e "y" como porcentajes, "`pos-x-prcnt y pos-y-percnt`", la fuente, "`font`", el tamaño de la letra, "`size`", la opacidad, "`opacity`", y el color del texto, "`color`".

```
(width (car (gimp-image-get-width img)))
(height (car (gimp-image-get-height img)))
```

En el cuerpo de la función, primero se obtienen las dimensiones de la imagen original, con su ancho y su largo, ya que son necesarias para más tarde poder calcular la posición del texto.

```
(pos-x (truncate (* (/ pos-x-prcnt 100) width)))
(pos-y (truncate (* (/ pos-y-prcnt 100) height)))
```

Para obtener las coordenadas exactas a partir de los porcentajes "x" e "y" proporcionados por el usuario, se dividen entre cien y se multiplican por el ancho o alto. La función "`truncate`" asegura que el resultado sea un valor entero para que pueda ser utilizado como coordenadas.

```
(gimp-context-set-foreground color)
```

A continuación se establece el color de primer plano que se utilizará para renderizar el texto, y así dibujar con el color especificado el texto.

```
(text-layer (car (gimp-text-layer-new img text font size PIXELS)))
```

Se genera la nueva capa de texto dentro de la imagen, usando la imagen, el texto, la fuente, el tamaño y el valor "`PIXELS`" que indica que el tamaño se expresa en píxeles.

```
(text-width (car (gimp-drawable-get-width text-layer)))
(text-height (car (gimp-drawable-get-height text-layer)))
(final-x (- pos-x (/ text-width 2)))
(final-y (- pos-y (/ text-height 2)))
```

Tras crear la nueva capa de texto, es necesario obtener sus dimensiones exactas para centrar correctamente el texto en las coordenadas calculadas. Después, se recalculan las coordenadas finales para ajustar el ancho y largo del texto, evitando que este quede desalineado.

```
(gimp-image-insert-layer img text-layer 0 -1)
(gimp-layer-set-offsets text-layer final-x final-y)
(gimp-layer-set-opacity text-layer opacity)
```

Finalmente, se inserta la nueva capa de texto dentro de la imagen, se establecen sus coordenadas absolutas definitivas y se ajusta la opacidad de la nueva capa.

```
(gimp-displays-flush)
```

Por último, se actualiza la visualización de la imagen en GIMP, reflejando los cambios que se han realizado.

3.4.3.2 add_text_plugin.py (Python-Fu)

```
import gi
import sys
gi.require_version("Gimp", "3.0")
gi.require_version("Gtk", "3.0")
from gi.repository import Gimp, GObject, Gtk

class AddSimpleText(Gimp.PlugIn):
    def do_query_procedures(self):
        return ["python-fu-add-simple-text"]

    def do_set_i18n(self, domain):
        return False

    def do_create_procedure(self, name):
        proc = Gimp.ImageProcedure.new(
            self, name, Gimp.PDBProcType.PLUGIN,
            self.run, None)

        proc.set_image_types("RGB*, GRAY*")
        proc.set_menu_label("Add text")
        proc.add_menu_path("<Image>/Filters/Python-Fu/")

        proc.add_string_argument("text", "Texto", "Texto a anadir", "Hola desde
GIMP", GObject.ParamFlags.READWRITE)
        proc.add_double_argument("x", "Posicion X", "Posicion X en pixeles", 0,
1000, 10, GObject.ParamFlags.READWRITE)
        proc.add_double_argument("y", "Posicion Y", "Posicion Y en pixeles", 0,
1000, 10, GObject.ParamFlags.READWRITE)
```

```
        default_font = Gimp.Font.get_by_name("Sans")
          proc.add_font_argument("font", "Fuente", "Fuente del texto", False,
default_font, True, GObject.ParamFlags.READWRITE)
        proc.add_double_argument("size", "Tamano", "Tamano fuente", 1, 200, 40,
GObject.ParamFlags.READWRITE)

        return proc

    def run(self, procedure, run_mode, image, drawables, config, run_data):

        if run_mode == Gimp.RunMode.INTERACTIVE:
            dialog = Gtk.Dialog(title="Add text", transient_for=None, flags=0)
            dialog.add_buttons(Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL,
                               Gtk.STOCK_OK, Gtk.ResponseType.OK)
            dialog.set_modal(True)
            dialog.set_default_size(300, 150)

            box = dialog.get_content_area()

            label_text = Gtk.Label(label="Texto a anadir:")
            entry_text = Gtk.Entry()
            entry_text.set_text(config.get_property("text"))

            label_x = Gtk.Label(label="Posicion X (px):")
            entry_x = Gtk.SpinButton.new_with_range(0, 10000, 1)
            entry_x.set_value(config.get_property("x"))

            label_y = Gtk.Label(label="Posicion Y (px):")
            entry_y = Gtk.SpinButton.new_with_range(0, 10000, 1)
            entry_y.set_value(config.get_property("y"))

            grid = Gtk.Grid(column_spacing=10, row_spacing=10, margin=10)
            grid.attach(label_text, 0, 0, 1, 1)
            grid.attach(entry_text, 1, 0, 1, 1)
            grid.attach(label_x, 0, 1, 1, 1)
            grid.attach(entry_x, 1, 1, 1, 1)
            grid.attach(label_y, 0, 2, 1, 1)
            grid.attach(entry_y, 1, 2, 1, 1)

            box.add(grid)
```

```
            dialog.show_all()

            response = dialog.run()

            if response == Gtk.ResponseType.OK:
                text = entry_text.get_text()
                x = entry_x.get_value()
                y = entry_y.get_value()
            else:
                dialog.destroy()
                  return procedure.new_return_values(Gimp.PDBStatusType.CANCEL,
None)

            dialog.destroy()

        else:
            return procedure.new_return_values(Gimp.PDBStatusType.CANCEL, None)

        font = config.get_property("font")
        size = config.get_property("size")
        unit = Gimp.Unit.pixel()

        text_layer = Gimp.TextLayer.new(image, text, font, size, unit)
        image.insert_layer(text_layer, None, 0)
        text_layer.set_offsets(int(x), int(y))

        Gimp.displays_flush()

        return procedure.new_return_values(Gimp.PDBStatusType.SUCCESS, None)

Gimp.main(AddSimpleText.__gtype__, sys.argv)
```

En esta implementación, la inserción de texto sobre imágenes se ha realizado mediante la utilización de Python-Fu integrado en GIMP 3.0. A diferencia de versiones anteriores de GIMP, la 3.0 ha introducido una arquitectura renovada para sus plugins en Python, basada en el módulo "`gi.repository`" y el sistema de objetos "`GObject`". Esto obliga a diseñar los plugins como clases que heredan de "`Gimp.PlugIn`", permitiendo registrar los procedimientos de manera directa dentro del sistema de procedimientos de GIMP. Además, el script está desarrollado para poder ser ejecutado directamente desde la interfaz gráfica del programa.

```
import gi
import sys
gi.require_version("Gimp", "3.0")
gi.require_version("Gtk", "3.0")
from gi.repository import Gimp, GObject, Gtk
```

El script comienza importando los módulos necesarios. Se especifica que se trabajará con GIMP 3.0 y GTK 3.0, cargando sus respectivas versiones. Posteriormente, se importan las librerías "`Gimp`", "`GObject`" y "`Gtk`" que proporcionarán las funcionalidades necesarias para la creación de procedimientos y para la gestión de ventanas de diálogo.

```
class AddSimpleText(Gimp.PlugIn):
     def do_query_procedures(self):
         return ["python-fu-add-text"]
```

A continuación se crea la nueva clase, que hereda de "`Gimp.PlugIn`", permitiendo registrar y gestionar el procedimiento de forma integrada dentro del programa. Además, la función quedará registrada bajo el nombre de "`python-fu-add-text`" dentro del entorno de GIMP.

```
def do_set_i18n(self, domain):
    return False
```

Este método establece si se va a emplear traducción de cadenas, como no es necesario se desactiva.

```
def do_create_procedure(self, name):
    proc = Gimp.ImageProcedure.new(
        self, name, Gimp.PDBProcType.PLUGIN,
        self.run, None)


    proc.set_image_types("RGB*, GRAY*")
    proc.set_menu_label("Add text")
    proc.add_menu_path("<Image>/Filters/Python-Fu/")


      proc.add_string_argument("text", "Texto", "Texto a anadir", "Hola desde
GIMP", GObject.ParamFlags.READWRITE)
      proc.add_double_argument("x", "Posicion X", "Posicion X en pixeles", 0,
1000, 10, GObject.ParamFlags.READWRITE)
      proc.add_double_argument("y", "Posicion Y", "Posicion Y en pixeles", 0,
1000, 10, GObject.ParamFlags.READWRITE)
      default_font = Gimp.Font.get_by_name("Sans")
      proc.add_font_argument("font", "Fuente", "Fuente del texto", False,
default_font, True, GObject.ParamFlags.READWRITE)
```

```
     proc.add_double_argument("size", "Tamano", "Tamano fuente", 1, 200, 40,
GObject.ParamFlags.READWRITE)


    return proc
```

Posteriormente, se instancia un nuevo procedimiento “ImageProcedure”, especificando el tipo de plugin y asignando la función “self.run” como lógica principal de ejecución. Se indican el tipo de imágenes sobre las cuales se podrá aplicar el procedimiento, así como la etiqueta utilizada dentro de GIMP en la ruta del menú que se especifica. A continuación se definen los argumentos que acepta el procedimiento, siendo el texto, las coordenadas “x” e “y” con sus rangos de valores, la fuente (que por defecto será “Sans”) y su tamaño. Una vez definidos todos los parámetros, se devuelve el procedimiento completamente configurado.

```
def run(self, procedure, run_mode, image, drawables, config, run_data):
```

El método “run” gestiona la lógica de ejecución cuando el procedimiento es invocado, recibe el propio procedimiento, el modo de ejecución, la imagen sobre la que se trabaja y la configuración de los parámetros.

```
if run_mode == Gimp.RunMode.INTERACTIVE:
      ...
else:
      return procedure.new_return_values(Gimp.PDBStatusType.CANCEL, None)
```

Se comprueba que el modo de ejecución sea en modo interactivo, y en caso contrario se cancela la operación. Esta decisión se tomó debido a problemas encontrados a la hora de ejecutar el procedimiento desde la terminal, estos problemas serán más detallados en el apartado “Problemas y soluciones”.

```
dialog = Gtk.Dialog(title="Add text", transient_for=None, flags=0)
dialog.add_buttons(Gtk.STOCK_CANCEL, Gtk.ResponseType.CANCEL,
                                Gtk.STOCK_OK, Gtk.ResponseType.OK)
dialog.set_modal(True)
dialog.set_default_size(300, 150)


box = dialog.get_content_area()


label_text = Gtk.Label(label="Texto a anadir:")
entry_text = Gtk.Entry()
entry_text.set_text(config.get_property("text"))


label_x = Gtk.Label(label="Posicion X (px):")
entry_x = Gtk.SpinButton.new_with_range(0, 10000, 1)
```

```
entry_x.set_value(config.get_property("x"))


label_y = Gtk.Label(label="Posicion Y (px):")
entry_y = Gtk.SpinButton.new_with_range(0, 10000, 1)
entry_y.set_value(config.get_property("y"))


grid = Gtk.Grid(column_spacing=10, row_spacing=10, margin=10)
grid.attach(label_text, 0, 0, 1, 1)
grid.attach(entry_text, 1, 0, 1, 1)
grid.attach(label_x, 0, 1, 1, 1)
grid.attach(entry_x, 1, 1, 1, 1)
grid.attach(label_y, 0, 2, 1, 1)
grid.attach(entry_y, 1, 2, 1, 1)


box.add(grid)
dialog.show_all()


response = dialog.run()
```

Todo este bloque corresponde a la construcción del cuadro de diálogo con "`Gtk.Dialog`", utilizando botones de confirmación y cancelación, así como los widgets necesarios para introducir el texto y la posición, mediante etiquetas "`Gtk.Label`" y campos de entrada "`Gtk.Entry`" y "`Gtk.SpinButton`".

```
if response == Gtk.ResponseType.OK:
    text = entry_text.get_text()
    x = entry_x.get_value()
    y = entry_y.get_value()
else:
    dialog.destroy()
    return procedure.new_return_values(Gimp.PDBStatusType.CANCEL, None)
```

Si el usuario pulsa "OK", los valores proporcionados se almacenarán en los respectivos parámetros, en caso contrario se abortará la operación.

```
font = config.get_property("font")
size = config.get_property("size")
unit = Gimp.Unit.pixel()
```

Una vez capturados los parámetros, se recuperan también los valores para la fuente y su tamaño.

```
text_layer = Gimp.TextLayer.new(image, text, font, size, unit)
```

```
image.insert_layer(text_layer, None, 0)
text_layer.set_offsets(int(x), int(y))
Gimp.displays_flush()
```

Finalmente, se crea la capa de texto y se inserta en la imagen, ubicada en las coordenadas especificadas. También se actualizan los displays de GIMP para reflejar los cambios visualmente en la interfaz.

```
return procedure.new_return_values(Gimp.PDBStatusType.SUCCESS, None)
```

A continuación el procedimiento retorna indicando que la ejecución ha finalizado correctamente.

```
Gimp.main(AddSimpleText.__gtype__, sys.argv)
```

Por último la clase queda registrada en el sistema interno de plugins de GIMP.

