**RESUMEN**

Este Trabajo de Fin de Grado se centra en la extensión de las funcionalidades del editor de imágenes GIMP (GNU Image Manipulation Program) mediante el desarrollo de scripts personalizados, con el objetivo de automatizar tareas frecuentes de procesamiento gráfico. GIMP es una potente herramienta de código abierto que, además de su uso interactivo, permite la creación de extensiones y scripts en distintos lenguajes, lo que posibilita su utilización desde la línea de comandos y su integración en flujos de trabajo automatizados.

El proyecto parte del interés por explorar las capacidades de scripting de GIMP en entornos donde el tratamiento masivo de imágenes resulta esencial, como el análisis científico o técnico. Para ello, se ha trabajado con la versión 3.0 del programa, instalada mediante Flatpak sobre un sistema Linux moderno, lo que ha permitido acceder a un entorno aislado, actualizado y compatible con tecnologías actuales.

El desarrollo se ha orientado a la implementación de scripts utilizando principalmente Scheme, el lenguaje nativo de GIMP a través de Script-Fu, aunque también se han explorado posibilidades con Python-Fu. Las funcionalidades implementadas incluyen la conversión de formatos de imagen, la binarización, la adición de texto y la generación de máscaras automáticas, con especial atención a su ejecución desde terminal. Este enfoque permite aprovechar GIMP como una herramienta versátil y potente, más allá de su uso convencional en edición gráfica.

En conjunto, este trabajo demuestra que GIMP, gracias a su sistema 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 y científico de imágenes.

**ABSTRACT**

This Final Degree Project focuses on extending the functionalities of the image editor GIMP (GNU Image Manipulation Program) through the development of custom scripts, with the goal of automating frequent graphic processing tasks. GIMP is a powerful open-source tool that, in addition to its interactive use, allows the creation of extensions and scripts in various languages, enabling its execution from the command line and integration into automated workflows.

The project stems from an interest in exploring GIMP’s scripting capabilities in environments where large-scale image processing is essential, such as scientific or technical analysis. For this purpose, the work has been carried out using version 3.0 of the program, installed via Flatpak on a modern Linux system. This setup provides access to an isolated, up-to-date environment compatible with current technologies.

Development has focused on implementing scripts mainly using Scheme, GIMP’s native scripting language via Script-Fu, although possibilities with Python-Fu have also been explored. The implemented functionalities include image format conversion, binarization, text addition, and automatic mask generation, with special emphasis on execution from the terminal. This approach allows GIMP to be used as a versatile and powerful tool, beyond its conventional use in graphic editing.

Overall, this project demonstrates that GIMP, thanks to its scripting system, can be transformed into an effective platform for the automation of graphic tasks, with applications ranging from basic editing to technical and scientific image analysis.

**Índice**

1. Introducción
2. Trabajos previos
3. Desarrollo  
    3.1 Entorno de desarrollo  
    3.2 Lenguajes y tecnologías empleadas  
    3.3 Diseño de los scripts  
    3.4 Funcionalidades implementadas  
      3.4.1 Conversión de formatos de imagen  
      3.4.2 Binarización  
      3.4.3 Adición de texto  
      3.4.4 Generación de máscaras y resaltado de regiones  
    3.5 Automatización desde terminal  
    3.6 Problemas encontrados y soluciones
4. Pruebas y resultados
5. Comparativa entre Scheme y Python
6. Conclusiones y trabajo futuro
7. Bibliografía
8. Anexos  
    Anexo A: Código ChangeFormat.scm  
    Anexo B: Código Binarize.scm  
    Anexo C: Código AddText.scm  
    Anexo D: Código agregar\_texto.py  
    Anexo E: Código Resaltar.scm  
    Anexo F: Código RegionHighlight.scm

**1 INTRODUCCION**

En las últimas décadas, el procesamiento digital de imágenes se ha consolidado como una disciplina fundamental en numerosos campos, que abarcan desde la medicina y la biología hasta la ingeniería, la astronomía y el arte digital. A medida que las tecnologías de captura y generación de imágenes han evolucionado, también lo han hecho las herramientas destinadas a su análisis, edición y automatización. En este contexto, el uso de scripts para automatizar tareas repetitivas ha cobrado una importancia creciente, al permitir reducir la carga de trabajo manual, minimizar errores humanos, estandarizar resultados y agilizar procesos que, de otro modo, requerirían una considerable inversión de tiempo y esfuerzo.

Entre las herramientas disponibles, GIMP (GNU Image Manipulation Program) se presenta como una de las opciones de código abierto más robustas y versátiles. Aunque históricamente ha estado vinculado al ámbito de la edición gráfica, su arquitectura extensible y su compatibilidad con lenguajes de scripting como Scheme y Python han favorecido su adopción en entornos académicos y científicos. Gracias a estas capacidades, GIMP permite automatizar operaciones sobre imágenes tanto de forma interactiva como por lotes, lo que lo convierte 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 se centra en la exploración, desarrollo y análisis de técnicas de scripting en GIMP, con el propósito de aprovechar su potencial como herramienta de automatización gráfica. Para ello, se ha optado por utilizar Scheme como lenguaje principal, a través de su interfaz Script-Fu, dada su integración nativa con el programa y su estabilidad en la versión 3.0 de GIMP. Esta versión, instalada mediante el sistema Flatpak en un entorno Linux moderno, proporciona una base técnica adecuada para el desarrollo de scripts reutilizables y ejecutables desde terminal, lo que permite integrarlos fácilmente en flujos de trabajo automatizados.

A lo largo del proyecto se implementan distintos scripts enfocados en tareas comunes dentro del procesamiento de imágenes, como la conversión de formatos, la binarización, la inserción automática de texto o la generación de máscaras aplicadas a imágenes microscópicas. Estos desarrollos no solo persiguen demostrar la viabilidad técnica del scripting en GIMP, sino también ofrecer soluciones prácticas a necesidades reales de automatización. Al mismo tiempo, el trabajo propone una reflexión sobre las capacidades del programa como plataforma extensible, analizando qué tipo de tareas y perfiles de usuario se benefician más de este enfoque: desde quienes buscan herramientas simples para agilizar procesos rutinarios, hasta quienes requieren un control detallado sobre las operaciones mediante programación.

Con todo ello, se busca no solo profundizar en las posibilidades del scripting como mecanismo de extensión funcional de GIMP, sino también valorar críticamente su papel dentro de los entornos técnicos actuales, destacando su aplicabilidad en ámbitos donde la automatización resulta esencial.

