VNiVERSiDAD
D SALAMANCA

MÁSTER EN GEOTECNOLOGÍAS CARTOGRÁFICAS

EN INGENIERÍA Y ARQUITECTURA

Escuela Politécnica Superior de Ávila

Aplicación para procesamiento digital de imágenes en Visual Basic.NET

PROYECTO FIN DE MÁSTER (Junio - 2013)

LUIS MARCOS RIVERA

TABLA DE CONTENIDO

TABLA DE ILUSTRACIONES

1 INTRODUCCIÓN

1.1 Resumen

En el presente documento, se insta al lector a adentrarse en el desarrollo de aplicaciones en entorno .NET, desde las fases más tempranas de desarrollo, hasta la utilización de los productos creados, estando todo ello orientado al procesamiento digital de imágenes. Además, se intentará ofrecer una visión clara y concisa al lector de todo lo relacionado con el procesamiento de imágenes y cómo implementarlo de cara al desarrollo de software. No obstante, se debe tener en cuenta que no se hará énfasis en ninguna aplicación de los métodos en concreto, más bien se intentará dar una visión global de qué es y cómo tratar de forma digital las imágenes.

Todo el proyecto se ha desarrollado en Visual Studio 2012 utilizando como lenguaje Visual Basic .NET, creando como fin último, una aplicación denominada *Apolo* y una biblioteca de clases denominada *ClaseImages.dll*.

1.2 Estado del arte

El procesamiento digital de imágenes es un campo utilizado hoy día en innumerables ramas de investigación, como son, medicina, análisis de cubiertas terrestres, información meteorológica, etc., por lo tanto hay que tener en cuenta que avanza a pasos agigantados y cada vez son más específicas sus aplicaciones. Dentro de este conjunto de técnicas se deben tener conocimientos muy específicos para cada aplicación, siendo generalmente desarrollados por un conjunto multidisciplinar en el que intervienen personas con conocimientos matemáticos, computacionales y, dentro de cada categoría, expertos en la materia a analizar (meteorólogos, médicos, etc.).

Debido a la complejidad de este campo, se pretende dar una visión genérica del procesamiento digital de imágenes y no únicamente una aplicación en concreto, sino las técnicas utilizadas habitualmente para conseguir los diferentes objetivos.

1.3 Contexto del trabajo desarrollado

El trabajo se ha desarrollado como parte del Máster Universitario en Geotecnologías Cartográficas en Ingeniería y Arquitectura, en concreto como proyecto fin de máster (en su rama de investigación). Para la consecución de este proyecto se ha tenido que seguir una evolución lógica que va, desde las bases de programación y procesamiento de imágenes adquiridas en asignaturas previas, hasta el alcance de los objetivos gracias a las bases sentadas y al conocimiento adquirido en su realización.

2 OBJETIVOS PERSEGUIDOS

El objetivo principal de este trabajo es el aprendizaje de las técnicas, tanto a nivel teórico como práctico, del desarrollo de algoritmos para el procesamiento digital de imágenes. Con todo ello, se ha intentado desarrollar un software (*Apolo*) que presente un entorno amigable, sencillo y limpio de cara al usuario. Además de la creación de una aplicación, se ha intentado crear un entorno para que aquellas personas que estén menos familiarizadas con el desarrollo de software, tengan muchas facilidades para poder desarrollar sus propios algoritmos teniendo una base teórica y práctica, todo ello incluido en una biblioteca de clases creada con este objetivo (*ClaseImagenes.dll*).

Se ha realizado énfasis en que el objetivo principal es el aprendizaje, y no el crear una aplicación realmente útil en algún campo de investigación, ya que como estudiante, el fin último de todo es aprender. No obstante, a partir de este trabajo se abren multitud de puertas a la hora de emprender otros proyectos relacionados, gracias a lo aprendido en torno al desarrollo de software y más en concreto a cómo tratar digitalmente una imagen para poder obtener valiosa información de ella.

Por último, indicar que siempre se ha tenido en mente que todo lo creado sea libre, y que cualquier persona pueda tener acceso sin ningún tipo de restricción, para así devolver una pequeña parte de lo que la comunidad aporta diariamente a las vidas de todos. Y por ello, se ha licenciado el código fuente como MIT para que cualquiera pueda modificarlo con cualquier fin (lucrativo o no lucrativo) sin ningún tipo de restricción.

3 INSTRUMENTACIÓN

En esta sección se presenta el conjunto de medios empleados para la creación de todo el proyecto y una breve explicación para poner en contexto al lector.

3.1 Visual Studio 2012 (Express)

La espina dorsal de todo el proyecto gira en torno a este software desarrollado por Microsoft. Se ha utilizado en su versión gratuita (Express) que se puede adquirir en la web oficial (http://www.microsoft.com/visualstudio/esn/downloads#d-2012-express). Este software es lo que se denomina una IDE o entorno de desarrollo integrado, e incluye un conjunto de herramientas para facilitar las diferentes tareas a los programadores.

En concreto se ha utilizado el lenguaje de programación Visual Basic .NET por ser un lenguaje de alto nivel y extremadamente simple en todos sus aspectos. Pero uno de los productos finales desarrollados (*ClaseImagenes.dll*) es compatible con todo el entorno .NET.

3.2 HelpNDoc 3

Este software se ha utilizado principalmente para desarrollar los archivos de documentación para *Apolo*. Permite crear archivos de ayuda (*.chm*) que son de gran apoyo puesto que el usuario final está habituado a este tipo de formatos. Además, se pueden exportar los proyectos como páginas web (*HTML*, *CSS*, *JavaScript*), *pdf*, etc. Mediante este software, se ha creado el manual de usuario para la aplicación *Apolo*, además del manual para desarrollar con la biblioteca *ClaseImagenes.dll*.

Se ha utilizado en su versión gratuita (http://www.helpndoc.com/download), que únicamente incluye una marca de agua en cada documento informando que se ha utilizado una versión de prueba.

3.3 Sandcastle

Software utilizado para la generación de documentación de código fuente y que provee una documentación visualmente muy atractiva y similar a la utilizada en la documentación *MSDN*. Se sirve de los comentarios *XML* incluidos en el código fuente desarrollado con .NET, y facilita al desarrollador la generación de archivos de ayuda (*.chm*) o como página web (*HTML*, *CSS*, *JavaScript*). Con él, se ha creado un manual con todos los miembros de la biblioteca de clases (*ClaseImagenes.dll*) y su correspondiente explicación de parámetros, comentarios, valores de retorno, etc.

Se puede descargar directamente desde el repositorio CodePlex (http://sandcastle.codeplex.com/).

3.4 Inno Setup Compiler

Software libre utilizado para la generación de archivos instalables a partir de proyectos. Se utiliza principalmente para generar un asistente de instalación y poder incluir todos los ficheros de forma sencilla en la carpeta de instalación.

Este programa se ha utilizado para crear un archivo de instalación y que así el usuario final tenga más fácil el acceso a *Apolo*. Se puede descargar en su página web oficial (http://www.jrsoftware.org/isdl.php).

3.5 GIMP

Aplicación orientada a edición de imágenes con el cual se pueden manipular imágenes y crear archivos vectoriales. Es un software libre y gratuito. Se ha utilizado para crear/manipular iconos que después han sido incluidos en Apolo.

Se puede descargar de forma gratuita desde la web del desarrollador (http://www.gimp.org/downloads/).

3.6 Git

Es un software de control de versiones dedicado a hacer más sencillo el manejo de las diferentes versiones de una aplicación y poder incluir nuevas funcionalidades teniendo siempre código de respaldo. Es software libre licenciado como GNU. En el contexto del proyecto, se ha utilizado para ir añadiendo funcionalidades a *Apolo* teniendo siempre versiones de respaldo y así evitar pérdidas de código.

Se puede descargar de forma gratuita desde su web (https://code.google.com/p/git-core/downloads/list).

3.7 Github

Es una plataforma de desarrollo colaborativa para alojar proyectos que utilizan el control de versiones Git. Hay dos modalidades, una de pago y otra gratuita, siendo esta última utilizada para alojar todo el código fuente de Apolo.

Se puede almacenar todo tipo de proyectos de forma gratuita (teniendo que ser públicos) en su web oficial (https://github.com/).

4 VISIÓN GLOBAL DEL PRODUCTO

El producto que se presenta responde a la demanda de crear un software totalmente gratuito para el aprendizaje y profundización de los diferentes algoritmos para tratamiento digital de imágenes.

El producto software resultante de este proyecto, *Apolo*, está pensado para ser un programa que sea totalmente extensible. Un producto en el que cualquier usuario sea capaz de añadir funcionalidades o modificar las existentes, de manera que pueda adaptarlos a sus necesidades. Uno de los grandes problemas de los programas privativos es que el usuario no puede adaptarlo a sus menesteres, sino que debe ceñirse exclusivamente a las herramientas que le proporciona el diseñador.

Puede uno, como lector, imaginarse un software libre y gratuito, en constante expansión, en el que cada usuario desarrolle de forma fácil y adaptándolo a sus necesidades, de forma que sus modificaciones puedan ser aprovechadas por el resto de usuarios. Esto es a grandes rasgos lo que pretende ser Apolo.

Con todo esto, no se depende de que el creador del software tenga en cuenta las necesidades de cada usuario, sino que cada usuario adapta el software en función de las suyas.

Además de Apolo, se ha creado una biblioteca de clases denominada *ClaseImagenes.dll*, siendo esta el esqueleto sobre el que se ha desarrollado Apolo, es decir, el usuario puede ampliar las funcionalidades de Apolo o bien puede crear un proyecto nuevo tomando como base ClaseImagenes.dll.

4.1 Antecedentes

Existen multitud de aplicaciones, tanto privativos como libres, para el procesamiento digital de imágenes, pero todas ellas o bien se centran en algún tipo de procesamiento en específico, o son tan completas que es excesivamente complejo utilizarlas. En cambio, Apolo pretende ser una herramienta generalista en el que el usuario sea parte activa del desarrollo y así, partiendo del esqueleto de ClaseImagenes.dll, pueda focalizar y orientar los esfuerzos a un tipo de aplicación en concreto.

4.2 Alcance

Como se ha comentado con anterioridad, se espera conseguir que la aplicación desarrollada sea sencilla de utilizar, recordando que los posibles usuarios puede que tengan conocimientos informáticos o de tratamiento de imágenes limitados. Se ha intentado también, que la aplicación sea estéticamente lo más atractiva posible.

La aplicación se ha desarrollado utilizando Visual Basic .NET sobre el IDE Visual Studio 2012, que comprende un ambiente de desarrollo con una interfaz gráfica de usuario muy potente, que facilita la creación de interfaces gráficas y, en cierta medida, la programación de las mismas. Este lenguaje, a pesar de no ser uno de los más utilizados en la actualidad, sigue siendo una herramienta potente, moderna y fácil de utilizar por lo que es apta tanto para desarrollos rápidos como para proyectos más complejos. Por

todo ello, y por el conocimiento previo del lenguaje, se optó por utilizarlo como base de todo el proyecto, y así, un usuario con unos conocimientos mínimos de programación y cierto estudio, podría desarrollar aplicaciones específicas para tratamiento digital de imágenes a partir de todo lo creado.

5 METODOLOGÍA

Antes de profundizar en el desarrollo del proyecto se va a concretar lo que es una metodología. Una metodología es el conjunto de pasos, procedimientos, técnicas, herramientas y el soporte documental que ayudan a desarrollar un producto software.

En este proyecto en concreto, se ha seguido un enfoque orientado a objetos (*POO*). La orientación a objetos es un paradigma de la ingeniería del software que basa la arquitectura de los sistemas en los datos / objetos que el sistema manipula. Estos objetos representan las entidades físicas y conceptuales del mundo real relacionadas con la aplicación.

En la evolución de la orientación a objetos hubo una avalancha de notaciones y técnicas, que se solucionaron con la estandarización, que dio lugar al método conocido con el nombre de *Método Unificado*. Este posteriormente se transformó en *UML* que es el que se usará en este proyecto.

5.1 ¿Por qué usar POO?

El desarrollo de un sistema consiste en una serie de iteraciones del tipo división-unión; hay que dividir el problema para comprenderlo e ir uniendo las partes divididas para construir la solución.

El proceso de división se realiza separando las distintas funciones del sistema y descomponiendo estas en subfunciones sucesivamente hasta obtener elementos simples. La arquitectura de un programa así realizado es un reflejo del sistema estudiado. Este método aporta resultados satisfactorios cuando las funciones están bien definidas y son estables en el tiempo. Sin embargo, al existir un fuerte acoplamiento entre arquitectura y funciones, las evoluciones importantes pueden suponer modificaciones estructurales importantes.

La descomposición orientada a objetos se basa en los objetos que integran la estructura y el comportamiento. Las funciones se representan mediante colaboraciones entre los objetos que componen el sistema.

El empleo de las metodologías orientadas a objetos se justifica con las siguientes razones:

- La orientación a objetos se acerca más a la forma de pensar de las personas, haciéndolo más comprensible y fácil de aplicar tanto para el creador de la aplicación, como para el posible usuario que desee realizar ampliaciones.
- Facilita la abstracción, centrándose en cada momento únicamente en los elementos que interesan y descartando los demás.
- Permite expresar características comunes sin repetir la información.
- Facilita la reutilización de diseños y código.
- Facilita la revisión y modificación de sistemas desarrollados.
- Origina sistemas más estables y robustos.

5.2 Modelo de proceso

El desarrollo de un software supone un gran esfuerzo que puede durar desde varios meses a varios años. Lo ideal es dividir el trabajo en pequeñas partes. Cada parte es una iteración que da lugar a un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo y los incrementos al crecimiento del producto.

En cada iteración se identifica y especifica los casos de uso relevantes a la hora de desarrollar la aplicación. Las iteraciones construyen los modelos resultantes incremento por incremento, pasando cada iteración por el estudio de requisitos, análisis, diseño, implementación y pruebas.

El proceso orientado a objetos se mueve a través de una espiral evolutiva que comienza con la comunicación del usuario. Es aquí donde se define el dominio del problema y se identifican las partes básicas del sistema.

La programación orientada a objetos hace hincapié en la reutilización, por tanto, se debe buscar las clases en una biblioteca de clases orientadas a objetos antes de construir una.

5.3 Planificación

5.3.1 Incrementos

Debido a la naturaleza del proyecto, se ha decidido que lo más adecuado es utilizar un modelo incremental. Crear una primera iteración e ir añadiendo nuevas funcionalidades es la idea que más se adapta a este tipo de proyecto, ya que se parte de unas funcionalidades básicas y se van aumentando a medida que se avanza en el proyecto.

Esto ayuda a no ver el programa como un todo, sino como pequeñas partes que se van incorporando. Debido a que la creación del programa se realiza por una única persona, esto hace que la tarea sea mucho más liviana, y permite un desarrollo del software mucho más eficiente.

En el primer incremento sólo se consideran los requisitos básicos del sistema, en este caso la especificación inicial de requisitos, los formatos de los archivos, los menús, los tipos de transformaciones, etc. Una vez realizado el primer incremento, se prueba y a partir de él se planifican las modificaciones a realizar en el siguiente incremento y así progresivamente hasta conseguir el producto final.

A continuación se muestran los principales incrementos realizados en el proyecto Apolo.

5.3.1.1 Incremento 1: Requisitos básicos del sistema

En este incremento se ha planificado el tiempo que se dispone para la realización del proyecto y las distintas tareas que se deberían realizar en cada momento. El tiempo inicial de trabajo será de 10 de la mañana a 2 de la tarde. Por tanto, se intentará realizar cada día o días una tarea distinta, con el fin de hacer más liviano el trabajo de realizar un proyecto largo.

Se inicia con la especificación de lo que va a realizar globalmente el programa, es decir, lo que se desea que realice el software sin pensar en las distintas partes específicas de cada módulo.

Dado que los conocimientos previos en programación no eran excesivos, además de no haber tenido antes ningún contacto con Visual Basic .NET (sólo con Visual Basic 6), la primera etapa tras la planificación del proyecto, fue el estudio del lenguaje y el modo de uso de su IDE.

5.3.1.2 Incremento 2: Planificación del software

En este incremento se ha tratado de dejar totalmente de lado la interfaz de usuario, y se ha centrado principalmente en el alcance del proyecto, es decir, en cuáles serán los objetivos finales y las funcionalidades que tendrá el software que se creará.

Posiblemente este sea uno de los incrementos más importantes, puesto que cualquier cambio en los requisitos iniciales debido a una mala planificación, puede incurrir en muchas horas de trabajo perdido. Principalmente, se han analizado las necesidades que debe de tener una aplicación informática de tratamiento digital de imágenes, y a partir de ahí, qué funcionalidades se van a implementar y cuáles no. Este incremento se ha nutrido principalmente del manejo de otros programas existentes, como pueden ser GIMP, Paint, Fireworks, etc. Con todo ello, cabe decir que este incremento no fue del todo satisfactorio ya que no fueron bien recogidas todas las especificaciones y alcance del proyecto y en ocasiones, hubo que rehacer código y perder varios días de trabajo.

5.3.1.3 Incremento 3: Estructura de la clase ClaseImagenes

Una vez bien definidos los requisitos del sistema y bien planificado la estructura del software, el siguiente paso ha sido planificar la estructura de la clase denominada *ClaseImagenes*. Dentro de este incremento se han analizado las diferentes funciones, eventos, propiedades, etc., que debería tener la clase para facilitar todo el desarrollo e intentar separar la parte lógica de la parte gráfica en la mayor medida posible.

Aquí se ha planificado cómo debería gestionar la clase todas las imágenes que recibe, almacenando todas ellas para que después se pueda tener acceso a ellas desde la aplicación principal. Además, se ha tenido en cuenta que se deberían crear diferentes eventos para *avisar* de cualquier cambio en una imagen. Dentro de este incremento, también se han observado las diferentes formas de entrada/salida de imágenes al programa, para así hacer más fácil al usuario final la adquisición de imágenes.

Dentro de este incremento únicamente se ha intentado tener en cuenta todos los posibles aspectos y necesidades de la clase, intentando dar una visión lo más general posible de la misma.

5.3.1.4 Incremento 3: Interfaz principal de usuario

En este incremento se ha intentado esbozar el aspecto de la interfaz de usuario, principalmente de la pantalla principal. Se ha tenido en cuenta que el usuario está acostumbrado a un tipo de aplicación que incluye una barra de herramientas superior, una barra lateral con información, y una barra en la parte inferior o barra de estado.

En este incremento se ha decido que la aplicación tenga dos zonas claramente diferenciables, siendo la de la parte izquierda la correspondiente a la imagen abierta y la parte derecha un control que está formado por tres pestañas y mostrará información sobre los histogramas, todas las imágenes transformadas y zoom.

En este incremento se ha intentado dar una visión general, sin entrar en detalles, del aspecto que debería tener la aplicación para cumplir con los requerimientos impuestos.

5.3.1.5 Incremento 4: Control de errores

Este incremento ha consistido en la creación de una clase para gestionar todos los errores que puedan suceder a la hora de utilizar la aplicación. Debido a que las diferentes funciones de tratamiento digital de imágenes pueden generar diferentes errores y de muy difícil detección, se ha optado por planificar una clase que, ante cualquier error ocurrido en el programa, lo captura (junto con una captura de pantalla), para que el usuario pudiese notificarlo.

5.3.2 Desarrollo del calendario

Se ha desarrollado un calendario para la creación de Apolo, dividiendo todo el proceso de desarrollo en una serie de bloques. En un primer momento la duración del proyecto iba a ser menor, pero debido a ciertos problemas, y a la incorrecta definición del alcance del proyecto, el calendario no se ajustó a los tiempos establecidos.

5.3.3 Esfuerzo

Hay que destacar que la mayor parte del tiempo se ha dedicado al estudio de la metodología utilizada para programar, además del estudio propio del lenguaje, ya que los conocimientos iniciales eran básicos, y a medida que se ha ido desarrollando el proyecto, éstos han aumentado considerablemente.

Otra parte importante ha sido el pensar y analizar las distintas posibilidades que se podían incorporar teniendo en cuenta las limitaciones de tiempo y las propias. Cabe destacar el intento de optimización de código y la administración correcta de los recursos que necesita el software.

Uno de las partes más complejas ha sido la creación de una aplicación *multihilo*, debido a que el manejo de controles desde un hilo diferente al principal es en ocasiones complejo, además de que el no tener únicamente un hilo en toda la aplicación, genera errores de muy difícil detección.

5.4 Análisis

5.4.1 Metodología de desarrollo

En este proyecto se ha seguido una metodología de desarrollo ágil de software, que se basa en procesos ágiles, es decir, se intenta evitar los formales caminos de las metodologías tradicionales enfocándose en las personas y los resultados.

Esta metodología promueve iteraciones en el desarrollo a lo largo de la vida del proyecto, minimizando los riesgos al desarrollar software en cortos períodos de tiempo. Estos períodos de tiempo se llaman iteraciones, las cuales pueden durar de unos días a varias semanas. Cada iteración debe de contener: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no debe agregar demasiada funcionalidad para justificar la finalización del proyecto, pero la meta es tener una versión de prueba al final de cada iteración. Esta prueba final consiste en el correcto funcionamiento de la parte desarrollada en la iteración. Cada vez que se finaliza una iteración, el proyecto debe pasar a manos de los revisores, que en este caso puede ser cualquier persona ajena al desarrollo del proyecto, y lo evalúa, comenta las cosas que más le han llamado la atención y los fallos más evidentes, y aconseja cambios en cualquiera de las funcionalidades que se han implantado en esta nueva iteración. Estos fallos pueden ser desde el aspecto visual, hasta la dificultad del entendimiento de cómo funciona el software.

Estos métodos ágiles enfatizan las comunicaciones cara a cara en vez de la documentación. Lo cual es más rápido, consigue un producto más cercano al público y en menor tiempo.

Como se ha ido viendo, el proyecto comenzó con sólo la estructura lógica y el aspecto visual y a medida que fueron avanzando las iteraciones o incrementos se fueron añadiendo nuevas funcionalidades y herramientas, hasta conseguir un producto final con varias herramientas y muchas funcionalidades. Siempre teniendo en cuenta las opiniones de las distintas personas, que iban adquiriendo el papel de revisores, al no disponer de un equipo de trabajo, han sido personas cercanas, como familia y amigos. Éstos revisaban las versiones después de cada iteración y comentaban sus impresiones.

5.4.2 Análisis del sistema

El objetivo del análisis de sistemas es realizar un documento de especificación de requisitos: especificar qué tiene que hacer el sistema y no cómo desarrollarlo.

Para el análisis del sistema se ha utilizado un método de análisis orientado a objetos.

Para desarrollar las ideas del proyecto se ha usado la técnica llamada *Brainstorming* (tormenta de ideas) en la cual se comentaba la aplicación que se quería desarrollar en ese momento, siendo lo más claro posible y teniendo en cuenta la familiarización de la persona con lo que quería realizar, cada uno aportaba sus ideas sin juzgar su validez siendo ésta evaluada por el desarrollador, adquiriendo las ideas más interesantes. Una vez adquiridas dichas ideas, se dedicaba un tiempo a valorar las ideas más prácticas y originales que se adaptaban mejor a la naturaleza del proyecto y a sus características, además de su viabilidad en función de la dificultad implícita de la idea y del tiempo requerido por ella.

5.4.3 Funciones del producto

Apolo está formado por varias clases. Primeramente se van a mostrar las funciones principales de las clases de la interfaz gráfica y luego se describirán cada una de las aplicaciones que componen el producto final.

5.4.3.1 Interfaz de usuario

Uno de los principales objetivos de un software de escritorio, sea del tipo que sea, es la interfaz gráfica. El software dispone de una interfaz agradable y sencilla, con numerosos accesos directos desde teclado para poder manejarlo en ausencia de ratón.

Se ha intentado que todos los contenidos están bien organizados y sea todo muy intuitivo y transparente para el usuario.

Es posible que la interfaz gráfica sea mejorable para hacerlo más atractivo, pero es un proyecto bastante largo y hay que abarcar todos los campos en un tiempo reducido. Cabe destacar que cualquier producto software de la actualidad está desarrollado usando un equipo de trabajo bien organizado y que cada miembro se ocupa de una parte del software, y en este caso, sólo una persona está a cargo tanto del desarrollo de ideas para el programa como de la implementación, pasando por otros campos como el diseño gráfico.

5.4.3.2 Estructura del código

En esta fase de análisis se ha intentado buscar una forma de que el código sea muy legible porque uno de los objetivos del programa, es que el resto de usuarios puedan desarrollar sobre él sin demasiado esfuerzo. Para ello se ha optado por la inclusión de comentarios *XML* ya que éstos se integran perfectamente con Visual Studio gracias a su asistente *IntelliSense*, y la total separación entre la clase principal del programa (*ClaseImagenes*) y el resto de la aplicación, y así, aunque la aplicación en su conjunto pueda no ser demasiado reutilizable y/o extensible, el grueso de la clase es totalmente extensible siguiendo unas mínimas pautas a la hora de ampliarla.

5.5 Diseño

La fase de diseño se centra en aplicar ciertas técnicas y principios cuyo objetivo consistirá en definir el sistema con el suficiente detalle como para permitir la posterior implementación de éste por parte del programador.

Teniendo bien claros los requisitos extraídos en la etapa de análisis, ésta etapa no debe resultar excesivamente compleja. El resultado de la fase de diseño debe ser una guía que recoja tanto los requisitos explícitos contenidos en la fase de análisis como los requisitos implícitos que se desea que cumpla el software, y que debe poder leer y entender las personas que utilicen, prueben y amplíen el software.

De manera breve se ilustrarán las distintas funciones y procedimientos que componen los distintos módulos, las distintas opciones que permiten y qué aporta.

5.5.1 Diseño de interfaz

En esta fase se debe diseñar cómo quiere que sea la estructura final de cara al usuario, es decir, intentar tomar un criterio estable a partir del cual se van a desarrollar las diferentes herramientas o módulos en la aplicación. Para ello, se tomó la decisión de utilizar colores estándar en los diferentes formularios y el mismo tipo de letra a un tamaño legible. Además, se ha pretendido crear una interfaz nada sobrecargada en el

que prime la información sobre el resto (no muchos objetos o controles pero bien distribuidos). Esto ha llevado a la creación de múltiples funciones muy similares pero en diferentes menús, es decir, dentro de unas funcionalidades que se podrían agrupar en un mismo formulario, se ha preferido separarla en varios para que el usuario tenga una experiencia lo más fluida y simple posible.

También se decidió incluir accesos rápidos en forma de combinación de teclas para hacer más ágil la utilización del software, estando bien referenciadas en la documentación. Otro punto que se tuvo en cuenta es la utilización del botón derecho para desplegar un menú contextual tan utilizado en aplicaciones y que el usuario está acostumbrado a utilizar.

En última instancia, se decidió utilizar combinaciones que más o menos son un *estándar de facto* en el mundo de las aplicaciones de tratamiento digital de imágenes, como puede ser el pulsar la tecla *shift* y la rueda del ratón para hacer zoom en zonas de una imagen, por ejemplo. Aquí también se decidió incluir una barra de progreso, pues el usuario siempre prefiere saber el progreso de una determinada transformación a un simple texto que sólo informe de que se está llevando a cabo una transformación pero no sepa el progreso de la misma.

5.5.2 Diseño del código

En esta fase se estudia, después de bien conocidos los requerimientos, la forma de estructurar el código. Se estudió cómo sería la mejor forma de diseñar el código, teniendo ya en mente una clara separación entre la clase *ClaseImagenes* y el resto de la aplicación. Para ello se decidió incluir una primera sección con propiedades de la clase y a continuación las funciones propias de la clase organizadas en tres grandes regiones, funciones tratamiento, funciones abrir/guardar, funciones extra. La región más importante es la primera, en la que se decidió en esta etapa dividirla en diferentes secciones para una mayor facilidad a la hora de revisar el código. Las secciones incluidas dentro de la región funciones tratamiento son las siguientes:

- Operaciones básicas: incluirá las transformaciones más simples dentro del procesamiento de imágenes digitales, como son, binarización, transformación a escala de grises, exposición, etc.
- Máscaras: en esta sección se alojarán las funciones para aplicar máscaras o kernel a imágenes, además de una serie de máscaras predefinidas.
- Operaciones aritméticas: se incluirán las operaciones aritméticas básicas píxel a píxel, como puede ser, suma, resta, división, etc.
- Operaciones lógicas: se incluirán las operaciones lógicas básicas píxel a píxel, como son el operador AND, OR y XOR.
- Operaciones morfológicas: en esta sección deberían incluirse los operadores morfológicos más comunes entre los que se encuentra la erosión, apertura o cálculo del perímetro.
- Operaciones estadísticas: contendrá operaciones estadísticas que se aplicarán a través de un kernel, por ejemplo, mediana o moda.
- Efectos: esta sección acogerá una serie de efectos en los que no se tendrá en cuenta las transformaciones para extraer información de imágenes, sino simplemente artísticas.

- Operaciones con dos imágenes: esta región albergará todo lo relacionado con unión, suma, resta, etc., de dos imágenes que darán como resultado una.
- Comparación de imágenes: albergará los diferentes métodos para determinar si dos imágenes son iguales o en qué proporción son distintas.

5.6 Implementación

5.6.1 Lenguajes

En la etapa de implementación, el principal objetivo es transformar el análisis y diseño realizados en las etapas anteriores, en una aplicación que satisfaga las necesidades de los usuarios. Para conseguir tal fin, se ha elegido el entorno de desarrollo Visual Studio 2012 junto con el lenguaje Visual Basic .NET, siendo un lenguaje de programación orientado a objetos.

Desarrollado en sus inicios por el alemán Alan Cooper para Microsoft, este lenguaje de programación es un dialecto de BASIC, con importantes agregados. Visual Basic .NET, posee una curva de aprendizaje muy rápida debido a su simplicidad en la sintaxis y a poseer una extensa documentación. A pesar de que actualmente no es ampliamente utilizado, es una herramienta muy adecuada para aplicaciones que no requieran de grandes alardes tecnológicos. Entre las ventajas principales se encuentran las siguientes:

- Posee una curva de aprendizaje muy rápida.
- Integra el diseño e implementación de formularios de Windows.
- Permite usar con facilidad la plataforma de los sistemas Windows, dado que tiene acceso prácticamente total a la API de Windows, incluidas librerías actuales.
- El código es gestionado gracias al framework .NET.
- Fácilmente extensible mediante bibliotecas de clases de cualquier lenguaje del entorno .NET.
- Posibilita añadir soporte para ejecución de scripts, VBScript o JScript, en las aplicaciones mediante Microsoft Script Control.
- Si bien permite desarrollar grandes y complejas aplicaciones, también provee un entorno adecuado para realizar pequeños prototipos rápidos.

Principalmente se escogió este entorno de desarrollo por ser un lenguaje de muy alto nivel (expresa los algoritmos de una manera adecuada a la capacidad cognitiva humana), y por la extensa documentación existente en castellano.

5.6.2 Bibliotecas utilizadas

Para el desarrollo completo del software, se utilizó una biblioteca de clases denominada *Newtonsoft.Json.Silverlight*. Ésta no es utilizada directamente por la clase *ClaseImagenes*, por lo que únicamente es utilizada de forma directa por la aplicación Apolo. Esta biblioteca se utiliza para analizar y extraer de forma sencilla información que está en formato *JSON*.

5.6.3 Detalles de la implementación

5.6.3.1 Esquema general

El diseño del esquema principal de la aplicación está formado por formularios cuya tamaño está fijado, a excepción del formulario principal. La decisión de realizarlo de esta manera se ha basado principalmente en los problemas producidos a la hora del reajuste de todos los controles cuando éstos se maximizan.

Se han elegido tonos típicos de aplicaciones *Windows Forms* por ser los colores a los que el usuario está habituado.

5.6.3.2 Desarrollo de Apolo

Una vez finalizada la etapa de análisis y diseño la siguiente etapa consiste en aplicar todo lo desarrollado e intentar implementarlo de la forma más rápida y eficaz sin perder la robustez en el sistema. Para ello se ha intentado en la medida de lo posible, probar cada fragmento de código para evitar posibles fallos y fomentar la reutilización de código.

A la hora de implementar todo lo desarrollado en las anteriores etapas, se ha decidido ir creando las diferentes opciones del programa a medida que se han ido añadiendo funcionalidades a *ClaseImagenes*. Quizá este no haya sido el mejor método para separar lógica de interfaz, pero es un método mucho más gratificante de cara al programador. El proceso es, añadir una función a la clase principal, y a continuación su correspondiente formulario e implementación con el resto del programa. Y este proceso se ha seguido durante todo el desarrollo. Una ventaja de hacerlo así, es que se garantiza en mayor medida el correcto funcionamiento tanto de la función, como de la interfaz asociada a esa función, puesto que se añade una funcionalidad y automáticamente se implementa y prueba. En caso contrario, si primero se crea todo el conjunto de funciones y procedimientos en la clase, y una vez realizado esto se adapta al resto del programa, es posible generar más inconsistencias entre la parte lógica y la parte gráfica.

5.7 Pruebas y validación

En este apartado se presentará cómo se ha procedido al realizar las pruebas de los distintos componentes del software.

5.7.1 Plan de pruebas

Según las recomendaciones de G. J. Myers:

1. Cada caso de prueba debe especificar el resultado esperado: es decir, el programa funciona correctamente y devuelve unos datos correctos.
2. El programador debe evitar probar sus propios programas: los usuarios elegidos para probar el proyecto siempre son personas ajenas al software.
3. Se debe analizar con detalle el resultado de cada parte: se debe comprobar con detenimiento que todas las funciones de cada herramienta funcionan correctamente y muestran los resultados de forma correcta.

4. Se deben incluir entradas válidas e inválidas: El usuario externo al proyecto prueba todos los formatos y se comprueba que no producen efectos no deseados.
5. Se debe:
   - Probar si el software no hace lo que debe. El programa debe captar los eventos necesarios y cargar los módulos que hacen falta.
   - Probar si el software no hace lo que no debe. El programa se vuelve inestable en algún caso en el que debería funcionar correctamente.
6. Los casos de prueba deben documentarse: se deben mostrar las pruebas realizadas en cada iteración más adelante.
7. Hay que asumir que hay errores: en el momento en el que programamos códigos largos suelen encontrarse muchos fallos.
8. Donde hay un defecto hay otros: en caso de que encontremos un fallo, ese fallo puede provocar otros más o menos graves que éste.
9. Las pruebas son una tarea creativa: no nos debemos quedar en las pruebas ideales, debemos probar el programa con todo tipo de casos y situaciones.

5.7.2 Especificación del diseño de pruebas

5.7.2.1 Durante el desarrollo de la aplicación

Las pruebas se desarrollaron con datos que se adaptaban a los requisitos del módulo que se estuviera implementando en cada momento. Estas pruebas consistían en comprobar el tratamiento de la información por parte del módulo, así como la recuperación y almacenamiento de dicha información. Con estas pruebas se evitaba la propagación de errores básicos a etapas posteriores.

5.7.2.2 Desarrollo de la aplicación finalizado

En este momento de finalización del desarrollo se debía comprobar que todos los subsistemas interactuaban correctamente. Además era necesario comprobar que toda funcionalidad se llevaba a cabo de igual manera independientemente de la versión de Windows utilizada por el usuario.

5.7.2.3 Especificación de los casos de prueba

La tarea de prueba de la aplicación fue realizada por 2 grupos de personas distintos:

5.7.2.3.1 Personas con nivel bajo de conocimientos informáticos

Estas personas tienen conocimientos básicos a nivel informático desarrollando sus tareas habituales con el PC con aplicaciones ofimáticas. En este grupo tienen cabida las personas que no están relacionadas con el ámbito del tratamiento digital de imágenes o que se están iniciando, y no detectaron fallos importantes sino que más bien sugirieron pequeños cambios estéticos orientados a facilitar su trabajo día a día

5.7.2.3.2 Personas con nivel avanzado de conocimientos informáticos

Este grupo está compuesto por personas que dedican profesionalmente o de forma habitual al mundo del retoque digital y procesamiento de imágenes e incluso del

desarrollo de software y tuvieron acceso tanto al código como a la documentación. Ellos sí detectaron algún pequeño error que fue subsanado sin mayor problema e hicieron sugerencias estéticas para una mejor experiencia de usuario.

5.7.2.4 Especificación de los procedimientos de prueba

Al ser la aplicación desarrollada para Windows, ésta debe de ser funcional en las distintas versiones existentes en el mercado. Se utilizaron para las pruebas varios equipos en las que estaban instaladas diferentes versiones como son Windows XP, Vista, 7 y 8. Se comprobó su correcto funcionamiento en todos los sistemas siempre y cuando tuviesen instalado Framework 4, tanto en versiones de 32/64 bits.

5.7.2.5 Conclusiones de las pruebas

En esta etapa se llegó a la conclusión que no existe un sistema exento de errores y que el desarrollo del software se convierte en una pequeña batalla contra el usuario intentando controlar todos los errores que éste pueda cometer, y restringiendo su interacción con el software a las acciones específicas que debe realizar.

6 RESULTADOS

Después de todo el proceso hasta la creación final de la aplicación, se puede decir que el resultado es satisfactorio. Se ha conseguido crear una aplicación útil para el tratamiento digital de imágenes y útil para el aprendizaje de los diferentes algoritmos que se llevan a cabo para mejorar una imagen.

El resultado final es una aplicación informática denominada *Apolo* que incluye una gran cantidad de funcionalidades para modificar, visualizar e interactuar con imágenes. Esta aplicación puede ser utilizada para diversos fines, como son investigación o retoque fotográfico.

Además, también se ha desarrollado de forma conjunta una biblioteca de clases (ClaseImagenes.dll) totalmente funcional y que engloba todo el proceso de creación de una aplicación orientada a procesamiento de imágenes. Quizá éste sea el producto más importante en todo el desarrollo, pues se trata de una biblioteca de clases totalmente funcional que permite, a partir de ella, crear aplicaciones para procesamiento digital de imágenes englobando todo el proceso. Además, se distribuye con código fuente, por lo que se permite modificar, es más, el fin último de todo el proyecto es la liberación del mismo para que la comunidad pueda ampliarlo.

7 CONCLUSIONES

Después de este largo viaje en el que tras varios meses de desarrollo, y después de volver a empezar de cero la aplicación (debido a no determinar bien el alcance del proyecto), se puede decir que el resultado es un software francamente estable, que sin grandes alardes cumple las expectativas esperadas y sobre todo ha hecho que se adquiera un mayor conocimiento de técnicas de programación y en concreto de procesamiento digital de imágenes.

Gracias a todo ello, se puede decir que ha sido un proyecto que ha merecido la pena, en el que ha habido momento un poco desesperantes, pero después de todo, casi la totalidad de los objetivos propuestos se han cumplido.