3.4.3.3 add_text.py (Python y OpenCV)

```
import cv2
import argparse


def main():
    parser = argparse.ArgumentParser(description="Agregar texto a una imagen usando OpenCV.")
    parser.add_argument('--image', type=str, required=True, help='Ruta de la imagen original.')
     parser.add_argument('--text', type=str, required=True, help='Texto a agregar.')
     parser.add_argument('--x', type=int, default=50, help='Posicion X del texto.')
     parser.add_argument('--y', type=int, default=50, help='Posicion Y del texto.')
    parser.add_argument('--font', type=int, default=0, help='Tipo de fuente OpenCV (0 a 7).')
     parser.add_argument('--size', type=float, default=1.0, help='Tamano del texto.')
   parser.add_argument('--color', type=int, nargs=3, default=[255, 255, 255], help='Color del texto en BGR (ej: 255 255 255).')
      parser.add_argument('--exit', type=str, default='imagen_salida.png', help='Ruta de la imagen de salida.')


    args = parser.parse_args()


    img = cv2.imread(args.image)
    if img is None:
```

```
        print(f"No se pudo cargar la imagen: {args.image}")
        return

    font = args.font if 0 <= args.font <= 7 else 0
    color = tuple(args.color)
     cv2.putText(img, args.text, (args.x, args.y), font, args.size, color, 2,
cv2.LINE_AA)

    cv2.imwrite(args.exit, img)
    print(f"Imagen guardada en: {args.exit}")

if __name__ == "__main__":
    main()
```

Además de las soluciones integradas en GIMP, se desarrolló una tercera alternativa utilizando Python junto a la biblioteca OpenCV. Esta solución es completamente independiente de GIMP y ha sido diseñada para funcionar de forma autónoma desde la terminal.

El script comienza importando las librerías necesarias, mencionadas en los anteriores scripts de Python, "`cv2`" y "`argparse`". Por otra parte, dentro de la función principal "`main`", se definen los argumentos de entrada, siendo la ruta de la imagen de entrada, el texto que se desea insertar en la imagen, las posiciones x e y del texto en píxeles, la fuente según los valores disponibles en OpenCV (de 0 a 7), el tamaño de la fuente, el color del texto y la ruta del archivo donde se almacenará la imagen resultante. A continuación, de la misma manera que en el resto de los scripts en Python con OpenCV, se carga la imagen original, y cuando haya un error en la carga el script finaliza y muestra un mensaje de error.

```
font = args.font if 0 <= args.font <= 7 else 0
color = tuple(args.color)
cv2.putText(img, args.text, (args.x, args.y), font, args.size, color, 2,
cv2.LINE_AA)
```

En este bloque se prepara el texto a insertar, validando primero el valor de la fuente. Después, se convierte el color proporcionado por el usuario en formato BGR y se inserta el texto con los parámetros ya preparados y especificando el tipo de suavizado con "`cv2.LINE_AA`", que permite obtener un renderizado de texto antialiasing.

Finalmente la imagen con el texto insertado se guarda en la ruta especificada y se muestra por consola un mensaje indicando el éxito de la operación y la ruta donde ha sido almacenada.

3.4.4 Expansión de selección

Dentro de las funcionalidades implementadas se ha desarrollado un procedimiento que permite expandir automáticamente una selección activa en una imagen, escalándola proporcionalmente a través de un factor de expansión definido por el usuario, mejorando la percepción visual de los detalles mediante una interpolación que añade píxeles para aumentar su resolución aparente. Este procedimiento puede resultar de gran utilidad en procesos de ajuste visual, de mejora de detalles en zonas seleccionadas o de preparación de regiones de interés para su posterior análisis o tratamiento gráfico.

3.4.4.1 Expansion.scm (Scheme)

```
(define (script-fu-expand img layer factor)
  (let* (
         ; Validar si hay seleccion activa
         (selection (= (car (gimp-selection-is-empty img)) FALSE))
        )

    (if (not selection)
        (gimp-message "No hay ninguna seleccion activa.")
        (begin
          ; Crear una copia de la capa activa
          (let* ((new-layer (car (gimp-layer-copy layer TRUE))))

            (gimp-image-undo-group-start img)
            (gimp-context-set-interpolation INTERPOLATION-CUBIC)
            (gimp-image-insert-layer img new-layer 0 -1)

            ; Aplicar mascara basada en la seleccion
            (let  ((mask  (car  (gimp-layer-create-mask  new-layer  ADD-MASK-
SELECTION))))
              (gimp-layer-add-mask new-layer mask)
              (gimp-layer-remove-mask new-layer MASK-APPLY))

            ; Comprobar si la capa resultante contiene algo visible
            (if (gimp-drawable-has-alpha new-layer)
                (let* (
                       (width (car (gimp-drawable-get-width new-layer)))
                       (height  (car (gimp-drawable-get-height new-layer)))
                       (offsets (gimp-drawable-get-offsets new-layer))
                       (offset-x (car offsets))
```

```
                        (offset-y (cadr offsets))
                        (new-width (* width factor))
                        (new-height  (* height factor))
                        (new-offset-x (- offset-x (/ (- new-width width) 2)))
                        (new-offset-y (- offset-y (/ (- new-height height) 2)))
                       )

                  ; Escalar y centrar
                  (gimp-layer-scale new-layer new-width new-height FALSE)
                  (gimp-layer-set-offsets new-layer new-offset-x new-offset-y)
                  (gimp-item-set-name new-layer "Seleccion Expandida"))

                ; Si no hay pixeles visibles, eliminar la capa nueva
                (begin
                  (gimp-image-remove-layer img new-layer)
                  (gimp-message "La seleccion no contiene pixeles visibles."))))

            (gimp-displays-flush)
            (gimp-image-undo-group-end img)))))

(script-fu-register
 "script-fu-expand"
 "Expand"
 "Duplica y expande solo la seleccion activa usando interpolacion cubica."
 "Nicolas Bravo"
 "Nicolas Bravo"
 "2025"
 "RGB*, GRAY*"
 SF-IMAGE "Imagen" 0
 SF-DRAWABLE "Capa" 0
 SF-ADJUSTMENT "Factor de expansión" '(1.5 0.1 10.0 0.1 1 2))

(script-fu-menu-register "script-fu-expand" "<Image>/Filters/Script-Fu")
```

Es importante destacar que esta funcionalidad está diseñada exclusivamente para ser utilizada de forma interactiva dentro de la interfaz de GIMP, ya que requiere que el usuario haya realizado previamente una selección activa sobre la imagen. Dado que el script depende de dicha selección, no resulta práctico invocarlo desde una terminal donde no es posible establecer selecciones manuales. Por esta misma razón, la implementación se ha desarrollado únicamente en Scheme, dado que este entorno resulta mucho más natural para operaciones interactivas dentro de la interfaz de GIMP.

```
(define (script-fu-expand img layer factor)
```

El procedimiento comienza con la definición de la función “`script-fu-expand`”, que recibe tres argumentos, la imagen, “`img`”, la capa activa sobre la que se va a trabajar, “`layer`”, y el factor de expansión, “`factor`”.

```
(let* (
         (selection (= (car (gimp-selection-is-empty img)) FALSE))
        )
(if (not selection)
        (gimp-message "No hay ninguna seleccion activa.")
```

Lo primero que realiza el script es comprobar si existe o no una selección activa en la imagen. Si no existe ninguna, el procedimiento informa al usuario con un mensaje y termina su ejecución sin realizar modificaciones.

```
(begin
          (let* ((new-layer (car (gimp-layer-copy layer TRUE))))
```

En caso de que sí que exista una selección activa, se inicia el proceso de expansión, empezando por crear una copia exacta de la capa activa. Esta copia permitirá aplicar las transformaciones sin afectar a la capa original, preservando así la integridad de la imagen original.

```
(gimp-image-undo-group-start img)
(gimp-context-set-interpolation INTERPOLATION-CUBIC)
(gimp-image-insert-layer img new-layer 0 -1)
```

El siguiente bloque de operaciones consiste en agrupar todo el conjunto de las modificaciones como una única acción dentro del historial de deshacer de GIMP. También se establece la interpolación cúbica que permite obtener una mejor calidad en el escalado de la selección, y posteriormente, la nueva capa se inserta en la imagen.

```
(let ((mask (car (gimp-layer-create-mask new-layer ADD-MASK-SELECTION))))
              (gimp-layer-add-mask new-layer mask)
              (gimp-layer-remove-mask new-layer MASK-APPLY))
```

A continuación se crea y se aplica una máscara sobre la nueva capa, basada en la selección activa. El objetivo de esta máscara es aislar únicamente la zona seleccionada de la imagen. Una vez creada se aplican los píxeles de la máscara directamente en la imagen, de modo que la nueva capa contiene solo el contenido de la selección.

```
(if (gimp-drawable-has-alpha new-layer)
    (let* (
           (width (car (gimp-drawable-get-width new-layer)))
```

```
          (height  (car (gimp-drawable-get-height new-layer)))
          (offsets (gimp-drawable-get-offsets new-layer))
          (offset-x (car offsets))
          (offset-y (cadr offsets))
          (new-width (* width factor))
          (new-height  (* height factor))
          (new-offset-x (- offset-x (/ (- new-width width) 2)))
          (new-offset-y (- offset-y (/ (- new-height height) 2)))
         )
```

Antes de proceder al escalado, el script comprueba si la nueva capa resultante tiene contenido visible, es decir, si la máscara aplicada ha dejado algún píxel activo, y en ese caso se calculan la anchura y altura actuales de la capa, sus posiciones relativas dentro de la imagen, y a partir de ahí se determinan las nuevas dimensiones escaladas multiplicándolas por el factor introducido por el usuario. También se recalculan los nuevos offsets para mantener la expansión centrada respecto a la selección original.

```
(gimp-layer-scale new-layer new-width new-height FALSE)
(gimp-layer-set-offsets new-layer new-offset-x new-offset-y)
(gimp-item-set-name new-layer "Seleccion Expandida"))
```

La capa es escalada a las nuevas dimensiones calculadas anteriormente, se reposiciona y renombra la capa para facilitar su identificación.

```
(begin
  (gimp-image-remove-layer img new-layer)
  (gimp-message "La seleccion no contiene pixeles visibles.")))
```

En el caso de que la selección no contuviera píxeles visibles tras aplicar la máscara, la nueva capa es eliminada de la imagen y se le informa al usuario de que la selección estaba vacía.

```
(gimp-displays-flush)
(gimp-image-undo-group-end img)
```

Una vez completadas todas las operaciones, se actualiza la interfaz gráfica para mostrar el resultado y se cierra el grupo de operaciones de deshacer antes creado, agrupando todas las acciones como una única operación de edición en el historial de GIMP.

3.4.5 Resaltado de regiones

Como última funcionalidad implementada dentro de este proyecto, se desarrolló un procedimiento que permite identificar y resaltar regiones de la imagen que presenten un color similar al de un píxel de referencia especificado por el usuario, generando una nueva capa transparente donde se destaca visualmente

la zona coincidente, ya sea copiando el color original o aplicando un color resaltado definido. Esta herramienta resulta particularmente útil en contextos donde es necesario analizar o destacar objetos homogéneos dentro de una imagen, como pueden ser células, partículas o zonas de color uniforme en imágenes microscópicas, biomédicas o técnicas.

3.4.5.1 RegionHighlight.scm (Scheme)

```
; Funcion auxiliar que compara dos colores RGB y verifica si son similares dentro de una tolerancia dada
(define (color-similar? c1 c2 tolerance)
  (let ((r1 (list-ref c1 0)) (g1 (list-ref c1 1)) (b1 (list-ref c1 2))
        (r2 (list-ref c2 0)) (g2 (list-ref c2 1)) (b2 (list-ref c2 2)))
    ; Compara componente por componente rgb dentro de la tolerancia
    (and (<= (abs (- r1 r2)) tolerance)
         (<= (abs (- g1 g2)) tolerance)
         (<= (abs (- b1 b2)) tolerance))))


(define (script-fu-highlight-region img drawable tolerance margin xref yref mode highlight-color opacity)
  (let* (
         ; Obtiene dimensiones de la imagen
         (width (car (gimp-image-get-width img)))
         (height (car (gimp-image-get-height img)))

         ; Obtiene el color del pixel de referencia en las coordenadas dadas
         (color-ref (car (gimp-drawable-get-pixel drawable xref yref)))

         ; Define los limites del area a recorrer segun el margen especificado
         (xmin (max 0 (- xref margin)))
         (xmax (min (- width 1) (+ xref margin)))
         (ymin (max 0 (- yref margin)))
         (ymax (min (- height 1) (+ yref margin)))

         ; Crea una nueva capa transparente
         (new-layer (car (gimp-layer-new img "Color Similar" width height RGBA-IMAGE 100 LAYER-MODE-NORMAL)))

         ; Convierte la opacidad (0-100) a valor alfa (0-255)
         (alpha (inexact->exact (round (* 2.55 opacity))))
        )
```

```
    ; Inserta la nueva capa en la imagen
    (gimp-image-insert-layer img new-layer 0 -1)

    ; Rellena la nueva capa con transparencia
    (gimp-drawable-fill new-layer 4)

    ; Recorre los pixeles dentro del area definida
    (do ((x xmin (+ x 1)))
        ((> x xmax))
      (do ((y ymin (+ y 1)))
          ((> y ymax))
        (let* (
               ; Obtiene el color del pixel actual
               (pixel (car (gimp-drawable-get-pixel drawable x y))))
          ; Si el color es similar al color de referencia
          (when (color-similar? pixel color-ref tolerance)
            (if (= mode 0)
                ; Modo 0: copia el color original a la nueva capa
                (gimp-drawable-set-pixel new-layer x y pixel)
                ; Modo 1: aplica un color resaltado con opacidad
                (gimp-drawable-set-pixel new-layer x y
                  (list (list-ref highlight-color 0)
                        (list-ref highlight-color 1)
                        (list-ref highlight-color 2)
                        alpha)))))))

    ; Refresca la pantalla
    (gimp-displays-flush)
    (gimp-message "Region resaltada en nueva capa.")))


(script-fu-register
 "script-fu-highlight-region"
 "Highlight Region"
 "Crea una nueva capa con los pixeles similares al indicado. Se puede resaltar
con el color original o un color escogido."
 "Nicolas Bravo"
 "Nicolas Bravo"
 "2025"
```

```
 "RGB*, GRAY*"
 SF-IMAGE      "Imagen"     0
 SF-DRAWABLE   "Capa"       0
 SF-ADJUSTMENT "Tolerancia" '(30 0 255 1 10 0)
 SF-ADJUSTMENT "Margen (px)" '(20 1 200 1 10 0)
 SF-ADJUSTMENT "Coordenada X del pixel" '(0 0 10000 1 10 0)
 SF-ADJUSTMENT "Coordenada Y del pixel" '(0 0 10000 1 10 0)
 SF-OPTION     "Modo de resaltado" '("Color original" "Color resaltado")
 SF-COLOR      "Color resaltado" '(255 0 0)
 SF-ADJUSTMENT "Opacidad (0–100)" '(50 0 100 1 5 0)
)


(script-fu-menu-register "script-fu-highlight-region" "<Image>/Filters/Script-
Fu")
```