**2 TRABAJOS PREVIOS**

El uso del scripting en GIMP ha sido objeto de múltiples iniciativas, publicaciones técnicas y proyectos independientes que han buscado ampliar el alcance del programa más allá de su uso tradicional como editor gráfico. En sus primeras versiones, GIMP incorporó Script-Fu como subsistema nativo para la ejecución de scripts escritos en una variante del lenguaje Scheme. Esta elección respondía a la intención de ofrecer a los usuarios una herramienta compacta, funcional y eficiente, capaz 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 limitado 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, condicionales, bucles y listas. Además, facilitó el uso de bibliotecas externas del ecosistema 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.

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 guías detalladas que explican cómo interactuar con la API, registrar nuevos procedimientos y manipular imágenes directamente desde scripts. Con el lanzamiento de GIMP 3.0, esta documentación se ha actualizado para adaptarse a los nuevos estándares del programa, cuya base tecnológica se apoya en GTK 3. Esta modernización ha mejorado el rendimiento general, la compatibilidad con pantallas de alta resolución y el soporte para lenguajes de programación contemporáneos.

Además, existen múltiples investigaciones y trabajos académicos que han utilizado el scripting en GIMP como herramienta de apoyo para el análisis automatizado de imágenes. En ámbitos como la fluorescencia, la hematología o la microbiología, 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. 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**

**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 basa en su estabilidad, amplia compatibilidad con herramientas de código abierto y facilidad para gestionar entornos aislados de desarrollo. Desde el inicio, se consideró fundamental trabajar con una versión moderna y actualizada del editor GIMP, dado que las versiones anteriores presentan limitaciones importantes en cuanto a compatibilidad con tecnologías recientes, especialmente en lo relativo al scripting en Python.

Inicialmente, se valoró el uso de GIMP 2.10, una versión aún ampliamente adoptada por la comunidad, pero cuyo entorno de scripting se basa en Python 2.7, lo que imposibilita el uso de bibliotecas modernas como OpenCV, NumPy o Pillow, todas ellas dependientes de Python 3. Esta limitación técnica, unida al hecho de que GIMP 3.0 representa la versión más reciente y avanzada del programa, motivó la decisión de optar por esta última. Su instalación mediante Flatpak no solo permite disponer de una versión compatible con Python 3 en sistemas como Ubuntu, sino que además garantiza un entorno aislado, actualizado y alineado con los desarrollos más actuales del proyecto GIMP.

La instalación mediante Flatpak ofrece varias ventajas relevantes: garantiza un entorno aislado del sistema base, evita conflictos de dependencias, facilita la portabilidad del software entre equipos y asegura actualizaciones automáticas con cada nueva versión oficial. No obstante, también introduce ciertas limitaciones, como restricciones de acceso a determinadas rutas del sistema de archivos o dificultades a la hora de utilizar bibliotecas externas desde Python-Fu. Estas limitaciones han condicionado parte del desarrollo, como se detalla en apartados posteriores.

Además, para la creación y edición de los scripts en Scheme se ha utilizado el editor de texto **Geany**, una herramienta ligera y altamente configurable que proporciona resaltado de sintaxis, plegado de código y otras funcionalidades que han facilitado la escritura de scripts de forma estructurada y eficiente.

En resumen, el entorno de desarrollo combina herramientas actualizadas, configuraciones personalizadas para facilitar la automatización, y un enfoque orientado a la portabilidad y estabilidad del sistema, garantizando así las condiciones necesarias para el desarrollo y prueba de scripts en GIMP 3.0.

**3.2 LENGUAJES Y TECNOLOGIAS EMPLEADAS**

El desarrollo de este proyecto se ha basado principalmente en el uso de dos lenguajes de scripting compatibles con GIMP: **Scheme** y **Python**. Ambos han sido considerados desde la fase inicial por sus respectivas ventajas, aunque la práctica y las limitaciones del entorno han llevado a priorizar uno sobre otro.

#### **Scheme (Script-Fu)**

El lenguaje Scheme, utilizado a través del sistema **Script-Fu**, constituye la tecnología principal empleada en este trabajo. Scheme es un lenguaje funcional, compacto y eficiente, que GIMP ha adoptado históricamente como su sistema de scripting nativo. Permite definir procedimientos personalizados que se integran directamente en el menú de GIMP, así como automatizar operaciones sobre imágenes desde la interfaz gráfica o desde la línea de comandos.

Uno de los principales motivos para centrar el desarrollo en Scheme ha sido su **plena integración con GIMP 3.0 bajo Flatpak**, sin necesidad de configuraciones adicionales ni dependencias externas. Su funcionamiento ha demostrado ser estable y fiable, lo que ha permitido desarrollar scripts reutilizables, orientados a tareas como la conversión de formatos, la binarización o la generación de máscaras. Además, Script-Fu facilita el registro de procedimientos, la definición de parámetros interactivos y el control de las capas y canales de una imagen, todo ello sin abandonar el entorno propio de GIMP.

#### **Python**

El lenguaje **Python** también ha sido considerado por su popularidad, legibilidad y potencia, así como por la posibilidad de integrar bibliotecas externas como OpenCV o NumPy. Sin embargo, durante las primeras fases del proyecto se identificaron **importantes limitaciones técnicas** en el uso de Python-Fu bajo la instalación Flatpak de GIMP 3.0. Entre ellas destacan errores de ejecución en la consola, restricciones de acceso a determinadas rutas del sistema y fallos en la carga de bibliotecas externas.

Debido a estas dificultades, el uso de Python se ha restringido a la **creación de scripts autónomos** ejecutables desde terminal, sin depender del entorno interno de GIMP. Estos scripts permiten, por ejemplo, realizar anotaciones de texto sobre imágenes utilizando OpenCV, lo cual complementa las funcionalidades desarrolladas en Scheme sin depender de la integración con el propio GIMP.

En conjunto, el proyecto se ha apoyado en tecnologías abiertas, estables y adaptadas al objetivo de automatizar tareas de procesamiento de imágenes mediante scripting. La elección de Scheme como lenguaje principal ha sido clave para garantizar la compatibilidad con GIMP 3.0 y la ejecución fiable de los scripts tanto en modo interactivo como automatizado.

**3.3 DISENO DE LOS SCRIPTS**

El desarrollo de los scripts en este proyecto no solo ha estado orientado a resolver tareas concretas de procesamiento de imágenes, sino también a establecer un diseño coherente, modular y reutilizable que facilite su integración en distintos flujos de trabajo. Dado que uno de los objetivos principales 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: desde la interfaz gráfica de GIMP y desde la línea de comandos mediante ejecución por lotes.

Para ello, se han seguido una serie de principios de diseño enfocados en la **modularidad**, la **reutilización** y la **parametrización**, buscando que cada script cumpla una función bien definida y pueda ser adaptado con facilidad a nuevas tareas o entornos. Se ha procurado que los parámetros necesarios para su funcionamiento —como rutas de entrada y salida, valores umbral o textos personalizados— puedan ser definidos por el usuario en el momento de la llamada, lo que habilita una ejecución automatizada y flexible sin requerir interacción posterior.