7.1 Desarrollos futuros

Aquí se muestra una lista de posibles ampliaciones/modificaciones del software desarrollado:

- Añadir diferentes herramientas de configuración del entorno.
- Incluir comentarios del código fuente y programa en inglés.
- Mejora en cuanto a eficiencia a la hora de ejecutar algunos algoritmos.
- Utilización del método Bitmap.LockBits para guardar directamente en memoria los datos de los píxeles y así hacer transformaciones mucho más rápidas.
- Crear la opción de abrir diferentes imágenes con sus instancias de clase independientes.
- Aumentar las funciones incluidas en la secuencia.

8 ANEXO I. GUÍA DE USUARIO APOLO

8.1 Introducción

8.1.1 Bienvenido

Bienvenido a la documentación de Apolo. En esta guía se ayudará a despejar cualquier duda o incidencia con la aplicación.

8.1.2 ¿Qué es Apolo?

Apolo es un software desarrollado en Visual Basic .NET (sobre en el IDE Microsoft Visual Express 2012 para escritorio de Windows) como proyecto fin de máster para el máster oficial en Geotecnologías Cartográficas en Ingeniería y Arquitectura organizado de forma conjunta por la Universidad de Salamanca (Coordinadora) y la Universidad de Valladolid.

Apolo es una herramienta creada para el procesamiento digital de imágenes teniendo como fin último el aprendizaje de los algoritmos básicos de manipulación de imágenes. El nombre Apolo viene a referirse al dios olímpico de la mitología griega y romana, y más en concreto, a su faceta como dios de las artes.

Apolo no pretende ser una herramienta excesivamente compleja, al contrario, pretende ser una herramienta concisa y muy transparente para el usuario final.

De cara a la creación de nuevas funciones para ampliar sus funcionalidades, Apolo pretende dar las máximas facilidades para promover su ampliación. Siguiendo la guía de desarrollo, y teniendo unos mínimos detalles en cuenta, es muy sencillo crear nuevas herramientas para Apolo.

8.1.3 Historia

Los autores de Apolo son innumerables, ya que a la hora de desarrollarlo, toda la comunidad de Internet ha influido. Desde los foros especializados en Visual Basic u otros lenguajes, hasta páginas especializadas en tratamiento de imágenes, entradas de blogs inspiradoras, otros programas de tratamiento de imágenes, etc. Detrás del teclado, a día de hoy (06/03/2013), sólo se encuentra Luis Marcos Rivera, pero el esfuerzo es de toda la comunidad.

8.1.4 Filosofía

La filosofía de Apolo es el conocimiento de las técnicas básicas para el procesamiento digital de imágenes. Apolo no pretende ser rápido, Apolo pretende ser fácil. Fácil desde el punto de vista de la programación. Apolo podría reducir la velocidad de procesamiento (en muchas ocasiones) a la mitad, pero por su filosofía, prefiere una claridad en todas sus funciones e instrucciones a una mayor velocidad de procesamiento. Y usted podría preguntarse por qué, porque el principal objetivo de Apolo es el aprendizaje de los algoritmos básicos de tratamiento de imágenes (como ya

se ha dicho) y eso se obtiene con claridad y simplicidad. Si no está convencido y cree que lo simple puede hacerse más rápido, no lo dude, colabore con el proyecto.

8.2 Características generales

A continuación se muestra un listado de las principales capacidades de Apolo:

- Soporta formatos de imágenes BMP, GIF, JPEG, PNG y TIFF.
- Posibilidad de abrir imágenes desde Facebook o buscar imágenes en BING.
- Posibilidad de compilar código fuente para crear nuevos algoritmos.
- Herramientas para transformaciones geométricas de imágenes, como son, sesgar, voltear o escalar.
- Procesamiento multihilo.
- Posibilidad de deshacer hasta 100 imágenes. Limitado por la memoria libre y posibilidad de configurarlo.
- Operadores morfológicos.
- Almacenamiento básico en la nube con Apolo Cloud.

8.3 Requerimientos del sistema

- Procesador de 1 GHz.
- 512 MB de memoria RAM.
- 70 MB de espacio en disco duro.
- Windows XP Home Edition o superior.

8.4 Licencia

Apolo está licenciado bajo licencia MIT (X11). Los términos se muestran a continuación.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

8.5 Manual de usuario

8.5.1 Primeros pasos

A continuación se muestran los primeros pasos para descargar, instalar y poner en marcha Apolo.

8.5.1.1 Adquisición

Apolo se puede adquirir desde la siguiente dirección:

http://actualizarapolo.p.ht/Setupx64.exe para SO de 64 bits.

http://actualizarapolo.p.ht/Setupx86.exe para SO de 32 bits.

8.5.1.2 Instalación

A la hora de instalar Apolo se debe tener en cuenta que existen dos versiones, una para sistemas operativos de 32 bits y otra para 64 bits. Si usted tiene un S.O. de 32 bits, debe seleccionar *Apolo_instalador_x86*, si su S.O. es de 64 bits, debe seleccionar *Apolo_instalador_x64*.

En el ejemplo mostrado a continuación, se instalará la versión de 64 bits, no obstante, los pasos son los mismo. Primeramente se ejecuta el archivo denominado *Apolo_instalador_x64*.

Ilustración 1. Icono instalador Apolo.

Una vez abierto el instalador, aparecerán una serie de ventanas que indican los pasos para instalar Apolo. La primera ventana muestra la bienvenida, a continuación se debe pulsar en el botón siguiente. En la siguiente ventana se muestran los términos de la licencia los cuales se deben aceptar y pulsar en siguiente.

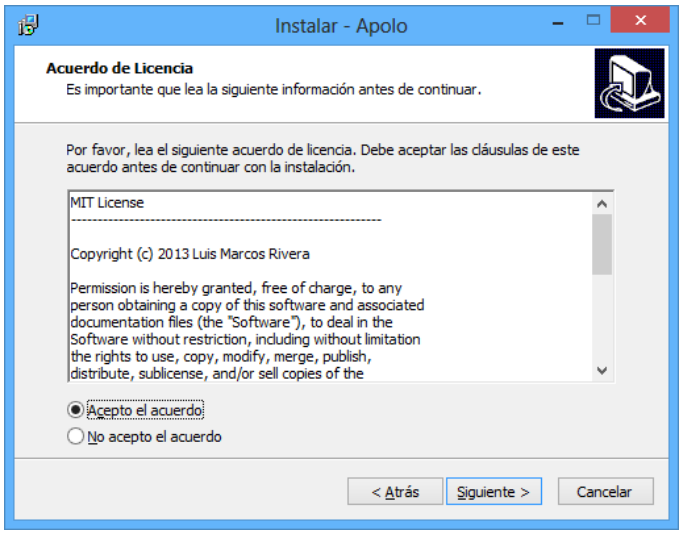

Ilustración 2. Acuerdo de licencia.

En el siguiente formulario se expone una característica básica que debe disponer el sistema operativo, y es que esté actualizado a la versión 4 de *Microsoft Framewok*.

Posteriormente, se selecciona la carpeta donde se quiere realizar la instalación. El siguiente paso da la opción de seleccionar el nombre del acceso directo y si se quiere crear uno en el escritorio.

Se debe pulsar en siguiente y aparecerá la opción de instalar.

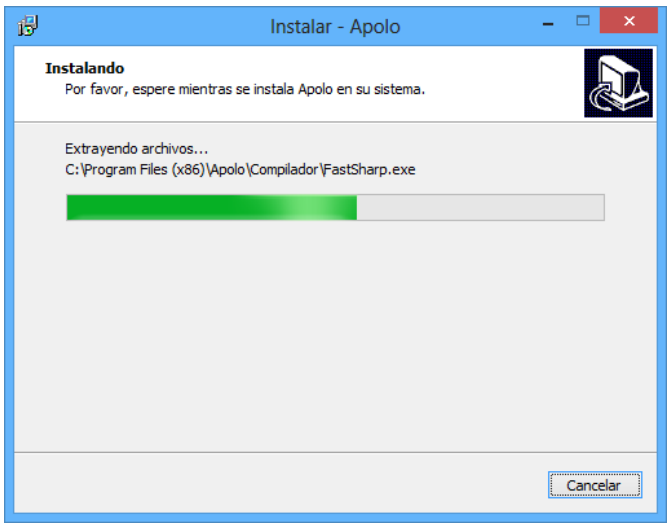

Ilustración 3. Progreso de la instalación.

Para finalizar, aparece la opción de ejecutar la aplicación, que se debe desmarcar puesto que la aplicación requiere permisos de administrador para ejecutarse.

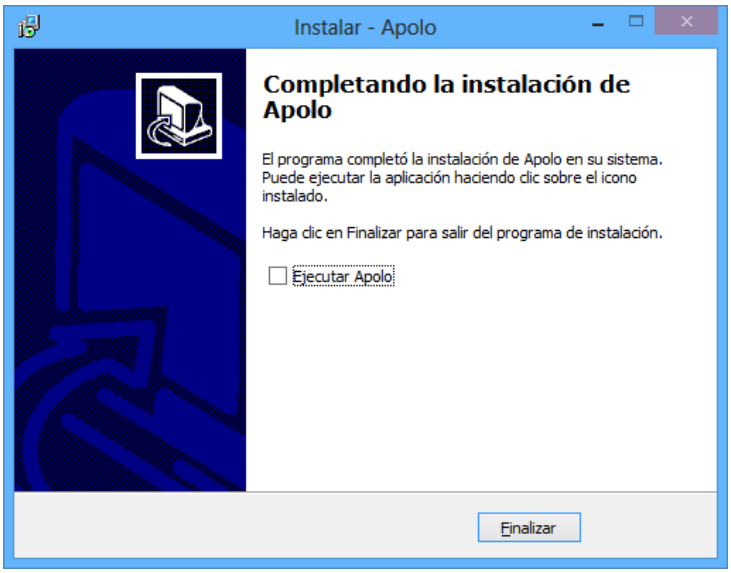

Ilustración 4.Paso final de la instalación.

Una vez instalada la aplicación, se localiza el acceso directo en el escritorio y se ejecuta.

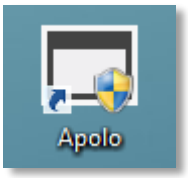

Ilustración 5. Icono Apolo.

Como se puede observar Apolo requiere privilegios de administrador para ejecutarse.

8.5.2 Aspecto general

En esta sección se mostrará cuál es el aspecto general de la aplicación, organización de los menús y secciones de la pantalla principal de Apolo. Primeramente, se va a mostrar una imagen que corresponde al formulario principal.

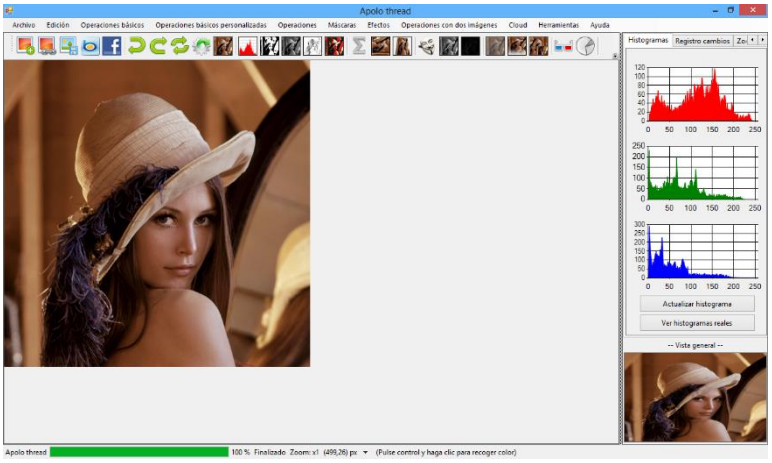

Ilustración 6. Aspecto general de Apolo.

Como se puede observar, Apolo tiene dos zonas claramente diferenciables. La zona de la izquierda donde se encuentra la imagen, y la zona de la derecha donde se puede observar información adicional.

En la parte superior, se halla el menú con todas las opciones y justo debajo de él, una serie de iconos con varios accesos rápidos a determinadas funcionalidades.

En la parte inferior se aloja una barra de estado, y otras opciones que determinan la posición y estado del ratón.

A continuación, se va a mostrar cada parte con detalle.

8.5.2.1 Imagen principal

Es la sección situada en la parte de la izquierda que muestra la imagen que actualmente se está manipulando. En la siguiente ilustración está remarcada con un recuadro de color rojo.

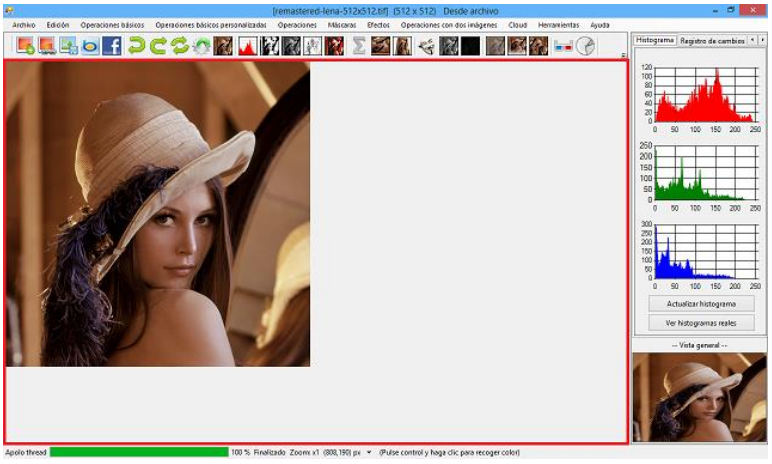

Ilustración 7. Imagen principal.

Tiene unas pocas pero interesantes funciones. La primera es que, si la imagen es mayor que la zona remarcada en el cuadro, se mostrarán uno *scroll* vertical/horizontal para poder desplazarse por toda la imagen (ver ilustración 8).

Para desplazarse por la imagen cuando ésta es mayor que el programa, se pueden mover directamente los *scrolls* o se puede hacer clic en cualquier zona de la imagen e ir arrastrando el ratón y automáticamente se moverán los *scrolls* para adaptarse a la nueva posición. Para moverse verticalmente también puede utilizarse la rueda del ratón.

Un acceso rápido para hacer zoom a la imagen es situarse en cualquier punto de la misma y manteniendo pulsada la tecla *control* de debe mover la rueda del ratón para aumentar o disminuir el tamaño de la imagen.

Pulsando (cuando se está situado encima de la imagen principal) la tecla *shift* y desplazando el ratón por la imagen, se muestra una ventana auxiliar con la zona ampliada.

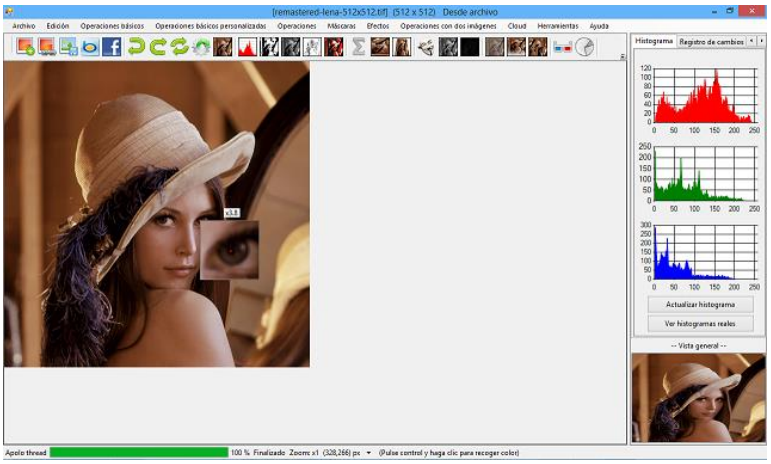

Ilustración 8. Zoom interactivo pulsando shift.

Para aumentar o disminuir el zoom de esta zona ampliada, simplemente hay que mover la rueda del ratón mientras se mantiene pulsado *shift*.

8.5.2.2 Barra lateral

Es la barra situada en la parte de la derecha, muestra valiosa información para comprender la imagen y alguna funcionalidad más. Primeramente se va a mostrar una imagen en detalle de ella.

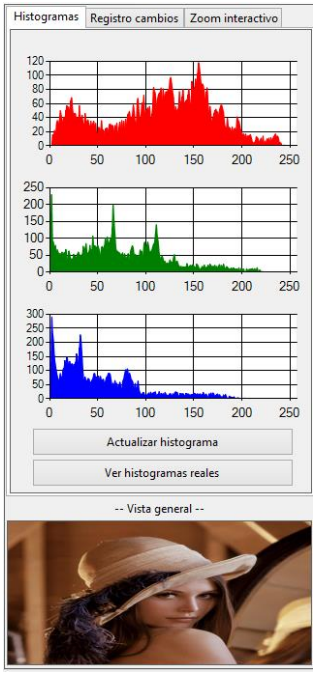

Ilustración 9. Detalle barra lateral.

Como se puede observar, en la parte superior hay tres opciones, histogramas, registro cambios y zoom interactivo.

8.5.2.2.1 Histogramas

En esta opción se muestra el histograma del canal rojo, verde y azul. Estos histogramas son aproximados, es decir, no contemplan el tamaño completo de la imagen, sino que se crean a partir de una reducción de la misma. No obstante, pulsando encima de cada uno se abre un nuevo formulario con el histograma real de la imagen o bien, haciendo clic en el botón inferior *Ver histogramas reales*.

8.5.2.2.2 Registro cambios

Si se hace clic en la segunda opción (*Registro cambios*), se puede observar todo el progreso de cambios en la imagen, además de una pequeña información de qué o cuál ha sido el cambio aplicado a la imagen. Si se pulsa en cualquiera de las imágenes mostradas en el registro, automáticamente pasa a ser la imagen actual en el programa principal. Para que se actualicen los cambios, simplemente hace falta cambiar de pestaña y volver a Registro de cambios.

Ilustración 10. Barra lateral, registro de cambios.

8.5.2.2.3 Zoom interactivo

La última opción, *Zoom interactivo*, muestra una imagen ampliada de la zona que se está seleccionando en la imagen principal. Es decir, si se quiere ver un detalle de la imagen, basta con seleccionar la pestaña Zoom interactivo y posteriormente situarse en la zona de la imagen que se desea ampliar.

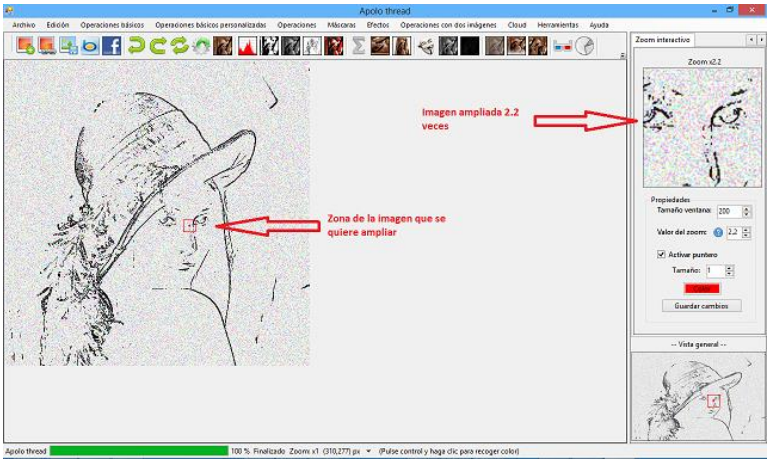

Ilustración 11. Zoom interactivo.

Dentro de esta sección, también hay una serie de opciones para manipular.

- Tamaño ventana: deja seleccionar el número de píxeles que tendrá el lado de la ventana zoom mostrada. A mayor tamaño, mayor superficie de la imagen mostrará.

- Valor del zoom: esta opción sirve para aumentar o disminuir el zoom. También se puede aumentar/disminuir pulsando la tecla *z* y moviendo la rueda del ratón.
- Activar puntero: Si esta opción se activa, se mostrará en la imagen un indicador de en qué zona se encuentra el ratón. A este indicador se le puede modificar el tamaño y color

Por último, una vez modificados los anteriores valores se debe pulsar en el botón Guardar cambios para que se actualicen.

8.5.2.3 Visión general

En la parte inferior de la barra, independientemente de cuál sea la opción seleccionada (*Histogramas*, *Registro cambios* o *Zoom interactivo*), se puede ver una imagen general de la imagen actual. Esta imagen no es meramente informativa, sino que tiene su función en imágenes que son mayores que el tamaño del programa principal. Cuando una imagen es mayor que el tamaño de la ventana, se muestran unos *scrolls* verticales y horizontales que permiten desplazarse, pero desde la imagen general (*visión general*), haciendo clic y arrastrando, se mueven los *scrolls* para ir a la zona seleccionada dentro de la imagen principal de Apolo.

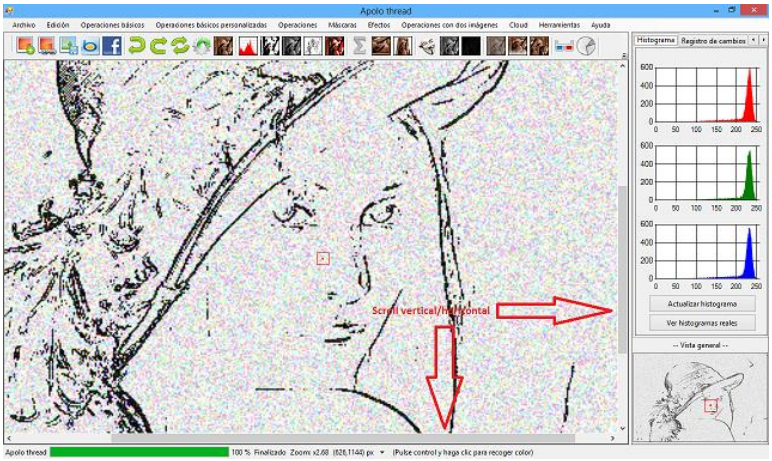

Ilustración 12. Visión general, detalle y scrolls.

Como se puede observar en la imagen anterior, al ser mayor que el tamaño de la ventana, se muestran los *scrolls* y pulsando en la imagen general (*Visión general*) y arrastrando, los *scrolls* se irán moviendo hasta la posición indicada, marcando además en ambas imágenes el punto seleccionado.

8.5.2.4 Barra de estado

Está situado en la parte inferior de Apolo y muestra una serie de información acerca del estado de la aplicación, posición del ratón, etc.

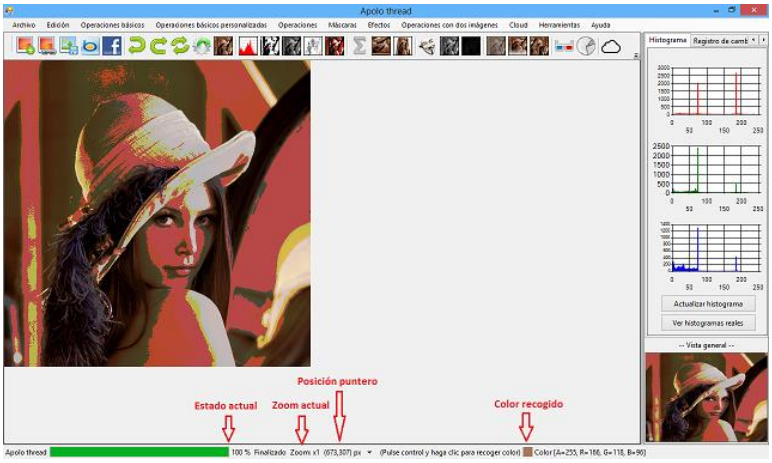

Ilustración 13. Detalle barra de estado.

A continuación se van a analizar las diferentes opciones con más detalle.

8.5.2.4.1 Estado actual

Muestra el estado actual en el que se encuentra el progreso. Tiene tres componentes:

- Barra progreso: típica barra de Windows con el progreso actual.
- Porcentaje progreso: en conjunción con la barra de progreso, muestra el porcentaje actual de la ejecución.
- Estado: indica qué se está realizando en el estado (momento) actual.

Cuando una función se está llevando a cabo, estos tres controles modifican su valor. En la siguiente ilustración se muestran los tres elementos mientras se está llevando a cabo una operación morfológica sobre la imagen (*Erosión*).

Ilustración 14. Barra de estado en progreso.

Como se puede observar el estado actual es un 26% de la función *Erosión*.

8.5.2.4.2 Zoom

Muestra el zoom actual con el que se está visionando la imagen.

8.5.2.4.3 Posición del cursor

Indica la posición actual del puntero con respecto a la imagen, determinando la coordenada X e Y. Como puede observarse en la siguiente imagen, se puede seleccionar entre 4 unidades de medida (pulgadas, centímetros, milímetros y píxeles).

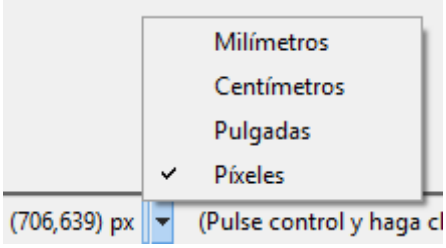

Ilustración 15. Detalle de coordenadas.

8.5.2.4.4 Color recogido

Si se mantiene pulsada la tecla *control* y se hace clic sobre la imagen, en la parte derecha de la barra de estado se muestra el color obtenido con sus cuatro componentes (*ARGB*).

8.5.2.5 Menú

En la parte superior se encuentra un *menustrip* con todas las opciones que tiene el programa. Dentro de cada rama hay otras sub-ramas para acceder a todas las funcionalidades. A continuación se muestra una imagen general de todos los menús.

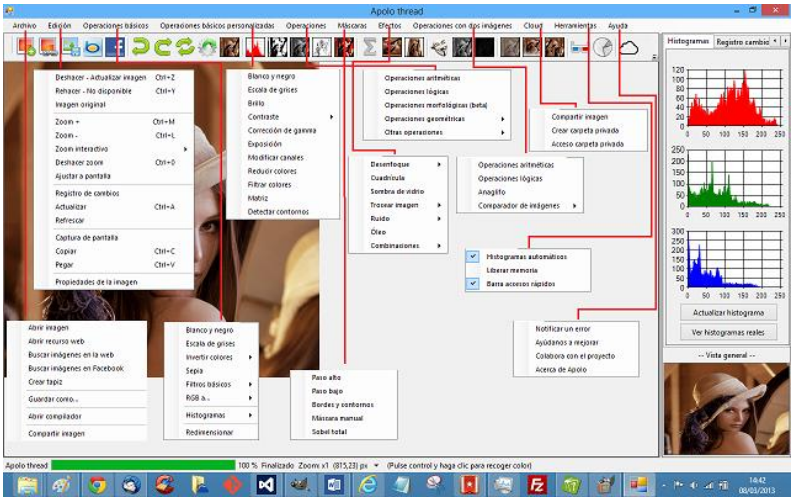

Ilustración 16. Menús de Apolo.

Estos menús pueden varias a lo largo del desarrollo de Apolo, por lo que tómese como una referencia más, en la versión de Apolo que usted esté ejecutando, puede que las opciones hayan variado.

8.5.2.6 Menú accesos rápidos

Justo debajo del menú anterior, se encuentran una serie de botones de acceso rápido a diferentes funciones del programa. En la siguiente imagen se muestra el detalle de los botones.

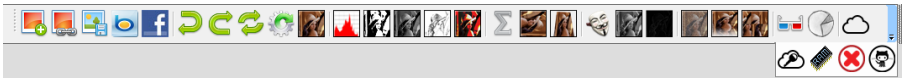

Ilustración 17. Barra de accesos rápidos.

Las funciones que incluye, ordenadas de izquierda a derecha son:

Abrir imagen, abrir recurso web, guardar como, abrir desde Bing, abrir desde Facebook, deshacer, rehacer, refrescar, actualizar, imagen original, histogramas, blanco y negro, escala de grises, detectar contornos, reducir colores, operaciones aritméticas, voltear, transformación afín, máscara manual, bordes y contornos, Sobel total, ruido, sombra de vidrio, anáglifo, comparar imágenes (vecinos), compartir imagen, sesión privada, liberar memoria, notificar error y colabora.

8.5.3 Aspectos básicos de una imagen

El concepto de imagen en Apolo se reduce a una matriz bidimensional cuyo tamaño es el ancho y alto de la imagen. Dentro de esa matriz, cada casilla se corresponde con un píxel. Cada píxel se compone de 4 canales; rojo, verde, azul y alfa. Y cada uno de esos canales puede tomar 256 valores diferentes.

8.5.3.1 Matriz bidimensional

Una imagen es un conjunto de píxeles, es decir, como una especie de malla donde cada cuadrado tiene un valor. Esta malla se determina por el alto y ancho de la imagen, es decir, una imagen cuya resolución es de 500 x 200 pixeles, se compondrá de 500 columnas y 200 filas, por lo tanto tendrá un total de 100000 píxeles.

8.5.3.2 Píxel

Un píxel es una casilla de la anteriormente citada malla, es decir, es la representación de un punto de la imagen en la pantalla del ordenador, y por lo tanto, puede considerarse como la unidad mínima homogénea en una imagen. Cada píxel está compuesto por 4 canales diferentes (rojo, verde, azul y alfa). Cada uno de estos canales tiene 256 valores (es decir, *2^8* valores), que van del 0 al 255. Cuando en un píxel uno de los canales tiene valor 0 quiere decir que dicho canal está apagado, es decir, si por ejemplo para un píxel el canal rojo es cero, ese píxel no tiene nada de rojo. En caso contrario, si el valor se torna a 255, quiere decir que ese canal tiene el máximo valor posible, siguiendo con el canal rojo, si es 255, quiere decir que no podría ser más rojo de lo que es.

- Cuando los canales *RGB* tienen valor 0, el color resultante es negro.
- Cuando los canales *RGB* tienen valor 255, el color resultante es blanco.
- Cuando el canal rojo tiene un valor 255 y *GB* tienen valor 0, el resultado es rojo puro.
- Cuando el canal verde tiene un valor 255 y *RB* tienen valor 0, el resultado es verde puro.

- Cuando el canal azul tiene un valor *255* y RG tienen valor 0, el resultado es azul puro.

La abreviatura de los canales se escribe *RGB* o *ARGB*, dependiendo si se incluye el canal alfa (Alfa-Red-Green-Blue).

8.5.3.3 ¿Qué representa el canal alfa?

Este canal representa el grado de transparencia del píxel, es decir, si el valor en el canal alfa es 255, el píxel se representará siendo totalmente opaco, en caso contrario (valor 0), el píxel será totalmente transparente.

Hay que tener en cuenta que no todos los formatos soportan transparencia en imágenes. Por ejemplo el formato *PNG* sí lo soporta, en cambio *JPG* no.

8.5.3.4 Formatos de imagen

Apolo trabaja directamente con mapas de bits, pero soporta varios formatos al abrir o exportar imágenes. Los formatos son BMP, GIF, JPEG, PNG y TIFF:

- BMP: Windows bitmap (.BMP) es el formato propio del programa, que viene con el sistema operativo Windows. Puede guardar imágenes de 24 bits (16,7 millones de colores), 8 bits (256 colores) y menos. Puede darse a estos archivos una compresión sin pérdida de calidad: la compresión RLE (Run-length encoding).
- GIF: (Compuserve GIF) es un formato gráfico utilizado ampliamente en la World Wide Web, tanto para imágenes como para animaciones.
- JPEG: (del inglés Joint Photographic Experts Group, Grupo Conjunto de Expertos en Fotografía) es el nombre de un comité de expertos que creó un estándar de compresión y codificación de archivos de imágenes fijas. Este comité fue integrado desde sus inicios por la fusión de varias agrupaciones en un intento de compartir y desarrollar su experiencia en la digitalización de imágenes. El formato JPEG utiliza habitualmente un algoritmo de compresión con pérdida para reducir el tamaño de los archivos de imágenes.
- PNG: (siglas en inglés de Gráficos de Red Portátiles, pronunciadas "ping") es un formato gráfico basado en un algoritmo de compresión sin pérdida para Bitmaps no sujeto a patentes. Este formato fue desarrollado en buena parte para solventar las deficiencias del formato GIF y permite almacenar imágenes con una mayor profundidad de contraste y otros importantes datos.
- TIFF: TIFF es un formato de archivo informático para imágenes. Las etiquetas también describen el tipo de compresión aplicado a cada imagen, que puede ser: Sin compresión PackBits Compresión Huffman modificado, el mismo que las imágenes de fax.

8.5.4 Referencia de funciones

Esta es la sección donde se muestran todas las funciones y capacidades de Apolo.

8.5.4.1 Archivo

Primer elemento de la barra superior. A continuación se muestra una imagen del menú.

Ilustración 18. Menú archivo.

8.5.4.1.1 Abrir imagen

Pulsando en la opción *Abrir imagen*, se muestra un cuadro de diálogo en el que se puede seleccionar una imagen.

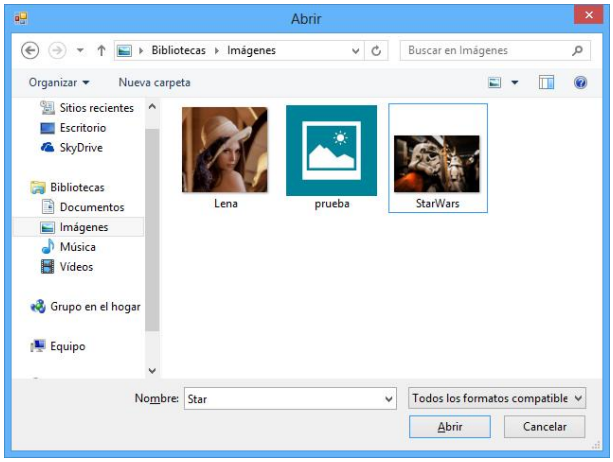

Ilustración 19. Abrir imagen desde archivo.

Los formatos admitidos son, BMP, GIF, JPEG, PNG y TIFF.

8.5.4.1.2 Abrir recurso web

Esta opción permite cargar directamente una URL que únicamente contenga una imagen en un formato compatible.

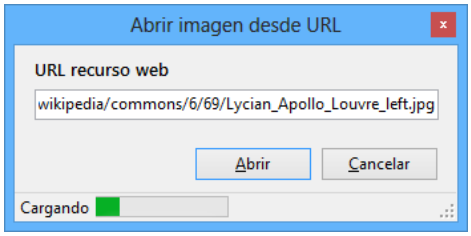

Ilustración 20. Abrir recurso web.

En la parte inferior muestra una barra de carga e información del estado (cargando, recurso no encontrado, etc.).

8.5.4.1.3 Buscar imágenes en la web

Con esta opción se puede buscar directamente imágenes en Internet. Las búsquedas de recursos se procesan mediante el API de Bing imágenes.

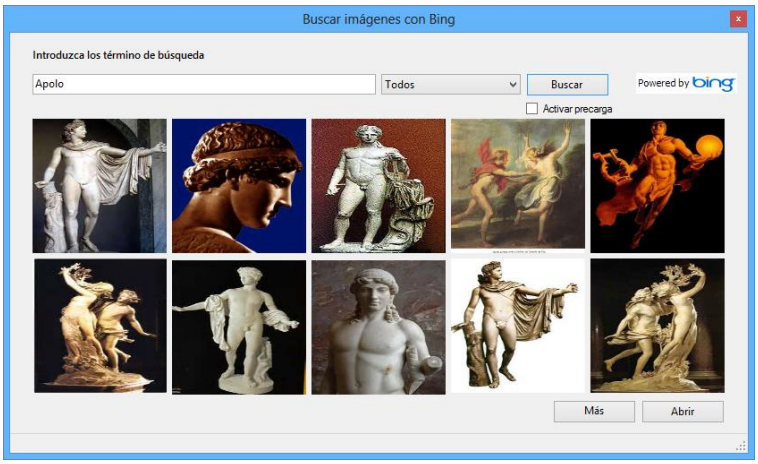

Ilustración 21. Buscar imágenes en la web.

Hay que introducir el/los término/s a buscar, seleccionar en el desplegable situado a la derecha del cuadro de búsqueda, el tamaño de la imagen y pulsar en el botón Buscar. Adicionalmente, se puede activar la casilla Activar precarga, y no se cargará una vista previa de las imágenes (miniatura), sino la imagen a tamaño real.

Una vez cargadas las imágenes, basta con hacer clic encima de ellas y pulsar en el botón *Abrir* y, tras unos instantes (en caso de estar activada la precarga es instantáneo), se verá la imagen en la aplicación principal.

Con el botón *Más*, se verán las siguientes 10 imágenes estando disponibles hasta un máximo de 50 imágenes por búsqueda.

8.5.4.1.4 Buscar imágenes en Facebook

Esta opción permite iniciar sesión con una cuenta de Facebook y poder abrir directamente las imágenes en las que se esté *etiquetado*.

Primeramente se muestra un formulario con el inicio de sesión de Facebook. Una vez introducidos los datos (correo electrónico y contraseña), se debe pulsar en el botón *Entrar*.

Ilustración 22. Inicio sesión Facebook.

Una vez se ha iniciado sesión, basta con hacer clic en la imagen que se quiera abrir y pulsar en el botón *Abrir*. Tras unos instantes, se cargará en el formulario principal.

Ilustración 23. Abrir imágenes de Facebook.

Pulsando en el botón Siguiente se muestran las siguientes imágenes, y pulsando Inicio se vuelve a las primeras 10.

8.5.4.1.5 Crear tapiz

Muestra un cuadro de diálogo para crear un tapiz rectangular con las dimensiones, color y nombre elegidas por el usuario.

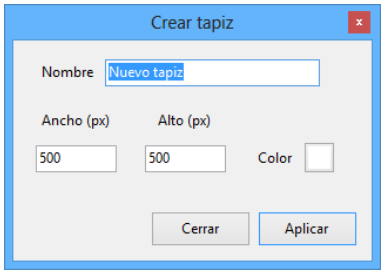

Ilustración 24. Crear tapiz.

En el botón del color, se despliega el cuadro de diálogo de Windows para seleccionar o crear un nuevo color.

8.5.4.1.6 Guardar como

Pulsando en la opción Guardar como, se muestra un cuadro de diálogo en el que se puede guardar una imagen en la ruta especificada.

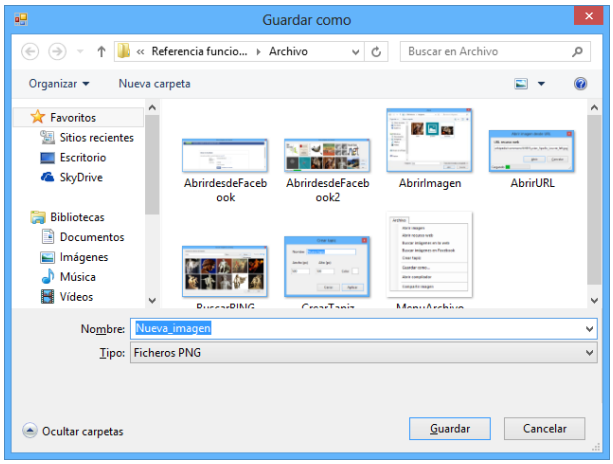

Ilustración 25. Guardar como.

Los formatos admitidos son, BMP, GIF, JPEG, PNG y TIFF.