Al igual que la funcionalidad de expansión de selección, este procedimiento ha sido diseñado específicamente para su uso interactivo desde dentro de GIMP, ya que requiere que el usuario seleccione el píxel de referencia indicando sus coordenadas dentro la imagen, su uso desde terminal no resulta práctico ni tiene sentido operativo. Por esta razón, la implementación se ha desarrollado exclusivamente en Scheme mediante Script-Fu, aprovechando la integración directa con la interfaz de usuario de GIMP.

```
(define (color-similar? c1 c2 tolerance)
  (let ((r1 (list-ref c1 0)) (g1 (list-ref c1 1)) (b1 (list-ref c1 2))
        (r2 (list-ref c2 0)) (g2 (list-ref c2 1)) (b2 (list-ref c2 2)))
    (and (<= (abs (- r1 r2)) tolerance)
         (<= (abs (- g1 g2)) tolerance)
         (<= (abs (- b1 b2)) tolerance))))
```

El procedimiento comienza definiendo una función auxiliar encargada de comparar dos colores RGB componente a componente. Esta comparación sirve para determinar si dos colores son los suficientemente similares, considerando una tolerancia introducida por el usuario. Si la diferencia absoluta de cada componente RGB está dentro del margen de tolerancia, los colores se consideran equivalentes.

```
(define (script-fu-highlight-region img drawable tolerance margin xref yref mode
highlight-color opacity)
```

A continuación se define el procedimiento principal con el nombre “`script-fu-highlight-region`” que recibe los parámetros de entrada, imagen activa, “`img`”, capa activa, “`drawable`”, margen de tolerancia para determinar la similitud entre colores, “`tolerance`”, margen de píxeles que delimita el área de búsqueda alrededor del píxel de referencia, “`margin`”, coordenadas “x” e “y” del píxel de referencia seleccionado, “`xref e yref`”, tipo de resaltado siendo este copiando

el color original o aplicándole un color para destacar, “mode”, color que se aplicará en modo resaltado, “highlight-color”, y opacidad que tendrá el color en modo resaltado, “opacity”.

```
(let* (
        (width (car (gimp-image-get-width img)))
        (height (car (gimp-image-get-height img)))
```

Dentro de la lógica de la funcionalidad primero se obtienen las dimensiones de la imagen, que permitirán limitar posteriormente el área a recorrer, evitando desbordamientos fuera de los bordes de la imagen.

```
(color-ref (car (gimp-drawable-get-pixel drawable xref yref)))
```

A continuación se obtiene el color del píxel de referencia, utilizando las coordenadas proporcionadas por el usuario. Este color obtenido será el color base con el que se compararán todos los píxeles dentro del área definida.

```
 (xmin (max 0 (- xref margin)))
 (xmax (min (- width 1) (+ xref margin)))
 (ymin (max 0 (- yref margin)))
 (ymax (min (- height 1) (+ yref margin)))
```

Se definen los límites del área de búsqueda a partir del margen especificado. Esto crea una “ventana” cuadrada centrada en el píxel de referencia, que será la zona de análisis.

```
(new-layer (car (gimp-layer-new img "Color Similar" width height RGBA-IMAGE 100
LAYER-MODE-NORMAL)))
(alpha (inexact->exact (round (* 2.55 opacity))))
```

Se crea una nueva capa transparente llamada “Color Similar” del mismo tamaño que la imagen original, donde se irán almacenando los píxeles resaltados. También se convierte la opacidad, que originalmente está en el rango 0-100%, al rango 0-255 que maneja el canal alfa de GIMP.

```
(gimp-image-insert-layer img new-layer 0 -1)
(gimp-drawable-fill new-layer 4)
```

La nueva capa es insertada en la imagen y rellenada completamente con transparencia antes de empezar a procesar los píxeles.

```
(do ((x xmin (+ x 1)))
    ((> x xmax))
  (do ((y ymin (+ y 1)))
      ((> y ymax))
    (let* (
```

```
          (pixel (car (gimp-drawable-get-pixel drawable x y))))
      (when (color-similar? pixel color-ref tolerance)
        (if (= mode 0)
            (gimp-drawable-set-pixel new-layer x y pixel)
            (gimp-drawable-set-pixel new-layer x y
              (list (list-ref highlight-color 0)
                    (list-ref highlight-color 1)
                    (list-ref highlight-color 2)
                    alpha)))))))
```

En este bloque central se recorren todos los píxeles uno a uno dentro del área seleccionada. Para cada píxel se compara su color con el color de referencia teniendo en cuenta la tolerancia especificada. Si el píxel es similar, se aplicará el resaltado dependiendo del modo seleccionado, si se seleccionó el modo "Color original", se copia el color original del píxel a la nueva capa, en cambio, si se seleccionó el modo "Color resaltado", se pinta en la nueva capa el píxel con el color y la opacidad indicadas por el usuario.

```
(gimp-displays-flush)
(gimp-message "Region resaltada en nueva capa.")
```

Finalmente, se actualiza la pantalla para reflejar los cambios realizados y se le informa al usuario de que la operación ha concluido correctamente.

3.5 Automatización desde terminal

Una de las principales ventajas de la utilización de GIMP como plataforma de procesamiento de imágenes es su capacidad de ser controlado no solo desde su interfaz gráfica, sino también desde la terminal mediante su ejecución en modo no interactivo. Esta característica permite integrar los scripts desarrollados en flujos de trabajo completamente automatizados, facilitando el procesamiento por lotes de grandes volúmenes de imágenes sin intervención manual.

Durante el desarrollo de este trabajo se han implementado diversas funcionalidades, algunas de las cuales han sido diseñadas específicamente para ser compatibles con la automatización desde la terminal. Esta posibilidad ha sido especialmente aprovechada en los scripts desarrollados en Scheme, que permiten ser ejecutados mediante el sistema batch de GIMP gracias a su perfecta integración con el motor interno de procedimientos del programa.

La invocación de estos scripts desde terminal se realiza mediante el siguiente patrón de ejecución.

```
flatpak run org.gimp.GIMP –no-interface –batch-interpreter=plug-in-script-fu-eval –batch "(script-fu-procedimiento parámetros)" –batch "(gimp-quit 0)"
```

Con esta invocación, GIMP se inicia sin abrir la interfaz gráfica ("`--no-interface`"), se especifica el intérprete de Script-Fu ("`--batch-interpreter`"), y finalmente se invoca el procedimiento deseado pasando los parámetros correspondientes.

Por ejemplo para la funcionalidad de conversión de formatos de imagen, el script puede ser ejecutado de forma completamente automatizada sobre un conjunto de imágenes simplemente recorriendo un directorio mediante un bucle "`for`" desde la terminal, invocando el procedimiento de conversión con cada archivo.

```
for img in /ruta/imagenes/*.jpg; do
  gimp --no-interface --batch-interpreter=plug-in-script-fu-eval --batch "(script-fu-convert-image-format \"$img\" \"/ruta/salida/\" \"png\")" --batch "(gimp-quit 0)"
done
```

De este modo, es posible convertir cientos o miles de imágenes entre formatos de forma desatendida, lo que resulta especialmente útil en contextos donde se trabaja con grandes lotes de datos gráficos, como sucede en entornos de investigación científica o documentación técnica.

Del mismo modo, funcionalidades como la binarización y la adición de texto han sido diseñadas para admitir su ejecución desde la terminal. Estas tareas permiten ser parametrizadas completamente mediante los argumentos de cada procedimiento, lo que facilita su integración en scripts más amplios.

Por el contrario, como se comentó anteriormente, funcionalidades como la expansión de selección o el resaltado de regiones no se han diseñado con la intención de ser ejecutadas automáticamente desde la terminal, ya que requieren de la interacción directa del usuario sobre la imagen, principalmente en la selección de regiones o en la elección de píxeles de referencia.

Además, los scripts desarrollados externamente en Python con OpenCV también permiten la automatización desde la terminal. Al no depender directamente de GIMP, estos scripts son especialmente útiles para su ejecución en servidores, sistemas de procesamiento masivo o entornos donde no se requiere un editor gráfico. La ejecución de estos scripts se realiza mediante el propio intérprete de Python con el siguiente patrón de ejecución.

```
python3 procedimiento.py –-parámetro1 –parámetro2
```

Gracias a estas posibilidades de automatización, el conjunto de scripts desarrollados permite cubrir tanto escenarios de procesamiento individual e interactivo, como flujos de trabajo completamente automatizados, adaptándose a distintas necesidades y perfiles de usuario dentro del ámbito de la automatización gráfica.

3.6 Problemas y soluciones

A lo largo del proyecto el flujo de trabajo no ha sido lineal, ya que cada versión, dependencia o modo de ejecución destapaban impedimentos que obligaban a cuestionar decisiones previas y refinar la estrategia a seguir. El presente apartado recoge algunos de esos contratiempos y describe las soluciones adoptadas para mantener la estabilidad del código y la portabilidad de los scripts. Lejos de ser simples incidencias, estos retos han marcado la identidad del proyecto, pues forzaron a explorar a fondo la API de GIMP 3.0, a diseñar utilidades defensivas y a establecer buenas prácticas que hoy permiten alternar entre el uso interactivo de la interfaz y la automatización por terminal.

El recorrido del proyecto empezó en GIMP 2.10, sin embargo, en las primeras investigaciones se comprobó que en esta versión el lenguaje Python-Fu seguía anclado a Python 2.7, y por tanto carecía de compatibilidad con las bibliotecas modernas de tratamiento de imágenes. Este hallazgo obligó a migrar al reciente GIMP 3.0, aun sabiendo que esta decisión traería nuevos retos de infraestructura y documentación, pero garantizaba acceso pleno a Python 3.

El modo batch en Python-Fu mostró ser el talón de Aquiles de la nueva versión ya que el intérprete "`--batch-interpreter=python-fu-eval`" no expone "Gimp.pdb", por tanto, los procedimientos registrados no son invocables como ocurría en la versión 2.10. Además, gran parte de la API se ha rediseñado pensando en la GUI y no en la ejecución no interactiva. Mientras no se resuelva, los plugins Python-Fu se mantienen como herramientas interactivas y, para flujos desatendidos, se delega la lógica de imagen a scripts externos en Python puro o a Script-Fu, cuyo intérprete batch sigue siendo plenamente funcional.

En Script-Fu aparecieron incompatibilidades más sutiles, ya que GIMP 3.0 en algunas situaciones devuelve vectores donde antes devolvía listas, como por ejemplo pasa con "`gimp-image-get-layers`", lo que provocó errores de tipo que se depuraron haciendo uso de los mensajes "`gimp-message`". Además, en Script-Fu, el procedimiento "`gimp-font-get-by-name`" no retorna un objeto GimpFont válido, como pone en su descripción, lo que complicó el uso del script "AddText.scm" desde la terminal debido a la imposibilidad de transformar un string a un GimpFont que necesita el procedimiento "`gimp-text-layer-new`". Por ello el uso de esta funcionalidad desde la terminal se delegó a la versión de Python puro con OpenCV.

La adopción total de GEGL como motor de procesamiento generó confusiones con los rangos de entrada en operaciones como "`gimp-drawable-threshold`" que siguen aceptando umbrales entre 0 y 255 en la capa de scripting, pero internamente se normalizan entre 0 y 1. Para mantener la coherencia con la documentación clásica, los scripts convierten los valores a punto flotante antes de hacer la llamada, de modo que el usuario puede trabajar con una escala de 0 a 100 más intuitiva.

Al diseñar "RegionHighlight.scm" surgió una limitación de la propia interfaz de GIMP debido a que no dispone de un procedimiento que devuelva automáticamente las coordenadas del píxel haciendo clic con el cursor sobre una imagen. La solución consistió en pedir al usuario que introduzca las coordenadas de referencia en el cuadro de diálogo manualmente, y posteriormente validar que estas coordenadas se encuentren dentro de los límites de la imagen. Aunque es menos intuitivo que un clic directo, el método resulta fiable y multiplataforma.

Por último, las operaciones que manipulan selecciones o canales alfa mostraron fallos cuando se ejecutaban sobre capas sin transparencia o con la selección vacía, por ello se añadieron comprobaciones previas y mensajes de aviso claros, evitando bloqueos y garantizando que los flujos por lotes continúen aun cuando alguna imagen no cumpla los requisitos.

Todos estos obstáculos, lejos de entorpecer el proyecto, obligaron a profundizar en la nueva arquitectura de GIMP 3.0 y a implantar prácticas de programación robusta. El resultado final son scripts estables, portables y capaces de operar tanto desde la interfaz como automáticamente desde la terminal cuando se requiera.

4 Pruebas y resultados

Una vez desarrollados los distintos scripts que conforman las funcionalidades implementadas, se procedió a su validación mediante un conjunto de pruebas específicas. El objetivo de estas pruebas es, por un lado, comprobar el correcto funcionamiento técnico de cada uno de los procedimientos, y por otro lado, mostrar de forma visual los resultados obtenidos, reflejando los efectos producidos sobre imágenes de prueba representativas.

En las pruebas realizadas se han contemplado tanto los modos de ejecución interactivo, directamente desde la interfaz gráfica de GIMP, como los modos no interactivos, ejecutando los scripts desde la terminal. De este modo, se han cubierto los distintos escenarios de uso previstos en el diseño de los scripts.

A lo largo de este capítulo se detallan, para cada funcionalidad, los parámetros empleados durante las pruebas, los entornos de ejecución, las imágenes utilizadas y los resultados obtenidos. Además se adjuntan capturas de pantalla que ilustran de forma directa el comportamiento y efecto de cada procedimiento sobre las imágenes procesadas.

