Universidad Central "Marta Abreu" de Las Villas

Facultad de Ingeniería Eléctrica

Trabajo de Diploma

MImg: *Toolbox* para el uso desde MATLAB de paquetes de PDI implementado en C/C++

Autor: Adrian Herrera Hernández.

Tutor: MSc. Roberto Díaz Amador.

Santa Clara

2015

"Año 57 de la Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Trabajo de Diploma

MImg: *Toolbox* para el uso desde MATLAB de paquetes de PDI implementado en C/C++.

Autor: Adrian Herrera Hernández.

ahhernandez@uclv.edu.cu

Tutor: MSc, Roberto Díaz Amador.

Profesor Auxiliar, CEETI, FIE

rdamador@uclv.edu.cu

Hago constar que el presente trabajo de diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Informática, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

_____

Firma del Autor

Los abajo firmantes certificamos que el presente trabajo ha sido realizado según acuerdo de la dirección de nuestro centro y el mismo cumple con los requisitos que debe tener un trabajo de esta envergadura referido a la temática señalada.

_____ _____

Firma del Tutor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

"Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica: la voluntad".

Albert Einstein.

DEDICATORIA

Le dedico este trabajo a toda mi familia por ser mi primera escuela y ser los forjadores de mi persona.

AGRADECIMIENTOS

A mis padres Nancy y Oriol por su amor, cariño y apoyo durante mi vida.
A mi hermano Ariel por su apoyo incondicional.
A mi tutor Roberto por su inmenso apoyo en la realización del presente trabajo.
A mi familia por trasmitirme todo su cariño.
A mis amigos del grupo y de la vida. A todos infinitas gracias.

TAREA TÉCNICA

1. Realizar una revisión bibliográfica sobre los paquetes de Procesamiento de Imágenes implementados en C/C++.
2. Implementar funciones en C/C++ para que puedan ser usados desde MATLAB.
3. Implementar una *toolbox* de funciones en MATLAB para usar las implementadas en C/C++.
4. Realizar una interfaz que ilustre el uso de la *toolbox* implementada.
5. Crear manual de usuario de la *toolbox*.

Firma del Autor

Firma del Tutor

.

RESUMEN

Las bibliotecas para el PDI implementadas eficientemente en C/C++ poseen un gran número de funciones con un excelente desempeño. De aquí la importancia de desarrollar una *toolbox* para la utilización de estas funciones desde MATLAB. Para la realización de este trabajo, de las bibliotecas disponibles se utiliza CImg. Esta biblioteca presenta las características deseables debido a su portabilidad y su fácil utilización desde MATLAB. Se realiza una descripción de las herramientas disponibles en MATLAB para la utilización de la biblioteca CImg, en particular del uso de funciones MEX. Además se realiza una descripción de la estructura de las funciones implementadas en la *toolbox* MImg. Se demuestra el uso de la *toolbox* mediante la realización de un conjunto de aplicaciones de PDI. Como resultado se obtiene interfaz que implementa un conjunto de funciones de la *toolbox* MImg de fácil uso, así con ejemplos del funcionamiento de las misma y su correspondiente manual de usuario.

ABSTRACT

PDI libraries for efficiently implemented in C / C ++ have a large number of functions with an excellent performance. Hence the importance of developing a toolbox for the use of these functions from MATLAB. To perform this work, CImg available libraries was used. This library provides the desirable characteristics due to its portability and ease to use from MATLAB. In this work, we provide a description of the tools available in MATLAB for the use of the CImg library, in particular the use of MEX functions. The final result in this work is a MATLAB toolbox, named MImg, in order to provide an easy interface between MATLAB users and CImg library. The use of this toolbox was demonstrated by performing a set of PDI applications. As result we provide a MATLAB guide interface that implements a set of functions of MImg toolbox. An User Manual and some examples are included too.

TABLA DE CONTENIDOS

INTRODUCCIÓN

A lo largo de las últimas décadas, el campo del procesamiento de imágenes ha sido objeto de investigación debido al incremento de dispositivos de captura de imágenes, su digitalización y la automatización de tareas que antes sólo podía realizar el ser humano. Gracias a esto se han desarrollado muchos métodos y algoritmos para el tratamiento y procesamiento de imágenes. La evolución de las computadoras también ha contribuido en el desarrollo de estos algoritmos y métodos, ya que la computación de las imágenes siempre ha sido un proceso muy costoso, y la comercialización de computadoras más potentes ha hecho posible el procesamiento de imágenes en tiempo razonable.

El entorno MATLAB se ha convertido en una herramienta muy generalizada para el cálculo científico gracias a su lenguaje de alto nivel basado en matrices que permite resolver problemas de cálculo con más rapidez y facilidad que los lenguajes de propósito general como el C/C++. De hecho, son muchas las empresas y los investigadores que usan MATLAB como entorno de trabajo. *The MathWorks*, la empresa desarrolladora de MATLAB, desde sus inicios ofrece un *Toolbox* de algoritmos para el trabajo con imágenes. No obstante, el software para el trabajo con estos algoritmos que se pueden encontrar en C/C++ posee funciones que no están implementadas en MATLAB. El software de MATLAB es relativamente nuevo y poco probado mientras que las bibliotecas existentes en C/C++ se llevan utilizando años y han sido objeto de numerosas revisiones, por lo que poseen un mayor grado de madurez. Por esto, parece razonable plantear la opción de utilizar todo ese software ya existente dentro de la plataforma MATLAB. Las interfaces que MATLAB incorpora hacia C/C++ permiten realizar esta conexión de una forma directa.

A partir de lo expuesto anteriormente el presente trabajo consta de los objetivos que se enuncian a continuación:

Objetivo General:

1. Desarrollar una *toolbox* que permita la utilización desde MATLAB de herramientas implementadas en C/C++.

Objetivos Específicos:

1. Realizar un estudio sobre los distintos paquetes de Visión por Computadoras implementados en otros lenguajes, principalmente en C++.
2. Desarrollar distintas funciones y/u objetos que permitan el uso desde MATLAB de estas herramientas.
3. Elaborar manual de usuario y/o ayuda de la *toolbox*.
4. Utilizar la *toolbox* desarrollada en un problema real.

El informe de este trabajo consta de tres capítulos, conclusiones, recomendaciones, bibliografía y anexos. En el primer capítulo aborda sobre el uso desde MATLAB de bibliotecas implementadas en C/C++. En el segundo capítulo se habla sobre el diseño de la *toolbox* MImg de MATLAB para el procesamiento de imágenes. En el tercer capítulo se realiza una demostración del uso de la *toolbox* MImg. Por último se dan las conclusiones y recomendaciones del trabajo, y los anexos.

CAPÍTULO 1. USO DESDE MATLAB DE BIBLIOTECAS IMPLEMENTADAS EN C/C++

1.1 Bibliotecas en C/C++ para el trabajo con imágenes.

Con la integración de bibliotecas externas dentro de MATLAB no solo se tiene acceso a una implementación más fiable de los algoritmos para el procesamiento de imágenes sino que se puede hacer uso de características avanzadas de estos que los actuales *toolbox* no incorporan.

Debido a que hay muchos objetivos comunes en el procesamiento de imágenes, existen unas bibliotecas que tienen implementados muchas de las operaciones más usadas. Las bibliotecas de procesamiento de imágenes facilitan la labor del programador que desee realizar una aplicación que requiera procesamiento de imágenes. Estas bibliotecas pueden ser especializadas incluyendo algunos algoritmos específicos y/o operaciones más frecuentes.

1.1.1 Biblioteca CImg

CImg[1] (Figura 1) en una biblioteca gratis y de fácil uso. Contiene una gran cantidad de funciones a la hora de trabajar con imágenes[1]. Su principal beneficio es su elevado grado de portabilidad. Las funciones de la biblioteca CImg aparecen implementadas en un único archivo: cimg.h[2].

[1] http://cimg.sourceforge.net

Figura 1.1 Logo de la biblioteca CImg.

El archivo cabecera cimg.h contiene todas las clases y funciones que componen la propia biblioteca[3]. Esta es una originalidad de la Biblioteca CImg. Esto significa en particular que:

- No se necesita pre-compilación de la biblioteca, ya que la recopilación de las funciones CImg se realiza al mismo tiempo que la compilación de su propio código C++.
- No hay dependencias complejas: Sólo se incluye el archivo CImg.h, y se obtienen un conjunto de herramientas para el procesamiento de imágenes en C++.
- Los miembros del grupo y las funciones están entre líneas, lo que lleva a un mejor rendimiento durante la ejecución del programa.
- La compilación se realiza sobre la marcha: solo las funciones CImg realmente utilizados por el programa son compiladas y aparecen en el programa ejecutable compilado. Esto conduce a un código muy compacto, sin ningún tipo de líneas no utilizadas.