8.5.4.1.7 Abrir compilador

Abre una ventana con un compilador para Visual Basic .NET. La aplicación originalmente ha sido creada por Matthew Manela y disponible en Github para su descarga. Se ha adaptado la aplicación para el fin último de Apolo, crear algoritmos para tratamiento de imágenes.

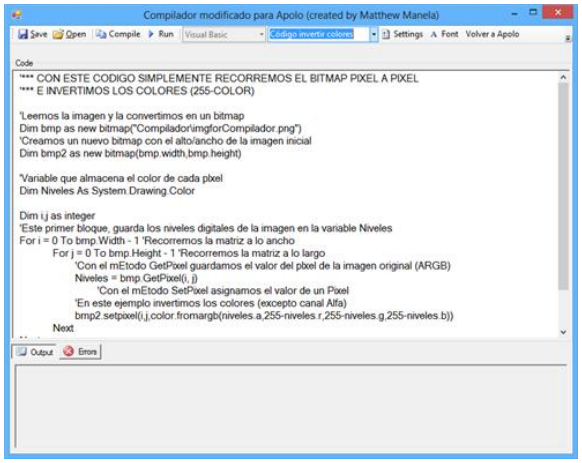

Ilustración 26. Compilador Visual Basic .NET.

En la parte superior hay un desplegable para seleccionar algoritmos para diferentes efectos, como son, invertir colores, aumentar brillo, etc. Siguiendo el esquema de cualquiera de los algoritmos, se pueden modificar los colores de forma muy fácil, y, una vez creado el código fuente, se debe pulsar en el botón superior denominado *Run*.

Al salir del compilador se mostrará una ventana con las imágenes almacenadas en el directorio del compilador.

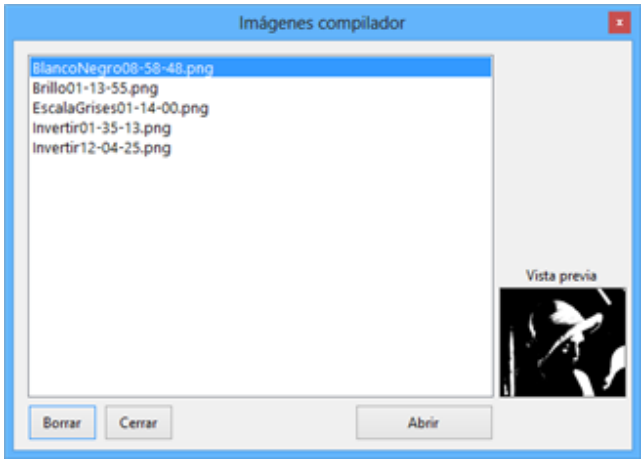

Ilustración 27. Abrir imágenes compilador.

En ocasiones, la aplicación se quedará congelada unos instantes hasta que muestra esta ventana. En caso de que no llegue a cargase correctamente se debería notificarlo como error en la sección *Ayuda*.

8.5.4.1.8 Compartir imágenes

Se trata de un servidor público donde poder compartir y ver las imágenes que suban los usuarios. Estás imágenes deben subirse con una pequeña descripción y su nombre.

Opcionalmente hay una sección para poder valorar las imágenes y así dar una opinión de la misma.

Una vez se ha pulsado sobre la opción Compartir imágenes, aparece un pequeño recuadro que indica que se está estableciendo conexión. Tras unos segundos se despliega una ventana donde poco a poco se irán cargando las 10 primeras imágenes disponibles.

Ilustración 28. Apolo Cloud.

En la parte izquierda se puede subir la imagen con su nombre y descripción. Pulsando en el botón Imagen actual se abrirá la imagen que actualmente está en la aplicación principal, para así poder compartirla. Desplegando el menú completo (pulsando en *Ver listado completo de imágenes*), se puede ver un listado con todas las imágenes del servidor y la posibilidad de valorarlas.

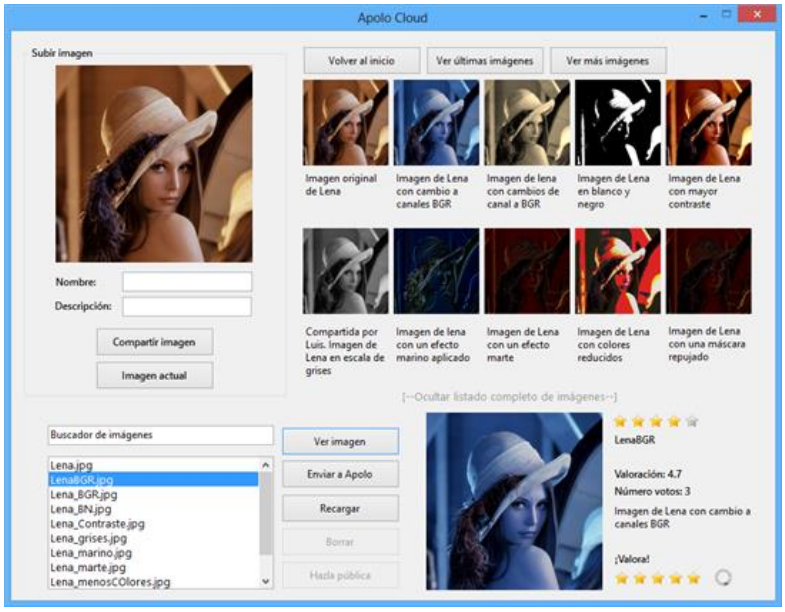

Ilustración 29. Valorar imagen.

Como se puede observar en la imagen, se está valorando con 5 estrellas la imagen actual (*LenaBGR*), cuya puntuación actual es de 4.7 estrellas.

Esta sección se desarrolla con más detalle en el apartado *Cloud*.

8.5.4.2 Edición

Segundo elemento de la barra superior. A continuación se muestra una imagen del menú.

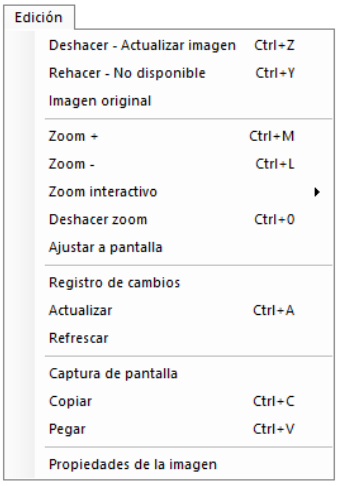

Ilustración 30. Menú edición.

8.5.4.2.1 Deshacer

Con esta opción se retrocede a la anterior imagen modificada. Además, muestra información de qué modificación se va a deshacer.

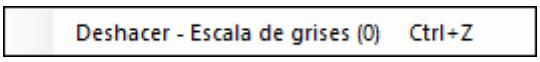

Ilustración 31. Deshacer escala de grises.

Como se puede observar en la imagen anterior, el proceso que se va a deshacer es una transformación en escala de grises.

8.5.4.2.2 Rehacer

Con esta opción se avanza a la siguiente imagen modificada en caso de haber retrocedido previamente. Además, muestra información de qué modificación se va a rehacer.

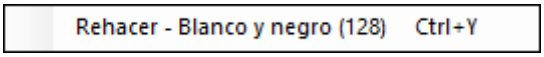

Ilustración 32. Rehacer blanco y negro.

Como se puede observar en la imagen anterior, el proceso que se va a rehacer es una transformación en blanco y negro.

8.5.4.2.3 Imagen original

Se muestra la imagen actual sin ninguna transformación, es decir, la imagen actual es mostrada como cuando se abrió por primera vez. La forma para abrir imágenes puede ser, desde archivo, desde BING, como recurso web o desde Facebook.

8.5.4.2.4 Zoom +

Se amplía en un 10% el tamaño de la imagen mostrada en ese momento.

Zoom –

Se disminuye en un 10% el tamaño de la imagen mostrada en ese momento. Hasta un mínimo máximo permitido. En caso de llegar al mínimo se mostrará un cuadro de diálogo informando de ello.

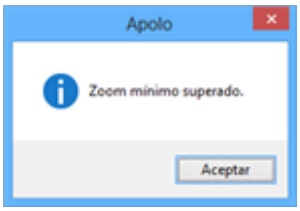

Ilustración 33. Zoom mínimo superado.

8.5.4.2.5 Zoom interactivo

Se disgrega en dos opciones, empezar y propiedades. En la opción empezar, se muestra un cuadro de diálogo informando de cómo utilizar esta opción. El proceso para utilizar este zoom es sencillo, simplemente se debe mantener pulsada la tecla SHIFT mientras se desplaza el cursor por la imagen. En ese momento se mostrará un recuadro adicional al lado del puntero con el fragmento de la imagen ampliada. Además, mientras se mantiene pulsada la tecla *shift*, si se da vueltas a la rueda del ratón se aumenta o disminuye el nivel de zoom.

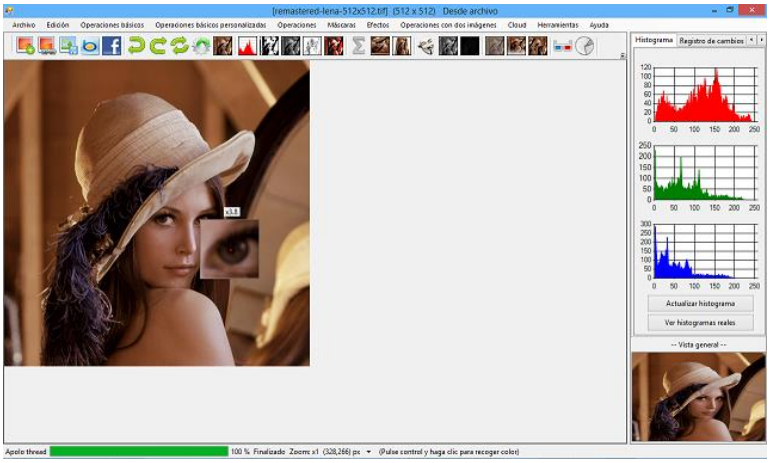

Ilustración 34. Zoom interactivo.

Si se pulsa la opción propiedades, se muestra un cuadro de diálogo con las siguientes posibilidades:

- Tamaño de la ventana: deja seleccionar el número de píxeles que tendrá el lado de la ventana zoom mostrada. A mayor tamaño, mayor superficie de la imagen se mostrará.
- Valor del zoom: esta opción sirve para aumentar o disminuir el zoom de la región mostrada. También se puede aumentar/disminuir pulsando la tecla *shift* y moviendo la rueda del ratón.
- Activar puntero: Si esta opción se activa, se mostrará en la imagen un indicador de en qué zona se encuentra el ratón. A este indicador se le puede modificar el tamaño y color
- Etiqueta zoom: si se activa esta opción se muestra en la parte superior de la ventana un texto indicando el nivel de zoom.

8.5.4.2.6 Deshacer zoom

Devuelve la imagen a su estado de zoom original (*x1*).

8.5.4.2.7 Ajustar a pantalla

Aumenta el ancho y alto de la imagen para que se ajuste a la zona del programa donde se muestra la imagen.

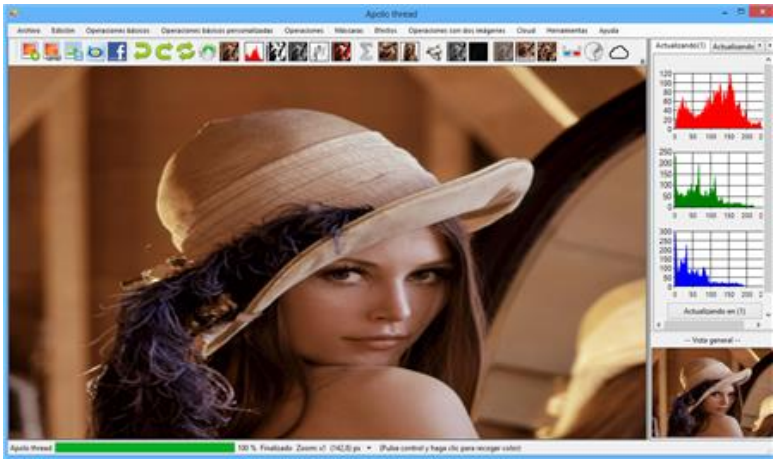

Ilustración 35. Ajustar a pantalla.

Como se puede observar en la imagen anterior, la fotografía de Lena ahora ocupa toda la zona principal del programa.

8.5.4.2.8 Registro de cambios

Si se hace clic en esta opción (*Registro de cambios*), se puede observar todo el progreso de cambios en la imagen, además de una pequeña información de qué o cuál ha sido el cambio aplicado a la imagen. Si se pulsa en cualquiera de las imágenes mostradas en el registro, automáticamente pasa a ser la imagen actual en el programa principal. Para que se actualicen los cambios, simplemente hace falta cerrar y volver a abrir la ventana.

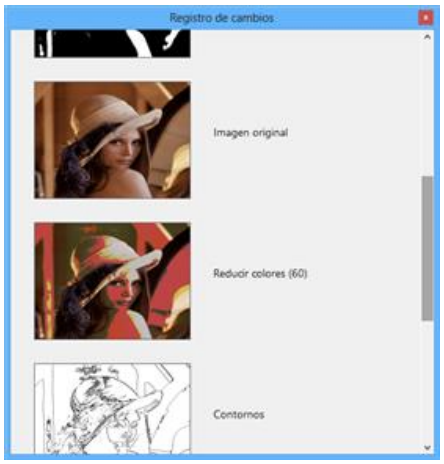

Ilustración 36. Registro de cambios.

8.5.4.2.9 Actualizar

Al pulsar esta opción, hace que la imagen mostrada actualmente en Apolo sea la primera del registro de imágenes. Imagínese que está retrocediendo su imagen a un estado anterior (deshacer) y quiere que la imagen actual pase a ser la primera en el orden de rehacer/deshacer, simplemente pulsando actualizar imagen pasará a serlo. Además, se actualizarán los histogramas laterales.

8.5.4.2.10 Refrescar

Actualiza el tamaño y posición de los *scrolls* verticales y horizontales de la imagen. En caso de cualquier fallo con los *scrolls*, pulsando esta opción se volverá a calcular su valor y a mostrarse en pantalla.

8.5.4.2.11 Captura de pantalla

Crea una captura de pantalla del estado actual, y la asigna al programa principal.

8.5.4.2.12 Copiar

Hace que la imagen actual de Apolo pase al portapapeles de Windows.

8.5.4.2.13 Pegar

Al seleccionar esta opción, si existe alguna imagen en el portapapeles de Windows, pasará a ser la imagen actual de Apolo.

8.5.4.2.14 Propiedades de la imagen

Muestra un formulario con dos pestañas con toda la información disponible de la imagen.

En la primera pestaña (General) se muestra la siguiente información:

Nombre de la imagen, tamaño en píxeles, tamaño en pulgadas, resolución, obtención imagen, formato de la imagen, número de píxeles, perfil de color.

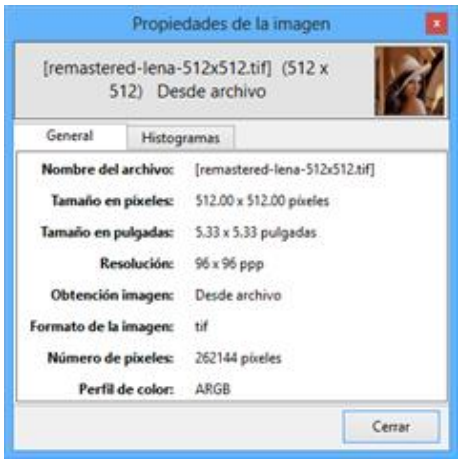

Ilustración 37. Propiedades de la imagen. General.

En la pestaña Histogramas, se muestra el histograma (rojo, verde y azul) de la miniatura de la imagen. En caso de querer visualizar los histogramas completos, basta con hacer clic en cualquiera de ellos.

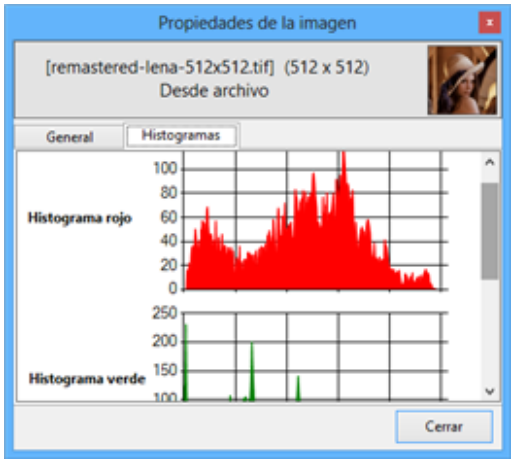

Ilustración 38. Propiedades de la imagen. Histogramas.

8.5.4.3 Operaciones básicas

Tercer elemento de la barra de herramientas. A continuación, se muestra una imagen.

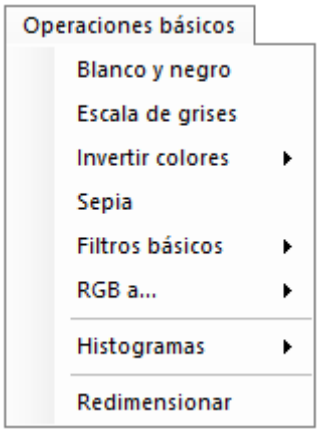

Ilustración 39. Menú operaciones básicas.

En este menú puede encontrar las funciones básicas para modificar en una imagen. En esta guía, se va a utilizar de ejemplo la mítica imagen de Lena, a continuación se observa en su estado original.

Ilustración 40. Imagen original Lena.

8.5.4.3.1 Blanco y negro

Binariza la imagen, es decir, cada píxel lo pasa a color blanco o negro. Esta función es muy sencilla, simplemente calcula la media de color para cada píxel (*(ColorRojo+ColorVerde+ColorAzul)/3*) y en función del valor de salida de esta media, el píxel pasa a ser blanco o negro. Si la media es mayor o igual de 128, el píxel pasa a tener en sus tres canales (rojo, verde, azul) un valor de 255 (blanco), en caso contrario pasa a tener un valor 0 en todos sus canales (negro).

Ilustración 41. Imagen de Lena en blanco y negro.

La imagen se ha binarizado y todos los valores son negros o blancos.

8.5.4.3.2 Escala de grises

Con esta función se pasa a tener una imagen en escala de grises, es decir, cada píxel tiene un tono de gris. Los tonos de gris se caracterizan por tener el mismo valor en todos los canales (RGB), siendo el menor valor el color negro (0, 0, 0) y el mayor el blanco (255, 255, 255).

Para aplicar esta función hay que calcular, para cada píxel, la media de sus tres componentes (*RGB*) y dicha media aplicarla a los tres canales en el píxel tratado. Si se tiene un píxel con valores; Rojo = 100, Verde = 200, Azul= 50, se debe calcular la media (*(100+200+50)/3*) y aplicar el valor resultante a los tres canales, por lo tanto, el píxel resultante tendría los siguientes valores para cada canal; Rojo = (100+200+50)/3, Verde = (100+200+50)/3, Azul = (100+200+50)/3, haciendo el cálculo sería, Rojo = 116, Verde = 116, Azul = 116.

En la siguiente ilustración, se puede observar la imagen de Lena en escala de grises.

Ilustración 42. Imagen de Lena en escala de grises.

La imagen ha pasado de tener diferentes tonalidades de gris, es decir, sus valores oscilan entre 0, 0, 0 y 255, 255, 255, siendo los tres canales *RGB* iguales.

8.5.4.3.3 Invertir colores (RGB/Rojo/Verde/Azul)

Esta función calcula la diferencia entre 255 (valor máximo de un píxel) y el valor actual, es decir, invierte los colores. Un ejemplo claro es tomar un píxel de color blanco, cuyos valores serían (para los canales *RGB*) de 255. Si se invierte el color, hay que restar 255 menos el valor del píxel actual que como es 255, daría un resultado de 0 y el píxel pasaría a ser negro.

Tiene 4 variantes esta función, RGB, Rojo, Verde, Azul.

- RGB: Invierte los colores en los 3 canales RGB.
- Rojo: Invierte sólo el canal rojo manteniendo el verde y azul con su valor original.
- Verde: Invierte sólo el canal verde manteniendo el rojo y azul con su valor original.
- Azul: Invierte sólo el canal azul manteniendo el rojo y verde con su valor original.

Ilustración 43. Imagen de Lena invertida.

Los valores de los tres canales se han invertido y lo que antes era oscuro ahora es claro.

8.5.4.3.4 Sepia

Crea una imagen modificando sus píxeles pasando a tener tonos sepia. Para poder realizar esta transformación aplica a cada canal un valor calculado en función de unos pesos específicos.

- Rojo = Rojo * Peso1 + Verde * Peso2 + Azul * Peso3
- Verde = Rojo * Peso4 + Verde * Peso5 + Azul * Peso6
- Azul = Rojo * Peso7 + Verde * Peso8 + Azul * Peso9

Es recomendable que la suma de los pesos para cada color se igual a 1, por ejemplo, para el canal Rojo la suma de Peso1 + Peso2 + Peso3 debería ser igual a 1 si no se quiere alterar demasiado los valores. No obstante es opcional, se hace una comprobación previa para determinar que el valor sea correcto y no mayor de 255.

Los valores para los tonos sepia son los recomendados por Microsoft y serían, 0.393, 0.769, 0.189, 0.349, 0.686, 0.168, 0.272, 0.534, 0.131.

Ilustración 44. Imagen de Lena en tonos sepia.

La imagen se ha transformado a tonos sepia.

8.5.4.3.5 Filtros básicos (Rojo/Verde/Azul)

Estos filtros calculan la media para cada píxel (*(ColorRojo+ColorVerde+ColorAzul)/3*) y el valor resultante lo asignan al canal al que se quiere aplicar el filtro y valor 0 para los dos canales restantes.

Tiene 3 variantes, filtro rojo, filtro verde y filtro azul:

- Filtro rojo: Calcula la media, para cada píxel, de los tres canales y asigna el resultado al canal rojo, mientras que al canal verde y azul les asigna valor 0.
- Filtro verde: Calcula la media, para cada píxel, de los tres canales y asigna el resultado al canal verde, mientras que al canal rojo y azul les asigna valor 0.
- Filtro azul: Calcula la media, para cada píxel, de los tres canales y asigna el resultado al canal azul, mientras que al canal verde y rojo les asigna valor 0.

Ilustración 45. Imagen de Lena con filtro rojo.

Se aprecia que la imagen ahora únicamente tiene colores rojos, es algo así como una escala de grises pero sólo de rojos.

8.5.4.3.6 RGB a… (BGR/GRB/RBG)

Esta función intercambia, para cada píxel, el valor de sus canales. Tiene tres variantes, BGR, GRB, RBG:

- RGB a BGR: Para cada píxel, el canal rojo pasa a tener el valor del canal azul, el canal azul pasa a tener el valor del canal rojo, y el canal verde mantiene su valor.
- RGB a GRB: Para cada píxel, el canal rojo pasa a tener el valor del canal verde, el canal verde pasa a tener el valor del canal rojo, y el canal azul mantiene su valor.
- RGB a RBG: Para cada píxel, el canal verde pasa a tener el valor del canal azul, el canal azul pasa a tener el valor del canal verde, y el canal rojo mantiene su valor.

Ilustración 46. Imagen de Lena RGB.

Como puede observarse la imagen se ha tornado a tonos morados tras alterar la composición de sus canales.

8.5.4.3.7 Histogramas

Muestra una serie de gráficas con el histograma acumulado de la imagen. El histograma acumulado no es más que el número de píxeles que tiene un determinado valor. Por ejemplo, para el canal rojo y el valor 50, sería el número de píxeles que tiene valor 50 en el canal rojo.

Dentro de los histogramas se puede seleccionar 5 formas de visualización: área (*spline*), área, columnas, línea (*spline*), línea. Además, el tamaño del histograma se adapta al tamaño de la ventana, es decir, si se quiere visualizar el histograma en un gran tamaño, basta con maximizar la ventana y se adaptará para que se vea más grande.

En la gráfica se muestra el valor del píxel en el eje de las X, y el número de píxeles en el eje de las Y.

Hay dos modalidades de visualizarlos:

- Detallado: Muestra el histograma individual, pudiendo seleccionar el canal rojo, verde, azul, escala de grises (media de los canales *RGB*) y canal alfa.

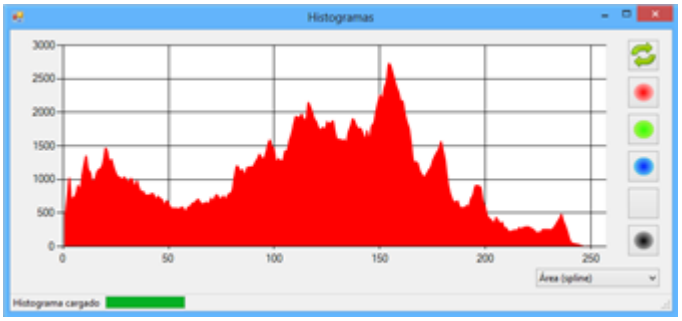

Ilustración 47. Histograma detallado del canal rojo.

- Todos: muestra los histogramas para los canales rojo, verde, azul y escala de grises en una misma ventana.

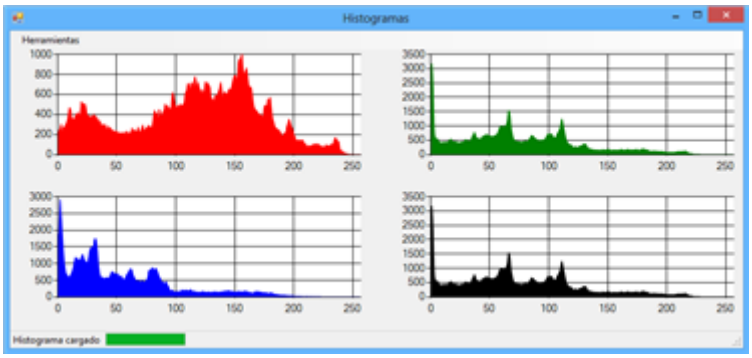

Ilustración 48. Todos los histogramas.

8.5.4.3.8 Redimensionar

Esta función posibilita cambiar el ancho y alto de una imagen. Esta modificación se permite hacer asignando directamente el ancho y alto o el porcentaje de ancho/alto que se quiere aumentar o disminuir.

Además, incluye la capacidad para seleccionar el tipo de interpolación con el que se quiere realizar el escalado de la imagen. Los diferentes tipos de interpolación son:

- Default: especifica el modo predeterminado.
- Low: especifica interpolación de calidad baja.
- High: especifica interpolación de calidad alta.
- Bilinear: especifica interpolación bilineal. No se ha aplicado ningún filtro previo. Este modo no es adecuado para comprimir una imagen hasta por debajo de un 50% de su tamaño original.
- Bicubic: especifica interpolación bicúbica. No se ha aplicado ningún filtro previo. Este modo no es adecuado para comprimir una imagen hasta por debajo de un 25% de su tamaño original.
- NearestNeighbor: especifica interpolación de elemento más cercano.
- HighQualityBilinear: especifica una interpolación bilineal de gran calidad. Se aplica un filtro previo para garantizar una compresión de gran calidad.
- HighQualityBicubic: especifica una interpolación bicúbica de gran calidad. Se aplica un filtro previo para garantizar una compresión de gran calidad. Este modo genera imágenes transformadas de la más alta calidad.

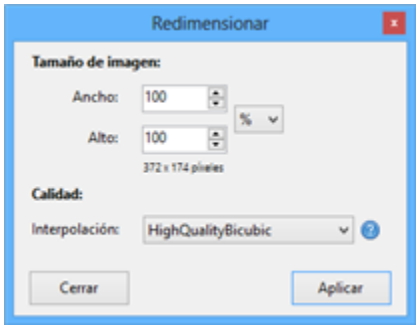

Ilustración 49. Menú de redimensionamiento de imagen.

8.5.4.4 Operaciones básicas personalizadas

Es la cuarta opción del menú superior. A continuación se muestra una imagen del menú.

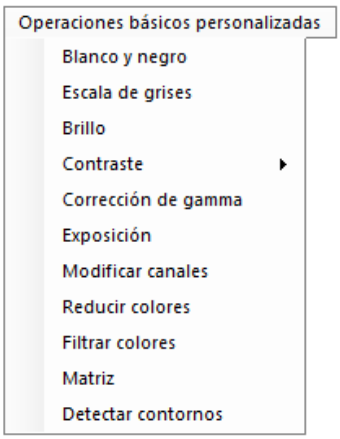

Ilustración 50. Menú de operaciones básicas personalizadas.

En este menú se encuentran las funciones básicas para modificar en una imagen pero, a diferencia del menú anterior, éstas pueden modificarse de forma manual. En esta guía, se va a utilizar de ejemplo la mítica imagen de Lena, a continuación se puede observar en su estado original.

Ilustración 51. Imagen original de Lena.

8.5.4.4.1 Blanco y negro

Binariza la imagen, es decir, cada píxel lo pasa a color blanco o negro. Esta función es muy sencilla, simplemente calcula la media de color para cada píxel (*(ColorRojo+ColorVerde+ColorAzul)/3*) y en función del valor de salida de esta media, el píxel pasa a ser blanco o negro.

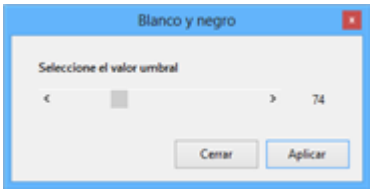

Ilustración 52. Menú blanco y negro.

Si la media es mayor o igual del valor seleccionado por el usuario (valor umbral), el píxel pasa a tener en sus tres canales (rojo, verde, azul) un valor de 255 (blanco), en caso contrario pasa a tener un valor 0 en todos sus canales (negro).

Ilustración 53. Imagen de Lena en blanco y negro (umbral 74).

La imagen se ha binarizado y todos los valores mayores de 74 son blancos y los menores son negros.

8.5.4.4.2 Escala de grises

Con esta función se pasa a tener una imagen en escala de grises, es decir, cada píxel tiene un tono de gris. Los tonos de gris se caracterizan por tener el mismo valor en todos los canales (*RGB*), siendo el menor valor el color negro (0, 0, 0) y el mayor en blanco (255, 255, 255).

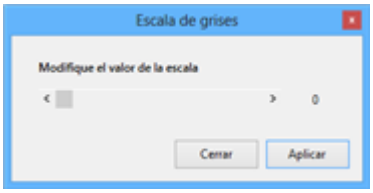

Ilustración 54. Menú escala de grises.

Para aplicar esta función hay que calcular, para cada píxel, la media de sus tres componentes (*RGB*) y dicha media aplicarla a los tres canales en el píxel tratado pero teniendo en cuenta el valor seleccionado por el usuario. Si se tiene un píxel con valores; Rojo = 100, Verde = 200, Azul= 50, se debe calcular la media (*(100+200+50)/3*) y aplicar el valor resultante a los tres canales, por lo tanto, el píxel resultante tendría los siguientes valores para cada canal; Rojo = (100+200+50)/3, Verde = (100+200+50)/3, Azul = (100+200+50)/3, haciendo el cálculo sería, Rojo = 116, Verde = 116, Azul = 116, y en función del valor seleccionado por el usuario, se calculará el valor de salida. Realmente lo que se hace es reducir el número de tonos de grises, desde la gama completa (con valor igual a 0), hasta blanco y negro (con valor igual a 127).

En la siguiente ilustración, se puede observar la imagen de Lena en escala de grises con un valor de 50.

Ilustración 55. Imagen de Lena en escala de grises (valor 50).

La imagen se ha pasado a escala de grises, pero una escala de grises reducida, es decir, no contiene los 256 valores diferentes de gris.

8.5.4.4.3 Brillo

En el apartado brillo se puede aumentar o disminuir la intensidad de los píxeles. Al aumentar, por ejemplo, en diez puntos el brillo, a cada canal de cada píxel se le suma 10. En caso de que el valor sea mayor de 255 o menor de 0, se le asigna 255 o 0 respectivamente a cada canal.

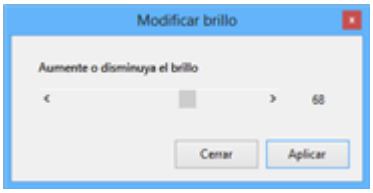

Ilustración 56. Menú modificar brillo.

En la siguiente ilustración se puede observar la imagen de Lena a la que se le ha aumentado 68 puntos.

Ilustración 57. Imagen de Lena con brillo aumentado (68).

De la imagen anterior se saca la conclusión de que la imagen se ha aclarado, es decir, sus valores son más cercanos al blanco (255).

8.5.4.4.4 Contraste (Contraste 1/Contraste 2)

Al aumentar el contraste de una imagen se aumenta la diferencia entre los diferentes colores, mientras que si se disminuye, se atenúan las diferencias.

En Apolo hay dos funciones para modificar el contraste:

Contraste 1 (recomendado): con esta opción se obtienen excelentes resultados a la hora de dar mayor contraste a imágenes para visualizar mejor los detalles. Se le debe

proporcionar un valor de entrada a partir del cual se calculará el valor de salida para cada píxel. Se le suma una unidad al valor de entrada y se divide entre la cuarta parte del número *Pi*, a este valor se le denomina *ValorCalculado*. Si se selecciona el canal rojo de un píxel cualquiera, llamémosle, *ValorRojo*, y se le aplica la siguiente modificación, *ValorRojo= ((ValorRojo – 128) * ValorCalculado) + 128*, se obtiene el valor de salida del canal rojo de ese píxel.

Aplicando esto a todos los píxeles y los tres canales, se obtiene la imagen de salida.

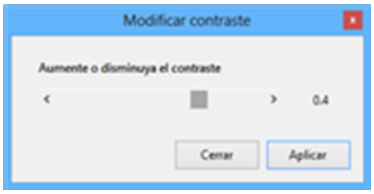

Ilustración 58. Menú modificar contraste.

A continuación se muestran dos histogramas, siendo el de la izquierda el de la imagen original y el de la derecha tras una modificación de 1.4.

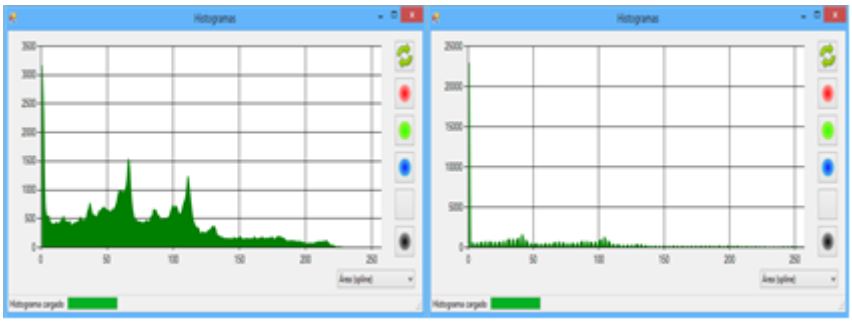

Ilustración 59. Diferencia histogramas.

Como se ve claramente, se han estirado los valores para adaptarse en mayor medida a todo el rango de valores.

En la siguiente imagen puede verse el resultado de aplicar 0,4 puntos de contraste en la imagen de Lena.

Ilustración 60. Imagen de Lena más contrastada (0,4 puntos).

Comparada con la imagen original, puede apreciarse que los colores claros son más claros y los oscuros han aumentado sustancialmente.

Contraste 2: esta función estira los valores hasta llegar a los extremos seleccionados. El menú tiene una opción de seleccionar un valor mínimo y un valor máximo. El valor mínimo es el valor más bajo que se quiere que tenga el píxel con menor valor, y el valor máximo es lo contrario. Estos valores se analizan por canal, es decir, se calculará previamente el valor de intensidad más bajo para cada canal RGB.

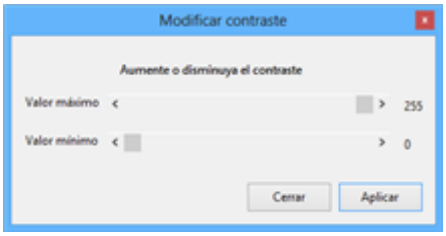

Ilustración 61. Menú modificar contraste.

Ejemplificando, si se selecciona como valor mínimo el 0 y valor máximo el 255, ajustará los niveles de la imagen para que se estiren hasta llegar a tener valores mínimos de 0 y máximos de 255.

Si se analiza con detenimiento esta función, en imágenes que cuyos valores mínimos y máximos son iguales o superiores que los máximos/mínimos de la imagen, no habrá modificación alguna.

En la imagen de la izquierda se muestra el histograma (del canal verde) de la imagen original, y la imagen de Lena a la que se ha dado un valor mínimo de 10 y máximo de 240.

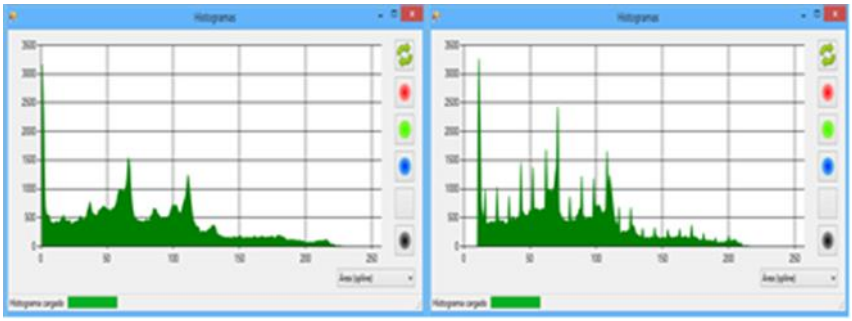

Ilustración 62. Diferencia histogramas.

Como se observa claramente en los valores más cercanos a 0 (puesto que abundan más que los cercanos a 255), se ve que se han desplazado hacia la derecha, es decir, han pasado a tener valor 10.

Ilustración 63. Imagen de Lena con contraste disminuido.

Si se compara con la imagen original, se detecta que es una imagen más apagada, menos contrastada.

8.5.4.4.5 Corrección de gamma

La corrección de gamma es la responsable de la sensación de contraste de una imagen, y se dice sensación porque no es algo inherente a la imagen, sino a los dispositivos que la registran, a los dispositivos que la reproducen y a la luminosidad del entorno donde se está observando.

Ilustración 64. Menú correción de gamma.

Si se accede al menú, se observa que requiere tres valores, uno para cada canal RGB. Y en función de esos valores de entrada se modificarán los diferentes canales para generar un valor de salida.

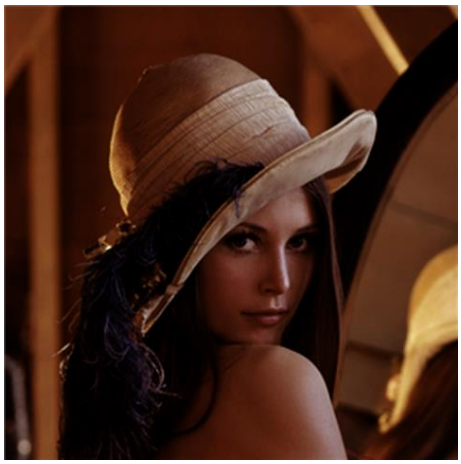

Ilustración 65. Imagen de Lena con corrección de gamma (0,53).