4.1 Pruebas con la conversión de formato de imagen

Para la validación de la funcionalidad de conversión de formato de imagen se han realizado pruebas sobre imágenes con extensión “JPG” para transicionarlas hacia otros formatos, ejecutando los distintos scripts desarrollados tanto desde GIMP como desde terminal, comprobando la correcta conversión entre formatos de entrada y salida, así como la preservación de la integridad de las imágenes.

En primer lugar se realizó la conversión directamente desde la interfaz de GIMP mediante el script “AddText.scm”, diseñado en Scheme. El usuario selecciona la imagen deseada, la ruta de salida, el nuevo nombre y el formato.

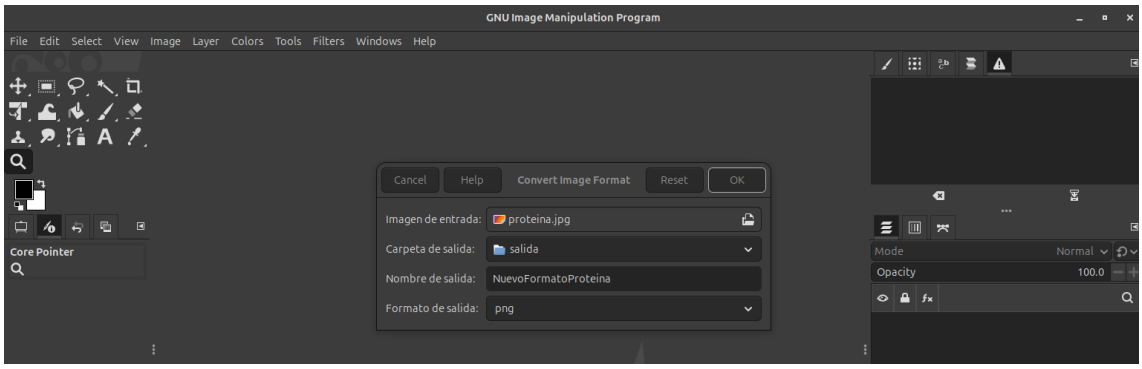

Ilustración 1: Ejecución de ChangeFormat.scm desde GIMP.

En esta prueba se partió de una imagen original en formato JPG, seleccionando la conversión a formato PNG. El resultado muestra que la imagen se mantiene visualmente idéntica, confirmando que el script manipula únicamente la extensión del archivo sin alterar su contenido gráfico.

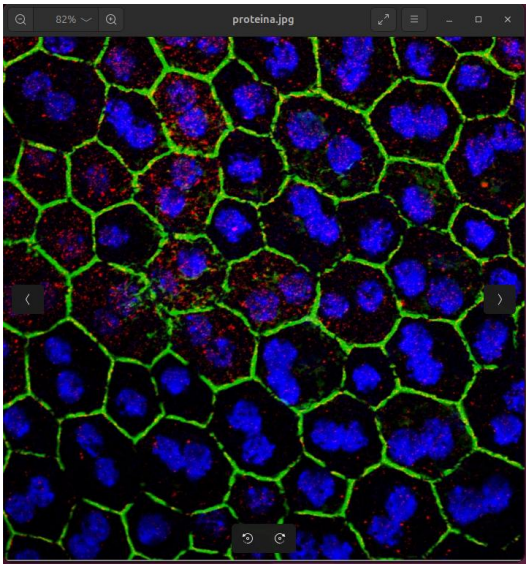

Ilustración 2: Imagen proteina.jpg original.

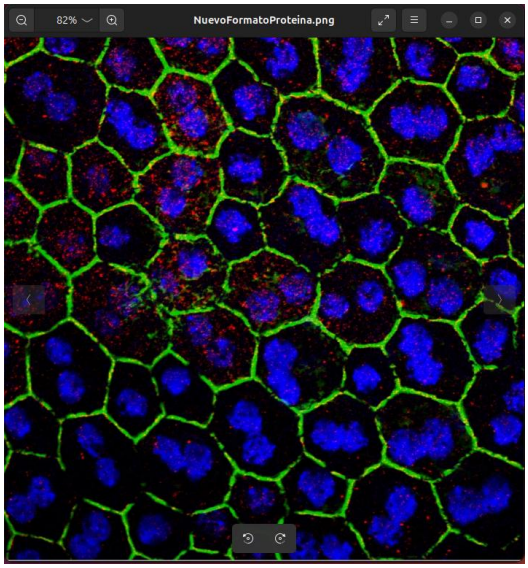

Ilustración 3: Imagen proteina.jpg convertida a PNG.

A continuación, se ejecutó el mismo script, esta vez en modo automatizado desde la terminal. Para ello, se utilizó la ejecución batch de Script-Fu dentro de GIMP, invocando el programa mediante Flatpak.

Ilustración 4: Ejecución en terminal de ChangeFormat.scm.

Tras la ejecución del comando, la imagen fue convertida de forma automática y almacenada correctamente en el directorio de salida especificado. El resultado obtenido es completamente equivalente al de la conversión manual.

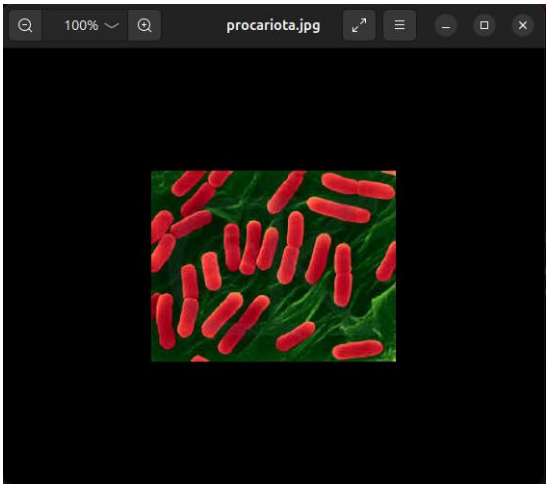

Ilustración 5: Imagen procariota.jpg original.

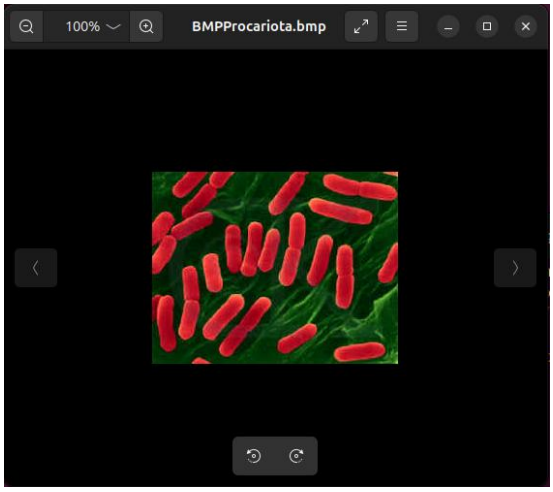

Ilustración 6: Imagen procariota.jpg convertida a BMP.

Por último, se validó el script externo desarrollado en Python utilizando la librería OpenCV, "change_format.py", ejecutado desde la terminal de manera completamente independiente a GIMP. El script fue invocado proporcionando los parámetros de entrada y salida.

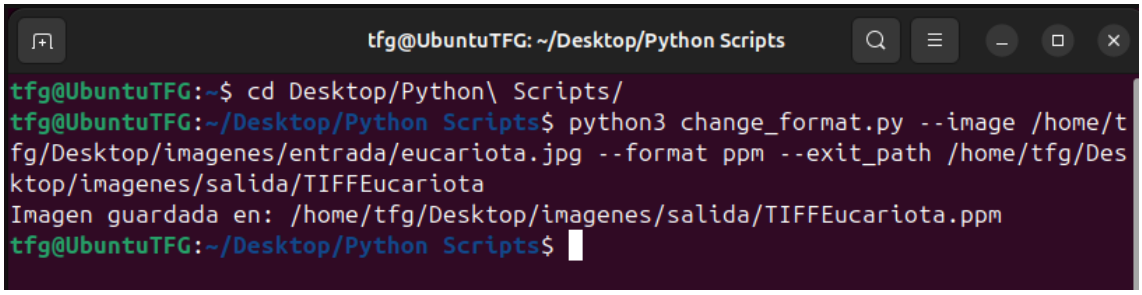

Ilustración 7: Ejecución en terminal de change_format.py.

La ejecución del script generó correctamente la imagen convertida, preservando tanto la resolución como el contenido visual de la imagen original, validando así el correcto funcionamiento de la versión independiente del conversor de formatos.

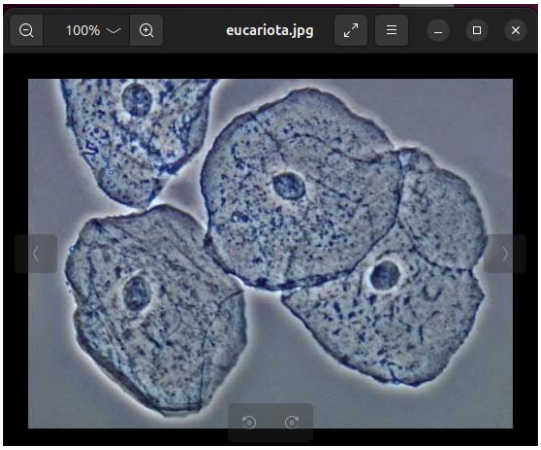

Ilustración 8: Imagen eucariota.jpg original.

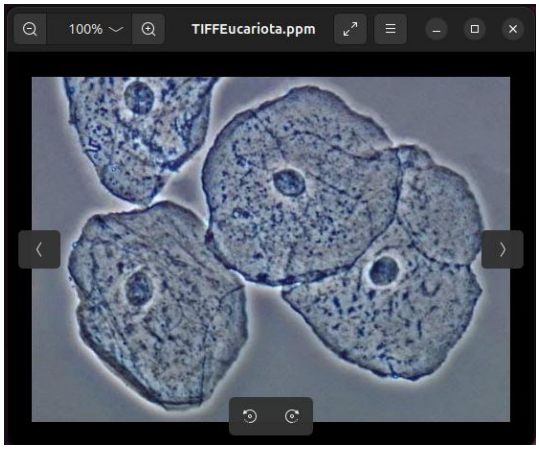

Ilustración 9: Imagen eucariota.jpg convertida a PPM.

En todos los casos, tanto para las ejecuciones interactivas como para las no interactivas, las imágenes resultantes fueron almacenadas correctamente en las rutas de destino indicadas, manteniendo inalteradas tanto la calidad de imagen como los atributos gráficos. Las pruebas de conversión han confirmado la estabilidad y funcionalidad de los procedimientos.

4.2 Pruebas con la binarización

Los scripts de binarización desarrollados permiten transformar imágenes a escala de grises para después hacerlo a imágenes binarias, según un umbral establecido por el usuario. Para validar esta funcionalidad se han realizado distintas pruebas empleando tanto el script desarrollado en Scheme como las versiones independientes de GIMP implementadas en Python con OpenCV, verificando su correcto funcionamiento.

En primer lugar, se probó la ejecución del script “Binarize.scm” desde la interfaz gráfica de GIMP. El usuario proporciona como entrada la imagen a procesar, el directorio de salida, el umbral y el nombre final.

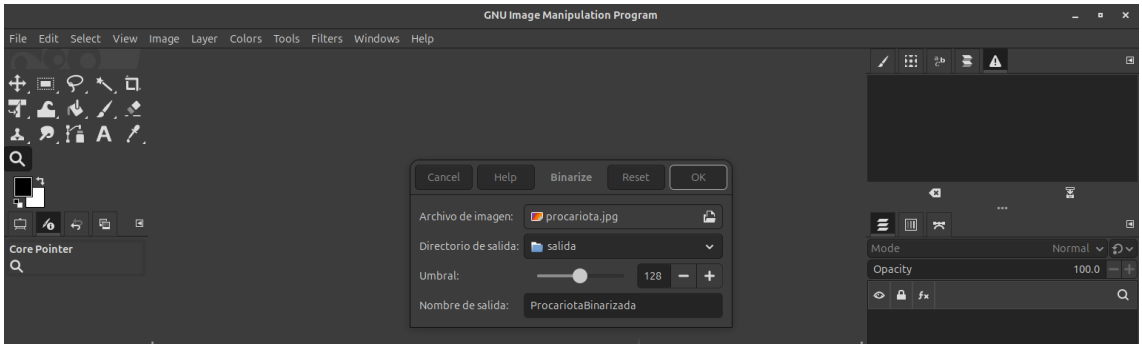

Ilustración 10: Ejecución de Binarize.scm desde GIMP.

En las pruebas se comprueba como el script transforma las imágenes a blanco y negro de acuerdo con la intensidad de los píxeles. Las estructuras principales de las imágenes son resaltadas correctamente según el nivel de corte especificado, observándose un comportamiento estable en imágenes con distintos niveles de contraste y detalle.

Ilustración 11: Imagen procariota.jpg binarizada.

Posteriormente, se validó el funcionamiento del mismo script ejecutándolo desde la terminal, utilizando el modo batch de GIMP bajo Flatpak. El comando empleado fue el siguiente.

Ilustración 12: Ejecución de Binarize.scm desde la terminal.

La imagen fue correctamente procesada de forma automatizada, almacenando el resultado binarizado en la ubicación de destino sin intervención manual, confirmando la viabilidad del procedimiento para ser utilizado dentro de flujos de procesamiento por lotes.