Estructura de la biblioteca

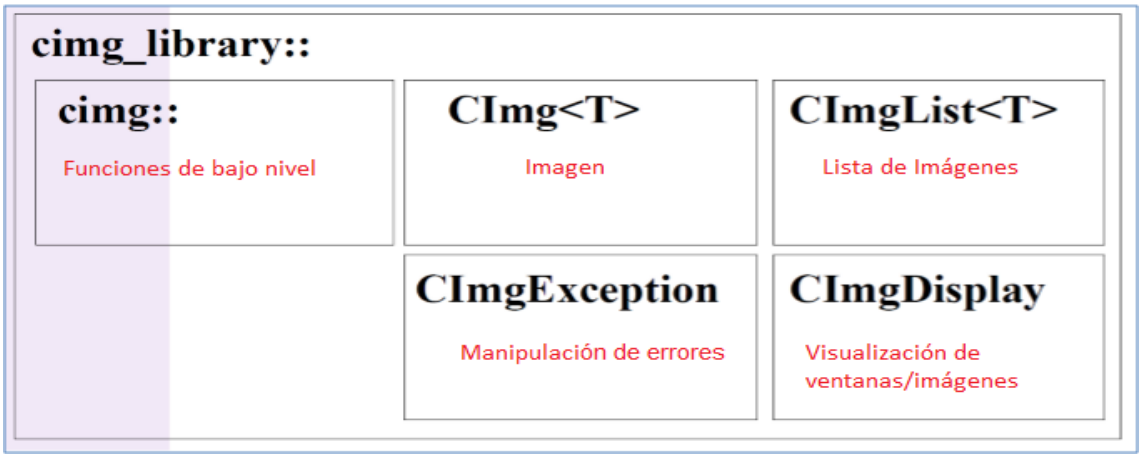


Figura 1.2 Clases implementadas en la biblioteca CImg.

Todas las clases de la biblioteca y las funciones se definen en el espacio de nombres cimg_library[4]. Generalmente, se utiliza este espacio de nombres como el espacio de nombres por defecto:

```
#include "CImg.h"

using namespace cimg_library;
```

Las clases implementadas en esta biblioteca se muestran en la figura 2:

- *cimg_library:: cimg* define un conjunto de funciones de bajo nivel y las variables utilizadas por la biblioteca. Funciones documentadas en este espacio de nombres se pueden utilizar de forma segura en su propio programa. No es conveniente utilizar la clase *cimg_library* :: cimg como un espacio de nombres por defecto, ya que contiene las funciones cuyos nombres ya están definidos en la norma de bibliotecas propias de C/C++. La clase *cimg_library* :: CImg representa imágenes de hasta 4-dimensiones de ancho, que contiene pixeles de tipo T (parámetro de plantilla). Esto es en realidad la clase principal de la biblioteca.
- *cimg_library :: CImgList* representa listas de *cimg_library :: images CImg* <T>. Se puede utilizar por ejemplo para almacenar diferentes marcos de una secuencia de imágenes.
- *cimg_library* :: *CImgDisplay* es capaz de mostrar imágenes o listas de imágenes en las ventanas gráficas de presentación. Como se puede adivinar, el código de esta clase es altamente dependiente del sistema, pero esto es transparente para el programador, como variables de entorno se establecen automáticamente por la biblioteca CImg.
- *cimg_library* :: *CImgException* (y sus subclases) son usadas por la biblioteca para producir excepciones cuando se producen errores. Estas excepciones pueden ser atrapadas con un bloque *try catch* (*CImgException).* Las subclases definen con precisión el tipo de errores encontrados.

Conociendo estas cuatro clases es suficiente para obtener beneficios de las funcionalidades de la biblioteca Cimg.

1.1.2 Biblioteca ORFEO

OTB (ORFEO *Toolbox*)[2] está implementado en C++ y se basa principalmente en ITK (*Insight Toolkit*). ITK es utilizada como elemento principal de OTB y por esa razón, muchas las clases de OTB heredan sus funcionalidades de algunas de las clases de ITK[5]. Para establecer una continuidad en el aprendizaje, la documentación de OTB, sigue las mismas líneas de organización y diseño que la documentación de ITK, de tal manera que, para el usuario, la navegación por los distintos métodos y clases de las dos bibliotecas se hace mucho más sencilla.

OTB fue creado desde su inicio de forma colaborativa. La enseñanza, la investigación y los usos comerciales, de este conjunto de herramientas son algunos de los objetivos previstos que se tenían para esta biblioteca. Los desarrolladores proponen, que de usarse, se colabore en su mejora mediante el reporte de errores, la contribución con nuevas clases y su difusión mediante cursos para llegar al mayor número posible de colaboradores.

La biblioteca ORFEO *Toolbox* provee de un conjunto de recursos (como los ofrecidos por software comerciales) para el tratamiento de imágenes hiperespectrales bastante útiles[6], además de las ventajas que supone ser software libre en la investigación y el estudio, así como, en el desarrollo de nuevas funcionalidades. Además provee de estas funcionalidades en código C por lo que su eficiencia, en comparación a otras *toolbox*, es mucho mayor[7].

Algoritmos implementados

[2] http://www.orfeo-toolbox.org/

- El algoritmo OSP fue inicialmente desarrollado para encontrar firmas espectrales utilizando el concepto de proyecciones ortogonales. El algoritmo hace uso de un operador de proyección ortogonal. El algoritmo utiliza el operador ortogonal de forma repetitiva hasta encontrar un conjunto de pixeles ortogonales a partir de un píxel inicial.
- El algoritmo N-FINDR utiliza una técnica basada en identificar los *endmembers* como los vértices del *simplex* de mayor volumen que puede formarse en el conjunto de puntos. N-FINDR no trabaja con todo el cubo de datos sino con una simplificación del mismo a tantas bandas como *endmembers* se deseen encontrar[8]. Para este tipo de reducciones se suele utilizar la técnica PCA (*Principal Component Analysis*) o MNF (*Minimum Noise Fraction*). El único parámetro que tiene este algoritmo es el número de *endmembers* a identificar.
- El algoritmo SSEE es un método representativo de las aproximaciones que consideran la información espacial y espectral de forma separada, no simultánea.
- El algoritmo AMEE es un método representativo de las aproximaciones que consideran la información espacial y espectral de forma simultánea. Este método de operaciones utiliza operaciones morfológicas extendidas de erosión y dilatación.
- El algoritmo, *Vertex Component Analysis,* sirve, al igual que los anteriores, para la separación en mezclas espectrales de *endmembers*[9]. Este algoritmo se basa en dos ideas principales:

  1. Los *endmembers* son los vértices de la envoltura convexa de un conjunto de (n+1) puntos independientes afines a un espacio euclídeo de dimensión o mayor, o simplex.
  2. La transformación afín de un simplex es también un simplex.

  VCA, al igual que el algoritmo N-FINDR, asume que existe pixeles puros en la imagen. El algoritmo iterativamente proyecta la información en una dirección ortogonal al subespacio abarcado por los *endmembers* calculados hasta ese momento, siendo el nuevo *endmember* el extremo de esa proyección. El algoritmo itera hasta que todos los *endmembers* han sido extraídos.

- *Iterative Error Analysis*. Una de las técnicas para el procesamiento de imágenes hiperespectrales es el remezclado espectral, el cual sirve en las situaciones en las que los distintos componentes de las firmas espectrales ocupan, de forma conjunta, un solo píxel debido a insuficiente precisión de los sensores que captan las imágenes. Selecciona *endmembers* que se corresponden en todo momento a pixeles pertenecientes al conjunto de datos original, no siendo posible generar *endmembers* artificiales.

1.1.3 Biblioteca OpenCV

OpenCV [3] (*Open source Computer Vision library*) es una biblioteca de código abierto desarrollado por Intel. Esta biblioteca proporciona un alto nivel funciones para el procesado de imágenes[10]. Estas bibliotecas permiten a los programadores para crear aplicaciones poderosas en el dominio de la visión digital. OpenCV ofrece muchos tipos de datos de alto-nivel como juegos, árboles, gráficos, matrices, etc. OpenCV es de código abierto para poder funcionar en muchas plataformas.

OpenCV implementa una gran variedad de herramientas para la interpretación de la imagen. Es compatible con *Intel Image Processing Library* (IPL) que implementa algunas operaciones en imágenes digitales como binarización, filtrado, estadísticas de la imagen, pirámides, etc.[11]. Además, esta biblioteca implementa técnicas de calibración (Calibración de la Cámara), detección de rasgos, para rastrear (Flujo Óptico), análisis de la forma (Geometría, Contorno que Procesa), análisis del movimiento (Plantillas del Movimiento, Estimadores), reconstrucción 3D (Transformación de vistas), segmentación de objetos y reconocimiento (Histograma, etc.).