En la imagen anterior se puede observar que está más contrastada pero también más apagad. Observando el histograma se apreciará que los valores se han desplazado hacia el 0 y se han repartido más por todo el histograma.

8.5.4.4.6 Exposición

Esta función aumenta o disminuye la luz en la imagen. El algoritmo simplemente divide cada canal de cada píxel entre el valor seleccionado en la menú mostrado a continuación.

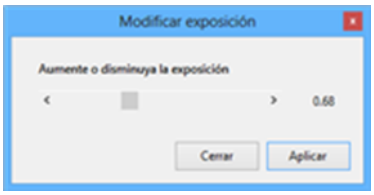

Ilustración 66. Menú modificar exposición.

Si se quiere aumentar la exposición (hacer más clara la imagen), se deben seleccionar valores menores de 1, en caso contrario mayores.

En la siguiente imagen se puede apreciar una modificación de exposición en 0,68 puntos.

Ilustración 67. Imagen de Lena sobreexpuesta.

Comparada con la imagen original, es una imagen más clara, cuyos valores han tendido a 255. Es una imagen sobreexpuesta, ya que está *quemada* por la luz.

8.5.4.4.7 Modificar canales

Esta función trabaja de una forma muy similar a aumentar brillo pero con la diferencia que la modifica de forma individual para cada canal.

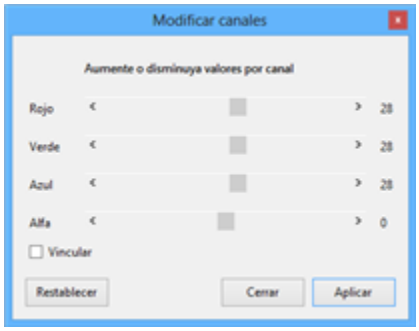

Ilustración 68. Menú modificar canales.

Como puede observarse en la imagen anterior, se puede seleccionar el valor a modificar para cada canal *ARGB* y este valor será lo que se aumentará o disminuirá a cada canal de cada píxel.

En la siguiente imagen se ha aumentado en 28 puntos el canal rojo y verde, y disminuido en 10 puntos el valor azul. El valor del canal alfa se ha conservado.

Ilustración 69. Imagen de Lena con canales modificados.

Como puede observarse la imagen se ha tornado hacia tonos amarillos, puesto que al aumentar los canales verde y rojo y disminuir el canal azul, se ha favorecido la obtención de colores amarillos (el color amarillo se crea con rojo más verde).

8.5.4.4.8 Reducir colores

Con esta función se limita el número de colores por canal. Esta limitación se efectúa desde los valores intermedios del rango del canal, es decir, los valores se empiezan a eliminar empezando por el intermedio (128).

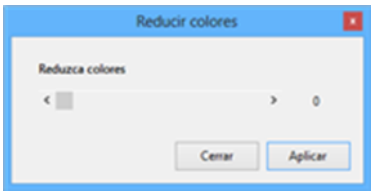

Ilustración 70. Menú reducir colores.

Seleccionando el valor 0, lo que ocurrirá es que para cada canal RGB únicamente quedarán dos colores el 1 y el 255. En la siguiente imagen se muestra a Lena a la que se ha aplicado reducción de colores a 0, es decir, sólo queda para cada canal el valor 1 y 255, por lo tanto el conjunto de la imagen sólo tendrá 2^3 colores.

Ilustración 71. Imagen de Lena con colores reducidos.

Como puede observarse la imagen tiene muy pocos colores. Viendo el histograma, únicamente hay valores 1 y 255 por cada canal.

8.5.4.4.9 Filtrar colores

Es una interesante opción que permite seleccionar un rango de colores y modificar su valor.

Dentro del menú se encuentran tres apartados, uno por canal como se ve en la ilustración.

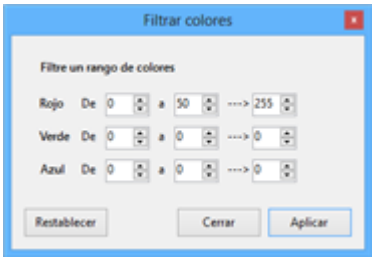

Ilustración 72. Menú filtrar colores.

En la imagen anterior los colores cuyo canal rojo está entre 0 y 50 pasarían a ser de 255. Esta transformación se ve reflejada en la siguiente imagen.

Ilustración 73. Imagen de Lena con colores filtrados.

Se aprecia en la imagen que las zonas más oscuras, donde cuyo canal rojo oscila entre 0 y 50, han pasado a tener un valor en dicho canal rojo de 255, un rojo intenso.

8.5.4.4.10 Corregir ojos rojos

Esta simple función permite hacer una corrección en fotografías que, por motivos de iluminación y debido al flash de la cámara, ha aparecido el efecto de ojos (pupilas) rojos. Para este ejemplo se utilizará la siguiente imagen.

Ilustración 74. Imagen original ojos rojos.

El menú para corregir este efecto cuenta con dos apartados, uno para el ojo de la izquierda y otro para el de la derecha. Además de dos botones (*Actuar sobre imagen inicial* y *Actuar sobre imagen actual*) para que los efectos se apliquen sobre la imagen inicial (la imagen capturada a la hora de abrir el menú Corregir ojos rojos) y otro para que el efecto se aplique sobre la imagen que muestra Apolo en su pantalla inicial.

Ilustración 75. Menú corregir ojos rojos.

Para cada ojo hay que proveer las coordenadas del centro del ojo, el radio y el valor umbral. Para obtener los datos de coordenadas, puede hacerse zoom sobre la imagen y obtener las coordenadas gracias a la barra inferior de estado. Con respecto al parámetro *valor umbral*, lo que hace es calcular, para cada píxel analizado, la siguiente operación, (*rojo / ((verde + azul) / 2)*), y si es superior el resultado al valor mínimo (por defecto 1,5), entonces quiere decir que se va a aplicar la modificación al píxel correspondiente.

Aplicando los parámetros mostrados en la anterior ilustración sobre la imagen con ojos rojos, el resultado sería el siguiente.

Ilustración 76. Imagen con ojos rojos corregidos.

Los resultados son bastante buenos, además de mantener un tono adecuado para los píxeles corregidos.

8.5.4.4.11 Matriz

Esta función trabaja de forma muy similar al efecto sepia, es más, puede decirse que es la base del efecto sepia. Lo que hace es asignar un peso a cada canal. En el siguiente ejemplo se muestra cómo se calcula el valor de cada canal en cada píxel.

Rojo = Rojo * Peso11 + Verde * Peso12 + Azul * Peso13

Verde = Rojo * Peso21 + Verde * Peso22 + Azul * Peso23

Azul = Rojo * Peso31 + Verde * Peso32 + Azul * Peso33

Donde cada peso corresponde a una valor de la matriz que se puede seleccionar en el menú.

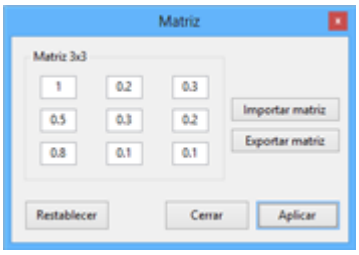

Ilustración 77. Menú matriz.

En la imagen anterior, se puede ver que el mayor peso se ha asignado al primer valor, por lo tanto, los resultados deberían ser tonos ligeramente rojizos.

Ilustración 78. Lena modificada por matriz.

Revisando el histograma de la anterior imagen se observa que los valores más altos están en el histograma rojo.

Este menú también permite guardar una matriz simplemente pulsando en *Exportar matriz*, y seleccionando un nombre. Una vez guardada la matriz, se puede recuperar pulsando en el botón *Importar matriz* y seleccionándola.

8.5.4.4.12 Detectar contornos

Esta función determina los contornos en una imagen. Para ello se sirve de 4 valores diferentes. El primer valor determina el umbral a partir del cual un píxel se asignará como contorno o no contornos (blanco o negro), y los tres restantes (rojo, verde, azul) son el peso que se asignará a cada canal.

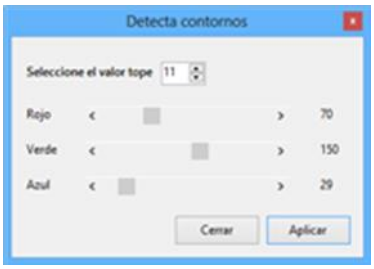

Ilustración 79. Menú detectar contornos.

Aplicando a la imagen de Lena los valores anteriores, se creará una imagen en que cada píxel únicamente puede ser negro (contorno) o blanco (relleno).

Ilustración 80. Imagen de Lena con contornos.

Como se aprecia en la imagen el resultado no es muy bueno ya que no determina bien los contornos. Se pueden ir modificando los valores para ajustarse a la imagen en concreto, pero automatizar esta función para que detecte siempre de forma correcta los contornos en cualquier tipo de imagen es muy complejo.

8.5.4.5 Operaciones

Quinto elemento de la barra superior de herramientas. A continuación se muestra su imagen.

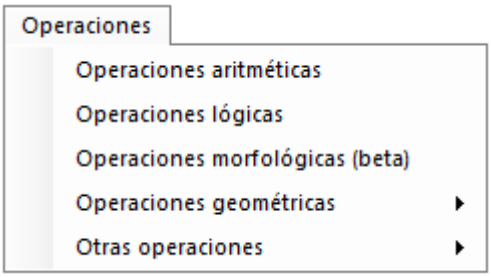

Ilustración 81. Menú operaciones.

Dentro de este menú se puede encontrar operaciones aritméticas, lógicas, morfológicas, geométricas, entre otras. La imagen que se va a utilizar para realizar las pruebas es Lena.

Ilustración 82. Imagen original de Lena.

8.5.4.5.1 Operaciones aritméticas

En este menú se puede realizar las operaciones básicas sobre una imagen, suma, resta, multiplicación y división. En la siguiente ilustración se muestra el menú correspondiente a *Operaciones aritméticas*.

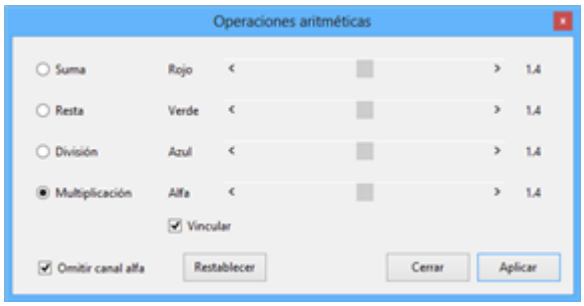

Ilustración 83. Menú operaciones aritméticas.

Como se puede observar, se dispone de 4 tipos de operaciones básicas. Además hay dos *checkbox*, uno de ellos para evitar que la operación afecte al canal alfa (*Omitir canal alfa*) y otro para vincular todas las barras de desplazamiento (*Vincular*).

Las operaciones se aplican píxel a píxel y canal a canal, así pues, en la imagen se ha seleccionado Multiplicación y 1,4 para cada canal, por lo tanto se multiplicará el valor de los tres canales *RGB* por 1,4.

Ilustración 84. Imagen de Lena multiplicada (1,4).

Tras la multiplicación puede observarse que la imagen se ha aclarado, ya que los valores han aumentado y están más cercanos a 255 (blanco).

8.5.4.5.2 Operaciones lógicas

Dentro de este menú se encuentran las funciones lógicas principales para tratar píxel a píxel. Se incluyen las opciones *AND*, *OR* y *XOR*. En la siguiente ilustración se muestra el menú correspondiente.

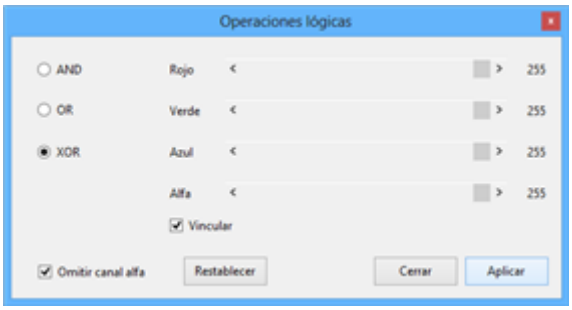

Ilustración 85. Menú operaciones lógicas.

Dentro del menú existen dos *checkbox*, uno de ellos para evitar que la operación afecte al canal alfa (*Omitir canal alfa*) y otro para vincular todas las barras de desplazamiento (*Vincular*).

Las operaciones se aplican píxel a píxel y canal a canal, así pues, en la imagen se ha seleccionado *XOR* y 255 para cada canal, por lo tanto se aplicará la operación *XOR* entre el valor del canal RGB evaluado y el 255.

Ilustración 86. Imagen de Lena operación XOR.

De la imagen anterior se saca la conclusión que realizar la operación XOR con 255 es lo mismo que realizar la inversión de colores sobre la imagen.

8.5.4.5.3 Operaciones estadísticas

Este menú incluye una serie de operaciones básicas estadísticas para calcular diferentes tipos de medias, mediana, rango a través de una matriz de convolución (*kernel*).

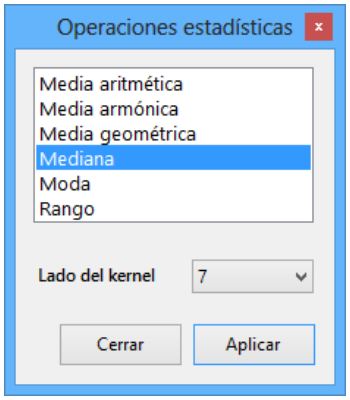

Ilustración 87. Menú operaciones estadísticas.

Como puede observarse en la anterior ilustración, en la parte superior aparece un listado con las diferentes operaciones a seleccionar y justo debajo una tripleta en la que poder seleccionar el ancho del *kernel*.

Si se aplica la mediana con lado de kernel 7, lo que se haciendo es evaluar grupos de 49 píxeles (7x7) y calcular la mediana para cada canal RGB de esos grupos, y el resultado asignarlo a los 49 píxeles.

Ilustración 88. Imagen de Lena mediana (lado 7).

Una vez visto el resultado aplicado sobre la imagen Lena, queda mucho más claro cómo funcionan los *kernels* aplicando filtros estadísticos.

8.5.4.5.4 Operaciones morfológicas

Dentro de este menú se encuentran las operaciones morfológicas básicas para aplicar en una imagen.

El formulario se compone de una serie de botones en la parte de la izquierda con las diferentes transformaciones morfológicas disponibles. Las transformaciones son: dilatación, erosión, apertura, cerradura, perímetro, top hat y bottom hat.

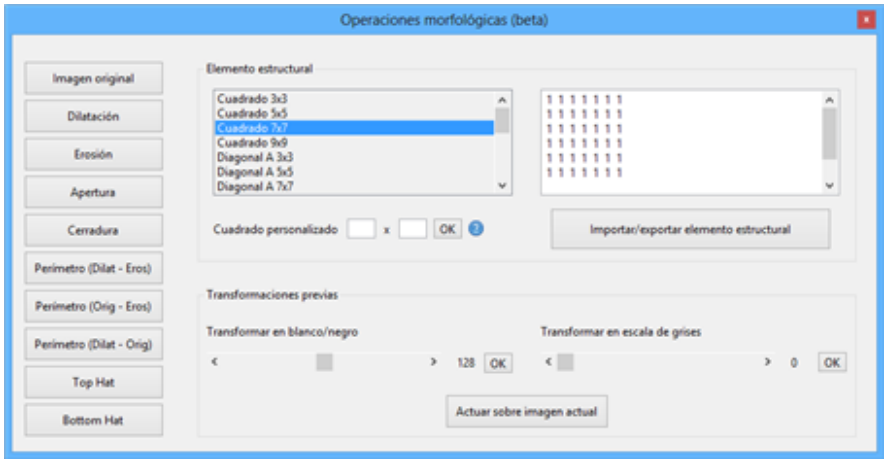

Ilustración 89. Menú operaciones morfológicas.

En la parte superior izquierda se encuentra el tipo de elemento estructural que se quiere seleccionar para aplicar la transformación, pudiéndose escoger de 3 formas diferentes:

- Cuadro de elementos estructurales: aparece un listado con múltiples elementos entre los que poder seleccionar. Dentro de él, hay desde elementos muy pequeños (3x3) a otros mayor (9x9) y además varios tipos de estructura (cuadrado, disco, diamante, etc.).
- Cuadrado personalizado: permite crear un elemento estructural cuadrado e impar, es decir, por ejemplo una matriz de 9x9 donde todos sus componentes serán 1.
- Importar/Exportar elemento estructural: esta opción muestra un nuevo cuadro de diálogo donde poder crear un elemento estructural determinado por el usuario. La única condición es que sea impar y cuadrado, pudiendo seleccionar entre 0 y/o 1. Una vez creado y guardado el elemento estructural se almacena como archivo XML para poder volver a abrirlo en un futuro.

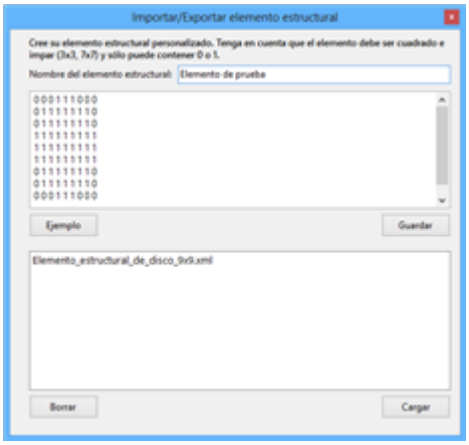

Ilustración 90. Menú importar/exportar elemento estructural.

Finalmente, dentro de este menú se encuentran también dos opciones para convertir la imagen en blanco y negro o en escala de grises, y un botón en la parte inferior para aplicar la transformación morfológica sobre la imagen mostrada en Apolo en ese momento.

Las transformaciones morfológicas se basan en buscar valores máximos/mínimos dentro del elemento estructural, para así aumentar o disminuir las zonas y poder, entre otros, detectar perímetros.

A continuación se va a aplicar a una imagen de una matrícula, una operación morfológica para determinar el perímetro. La imagen original pasada a blanco y negro es la siguiente.

Ilustración 91. Imagen matrícula en blanco y negro.

A esta imagen se le aplica (con un elemento estructural cuadrado de 3x3) la operación perímetro.

Ilustración 92. Imagen matrícula perímetro.

Como puede observarse en la ilustración anterior, ahora los números y letras de la matrícula están definidos de forma excelente por su contorno.

Una aplicación de este tipo de transformaciones es la detección de bordes y a partir de ella la detección de patrones.

8.5.4.5.5 Operaciones geométricas (Reflexión/Traslación/Volteado)

Dentro de este menú se encuentran las principales operaciones geométricas para aplicar sobre una imagen (matriz).

- Reflexión: se puede aplicar una reflexión horizontal o vertical que volteará la matriz de tal forma que el resultado sería el mismo que si se pusiese un espejo horizontal o verticalmente en una esquina de la imagen.

Ilustración 93. Imagen de Lena reflexión horizontal.

- Traslación: desplaza la imagen horizontal o verticalmente el número de píxeles seleccionado en el menú. Los píxeles desplazados se rellenarán con color negro (0, 0, 0) pero con su canal alfa a 0 (transparente).

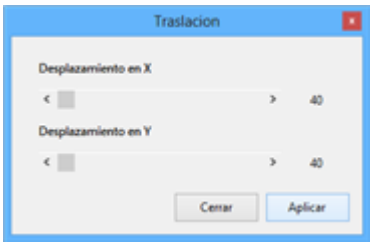

Ilustración 94. Menú traslación.

- Volteo: despliega un menú con las diferentes opciones de volteo predefinidas en .NET. Con ellas se puede realizar giros y reflexiones de una imagen de forma muy rápida.

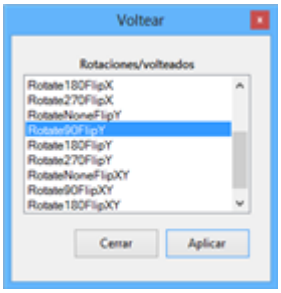

Ilustración 95. Menú volteos.

Los tipos de rotación son los siguientes:

- RotateNoneFlipNone: indica que no hay ni giro en el sentido de las agujas del reloj ni volteo.
- Rotate90FlipNone: indica un giro de 90 grados en el sentido de las agujas del reloj, sin volteo.
- Rotate180FlipNone: indica un giro de 180 grados en el sentido de las agujas del reloj, sin volteo.
- Rotate270FlipNone: indica un giro de 270 grados en el sentido de las agujas del reloj, sin volteo.
- RotateNoneFlipX: indica que no hay un giro en el sentido de las agujas del reloj seguido de un volteo horizontal.
- Rotate90FlipX: indica un giro de 90 grados en el sentido de las agujas del reloj, seguido de un volteo horizontal.
- Rotate180FlipX: indica un giro de 180 grados en el sentido de las agujas del reloj, seguido de un volteo horizontal.
- Rotate270FlipX: indica un giro de 270 grados en el sentido de las agujas del reloj, seguido de un volteo horizontal.
- RotateNoneFlipY: indica que no hay un giro en el sentido de las agujas del reloj seguido de un volteo vertical.
- Rotate90FlipY: indica un giro de 90 grados en el sentido de las agujas del reloj, seguido de un volteo vertical.
- Rotate180FlipY: indica un giro de 180 grados en el sentido de las agujas del reloj, seguido de un volteo vertical.
- Rotate270FlipY: indica un giro de 270 grados en el sentido de las agujas del reloj, seguido de un volteo vertical.
- RotateNoneFlipXY: indica que no hay giro en el sentido de las agujas del reloj seguido de un volteo horizontal y vertical.
- Rotate90FlipXY: indica un giro de 90 grados en el sentido de las agujas del reloj, seguido de un volteo horizontal y vertical.
- Rotate180FlipXY: indica un giro de 180 grados en el sentido de las agujas del reloj, seguido de un volteo horizontal y vertical.
- Rotate270FlipXY: indica un giro de 270 grados en el sentido de las agujas del reloj, seguido de un volteo horizontal y vertical.

Ilustración 96. Imagen de Lena volteo.

En la anterior imagen se muestra un volteo Rotate90FlipY.

8.5.4.5.6 Transformación afín (Manual/Personalizada)

Una transformación afín consiste en girar, escalar y trasladar una matriz. En Apolo hay dos opciones disponibles, una manual y otra automática (*personalizada*).

En la opción manual hay que rellanar los datos con los valores elegidos. Los datos son, giro (en grados), escalado (para X e Y), traslación (para X e Y) y por último el tamaño de la imagen. Además hay posibilidad de seleccionar el modo *Matrix* o *Drawimagen*, teniendo ambos resultados equivalentes.

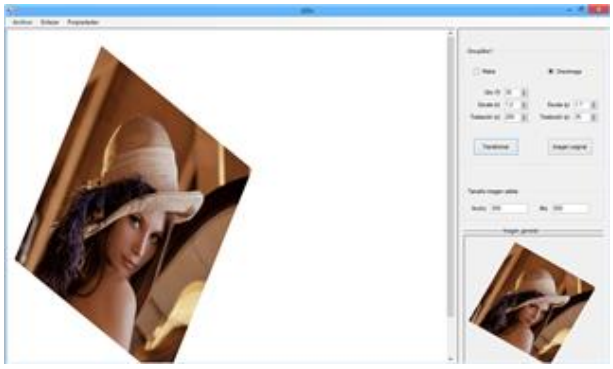

Ilustración 97. Menú transformación afín manual.

Como se puede observar en la anterior imagen, se ha aplicado a la imagen de Lena un giro de 30º, escalado 1.2 en x y 1.7 en y, y una traslación de 200 en x y 30 en y.

En la opción personalizada se deben seleccionar 4 coordenadas de origen y cuatro de salida, y de esta manera se ajusta automáticamente la imagen para que las coordenadas de entrada pasen a ser las de salida.

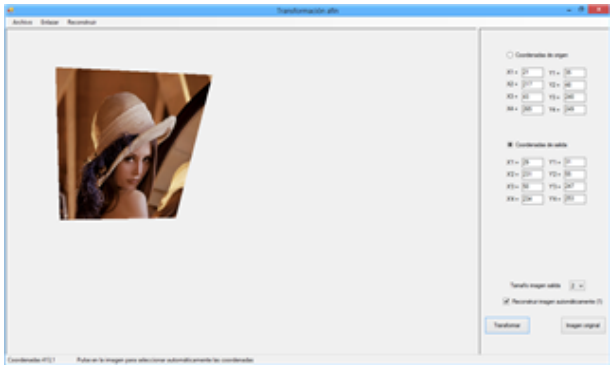

Ilustración 98. Menú transformación afín personalizada.

Tras una transformación de este tipo, la imagen queda resquebrajada, tal y como se observa en la siguiente ilustración.

Ilustración 99. Imagen de Lena transformación afín personalizada.

La imagen se puede reconstruir activando el checkbox *Reconstruir imagen automáticamente* o desde el menú *Reconstruir>Reconstruir imagen*. Esta transformación interpola el píxel a partir de los valores más próximos, pudiendo seleccionar el número de píxeles para realizar la interpolación (*Reconstruir>Propiedades*).

Ilustración 100. Imagen de Lena reconstruida.

Como puede observarse, tras la reconstrucción apenas se nota la diferencia.

8.5.4.5.7 Density slicing (Automático/Manual)

Esta función es una de las formas más simples de segmentación. El método consiste en pasar la imagen a escala de grises e ir asignando por rangos, diferentes colores. Por

ejemplo, para los valores de 0 a 50, el color rojo, de 51 a 100 el verde, etc. Hay dos métodos disponibles, uno automático y otro manual.

El método automático consiste en seleccionar el número de divisiones y los colores que se quieren asignar a cada división. Estas divisiones las calculará Apolo para que sean todas con el mismo valor o muy aproximado.

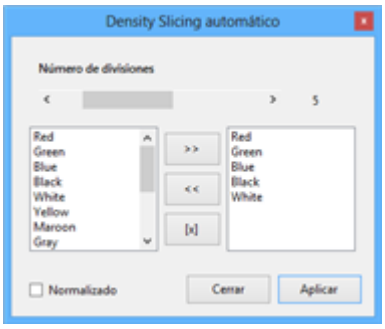

Ilustración 101. Menú density slicing automático.

Como puede observarse hay un *checkbox* para marcar denominado *Normalizado*. Si se activa esta opción, Apolo busca el menor y el mayor valor de gris y hace las divisiones entre esos dos valores en vez de entre 0 y 255. Se puede deducir que si la imagen ocupa todo el rango de grises, la versión normal y normalizada serán iguales.

Ilustración 102. Imagen de Lena density slicing automático.

En la imagen anterior se muestra a Lena después de aplicar la función density slicing con 5 colores (rojo, azul, verde, negro y blanco).

La opción density slicing manual, es similar a la anterior a diferencia de que es el propio usuario quien decide el rango de cada división. Por ejemplo, se pueden seleccionar 3 divisiones y a la primera asignar el rango de 0 a 10, a la segunda de 11 a 250 y a la tercera de 251 a 255.

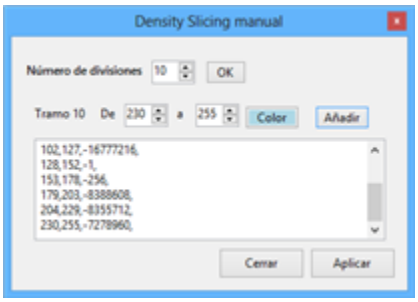

Ilustración 103. Menú density slicing manual.

Además, este menú permite seleccionar una mayor gama de colores mostrando un cuadro de diálogo con la opción de crear colores personalizados.

Ilustración 104. Imagen de Lena density slicing manual.

En la imagen anterior se muestra a Lena después de una transformación con 10 divisiones.

8.5.4.6 Máscaras

Es la sexta opción en la barra superior de opciones. A continuación se muestra una imagen del menú.

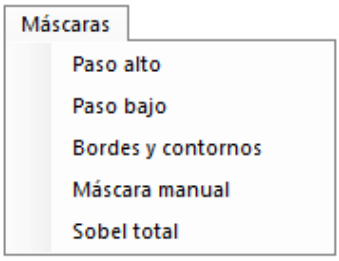

Ilustración 105. Menú máscaras.

Todos los filtros expuestos en esta sección se aplicarán sobre la imagen Lena. En la siguiente ilustración se muestra.

Ilustración 106. Imagen original de Lena.

Las máscaras se sirven de un método matemático denominado máscara de convolución. Convolución es el tratamiento de una matriz por otra que se llama *kernel*. Este *kernel* no es más que una matriz (de 3x3) cuyo píxel central será el evaluado y los píxeles adyacentes son sus vecinos. De esta manera el valor del píxel evaluado se transformará en función del valor de sus vecinos. Ejemplificándolo, imagínese una matriz de 3x3 y cuyos valores para los tres canales RGB son:

20,30,200	21,32,190	30,40,202
40,21,180	50,33,190	20,41,203
21,30,200	22,31,192	40,35,201

Tabla 1. Valores de los píxeles.

A continuación, se aplicará un *kernel* con los siguientes valores:

1	1	1
1	2	1
1	1	1

Tabla 2. Kernel 3x3.

Este kernel actuará sobre todos los píxeles de la imagen, en este caso, se va a aplicar sobre el píxel central (50, 33, 190), por lo tanto, las operaciones a realizar para calcular el nuevo valor del píxel serían:

- Para el canal rojo: (20*1+21*1+30*1+40*1+**50*2**+20*1+21*1+22*1+40*1)/9
- Para el canal verde: (30*1+32*1+40*1+21*1+**33*2**+41*1+30*1+31*1+35*1)/9
- Para el canal azul: (200*1+190*1+202*1+40*180+**190*2**+203*1+200*1+192*1+201*1)/9

En negrita se ha resaltado el píxel central.

Como se comprueba, la operación que se realiza es la suma entre todos los valores de los vecinos (del *kernel*) en función de sus pesos y se divide entre el total de la suma de los pesos. En la siguiente imagen se puede observar gráficamente (imagen extraída de la documentación de GIMP):

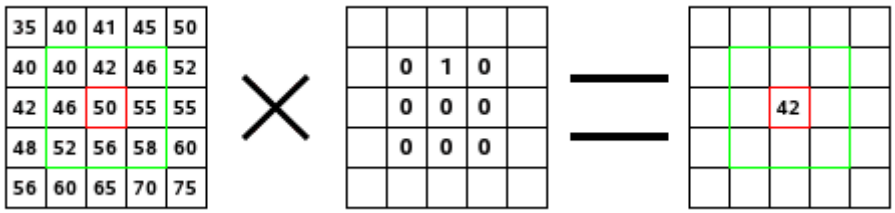

Ilustración 107. Ejemplo de kernel (extraído de GIMP).

Adicionalmente, el resultado puede dividirse con la suma de dos valores (*Desviación* + *factor*). Estos valores de forma predefinida suman 1 para no afectar al resultado final.

Apolo sólo provee de máscaras de 3x3, actualmente. Si usted requiere máscaras mayores, puede notificarlo y en poco tiempo lanzaremos una actualización que incluya máscaras mayores, o incluso, puede seguir la documentación técnica para desarrollarlo usted, ¡es muy sencillo!

8.5.4.6.1 Paso alto

Este tipo de máscaras se utiliza para resaltar detalles de la imagen y enfocarlas. Lo que hace es asignar el peso más alto al píxel central (evaluado) y a los vecinos valores negativos o 0.

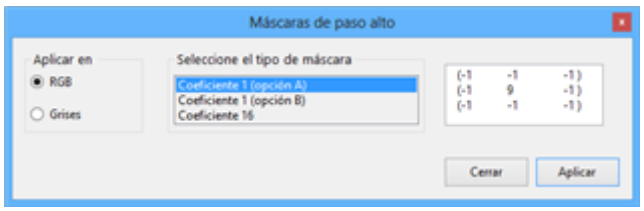

Ilustración 108. Menú máscaras de paso alto.

De la imagen anterior se extrae que, en la máscara seleccionada el valor del píxel central será de 9 y los píxeles vecinos de -1. Si se aplica esta máscara a la imagen original de Lena, se obtiene el siguiente resultado.

Ilustración 109. Imagen de Lena filtro paso alto.

Los detalles puntuales se han resaltado, pero por contra, se ha introducido bastante ruido en la imagen.

8.5.4.6.2 Paso bajo

Este tipo de filtros se utiliza generalmente para eliminar ruido en una imagen, suavizarla o desenfocarla. Lo que hace es asignar pesos similares a todos los píxeles del kernel y así suavizar los colores.

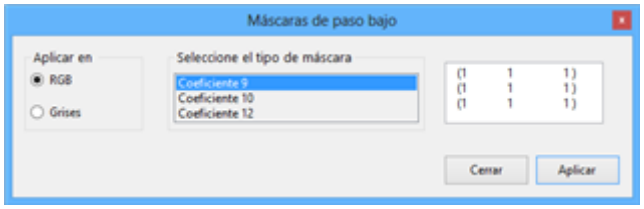

Ilustración 110. Menú máscaras de paso bajo.

En la imagen anterior se observa que la máscara está formada únicamente por 1. De esta manera se consiguen suavizar todos los colores de la imagen. Aplicando el filtro de la ilustración sobre la imagen original de Lena, se observan sus resultados.

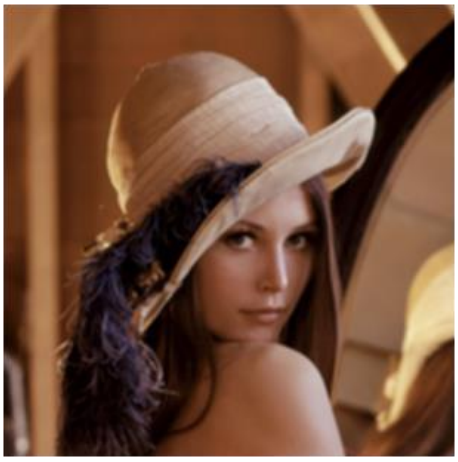

Ilustración 111. Imagen de Lena filtro de paso bajo.

En la imagen anterior se observa cómo se han disminuido los detalles en la imagen y se aprecia que la imagen está desenfocada. Esto es debido a que el rango de colores ha disminuido y las diferencias entre colores son menores que antes.

8.5.4.6.3 Bordes y contornos

Las máscaras agrupadas en este menú se utilizan generalmente para detectar bordes de una imagen o contornos. Lo que se busca al aplicar estos kernels es buscar los píxeles límite o frontera para así determinar el perímetro dentro de la imagen.

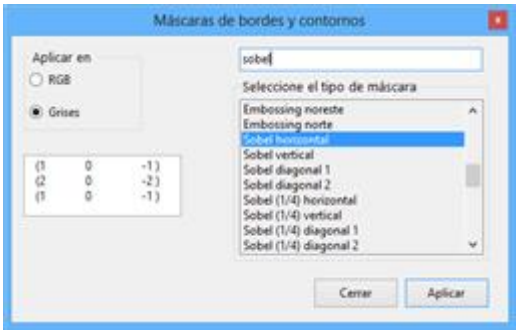

Ilustración 112. Menú de bordes y contornos.

En la ilustración anterior se observa el menú de bordes, el cual dispone de un buscador para localizar de forma más fácil la máscara seleccionada. En el siguiente caso, se va a aplicar la máscara de *Sobel*. Esta máscara está pensada para buscar puntos frontera en la imagen y así poder visualizar los bordes.

Ilustración 113. Imagen de Lena Sobel horizontal.

Se ha aplicado la máscara en escala de grises para obtener unos mejores resultados. Como puede observarse la detección de contornos con *Sobel* es bastante buena. Si se aplica a continuación un filtro de paso bajo se podría eliminar el exceso de ruido.

8.5.4.6.4 Máscara manual

Esta opción permite crear un *kernel* de 3x3 a elección libre del usuario. Además de los 9 valores de la máscara, se puede seleccionar el valor del factor y la desviación, en caso de querer aplicarla y si se quiere aplicar en el espacio *RGB* o en escala de grises.

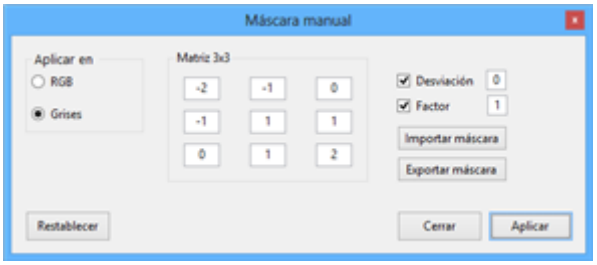

Ilustración 114. Menú máscaras manuales.

Opcionalmente, se puede exportar la máscara y se creará un archivo XML que se almacenará en la configuración de Apolo. Este archivo (máscara) puede volver a utilizarse pulsando la opción *Importar máscara* y buscándola por su nombre.

A continuación se puede ver la imagen de Lena a la que se ha aplicado una máscara personalizada con los valores mostrados en la ilustración anterior.

Ilustración 115. Imagen de Lena máscara manual (repujado).

La máscara se ha aplicado sobre la escala de grises y sus valores corresponden a una máscara conocida como *repujado*.

8.5.4.6.5 Máscara manual (más tamaños)

Esta opción permite crear un kernel personalizado a elección libre del usuario. El kernel podrá tener tamaño de 5x5, 7x7, 9x9 o 1x1. Además de los valores de la máscara, se puede seleccionar el valor del factor y la desviación en caso de querer aplicarlo. Para aplicar esta máscara en escala de grises, debe transformar previamente la imagen a escala de grises.

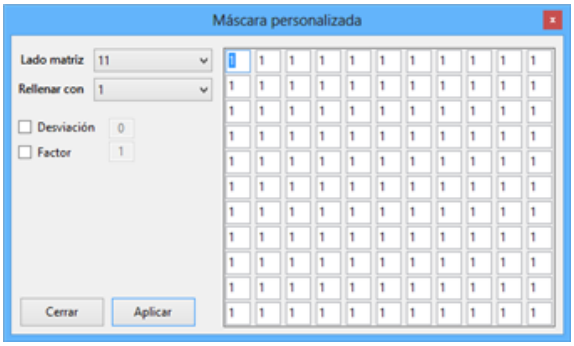

Ilustración 116. Menú máscara manual (más tamaños).

Como puede observarse en la anterior ilustración, también hay un *combo* para rellenar todas las casillas de la matriz con el número seleccionado. Si se aplica a la imagen de Lena la matriz que se ve en la ilustración, el resultado sería el siguiente.

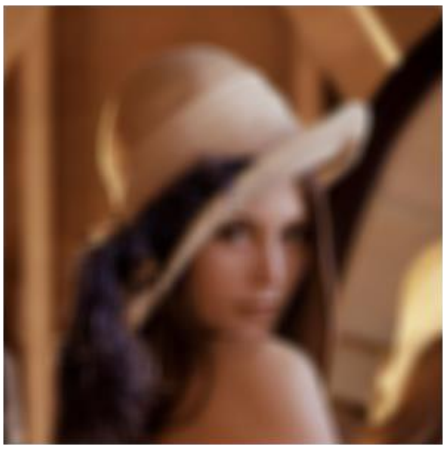

Ilustración 117. Menú máscara manual (11x11).