Los scripts han sido implementados en su mayoría utilizando **Script-Fu**, el sistema de scripting de GIMP basado en el lenguaje **Scheme**. Cada script se estructura en tres partes fundamentales:

* **Definición**: se especifica el nombre del procedimiento, los parámetros de entrada requeridos (como rutas de archivos, valores numéricos o cadenas de texto) y los tipos de datos esperados.
* **Lógica funcional**: se describen las operaciones que el script realiza sobre la imagen o la capa activa, tales como duplicar capas, convertir a escala de grises, aplicar umbrales o guardar el resultado en una ubicación determinada.
* **Registro**: mediante la función script-fu-register, el procedimiento queda integrado en el entorno de GIMP, visible bajo una categoría específica del menú (por ejemplo, “Filtros > Custom”), lo que permite su ejecución desde la interfaz gráfica del programa.

Un ejemplo básico de esta estructura puede verse a continuación, donde cada bloque representa un componente esencial de cualquier script Script-Fu:

;; Definición del procedimiento  
(define (nombre-del-procedimiento parametro-texto parametro-numero)  
  
 ;; Lógica funcional (estructura mínima)  
 (begin  
 (gimp-message "El script se ha ejecutado correctamente.")  
 )  
)  
  
;; Registro del procedimiento  
(script-fu-register  
 "nombre-del-procedimiento"  
 "<Image>/Ruta/Del/Menu/Script Generico"  
 "Descripción breve del script"  
 "Autor"  
 "Licencia o institución"  
 "2025"  
 ""  
 SF-STRING "Texto de entrada" ""  
 SF-VALUE "Número de entrada" 0  
)

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

Por último, se ha cuidado especialmente la **organización y legibilidad del código fuente**, incorporando comentarios explicativos y manteniendo una separación clara entre funciones. Esta práctica no solo facilita su comprensión y mantenimiento, sino que también 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 este trabajo y con los principios de desarrollo sostenibles en entornos de procesamiento técnico o científico.

**3.4 FUNCIONALIDADES IMPLEMANTADAS**

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

Las principales tareas automatizadas han sido:

* **Conversión de formato de imagen**: scripts que permiten convertir imágenes entre distintos formatos, como .jpg, .png o .tiff, de forma automatizada y sin intervención manual, útil para normalizar grandes lotes de archivos.
* **Binarización**: scripts que transforman una imagen en escala de grises y aplican un umbral fijo para generar una versión binaria, adaptada a tareas de segmentación o análisis visual en microscopía y otras áreas científicas.
* **Adición de texto**: desarrollo de scripts que permiten insertar texto sobre la imagen de forma programada, útil para anotar, numerar o clasificar imágenes dentro de un conjunto de trabajo.
* **Generación de máscaras y resaltado de regiones**: implementación de procedimientos que permiten crear capas con máscaras o destacar zonas específicas de la imagen, facilitando la visualización de áreas de interés o la preparación de datos para análisis posterior.

En conjunto, estas funcionalidades demuestran la capacidad de GIMP para actuar como una herramienta flexible y automatizable, capaz de adaptarse a flujos de trabajo personalizados. Los scripts desarrollados no solo resuelven tareas específicas, sino que también sirven como base para su extensión futura o su integración en procesos más complejos.

**3.4.1 CONVERSION DE FORMATO DE IMAGEN**

Una de las primeras funcionalidades implementadas ha sido la conversión automática de imágenes entre distintos formatos. Este tipo de tarea es habitual en flujos de trabajo que requieren homogeneizar extensiones de archivo o preparar imágenes para su uso en otras herramientas que solo aceptan ciertos formatos.

El script, escrito en Scheme mediante Script-Fu, carga una imagen desde una ruta especificada, extrae su nombre base, genera una nueva ruta de salida con la extensión deseada y guarda el resultado en el formato especificado. El procedimiento fue diseñado para ejecutarse en modo no interactivo y sin intervención del usuario, lo que permite su uso por lotes desde terminal.

#### **Código del script (resumen)**

scheme

CopyEdit

(define (script-fu-convert-image-format inFilename outFolder outExtension)  
 (let\* ((image (car (gimp-file-load RUN-NONINTERACTIVE inFilename inFilename)))  
 (drawable (car (gimp-image-get-active-layer image)))  
 (basename (car (strbreakup (car (gimp-image-get-name image)) ".")))  
 (outputPath (string-append outFolder "/" basename "." outExtension)))  
 (file-png-save RUN-NONINTERACTIVE image drawable outputPath outputPath 0 9 0 0 0 0 0)  
 (gimp-image-delete image)  
 (gimp-message (string-append "Saved as: " outputPath))))

Este script queda registrado en el menú “Filters > Custom” de GIMP para su uso interactivo, pero su principal utilidad reside en su capacidad de ejecutarse desde la terminal de forma completamente automatizada.

#### **Uso desde terminal**

El script puede ejecutarse desde terminal mediante el siguiente comando:

bash

CopyEdit

flatpak run org.gimp.GIMP --no-interface --batch-interpreter=plug-in-script-fu-eval \  
--batch '(script-fu-convert-image-format "/ruta/imagen.jpg" "/ruta/salida" "png")' \  
--batch '(gimp-quit 0)'

Para simplificar su invocación, se añadió al archivo ~/.bashrc el siguiente alias:

bash

CopyEdit

alias gimp='flatpak run org.gimp.GIMP --no-interface --batch-interpreter=plug-in-script-fu-eval'

Esto permite ejecutar el mismo script de forma más concisa:

bash

CopyEdit

gimp --batch '(script-fu-convert-image-format "/ruta/imagen.jpg" "/ruta/salida" "png")' \  
--batch '(gimp-quit 0)'

#### **Observaciones**

* El script utiliza funciones como strbreakup para extraer el nombre base del archivo, y file-png-save para guardar la imagen en el nuevo formato.
* Actualmente, el guardado se realiza en formato .png, pero puede modificarse fácilmente para soportar otros formatos como .jpg o .tiff, cambiando la función de guardado.
* Su ejecución en modo no interactivo lo hace especialmente útil para integrarse en flujos de trabajo por lotes o scripts externos.

**3.4.2 BINARIZACION**

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

El script desarrollado en Scheme permite aplicar binarización con un umbral fijo sobre una imagen, generando una versión en blanco y negro a partir de la conversión previa a escala de grises. Está diseñado para trabajar tanto de forma interactiva como desde la terminal, y permite ajustar el valor del umbral manualmente.