El rasgo esencial de la biblioteca junto con funcionalidad y la calidad es su desempeño. Los algoritmos están basados en estructuras de datos muy flexibles, acoplados con estructuras IPL; más de la mitad de las funciones han sido optimizadas aprovechándose de la Arquitectura de Intel[12].

[3] http://opencv.org/

La estructura *Iplimage* se usa para crear y manejar imágenes. Esta estructura tiene gran cantidad de campos, algunos de ellos son más importantes que otros. Por ejemplo el *width* es la anchura del *Iplimage*, *height* es la altura, *depth* es la profundidad en bits y *nChannels* el número de canales (uno por cada nivel de gris de las imágenes y tres para las imágenes coloridas)[13]. En cuanto a análisis de movimiento y seguimiento de objetos, ofrece una funcionalidad interesante. Incorpora funciones básicas para modelar el fondo para su posterior sustracción, generar imágenes de movimiento MHI (*Motion History Images*) para determinar dónde hubo movimiento y en qué dirección, algoritmos de flujo óptico, etc. La biblioteca contiene una interface gráfica llamada *highGUI*. Esta interfaz gráfica es muy importante porque se necesita bajo OpenCV para visualizar imágenes[14].

Tipos de datos

Hay varios tipos de datos fundamentales y varios tipos de datos auxiliares para hacer OpenCV más simple y uniforme.

Los tipos de los datos fundamentales incluyen tipos de vectores como: *IplImage* (imagen de IPL), *CvMat* (matriz) y tipos mixtos: *CvHistogram* (histograma multi-dimensional)[15].

Los tipos de datos auxiliares incluyen: *CvPoint* (2d punto), *CvSize* (anchura y altura), *CvTermCriteria* (criterio de la terminación para los procesos iterativos), *IplConvKernel* (núcleo de la circunvolución), *CvMoments* (momentos espaciales), etc[16].

Convenciones que se utilizan

El software de OpenCV tiene las siguientes convenciones:

- Los identificadores constantes están en mayúsculas; por ejemplo, CV_SEQ_KIND_GRAPH.
- Todos los nombres de las funciones usadas para el procesado de imagen tienen el prefijo del cv, por ejemplo: "*cvCreateImage*", "*cvSobel*", "*cvAdd*".

- Todas las funciones externas de OpenCV empiezan con el prefijo cv, y todas las estructuras con prefijo Cv.

Cada nueva parte de una función empieza con un carácter en mayúscula, por ejemplo: *cvContourTree.* Los nombres de funciones en OpenCV tienen el siguiente formato: cv [*action*] [t*arget*] [*mod*] ()[17] donde:

action: la acción indica la funcionalidad del centro, por ejemplo, -*Set*-, -*Create*-, -*Convert*-

target: indica el área donde el procesado de la imagen está siendo establecido, por ejemplo : -*Find Contours, -ApproxPoly.* En la mayoría de los casos, target consiste en dos o más palabras, por ejemplo: *MatchContourTree*. Algunos nombres de funciones consisten en una acción u objetivo solamente, por ejemplo: las funciones, *cvUnDistort* o *cvAcc* respectivamente[18].

mod: es un campo opcional; indica una modificación de la funcionalidad de la función. Por ejemplo: en la función cvFindExtrinsicCameraParams_64d, _64d, indica que es una función constante particular de 64d valores.

1.1.4 La API EmguCV

EmguCV es una biblioteca de C# que envuelve a la biblioteca OpenCV de C++. Es decir, es una forma de utilizar todas las funciones que nos proporciona OpenCV desde un programa escrito en C#. Hay bibliotecas que implementan en C# las funciones de OpenCV, pero este lenguaje es algo más lento a la hora de trabajar con matrices, por lo que se recomienda implementar las funciones en C++ (con una mayor cantidad de compiladores que optimizan el código) y después invocar a esas funciones desde C#. Esto es lo que realiza EmguCV[19]. Así, por ejemplo, para abrir una imagen se puede utilizar la función *CvInvoke.cvShowImage*.

Debido a que trabajar mediante *invokes* puede resultar algo incómodo, y

puede ensuciar el código, resultando más difícil las labores de desarrollo y mantenimiento de aplicaciones, EmguCV proporciona algunas clases que engloban a algunas funciones de OpenCV. Así, por ejemplo, se tiene la clase *Image*. Esta clase contiene muchas funciones escritas a través de *invokes*, pero eso se realiza de manera transparente a nosotros. Así, por ejemplo, para realizar un objeto en una imagen, utilizaremos la función image.Draw y esta función invoca a la función de OpenCV cvRectangle de la siguiente manera *CvInvoke.cvRectangle*. Así, nosotros dispondremos de una capa que nos aísla de la comunicación con OpenCV, realizándose todas las llamadas de manera transparente para el desarrollador. El tratamiento de errores en EmguCV es sencillo[20]. Cuando la función invocada de OpenCV devuelve un error, se lanza una excepción *CvException*. De este modo, se aísla del tratamiento de errores de OpenCV y es posible realizar un tratamiento de errores típico de C# basado en excepciones. A diferencia de otras biblioteca envolventes de OpenCV como *OpenCVDotNet*, *SharperCV* o Code Project, que utilizan código no seguro, EmguCV está escrito enteramente en C#. El beneficio es que puede ser compilado en mono y por lo tanto puede correr en cualquier plataforma Mono, inclusive Linux, Solaris y Mac OS X [9].

1.1.5 Biblioteca Mamba

Mamba[4] es una biblioteca de morfología matemática, su nombre en realidad es sinónimo de Morfología Matemática, (En aras de la simplicidad, la biblioteca se conoce como Mamba). Ofrece un amplio conjunto de funciones necesarias para realizar operaciones comunes que se utilizan en la morfología matemática, como la erosión, la dilatación, etc. pero también las más complejas (operadores geodésicas, transformaciones de cuencas, etc.)[21].

Mamba está destinada a ser una biblioteca rápida y fácil para la codificación de algoritmos de morfología matemática. En cuanto a código uno de sus objetivos es ser tan portátil como

[4] http://mamba-image.org

sea posible. Esto significa que si se necesita transferir su algoritmo para un sistema específico, se puede adaptar fácilmente el código C para dicho sistema[22].

Trabajo implícito de la biblioteca

Como consecuencia de las reglas de codificación aplicadas a Mamba, parámetros de funciones para el trabajo con imágenes siempre comienzan con los argumentos de entrada y terminan con los argumentos de salida (Esta es una regla general, y pueden haber algunas excepciones para determinadas funciones)[23]. Esto es cierto para imágenes binarias, en escala de grises y de 32 bits. La mayor parte del tiempo, imágenes argumentos son *imIn* cuando la imagen es una entrada y *imOut* cuando es la salida de la función. Cuando las funciones requieren argumentos distintos argumentos para imágenes (valores escalares, borde o configuración de la cuadrícula, etc.), se colocan después de los argumentos de imagen. Todos los pixeles de la imagen se almacenan con enteros sin signo[24].

Arquitectura y el diseño de la biblioteca

Mamba se basa en una arquitectura muy simple. El código C implementa funciones muy básicas que están codificadas para ser específicas, rápidas y simples. En cada función se realiza solamente una operación y se realiza de la mejor manera posible. El código C se compila para crear la biblioteca núcleo que es una colección de operaciones simples. SWIG (Generador de interfaz de envoltura simple) se utiliza para crear la interfaz entre el código C y el código Python[25]. Esta herramienta hace que sea más fácil agregar nuevas funciones dentro de la parte de C. El código Python se divide en varios módulos y paquetes. Este módulo implementa la clase *imageMb* que es la estructura de datos central de la biblioteca. Esta clase es una extensión, más amigable de Python, versión de la estructura de datos utilizada para representar los datos de imágenes dentro de la biblioteca central C. El módulo Mamba también envuelve las funciones básicas para simplificarlos y hacerlos compatibles con la clase *imageMb*.

Debido a que la biblioteca central C no ofrece funciones para leer archivos de imagen, Mamba se basa en el *Python Imaging Library* (PIL) para hacerlo. El módulo *mambaUtils* es

una interfaz para PIL que asegura imágenes correctamente cargadas y convertidas a un formato que es compatible con la representación interna de datos Mamba[26]. La capacidad de visualización se realiza mediante. Este módulo también utiliza funciones *MambaUtils* para el manejo de la imagen (como guardar o cargar), realizado directamente mediante un menú contextual. El módulo *MambaExtra* se construye en la parte superior del módulo Mamba. Implementa funciones o clases que son, como su nombre lo dice, extras a la biblioteca. Este módulo se basa en PIL y Mamba debido a la variedad de "extra". El módulo *MambaDraw* implementa todas las funciones de dibujo que vienen con la biblioteca[27]. Se basa únicamente en funciones y clases que se encuentran en el módulo Mamba. Lo mismo ocurre con el paquete *MambaComposed* que es donde todas las funciones básicas que se encuentran en el *MambaDulemo* se combinan para crear operadores matemáticos de morfología más elaborados. Este paquete se divide en múltiples *Dulemos* para facilitar la lectura (archivos más pequeños que agrupan los operadores por familia).