Tras aplicar la máscara, la imagen ha quedado totalmente borrosa, debido a que se ha aplicado una máscara de paso bajo.

8.5.4.6.6 Sobel total

La función *Sobel total* es una aplicación específica de las anteriores. Simplemente aplica sobre la imagen original los cuatro tipos de máscara *Sobel* (horizontal, vertical, diagonal 1, diagonal 2) y una vez almacenadas esas imágenes, las une para obtener el resultado final.

Ilustración 118. Imagen de Lena Sobel total.

Comparándola con la imagen en la que sólo se había aplicado Sobel horizontal, el nivel de ruido es inferior y el detalle de los bordes se ha resaltado. Si sobre esta imagen se intentan eliminar los valores menos blancos, el resultado sería una muy buena detección de bordes.

8.5.4.7 Efectos

Séptima opción del menú superior. A continuación se muestra una imagen.

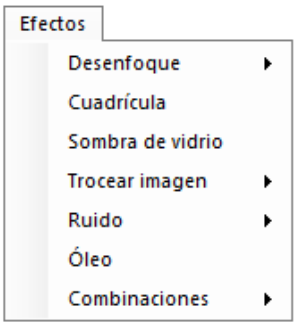

Ilustración 119. Menú efectos.

En esta sección se encuentran las funciones para aplicar efectos artísticos a imágenes. Muchos de estos efectos se crean a partir de otras funciones de Apolo. La imagen sobre la que se aplicarán será la mítica imagen de Lena.

Ilustración 120. Imagen original de Lena.

8.5.4.7.1 Desenfoque distorsión

Con esta función se crea un efecto de distorsión o desenfoque de la imagen. Dentro del menú se puede seleccionar el nivel de desenfoque, desde no aplicar nada (0) hasta el máximo permitido (10).

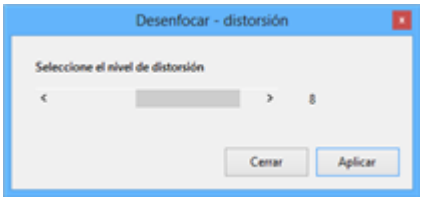

Ilustración 121. Menú desenfoque distorsión.

La forma de actuar de la función es muy sencilla, para calcular el valor del píxel actual (posición x=20, y=20) utiliza en nivel de distorsión proporcionado por el usuario (3), y asigna el nuevo valor seleccionando de forma aleatoria entre los vecinos de la posición actual, más en concreto entre la matriz dada por el valor proporcionado por el usuario.

Aplicando una distorsión de 8 puntos a la imagen de Lena se observa el resultado.

Ilustración 122. Imagen de Lena distorsionada.

Al visualizar la imagen transformada se ve claramente cómo funciona la distorsión. Las esquinas son difusas y se crea un efecto de *emborronado* de la imagen.

8.5.4.7.2 Desenfoque movimiento

Este tipo de desenfoque crea una especia de duplicidad de imágenes estando una de ellas movida. Dentro del menú se puede seleccionar el valor de desplazamiento vertical y horizontal que se quiere aplicar a la imagen.

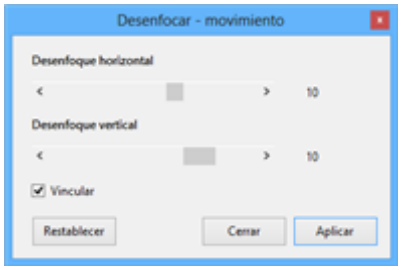

Ilustración 123. Menú desenfoque movimiento.

La función simplemente crea una copia de la imagen original y la desplaza el valor seleccionado por el usuario. En las zonas libres (zonas sin superposición) el valor de los píxeles es el mismo que el de la imagen original, y en las zonas no libres se suma el píxel original y el mismo píxel al que se ha sumado/restado el valor del desenfoque (las coordenadas) y se divide entre dos.

Ilustración 124. Imagen de Lena desenfocada.

La imagen anterior muestra a Lena tras un desenfoque vertical y horizontal de +10. El resultado es la imagen original a la que se le ha superpuesto la misma imagen movida 10 píxeles en X e Y.

8.5.4.7.3 Desenfoque (*Blur*)

Este tipo de desenfoque utiliza una máscara de convolución que suaviza los colores de la imagen. El proceso está explicado en la sección de máscaras.

Si se aplica el efecto *Blur* sobre la imagen Lena, se obtienen los siguientes resultados.

Ilustración 125. Imagen de Lena desenfoque Blur.

A la imagen se le ha aplicado una máscara de paso bajo cuyos valores de la matriz son todo unos.

8.5.4.7.4 Pixelado

Al aplicar este efecto sobre una imagen se despliega un menú que permite seleccionar el valor del pixelado que oscila desde 1 a 20.

Ilustración 126. Menú pixelar.

Al aplicar el pixelado sobre una imagen lo que se está realizando es una media de los píxeles seleccionados (este número depende de valor seleccionado por el usuario) y aplicándolos a todos ellos. Por ejemplo, si selecciona como valor 3, se divide la imagen en cuadrados de 3x3 (9 píxeles en total) y se calcula la media de esos cuadrados. Una vez calculada la media, ésta se asigna a todos los píxeles del cuadrado correspondiente.

Aplicando un pixelado de 12 a la imagen de Lena, se obtiene el siguiente resultado.

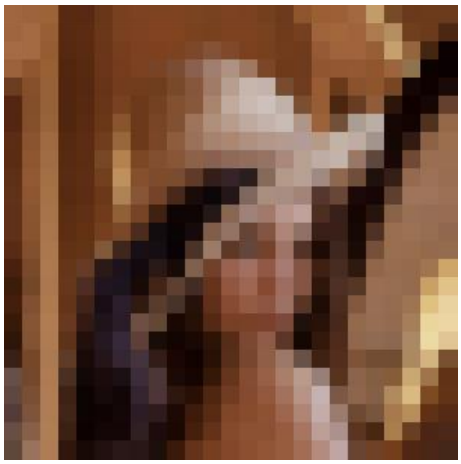

Ilustración 127. Imagen de Lena pixelada.

Cada cuadrado de la imagen tiene el mismo valor en sus canales *RGB*. Este valor ha sido calculado previamente con los valores originales de la imagen.

8.5.4.7.5 Cuadrícula

Aplicando esta función a una imagen, se crea una malla regular del ancho/alto y color seleccionado.

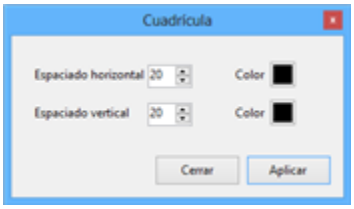

Ilustración 128. Menú cuadrícula.

Como se observa en la imagen anterior, el espaciado horizontal se puede seleccionar en dos controles numéricos. Para seleccionar el color se debe pulsar en los botones los cuales desplegarán el menú clásico de Windows para seleccionar un color predefinido o personalizado.

Ilustración 129. Imagend de Lena con cuadrícula.

A Lena se le ha aplicado una malla con 20 píxeles de ancho y alto entre línea y línea, y color negro tanto para las líneas horizontales como a las verticales.

8.5.4.7.6 Sombre de vidrio

Este efecto crea una imagen espejo en la parte inferior de la imagen original, pudiendo estar ésta atenuada. Desde el menú se puede seleccionar el tamaño de la sombra y si quiere atenuarse.

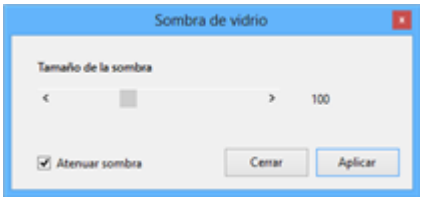

Ilustración 130. Menú sombra de vidrio.

El proceso para crear la sombra contiene varios pasos. El primero es recortar desde abajo el tamaño seleccionado por el usuario a partir de la imagen original. Esa parte se rota 180º y se pega en la parte inferior de la imagen original, creando así una especie de espejo. Adicionalmente, si se selecciona la casilla *Atenuar sombra*, a la parte que se ha pegado se le disminuye el canal alfa de forma progresiva.

Para apreciar bien el efecto se ha utilizado otra imagen distinta de Lena. La imagen original se muestra a continuación.

Ilustración 131. Imagen original botella.

Tras aplicar a la anterior imagen una sombra de 100 píxeles y con atenuación de sombra, el resultado es el siguiente.

Ilustración 132. Imagen botella sombra de vidrio.

En la imagen anterior se ve claramente el proceso, primeramente se recorta la imagen desde abajo y se coloca en la parte inferior una vez rotada 180º. Para finalizar se recorre la sombra disminuyendo progresivamente el valor alfa.

8.5.4.7.7 Trocear imagen (Tres partes/Seis partes)

Esta función divide la imagen en el número de bloques seleccionado y los desordena. Tomando como ejemplo las seis partes, la función a partir de la imagen original obtiene seis partes dividiéndola tres veces en vertical y una en horizontal. Una vez obtenidas las seis partes, intercambia posiciones entre ellas para obtener el efecto.

Ilustración 133. Imagen de Lena seis partes.

En la ilustración anterior se puede ver la imagen de Lena después de aplicar la función trocear imagen, en concreto la opción de seis partes.

8.5.4.7.8 Ruido aleatorio

Esta función calcula el nuevo valor de colores de forma aleatoria. El grado de aleatoriedad se calcula en función del valor seleccionado por el usuario en el menú.

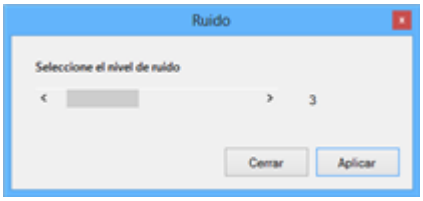

Ilustración 134. Menú ruido aleatorio.

Aplicando un grado de aleatoriedad de 3 a la imagen Lena, se obtiene el siguiente resultado.

Ilustración 135. Imagen de Lena con ruido aleatorio.

Como se puede observar el ruido aplicado no tiene un patrón fijo, sino que se calcula de forma aleatorio (*pseudoaleatoria*) para cada canal *RGB* de cada píxel.

8.5.4.7.9 Ruido desplazado

Esta función es similar al ruido aleatorio a diferencia de que el grado de aleatoriedad se calcula en función del valor evaluado.

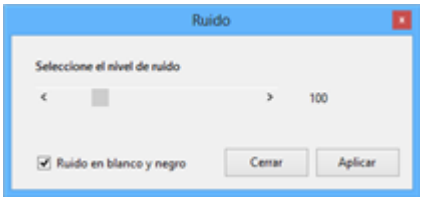

Ilustración 136. Menú ruido desplazado.

Más en concreto, al aplicar este efecto sobre una imagen (ruido 100), para cada píxel se obtiene su valor y se le suma o resta un valor aleatorio entre el valor actual más/menos el valor seleccionado (100). Opcionalmente se puede marcar Ruido en blanco y negro y el valor generado aleatoriamente será el mismo en los tres canales *RGB*.

En la ilustración anterior se ha aplicado a la imagen Lena un ruido de 100 puntos y seleccionando la opción Ruido en blanco y negro.

Ilustración 137. Imagen de Lena ruido desplazado.

Tras aplicar este efecto se observa que a la imagen se le ha añadido una especie de neblina en tonos de gris.

8.5.4.7.10 Óleo

Crea un efecto artístico que simula una pintura al óleo. Este efecto permite ser modificado por dos parámetros, el nivel de contornos y la reducción de colores.

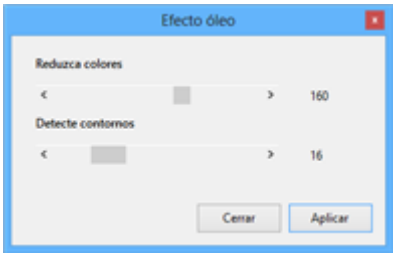

Ilustración 138. Menú óleo.

El efecto se basa en, a partir de la imagen seleccionada crea dos, una en la que se ha aplicado la función *Detectar contornos* (explicada en la sección de *Operaciones básicas personalizadas*, *Detectar contornos*) y otra en la que se utiliza la función Reducir colores (explicada en la sección de *Operaciones básicas personalizadas*, *Reducir colores*). La primera imagen (detectar contornos) es una imagen que contiene exclusivamente valores negros y blancos y se recorre en busca de píxeles no negros (blancos). Una vez ha encontrado un píxel blanco, busca en la imagen con colores reducidos las coordenadas de ese píxel y a la imagen de salida le asigna el valor del píxel de esta imagen (colores reducidos). En caso de ser negro lo pinta de color negro.

Ilustración 139. Imagen de Lena efecto óleo.

Como puede observarse tras aplicar a Lena un óleo (contornos = 16, colores = 160), se ve claramente que la imagen resultante es la imagen con colores reducidos a la que se le ha superpuesto la parte de color negro (contornos) de la imagen Lena original después de aplicar la detección de contornos.

8.5.4.7.11 Combinaciones

En este apartado, los efectos se crean a partir de operaciones aritméticas o lógicas entre dos imágenes (véase la sección *Operaciones con dos imágenes*).

8.5.4.7.12 Efecto Marte

El efecto se crea restando la imagen original menos la imagen original a la que se le ha aplicado una máscara. En concreto una máscara en escala de grises denominada *Repujado*.

Los resultados aplicándolos sobre la imagen de Lena son los siguientes.

Ilustración 140. Imagen de Lena efecto Marte.

El resultado es Lena en tonos rojizos.

8.5.4.7.13 Efecto antiguo sobreexpuesto

El efecto se crea sumando la imagen original más la imagen original a la que se le ha aplicado una máscara. En concreto una máscara en escala de grises denominada *Repujado*.

Los resultados aplicándolos sobre la imagen de Lena son los siguientes.

Ilustración 141. Imagen de Lena efecto antiguo sobreexpuesto.

La imagen ha pasado a tener valores mucho más altos en sus tres canales *RGB* y da la sensación de estar sobreexpuesta.

8.5.4.7.14 Efecto marino

El efecto se crea restando la imagen original a la que se le ha aplicado una máscara, menos la imagen original. En concreto una máscara en escala de grises denominada *Repujado*. Los resultados aplicándolos sobre la imagen de Lena son los siguientes.

Ilustración 142. Imagen de Lena efecto marino.

La imagen de Lena ha pasado a tener tonos azulados.

8.5.4.7.15 Aumentar rasgos

El efecto se crea mediante dos operaciones aritméticas. Primeramente se aplica a la imagen original dos máscaras en escala de grises. En concreto una máscara denominada *Prewitt* tanto en su opción horizontal como vertical. Estas dos imágenes se unen. El último paso es restar la imagen original menos la resultante del proceso anterior. Los resultados aplicándolos sobre la imagen de Lena son los siguientes.

Ilustración 143. Imagen de Lena efecto aumentar rasgos.

Como puede observarse las zonas oscuras han aumentado. Puede notarse con mayor intensidad en la zona de las cejas, ojos y pelo. Se observa que el perfil ha aumentado.

8.5.4.7.16 Disminuir rasgos

El efecto se crea mediante dos operaciones aritméticas. Primeramente se aplica a la imagen original dos máscaras en escala de grises. En concreto una máscara denominada *Prewitt* tanto en su opción horizontal como vertical. Estas dos imágenes se unen. El último paso es sumar la imagen original más la resultante del proceso anterior. Los resultados aplicándolos sobre la imagen de Lena son los siguientes.

Ilustración 144. Imagen de Lena efecto disminuir rasgos.

Como puede observarse las zonas oscuras han disminuido. Puede notarse con mayor intensidad en la zona de las cejas.

8.5.4.7.17 Contorno sombreado (Contenido/Desmedido)

Hay dos efectos disponibles, uno denominado contenido y otro desmedido.

El efecto sombreado contenido se crea restando la imagen original menos la imagen original a la que se le ha aplicado una máscara. En concreto una máscara en los canales *RGB* denominada *Repujado*.

Los resultados aplicándolos sobre la imagen de Lena son los siguientes.

Ilustración 145. Imagen de Lena efecto contornos contenidos.

El efecto sombreado desmedido se crea restando la imagen original a la que se le ha aplicado una máscara menos la imagen original. En concreto una máscara en los canales *RGB* denominada *Repujado*.

Los resultados aplicándolos sobre la imagen de Lena son los siguientes.

Ilustración 146. Imagen de Lena efecto contornos desmedidos.

Como puede observarse, en la segunda ilustración los contornos son más abruptos.

8.5.4.7.18 Aumentar luz

El efecto se crea mediante dos operaciones aritméticas. Primeramente se transforma la imagen original a escala de grises, y se resta la imagen original menos la transformada a escala de grises. El último paso es sumar la imagen original más la resultante del proceso anterior. Los resultados aplicándolos sobre la imagen de Lena son los siguientes.

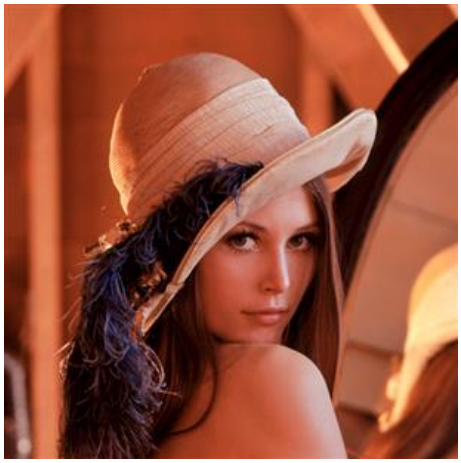

Ilustración 147. Imagen de Lena efecto aumentar luz.

La imagen de Lena al transformarse se torna a tonos más cálidos, con un efecto de mayor luz.

8.5.4.7.19 Marcos

En la sección de efectos se incluyen una serie de marcos para envolver a imágenes. En concreto están disponibles 4 marcos individuales y uno denominado marco de cine, en

el que se pueden incluir 6 imágenes. A continuación se muestra el menú de este último marco citado.

Ilustración 148. Menú marco de cine.

Como puede observarse, se puede seleccionar el tamaño del marco y, para abrir las diferentes imágenes, basta con pulsar encima de alguna y se abre un cuadro de diálogo para abrir la imagen. El resultado del marco se muestra a continuación.

Ilustración 149. Imagen de Lena marco de cine.

Además de este marco, como ya se citaba, existen 4 marcos más, en la siguiente imagen se observa uno de ellos.

Ilustración 150. Imagen de Lena con marco.

8.5.4.8 Operaciones con dos imágenes

Situado en la octava posición de la barra superior de herramientas. A continuación se muestra una imagen del menú.

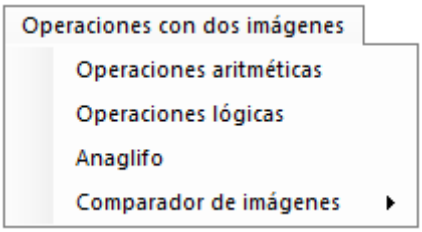

Ilustración 151. Menú operaciones con dos imágenes.

Este menú engloba operaciones aritméticas, lógicas, creación de anáglifos y comparación de imágenes. Las imágenes utilizadas durante esta sección del manual de ayuda, son las dos siguientes.

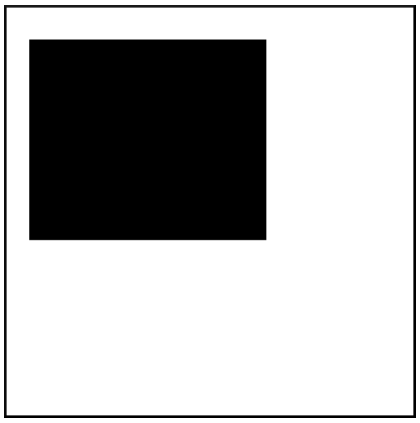

Ilustración 152. Imagen original 1.

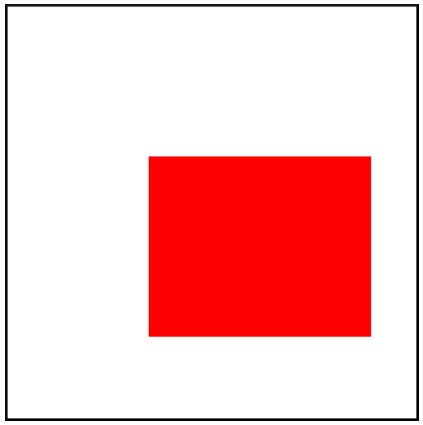

Ilustración 153. Imagen original 2.

8.5.4.8.1 Operaciones aritméticas (Suma/Resta/División/Multiplicación)

En este menú se encuentran las operaciones aritméticas básicas con dos imágenes. Como se puede observar en la siguiente imagen, se puede unir, sumar, restar, multiplicar y dividir dos imágenes. Además hay un *checkbox* para que las operaciones no afecten al canal alfa.

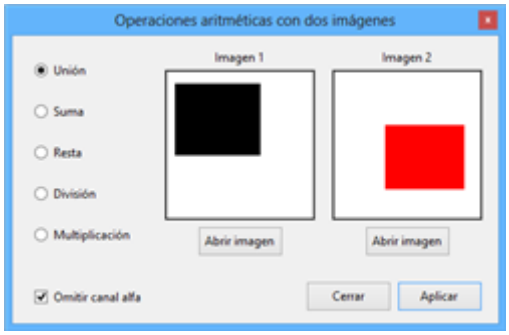

Ilustración 154. Menú operaciones aritméticas con dos imágenes.

Un paso previo a la hora de ejecutar la operación es cuadrar las imágenes, es decir, buscar entre las dos imágenes el ancho y alto mínimo, y la imagen de salida tendrá esos valores. Por ejemplo, si se tiene una imagen de 100x75 y otra de 75x100, la imagen resultante tras una operación aritmética será de 75x75.

La operación *Unión* busca los píxeles con las mismas coordenadas, suma sus 3 canales *RGB* y los divide entre dos, es decir, el valor de la imagen de salida será la media (para cada canal) de los valores de los píxeles de las imágenes de entrada. Si se aplica una unión a las dos imágenes de prueba el resultado sería el siguiente.

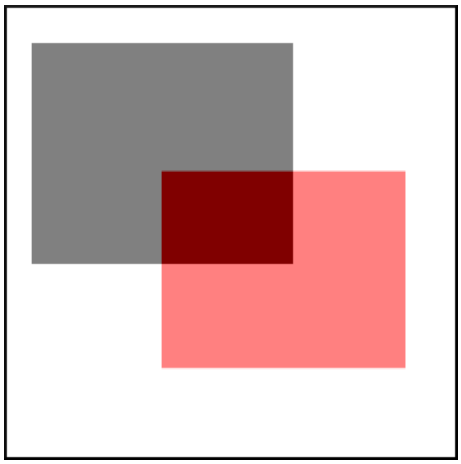

Ilustración 155. Unión de imágenes.

Como puede observarse, las zonas comunes (intersección de rojo y negro) tienen un color resultado de sumar negro (0, 0, 0) y rojo (255, 0, 0) y dividirlo entre dos (128, 0, 0).

Las zonas que son blancas en ambas imágenes seguirán siendo blancas, y las zonas negras que interseccionan con blanco pasarán a tener valor gris intermedio (128, 128, 128), y por último, las zonas rojas con intersección blanca tendrán un valor de 255 en el canal rojo y 128 en los canales restantes.

La operación *Suma* busca los píxeles con las mismas coordenadas y suma sus 3 canales *RGB* de los valores de los píxeles de las imágenes de entrada. En caso de que algún valor sobrepase los 255, automáticamente pasa a tener valor 255. Si se aplica la suma a las dos imágenes de prueba el resultado sería el siguiente.

Ilustración 156. Suma de imágenes.

Como puede observarse en la imagen, todo ha pasado a tener color blanco (255, 255, 255), excepto la zona común de negro y rojo. Esto es debido a lo siguiente; al juntar las imágenes, si en cualquiera de las dos es valor del píxel es blanco, automáticamente el píxel de salida pasará a ser blanco. Únicamente donde se junta el cuadrado negro y rojo, el píxel de salida pasa a ser rojo, ya que si se suma negro (0, 0, 0) más rojo (255, 0, 0), el resultado será rojo.

La operación *Resta* busca los píxeles con las mismas coordenadas y resta sus 3 canales RGB de los valores de los píxeles de las imágenes de entrada. En caso de que algún valor sea inferior a 0, automáticamente pasa a tener valor 0. Si se aplica la resta de la imagen 1 (imagen con cuadrado negro) menos la imagen 2 (imagen con cuadrado rojo) el resultado es el siguiente.

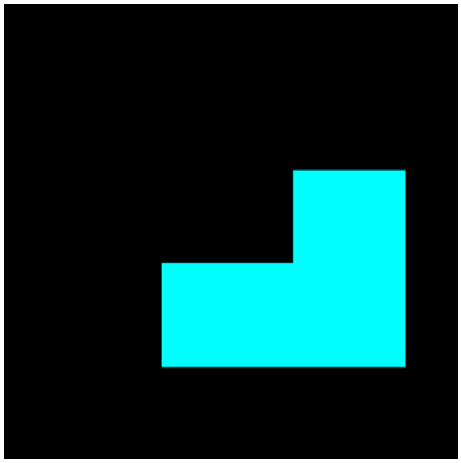

Ilustración 157. Resta de imágenes.

En la ilustración anterior se puede ver lo sucedido. La parte negra ha sido creada o bien, restando blanco (255, 255, 255) menos blanco (255, 255, 255), negro (0, 0, 0) menos blanco (255, 255, 255) o negro (0, 0, 0) menos rojo (255, 0, 0). El color azulado se ha creado en las zonas donde no hay intersección entre el cuadrado rojo y el negro, debido a que se ha restado el valor blanco (255, 255, 255) menos el rojo (255, 0, 0) siendo su resultado un color sin rojo y con los canales verde y azul al máximo (0, 255, 255).

La operación *División* busca los píxeles con las mismas coordenadas y divide sus 3 canales *RGB* de los valores de los píxeles de las imágenes de entrada. En caso de que algún valor del denominador sea 0, automáticamente pasa a tener valor 1. Si se aplica la división de la imagen 1 (imagen con cuadrado negro) menos la imagen 2 (imagen con cuadrado rojo) el resultado es el siguiente.

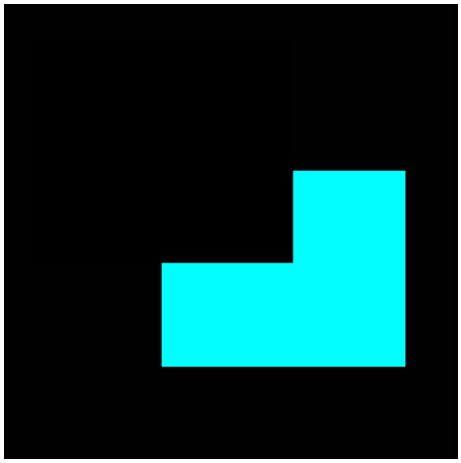

Ilustración 158. División de imágenes.

Aparentemente el resultado es igual que en la resta, pero nada más lejos de la realidad. Si se analizan en detalle los píxeles, no toda la parte negra realmente es negro puro. Esto es debido a que cuando se divide blanco (255, 255, 255) entre blanco (255, 255, 255) el

resultado es todos unos (1, 1, 1). En caso de dividir negro (0, 0, 0) entre blanco (255, 255, 255), el resultado será negro puro (0, 0, 0). En la parte que interseccionan los cuadrados rojo y negro la división será negro puro, ya que se divide entre 0. Por último, en la zona blanca (255, 255, 255) con intersección roja (255, 0, 0), se debe pasarán los ceros del denominador a unos, y por lo tanto el color mostrado será 1 para el canal rojo y 255 para los canales verde y azul.

La operación *Multiplicación* busca los píxeles con las mismas coordenadas y multiplica sus 3 canales *RGB* de los valores de los píxeles de las imágenes de entrada. En caso de que algún valor sobrepase los 255, automáticamente pasa a tener valor 255. Si se aplica la multiplicación a las dos imágenes de prueba el resultado sería el siguiente.

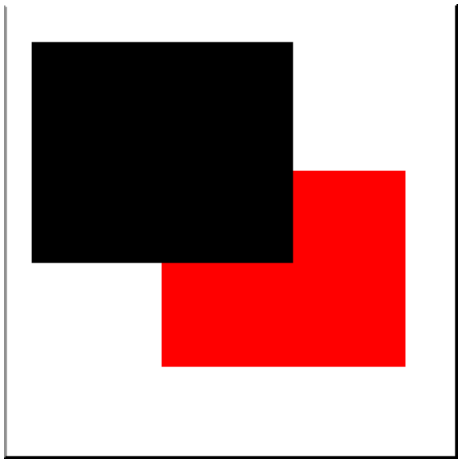

Ilustración 159. Multiplicación de imágenes.

El único detalle a tener en cuenta en esta operación es que, como puede observarse, el cuadrado negro solapa al rojo debido a que al multiplicar negro (0, 0, 0) por rojo (255, 0, 0) obviamente el resultado será negro.

8.5.4.8.2 Operaciones lógicas (AND/OR/XOR)

En este menú se encuentran las operaciones lógicas básicas con dos imágenes. Como se puede observar en la siguiente imagen, se puede aplicar el operador *AND*, *OR* y *XOR* a dos imágenes. Además hay un *checkbox* para que las operaciones no afecten al canal alfa.

No obstante hay que tener en cuenta que este tipo de operaciones tiene más sentido aplicadas sobre imágenes binarias.

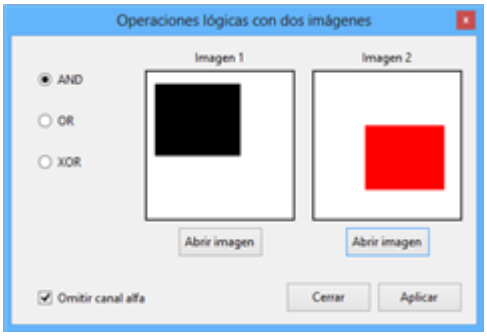

Ilustración 160. Menú operaciones lógicas con dos imágenes.

Un paso previo a la hora de ejecutar la operación es cuadrar las imágenes, es decir, buscar entre las dos imágenes el ancho y alto mínimo, y la imagen de salida tendrá esos valores. Por ejemplo, si tenemos una imagen de 100x75 y otra de 75x100, la imagen resultante tras una operación aritmética será de 75x75.

El operador *AND* busca los píxeles con las mismas coordenadas, y para cada canal RGB aplica el operador lógico AND. A continuación se muestra el operador AND aplicado a la imagen con el cuadrado negro y el cuadrado rojo.

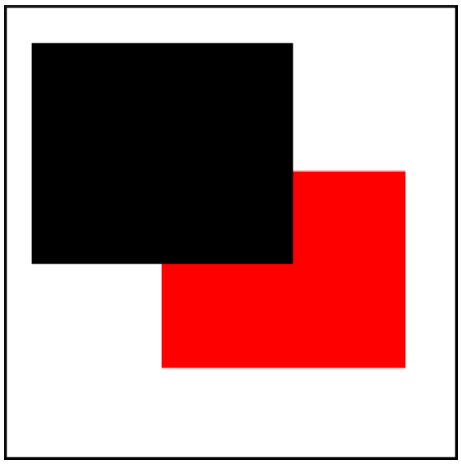

Ilustración 161. Operación AND de dos imágenes.

El resultado de la imagen anterior es fruto del operador *AND*, y de esta manera se unen las imágenes, por lo tanto, las zonas blancas y negras permanecerán del mismo color, y las zonas rojas con intersección negra pasarán a ser negras.

El operador *OR* busca los píxeles con las mismas coordenadas, y para cada canal *RGB* aplica el operador lógico *OR*. A continuación se muestra el operador *OR* aplicado a la imagen con el cuadrado negro y el cuadrado rojo.

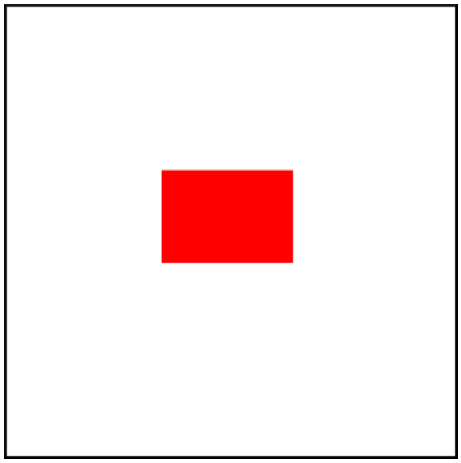

Ilustración 162. Operación OR de dos imágenes.

Al aplicar este operador a las imágenes, el resultado es todo el interior blanco y la zona de intersección de cuadrados pasa a tener color rojo.

El operador *XOR* busca los píxeles con las mismas coordenadas, y para cada canal *RGB* aplica el operador lógico *XOR*. A continuación se muestra el operador *XOR* aplicado a la imagen con el cuadrado negro y el cuadrado rojo.

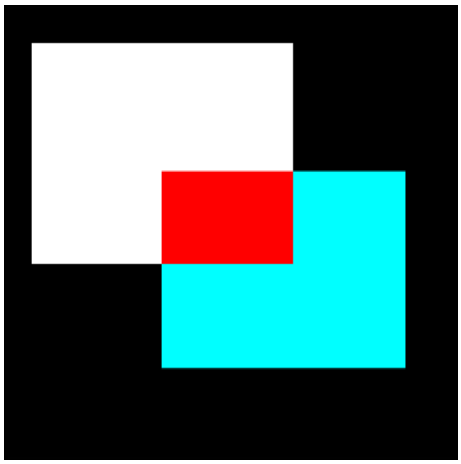

Ilustración 163. Operación XOR de dos imágenes.

Este operador aplicado a dos imágenes invierte todos los colores de la imagen excepto las zonas de intersección que pasan a tener color rojo.

8.5.4.8.3 Anaglifo

Los *anaglifos* son, imágenes de dos dimensiones capaces de provocar un efecto tridimensional, cuando se ven con lentes especiales (lentes de color diferente para cada ojo).

Para poder ver estas imágenes en 3D, serían necesarias unas *gafas anafligo*.

La imagen se forma a partir de dos imágenes muy próximas. Los pasos son los siguientes:

- A la imagen de la izquierda, se le quitan los colores verde y azul, y a la derecha se le quita la componente rojo.
- Se suman las imágenes.

El menú disponible para efectuar esta operación permite seleccionar la imagen de la izquierda y la de la derecha (que deben tener un solape amplio).

Ilustración 164. Menú anaglifo.

Una vez aplicado el efecto sobre una imagen, el resultado sería una imagen anaglifo.

Ilustración 165. Imagen anaglifo.

Como se observa en la anterior ilustración, aparentemente la imagen es normal excepto la zona que se verá en 3D. Esta zona tiene un borde izquierdo de color rojo y un borde derecho de color verde más azul. Visualizando esta imagen con *gafas anafligo* se observaría un efecto tridimensional.

8.5.4.8.4 Comparador de imágenes (Local/Vecinos)

Este menú permite hace comparaciones de imágenes para determinar el grado de similitud entre ellas. Las comparaciones se pueden hacer analizando píxeles únicos en comparación con sus homólogos de la otra imagen, o comparando en función de sus vecinos.

Previamente a comparar las imágenes se deben cuadrar para que tengan el mismo tamaño, esto lo hace automáticamente Apolo. Este proceso trata de buscar entre las dos imágenes el ancho y alto mínimo, y la imagen de salida tendrá esos valores. Por ejemplo, si tenemos una imagen de 100x75 y otra de 75x100, la imagen resultante tras una operación aritmética será de 75x75.

En la opción *comparador imágenes local*, se analizan dos imágenes.

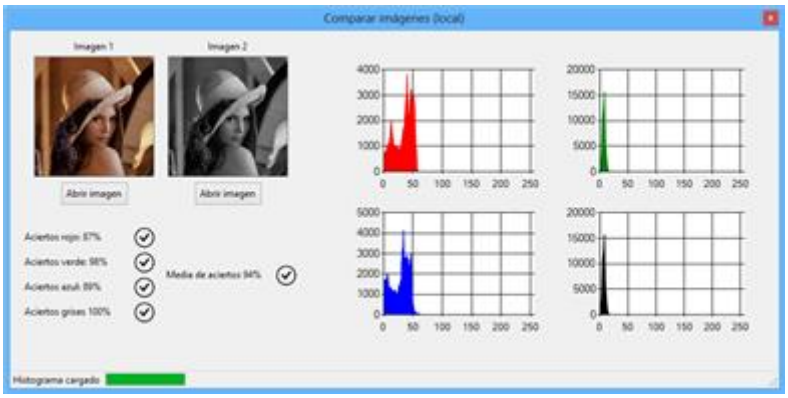

Ilustración 166. Menú comparar imágenes local.

En este caso de prueba se analizará la imagen de Lena original y pasada a escala de grises. Una vez cuadradas las imágenes, se van recorriendo ambas calculando la diferencia (píxel a píxel) entre Lena original y Lena escala de grises. Ejemplificándolo, si por ejemplo el píxel 20,20 de la imagen original tiene un valor (en el canal rojo) de 20, y la imagen en escala de grises un valor de 22, la diferencia es de 2. Cuando la diferencia es 0, el valor es el mismo y se considera un acierto, en cambio, cuando la diferencia es 255 el porcentaje de error en ese píxel es de 100%. Todo este proceso se realiza para los tres canales *RGB*.

Además, se muestra un histograma aproximado del número de aciertos. Analizando el histograma, en la parte de la izquierda (cercano al 0) están los valores con menos porcentaje de error y en la parte de la derecha (255) los valores con mayor porcentaje de error.

Para el caso analizado, el porcentaje total de acierto es del 94% y queda claro que la función trabaja de forma correcta, puesto que el porcentaje de aciertos en escala de grises es 100%. Esto último quiere decir que al pasar la imagen original a escala de grises y comparándola con la segunda imagen (que está en escala de grises) indica que todos los píxeles son iguales.

En la opción *comparador imágenes vecinos*, se analizan dos imágenes.

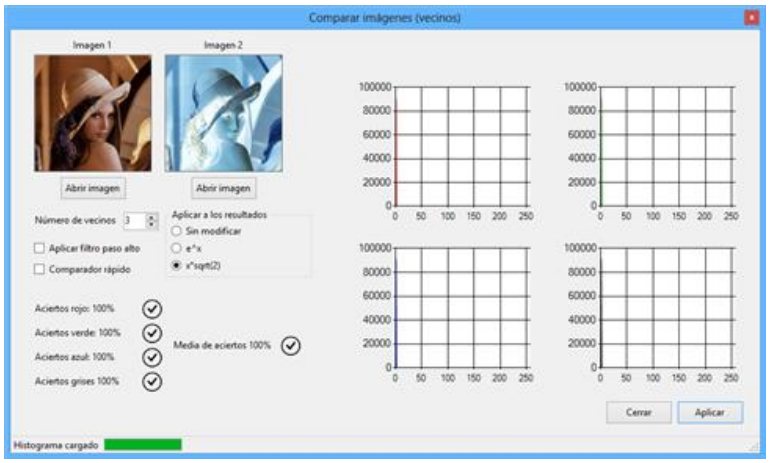

Ilustración 167. Menú comparar imágenes vecinos.