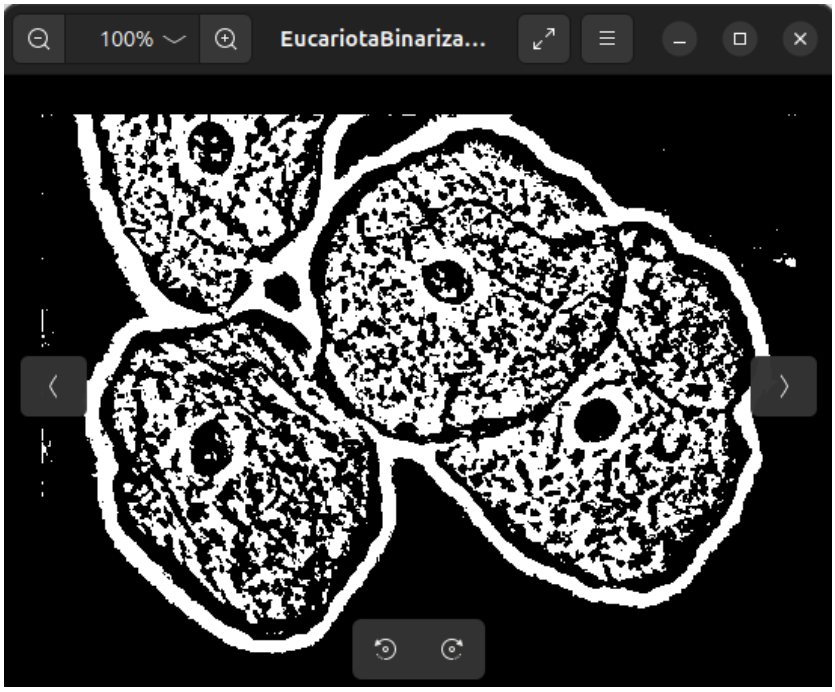

Ilustración 13: Imagen eucariota.jpg binarizada.

Adicionalmente, se validó el script externo desarrollado en Python mediante OpenCV, "binarize_py.py", ejecutado íntegramente desde la terminal sin dependencia de GIMP.

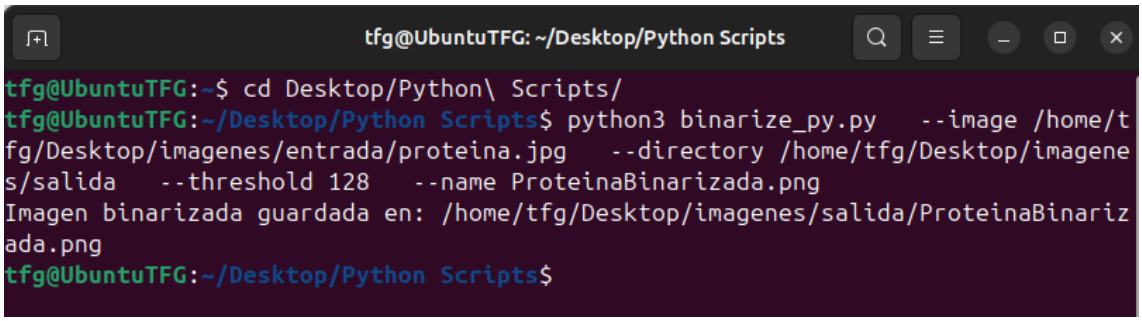

Ilustración 14: Ejecución de binarize_py.py desde la terminal.

La imagen fue correctamente cargada, procesada y binarizada según el umbral proporcionado, generando el archivo de salida especificado. El script se comportó de forma estable ofreciendo resultados completamente equivalentes a los obtenidos mediante los scripts integrados dentro de GIMP.

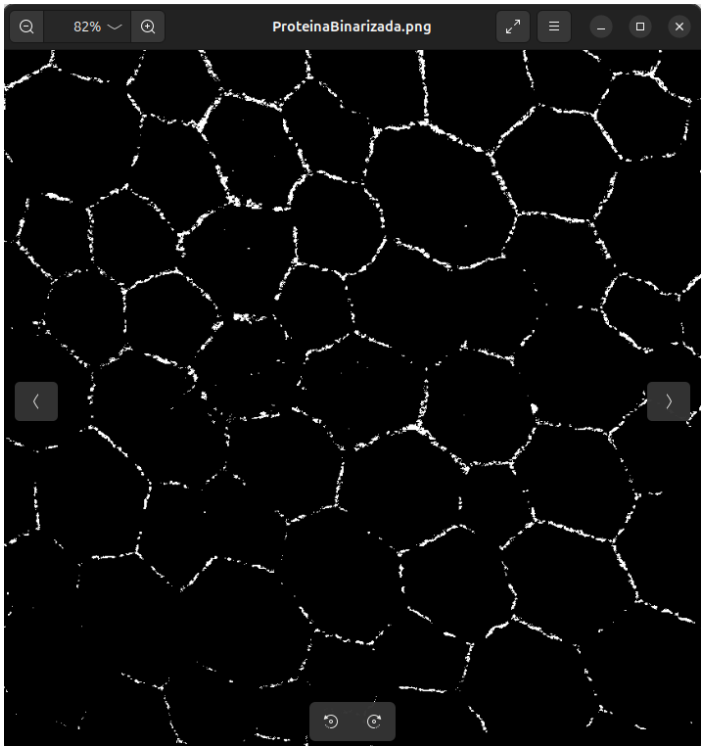

Ilustración 15: Imagen procariota.jpg binarizada.

En todos los casos, tanto en ejecución interactiva como no interactiva, las imágenes resultantes muestran correctamente el efecto de binarización deseado, dividiendo las imágenes en dos niveles de intensidad según el umbral indicado. Las pruebas confirman tanto el correcto funcionamiento de los procedimientos desarrollados y su robustez frente a diferentes imágenes, como su correcto almacenamiento en los directorios especificados sin errores, conservando el formato y la estructura esperados.

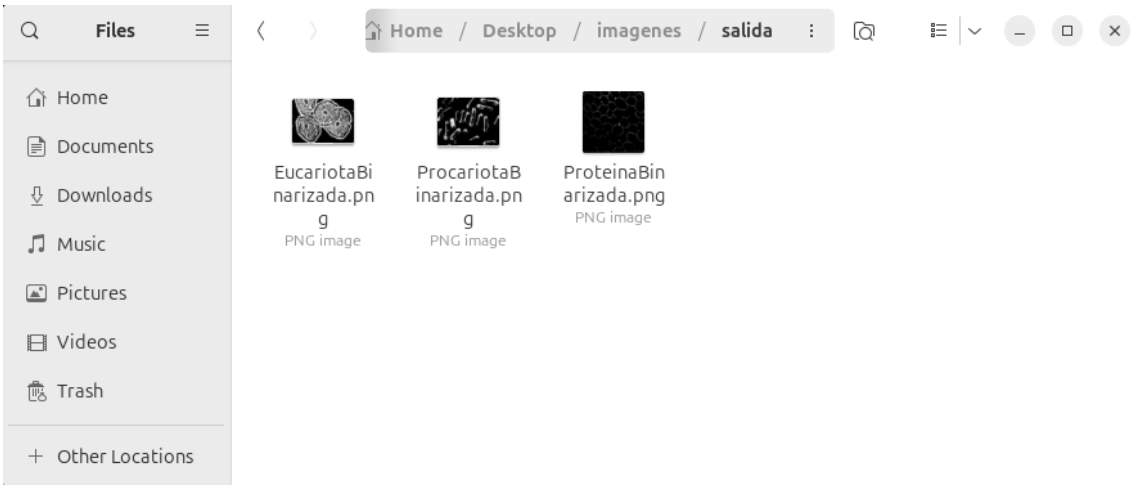

Ilustración 16: Ruta de salida con imágenes binarizadas.

4.3 Pruebas con la adición de texto

La funcionalidad de adición automática de texto permite insertar anotaciones sobre las imágenes con un control total sobre el texto, su posición, tamaño, fuente, color y opacidad. Como en anteriores capítulos se ha comentado, esta funcionalidad se ha desarrollado en tres variantes distintas, adaptadas tanto al entorno de GIMP como a entornos automatizados externos, permitiendo cubrir todos los escenarios de uso previstos en el proyecto.

La primera prueba se realizó ejecutando el script desarrollado en Scheme, "AddText.scm", ejecutado desde la interfaz gráfica de GIMP.

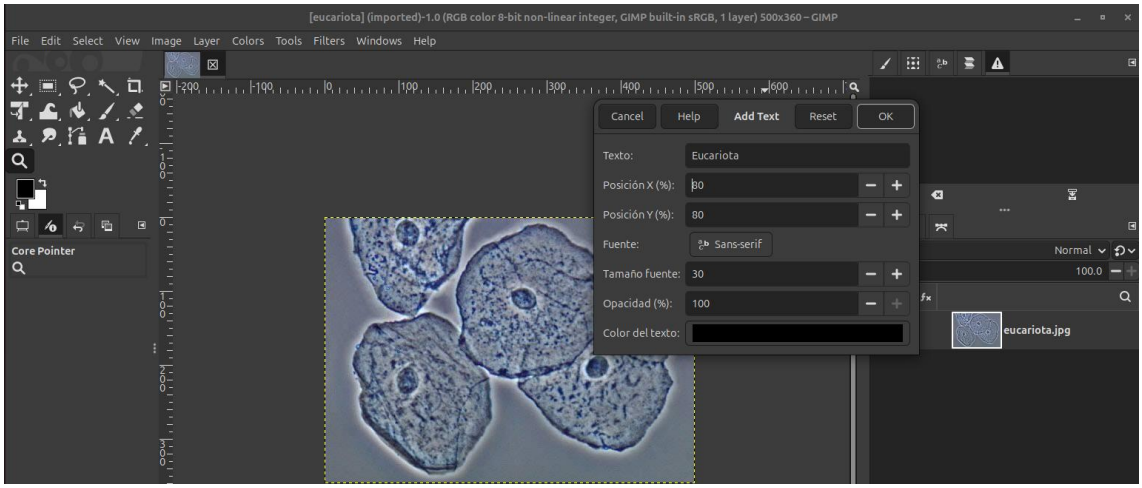

Ilustración 17: Ejecución de AddText.scm desde GIMP.

El script, tras calcular las posiciones absolutas en función del tamaño de la imagen, creó una nueva capa de texto independiente donde insertó el texto. Gracias a este enfoque, la imagen quedó completamente intacta, pudiéndose activar, ocultar o modificar la capa de texto por separado.

Ilustración 18: Imagen eucariota.jpg con texto añadido.

La prueba demostró que el script en Scheme inserta correctamente el texto centrado en las posiciones calculadas, respeta la fuente de texto, el tamaño de

esta, y aplica tanto el color como la opacidad de forma precisa, ofreciendo un total control sobre la estética del texto.

La segunda variante fue probada mediante el plugin desarrollado en Python-Fu, “add_text_plugin.py”, específicamente adaptado a la arquitectura de GIMP 3.0 (basada en GObject y gi.repository). El plugin fue ejecutado desde la interfaz del programa, donde el usuario interactuó con un cuadro de diálogo GTK para introducir el texto y la posición.

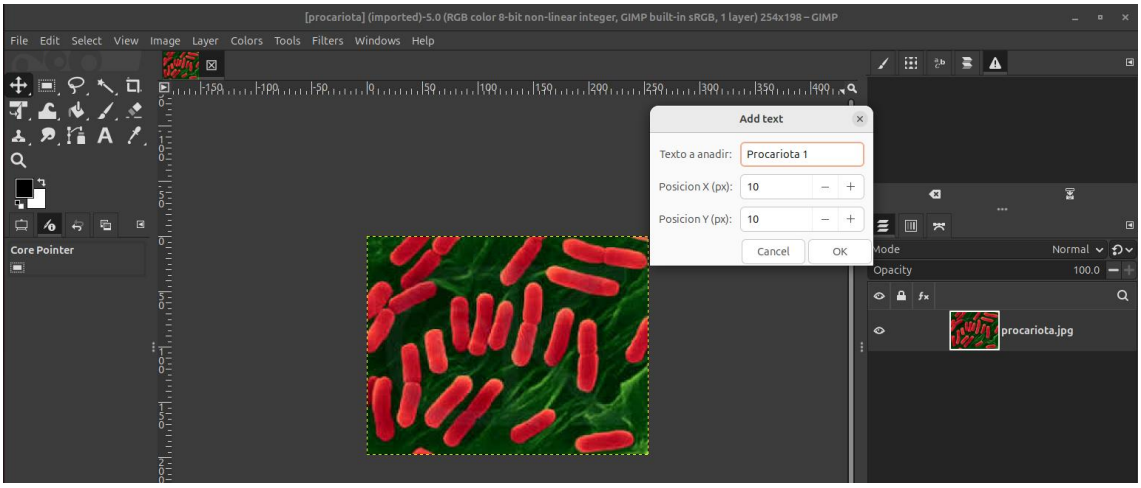

Ilustración 19: Ejecución de add_text_plugin.py desde GIMP.

Al igual que en el caso anterior, el texto se insertó en una nueva capa de texto dentro de la imagen, permitiendo su edición posterior de forma totalmente independiente. Como se puede observar en la siguiente imagen, la nueva capa aparece en la lista de capas, manteniendo el flujo habitual de trabajo por capas propio de GIMP.

Ilustración 20: Imagen procariota.jpg con texto añadido.

La prueba confirmó que la integración del plugin en GIMP es completa, tanto a nivel funcional como de experiencia de usuario, mostrando estabilidad y precisión en la inserción del texto.

Por último, se probó el script desarrollado externamente en Python puro con OpenCV, “add_text.py”, orientado a la ejecución automatizada desde la terminal. En esta versión, el texto se inserta directamente sobre los píxeles de la imagen cargada, sin la creación de capas ya que OpenCV no gestiona capas nativas como en GIMP.

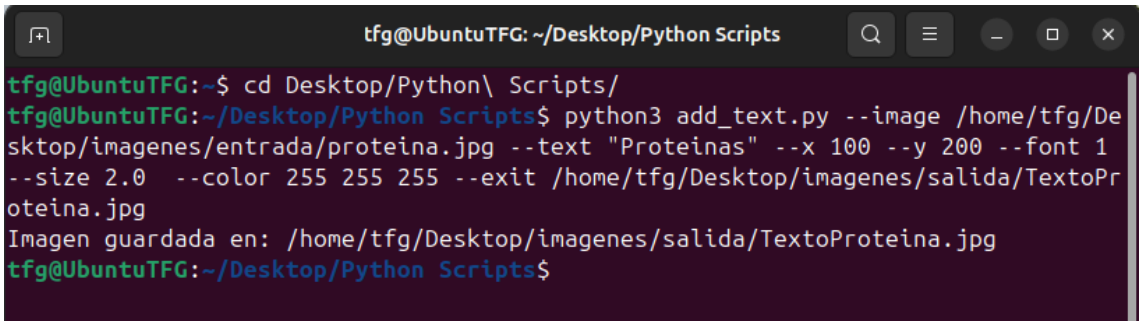

Ilustración 21: Ejecución de add_text.py desde la terminal.