1.1.6 Biblioteca ITK

ITK[5] (*Insigth Segmentation and Registration Toolkit*), es una biblioteca de código abierto de segmentación y registro de imágenes escrita en C++[28]. No implementa una interfaz gráfica o de visualización, tarea que es llevada a cabo por otras herramientas, como VTK[29]. Igualmente, esta herramienta provee de una mínima funcionalidad para el manejo de archivos. Este sistema incluye importantes algoritmos de registro y segmentación en dos, tres y más dimensiones. También tiene soporte para procesamiento paralelo y multihilo. ITK está basado en una arquitectura de flujo de datos. Esto significa que hay objetos de datos que son procesados por objetos de procesamiento (filtros)[30].

Implementación

[5] http://www.itk.org

La implementación en C++ de las ITK se ha realizado lo más genéricamente posible usando clases templatizadas. Este uso de clases templatizadas permite que el código sea muy eficiente y que la mayor parte de los errores sean descubiertos en tiempo de compilación en vez de en tiempo de ejecución. ITK utiliza un modelo de programación conocido como programación extrema[31]. La programación extrema convierte el proceso de programación usual en un proceso iterativo y simultáneo de diseño, implementación, testeo y lanzamiento. Los puntos clave de esta programación extrema deben ser la comunicación y el testeo. La comunicación entre los distintos miembros de la comunidad ITK permite la rápida evolución del software, mientras que el testeo mantiene el software estable[32].

ITK es implementado en C ++. Es multiplataforma, utilizando un entorno de compilación conocida como *CMake* para gestionar el proceso de compilación de una manera independiente de la plataforma. Además, un proceso de envoltura automatizada (por cable) genera interfaces entre C ++ y los lenguajes de programación interpretado como Tcl, Java y Python[33]. Esto permite desarrollar software usando una variedad de lenguajes de programación. C ++ estilo de aplicación de ITK se refiere a la programación como genérico, es decir, que utiliza plantillas de modo que el mismo código se puede aplicar genéricamente a cualquier clase o tipo que ocurre para apoyar las operaciones utilizadas. Tal plantillas C ++ logran que el código sea muy eficiente, y que muchos problemas de software se descubran en tiempo de compilación, en vez de en tiempo de ejecución del programa[34]. ITK utiliza un modelo de desarrollo de software que se refiere a la programación como "extrema". La programación extrema colapsa la metodología habitual de creación de software en un proceso simultáneo. Las características principales de la programación extrema son la comunicación y las pruebas. La comunicación entre los miembros de la comunidad ITK es lo que ayuda a gestionar la rápida evolución del software. La prueba es lo que mantiene la estabilidad del software. En ITK, un proceso de prueba extensa, es el que mide la calidad sobre una base diaria, lo que refleja la calidad del software en cualquier momento[35].

1.1.7 La API Java2D

La API Java2D amplía muchas de las capacidades graficas de la biblioteca AWT (*Abstract Window Toolkit* - Herramientas Abstractas de Ventanas), permitiendo la creación de mejores interfaces de usuario y de aplicaciones Java mucho más impactantes visualmente. El rango que abarcan todas estas mejoras es muy amplio, ya que comprende el renderizado, la definición de figuras geométricas, el uso de fuentes de letras, la manipulación de imágenes y el enriquecimiento en la definición del color[36]. También permite la creación de bibliotecas personalizadas de gráficos avanzados o de efectos especiales de imagen e incluso puede ser usada para el desarrollo de animaciones u otras presentaciones multimedia al combinarla con otras APIs de Java, como puedan ser JMF (*Java Media Framework* - Entorno de Trabajo de Java para Medios Audiovisuales) o Java 3D.

Renderizado de Imágenes 2D

El proceso de renderizado es un proceso que define un método de cálculo más o menos complejo desarrollado por un ordenador destinado a producir una imagen o secuencia de imágenes. En otros contextos, se habla de renderizado para definir el proceso por el cual se pretende dibujar, en un lienzo de dos dimensiones, un entorno en tres dimensiones formado por estructuras poligonales tridimensionales, luces, texturas y materiales, simulando ambientes y estructuras físicas verosímiles. También se habla de renderización en el contexto de procesos 2D que requieren cálculos complejos como la edición de video, la animación o el desarrollo de efectos visuales[20]. En la API Java2D el proceso de renderizado se hace a través de la clase java.awt.Graphics2D, que es una clase que extiende a *java.awt.Graphics* proporcionándole un control más potente sobre la presentación de texto, imágenes o figuras geométricas. Un objeto *Graphics* (que es una clase abstracta) representa el lienzo abstracto y el contexto en el que puede dibujarse cualquier cosa; es te lienzo puede estar enlazado con un área física de un monitor, o representar una imagen en memoria que sólo se desea manipular y no tiene representación directa durante este proceso.

Tratamiento de Imágenes con Java2D

Una imagen es, básicamente, un *array* bidimensional de puntos de colores a cada uno de los cuales se le llama píxel. Aunque la superficie sobre la que se dibuja la imagen también está compuesta por pixeles, cada píxel de la imagen no tiene por qué corresponderse directamente con un píxel de la superficie de dibujo, ya que puede existir un cambio de escala con el que la imagen original se dibuje en pantalla más grande o más pequeña de lo que es realmente; por tanto son conceptos distintos. Una imagen tiene un ancho y un largo expresados en pixeles y, por el motivo anterior, un sistema de coordenadas independiente de cualquier superficie de dibujo[20]. En los siguientes apartado veremos las posibilidades básicas que ofrece Java2D para trabajar con imágenes, incluyendo cambios de brillo, traslaciones, giros, tratamiento de las componentes de color, difuminaciones, suavizados, filtros, etc.

1.1.8 La API Aforge.NET

Aforge.NET es un *framework* para C# diseñado para desarrolladores e investigadores en los campos de visión artificial, inteligencia artificial, procesamiento digital de imágenes, redes neuronales, algoritmos genéticos y máquinas de aprendizaje, además de otras características que se van a agregando al proyecto, debido a que es un proyecto de código abierto[37]. El trabajo para la mejora del núcleo está en constante progreso, lo que significa que nuevas características y paquetes son agregados constantemente. El núcleo no sólo contiene bibliotecas diferentes y sus fuentes, además contiene muchas aplicaciones de ejemplo, que demuestra la utilidad de este *framework*, además de su correspondiente documentación.

Estructura de Aforge.NET

Aforge.Net está compuesto por varios paquetes, cada uno con una funcionalidad diferenciada. Algunos de los más importantes son: *Aforge*: Es el núcleo de *framework* Aforge.NET, contiene las clases que serán utilizadas por el resto de paquetes. *AForge.Controls*: Contiene distintos controles de entrada y salida. *AForge.Genetic*: Contiene las distintas interfaces y clases de computación

genética. *Aforge.Imaging*: Contiene las distintas interfaces y clases para las distintas rutinas de procesamiento de imágenes. Tiene varios subpaquetes, como *AForge.Imaging.Filters* para las clases de los filtros, o *AForge.Imaging.Textures* para los distintos tipos de texturas. *AForge.MachineLearning*: Contiene las distintas interfaces y clases para implementar los diferentes algoritmos de máquinas de aprendizaje *AForge.Math*: Contiene las distintas interfaces y clases con utilidades geométricas. *AForge.Neuro*: Contiene las distintas interfaces y clases que implementan las redes neuronales. *AForge.Robotics*: Contiene las distintas interfaces y clases que dan soporte a kits robóticos. *AForge.Video*: Contiene las distintas interfaces y clases para el procesamiento de video[20].

1.2 Comparación entre las bibliotecas

La tabla 1.1 muestra las principales ventajas y desventajas de las bibliotecas en C/C++ para el trabajo con imágenes.

Tabla 1.1. Funciones implementadas en la *toolbox* MImg.