En este caso de prueba se analizará la imagen de Lena original e invertida. El proceso que realiza el comparador es analizar las diferencias entre el píxel evaluado y sus vecinos, y el mismo proceso para la segunda imagen. Una vez calculadas estas diferencias, se calcula la diferencia total entre las dos imágenes y se va acumulando el resultado. En el siguiente ejemplo se puede ver dos píxeles y sus respectivos 8 vecinos. Únicamente se analizará el color rojo.

Píxel de la primera imagen:

81	81	80
79	**80**	80
81	80	79

Tabla 3. Píxel central y sus vecinos.

Píxel de la segunda imagen:

80	81	80
79	**81**	79
81	82	79

Tabla 4. Píxel central y sus vecinos.

Se calcula el valor absoluto de las diferencias entre el píxel central y sus vecinos.

Abs(80-81)	Abs(80-81)	Abs(80-80)
Abs(80-79)	**Abs(80-80)**	Abs(80-80)
Abs(80-81)	Abs(80-80)	Abs(80-79)

Tabla 5. Diferencias para el primer píxel.

Abs(81-80)	Abs(81-81)	Abs(81-80)
Abs(81-79)	**Abs(81-81)**	Abs(81-79)
Abs(81-81)	Abs(81-82)	Abs(81-79)

Tabla 6. Diferencias para el segundo píxel.

Tras calcular las diferencias, se procede a calcular la diferencia en valor absoluto entre los valores obtenidos. El resultado se va acumulando.

Abs(80-81) - Abs(81-80)	Abs(80-81) - Abs(81-81)	Abs(80-80) - Abs(81-80)
Abs(80-79) - Abs(81-79)	**Abs(80-80) - Abs(81-81)**	Abs(80-80) - Abs(81-79)
Abs(80-81) - Abs(81-81)	Abs(80-80) - Abs(81-82)	Abs(80-79) - Abs(81-79)

Tabla 7. Diferencias entre los píxeles.

Se opera y se acumula el resultado.

ResultadoAcumulado= (Abs(1-1)) + (Abs(1-0)) + (Abs(0-1)) + (Abs(1-2)) + (Abs(0-0)) + (Abs(0-2)) + (Abs(1-0)) + (Abs(0-1)) + (Abs(1-2))

ResultadoAcumulado=8

Por lo tanto, en el ejemplo, el porcentaje de acierto para el píxel evaluado es de 8 (nótese que si el valor es 0 el porcentaje de acierto es del 100%, y si es 255 el porcentaje de acierto es del 0%).

Dentro de la función comparador de imágenes vecinos, se pueden seleccionar varios parámetros para obtener o ajustar el resultado. Uno de ellos es el número de vecinos evaluados (en el ejemplo anterior son 3).

Previo al comparador de imágenes, se puede activar la opción Aplicar filtro de paso alto, y antes de evaluar las imágenes se aplica una máscara de filtro de paso alto que aumenta los detalles de la imagen (más información sobre este tipo de filtros en la sección de *máscaras*, *filtro paso alto*).

Si se activa la opción comparador rápido, se reducen las imágenes a 50x50 y se comparan con ese tamaño. No obstante, al reducir las imágenes se pierde información de la misma, por lo tanto los resultados pueden ser imprevisibles.

Por último, se permite aplicar a los resultados una función (sin modificar, *e^x*, *x*sqrt(2)*). El aplicar una función lo que hará es exagerar los resultados. Si por ejemplo se selecciona *e^x*, como la curva de esta función en los primeros valores no es muy alta, los valores cercanos al cero (valores con porcentaje de acierto muy alto) apenas se verán afectados, en cambio, a medida que se aleja del 0 los valores se disparan y de esta manera se puede exagerar el número de errores.

Como puede observarse en la anterior ilustración, el porcentaje de aciertos es del 100%. Esto se debe a que las imágenes que se están comparando son la misma a excepción de que a la segunda se le han invertido los colores, pero esto no altera a las diferencias entre sus vecinos que seguirán siendo las mismas.

8.5.4.9 Cloud

Antepenúltima opción del menú superior. En la siguiente imagen se muestra su aspecto y contenido.

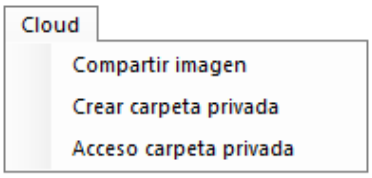

Tabla 8. Menú Cloud.

En esta sección se pueden compartir las imágenes creadas con Apolo para que éstas sean visibles al resto de usuario de Apolo. Además añade funcionalidades básicas para crear su propia carpeta privada a la que únicamente se podrá acceder con sus credenciales, y así disponer de su espacio para guardar sus imágenes privadas.

Con respecto al almacenamiento en el servidor, no se garantiza que las imágenes estén disponibles siempre, podrían perderse sin previo aviso y sin posibilidad de recuperación. Apolo sí garantiza su privacidad, desde el equipo de desarrollo se asegura la no publicación de las imágenes tanto públicas como de sesiones privadas. No obstante, tenga en cuenta que el código fuente es de libre distribución y algún usuario *malévolo* podría hacerse con las contraseñas del servidor para usos fraudulentos.

8.5.4.9.1 Compartir imagen

Se trata de un servidor público donde poder compartir y ver las imágenes que suban los usuarios. Estás imágenes deben subirse con una pequeña descripción y su nombre. Opcionalmente hay una sección para poder valorar las imágenes y así dar una opinión de la misma.

Una vez se ha pulsado sobre la opción *Compartir imágenes*, aparece un pequeño recuadro que indica que se está estableciendo conexión y tras él, se desplegará el formulario principal de Apolo Cloud.

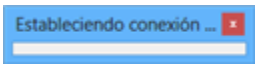

Ilustración 168. Cuadro de diálogo conexión.

Una vez abierta la pantalla principal, se pueden observar las primeras diez imágenes ordenadas alfabéticamente por su nombre. Pulsando los botones superiores, se irán mostrando más imágenes, volver al principio e ir al final.

A continuación se muestra una captura de pantalla del proceso de compartir una imagen. Primeramente se debe seleccionar la imagen que se quiere subir, para ello existe un botón en la parte de la izquierda (*Imagen actual*) que hace que la imagen a subir sea la que está en el formulario principal de Apolo. Una vez se ha seleccionado la imagen que se quiere subir, se deben rellenar los campos *Nombre* (nombre que aparecerá visible en Apolo Cloud) y *Descripción* (una pequeña descripción de que informe sobre la imagen).

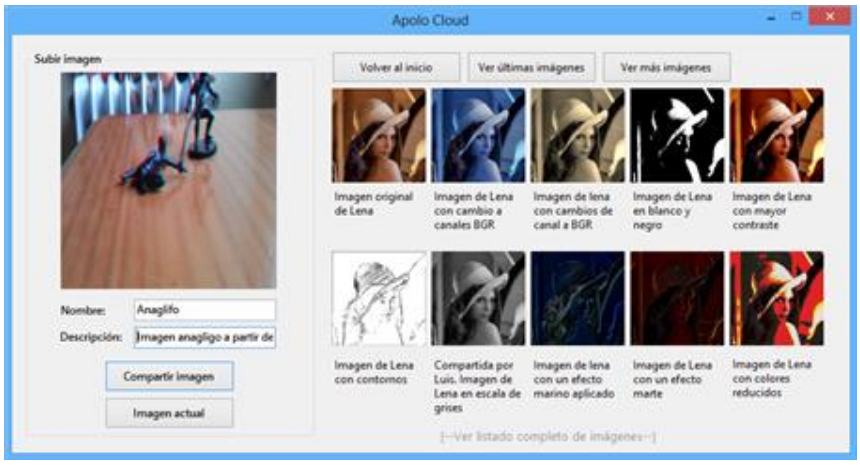

Ilustración 169. Formulario principal Cloud.

Tras este proceso se debe pulsar en el botón Compartir imagen y aparecerá un círculo que indica que se está actualizando los datos.

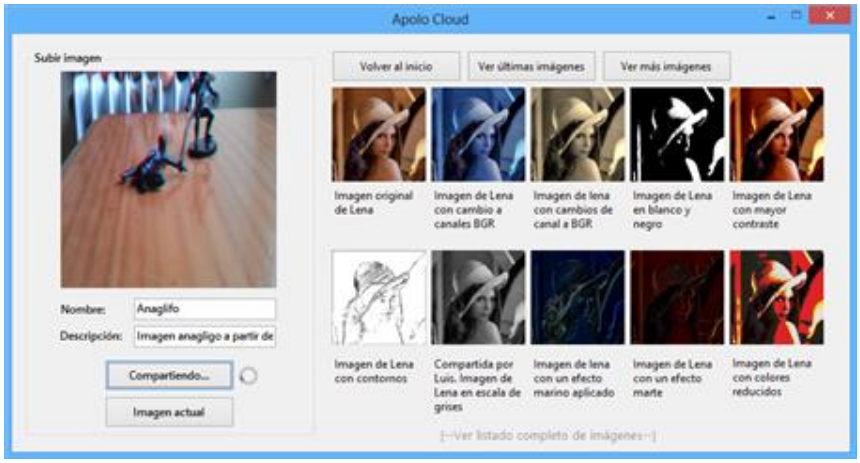

Ilustración 170. Compartiendo imagen.

Tras unos segundos aparecerá un cuadro de diálogo informando de la subida de la imagen. Indicará si el proceso ha sido satisfactorio o si ha ocurrido algún error. En la siguiente imagen se observa el cuadro correspondiente a una subida con éxito al servidor.

Ilustración 171. Imagen compartida con éxito.

Desplegando el menú completo (pulsando en *Ver listado completo de imágenes* en la parte inferior), se puede ver un listado con todas las imágenes del servidor y la posibilidad de valorarlas.

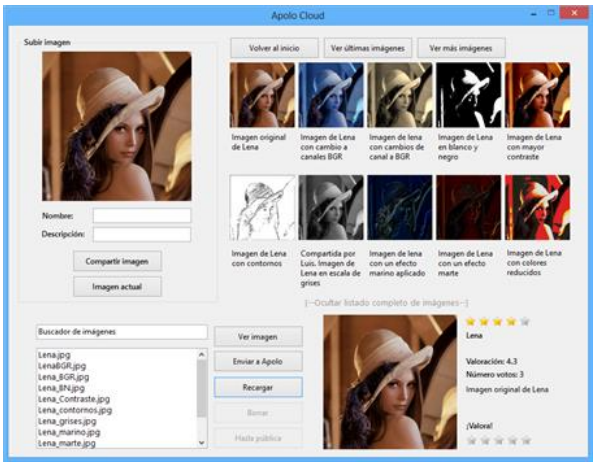

Ilustración 172. Vista completa Cloud.

En la parte izquierda se encuentra la lista con todas las imágenes disponibles, además de un pequeño buscador. Para valorar una imagen primeramente hay que buscarla en la lista y posteriormente pulsar en el botón *Ver imagen*. Tras unos segundos se mostrará en la parte de la derecha la imagen con su nombre, valoración (del 1 al 5), número de votos y descripción. Para valorar la imagen, basta con hacer clic en las estrellas de la parte inferior y esperar unos segundos a que la valoración se haga efectiva.

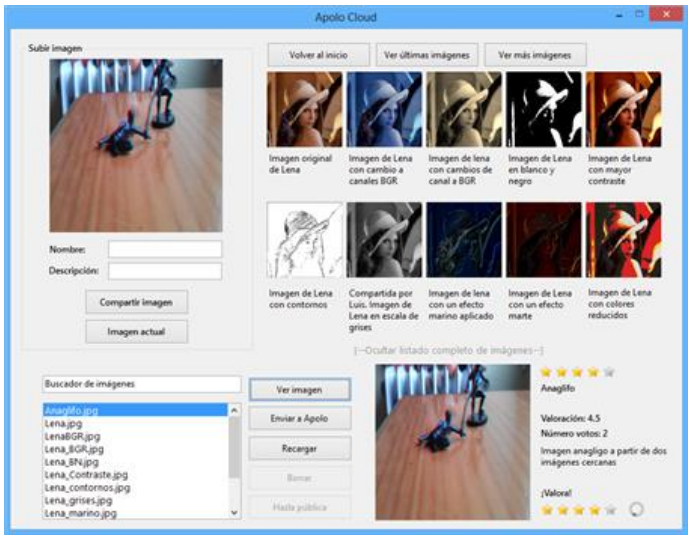

Ilustración 173. Valorando imagen 4 estrellas.

Si no ha ocurrido nada durante el proceso de valoración, aparecerá un cuadro de diálogo agradeciendo su valoración y en unos segundos se recargará la imagen para que aparezca con su nueva valoración.

Por último, el botón *Enviar a Apolo* manda la imagen seleccionada en ese momento a la aplicación principal, pudiendo enviar las imágenes también directamente haciendo clic encima de ella. El botón Recargar actualiza los datos de la lista para visualizar todos los cambios hasta el momento.

8.5.4.9.2 Crear carpeta privada

Este menú sirve para crear una sesión privada donde almacenar imágenes de forma privada.

Ilustración 174. Crear carpeta privada.

Primeramente hay que rellenar los datos que aparecen en la anterior imagen, y pulsar el botón Crear. Tras unos segundos, y si no ha ocurrido ningún error, se abrirá su sesión privada en la que poder guardar las imágenes de forma no pública.

Si al crear la carpeta privada, se selecciona la opción *Recordar credenciales*, su usuario y contraseña se almacenarán en Apolo y serán recordadas hasta la próxima instalación y hasta que el usuario las modifique.

8.5.4.9.3 Acceso a carpeta privada

Esta opción permite, una vez creada una carpeta privada, iniciar sesión con sus credenciales para poder acceder a su carpeta privada.

Ilustración 175. Acceso a carpeta privada.

Una vez iniciada la sesión, automáticamente se abre la ventana principal de Apolo Cloud con su sesión. La carpeta es la misma que en la opción de Compartir imagen a excepción de que en la parte inferior aparecen dos botones, uno para hacer la imagen pública (automáticamente podrá verse en la sesión compartida de Apolo Cloud, junto con el nombre del usuario que la comparte), y la opción de eliminar una imagen.

Si no recuerda los datos de su sesión, puede pulsar (dentro de Acceso a carpeta privada) en la opción de *recuperación de contraseña* y aparecerá un pequeño menú donde debe introducir su cuenta de correo.

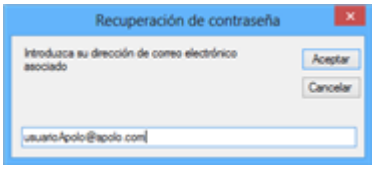

Ilustración 176. Recuperación contraseña.

Tras unos segundos, y si ha introducido correctamente la dirección, le será enviado a su correo electrónico su usuario y contraseña. Tenga en cuenta que estos datos se envían directamente como texto plano, es decir, alguien con acceso a su correo los podría visualizar.

8.5.4.10Herramientas

Penúltima opción de la barra superior de herramientas. En la siguiente ilustración se muestra su aspecto y funciones.

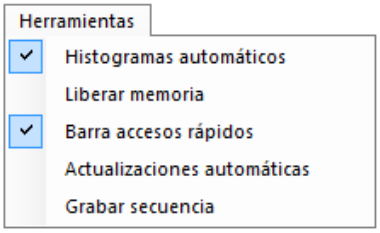

Ilustración 177. Menú herramientas.

Se compone de varias opciones para modificar el aspecto y funcionalidades de Apolo, además de la posibilidad de grabar una secuencia de actuación sobre una imagen.

8.5.4.10.1 Histogramas automáticos

Esta opción activa o desactiva los histogramas mostrados en la parte derecha de Apolo. Si la opción se mantiene activa, al hacer una modificación sobre la imagen principal en 5 segundos se actualizan los histogramas.

8.5.4.10.2 Liberar memoria

Una vez seleccionada esta opción, se muestra un cuadro de diálogo advirtiendo de que se perderán todas las imágenes almacenadas. Esto quiere decir que desde el momento en que se libera la memoria, no se podrá hacer retroceso de las imágenes puesto que sólo permanecerá la imagen actual.

Este proceso puede ralentizar el programa unos instantes, e incluso generar algún error no controlado. En este último caso, es aconsejable notificarlo.

8.5.4.10.3 Barra de accesos rápidos

Esta opción muestra u oculta la barra superior de accesos rápidos, formada por imágenes y situada en la parte superior, justo debajo de la barra de herramientas.

Actualizaciones automáticas

Esta opción, en caso de estar activada, al arrancar el programa busca actualizaciones, y en caso de encontrar alguna versión más reciente de Apolo lo notifica dando la opción de descargar e instalar la aplicación en ese momento.

8.5.4.10.4 Grabar secuencia

Esta interesante función, permite automatizar y exportar una serie de funciones predefinidas para poder ahorrar tiempo a la hora de modificar imágenes. El formulario puede verse en la siguiente imagen.

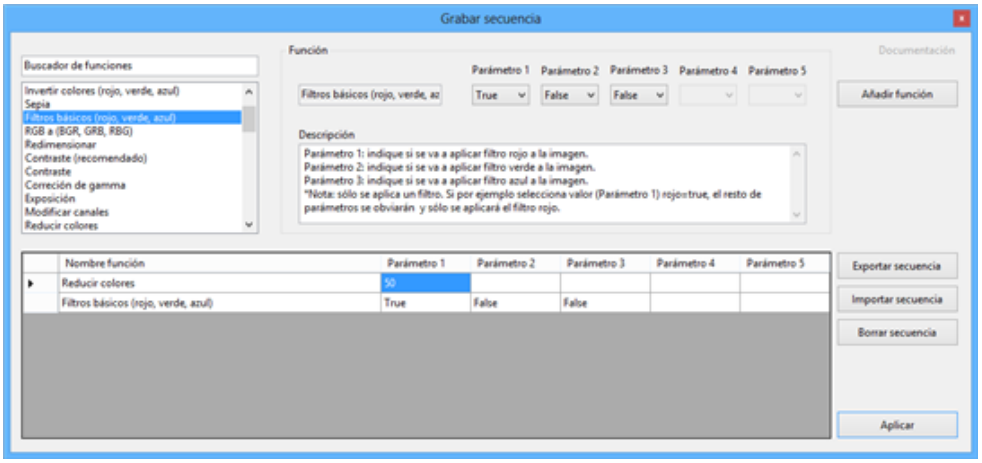

Ilustración 178. Grabar secuencia.

Lo que se pretende con esta función es poder seleccionar una serie de funciones (con sus correspondientes parámetros) y poder aplicarlas de forma automática. Como puede observarse en la imagen. En la parte de la izquierda se muestran todas las funciones disponibles que se pueden utilizar. Una vez seleccionada la función (en la imagen *Filtro básico*), en la parte de la derecha se muestran los parámetros disponibles y justo debajo la descripción de cada parámetro. Adicionalmente, si se quiere saber más sobre la función, se puede pulsar en la opción habilitada en la parte superior derecha para abrir la documentación.

Después de haber seleccionado la función y sus parámetros, se debe pulsar en *Añadir función*, y se mostrará en la tabla inferior. Dentro de la tabla se puede modificar los parámetros directamente sin necesidad de volver a añadir una función.

Una vez creada la secuencia y si se quiere guardar para posteriores usos, se debe pulsar en el botón *Exportar secuencia* y mostrará un cuadro de diálogo para definir su nombre. Tras guardarlo, pulsando en el botón *Importar secuencia* se mostrarán todas las secuencias guardadas para poder abrirlas.

Finalmente, para aplicar la secuencia creada o importada, basta con pulsar el botón *Aplicar* y se ejecutarán los pasos. En caso de que no se realicen todos los pasos, revise que los parámetros de las funciones son correctos. Tenga en cuenta que si algún parámetro es incorrecto, no se mostrará un mensaje de error sino que se detendrá la ejecución de la secuencia.

8.5.4.11Ayuda

Última opción de la barra superior de herramientas. En la siguiente ilustración se muestra el menú.

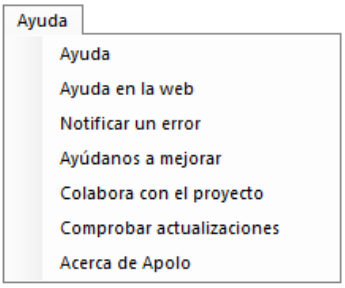

Ilustración 179. Menú ayuda.

Se compone de las opciones básicas de ayuda para Apolo.

8.5.4.11.1 Ayuda

Abre la ayuda de Apolo en formato *CHM*.

8.5.4.11.2 Ayuda en la web

Abre la ayuda de Apolo en el navegador predeterminado.

8.5.4.11.3 Notificar un error

Muestra un cuadro de diálogo donde se puede informar el equipo de desarrolladores de un error existente en Apolo. El formulario se muestra a continuación.

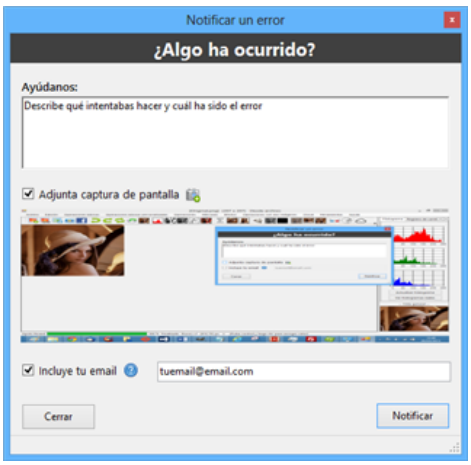

Ilustración 180. Notificar un error.

Como puede observarse en la ilustración anterior, hay un campo en el que poder explicar cómo ha ocurrido el error y en qué le ha afectado. Además hay una opción para adjuntar una captura de pantalla, siendo esto de gran ayuda si el error es visual. Si se selecciona la opción de incluir email, se mantendrá informado al usuario de los avances y detección del error.

Apolo mantiene sus datos a salvo, no se preocupe, su email no será destinado a ningún fin más que mantenerle informado del error. Sus datos no se almacenarán en ningún sitio.

8.5.4.11.4 Ayúdanos a mejorar

En este menú se muestra una pequeña encuesta que se puede rellenar para fomentar la interactividad con el usuario final y así poder adecuar Apolo a las necesidades del usuario.

Ilustración 181. Encuesta.

Si contesta esta encuesta, Apolo y todo el proyecto lo agradecerá.

8.5.4.11.5 Colabora con el proyecto

Informa de los métodos para colaborar de forma activa con el proyecto.

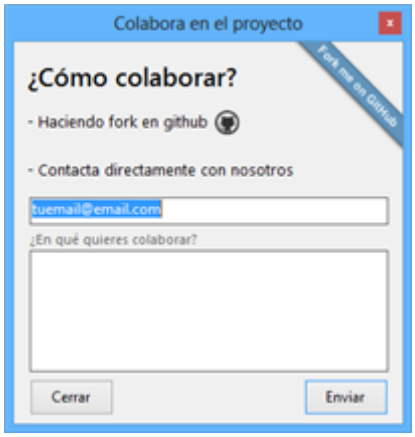

Ilustración 182. Colaborar con el proyecto.

Se puede colaborar rellenando directamente el formulario, o pulsando en alguna de las dos imágenes que abrirán su navegador en la ventana principal del proyecto alojado en *Github*.

8.5.4.11.6 Comprobar actualizaciones

Una vez se ha seleccionado esta función, tras unos instantes se mostrará un cuadro de diálogo informando si hay alguna actualización disponible.

En caso de haber actualizaciones, se da la opción de descargarlas e instalar en ese mismo momento.

8.5.4.11.7 Acerca de Apolo

Abre el cuadro de diálogo con la información básica de Apolo (versión, autores, etc.). Además se detecta la versión de Apolo y, en caso de haber una nueva versión, se exponen sus mejores y aparece la opción de descargarla.

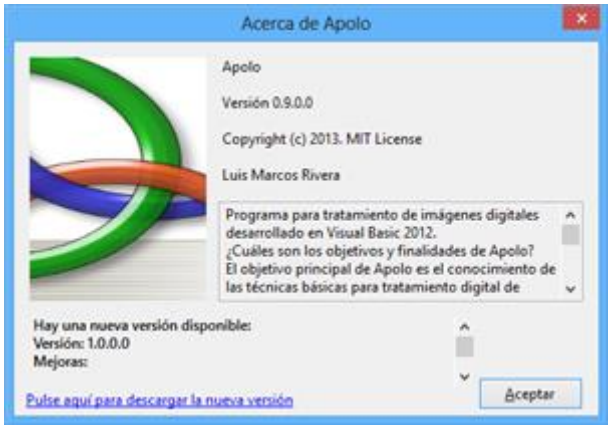

Ilustración 183. Menú Acerca de Apolo.

9 ANEXO II. DOCUMENTACIÓN TÉCNICA. DESARROLLAR CON CLASEIMAGENES.DLL.

9.1 ¿Qué es ClaseImagenes.dll?

ClaseImagenes.dll es una biblioteca de clases desarrollada en Visual Basic .NET, siendo ésta compatible con Microsoft .NET Framework 4. Aunque se ha desarrollado sobre Framework 4, la mayoría del código es compatible con Framework 2, por lo que usted puede desarrollar una aplicación sirviéndose de ClaseImagenes.dll aunque su aplicación esté destinada para clientes Framework 2. Además, gracias al ecosistema .NET, la biblioteca puede ser utilizada desde cualquier otro lenguaje del entorno, como es C# o F#.

La biblioteca está compuesta por una gran cantidad de funciones para tratamiento digital de imágenes, desde transformaciones a escala de grises, binarización, hasta otras más complejas como grabar secuencias de transformaciones para poder automatizarlas. Además, se ha creado con el objeto de que su distribución sea libre tanto en formato de biblioteca de clases (*dll*) para un uso rápido, como en forma de clase (con código fuente) para poder hacer las modificaciones que el usuario crea necesaria. Uno de los objetivos de la biblioteca ha sido el que un usuario que la utilice, se encuentre con un conjunto tan amplio que le permita gestionar todo lo relacionado con el tratamiento digital de imágenes, y que además de las transformaciones, pueda gestionar el zoom, hacer/deshacer, obtención de imágenes (desde archivo, BING, recurso web, etc), y mucho más, de forma totalmente transparente, simplemente teniéndose que preocupar de llamar a las funciones, propiedades o procedimientos oportunos y olvidándose del resto.

9.2 ¿Cómo empezar a utilizar ClaseImagenes.dll?

9.2.1 ¿Cómo adquirirla?

Las vías de adquisición son varias. Usted puede encontrarla en forma de clase (con código fuente), descargando la aplicación Apolo (disponible en git/github y en sourceforge), o bien adquiriendo la biblioteca. A continuación se muestran las diferentes vías para adquirirla:

- En sourceforge: http://sourceforge.net/users/luis-net
- Accediendo a la web: http://algoimagen.blogspot.com.es/
- Clonando el repositorio o haciendo *fork* en Github:
  - git clone https://github.com/LuisM000/TratamientoImagenesVB.NET.git
  - *Fork* en https://github.com/LuisM000/TratamientoImagenesVB.NET
- Enviando un correo a: luis.m.r@outlook.com // u138568@usal.es

9.2.2 ¿Cómo agregar la biblioteca de clases a un proyecto?

Una vez ha adquirido la clase y quiere empezar a utilizarla, abra su IDE (Visual Studio o Mono) y agréguela. En el ejemplo siguiente se va a desarrollar todo el proceso con Visual Studio 2012 Express.

Primeramente, se abre VS Express y se selecciona nuevo proyecto (Archivo/Nuevo proyecto), en este caso un proyecto *Windows Forms*, y tras darle un nombre (*ProyectoUsoClaseImagenes*) se hace clic en *Aceptar*.

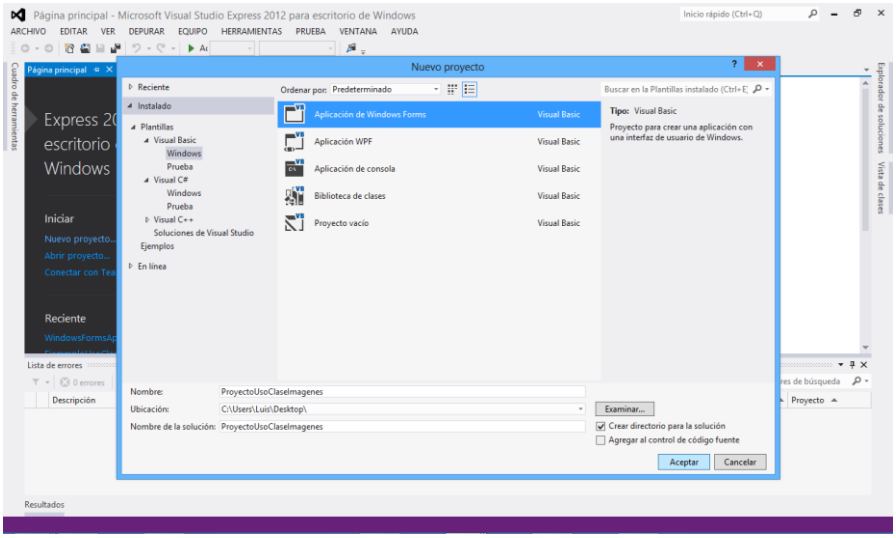

Ilustración 184. Proyecto Windows Forms.

Tras unos segundos tendrá el formulario principal de su proyecto. Ya sólo queda el último paso, agregar la biblioteca ClaseImagenes.dll. Lo único que tiene que hacer es, en el explorador de soluciones (*Ver/Explorador de soluciones*), pulsar encima del proyecto con el botón derecho y seleccionar *Agregar referencia*, tal y como se muestra en la siguiente imagen.

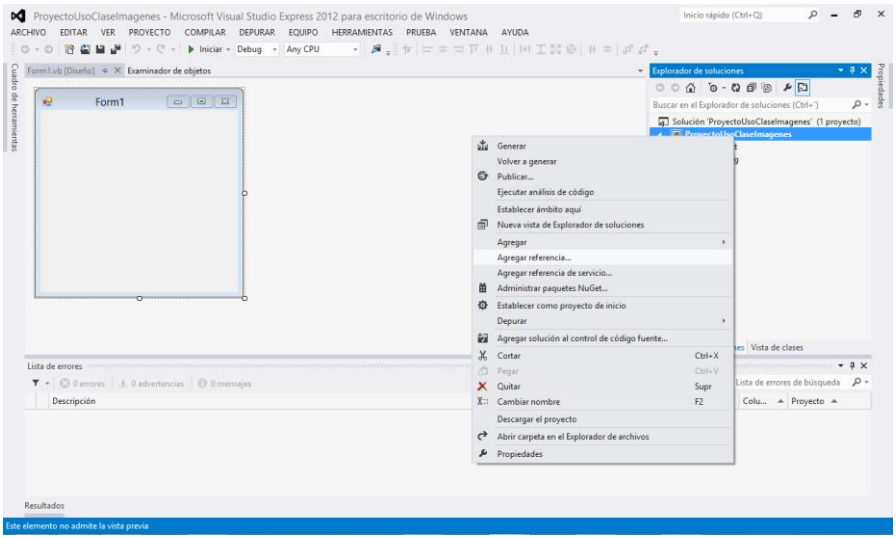

Ilustración 185. Agregar referencia a biblioteca.

Una vez realizado esto, se mostrará un cuadro de diálogo donde se pueden agregar las diferentes referencias al proyecto. Se selecciona la opción *Examinar* y debe localizar ClaseImagenes.dll.

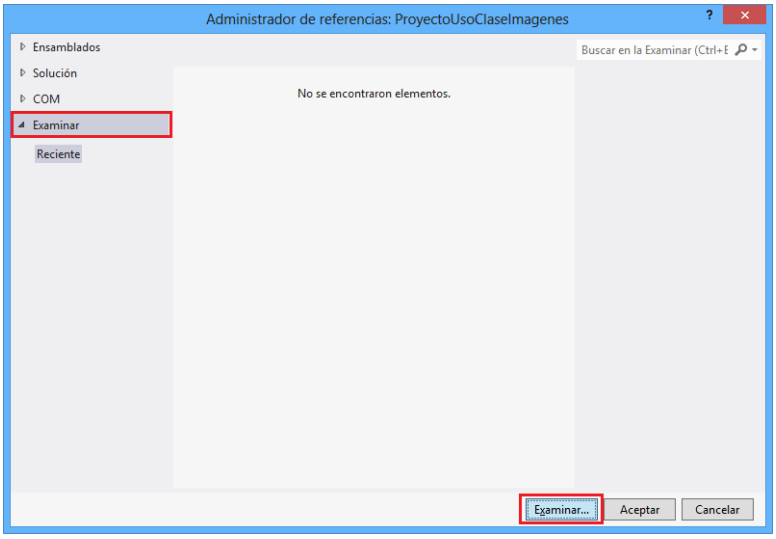

Ilustración 186. Ventana para agregar referencias.

Una vez se ha importado la biblioteca de clases, se debe pulsar en aceptar y su proyecto ya está listo para empezar a utilizar la biblioteca de clases ClaseImagenes.dll.

9.2.3 Primer ejemplo

En el anterior paso se ha agregado la referencia a la biblioteca de clases en un proyecto *Windows Forms*. Ahora sólo queda incluir en la cabecera del código la sentencia *Imports* para hacer referencia a las funciones sin necesidad de poner el espacio de nombres.

Haga doble clic en una zona vacía del formulario principal y una vez se haya mostrado el código, tiene que escribir en la parte superior lo siguiente:

```
Imports ClaseImagenes.Apolo
```

En la imagen se observa el proceso.

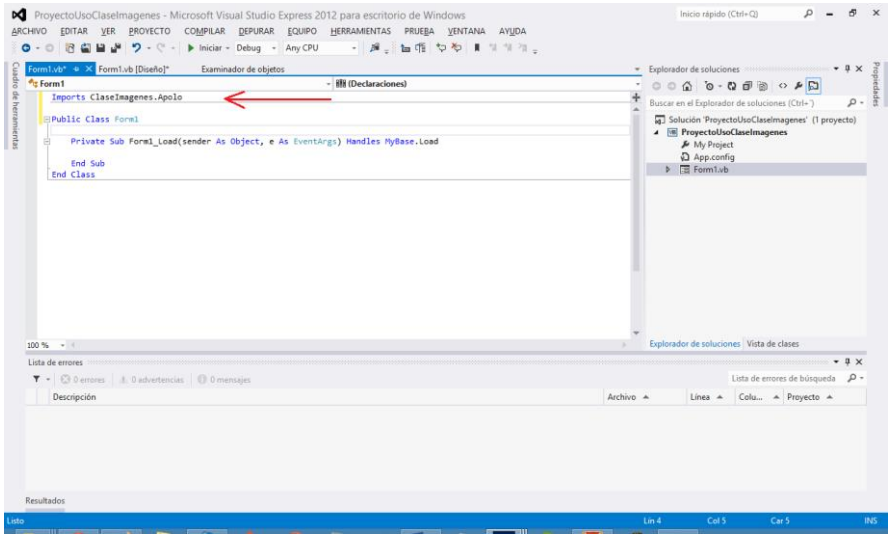

Ilustración 187. Sentencia Imports.

Una vez realizado esto, se va a mostrar un pequeño ejemplo de cómo transformar una imagen a tonos sepia utilizando la función *Sepia* de la biblioteca de clases. Para ello, diríjase a la parte de diseño de su formulario y agregue dos *Picturebox* arrastrándolos directamente desde el *Cuadro de herramientas* (*Ver*/*Cuadro de herramientas*) y también agregue un botón (*Button*). A uno de los dos *Picturebox* se le deberá añadir una imagen desde las propiedades del *Picturebox* (*Ver*/*Ventana de Propiedades*), para ello se deberá utilizar la propiedad *Image* y se importará una imagen desde su pc.

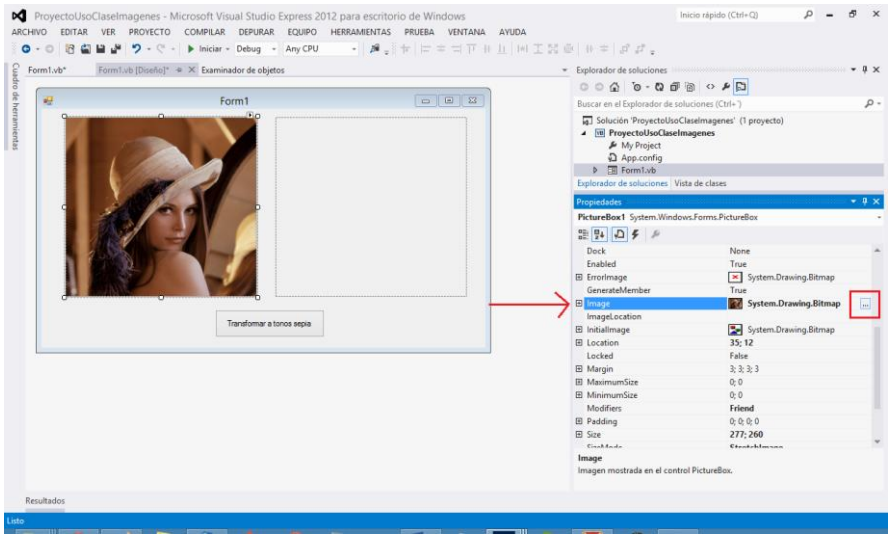

Ilustración 188. Importar imagen a Picturebox.