Aunque técnicamente en este caso el texto queda embebido directamente sobre la imagen, esta solución es altamente útil para flujos completamente automatizados donde no se requiere edición posterior, y a pesar de no disponer de capas como en los casos anteriores, el resultado visual es equivalente para los casos de uso donde no es necesario preservar la separación entre imagen y texto.

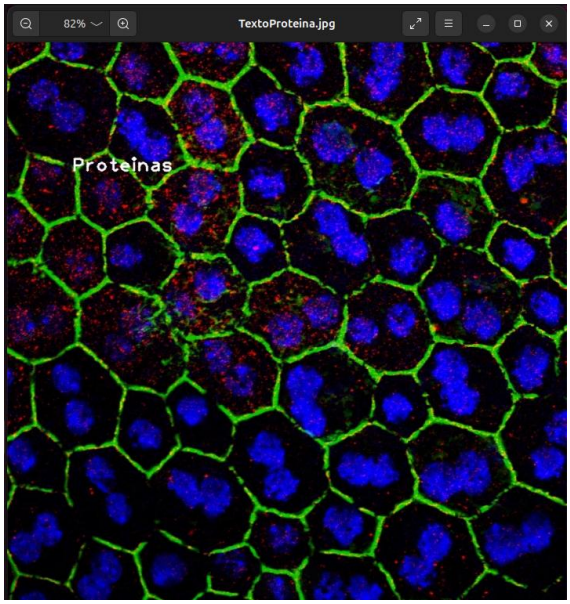

Ilustración 22: Imagen proteina.jpg binarizada.

Tras la ejecución del script, la imagen resultante se guardó correctamente en la ruta de salida especificada por el parámetro “`--exit`” y se verificó que el archivo de salida contenía la imagen con el texto insertado en la posición y con los atributos definidos.

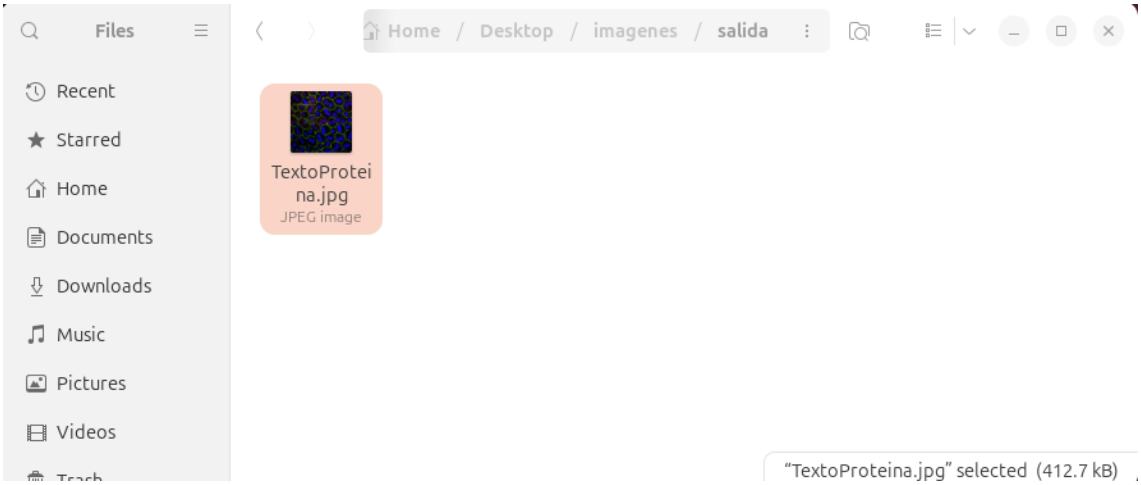

Ilustración 23: Ruta de salida con imagen binarizada.

En conjunto, las pruebas realizadas han demostrado la robustez de los tres procedimientos de inserción de texto desarrollados. El uso de capas en los scripts integrados dentro de GIMP ofrece gran flexibilidad para tareas manuales y semiautomáticas, mientras que la variante en Python con OpenCV es óptima para tareas de procesamiento masivo donde se prioriza la automatización frente a la edición posterior.

4.4 Pruebas con la expansión de una selección

La funcionalidad de expansión de una selección permite aumentar de forma proporcional un área seleccionada dentro de una imagen. Esta operación requiere siempre partir de una selección activa previa, por lo cual solo ha sido implementada en modo interactivo desde dentro de GIMP, mediante Script-Fu.

Para validar esta funcionalidad, se realizaron las pruebas en la interfaz de GIMP. En primer lugar, se abrió la imagen de prueba sobre la cual se realizó una selección manual mediante las herramientas de selección que ofrece GIMP. A partir de dicha selección activa, se ejecutó el script "Expansion.scm", proporcionando como parámetro de entrada el factor de expansión deseado.

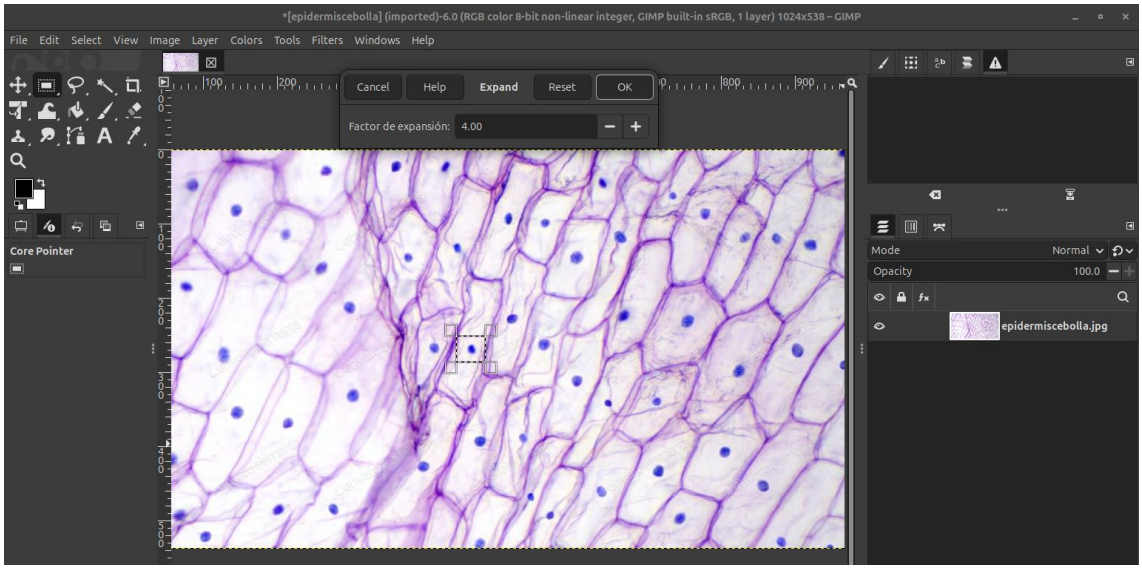

Ilustración 24: Ejecución de Expand.scm desde GIMP sobre la imagen epidermiscebolla.jpg.

El script, tras verificar la existencia de una selección activa, generó internamente una copia de la capa original, aplicó una máscara basada en la selección y procedió a escalar la región seleccionada de forma proporcional. El resultado obtenido muestra como la selección es ampliada manteniendo su forma. Gracias a la aplicación de la interpolación cúbica configurada en el script, la calidad visual del contenido ampliado se mantiene estable. Además de mostrar la imagen resultante tras la expansión, se ha incluido una imagen adicional con la zona seleccionada a mayor nivel de zoom para observar con mayor claridad la diferencia de la resolución y de la precisión del escalado aplicado por el script frente a la imagen original.

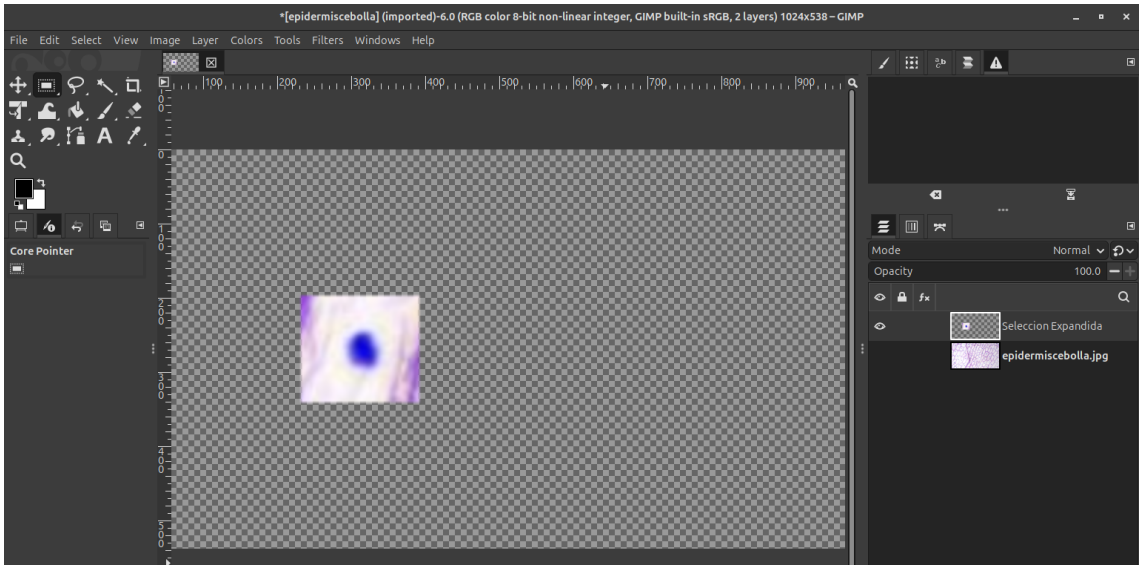

Ilustración 25: Selección sobre epidermiscebolla.jpg expandida.

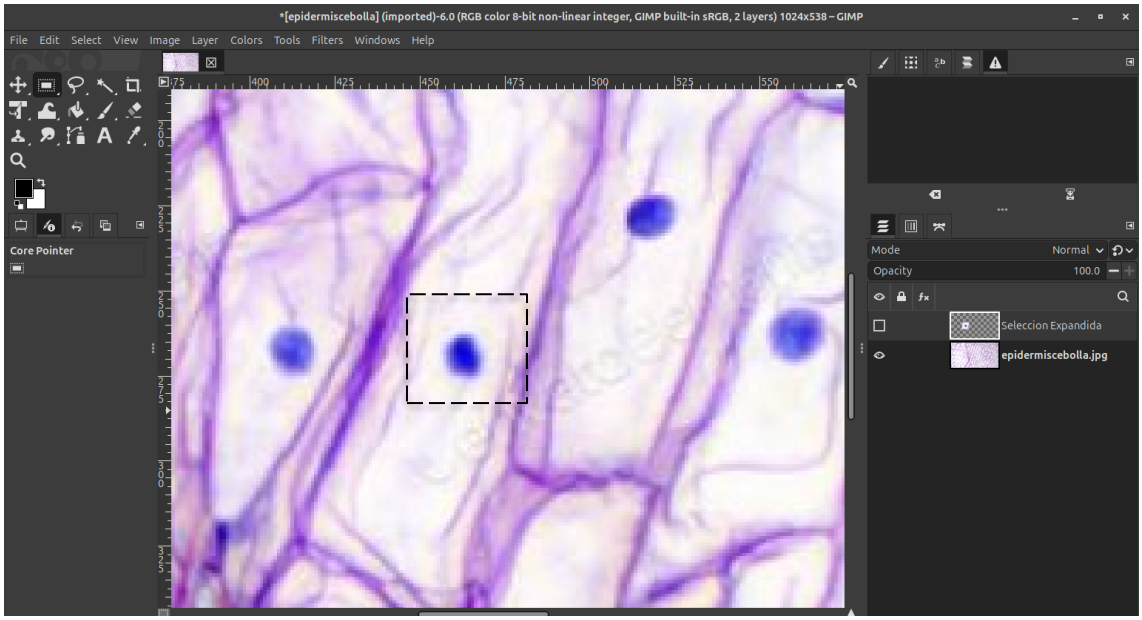

Ilustración 26: Zoom sobre la imagen epidermiscebolla.jpg original.

Adicionalmente, el script inserta la región expandida sobre una nueva capa dentro de la imagen. Esta decisión de diseño es importante, ya que permite al usuario disponer de la región expandida como capa independiente, facilitando su posterior edición, eliminación o modificación sin afectar a la imagen original. La existencia de la nueva capa permite, por ejemplo, aplicar filtros adicionales únicamente sobre la zona expandida o comparar visualmente la diferencia de escala respecto a la selección inicial.

En todas las pruebas realizadas, el script ha gestionado correctamente la selección activa y ha generado las capas ampliadas con precisión, manteniendo tanto la integridad visual como la estructura del trabajo por capas. El procedimiento ha resultado ser completamente estable, cumpliendo con los objetivos de expandir regiones seleccionadas dentro de la imagen de forma automatizada pero controlada por el usuario.

4.5 Pruebas con el resaltado de regiones

La funcionalidad de resaltado de regiones permite identificar y destacar de forma visual aquellas zonas de la imagen que presentan un color similar al de un píxel de referencia, dentro de un margen y de una tolerancia especificados por el usuario. Al igual que la expansión de selección, este procedimiento requiere interacción directa del usuario para seleccionar el píxel de referencia dentro de la imagen. Por este motivo, ha sido implementado exclusivamente en modo interactivo en Scheme.

En la primera prueba, el usuario seleccionó manualmente las coordenadas del píxel de referencia, y a partir de ese punto, el script recorrió el área definida y comparó el color de cada píxel con el color del píxel de referencia, considerando la tolerancia cromática especificada. En esta prueba el modo de funcionamiento

seleccionado fue el de “Color original”, que copia cada píxel similar manteniendo, como su nombre indica, su color original.