Bibliotecas	**Ventajas**	**Desventajas**
CImg	Es de código abierto, de fácil uso y posee un elevado grado de portabilidad. Las funciones de la biblioteca CImg aparecen implementadas en un único archivo.	Posee un número limitado de funciones para el procesamiento digital de imágenes.
ORFEO	Es de código abierto, está desarrollada en código C por lo que tiene gran eficiencia. Posee gran flexibilidad.	Se ha desarrollado originalmente para las imágenes de muy alta resolución (VHR), Su utilización desde MATLAB

		es muy compleja.
OpenCV	Posee una excelente funcionalidad y una gran calidad de desempeño. Los algoritmos están basados en estructuras de datos muy flexibles. Más de la mitad de las funciones han sido optimizadas, aprovechándose de la Arquitectura de Intel.	Posee muy poca portabilidad, Su utilización desde MATLAB es muy compleja.
Mamba	Es rápida y fácil para la codificación de algoritmos de morfología matemática. Se basa en una arquitectura muy simple. Sus funciones están codificadas para ser específicas, rápidas y simples.	La visualización de imágenes puede producir interferencias con IDLE. Se pueden producir errores al utilizar Mamba con otras bibliotecas de GUI.
ITK	Es una biblioteca de código abierto, incluye importantes algoritmos de registro y segmentación.	No implementa una interfaz gráfica o de visualización. Implementa fundamentalmente funciones de visualización y segmentación de imágenes.

1.3 Conclusiones del capítulo

En este capítulo se realizó un estudio de diferentes bibliotecas que se pueden utilizar para el desarrollo de aplicaciones de PDI. Las bibliotecas en C/C++ para el trabajo con imágenes poseen un gran desempeño. Su desarrollo en código C/C++ hace que posean una gran

eficiencia. Su flexibilidad, eficiencia y simplicidad hacen que sea muy utilizadas. De las bibliotecas disponibles CImg presenta las características deseables debido a su portabilidad y su fácil utilización desde MATLAB.

CAPÍTULO 2. MImg: *TOOLBOX* DE MATLAB PARA EL PROCESAMIENTO DE IMÁGENES

2.1 Uso de ficheros MEX

Se puede llamar subrutinas en C/C++ o Fortran desde la línea de comandos de MATLAB como si fueran las funciones incorporadas. Estos programas son llamados archivos-MEX binarios, los cuales son subrutinas dinámicamente vinculadas que el intérprete de MATLAB carga y ejecuta.

Los archivos MEX tienen algunas aplicaciones:

- Los grandes programas de C y Fortran pre-existentes pueden ser llamados desde MATLAB sin tener que ser reescritos como archivos .m.
- Las rutinas de rendimiento críticas pueden ser reemplazadas por implementaciones en C/C++.

Los archivos MEX no son apropiados para todas las aplicaciones. MATLAB es un ambiente productivamente alto cuya especialidad es eliminar la programación con complejidad temporal alta. En general, se debe hacer la mayor parte de la programación en MATLAB y no se debe usar archivos MEX a menos que su aplicación lo requiera.

La figura 2.1 ilustra el la creación y uso de los ficheros MEX.

El código fuente, implementado en C/C++, se compila usando la función “mex” de MATLAB. El producto de esta compilación es un archivo ejecutable desde MATLAB con extensión .mex. La función “mex” utiliza un compilador de C/C++ para realizar esta tarea. Los compiladores que soporta la función “mex” son:

- Microsoft Windows SDK 7.1 (requires .NET Framework 4.0).
- Microsoft Visual C++ 2013 Professional.
- Microsoft Visual C++ 2012 Professional.
- Microsoft Visual C++ 2010 Professional SP1.
- Microsoft Visual C++ 2008 Professional SP1 and Windows SDK 6.1.
- Intel C++ Composer XE 2013.
- Intel C++ Composer XE 2011.

La figura 2.2 muestra un ejemplo sobre el modo de selección del compilador por parte de la función "mex". En este ejemplo, hay instalados dos compiladores válidos (Lcc-win32 C 2.4.1 y Microsoft Visual C++ 2010), y se selecciona para ser usado por la función "mex" el segundo de ellos (Microsoft Visual C++ 2010).

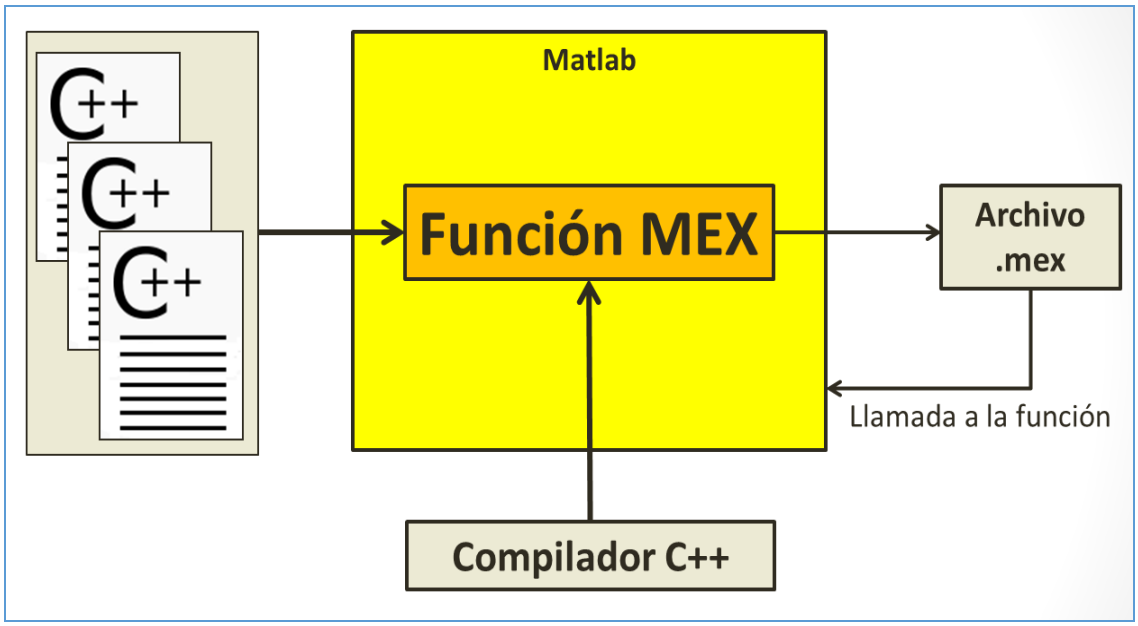


Figura 2.1. Creación y uso de los ficheros MEX.

```
>> mex -setup
Welcome to mex -setup.  This utility will help you set up
a default compiler.  For a list of supported compilers,
see
http://www.mathworks.com/support/compilers/R2011b/win32.h
tml

Please choose your compiler for building MEX-files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc-win32 C 2.4.1 in
C:\PROGRA~1\MATLAB\R2011b\sys\lcc
[2] Microsoft Visual C++ 2010 in C:\Program
Files\Microsoft Visual Studio 10.0
[0] None
Compiler: 2
```

Figura 2.2. Selección del compilador por parte de la función "mex".

La figura 2.3 muestra la forma en que se pasan los parámetros de entrada a un fichero MEX, las rutinas que son ejecutadas y como se devuelven los resultados a MATLAB.

En este ejemplo, la sintaxis del fichero MEX `func` es `[C,D] = func(A,B)`. En la figura, la llamada a la función `func` ordena a MATLAB pasar las variables A y B al fichero MEX. Las variables C y D no tienen asignado ningún valor.

La rutina `func.c` usa la función `mxCreate*` para crear el arreglo de MATLAB para los argumentos de salida. Esta asigna `plhs[0]` and `plhs[1]` a los apuntadores a los arreglos que se crean. Para ello se usa la función `mxGet*` para acceder a los datos de los parámetros de entrada `prhs[0]` and `prhs[0]`. Finalmente, se llama a la rutina pasando los apuntadores a los datos de entrada y salida como parámetros de la función. En el

momento del retorno a MATLAB, el valor de `plhs[0]` se asigna a `C` y `plhs[1]` a `D`, respectivamente.