#### **Código del script (resumen)**

scheme

CopyEdit

(define (script-fu-binarize-from-file inFilename outFolder threshold)  
 (let\* ((image (car (gimp-file-load RUN-NONINTERACTIVE inFilename inFilename)))  
 (drawable (car (gimp-image-get-active-layer image)))  
 (copy-layer (car (gimp-layer-copy drawable TRUE)))  
 (basename (car (strbreakup (car (gimp-image-get-name image)) ".")))  
 (outputPath (string-append outFolder "/" basename "\_bin.png")))  
 (gimp-image-add-layer image copy-layer 0)  
 (gimp-drawable-desaturate copy-layer DESATURATE-LIGHTNESS)  
 (gimp-threshold copy-layer threshold 255)  
 (gimp-displays-flush)  
 (file-png-save RUN-NONINTERACTIVE image copy-layer outputPath outputPath 0 9 0 0 0 0 0)  
 (gimp-image-delete image)  
 (gimp-message (string-append "Binarized image saved at: " outputPath))))

Este procedimiento carga una imagen desde el disco, duplica la capa activa, la convierte a escala de grises utilizando el modo de “luminosidad” (DESATURATE-LIGHTNESS) y aplica un umbral que define qué píxeles se convertirán en blancos o negros. El resultado se guarda automáticamente como una nueva imagen binarizada.

#### **Uso desde terminal**

Puede ejecutarse desde terminal con el siguiente comando:

bash

CopyEdit

gimp --batch '(script-fu-binarize-from-file "/ruta/imagen.jpg" "/ruta/salida" 128)' --batch '(gimp-quit 0)'

o, si se prefiere la llamada directa sin alias:

bash

CopyEdit

flatpak run org.gimp.GIMP --no-interface --batch-interpreter=plug-in-script-fu-eval \  
--batch '(script-fu-binarize-from-file "/ruta/imagen.jpg" "/ruta/salida" 128)' \  
--batch '(gimp-quit 0)'

El valor del umbral (128 en el ejemplo anterior) puede ajustarse según las necesidades del análisis. Cuanto más bajo sea el valor, mayor cantidad de píxeles se considerarán blancos.

#### **Observaciones**

* La duplicación de la capa garantiza que la imagen original no se modifique durante el proceso.
* El resultado se guarda como un archivo independiente con sufijo \_bin para evitar sobrescribir la imagen de entrada.
* El script está preparado para integrarse en flujos de procesamiento masivo, funcionando correctamente en modo no interactivo.

### **3.4.3 Adición de texto**

Añadir texto sobre imágenes de forma programada es una funcionalidad útil en numerosos contextos, desde la anotación científica hasta la generación automatizada de gráficos informativos. En este proyecto se han desarrollado dos aproximaciones para esta tarea: una basada en Script-Fu (Scheme) y otra mediante Python y OpenCV.

#### **3.4.3.1 Script en Scheme (Script-Fu)**

El script en Scheme fue concebido para integrarse directamente en GIMP y permitir la adición de texto sobre una imagen mediante la creación de una nueva capa de texto. Se utilizaron funciones como gimp-text-layer-new y gimp-image-add-layer para insertar texto con estilo, tamaño y posición definidos por el usuario.

##### **Fragmento de código relevante**

scheme

CopyEdit

(define (script-fu-add-text img drawable texto fuente tamano x y)  
 (let\* ((text-layer (car (gimp-text-layer-new img texto fuente tamano PIXELS TRUE)))  
 (color (list 0.0 0.0 0.0))) ; texto negro  
 (gimp-image-add-layer img text-layer -1)  
 (gimp-layer-set-offsets text-layer x y)  
 (gimp-displays-flush)))

Este código crea una nueva capa de texto sobre la imagen img con el contenido especificado en texto, usando la fuente y tamaño indicados, y posiciona la capa en las coordenadas (x, y).

##### **Limitaciones**

Durante las pruebas en GIMP 3.0 bajo Flatpak se detectaron errores sistemáticos al invocar funciones como gimp-text-layer-new o gimp-text-fontname, lo que impidió validar completamente el funcionamiento del script. Aunque está estructuralmente correcto, la ejecución en este entorno concreto resultó inestable o directamente fallida.

##### **Estado actual**

El script está implementado y registrado en el menú de GIMP, pero su ejecución puede fallar según la configuración del entorno. Por esta razón, se planteó una solución alternativa fuera de GIMP utilizando Python.

#### **3.4.3.2 Script en Python (OpenCV)**

Como alternativa robusta, se desarrolló un script en Python utilizando la biblioteca OpenCV, pensado para ejecutarse de forma autónoma desde la terminal. Esta solución no depende de GIMP, lo que permite una mayor estabilidad y portabilidad.

##### **Fragmento de código clave**

python

CopyEdit

cv2.putText(imagen, args.texto, (args.x, args.y), font, args.tamano, color, 2, cv2.LINE\_AA)  
cv2.imwrite(args.salida, imagen)

Este fragmento aplica el texto sobre la imagen cargada, utilizando los parámetros recibidos por línea de comandos. El resultado se guarda en la ruta indicada.

##### **Ejecución desde terminal**

El script se llama así:

bash

CopyEdit

python3 agregar\_texto.py \  
 --imagen /ruta/imagen.jpg \  
 --texto "Ejemplo" \  
 --x 100 --y 100 \  
 --fuente 0 \  
 --tamano 1.5 \  
 --color 0 0 0 \  
 --salida /ruta/imagen\_texto.png

##### **Ventajas**

* Compatible con cualquier sistema que tenga Python 3 y OpenCV.
* Permite personalización completa del texto (fuente, tamaño, color, posición).
* Muy adecuado para entornos donde se requiere procesar imágenes sin depender de interfaces gráficas.

##### **Aplicabilidad**

Este script ha demostrado ser una alternativa eficaz y estable frente a las limitaciones del entorno Script-Fu en GIMP 3.0. Además, permite extender la funcionalidad del proyecto más allá de GIMP, integrando operaciones de anotación en flujos externos.

### **3.4.4 Generación de máscaras y resaltado de regiones**

El resaltado de zonas específicas dentro de una imagen es una funcionalidad clave en tareas de análisis visual, especialmente en campos como la microscopía, donde puede ser necesario destacar estructuras de interés para su posterior estudio o documentación. Para ello, se han desarrollado dos scripts complementarios que permiten generar máscaras y resaltar regiones de manera automática: uno basado en la selección activa (Resaltar.scm) y otro mediante coordenadas predefinidas (RegionHighlight.scm).

#### **3.4.4.1 Script de resaltado a partir de selección activa**

El script Resaltar.scm permite aplicar un efecto de oscurecimiento o desenfoque al área fuera de una selección activa, generando así un efecto visual que resalta únicamente la región seleccionada. Esta funcionalidad es útil cuando se trabaja con imágenes complejas y se desea centrar la atención en una zona específica.