Durante la ejecución del script, en la ventana de diálogo donde se introducen los parámetros, es posible observar que los campos “Color resaltado” y “Opacidad” siguen estando visibles y modificables, aunque el modo escogido sea el de “Color original”. No obstante, aunque estos valores puedan ser modificados por el usuario, no afectan al resultado mientras esté activo este modo, ya que el script ignora estos parámetros y conserva los colores originales de cada píxel seleccionado. Este comportamiento se debe a limitaciones propias de la API de GIMP 3.0 ya que el sistema actual de definición de parámetros no permite condicionar la visibilidad o bloqueo de campos en función de los valores introducidos en otros parámetros. Por tanto, todos los argumentos definidos aparecen siempre activos en la interfaz, aunque el modo seleccionado no los necesite.

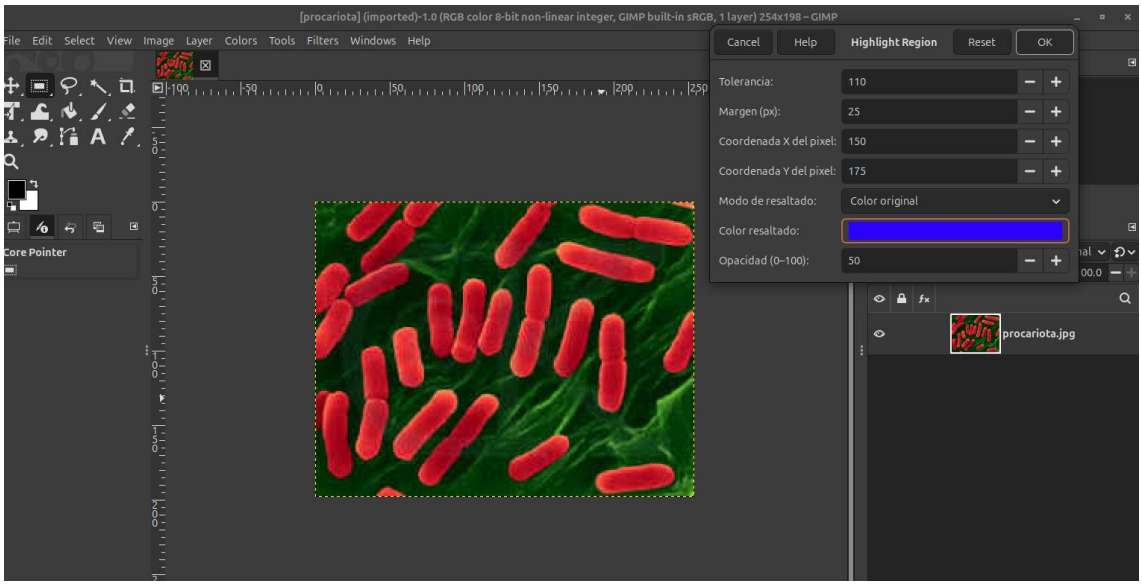

Ilustración 27: Ejecución de RegionHighlight.scm desde GIMP en modo “Color original”.

La nueva capa generada fue insertada de forma independiente dentro de la imagen, manteniendo intacta la capa original. Este diseño por capas facilita ediciones, análisis o tratamientos adicionales sobre la región resaltada, sin afectar al resto de la imagen original.

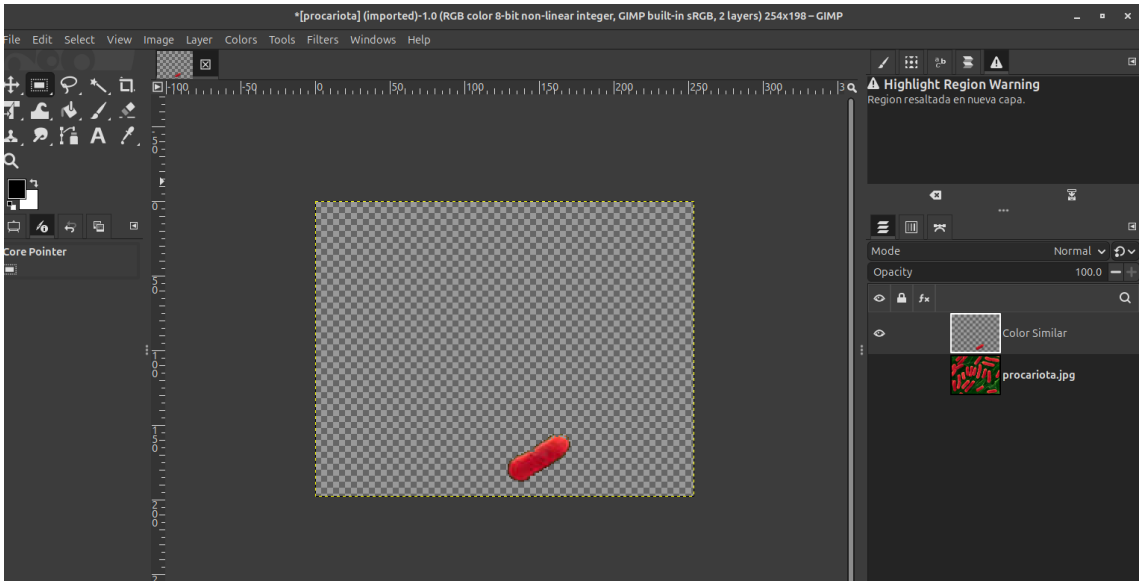

Ilustración 28: Resaltado de célula procariota en distinta capa.

En la segunda prueba, se evaluó el funcionamiento del script en modo “Color resaltado”. En este caso, una vez identificados los píxeles similares, el script sustituyó su color por un color y una opacidad especificadas por el usuario. Este modo resulta especialmente útil cuando el objetivo es enfatizar visualmente la región de interés respecto al resto de la imagen.

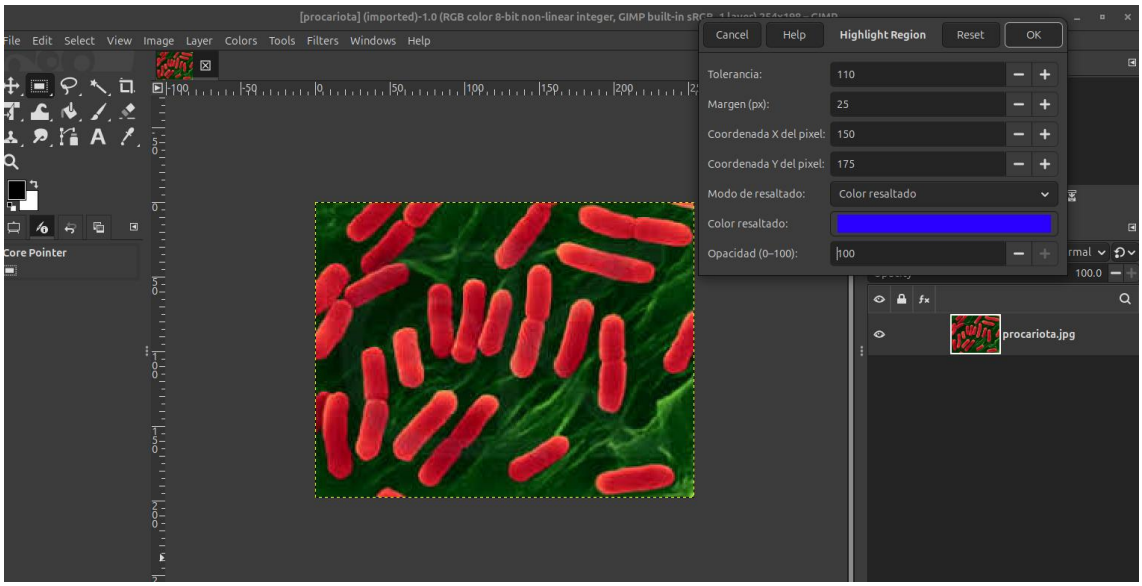

Ilustración 29: Ejecución de RegionHighlight.scm desde GIMP en modo “Color resaltado”.

El uso de un color de contraste y una opacidad controlada permite destacar la zona seleccionada de forma clara sin ocultar el fondo original. Además, como en el caso anterior, la capa generada fue insertada como una nueva capa independiente, permitiendo gestionar el resaltado de forma no destructiva y flexible en el entorno de trabajo de GIMP.

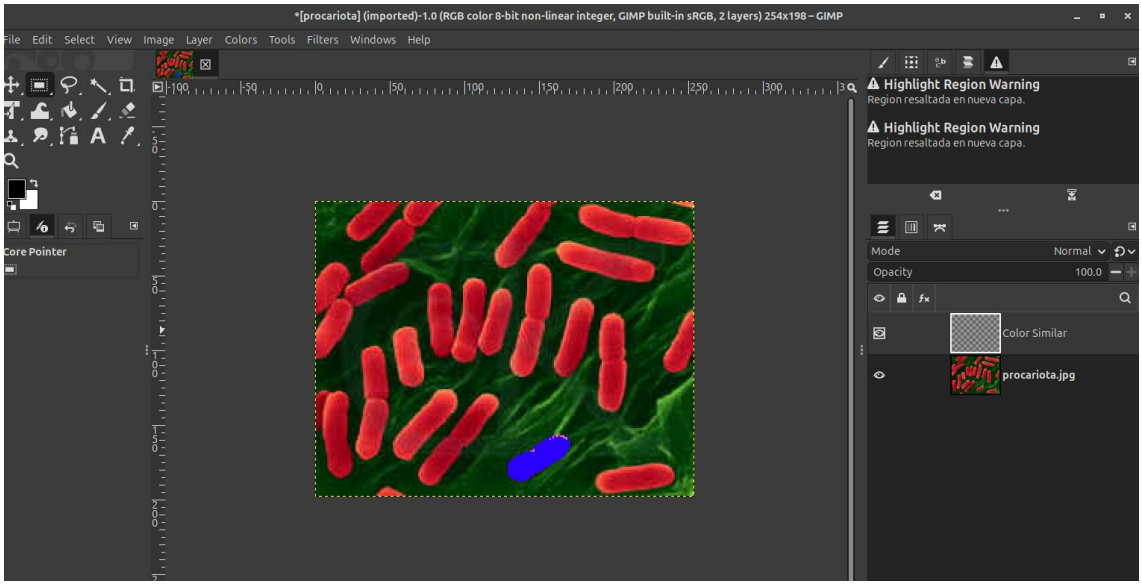

Ilustración 30: Resaltado a color de célula procariota sobre imagen original.

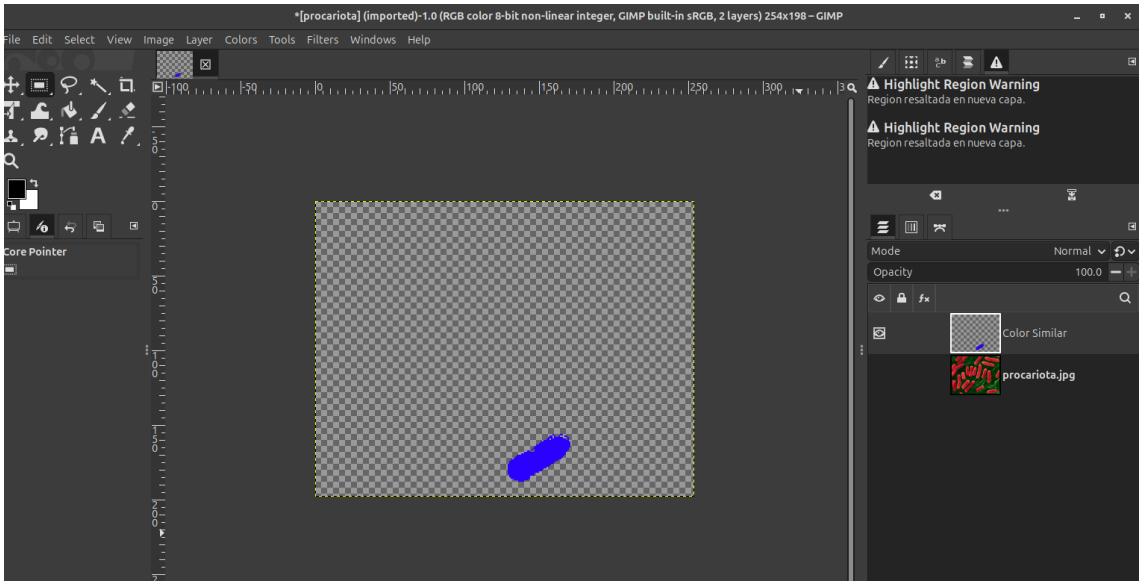

Ilustración 31: Resaltado a color de célula procariota en distinta capa.

En todas las pruebas realizadas en ambos modos, el procedimiento ha identificado correctamente las regiones cromáticamente similares al píxel de referencia, aplicando el resaltado de forma precisa y estable. La incorporación de las capas de salida como nuevas capas independientes dentro de la imagen ha permitido conservar la imagen original y facilitar la manipulación posterior de las zonas resaltadas.

5 Aplicaciones prácticas de las funcionalidades

Uno de los objetivos de este proyecto era demostrar que la automatización en GIMP puede trascender la prueba de concepto y convertirse en una herramienta fiable dentro de flujos de trabajo reales. Para ello se implementaron cinco funcionalidades, con sus respectivas variantes, que cubren necesidades muy diferentes pero igualmente frecuentes en entornos científicos, industriales o creativos.

La conversión de formatos de imágenes se diseñó tanto como filtro de menú en la interfaz de GIMP como comando de batch. De esta forma resulta posible normalizar colecciones completas de archivos con una llamada en la terminal, pero también permite convertir una imagen puntual sin abandonar el entorno gráfico. Esta doble vía es especialmente útil cuando un proyecto combina fases manuales de revisión con fases masivas de procesado, un ejemplo de ello podría ser el siguiente, un investigador revisa y corrige un par de muestras en la interfaz, y una vez validados los parámetros, lanza el mismo el procedimiento sobre cientos o miles de imágenes desde la terminal. A cambio de unos segundos de ejecución, se evita la dispersión de formatos y se garantiza que las imágenes resultantes serán aceptadas por los diversos programas de destino, ya sean editores de maquetación, sistemas de visión artificial, repositorios web u otros.