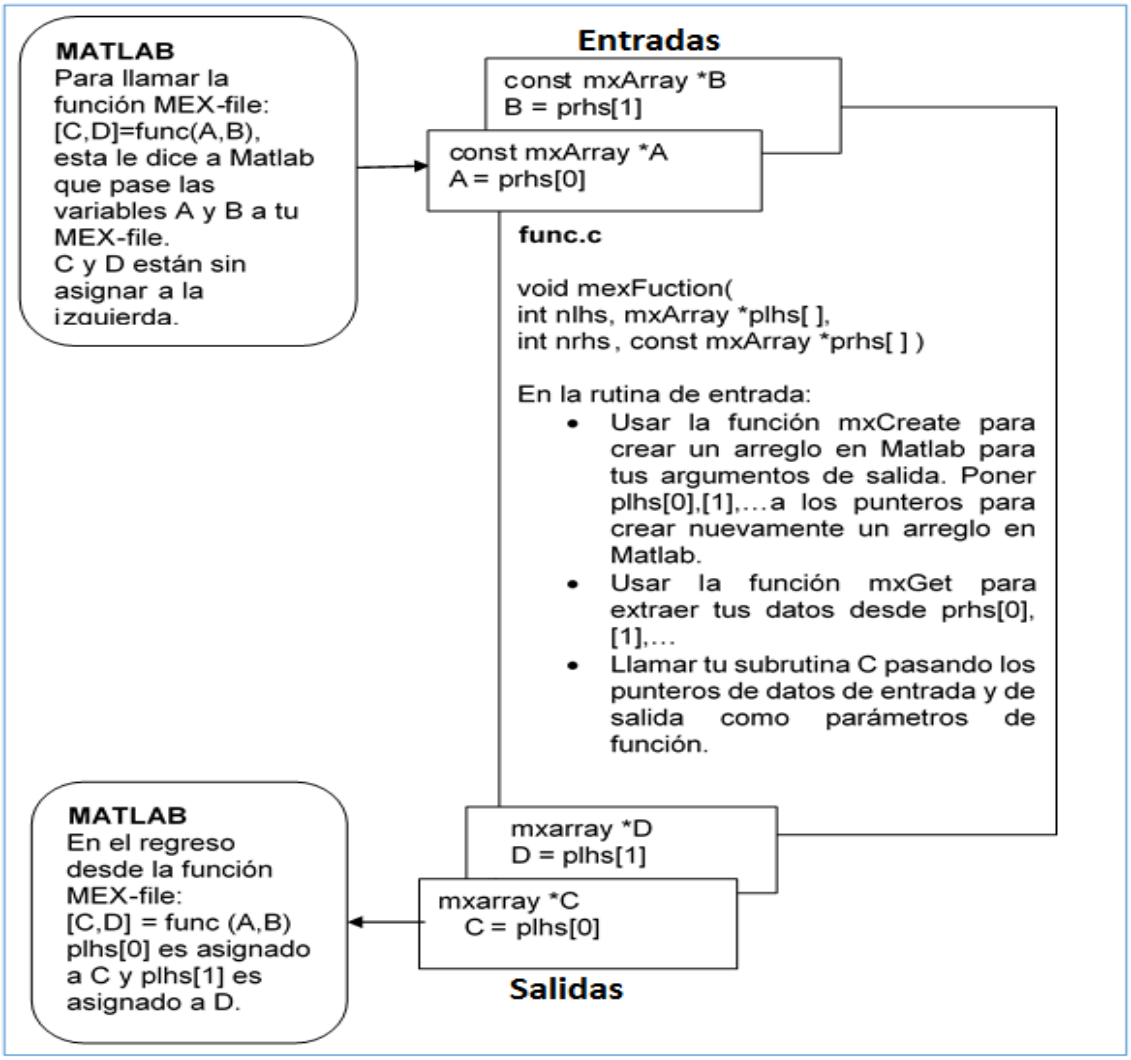


Figura 2.3. Creación y uso de los ficheros MEX.

2.2 Diseño de la *Toolbox*

2.1.1 Casos de uso y actores.

La figura 2.4 muestra un diagrama simplificado de casos de uso y actores de la *toolbox* MImg. Los diagramas de casos de uso y actores, propios del Lenguaje Unificado de Modelación UML [***], representan de una forma simplificada las distintas formas en que

un sistema puede ser usado por los diferentes tipos de usuarios (llamados actores) que éste pudiera tener.

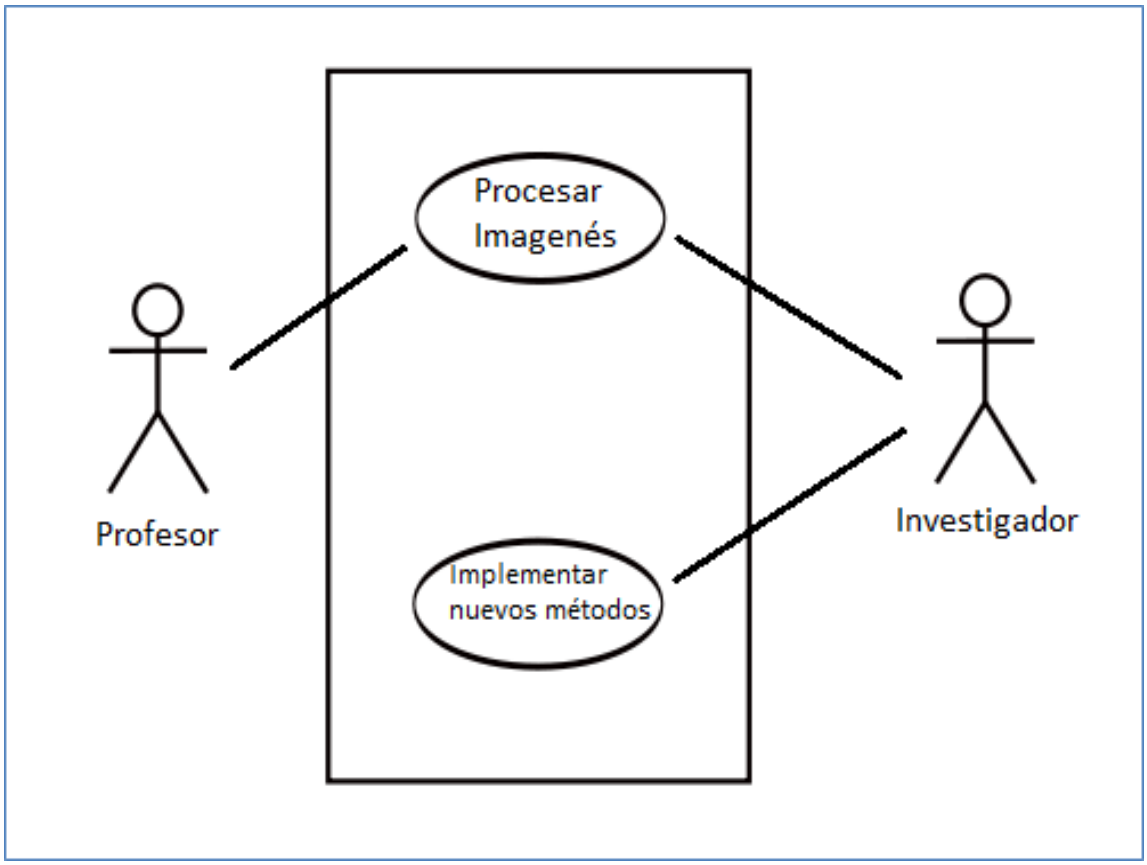


Figura 2.4. Diagrama de casos de uso y actores de la *toolbox* MImg.

Los casos de uso del sistema son los siguientes:

- Procesar imágenes: Las funciones implementadas en la *toolbox* MImg son básicamente para ser utilizadas en sistemas de procesamiento de imágenes.
- Implementar algoritmos de PDI: Las funciones implementadas se pueden usar, junto a las propias de MATLAB, para implementar nuevos algoritmos.

Los actores del sistema son los siguientes:

- Profesor: Este actor es aquel que usa la *toolbox* MImg con fines docentes.
- Investigador: Este actor es aquel que usa la *toolbox* MImg con fines de investigación.

2.1.2 Diseño de la *toolbox* MImg.

El diseño de la *toolbox* MImg se muestra en la figura 2.2.

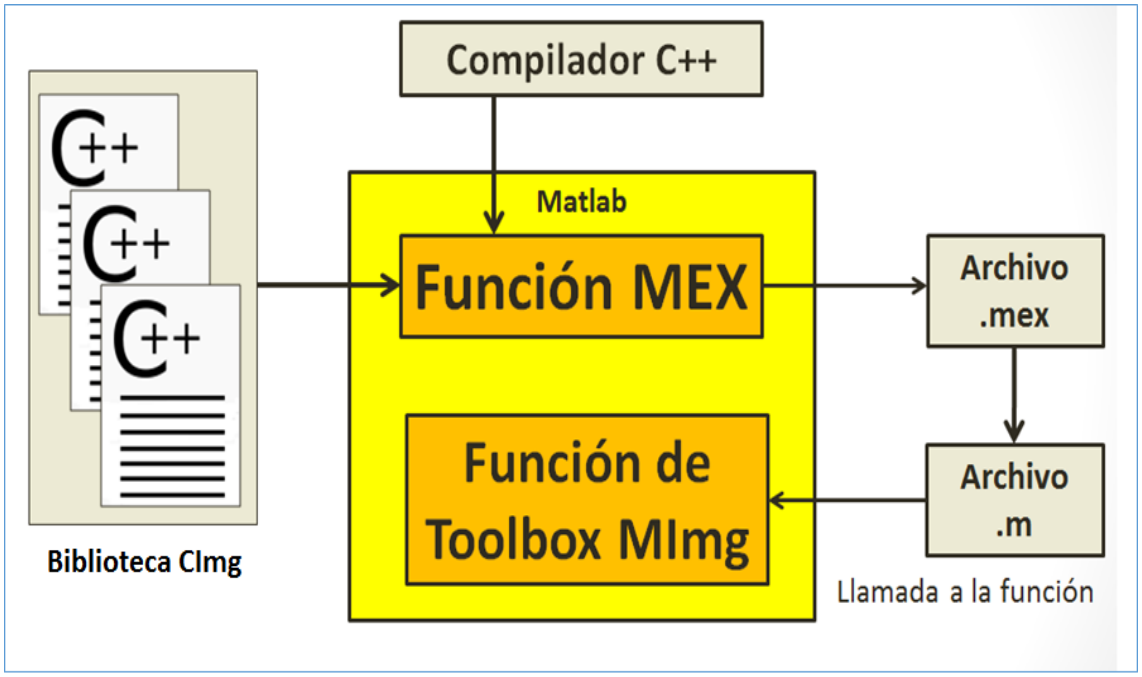


Figura 2.5. Diseño de la *toolbox* MImg.