##### **Fragmento de código relevante**

scheme

CopyEdit

(define (script-fu-resaltar img drawable)  
 (let\* ((layer-copy (car (gimp-layer-copy drawable TRUE)))  
 (mask (car (gimp-layer-create-mask layer-copy ADD-SELECTION-MASK))))  
 (gimp-image-add-layer img layer-copy -1)  
 (gimp-layer-add-mask layer-copy mask)  
 (gimp-layer-set-mode layer-copy MULTIPLY-MODE)  
 (gimp-displays-flush)))

Este código crea una copia de la capa activa, genera una máscara basada en la selección del usuario, y aplica el modo de fusión “Multiply” para oscurecer el fondo y resaltar la zona seleccionada.

##### **Uso desde GIMP**

Este script está pensado para usarse de forma interactiva, una vez que el usuario ha realizado manualmente una selección en la imagen. Actualmente no incluye parámetros para automatizar la selección desde terminal, por lo que su ejecución no interactiva está limitada.

##### **Observaciones**

* Ideal para presentaciones, informes o documentación visual.
* Puede adaptarse para generar distintos estilos de énfasis (inversión de máscara, desenfoque, opacidad).

#### **3.4.4.2 Script de resaltado por coordenadas predefinidas**

El script RegionHighlight.scm fue diseñado para automatizar completamente el proceso de resaltado, definiendo internamente una región rectangular de interés. Esta aproximación es útil cuando se conoce de antemano la zona a destacar o se trabaja con imágenes homogéneas.

##### **Fragmento de código relevante**

scheme

CopyEdit

(define (script-fu-region-highlight img drawable)  
 (let\* ((layer-copy (car (gimp-layer-copy drawable TRUE)))  
 (x 100) (y 100) (width 200) (height 150)  
 (mask (car (gimp-layer-create-mask layer-copy ADD-BLACK-MASK))))  
 (gimp-image-add-layer img layer-copy -1)  
 (gimp-layer-add-mask layer-copy mask)  
 (gimp-rect-select img x y width height CHANNEL-OP-REPLACE FALSE 0)  
 (gimp-selection-invert img)  
 (gimp-edit-bucket-fill mask FG-BUCKET-FILL NORMAL-MODE 100 0 FALSE 0 0)  
 (gimp-layer-set-mode layer-copy MULTIPLY-MODE)  
 (gimp-displays-flush)))

Este script realiza los siguientes pasos:

1. Crea una copia de la capa activa.
2. Aplica una máscara negra sobre toda la imagen.
3. Define una selección rectangular e invierte la selección.
4. Rellena la zona invertida para oscurecer todo excepto la región deseada.

##### **Uso desde terminal**

El script puede ejecutarse desde la terminal con:

bash

CopyEdit

gimp --batch '(script-fu-region-highlight RUN-NONINTERACTIVE)' --batch '(gimp-quit 0)'

Para que funcione correctamente en modo no interactivo, debe aplicarse a una imagen cargada desde archivo y registrarse un procedimiento con parámetros adecuados.

##### **Ventajas**

* Permite automatizar el resaltado en lotes sin necesidad de interacción manual.
* Es modificable para adaptar el área resaltada, color del fondo o tipo de máscara.
* Útil para destacar coordenadas constantes en imágenes estructuradas (por ejemplo, cultivos celulares en placas con posiciones estándar).

### **3.5 Automatización desde terminal**

Una de las ventajas más destacadas del scripting en GIMP es la posibilidad de ejecutar procedimientos en modo no interactivo, es decir, sin necesidad de abrir la interfaz gráfica del programa. Esto resulta especialmente útil para procesar grandes volúmenes de imágenes de forma automática, integrando los scripts en flujos de trabajo por lotes, scripts de shell o incluso sistemas cron programados.

Durante el desarrollo de este proyecto, se ha trabajado con GIMP 3.0 instalado mediante Flatpak, lo que implica ciertas particularidades a la hora de lanzar el programa desde terminal. Para ejecutar scripts de Scheme (Script-Fu) fuera del entorno gráfico, es necesario utilizar el modo --no-interface, el intérprete plug-in-script-fu-eval y la sintaxis adecuada para pasar los parámetros.

#### **Estructura general del comando**

bash

CopyEdit

flatpak run org.gimp.GIMP --no-interface \  
 --batch-interpreter=plug-in-script-fu-eval \  
 --batch '(script-fu-nombre-script "param1" "param2" ...)' \  
 --batch '(gimp-quit 0)'

#### **Facilitación mediante alias**

Para simplificar este proceso y evitar repetir la estructura completa en cada llamada, se ha añadido un alias personalizado al archivo ~/.bashrc:

bash

CopyEdit

alias gimp='flatpak run org.gimp.GIMP --no-interface --batch-interpreter=plug-in-script-fu-eval'

Con este alias activo, cualquier script puede ejecutarse directamente con:

bash

CopyEdit

gimp --batch '(script-fu-nombre-script "param1" "param2" ...)' --batch '(gimp-quit 0)'

#### **Ejemplo práctico**

Para convertir una imagen .jpg a .png mediante el script de conversión, basta con ejecutar:

bash

CopyEdit

gimp --batch '(script-fu-convert-image-format "/ruta/imagen.jpg" "/ruta/salida" "png")' --batch '(gimp-quit 0)'

Del mismo modo, para aplicar binarización con un umbral de 128:

bash

CopyEdit

gimp --batch '(script-fu-binarize-from-file "/ruta/imagen.jpg" "/ruta/salida" 128)' --batch '(gimp-quit 0)'

#### **Ventajas de este enfoque**

* **Velocidad**: no se necesita abrir la interfaz gráfica de GIMP.
* **Automatización**: permite integrar GIMP en scripts bash, procesos por lotes o tareas programadas.
* **Reproducibilidad**: se pueden definir procesos con parámetros exactos, sin depender de acciones manuales.

#### **Limitaciones**

* La terminal debe tener acceso a Flatpak y al alias correctamente configurado.
* No todos los scripts de GIMP están preparados por defecto para ser ejecutados en modo no interactivo; es necesario diseñarlos específicamente para ello.

### **3.6 Problemas encontrados y soluciones**

Durante el desarrollo del proyecto se han identificado diversos problemas técnicos y operativos relacionados con el entorno de trabajo, las herramientas utilizadas y las funcionalidades implementadas. Este apartado recoge los principales obstáculos encontrados, así como las decisiones adoptadas para resolverlos o mitigarlos de forma eficaz.

#### **Inestabilidad de Python-Fu en GIMP 3.0 (Flatpak)**

Uno de los problemas más relevantes fue la **limitada funcionalidad de Python-Fu bajo GIMP 3.0 instalado con Flatpak**. Aunque esta versión permite el uso de Python 3 (a diferencia de GIMP 2.10, que solo soporta Python 2.7), en la práctica se detectaron fallos importantes:

* Errores al ejecutar scripts desde la consola de GIMP.
* Imposibilidad de cargar bibliotecas externas como OpenCV desde Python-Fu.
* Restricciones de acceso a rutas del sistema debido al aislamiento del entorno Flatpak.