La binarización por umbral sigue una filosofía parecida. En la interfaz de GIMP la rutina aparece como un filtro sencillo que permite experimentar visualmente con distintos valores de corte, al terminar, el resultado se guarda en una imagen distinta, lo que permite ajustar la opacidad o mezclar la máscara con la imagen original sin destruirla. Esta misma lógica también está disponible como script no interactivo, una línea de terminal bastaría para alimentar un reconocedor de caracteres, cuantificar colonias bacterianas o inspeccionar fotografías industriales, porque esta funcionalidad lee la imagen, calcula la máscara y produce un archivo binarizado listo para su consumo por otras aplicaciones. De este modo se conserva la agilidad del ajuste manual, pero se habilita el procesamiento por lotes característico de laboratorios o líneas de producción.

En cuanto a la inserción automática de texto, en este proyecto se ofrecen tres caminos distintos. Los scripts de Script-Fu y Python-Fu crean la anotación sobre una capa de texto nueva, de modo que el usuario pueda recolocarla o cambiarle la fuente incluso después de haber aplicado otros ajustes a la imagen. Por otra parte, el script externo en Python con OpenCV incrusta el texto directamente sobre la imagen, lo que lo hace útil cuando se necesita rapidez y no es relevante conservar la capa editable. Esa flexibilidad cubre desde la creación puntual de material docente, donde interesa poder reescribir el enunciado, hasta la estampación automática de fechas, firmas digitales o marcas de agua en miles de fotografías.

La expansión de una selección activa es una función eminente visual que solo tiene sentido dentro de la sesión interactiva. La estrategia de este script es preservar la imagen original, y permitir comparaciones rápidas entre el detalle ampliado y la región de partida, o si es necesario, aplicar filtros específicos solo sobre la zona expandida. Por ejemplo, en el ámbito editorial, se podría traducir en recortes listos para revistas o pósteres científicos, o en ámbitos de inspección de defectos podría facilitar al ingeniero acercarse a la soldadura sin perder la referencia de su contexto.

Por último, el resaltado cromático de regiones aprovecha la potencia de las capas para aislar los píxeles de un tono determinado. Para las dos modalidades que se han implementado en este script, y gracias a sus cualidades no destructivas, un analista podría encender o apagar la capa, modificar la opacidad global o cambiar el modo de fusión para enfatizar más o menos la zona de interés. Aunque el procedimiento se invoca desde la interfaz, el resultado podría ser exportado a ficheros independientes y alimentar herramientas de teledetección, inspección farmacéutica o restauración artística.

En conjunto, estas funcionalidades muestran cómo un mismo núcleo de código puede ofrecer tanto ensayo visual inmediato desde dentro de la interfaz, como la ejecución masiva y desatendida desde la terminal, y cuando procede, también la organización en capas que mantiene cada paso reversible. Esa combinación convierte a GIMP en una pieza versátil que encaja por igual en flujos artesanales, laboratorios de investigación o cadenas de producción plenamente automatizadas.

6 Conclusión y trabajos futuros

El punto de partida de este Trabajo de Fin de Grado era comprobar si GIMP 3.0, todavía en fase temprana de adopción, podía asumir un papel estable como motor de automatización en entornos técnicos y científicos. A lo largo del proyecto se han desarrollado y validado varios procedimientos que abarcan desde la conversión de formatos hasta la binarización, la inserción de texto, la expansión selectiva y el resaltado cromático de regiones. El resultado confirma que, pese a la juventud de su ecosistema, la nueva versión del programa ofrece la robustez necesaria para integrarse con solvencia en flujos de trabajo mixtos, donde conviven tareas manuales de inspección y macros de procesado por lotes.

Uno de los hallazgos más significativos es la coexistencia práctica de Scheme y Python dentro de esta última versión de GIMP. Scheme ha demostrado ser la opción más estable cuando se trabaja dentro del contenedor Flatpak, porque sus procedimientos tradicionales continúan estando disponibles, además de la amplia documentación que sigue siendo válida y la ejecución sin restricciones de sandbox. Python, por otro lado, ha aportado el puente con bibliotecas de visión artificial como OpenCV cuando se ejecuta fuera de GIMP, o con la nueva arquitectura implementada cuando el contexto requiere integración directa en la interfaz. En conjunto, el proyecto demuestra que la elección entre ambos lenguajes no es excluyente, sino que permite combinar la solidez de los procedimientos clásicos con la versatilidad de un ecosistema científico actual.

En relación con los objetivos formulados al comienzo del proyecto, puede afirmarse que todos se han alcanzado de manera satisfactoria. En cuanto al estudio de GIMP y de sus posibilidades de extensión mediante scripts, el análisis de la nueva arquitectura de GIMP 3.0 ha confirmado la viabilidad de automatizar flujos técnicos. Respecto a la selección de las funcionalidades a considerar, se cubrió eligiendo la conversión de formatos de imagen, la binarización, la adición de texto, la expansión de selecciones y el resaltado de regiones. Para la implementación de dichas funcionalidades en scripts Scheme y Python, se desarrollaron procedimientos solidos en Scheme, plenamente compatibles con el sandbox, y scripts en Python capaces de invocar librerías externas como OpenCV. Finalmente, con la realización de ejemplos descriptivos del código y su documentación asociada, se logró con una guía paso a paso, capturas ilustrativas y comentarios en línea que facilitan la replicación y el aprendizaje. En conjunto, los cuatro objetivos iniciales se han cumplido de forma íntegra y proporcionan una base robusta para futuras ampliaciones.

En cuanto a líneas de desarrollo futuro, podríamos considerar la refactorización de los procedimientos hacia equivalentes GEGL nativos tan pronto como la API de scripting los exponga de manera estable. Esto permitiría, entre otras mejoras, aprovechar el cálculo en coma flotante y la gestión de color HDR sin efectuar conversiones intermedias. También se podría considerar el empaquetado del conjunto de scripts como una extensión oficial para GIMP, acompañándolo de un asistente gráfico que permita al usuario final ajustar los parámetros sin editar la línea de comandos, esta interfaz podría construirse en GTK o mediante un web frontend que dialogue con GIMP por D-Bus. Del mismo modo, resulta prometedora la integración con flujos de integración continua, como por ejemplo sincronizando los scripts desde GitHub o GitLab, facilitando así la validación de la calidad de imágenes científicas conforme se van generando los datos experimentales.

Como posibles trabajos futuros podríamos encontrar la exploración de técnicas de aprendizaje automático. Un ejemplo de ello consistiría en reemplazar el umbral fijo de la binarización por un modelo de segmentación entrenado con herramientas como PyTorch, ejecutado dentro de Flatpak gracias a la futura compatibilidad de GPU. De la misma manera, la detección cromática podría enriquecerse con un clustering automático para ganar precisión en campos como la restauración artística y la teledetección. Finalmente, resulta aconsejable estudiar la portabilidad multiplataforma, empaquetar los scripts como una imagen Docker o un instalador para Windows que podrían ampliar su adopción en laboratorios donde Linux no sea el sistema predominante.

En resumen, el proyecto demuestra que GIMP 3.0 es una plataforma viable para la automatización avanzada de imágenes aprovechando la madurez de Scheme y el potencial científico de Python. Las funcionalidades implementadas cubren necesidades concretas y cotidianas, pero también sirven como base extensible para incorporar técnicas emergentes, tanto del ámbito gráfico tradicional como del deep learning. El trabajo concluye, por tanto, en que la apuesta por GIMP 3.0 ha cumplido los objetivos iniciales, y además establece un terreno fértil para mejoras continuas y proyectos de investigación futuros que encuentren en el scripting una herramienta de primer orden.

7 Análisis de Impacto

El trabajo realizado contempla la dimensión técnica y poner de manifiesto la relevancia que el software libre y la automatización mediante scripting pueden alcanzar en distintos ámbitos. En el plano personal, el proyecto ha sido un proceso de aprendizaje integral, requiriendo un aprendizaje previo del lenguaje Scheme, la automatización por terminal y la familiarización con bibliotecas de visión artificial, exigiendo un análisis crítico, una resolución de problemas y una toma de decisiones autónoma que, en conjunto, consolidan la madurez profesional del autor.

En el entorno profesional y empresarial, la posibilidad de convertir formatos, binarizar o anotar en grandes lotes de imágenes sin intervención manual, se traduce en ahorros de tiempo y reducción de errores para laboratorios, estudios de diseño, departamentos de documentación o instituciones docentes. El hecho de disponer de scripts modulables y reutilizables, sin costes de licencia, refuerza la competitividad de organizaciones con recursos limitados y posiciona a quienes dominan estas herramientas como perfiles especialmente valiosos en el mercado laboral.

Respecto al campo social y educativo, el uso de GIMP y la publicación abierta de los scripts contribuyen a democratizar el acceso a la tecnología. En espacios como centros de formación, colectivos comunitarios o particulares con recursos escasos se pueden beneficiar de estas técnicas de procesamiento que, en otras circunstancias, quedarían restringidas a software comercial. De esta forma el proyecto se alinea con la educación de calidad (ODS 4) y la reducción de las desigualdades (ODS 10), favoreciendo que el conocimiento circule sin barreras.

El impacto económico se evidencia en la eliminación de gastos asociados a licencias y en la prolongación del ciclo de vida de los equipos, ya que las herramientas libres permiten trabajar en hardware modesto. Además, la automatización evita la externalización de tareas repetitivas y libera horas de trabajo para actividades de mayor valor añadido, en consonancia con el trabajo decente y el crecimiento económico (ODS 8).

Aunque el efecto medioambiental de un proyecto de software es indirecto, la optimización de procesos se traduce en una reducción del consumo energético y de materiales. Esta eficiencia operativa se relaciona tanto con la producción y consumo responsables (ODS 12), como con la acción por el clima (ODS 13), al minimizar la huella de cada flujo de trabajo gráfico.

En el plano cultural, la accesibilidad a herramientas de edición favorece la creación y preservación del patrimonio visual por parte de comunidades que tradicionalmente han tenido un acceso limitado a los medios digitales. Favorecer la producción y el intercambio abierto de contenidos respalda la sostenibilidad de ciudades y comunidades (ODS 11), y las alianzas para lograr objetivos (ODS 17), al fomentar colaboraciones basadas en conocimiento compartido.

Varias decisiones del proyecto se tomaron deliberadamente para maximizar este impacto positivo, como la elección de GIMP como plataforma principal por su licencia libre, o la preferencia por el uso de Scheme que es más estable y ligero en el programa, o el diseño modular de scripts con parámetros claros y ejecución por terminal, que facilita su adaptación e integración en otros entornos. El hecho de publicar el código y la documentación de forma abierta hace que el trabajo se convierta en un punto de partida que otros usuarios pueden ampliar,

contribuyendo así a una comunidad más rica y a un ecosistema tecnológico verdaderamente sostenible.

8 Bibliografía

[1] IObit, "¿Qué es GIMP y para qué sirve? Descubre la mejor alternativa gratuita a Photoshop," *Blog IObit*, s. f. [En línea]. Disponible: https://blog.iobit.com/es/que-es-gimp-718. IObit Blog

[2] GIMP Documentation Team, "Scripting," *GIMP User Manual* 3.0, s. f. [En línea]. Disponible: https://docs.gimp.org/3.0/en/gimp-scripting.html.

[3] Linux-Terminal.com, "GIMP 3.0 Released! How to install on Ubuntu 24.04/22.04/20.04," 2025. [En línea]. Disponible: https://es.linux-terminal.com/?p=8672. cn.linux-terminal.com

[4] GIMP Documentation Team, "Tutorial: Uso de Script-Fu," *Manual de Usuario de GIMP* 3.0, s. f. [En línea]. Disponible: https://docs.gimp.org/es/gimp-using-script-fu-tutorial.html.

[5] GIMP Documentation Team, "Python-Fu," *GIMP User Manual* 3.0, s. f. [En línea]. Disponible: https://docs.gimp.org/3.0/en/gimp-filters-python-fu.html.

[6] Grupo 6, "9310 U3 — Semana V," Scribd, 2024. [En línea]. Disponible: https://es.scribd.com/document/707425282/9310-U3-Semana-v-Grupo6.

[7] ForoGIMP, "Portal de la comunidad," *ForoGIMP.com*, s. f. [En línea]. Disponible: http://forogimp.com/modules/newbb/.

[8] GIMP Team, "GIMP 3.0 Released," *GIMP News*, 16-mar-2025. [En línea]. Disponible: https://www.gimp.org/news/2025/03/16/gimp-3-0-released/. GIMP

[9] GIMP Documentation Team, "Structure of Dialogs," *GIMP User Manual* 3.0, s. f. [En línea]. Disponible: https://docs.gimp.org/es/gimp-dialogs-structure.html.

[10] Reddit, "What are the pros and cons of Flatpaks?," *r/flatpak* (hilo de discusión), 6-ene-2023. [En línea]. Disponible: https://www.reddit.com/r/flatpak/comments/zx4sp6/what_are_the_pros_and_cons_of_flatpaks/.

[11] segu-info, "Finaliza el soporte de Python 2," *Blog Segu-info*, 1-ene-2020. [En línea]. Disponible: https://blog.segu-info.com.ar/2020/01/finaliza-el-soporte-de-python-2.html.

[12] F. Schoenitzer, "GIMP 3.0 Plugin Ressources," *Schoenitzer.de Blog*, 2025. [En línea]. Disponible: https://schoenitzer.de/blog/2025/Gimp%203.0%20Plugin%20Ressources.html.

[13] GIMP Documentation Team, "Using GEGL for configuration," *GIMP User Manual* 2.8, s. f. [En línea]. Disponible: https://docs.gimp.org/2.8/es/gimp-config-use-gegl.html.

[14] GIMP Developers, "Writing a Python Plug-in for GIMP," *developer.gimp.org*, s. f. [En línea]. Disponible: https://developer.gimp.org/resource/writing-a-plug-in/tutorial-python/.

[15] K. Cozens, "python-fu vs script-fu," *gimp-user* mailing-list archive, 22-mar-2016. [En línea]. Disponible: https://www.gimpusers.com/forums/gimp-user/17994-python-fu-vs-script-fu.

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jun 04 20:23:43 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)