Una vez se ha importado la imagen, se va a proceder a utilizar el evento clic del botón. Para hacer uso de él, se debe hacer doble clic sobre el botón (en la imagen anterior el botón se llama *Transformar a tonos sepia*) y se mostrará de nuevo el código fuente, en este caso asociado al clic del botón. Ahora, lo siguiente es crear un objeto de la clase *Tratamiento* y utilizar la función *Sepia*. Antes que nada, indicar que el formato de imágenes que soporta la clase en casi todas las funciones es *Bitmap*, por lo que hay que transformar la imagen a bitmap previsamente. El código sería el siguiente:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
    'Transformamos la imagen a Bitmap
     Dim ImagenBMP As New Bitmap(PictureBox1.Image)
     'Instanciamos a la clase
     Dim objetoImagenes As New TratamientoImagenes
     'Asignamos al otro Picturebox el resultado de transformar a tonos sepia
     PictureBox2.Image = objetoImagenes.sepia(ImagenBMP)
End Sub
```

Una vez realizado esto, se pulsa en *Iniciar* (*F5*) y se después de compilar se abrirá el formulario.

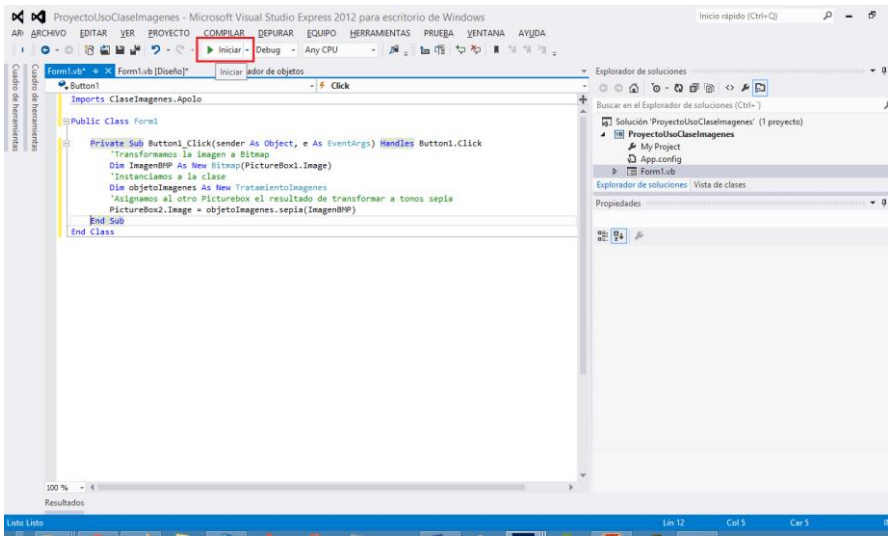

Ilustración 189. Iniciar depuración/compilación.

El resultado es el formulario con la imagen, y tras pulsar el botón, se mostrará en el *Picturebox* de la derecha la imagen transformada a tonos sepia.

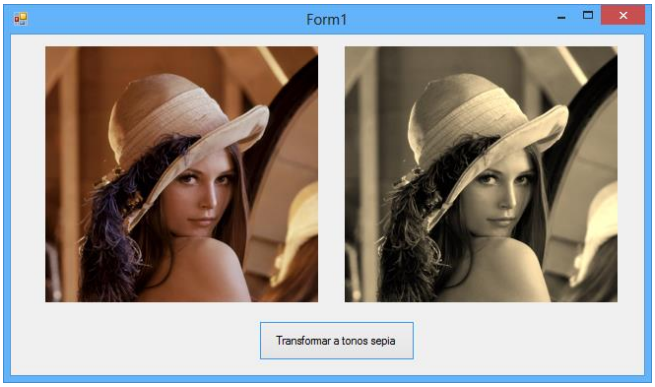

Ilustración 190. Resultado del primer ejemplo.

Como puede observarse el proceso es muy sencillo, simplemente hay que transformar la imagen a *Bitmap*, crear un nuevo objeto de la clase (instanciar clase) y llamar a la función que devolverá la imagen transformada. Tenga en cuenta que, si no se hace al principio del proyecto el *Imports*, el código para crear el objeto sería así:

```
'Instancia a la clase sin sentencia Imports
 Dim objetoImagenes As New ClaseImagenes.Apolo.TratamientoImagenes
```

9.2.4 Referencia de funciones

Para ver cómo funcionan las diferentes funciones, propiedades, eventos, etc., de la biblioteca ClaseImagenes.dll puede revisar la documentación técnica (disponible en la aplicación Apolo) o bien utilizar el examinador de objetos de Visual Studio.

Para utilizar esta última forma, simplemente y tras agregar la biblioteca a su proyecto, vaya al menú *Ver* y pulse la opción *Examinador de objetos*, y se mostrarán todas las funcionalidades de las clases, con ejemplos, explicación de parámetros y comentarios. Y de esta forma podrá ver todos los miembros de la clase y su funcionalidad. En la siguiente imagen se muestra lo explicado.

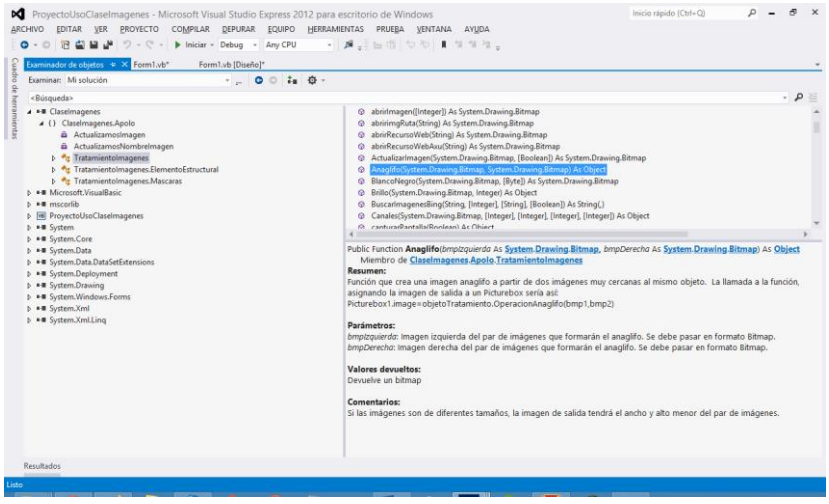

Ilustración 191. Examinador de objetos.

Además, como toda la biblioteca está documentada de acuerdo al procedimiento de Visual Studio y sus comentarios *XML*, al llamar a cualquier miembro de la clase o a sus parámetros, se muestra toda la información en *IntelliSense* de Visual Studio.

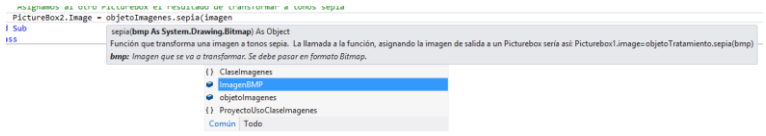

Ilustración 192. IntelliSense de Visual Studio.

9.2.5 Guía para crear un proyecto con ClaseImagenes

En este apartado se intentará iniciar a crear una pequeña aplicación para ver el funcionamiento básico de cómo crear un proyecto valiéndose de las funcionalidades de la biblioteca de clases desarrollada.

9.2.5.1 Aplicación básica

En primer lugar, se mostrará una imagen del resultado final de la aplicación que se va a desarrollar en esta breve guía.

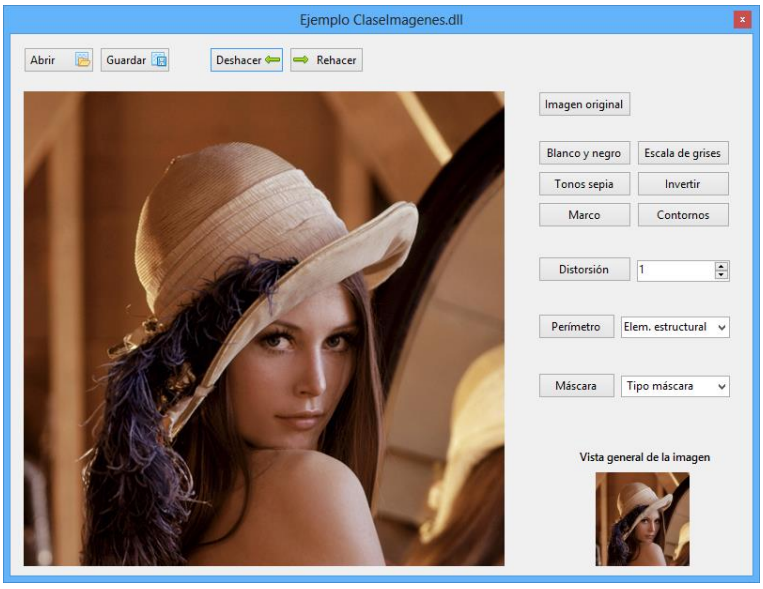

Ilustración 193. Aplicación de prueba.

Como puede observarse, el programa tiene varios botones para aplicar diferentes transformaciones, además de una imagen general (en la parte inferior derecha) y opciones para hacer/deshacer las diferentes transformaciones y abrir/guardar imágenes.

Una vez creado el formulario con los botones, se va a empezar a mostrar el código fuente. Como se había indicado anteriormente, en primer lugar se debe agregar la biblioteca *dll*, y después hacer la sentencia *Imports*.

```
Imports ClaseImagenes.Apolo
```

El siguiente paso será crear un objeto para poder gestionar todos los miembros de la clase.

```
'Se declara un objeto que se utilizará en todo el programa
Dim objetoTratamiento As New TratamientoImagenes
```

Ahora se va a mostrar el código fuente de los diferentes botones.

```
'BOTÓN ABRIR IMAGEN
    Private Sub Button4_Click(sender As Object, e As EventArgs) Handles
Button4.Click
        'Asignamos a una bitmap el resultado de la imagen abierta
        Dim bmpAbierto As Bitmap = objetoTratamiento.abrirImagen()
        'Si se ha seleccionado una imagen (no está vacío), se asigna al
Picturebox
        If bmpAbierto IsNot Nothing Then
            PictureBox1.Image = bmpAbierto
        End If
    End Sub

    'BOTÓN GUARDAR IMAGEN
    Private Sub Button5_Click(sender As Object, e As EventArgs) Handles
Button5.Click
        'Mostramos cuadro de diálogo para guardar
        objetoTratamiento.guardarcomo(PictureBox1.Image, 4)
    End Sub

    'BOTÓN IMAGEN ATRÁS (DESHACER)
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        'Se selecciona la imagen anterior
        PictureBox1.Image = objetoTratamiento.ListadoImagenesAtras
    End Sub

    'BOTÓN IMAGEN ADELANTE (REHACER)
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
        'Se selecciona la imagen siguiente
        PictureBox1.Image = objetoTratamiento.ListadoImagenesAdelante
    End Sub

    'BOTÓN DE IMAGEN ORIGINAL
    Private Sub Button3_Click(sender As Object, e As EventArgs) Handles
Button3.Click
        'Carga la última imagen abierta como original
        PictureBox1.Image = objetoTratamiento.ImagenOriginalGuardada
    End Sub

    'BOTÓN BLANCO Y NEGRO
    Private Sub Button6_Click(sender As Object, e As EventArgs) Handles
Button6.Click
        'Creamos un bitmap con la imagen actual
        Dim bmp As New Bitmap(PictureBox1.Image)
        'Transformamos la imagen a blanco y negro (umbral 128)
        PictureBox1.Image = objetoTratamiento.BlancoNegro(bmp, 128)
```

```
    End Sub

    'BOTÓN ESCALA DE GRISES
    Private Sub Button7_Click(sender As Object, e As EventArgs) Handles
Button7.Click
        'Creamos un bitmap con la imagen actual
        Dim bmp As New Bitmap(PictureBox1.Image)
        'Transformamos la imagen a escala de grises
        PictureBox1.Image = objetoTratamiento.EscalaGrises(bmp)
    End Sub

    'BOTÓN A TONOS SEPIA
    Private Sub Button8_Click(sender As Object, e As EventArgs) Handles
Button8.Click
        'Creamos un bitmap con la imagen actual
        Dim bmp As New Bitmap(PictureBox1.Image)
        'Transformamos la imagen a tonos sepia
        PictureBox1.Image = objetoTratamiento.sepia(bmp)
    End Sub

    'BOTÓN INVERTIR COLORES
    Private Sub Button10_Click(sender As Object, e As EventArgs) Handles
Button10.Click
        'Creamos un bitmap con la imagen actual
        Dim bmp As New Bitmap(PictureBox1.Image)
        'Invertimos los colores de la imagen (para los tres canales RGB)
        PictureBox1.Image = objetoTratamiento.Invertir(bmp, True, True, True)
    End Sub

    'BOTÓN APLICAR MARCO
    Private Sub Button9_Click(sender As Object, e As EventArgs) Handles
Button9.Click
        'Creamos un bitmap con la imagen actual
        Dim bmp As New Bitmap(PictureBox1.Image)
        'Se envuelve la imagen en un marco (el número 3)
        PictureBox1.Image = objetoTratamiento.marco(bmp, 2)
    End Sub

    'BOTÓN DETECTAR CONTORNOS
    Private Sub Button14_Click(sender As Object, e As EventArgs) Handles
Button14.Click
        'Creamos un bitmap con la imagen actual
        Dim bmp As New Bitmap(PictureBox1.Image)
        'Se calculan los contornos
        PictureBox1.Image = objetoTratamiento.contornos(bmp, 15, 100, 100, 100)
    End Sub

    'BOTÓN DE DISTORSIÓN
    Private Sub Button13_Click(sender As Object, e As EventArgs) Handles
Button13.Click
        'Creamos un bitmap con la imagen actual
        Dim bmp As New Bitmap(PictureBox1.Image)
        'Se calculan los contornos
        PictureBox1.Image = objetoTratamiento.Distorsion(bmp,
NumericUpDown1.Value)
    End Sub

    'BOTÓN PERÍMETRO (OPERACIÓN MOROFOLÓGICA)
    Private Sub Button11_Click(sender As Object, e As EventArgs) Handles
Button11.Click
        'Creamos un bitmap con la imagen actual
        Dim bmp As New Bitmap(PictureBox1.Image)
```

```
        'Creamos un objeto que contiene elementos estructurales predefinidos
        Dim objetoEstructura As New TratamientoImagenes.ElementoEstructural
        'Se crea una matriz de dos dimensiones que alojará el elemento
estructural y le damos un valor inicial
        Dim ElemenEstructural(,) As Integer = objetoEstructura.Cuadrado3x3

        Select Case ComboBox1.SelectedItem
            Case "Cuadrado 3x3"
                ElemenEstructural = objetoEstructura.Cuadrado3x3
            Case "Cuadrado 5x5"
                ElemenEstructural = objetoEstructura.Cuadrado5x5
            Case "Diamante 5x5"
                ElemenEstructural = objetoEstructura.Diamante5x5
        End Select
        'Transformamos el bitmap calculando el perímetro mediante una operación
morfológica
        PictureBox1.Image =
objetoTratamiento.MorfologicasPerimetroDilatOrigin(bmp, ElemenEstructural)
    End Sub

    'BOTÓN MÁSCARAS
    Private Sub Button12_Click(sender As Object, e As EventArgs) Handles
Button12.Click
        'Creamos un bitmap con la imagen actual
        Dim bmp As New Bitmap(PictureBox1.Image)
        'Creamos un objeto que contiene elementos estructurales predefinidos
        Dim objetoMascara As New TratamientoImagenes.Mascaras
        'Se crea una matriz de dos dimensiones que alojará la máscara y le
damos un valor inicial
        Dim Mascara(,) As Double = objetoMascara.LineasVerticales

        Select Case ComboBox2.SelectedItem
            Case "Líneas verticales"
                Mascara = objetoMascara.LineasVerticales
            Case "Repujado"
                Mascara = objetoMascara.Repujado
            Case "Laplaciana"
                Mascara = objetoMascara.Laplaciana1
        End Select
        'Transformamos el bitmap operando a través de una máscara
        PictureBox1.Image = objetoTratamiento.mascara3x3RGB(bmp, Mascara)
    End Sub
```

Se puede observar, que el código es muy sencillo, simplemente (para las transformaciones) se define el *Bitmap* que se enviará a la clase y los parámetros si los hubiese. Con respecto a las opciones de hacer/deshacer la sentencia es muy simple y es la propia clase quien gestiona todo.

Un último detalle para poder gestionar el evento de la imagen de detalle que está en la parte inferior, primeramente se debe incluir al manejador en el *load* del formulario.

```
 'INICIO DE FORMULARIO
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Asignamos el gestor que controle cuando sale imagen
         AddHandler objetoTratamiento.actualizaBMP,
           New ActualizamosImagen(AddressOf actualizarPicture)
End Sub
```

Una vez creado el manejador, se incluye un procedimiento que recibirá un parámetro como *Bitmap* y se asignará al *Picturebox2*, y así cuando se modifique (mediante la

biblioteca ClaseImagenes) un *Bitmap* enviado, se mostrará también el cambio en el *Picturebox2*.

```
Sub actualizarPicture(ByVal bmp As Bitmap)
    'Asignamos al Picturebox2 cuando se detecta una transformación de la imagen
    PictureBox2.Image = bmp
End Sub
```

El código fuente completo de la aplicación se puede descargar en el siguiente enlace, http://sdrv.ms/17eui2G.

9.2.5.2 Añadidos a la aplicación básica

Ahora se va a mostrar cómo gestionar diferentes opciones para utilizar varios miembros que dispone la clase. Como primer paso, a la aplicación se le va a añadir un *ToolTip* que mostrará información de qué proceso se va a rehacer o deshacer. Este *ToolTip* se muestra al situar el cursor encima de los botones *Deshacer* o *Rehacer*.

Como primer paso, se va incluirá del cuadro de herramientas un *ToolTip* (denominado *ToolTip1*). El siguiente paso será utilizar el evento de entrada de cursor, asociado a los dos botones. Este evento se denomina *MouseEnter*.

```
'GESTIONAMOS EVENTO DE ENTRADA DEL CURSO EN EL BUTTON 1 (BOTÓN DE DESHACER)
    Private Sub Button1_MouseEnter(sender As Object, e As EventArgs) Handles
Button1.MouseEnter
        'Le damos título al tooltip
        ToolTip1.ToolTipTitle = "Deshacer:"
        'ASociamos al button1 (deshacer) la información de la opción deshacer
        ToolTip1.SetToolTip(Button1, objetoTratamiento.ListadoInfoAtras)
    End Sub
    'GESTIONAMOS EVENTO DE ENTRADA DEL CURSO EN EL BUTTON 2 (BOTÓN DE REHACER)
    Private Sub Button2_MouseEnter(sender As Object, e As EventArgs) Handles
Button2.MouseEnter
        'Le damos título al tooltip
        ToolTip1.ToolTipTitle = "Rehacer:"
        'ASociamos al button1 (deshacer) la información de la opción rehacer
        ToolTip1.SetToolTip(Button2, objetoTratamiento.ListadoInfoAdelante)
    End Sub
```

Ahora se va a mostrar una imagen de qué haría este código mostrado anteriormente.

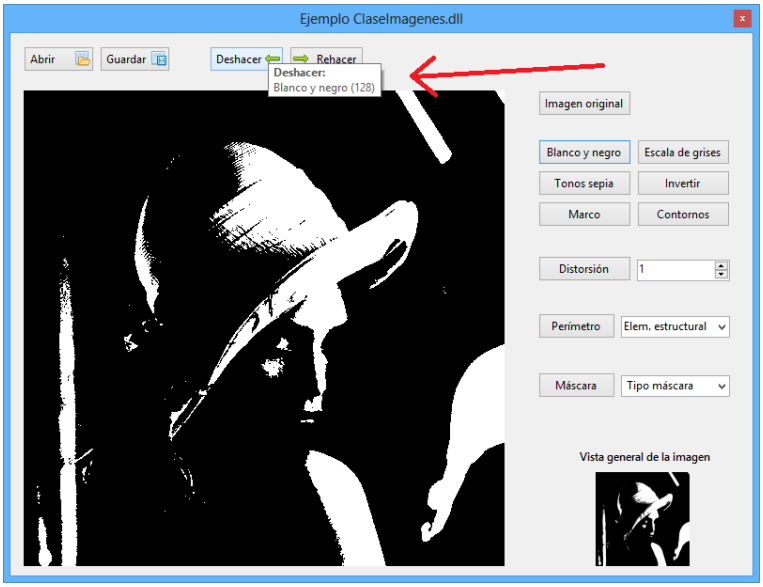

Ilustración 194. Tooltip en botón deshacer.

Ahora se incluirá en la parte superior del programa, en el nombre del formulario, el nombre y tamaño de la imagen que se abra. Para ello, hay que gestionar un evento similar al visto anteriormente que actualizaba la imagen general. Primeramente en el load del formulario se añade el manejador.

```
    'INICIO DE FORMULARIO
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        'Cargamos la imagen en un bitmap (la imagen se ha guardado como recurso
        Dim bmpInicial As New Bitmap(My.Resources.remastered_lena_512x512)
       'ASignamos al picturebox la imagen actualizándola (como imagen original)
        PictureBox1.Image = objetoTratamiento.ActualizarImagen(bmpInicial,
True)

        'Asignamos el gestor que controle cuando sale imagen
        AddHandler objetoTratamiento.actualizaBMP,
           New ActualizamosImagen(AddressOf actualizarPicture)
        'Asignamos el gestor que controle cuando se abre una imagen nueva
        AddHandler objetoTratamiento.actualizaNombreImagen,
          New ActualizamosNombreImagen(AddressOf actualizarNombrePicture)
    End Sub
```

Parte del código anterior ya había sido incluido, lo que interesa realmente es la última sentencia que hará que cuando se produzca un evento, se ejecute el procedimiento *actualizarNombrePicture*. Como siguiente paso, se deberá crear el procedimiento con el nombre indicado anteriormente.

```
 'Realizamos esto cuando recibimos el evento de que se ha abierto nuevo imagen
    Sub actualizarNombrePicture(ByVal nombre() As String)
        'Cambiamos el nombre del formulario principal, y le damos nombre,
tamaño y desde dónde se ha abierto
        Me.Text = "[" & nombre(0) & "]  " & "(" & nombre(1) & " x " &
         nombre(2) & ")   " & nombre(3)
    End Sub
```

Como resultado, y tras abrir una imagen, se mostrará lo siguiente en la cabecera del formulario.

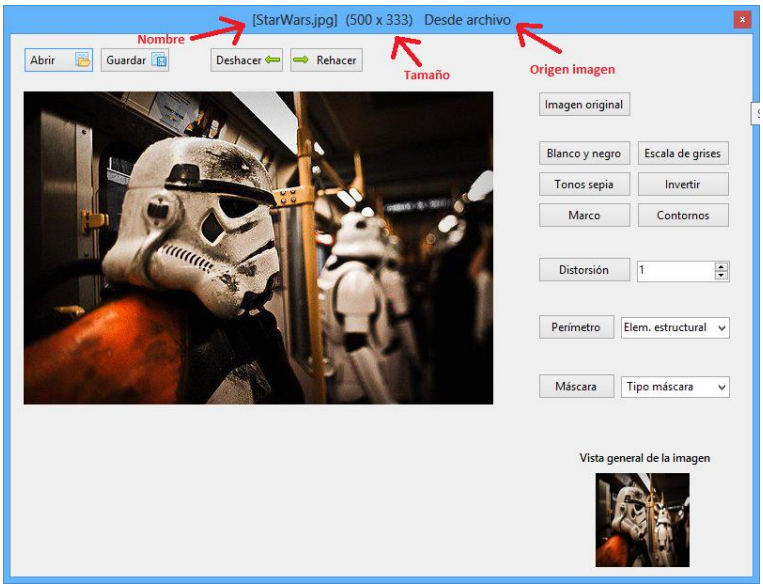

Ilustración 195. Título del formulario.

9.2.5.3 Operaciones avanzadas

Si ha seguido este pequeño tutorial, ya tienes nociones básicas de cómo funciona la clase, pero, ¿no se ha dado cuenta de que mientras se efectúa una transformación el programa se queda congelado? Esto es debido a que leer todos los píxeles de la imagen y luego modificarlos es un proceso muy costoso. La solución es crear un hilo de ejecución aparte que se ejecutará al hacer alguna transformación. Para ejemplificar todo esto, se va a crear un proyecto de cero. Como se ha hecho hasta ahora, se abre Visual Studio y se agrega la referencia a de la biblioteca de clases, se hace el *Imports* y se crea un objeto de la clase para utilizarlo en todo el formulario.

Una vez hecho esto, en el formulario se va a incluir dos *Picturebox* uno de ellos mostrará la imagen original y otro la imagen transformada, además de dos botones. Para ver que realmente al hacer la transformación mediante un hilo el programa sigue activo y no se bloquea, se va a incluir un pequeño *Picturebox* donde se mostrará un *GIF* de carga. Este gif, cuando el programa esté congelado no se llegará a mostrar (o se mostrará estático, es decir, sin animación). El resultado sería algo así.

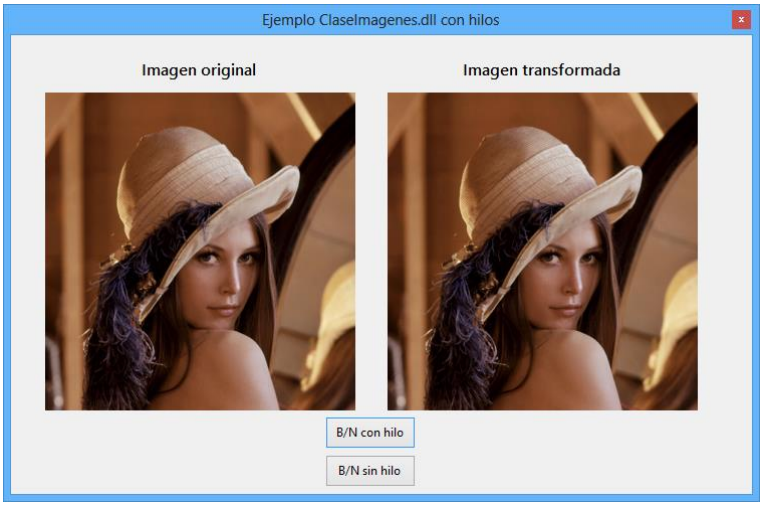

Ilustración 196. Formulario ejemplo con Backgroundworker.

Como se observa, el *Picturebox* que mostrará un *GIF* no se ve, realmente sólo se mostrará cuando se haga la transformación *B/N* (blanco y negro).

Después de lo visto anteriormente, es fácil saber qué código se asociará al botón B/N sin hilo, puesto que se manejará de forma normal. El código es el siguiente.

```
   'BOTÓN DE EJECUCIÓN NORMAL
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
        'Primeramente, se muestra el gif que debería moverse
        PictureBox3.Image = My.Resources.cargandogris
        'Asignamos al Picturebox2 la imagen del Picturebox1 transformada a
blanco y negro
        PictureBox2.Image = objetoTratamiento.BlancoNegro(PictureBox1.Image)
        'Ocultamos la imagen del gif
        PictureBox3.Image = Nothing
    End Sub
```

La única diferencia con el explicado anteriormente es que muestra un pequeño *GIF* en el *Picturebox3*.

Ahora llega lo interesante, que es el botón que trabajará con hilos. Primeramente se debe arrastrar del cuadro de herramientas un control que se denomina *BackgroundWorker*. De este control se van a utilizar dos eventos, el evento que gestiona cuando el *BackgroundWorker* está funcionando (*DoWork*) y el que se activa justo cuando ha acabado de realizar la tarea (*RunWorkerCompleted*). Primeramente se muestra el código del botón.

```
    'BOTóN DE EJECUCIÓN MEDIANTE BACKGROUNDWORKER
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        'Este primer condicional evalúa si el hilo de ejecución
(BackgroundWorker1),
        'está ocupado. Sólo se accionará si no está en funcionamiento.
        If BackgroundWorker1.IsBusy = False Then
            'Primeramente, se muestra el gif que debería moverse
            PictureBox3.Image = My.Resources.cargandogris
            'Este procedimiento activa el BackgroundWorker1 e inicia la
ejecución en segundo plano
```

```
            BackgroundWorker1.RunWorkerAsync()
        End If
    End Sub
```

Lo que hace el código es, primeramente evalúa que el *BackgroundWorker* no esté activo, y si esto es cierto, muestra el *GIF* y llama al *BackgroundWorker* para que ejecute su tarea. El último paso es introducir el código asociado a los eventos de acción del *BackgroundWorker* y el código cuando éste finaliza la tarea.

```
'CÓDIGO EJECUTADO EN SEGUNDO PLANO
    Private Sub BackgroundWorker1_DoWork(sender As Object, e As
System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork
        'Asignamos al Picturebox2 la imagen del Picturebox1 transformada a
blanco y negro
        PictureBox2.Image = objetoTratamiento.BlancoNegro(PictureBox1.Image)
    End Sub
    'CÓDIGO CUANDO SE ACABA LA EJECUCIÓN EN SEGUNDO PLANO
    Private Sub BackgroundWorker1_RunWorkerCompleted(sender As Object, e As
System.ComponentModel.RunWorkerCompletedEventArgs) Handles
BackgroundWorker1.RunWorkerCompleted
        'Ocultamos la imagen del gif
        PictureBox3.Image = Nothing
    End Sub
```

En resumen, los pasos son 3, primeramente desde el botón se llama al proceso principal del *BackgroundWorker* (*DoWork*), éste ejecuta el proceso y por último se ejecuta el código cuando el *BackgroundWorker* ha finalizado (*RunWorkerCompleted*).

Ahora la prueba final, se ejecuta el programa y se demuestra que el *GIF* da vueltas cuando se ejecuta el botón que funciona con un *BackgroundWorker* y no da vueltas en el otro caso.

El código fuente se puede descargar desde aquí, http://sdrv.ms/17euyif.

9.2.5.4 Trabajar desde otros formularios

Si usted quiere trabajar como se ha mostrado anteriormente (mediante *BackgroundWorker*) pero desde otros formularios que no son el principal, tiene que tener unas consideraciones previas.

El caso en concreto es, por ejemplo, si tiene un formulario principal donde se muestra la imagen actual y otro *Picturebox* donde se muestra una imagen general (un ejemplo similar al visto anteriormente) y tiene un segundo formulario donde por ejemplo se muestra un *HScrollBar* para seleccionar el umbral de blanco y negro. Si en este caso quiere trabajar con hilos deberá tener unos detalles en cuenta. Los formularios serían así.

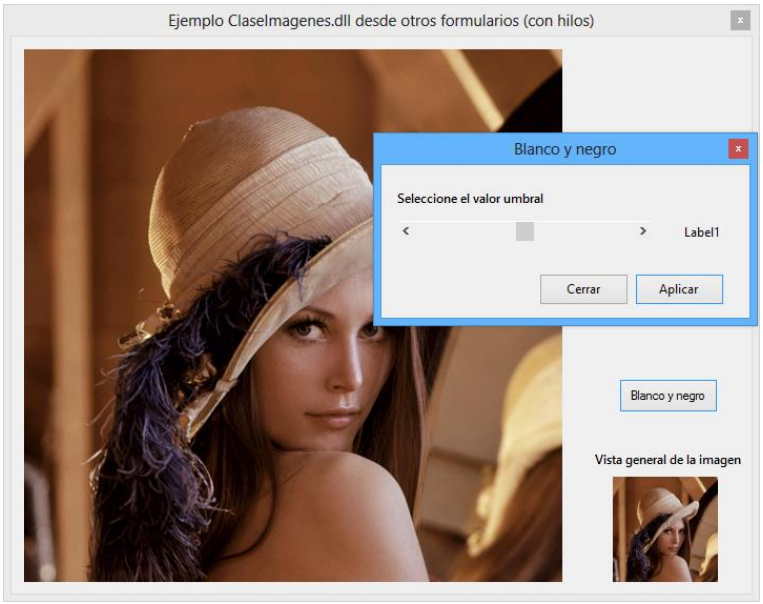

Ilustración 197. Aplicación con varios formularios.

En primer lugar se va a mostrar el código del primer formulario.

```
Imports ClaseImagenes.Apolo
Public Class Form1

    'se declara un objeto que se utilizará en toda el programa
    Dim objetoTratamiento As New TratamientoImagenes

    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        'Asignamos el gestor que controle cuando sale imagen
        AddHandler objetoTratamiento.actualizaBMP,
          New ActualizamosImagen(AddressOf actualizarPicture)
    End Sub
    'Realizamos esto cuando recibimos el evento de que se ha modificado la
imagen
    Sub actualizarPicture(ByVal bmp As Bitmap)
    'Asignamos al Picturebox2 cuando se detecta una transformación de la imagen
        PictureBox2.Image = bmp
    'ASingamos al Picturebox1 cuando se detecta una transformación de la imagen
        PictureBox1.Image = bmp
    End Sub

    'BOTÓN BLANCO Y NEGRO
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        'Abrimos el formulario 2
        BlancoNegro.Show()
    End Sub
End Class
```

El detalle que hay que tener en cuenta es que, en el procedimiento que gestiona el evento cargado en el *Load* del formulario principal (el procedimiento se llama *actualizarPicture*), también modificamos la imagen del *Picturebox1* (el principal), y esto, ¿por qué? Un poco más adelante se verá el motivo.

Ahora se mostrará el código del segundo formulario.

```
Imports ClaseImagenes.Apolo
Public Class BlancoNegro

    'se declara un objeto que se utilizará en toda el programa
    Dim objetoTratamiento As New TratamientoImagenes
    'Creamos un bitmap con la imagen del formulario principal
    Dim bmp As New Bitmap(Form1.PictureBox1.Image)

    Private Sub BlancoNegro_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        Label1.Text = HScrollBar1.Value
        'Asignamos el gestor que controle cuando sale imagen (hacemos la
referencia al procediento del Formulario principal)
        AddHandler objetoTratamiento.actualizaBMP, New
ActualizamosImagen(AddressOf Form1.actualizarPicture)
    End Sub

    'BOTÓN DE APLICAR
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles
Button1.Click
        'Si el hilo (BackgroundWorker1) no está en ejecución, entonces
        If BackgroundWorker1.IsBusy = False Then
            'Se llama al procedimiento en segundo plano
            BackgroundWorker1.RunWorkerAsync()
        End If
    End Sub

    Private Sub BackgroundWorker1_DoWork(sender As Object, e As
System.ComponentModel.DoWorkEventArgs) Handles BackgroundWorker1.DoWork
        'ASingamos al Picturebox1 del formulario principal la imagen
transformada
        Form1.PictureBox1.Image = objetoTratamiento.BlancoNegro(bmp,
HScrollBar1.Value)
    End Sub

    'BOTÓN DE CERRAR
    Private Sub Button2_Click(sender As Object, e As EventArgs) Handles
Button2.Click
        'Se cierra el formulario
        Me.Close()
    End Sub