**Solución adoptada**: Se decidió **reducir el papel de Python-Fu en el proyecto** y **priorizar el desarrollo en Scheme**, aprovechando su integración nativa y estabilidad en GIMP 3.0. Python se mantuvo únicamente como herramienta externa, utilizada para tareas complementarias fuera de GIMP (como el script de adición de texto con OpenCV).

#### **Fallo de funciones de texto en Script-Fu**

Durante el desarrollo del script AddText.scm, se comprobó que funciones como gimp-text-layer-new o gimp-text-fontname provocaban errores o simplemente no funcionaban al ejecutarse bajo GIMP 3.0 Flatpak. Esto imposibilitó la creación estable de capas de texto desde Script-Fu.

**Solución adoptada**: Ante la imposibilidad de resolver el fallo desde Scheme, se optó por implementar una alternativa en **Python autónomo** (fuera de GIMP), que permite insertar texto sobre imágenes utilizando la biblioteca OpenCV y ejecutar el proceso completamente desde terminal.

#### **Limitaciones del scripting en modo no interactivo**

Aunque Script-Fu está diseñado para permitir la ejecución en modo no interactivo desde terminal, no todos los scripts ofrecían compatibilidad inmediata con este modo. Algunos procedimientos requerían ajustes para evitar diálogos interactivos o dependencias del entorno gráfico.

**Solución adoptada**: Todos los scripts fueron diseñados o modificados específicamente para aceptar parámetros desde terminal, procesar imágenes sin intervención del usuario, y finalizar automáticamente con gimp-quit, permitiendo su uso en flujos por lotes.

#### **Falta de documentación actualizada para GIMP 3.0**

Durante el proyecto se detectó que parte de la documentación oficial y ejemplos de scripting estaban desactualizados o aún referidos a versiones anteriores del programa (GIMP 2.x). Esto generó incertidumbre al probar ciertas funciones que habían sido modificadas o eliminadas en GIMP 3.0.

**Solución adoptada**: Se recurrió a la exploración práctica mediante prueba y error, revisión de foros especializados y análisis directo del comportamiento del programa. Además, se reforzó el uso de funciones bien documentadas y compatibles para asegurar estabilidad.

#### **Limitaciones del entorno Flatpak**

Aunque Flatpak garantiza un entorno limpio y aislado, también impone restricciones importantes sobre rutas de acceso, permisos y disponibilidad de ciertos recursos del sistema. Estas limitaciones afectaron especialmente al uso de Python dentro de GIMP.

**Solución adoptada**: Para mitigar este efecto, se usaron rutas explícitas y se configuró un **alias en el archivo ~/.bashrc** que permite lanzar GIMP en modo no interactivo con el intérprete adecuado. Esto simplificó enormemente las pruebas y la automatización desde terminal.

En conjunto, los problemas encontrados han servido para afianzar la elección de Scheme como lenguaje principal para scripting dentro de GIMP 3.0, y para establecer estrategias prácticas que aseguran la ejecución estable de los scripts tanto en entorno gráfico como desde línea de comandos.

### **4. Pruebas y resultados**

Una vez implementadas las funcionalidades descritas en el capítulo anterior, se procedió a realizar una serie de pruebas para verificar su correcto funcionamiento tanto en modo interactivo desde la interfaz de GIMP como en modo no interactivo mediante terminal. Las pruebas se enfocaron en evaluar la estabilidad de los scripts, la fidelidad de los resultados obtenidos, la facilidad de uso en diferentes contextos y la capacidad de los procedimientos para integrarse en flujos automatizados.

#### **4.1 Pruebas en modo interactivo**

Las pruebas en modo interactivo se realizaron desde el entorno gráfico de GIMP 3.0. Todos los scripts desarrollados en Script-Fu fueron correctamente registrados en el menú “Filtros > Custom” y se comportaron como se esperaba en tareas como conversión de formatos, binarización y resaltado de regiones. Cada script fue probado con diferentes imágenes de entrada, variando los parámetros definidos para comprobar su flexibilidad.

El único caso con resultados inconsistentes fue el script de adición de texto en Scheme, que no pudo ejecutarse de forma estable debido a errores relacionados con las funciones de creación de capas de texto, probablemente por incompatibilidades con la versión actual de GIMP o su entorno Flatpak. Este comportamiento justificó la implementación de una alternativa funcional en Python, tal como se detalla en el apartado correspondiente.

#### **4.2 Pruebas en modo no interactivo (terminal)**

Uno de los objetivos principales del proyecto fue permitir la ejecución de los scripts desde terminal, sin intervención manual. Para ello, se validó el comportamiento de cada script invocado mediante comandos con --batch, utilizando el alias personalizado gimp configurado en el archivo ~/.bashrc.

A continuación, se recogen ejemplos reales de comandos utilizados y sus resultados:

**Conversión de formato**

bash

CopyEdit

gimp --batch '(script-fu-convert-image-format "/home/tfg/Desktop/img.jpg" "/home/tfg/Desktop/salida" "png")' --batch '(gimp-quit 0)'

**Resultado esperado**: generación automática del archivo img.png en la ruta de salida, con nombre base conservado.

**Binarización**

bash

CopyEdit

gimp --batch '(script-fu-binarize-from-file "/home/tfg/Desktop/img.jpg" "/home/tfg/Desktop/salida" 128)' --batch '(gimp-quit 0)'

**Resultado esperado**: imagen binarizada correctamente con sufijo \_bin.png.

**Resaltado por coordenadas**

bash

CopyEdit

gimp --batch '(script-fu-region-highlight "/home/tfg/Desktop/img.jpg")' --batch '(gimp-quit 0)'

**Resultado esperado**: creación de una nueva capa oscurecida con una región rectangular claramente destacada.

**Anotación con texto en Python**

bash

CopyEdit

python3 agregar\_texto.py --imagen "/home/tfg/Desktop/img.jpg" --texto "Muestra A" --x 50 --y 50 --fuente 0 --tamano 1.2 --color 255 0 0 --salida "/home/tfg/Desktop/img\_texto.png"

**Resultado esperado**: imagen con texto “Muestra A” en rojo, insertado en las coordenadas dadas.

#### **4.3 Resultados y evaluación**

Los scripts desarrollados se han comportado de manera estable, cumpliendo con los objetivos funcionales definidos para cada uno. En particular, destaca:

* La solidez de los scripts desarrollados en Scheme, que han demostrado ser plenamente compatibles con GIMP 3.0 bajo Flatpak.
* La capacidad de ejecutar los procedimientos desde terminal en modo totalmente automatizado.
* La viabilidad de complementar GIMP con herramientas externas, como el script en Python con OpenCV, para suplir limitaciones internas del entorno.