Independientemente que las funciones implementadas en la biblioteca CImg pueden utilizarse directamente mediante los archivos .mex generados, la *toolbox* MImg implementa archivos .m que sirven de intermediarios entre los archivos .mex y el MATLAB. El uso de estos archivos .m es recomendable en vez del uso directo de los .mex debido a que el análisis de los requerimientos de cada función se realiza en aquellos y no en el código implementado en C/C++.

Las funciones implementadas en la *toolbox* MImg se listan en la Tabla 2.1 junto con una breve descripción. Asimismo, se muestra el nombre de los archivos .mex respectivo de cada función de la *toolbox*.

Tabla 2.1. Funciones implementadas en la *toolbox* MImg.

Archivo .m	**Archivo .cpp**	**Breve descripción**

miblur.m	ciblur.cpp	Realiza una difusión de la imagen utilizando el filtro de Canny-Deriche.
migetgradient.m	cigetgradient.cpp	Calcula el gradiente de la imagen en los ejes X,Y.
minoise.m	cinoise.cpp	Añade ruido aditivo aleatorio a los valores de la imagen.
minormalize1.m	cinormalize1.cpp	Se obtienen valores linealmente normalizados de un instante de la imagen, entre el valor mínimo y el valor máximo deseado.
minormalize2.m	cinormalize2.cpp	Normaliza múltiples valores de pixeles de un instante de la imagen con respecto a la norma L2
midilate.m	cidilate.cpp	Realiza una dilatación de la imagen con un elemento estructurante.
miequalize.m	ciequalize.cpp	Realiza la ecualización del histograma.
mierode.m	cierode.cpp	Realiza una erosión de la imagen con un elemento estructurante.
midistance1.m	cidistance1.cpp	Calcula la transformada de la distancia de acuerdo a un valor especificado.
midistance2.m	cidistance2.cpp	Calcula la transformada de la distancia según el valor especificado, con una métrica personalizada.

`mihaar.m`	`cihaar.cpp`	Calcula la transformada wavelet multisecular Haar.
`miresize.m`	`ciresize.cpp`	Realiza un redimecionamiento de la imagen.
`miRGBtoHSI.m`	`ciRGBtoHSI.cpp`	Convierte el color de los pixeles de (R,G,B) a (H,S,I).
`mirotate.m`	`cirotate.cpp`	Realiza una rotación de la imagen.
`mishift.m`	`cishift.cpp`	Realiza un traslado de la imagen.
`mithreshold.m`	`cithreshold.cpp`	Realiza una ubralización de la imagen.

2.2 Diseño de la *Toolbox*

La *toolbox* MImg implementa 16 funciones para el PDI. Estas funciones pueden utilizarse para realizar transformaciones de imágenes, filtrado, etc. A continuación se realiza una breve descripción de estas funciones, donde los parámetros de entrada y de salida se resaltan con la fuente Courier New.

2.2.1 Función minoise

Esta función añade ruido aleatorio a los valores de la imagen. La sintaxis del encabezado de la función `minoise` es la siguiente:

```
Function noiseim = minoise(im, sigma, noiset)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `sigma`: Amplitud del ruido aditivo aleatorio. Si sigma < 0, representa un porcentaje

del rango de valores globales.

- `noiset`: Tipo de ruido aditivo. Los valores de esta variable pueden ser:

  0 = Gaussiano.

  1 = Uniforme.

  2 = Sal y Pimienta.

  3 = Poisson.

  4 = Rician.

PARÁMETROS DE SALIDA:

`noiseim`: Imagen contaminada con ruido aditivo aleatorio.

2.2.2 Función miblur

Esta función realiza una difusión de la imagen utilizando el filtro de Canny-Deriche. La sintaxis del encabezado de la función `miblur` es la siguiente:

```
Function noiseim = miblur(im, sigmax, sigmay, sigmaz)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `sigmax`: Difusión a lo largo del eje-X.
- `sigmay`: Difusión a lo largo del eje-Y.
- `sigmaz`: Difusión a lo largo del eje-Z.

PARÁMETROS DE SALIDA:

`blurim`: Imagen difusa.

2.2.3 Función midilate

Esta función realiza una dilatación de la imagen con un elemento estructurante cuadrado. La sintaxis del encabezado de la función `midilate` es la siguiente:

```
Function dilateim = midilate(im, mask)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `mask`: Tamaño del elemento estructurante.

PARÁMETROS DE SALIDA:

`dilateim` : Imagen dilatada.

2.2.4 Función midistance1

Esta función calcula la transformada de la distancia de acuerdo a un valor especificado. La sintaxis del encabezado de la función `midistance1` es la siguiente:

```
distanceim = midistance(im, const_T)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `const_T`: Constante T.

PARÁMETROS DE SALIDA:

`distanceim` : Transformada de distancia de la imagen.

2.2.5 Función midistance2

Esta función calcula la transformada de la distancia según el valor especificado, con una métrica personalizada. La sintaxis del encabezado de la función `midistance2` es la siguiente:

```
distanceim = midistance(im, const_T ,metric_mask)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `const_T`: Constante T.

- `metric_mask`: Métrica personalizada.

PARÁMETROS DE SALIDA:

`distanceim` : Transformada de distancia de la imagen.

2.2.6 Función miequalize

Esta función realiza la ecualización del histograma. La sintaxis del encabezado de la función `miequalize` es la siguiente:

```
equalizeim = miequalize(im, nb_levels, value_min, value_max)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `nb_levels`: Número de niveles de histograma usado para la ecualización.
- `value_min`: Valor mínimo del píxel, considerado para el cálculo del histograma.
- `value_max`: Valor máximo del píxel, considerado para el cálculo del histograma.

PARÁMETROS DE SALIDA:

`equalizeim` : Imagen con el histograma ecualizado.

2.2.7 Función mierode

Esta función realiza una erosión de la imagen con un elemento estructurante cuadrado. La sintaxis del encabezado de la función `mierode` es la siguiente:

```
Function erodeim = midilate(im, mask)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `mask`: Tamaño del elemento estructurante.

PARÁMETROS DE SALIDA:

`erodeim`: Imagen erosionada.

2.2.8 Función migetgradient

La función `migetgradient` calcula una lista de imágenes que se corresponden con sus gradientes en los ejes X, Y utilizando un método especificado. La sintaxis del encabezado de la función `migetgradient` es la siguiente:

```
[gradX, gradY, gradAbs] = migetgradient(im, scheme)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `scheme`: Esquema numérico utilizado para el cálculo del gradiente. Los valores de esta variable pueden ser:

  -1 = Diferencias finitas hacia atrás.

  0 = Diferencias finitas centradas.

  1 = Diferencias finitas hacia adelante.

  2 = Usando la máscara de Sobel.

  3 = Utilizando una máscara invariante a la rotación (por defecto).

  4 = Usando el filtro recursivo de Deriche.

PARÁMETROS DE SALIDA:

- `gradX`: Gradiente en el eje x.
- `gradY`: Gradiente en el eje Y.
- `gradAbs`: $\sqrt{G_X^2 + G_Y^2}$, donde G_X y G_Y son los gradientes en el eje X y Y, respectivamente.

`[gradX, gradY, gradAbs]`: Calcula la lista de las imágenes, correspondiente a los gradientes-XY de la imagen.

2.2.9 Función haar

La función `mihaar` calcula la transformada wavelet multiescalar Haar. La sintaxis del encabezado de la función `mihaar1` es la siguiente:

```
Function haarim = mihaar(im, axis, invert, nb_scales)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `axis`: Ejes considerados para la transformada.
- `invert`: Inverso de la transformada directa.
- `nb_scales`: Número de escalas utilizadas en la transformada.

PARÁMETROS DE SALIDA:

`haarim`: Transformada wavelet multiescalar haar de la imagen.

2.2.10 Función minormalize1

Con la función `minormalize1` se obtienen valores linealmente normalizados de un instante de la imagen, entre el valor mínimo y el valor máximo deseado. La sintaxis del encabezado de la función minormalize1 es la siguiente:

```
Function normalizeim = minormalize1(im, min, max)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `min`: Valor mínimo deseado de la imagen resultante.
- `max`: Valor mínimo deseado de la imagen resultante.