    'EVENTO DE MODIFICACIÓN DEL SCROLL
    Private Sub HScrollBar1_Scroll(sender As Object, e As ScrollEventArgs)
Handles HScrollBar1.Scroll
        'Se pone en el label el valor actual del Hscrollbar
        Label1.Text = HScrollBar1.Value
    End Sub
End Class
```

Como se observa en el código anterior, también se está gestionando el evento que hace cambiar el *Picturebox1* y *Picturebox2* al detectar cambio, esto va en relación con lo indicado anteriormente con respecto al procedimiento *actualizarPicture*.

Se ejecuta el programa y funciona perfectamente, es decir, se está trabajando con hilos desde otro formulario.

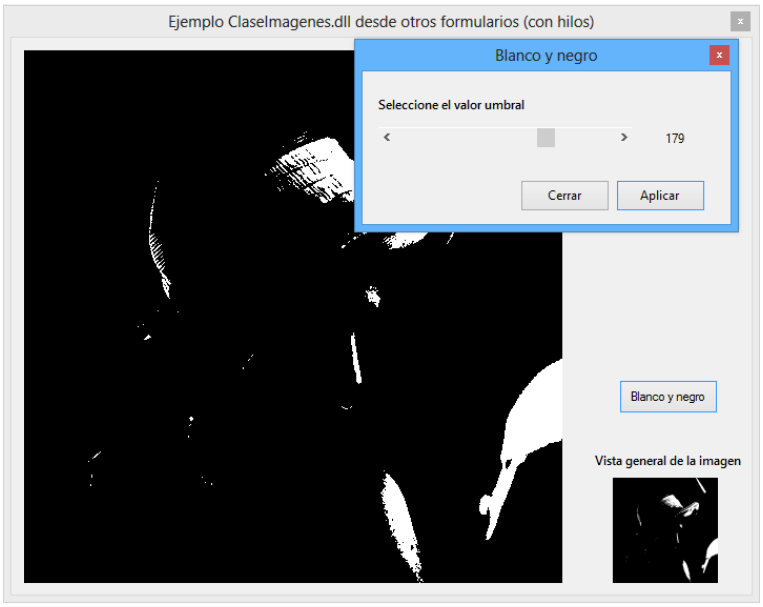

Ilustración 198. Aplicación funcionando con varios formularios.

Ahora se va a ver qué pasaría si, en el procedimiento *actualizarPicture* se quita la sentencia que asignaba al *Picturebox1* el *Bitmap* recibido.

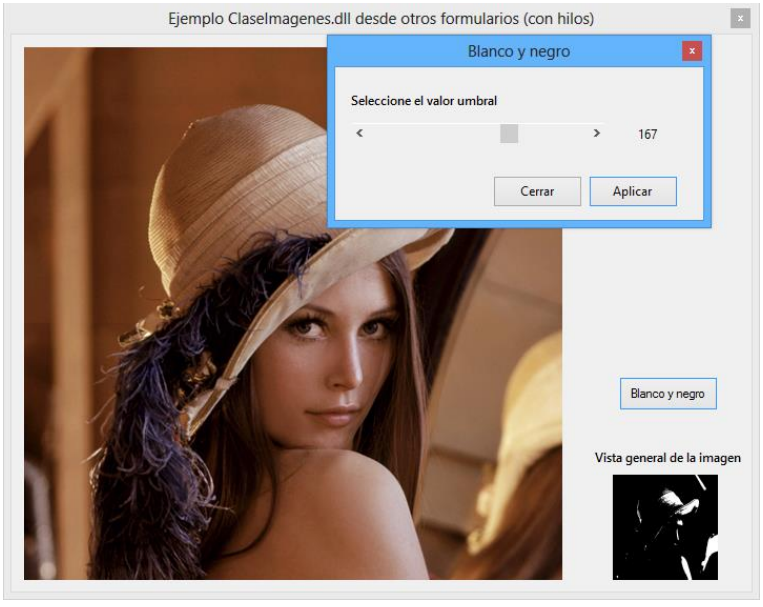

Ilustración 199. Transformación fallida.

¿Qué ha pasado? Como puede verse en la imagen, se ha realizado una transformación y sólo se ha actualizado la imagen secundaria. Esto es debido a que al trabajar con hilos desde otro formulario la sentencia "Form1.PictureBox1.Image = objetoTratamiento.BlancoNegro(bmp, HScrollBar1.Value)" no gestiona de forma correcta la actualización del *Picturebox*. La solución a este problema pasa por, en primer lugar, en el procedimiento (*actualizarPicture*) del formulario principal que gestiona el

evento de transformación en la imagen, incluir también al *Picturebox* principal ("Picturebox1.image=bmp") y en segundo lugar, en todos los formularios diferentes del principal, se debe incluir en el load al manejador ("AddHandler objetoTratamiento.actualizaBMP, New ActualizamosImagen(AddressOf Form1.actualizarPicture").

El código fuente se puede descargar en la siguiente dirección, http://sdrv.ms/17euTBJ.

9.3 ¿Cómo modificar ClaseImagenes?

En este apartado se va a mostrar las pautas para adquirir, modificar y compilar ClaseImagenes y así poder adaptarlo a sus necesidades.

9.3.1 ¿Cómo adquirirla?

Las vías de adquisición son varias para adquirir el código fuente. Descargando la aplicación Apolo (disponible en git/github y en sourceforge), o bien adquiriendo la clase. A continuación se muestran las diferentes vías para adquirirla:

- Desde sourceforge: http://sourceforge.net/users/luis-net
- Accediendo a la web: http://algoimagen.blogspot.com.es/
- Clonando el repositorio o haciendo fork en Github:
  - git clone https://github.com/LuisM000/TratamientoImagenesVB.NET.git
  - Fork en https://github.com/LuisM000/TratamientoImagenesVB.NET
- Enviando un correo a: luis.m.r@outlook.com // u138568@usal.es

9.3.2 ¿Cómo trabajar píxel a píxel?

Una vez ha adquirido la clase y la tiene lista para usarse, ya puede empezar a modificar su código. Como puede observarse la clase está dividida en una primera zona donde están las propiedades y después tres grandes bloques:

- Funciones tratamiento: engloba todo el conjunto de funciones que transforman una imagen que se envía a dicha función. Por ejemplo, blanco y negro, sepia, contornos, etc.
- Funciones abrir: incluye las funciones y procedimientos que se utilizan para abrir imágenes desde varias vías (recurso web, local) y guardarlas.
- Funciones extra: están el resto de funciones que no se hallan en los apartados anteriores.

En esta guía se va a desarrollar una función que transforme una imagen en escala de grises. Esta función ya está incluida en la clase, pero es un fácil ejemplo para aprender cómo se deben incluir nuevas funciones.

Una imagen se compone de un número de píxeles dado por el ancho y alto, y cada píxel está compuesto por 4 canales. Esto cuatro canales (rojo, verde, azul y alfa), están a su vez compuestos por 256 valores posibles, es decir, un byte de información. Al transformar una imagen, generalmente se hacen cambios sobre esos píxeles, es decir, se altera su composición *ARGB* (alfa, rojo, verde, azul).

En el ejemplo que se va a mostrar a continuación se pasará los valores a tonos de gris. Los tonos de gris se caracterizan por tener el mismo valor en todos los canales (*RGB*), siendo el menor valor el color negro (0, 0, 0) y el mayor en blanco (255, 255, 255). Como puede observarse, el canal alfa (la transparencia) no se va a modificar.

Para aplicar esta función hay que calcular, para cada píxel, la media de sus tres componentes (*RGB*) y dicha media aplicarla a los tres canales en el píxel tratado. Si se tiene un píxel con valores; Rojo = 100, Verde = 200, Azul= 50, se debe calcular la media ((100+200+50)/3) y aplicar el valor resultante a los tres canales, por lo tanto, el píxel resultante tendría los siguientes valores para cada canal; Rojo = (100+200+50)/3, Verde = (100+200+50)/3, Azul = (100+200+50)/3, haciendo el cálculo sería, Rojo = 116, Verde = 116, Azul = 116.

Una vez que se conoce la teoría de cómo actuar, se va a proceder a incluir todo esto en la clase.

9.3.2.1 Paso 1. ¿Dónde incluir la función escala de grises?

Si se quiere mantener un orden, debe tener en cuenta la ubicación de la función que se vaya a crear. En este caso, la función es una transformación a escala de grises, por lo tanto su ubicación dentro de la clase estará en la región *FuncionesTratamiento*/*Operaciones básicas*. Allí se alojará la función a la que se va a denominar *EscalaGrises*.

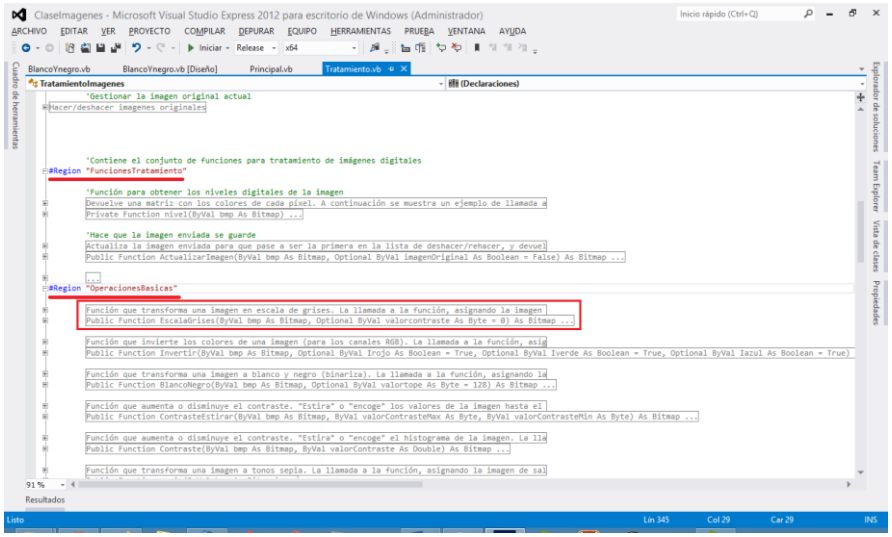

Ilustración 200. Ubicación de la función.

9.3.2.2 Paso 2. Definir la función y sus parámetros.

Una vez se sabe dónde se va a situar, el siguiente paso es crear la función. En este caso, se ha optado por que la función sea pública (puesto que debe ser accesible desde fuera de la clase) y su nombre sea *EscalaGrises*. Esta función, recibirá un único parámetro denomida *bmp* que será un *System.Drawing.Bitmap*, y será enviado por valor. La función

devuelve la imagen transformada a escala de grises, siendo un *Bitmap*. El esqueleto de la función sería así:

```
Public Function EscalaGrises(ByVal bmp As Bitmap) As Bitmap
End sub
```

9.3.2.3 Paso 3. Clonar imagen recibida y leer los valores para cada píxel.

El siguiente paso es crear un *Bitmap* auxiliar para no utilizar directamente el que se recibe como parámetro, que, aunque sea enviado por valor, es recomendable utilizar una copia para todo el proceso. El código sería así:

```
'Copia del bitmap original
Dim bmp2 As Bitmap = bmp
```

A continuación, se van a leer los valores de los píxeles del *Bitmap*. Para ello, existe una función implementada en *ClaseImagenes* que se denomina *nivel*. Esta función devuelve una matriz bidimensional, donde las dimensiones son el ancho/alto del *Bitmap* enviado, y para cada celda de la matriz (cada celda representa un píxel), están disponibles los 4 canales *ARGB*. Esta matriz es de tipo *System.Drawing.Color*.

```
'Matriz que almacenará los niveles digitales de la imagen
Dim Niveles(,) As System.Drawing.Color
'Se obtienen los valores de la imagen
Niveles = nivel(bmp2)
```

9.3.2.4 Paso 4. Recorrer la imagen y calcular el nivel de gris.

Una vez se ha obtenido la matriz con todos los valores de cada píxel, se debe calcular la media de cada píxel para así obtener la imagen en escala de grises. Como primer paso, se van a declarar 5 variables, conteniendo 3 de ellas el valor rojo, verde y azul de cada píxel, una cuarta variable que calculará la media y la última será un *Bitmap* que tendrá el ancho y alto de la imagen original (*bmp*).

```
'Variable para calcular la media
Dim media As Double
'Variables auxiliares que contendrán los valores RGB
Dim rojoaux, verdeaux, azulaux As Double
'Variables que crea una copia de la imagen original
Dim bmpSalida As New Bitmap(bmp.Width, bmp.Height)
```

A continuación, se va a recorrer toda la matriz que contiene los valores de los píxeles de la imagen (niveles digitales). Se va a utilizar un doble bucle *For*, que recorrerá columna por columna la matriz.

```
For i = 0 To Niveles.GetUpperBound(0)  'Se recorre cada píxel de la imagen

        For j = 0 To Niveles.GetUpperBound(1)

        Next
Next
```

Una vez se ha visto cómo recorrer la imagen, se va a proceder a leer los valores de la matriz que contiene los niveles digitales y a calcular su media para asignarla al bitmap de salida (bmpSalida).

```
For i = 0 To Niveles.GetUpperBound(0)  'Recorremos la matriz
       For j = 0 To Niveles.GetUpperBound(1)

            'Se obtienen los valores de la matriz niveles
            rojoaux = Niveles(i, j).R
            verdeaux = Niveles(i, j).G
            azulaux = Niveles(i, j).B

            'Se calcula la media
            media = CInt((rojoaux + verdeaux + azulaux) / 3

            'Se asigna la media a los tres canales RGB
            rojoaux = media
            verdeaux = media
            azulaux = media

            'Se incluye también el canal alfa (opcional)
            alfa = Niveles(i, j).A

            'Con el procedimiento SetPixel se asignan los colores
            bmpSalida.SetPixel(i, j, Color.FromArgb(alfa, rojoaux, verdeaux,
azulaux))
        Next
Next
```

Tras realizar esto, la imagen transformada (*bmpSalida*) a escala de grises ya está creado. Sólo falta retornar el valor.

```
'Se devuelve el bitmap pasado a escala de grises
Return bmpSalida
```

9.3.2.5 Paso 5. Incluir eventos y propiedades de la clase.

La función desarrollada en los puntos anteriores ya es totalmente funcional, pero hay que tener en cuenta unos pequeños detalles que harán que la función sea completamente funcional.

A continuación, se muestra el código fuente de 3 acciones que es conveniente realizar a la hora de crear una función. Una de ellas se gestionará desde la propiedad *progreso*. Esta propiedad se trata de un vector con dos posiciones, en la primera se incluirá el porcentaje de progreso de la transformación y en la segunda, el tipo de transformación que se llevará a cabo. Además de esta propiedad, también se debe almacenar la imagen con el método *guardarimagen* que la almacenará para que pueda que se pueda hacer *deshacer/rehacer* las transformaciones. Por último, hay que generar el evento *actualizaBMP* para que se detecte que hay un cambio en la imagen (y así poder actualizar un *Picturebox* que funcione como imagen miniatura, por ejemplo).

El código fuente completo de clase con estos detalles (resaltados en color), se muestra a continuación:

```
Public Function EscalaGrises(ByVal bmp As Bitmap) As Bitmap
    'Copia del bitmap original
```

```
    Dim bmp2 As Bitmap = bmp

    'Matriz que almacenará los niveles digitales de la imagen
    Dim Niveles(,) As System.Drawing.Color
    'Se obtienen los valores de la imagen
    Niveles = nivel(bmp2)

    'Se actualiza el porcentaje de transformación
    porcentaje(0) = 0
    'Se indica la transformación que se está llevando a cabo
    porcentaje(1) = "Transformando imagen a escala de grises"

    'Variable para calcular la media
    Dim media As Double
    'Variables auxiliares que contendrán los valores ARGB
    Dim rojoaux, verdeaux, azulaux As Double
    Dim alfa As Byte
    'Variables que crea una copia de la imagen original
    Dim bmpSalida As New Bitmap(bmp.Width, bmp.Height)

    For i = 0 To Niveles.GetUpperBound(0)  'Recorremos la matriz
       For j = 0 To Niveles.GetUpperBound(1)

          'Se obtienen los valores de la matriz niveles
          rojoaux = Niveles(i, j).R
          verdeaux = Niveles(i, j).G
          azulaux = Niveles(i, j).B

          'Se calcula la media
          media = CInt((rojoaux + verdeaux + azulaux) / 3

          'Se asigna la media a los tres canales RGB
          rojoaux = media
          verdeaux = media
          azulaux = media

          'Se incluye también el canal alfa (opcional)
          alfa = Niveles(i, j).A

          'Con el procedimiento SetPixel se asignan los colores
          bmpSalida.SetPixel(i, j, Color.FromArgb(alfa, rojoaux, verdeaux,
azulaux))

       Next

    'Se actualiza el porcentaje a medida que se recorre un columna de la matriz
    porcentaje(0) = ((i * 100) / bmp2.Width)

    Next

    'Se indica que la transformación ha finalizado
    porcentaje(1) = "Finalizado"
    'Se almacena la imagen para poder hacer retroceso (deshacer/hacer)
    'Se almacena la imagen y la transformación efectuada
    guardarImagen(bmpSalida, "Imagen transformada a escala de grises")
    'Se genera el evento que informa de que la imagen se ha transformado
    RaiseEvent actualizaBMP(bmpSalida)

    'Se devuelve el bitmap pasado a escala de grises
    Return bmpSalida

End Function
```

Tras realizar esto, la clase ya está de acuerdo con la estructura general de *ClaseImagenes*.

9.3.2.6 Paso 6. Documentar función.

El último paso para completar la función es documentarla con los comentarios *XML* de Visual Studio. Para ello, hay que situarse encima de la función y pulsar tres veces el carácter de comentario (‘). Automáticamente aparece una pequeña estructura *XML* donde se podrá incluir diversa información sobre la función. Para ver todos los detalles de los comentarios, es conveniente acceder a la siguiente web, http://msdn.microsoft.com/es-es/magazine/dd722812.aspx.

Ahora, y después de pulsar tres veces el carácter de comentario encima de la función se va a rellenar la información, y el resultado sería el siguiente.

```
''' <summary>
''' Función que transforma una imagen en escala de grises.
''' <example>Asignando la salida a un Picturebox, la llamada sería así:
''' <code>Picturebox1.image=objetoTratamient.EscalaGrises(bmp)</code></example>
''' </summary>
''' <param name="bmp">Imagen que se va a transformar en formato Bitmap.</param>
''' <returns>Devuelve un bitmap.</returns>
```

Una vez completado esto, la clase está lista para implementarse en la clase *ClaseImagenes*. Como se podría observar, tras incluir los comentarios *XML*, el *Examinador de objetos* mostraría toda la información de la función e *IntelliSense* también.

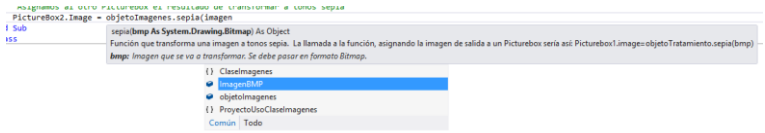

Ilustración 201. Detalle información IntelliSense.

9.3.2.7 Paso 7. Compilar y crear la biblioteca de clases.

Una vez se han seguido todos los pasos anteriores, la clase es totalmente funcional y está totalmente integrada con la clase *ClaseImagenes*. Ahora, si se quiere exportar la clase como Biblioteca de clases, hay que abrir un nuevo proyecto de Visual Basic .NET como Biblioteca de clases. Se le da un nombre y se acepta. En unos segundos tendrá creado un nuevo proyecto.

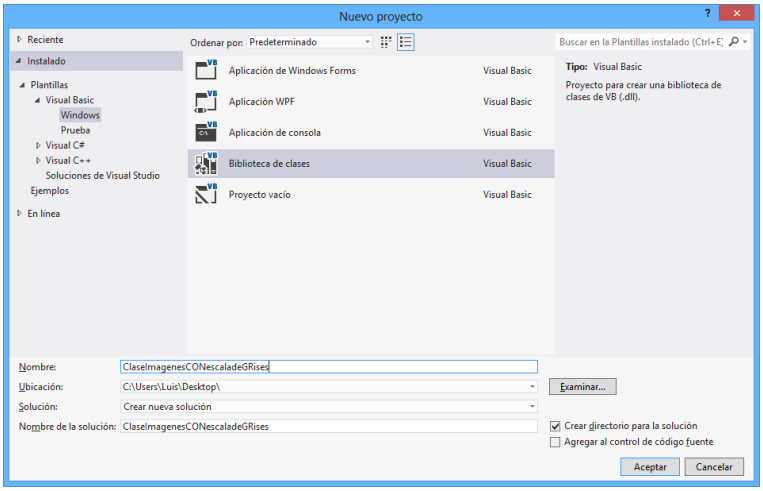

Ilustración 202. Nuevo proyecto como biblioteca de clases.

Una vez realizado esto, aparecerá un único elemento denominado *Class1.vb* el cual eliminaremos. Como siguiente paso, se busca y se agrega la clase ClaseImagenes modificada, y se tendrá un proyecto como Biblioteca de clases el cual únicamente tendrá nuestra clase.

Al agregar la clase se observará que no reconoce las variables *Bitmap*, entre otras. Para solucionar esto, basta con hacer clic con el botón derecho encima del proyecto (en el *Explorado de soluciones*) y seleccionar *Agregar referencia*. En la sección *Ensamblados/Framework* se deben incluir las referencias a *System.Drawing* y *System.Windows.Forms*.

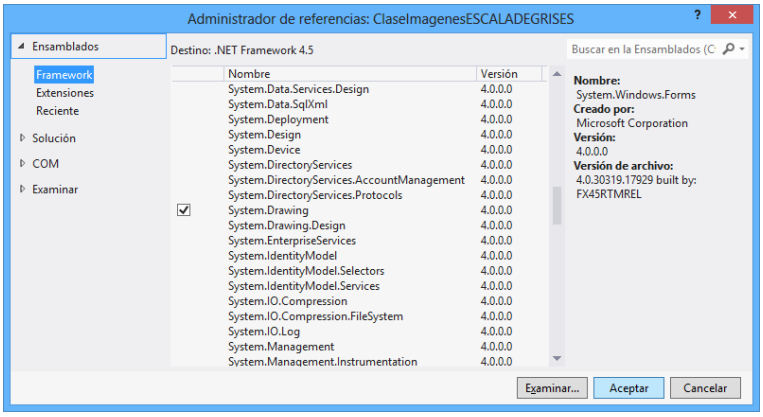

Ilustración 203. Agregar ensamblados.

Ahora sólo faltaría un último detalle. Como se observa en la zona que determina los errores de código, hay 6 errores referentes a que no se encuentra el recurso (*Resources*) de marco, *CineFotos*, *negativoMarco*, *marcoNegro*, *MarcoOndul*. Esto es debido a que la clase a la hora de crear marcos se sirve de 5 imágenes. Para solucionarlo, basta con buscar en la carpeta *Resources* de la clase original (*ClaseImagenes*), estos 5 archivos, y agregarlos como recurso a la biblioteca de clases que se está creando. Esto se hace

desde el *Explorador de soluciones* haciendo doble clic en la sección *My Project* y en la ventana que se abre buscando la sección *Recursos* y agregando las 5 imágenes (que se encuentran como se ha dicho en la carpeta *Resources* del proyecto original).

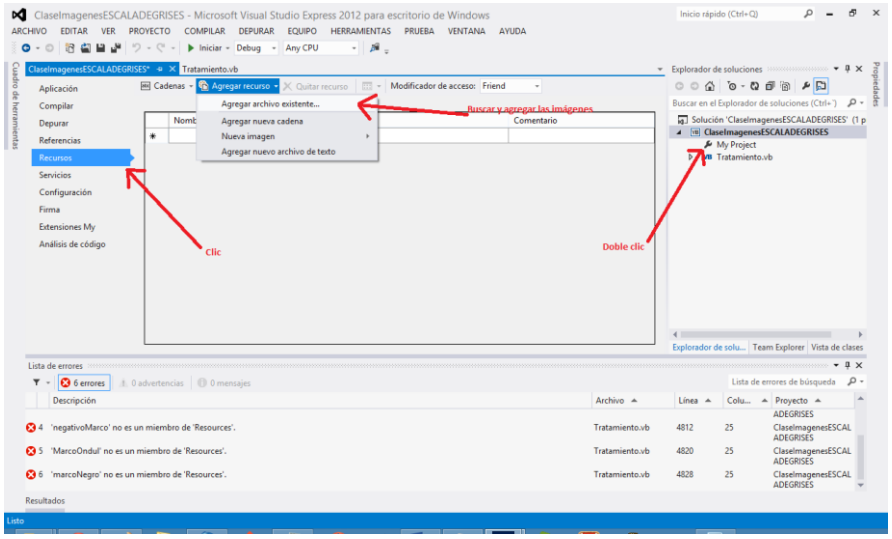

Ilustración 204. Agregar imágenes como recurso.

Una vez agregadas las imágenes, se hace clic en la barra superior en la opción *Compilar*/*Compilar solución*. En unos segundos se habrá compilado correctamente y dentro de la carpeta del proyecto, en la ruta *bin/Debug* o *bin/Release* (depende de cómo se haya compilado), estará disponible la biblioteca lista para su uso.

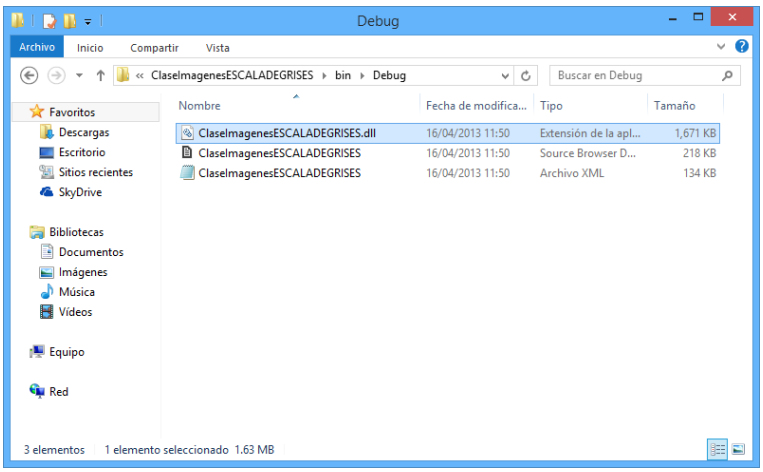

Ilustración 205. Ruta biblioteca de clases.

9.3.3 Eventos y propiedades a tener en cuenta.

Como se ha visto en el desarrollo anterior, hay una serie de eventos y propiedades que hay que tener en cuenta a la hora de desarrollar funciones para ClaseImagenes.

9.3.3.1 Eventos

Hay dos eventos en la clase, uno para indicar que se ha recibido una imagen y ésta se ha transformado y otro para indicar que se ha abierto una imagen nueva y por lo tanto, se ha modificado el nombre, tamaño y origen de la imagen original actual. El código fuente sería:

```
'Se genera el evento que informa de que la imagen se ha transformado
RaiseEvent actualizaBMP(bmpSalida)

'Se genera un evento que informa de que se ha abierto una imagen nueva. Se
utiliza la función nombreImagen (incluida en la clase) para obtener el nombre
de la imagen y enviamos nombre, ancho, alto y origen de la imagen
RaiseEvent actualizaNombreImagen({nombreImagen("C:/Users/imagenPrueba.jpg"),
bmp.Width, bmp.Height, "Desde archivo"})
```

9.3.3.2 Propiedades

También existen un conjunto de propiedades que debe utilizarse a la hora de implementar funciones dentro de la clase.

La variable *porcentaje* (accesible desde fuera de la clase como propiedad denominada *estadoCarga*) sirve para indicar el porcentaje realizado en una transformación y el tipo de transformación realizado. Esta variable en un vector con dos posiciones, en la primera se incluye el porcentaje y en la segunda el tipo de transformación. Esta variable es muy útil a la hora de recorrer los píxeles de una imagen. A continuación se muestra un ejemplo.

```
'Se actualiza el porcentaje de transformación
porcentaje(0) = 0
'Se indica la transformación que se está llevando a cabo
porcentaje(1) = "Transformando imagen a escala de grises"
```

Hay otra propiedad (que realmente no se utiliza directamente, sino mediante el método *guardarImagen*) que almacena una imagen cuando se ha realizado una transformación, es decir, cuando se crea una función y justo antes de devolver la imagen transformada se debe llamar al método de la siguiente forma.

```
'Se almacena la imagen y la transformación efectuada
guardarImagen(bmpSalida, "Imagen transformada a escala de grises")
```

9.4 Prioridades actuales de desarrollo

A continuación se muestra una lista con los siguientes objetivos para la clase ClaseImagenes:

- Utilizar lo mínimo posible la función *nivel* y leer los valores de los píxeles en el mismo bucle *For* en el que se modifican los píxeles. Se puede hacer con función *GetPixel* de las variables *Bitmap*.
- Utilizar el método *Bitmap.LockBits* para guardar directamente en memoria los datos de los píxeles y así hacer transformaciones mucho más rápidas.
- Modificar la clase para que se puedan hacer diferentes instancias y que cada una tenga su lista de imágenes para *deshacer/rehacer* independientes. Esto se puede hacer omitiendo la sentencia *Shared* en las variables asociadas a las propiedades.
- Añadir todas las funciones a la función secuencia.

9.5 Anexo I. Nombres admitidos en la función secuencia

A continuación se muestra una lista con los nombres y parámetros admitidos para la función secuencia.

```
Case "Blanco y negro"
bmpSalida = Me.BlancoNegro(bmpSalida, datosSecuencia(i, 1))

Case "Escala de grises"
bmpSalida = Me.EscalaGrises(bmpSalida, datosSecuencia(i, 1))

Case "Brillo"
bmpSalida = Me.Brillo(bmpSalida, datosSecuencia(i, 1))

       Case "Invertir colores (rojo, verde, azul)"
bmpSalida = Me.Invertir(bmpSalida, Convert.ToBoolean(datosSecuencia(i, 1)),
Convert.ToBoolean(datosSecuencia(i, 2)), Convert.ToBoolean(datosSecuencia(i,
3)))

       Case "Sepia"
       bmpSalida = Me.sepia(bmpSalida)

Case "Filtros básicos (rojo, verde, azul)"
bmpSalida = Me.filtrosBasicos(bmpSalida, Convert.ToBoolean(datosSecuencia(i,
1)), Convert.ToBoolean(datosSecuencia(i, 2)),
Convert.ToBoolean(datosSecuencia(i, 3)))

Case "RGB a (BGR, GRB, RBG)"
bmpSalida = Me.RGBto(bmpSalida, Convert.ToBoolean(datosSecuencia(i, 1)),
Convert.ToBoolean(datosSecuencia(i, 2)), Convert.ToBoolean(datosSecuencia(i,
3)))

Case "Redimensionar"
bmpSalida = Me.Redimensionar(bmpSalida, New Rectangle(0, 0, datosSecuencia(i,
1), datosSecuencia(i, 2)), Drawing2D.InterpolationMode.HighQualityBicubic)

Case "Contraste (recomendado)"
       bmpSalida = Me.Contraste(bmpSalida, datosSecuencia(i, 1))

Case "Contraste"
bmpSalida = Me.ContrasteEstirar(bmpSalida, datosSecuencia(i, 1),
datosSecuencia(i, 2))

Case "Correción de gamma"
bmpSalida = Me.Gamma(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i, 2),
datosSecuencia(i, 3))

Case "Exposición"
```

```
        bmpSalida = Me.Exposicion(bmpSalida, datosSecuencia(i, 1))

Case "Modificar canales"
bmpSalida = Me.Canales(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i, 2),
datosSecuencia(i, 3), datosSecuencia(i, 4))

Case "Reducir colores"
bmpSalida = Me.reducircolores(bmpSalida, 127 - (datosSecuencia(i, 1) / 2))

Case "Filtrar colores (rojo)"
bmpSalida = Me.filtroColoresRango(bmpSalida, datosSecuencia(i, 1),
datosSecuencia(i, 2), datosSecuencia(i, 3))

Case "Filtrar colores (verde)"
bmpSalida = Me.filtroColoresRango(bmpSalida, 0, 0, 0, datosSecuencia(i, 1),
datosSecuencia(i, 2), datosSecuencia(i, 3))

Case "Filtrar colores (azul)"
bmpSalida = Me.filtroColoresRango(bmpSalida, 0, 0, 0, 0, 0, 0,
datosSecuencia(i, 1), datosSecuencia(i, 2), datosSecuencia(i, 3))

Case "Detectar contornos"
bmpSalida = Me.contornos(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i, 2),
datosSecuencia(i, 3), datosSecuencia(i, 4))

Case "Operación aritmética - Suma"
bmpSalida = Me.Suma(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i, 2),
datosSecuencia(i, 3), datosSecuencia(i, 4), False)

Case "Operación aritmética - Resta"
bmpSalida = Me.Resta(bmpSalida, -datosSecuencia(i, 1), -datosSecuencia(i, 2), -
datosSecuencia(i, 3), -datosSecuencia(i, 4), False)

Case "Operación aritmética - División"
bmpSalida = Me.Division(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i, 2),
datosSecuencia(i, 3), datosSecuencia(i, 4), False)

Case "Operación aritmética - Multiplicación"
bmpSalida = Me.Multiplicacion(bmpSalida, datosSecuencia(i, 1),
datosSecuencia(i, 2), datosSecuencia(i, 3), datosSecuencia(i, 4), False)

Case "Operación lógicas - AND"
bmpSalida = Me.OperAND(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i, 2),
datosSecuencia(i, 3), datosSecuencia(i, 4), False)

Case "Operación lógicas - OR"
bmpSalida = Me.OperOR(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i, 2),
datosSecuencia(i, 3), datosSecuencia(i, 4), False)

Case "Operación lógicas - XOR"
bmpSalida = Me.OperXOR(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i, 2),
datosSecuencia(i, 3), datosSecuencia(i, 4), False)

Case "Reflexión horizontal"
        bmpSalida = Me.Reflexion(bmpSalida, True, False)

Case "Reflexión vertical"
bmpSalida = Me.Reflexion(bmpSalida, False, True)

Case "Traslación"
bmpSalida = Me.Traslacion(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i,
2))
```

```
Case "Voltear"
       bmpSalida = Me.Volteados(bmpSalida, Volteado(datosSecuencia(i, 1)))

Case "Density Slicing automático"
       'Calculamos el número de colores
Dim matrizStringColores() As String = {"Red", "Green", "Blue",
"Black", "White", "Yellow", "Maroon", "Gray", "LightGreen", "LightBlue",
"Orange", "Pink", "Gold", "DarkBlue", "DarkGreen"}
Dim matrizColores(datosSecuencia(i, 1) - 1) As Color
       For ii = 0 To CInt(datosSecuencia(i, 1)) - 1
       matrizColores(ii) = Color.FromName(matrizStringColores(ii).ToString)
       Next
If Convert.ToBoolean(datosSecuencia(i, 2)) = True Then 'Si es normalizada
bmpSalida = Me.DensitySlicingNormalizado(bmpSalida, datosSecuencia(i, 1),
matrizColores)
       Else
bmpSalida = Me.DensitySlicing(bmpSalida, datosSecuencia(i, 1), matrizColores)
       End If

Case "Sobel total"
       bmpSalida = Me.sobelTotal(bmpSalida)

Case "Desenfoque - Distorsión"
       bmpSalida = Me.Distorsion(bmpSalida, datosSecuencia(i, 1))

Case "Desenfoque - Movimiento"
bmpSalida = Me.desenfoque(bmpSalida, datosSecuencia(i, 1), datosSecuencia(i,
2))

Case "Desenfoque - Blur"
       Dim objetoMascara As New TratamientoImagenes.Mascaras
       Dim mascara = objetoMascara.LOW9
       bmpSalida = Me.mascara3x3RGB(bmp, mascara, , )

Case "Pixelado"
       bmpSalida = Me.Pixelar(bmp, datosSecuencia(i, 1))

Case "Cuadrícula"
bmpSalida = Me.cuadricula(bmp, Color.FromName(datosSecuencia(i, 2)),
Color.FromName(datosSecuencia(i, 4)), datosSecuencia(i, 1), datosSecuencia(i,
3))

Case "Sombra de vidrio"
bmpSalida = Me.SombraVidrio(bmp, datosSecuencia(i, 1),
Convert.ToBoolean(datosSecuencia(i, 2)))

Case "Trocear imagen - Tres partes"
       bmpSalida = Me.ImagenTresPartes(bmp)

Case "Trocear imagen - Seis partes"
       bmpSalida = Me.ImagenSeisPartes(bmp)

Case "Ruido aleatorio"
       bmpSalida = Me.RuidoAleatorio(bmp, datosSecuencia(i, 1))

Case "Ruido desplazado"
bmpSalida = Me.RuidoProgresivo(bmp, datosSecuencia(i, 1),
Convert.ToBoolean(datosSecuencia(i, 2)))

Case "Óleo"
```

```
bmpSalida = Me.Oleo(bmp, datosSecuencia(i, 1), 127 - (datosSecuencia(i, 2) /
2))

Case "Efecto Marte"
       bmpSalida = Me.EfectoMarte(bmp)

Case "Efecto antiguo sobreexpuesto"
       bmpSalida = Me.EfectoAntigSobreex(bmp)

Case "Efecto marino"
       bmpSalida = Me.EfectoMarino(bmp)

Case "Aumentar rasgos"
       bmpSalida = Me.EfectoAumentarRasgos(bmp)

Case "Disminuir rasgos"
       bmpSalida = Me.EfectoDisminuirRasgos(bmp)

Case "Contorno sombreado - Contenido"
       bmpSalida = Me.EfectoContornoSombreado(bmp)

Case "Contorno sombreado - Desmedido"
       bmpSalida = Me.EfectoContornoSombreado2(bmp)

Case "Aumentar luz"
       bmpSalida = Me.EfectoAumentarLuz(bmp)

End Select
```

10 ANEXO III. BIBLIOGRAFÍA

10.1 Libros

- CHARTE, FRANCISCO (2010). Visual Basic 2010 (Guía práctica). Anaya multimedia.
- GROUSSARD, THIERRY. Visual Basic 2010 (VB.NET). Los fundamentos del lenguaje. ENI
- M. ALCAÑIZ RAYA, V. GRAU COLOMER, MC. JUAN LIZANDRA, C. MONSERRAT ARANDA, JM. NAVARRO SOLER, E. MOLTÓ GARCÍA (1999). Procesamiento Digital de Imagen. Servicio de Publicaciones UPV.
- RODRÍGUEZ MORALES, ROBERTO (2011). Procesamiento y análisis digital de imágenes. RA-MA.

10.2 Programas informáticos

- SPENCER KIMBALL, PETER MATTIS y el equipo de desarrollo de GIMP [en línea]: GNU Image Manipulation Program. Versión 2.8.4. Programa computacional.

10.3 Recursos en línea

- Procesamiento digital de imágenes. (2013, 9 de marzo). Wikipedia, La enciclopedia libre. Fecha de consulta: 10:32, abril 11, 2013 desde http://es.wikipedia.org/w/index.php?title=Procesamiento_digital_de_im%C3%A1genes&oldid=64532776.
- Apolo. (2013, 29 de abril). Wikipedia, La enciclopedia libre. Fecha de consulta: 12:33, marzo 3, 2013 desde http://es.wikipedia.org/w/index.php?title=Apolo&oldid=66543755.
- Imagen de mapa de bits. (2013, 11 de marzo). Wikipedia, La enciclopedia libre. Fecha de consulta: 10:35, marzo 29, 2013 desde http://es.wikipedia.org/w/index.php?title=Imagen_de_mapa_de_bits&oldid=64807093.
- Graphics Interchange Format. (2013, 12 de abril). Wikipedia, La enciclopedia libre. Fecha de consulta: 10:40, marzo 29, 2013 desde http://es.wikipedia.org/w/index.php?title=Graphics_Interchange_Format&oldid=66186068.
- Joint Photographic Experts Group. (2013, 9 de abril). Wikipedia, La enciclopedia libre. Fecha de consulta: 10:40, marzo 29, 2013 desde http://es.wikipedia.org/w/index.php?title=Joint_Photographic_Experts_Group&oldid=66094758.
- Portable Network Graphics. (2013, 15 de marzo). Wikipedia, La enciclopedia libre. Fecha de consulta: 11:03, marzo 29, 2013 desde http://es.wikipedia.org/w/index.php?title=Portable_Network_Graphics&oldid=65234653.

- TIFF. (2013, 8 de marzo). Wikipedia, La enciclopedia libre. Fecha de consulta: 11:08, marzo 29, 2013 desde http://es.wikipedia.org/w/index.php?title=TIFF&oldid=64503190.
- Efecto de ojos rojos. (2013, 27 de marzo). Wikipedia, La enciclopedia libre. Fecha de consulta: 12:22, marzo 30, 2013 desde http://es.wikipedia.org/w/index.php?title=Efecto_de_ojos_rojos&oldid=65681771.
- Transformación afín. (2013, 9 de marzo). Wikipedia, La enciclopedia libre. Fecha de consulta: 10:04, abril 3, 2013 desde http://es.wikipedia.org/w/index.php?title=Transformaci%C3%B3n_af%C3%ADn&oldid=64632988.
- Anaglifo. (2013, 8 de marzo). Wikipedia, La enciclopedia libre. Fecha de consulta: 10:30, abril 3, 2013 desde http://es.wikipedia.org/w/index.php?title=Anaglifo&oldid=64518758.
- Entender la correción de gamma. (2008, 14 de septiembre). Efecto HD. Fecha de consulta: 15:30, abril 5, 2013 desde http://www.efectohd.com/2008/09/entender-la-correcion-de-gamma.html.
- RotateFlipType (Enumeración). (2013). MSDN. Fecha de consulta: 15:39, abril 5, 2013 desde http://msdn.microsoft.com/es-es/library/system.drawing.rotatefliptype.aspx.
- Pseudocolor y Falso color. (Sin fecha). Universidad de Valencia. Fecha de consulta: 10:03, abril 5, 2013 desde http://www.uv.es/gpoei/eng/Pfc_web/generalidades/pseudocolor/pseudocolor.htm.
- Matriz de convolución. (Sin fecha). Documentación GIMP. Fecha de consulta: 11:44, marzo 28, 2013 desde http://docs.gimp.org/es/plug-in-convmatrix.html.
- Filtros de realce. (1995, 1 de abril). Escuela Técnica Superior de Ingenieros de Minas, Universidad de Oviedo. Fecha de consulta: 11:55, marzo 28, 2013 desde http://www6.uniovi.es/vision/intro/node43.html.
- Operaciones especiales. (1999). Departamente de teoría de la señal y comuniaciones, Universidad Carlos III de Madrid. Fecha de consulta: 10:55, marzo 28, 2013 desde http://www.tsc.uc3m.es/imagine/Curso_ProcesadoBasico/Contenido/OperacionesEspaciales/OperacionesEspaciales.html#pasoAlto.
- Imagen digital: conceptos básicos. (Sin fecha). INTEF. Fecha de consulta: 16:43, abril 10, 2013 desde http://platea.pntic.mec.es/~lgonzale/tic/imagen/conceptos.html.
- Programa de manipulación de imágenes de GNU. (2012). GNU Image Manipulation Program. Fecha de consulta: 10:02, abril 15, 2013 desde http://docs.gimp.org/2.8/es/.
- Foro de lenguaje VB.NET. (2010, 24 de septiembre). MSDN. Fecha de consulta: 12:02, abril 1, 2013 desde http://social.msdn.microsoft.com/Forums/es-es/vbes/threads.
- Crear anaglifos en Visual Basic .NET. Tratamiento de imágenes. Parte X. (2013, 5 de marzo). Procesamiento digital de imágenes (y más) en VB .NET. Fecha de consulta: 10:53, abril 2, 2013 desde

http://algoimagen.blogspot.com.es/2013/03/crear-anaglifos-en-visual-basic-net.html.

11 ANEXO IV. LICENCIA DEL DOCUMENTO

Este documento ha sido liberado bajo Licencia GFDL 1.3 (GNU Free Documentation License). Se incluyen los términos de la licencia en inglés al original del mismo.