En conjunto, los resultados validan el enfoque adoptado en el proyecto, evidenciando que GIMP puede ser convertido en una herramienta de procesamiento automatizado de imágenes cuando se combina adecuadamente con scripting y ejecución por lotes.

### **5. Comparativa entre Scheme y Python**

Uno de los objetivos iniciales del proyecto fue explorar las posibilidades de scripting en GIMP utilizando dos lenguajes distintos: Scheme, a través de la interfaz Script-Fu, y Python, mediante el sistema Python-Fu o scripts externos con bibliotecas como OpenCV. Esta sección recoge una comparativa entre ambos enfoques, basada tanto en documentación técnica como en la experiencia práctica adquirida durante el desarrollo.

#### **5.1 Integración con GIMP**

**Scheme** está integrado de forma nativa en GIMP desde sus primeras versiones. Su sistema Script-Fu permite definir procedimientos que se registran fácilmente en los menús del programa y que pueden ejecutarse tanto desde la interfaz gráfica como desde terminal en modo no interactivo. Esta integración directa garantiza un funcionamiento estable, especialmente en la versión 3.0 de GIMP instalada mediante Flatpak.

**Python**, por su parte, depende del módulo Python-Fu para integrarse en GIMP. Aunque esta integración es más potente a nivel de lenguaje, en la práctica ha demostrado ser inestable en GIMP 3.0 bajo Flatpak, con errores frecuentes, dificultades para cargar bibliotecas externas y restricciones de acceso a determinadas rutas del sistema.

**Conclusión**: Scheme ofrece una integración más fiable y directa con GIMP, mientras que Python presenta limitaciones técnicas importantes en el entorno actual.

#### **5.2 Facilidad de desarrollo**

**Scheme** es un lenguaje funcional con una sintaxis poco familiar para muchos usuarios. Requiere comprender estructuras como listas anidadas, ausencia de estructuras de control tradicionales y uso extensivo de funciones predefinidas. Esta curva de aprendizaje puede ser una barrera inicial, aunque sus scripts tienden a ser compactos y eficientes una vez dominados.

**Python**, en cambio, es un lenguaje ampliamente conocido, con una sintaxis clara y una enorme comunidad de usuarios. Permite trabajar con estructuras de datos complejas, bucles, condicionales y bibliotecas externas de forma más cómoda, lo que lo convierte en una herramienta poderosa y accesible para desarrolladores de distintos niveles.

**Conclusión**: Python es más fácil de aprender y usar para la mayoría de usuarios, pero Scheme es suficiente y adecuado cuando se limita al scripting interno de GIMP.

#### **5.3 Potencial y extensibilidad**

**Scheme**, limitado a las funciones expuestas por GIMP a través de Script-Fu, es ideal para automatizar tareas dentro del propio programa, pero no permite acceso directo a bibliotecas externas o interacción con el sistema operativo más allá del contexto de GIMP.

**Python**, especialmente cuando se usa fuera de GIMP, permite aprovechar todo su ecosistema, incluyendo bibliotecas como OpenCV, NumPy, PIL o Pandas. Esto lo convierte en una opción ideal para tareas avanzadas de análisis de imagen, procesamiento científico o integración con otras herramientas.

**Conclusión**: Python tiene un potencial mucho mayor cuando se usa como herramienta externa. Scheme, en cambio, es más limitado pero suficiente para tareas internas a GIMP.

#### **5.4 Compatibilidad con GIMP 3.0 y Flatpak**

Durante este proyecto se utilizó GIMP 3.0 instalado mediante Flatpak. En este entorno, Scheme ha demostrado ser plenamente compatible y estable. Python, en cambio, presentó errores críticos y dificultades técnicas no resueltas, que han limitado su uso a scripts externos.

**Conclusión**: En el entorno actual (GIMP 3.0 + Flatpak), Scheme es la opción más estable y recomendada para scripting integrado.

**Resumen de la comparativa**:

|  |  |  |
| --- | --- | --- |
| **Criterio** | **Scheme (Script-Fu)** | **Python (Python-Fu / externo)** |
| Integración con GIMP | Nativa y estable | Limitada y problemática en Flatpak |
| Facilidad de desarrollo | Sintaxis compleja | Legible, versátil y extendida |
| Acceso a bibliotecas externas | No | Sí (cuando se usa fuera de GIMP) |
| Ejecución desde terminal | Sí (sin interfaz) | Sí (cuando se usa como script) |
| Compatibilidad con Flatpak | Total | Parcial / inestable |

### **6. Comparativa entre GIMP 2.10 y GIMP 3.0**

Durante las fases iniciales del proyecto se valoró trabajar con GIMP 2.10, versión ampliamente utilizada y bien documentada. Sin embargo, tras realizar pruebas con ambas versiones, se identificaron diferencias significativas que motivaron el uso definitivo de GIMP 3.0. A continuación, se presenta una comparativa estructurada entre ambas versiones, con especial atención a sus implicaciones para el desarrollo de scripts.

#### **6.1 Compatibilidad con Python**

GIMP 2.10 utiliza de forma nativa Python 2.7, un lenguaje que ha quedado obsoleto y que no es compatible con bibliotecas modernas como OpenCV o NumPy en sus versiones actuales. Esta limitación supone un obstáculo importante si se desea integrar GIMP con herramientas de análisis científico o flujos de trabajo actuales en Python 3.

GIMP 3.0, en cambio, incorpora soporte para Python 3, lo cual representa una mejora importante desde el punto de vista técnico. No obstante, al instalarse mediante Flatpak (la única vía estable actualmente en Ubuntu 24), se presentan dificultades relevantes: errores de ejecución en Python-Fu, restricciones de acceso a bibliotecas del sistema, y problemas al cargar módulos externos como OpenCV.

**Conclusión**: Aunque GIMP 3.0 ofrece compatibilidad con Python 3, las restricciones impuestas por Flatpak limitan su funcionalidad. GIMP 2.10 es más estable con Python-Fu, pero su dependencia de Python 2.7 lo hace poco viable en contextos modernos.

#### **6.2 Estabilidad del scripting en Scheme (Script-Fu)**

El sistema Script-Fu, basado en Scheme, está presente y operativo en ambas versiones. En GIMP 2.10, los scripts se ejecutan correctamente, pero algunos procedimientos y funciones presentan diferencias de comportamiento según la versión exacta del programa o del sistema operativo.

GIMP 3.0 mantiene la compatibilidad con Script-Fu de forma más coherente, con mejoras en el sistema de menús, en la gestión de capas y en la integración con la arquitectura GTK 3. Durante el desarrollo del proyecto, todos los scripts en Scheme se comportaron de forma estable en GIMP 3.0, tanto desde la interfaz como desde terminal.

**Conclusión**: Script-Fu es más estable y predecible en GIMP 3.0, y resulta plenamente funcional en su entorno Flatpak.

#### **6.3 Modernización y arquitectura del programa**

GIMP 2.10 está construido sobre GTK 2, lo que limita su integración con entornos gráficos modernos, su compatibilidad con pantallas de alta resolución y la evolución de su interfaz. Aunque sigue siendo una versión madura y funcional, su base tecnológica está desactualizada.

GIMP 3.0 introduce una renovación estructural al adoptar GTK 3, lo que mejora la apariencia visual, la respuesta de la interfaz, la modularidad interna y la compatibilidad con tecnologías actuales como Wayland. Estas mejoras lo convierten en una base más sólida para desarrollos futuros.

**Conclusión**: GIMP 3.0 es tecnológicamente superior y mejor preparado para entornos modernos, a pesar de algunos detalles aún en transición.

#### **6.4 Documentación y soporte comunitario**

La documentación de GIMP 2.10 está más consolidada y cuenta con una gran cantidad de ejemplos, tutoriales y foros activos. Esto facilita el aprendizaje y la resolución de problemas comunes, especialmente para usuarios que comienzan a trabajar con scripting.

GIMP 3.0, al ser una versión más reciente, aún está en proceso de consolidar su documentación. Muchas referencias todavía apuntan a funciones antiguas o no actualizadas, y parte de la API está en evolución. Aun así, la comunidad sigue siendo activa y se están generando nuevos recursos constantemente.

**Conclusión**: GIMP 2.10 ofrece mejor documentación en la actualidad, pero GIMP 3.0 está evolucionando rápidamente y representa el futuro de la plataforma.

**Resumen de la comparativa**:

|  |  |  |
| --- | --- | --- |
| **Criterio** | **GIMP 2.10** | **GIMP 3.0** |
| Arquitectura | GTK 2 (obsoleta) | GTK 3 (moderna, compatible con Wayland) |
| Python | Python 2.7 | Python 3 (limitado por Flatpak) |
| Script-Fu (Scheme) | Compatible, algo inconsistente | Estable, plenamente funcional |
| Documentación y soporte | Amplia y consolidada | En expansión, aún incompleta |
| Compatibilidad con sistemas | Alta, pero desactualizada | Alta, preparada para el futuro |

### **7. Análisis del impacto potencial**

Además del desarrollo técnico, este trabajo de fin de grado permite reflexionar sobre el impacto que puede tener en diferentes ámbitos el uso de herramientas de software libre y la automatización de tareas mediante scripting. A continuación se analiza el alcance de los resultados obtenidos en contextos personales, profesionales, sociales, económicos, medioambientales y culturales, así como su vinculación con los Objetivos de Desarrollo Sostenible (ODS) definidos en la Agenda 2030 de Naciones Unidas.

#### **Impacto personal**

A nivel personal, el desarrollo de este proyecto ha supuesto una experiencia de aprendizaje integral. No solo ha permitido profundizar en aspectos técnicos como el lenguaje Scheme, la automatización por terminal o el uso de bibliotecas de procesamiento de imágenes, sino que también ha fomentado habilidades de análisis, resolución de problemas y adaptación a entornos cambiantes. La necesidad de tomar decisiones técnicas con autonomía y justificar metodológicamente cada paso ha contribuido de forma directa al crecimiento como desarrollador y como estudiante.

#### **Impacto en el ámbito profesional y empresarial**

La automatización de tareas repetitivas en entornos gráficos, como la conversión de imágenes, la binarización o la anotación masiva, tiene una aplicación directa en diversos sectores profesionales: desde laboratorios científicos hasta agencias de diseño, pasando por departamentos de documentación técnica, instituciones educativas o industrias que trabajan con inspección visual. La posibilidad de adaptar y reutilizar scripts específicos permite ahorrar tiempo, reducir errores y aumentar la eficiencia, sin necesidad de recurrir a herramientas comerciales. Esta capacidad de aplicar soluciones técnicas sostenibles y personalizadas en contextos reales refuerza también el valor de este tipo de conocimientos en el mercado laboral.

#### **Impacto social y educativo**

El uso de GIMP, una herramienta completamente libre y gratuita, y el desarrollo de scripts que pueden compartirse y adaptarse libremente, contribuyen a democratizar el acceso a tecnologías de procesamiento de imagen. Esto resulta especialmente relevante en contextos educativos o comunitarios con recursos limitados, donde las barreras de acceso al software comercial pueden limitar las oportunidades de formación y creación. En este sentido, el proyecto se alinea claramente con los **ODS 4 (Educación de calidad)** y **ODS 10 (Reducción de las desigualdades)**, al promover un uso inclusivo y abierto de la tecnología.

#### **Impacto económico**

Desde el punto de vista económico, el uso de herramientas libres como GIMP reduce los costes asociados a licencias y mantenimiento de software propietario. Esta ventaja es especialmente significativa para pequeñas organizaciones, instituciones públicas, centros educativos o usuarios independientes. La reutilización de scripts para automatizar tareas que de otro modo requerirían intervención manual o herramientas especializadas demuestra que es posible construir soluciones eficientes y de bajo coste, lo que se vincula con el **ODS 8 (Trabajo decente y crecimiento económico)**.

#### **Impacto medioambiental**

Aunque el impacto medioambiental de un proyecto de software es indirecto, la automatización de tareas mediante scripts puede contribuir a una mayor eficiencia en el uso de los recursos informáticos. Reducir el tiempo de procesamiento, evitar operaciones redundantes y sustituir tareas físicas (como la impresión de imágenes para anotación) por flujos digitales, puede suponer una disminución del consumo energético y de materiales. Esta eficiencia técnica se alinea con los objetivos del **ODS 12 (Producción y consumo responsables)** y **ODS 13 (Acción por el clima)**.

#### **Impacto cultural**

El acceso libre a herramientas de edición y procesamiento de imágenes también tiene un componente cultural. Permite que más personas participen en la creación de contenido visual, artístico o documental, sin barreras económicas o técnicas. El fomento de una cultura colaborativa y abierta, basada en el software libre y el intercambio de conocimientos, es parte del espíritu del **ODS 11 (Ciudades y comunidades sostenibles)** y del **ODS 17 (Alianzas para lograr los objetivos)**, al promover la cooperación y la difusión de soluciones compartidas.

#### **Decisiones motivadas por el impacto**

Varias decisiones técnicas adoptadas durante el desarrollo del proyecto han estado guiadas por la voluntad de maximizar el impacto positivo del trabajo. Entre ellas destaca la elección de GIMP como herramienta principal, priorizando el uso de software libre frente a soluciones comerciales. Asimismo, se ha optado por Scheme como lenguaje principal por su estabilidad en GIMP 3.0 y su bajo consumo de recursos. Además, todos los scripts han sido diseñados para ser modulares, parametrizables y ejecutables desde terminal, lo que facilita su adaptación y reutilización por parte de otros usuarios.