PRAMETROS DE SALIDA:

`normalizeim`: Imagen normalizada.

2.2.11 Función minormalize2

La función `minormalize2` normaliza múltiples valores de pixeles de un instante de la imagen con respecto a su norma L2. La sintaxis del encabezado de la función `minormalize2` es la siguiente:

```
Function normalizeim = minormalize2(im)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.

PRAMETROS DE SALIDA:

`normalizeim`: Imagen normalizada.

2.2.12 Función miresize

La función realiza un redimecionamiento de la imagen. La sintaxis del encabezado de la función `minormalize1` es la siguiente:

```
Filteredim = miresize(im, sizex, sizey, sizez, sizec,
interpolation_type, border_conditions)
```

PARÁMETROS DE ENTRADA:

`im`: Imagen de entrada.

`sizex`: Tamaño a lo largo del eje X.

- `sizey`: Tamaño a lo largo del eje Y.
- `sizez`: Tamaño a lo largo del eje Z.
- `sizec`: Tamaño a lo largo del eje C.
- `interpolation_type`: Método de interpolación.

  -1 = Sin interpolación: cambio de tamaño de memoria en bruto

  0 = Sin interpolación.

  1 = Interpolación vecino más cercano.

2 = Interpolación promedio móvil.

3 = Interpolación linear.

4 = Interpolación de rejilla.

5 = Interpolación bicúbica.

6 = Interpolación lanczos.

- `border_conditions`: Condición de borde.

centrado_x Establecer el tipo de centrado (solo si interpolation_type=0).

centrado_y Establecer el tipo de centrado (solo si interpolation_type=0).

centrado_z Establecer el tipo de centrado (solo si interpolation_type=0).

centrado_c Establecer el tipo de centrado (solo si interpolation_type=0).

PARÁMETROS DE SALIDA:

`filteredim`: Imagen redimencionada.

2.2.13 Función miRGBtoHSI

La función `miRGBtoHSI` convierte el color de los pixeles de (R,G,B) a (H,S,I). La sintaxis del encabezado de la función `miRGBtoHSI` es la siguiente:

```
Function RGBtoHSIim = miRGBtoHSI(im)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.

PRAMETROS DE SALIDA:

`RGBtoHSIim`: Imagen en HSI.

2.2.14 Función mirotate

La función `mirotate` realiza una rotación de la imagen. La sintaxis del encabezado de la función `mirotate` es la siguiente:

`filteredim = mirotate(im, angle, cx, cy, zoom, border_conditions, cond)`

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `angle`: Ángulo de rotación (en grados).
- `cx`: Centro de rotación por las X.
- `cy`: Centro de rotación por las Y.
- `zoom`: ampliación.
- `border_conditions`: Condición de borde.

0 = derichlet.

1 = neumann.

2 = cíclico.

- `cond`: Tipo de rotación:

0 = valor cero en los bordes.

1 = píxel cercano.

2 = cíclico.

PARÁMETROS DE SALIDA:

`rotateim`: Imagen rotada.

2.2.15 Función mishift

La función `mishift` realiza un traslado de la imagen. La sintaxis del encabezado de la función `mishift` es la siguiente:

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `deltax`: Traslado a lo largo del eje-X.
- `deltay`: Traslado a lo largo del eje-Y.

- `deltaz`: Traslado a lo largo del eje-Z.
- `deltac`: Traslado a lo largo del eje-C.
- `border_conditions`: Condición de borde.

0 = derichlet.

1 = neumann.

2 = fourier.

PARÁMETROS DE SALIDA:

`shiftim`: Imagen trasladada.

2.2.16 Función mithreshold

La función `mithreshold` realiza una ubralización de la imagen. La sintaxis del encabezado de la función `mithreshold` es la siguiente:

```
thresholdim = mithreshold(im, value, soft_threshold, strict_threshold)
```

PARÁMETROS DE ENTRADA:

- `im`: Imagen de entrada.
- `value`: Valor del umbral.
- `soft_threshold`: Indica si el umbral es suave.
- `strict_threshold`: Indica si el umbral es estricto.

PARÁMETROS DE SALIDA:

`thresholdim` : Imagen umbralada.

2.3 Conclusiones del capítulo

En este capítulo se realizó una descripción de las herramientas disponibles en MATLAB para la utilización de bibliotecas escritas en C, en particular del uso de funciones MEX. Adicionalmente se realizó el esquema de casos de uso y actores y se explica el diseño de la

toolbox que se propone. Finalmente se listaron y explicaron las funciones contenidas en MImg.

CAPÍTULO 3. APLICACIÓNSTRACIONES DEL USO DE LA *TOOLBOX* MImg

3.1 Costo computacional

En la tabla 3.1 se lista el costo computacional, determinado por el tiempo de ejecución, de un conjunto de funciones de la *toolbox* MImg. En la tabla 3.2 se muestra el tiempo de ejecución de una aplicación de PDI que utiliza algunas funciones de la *toolbox* MImg. Los tiempos de ejecución se obtuvieron en MATLAB mediante la función *profile*. Se tomó el menor tiempo que se obtuvo con las funciones de MATLAB y el mayor que se obtuvo con las funciones de la *toolbox* MImg.

Tabla 3.1. Tiempos de ejecución, de un conjunto de funciones de la *toolbox* MImg y sus equivalentes en MATLAB.

Función de MATLAB	**Tiempo de ejecución (s)**	**Función de la *toolbox* MImg**	**Tiempo de ejecución (s)**
imnoise	0.033	minoise	0.010
histeq	0.034	miequalize	0.004
imresize	0.036	miresize	0.022

imrotate	0.027	mirotate	0.004
bwdist	0.006	midistance1	0.004

Tabla 3.2. Tiempos de ejecución, de una aplicación de PDI.

Ejercicio de PDI	**Tiempo de ejecución (s) mediante MATLAB**	**Tiempo de ejecución (s) mediante la *toolbox* MImg**
Detección de bordes mediante el cálculo del gradiente en imagen contaminada con ruido Poisson	0.539	0.245

3.2 Set de aplicaciones

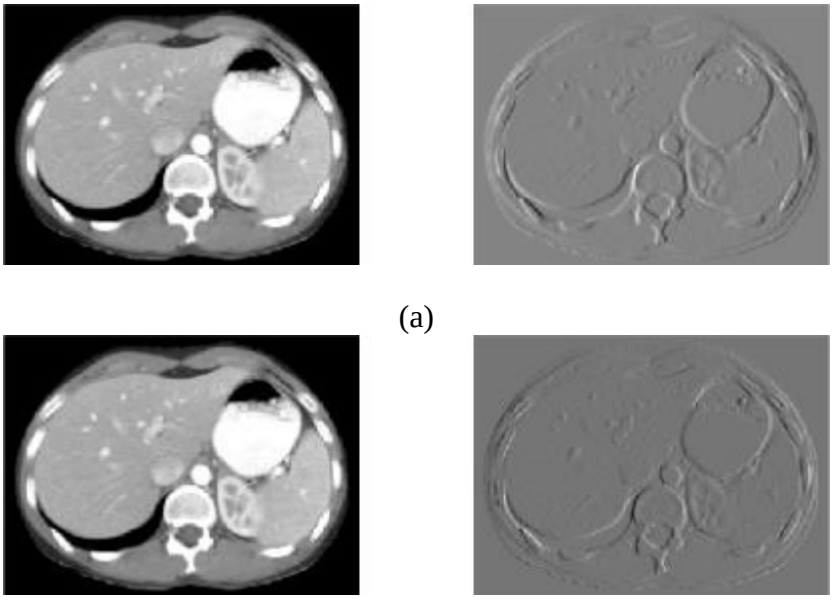
Las siguientes aplicaciones demuestran el uso de las funciones implementadas en la *toolbox* MImg.**Detección de bordes mediante el cálculo del gradiente**

La siguiente aplicación ilustra la detección de bordes en imágenes mediante el cálculo del gradiente. En esta aplicación se utiliza la función de la *toolbox* MImg:

- `migetgradient`

La aplicación consta de los siguientes pasos:

1. Se carga la imagen médica "CT_AbdomenLiver".
2. Se calcula el gradiente usando los siguientes métodos:
    a. Máscara de Sobel.
    b. Filtro recursivo de Deriche.
3. Se visualizan los resultados.



(a)

(b)

Resultado del cálculo del gradiente: (a) utilizando Máscara de Sobel y (b) utilizando filtro recursivo de Deriche.

3.2.2 Normalización de imágenes

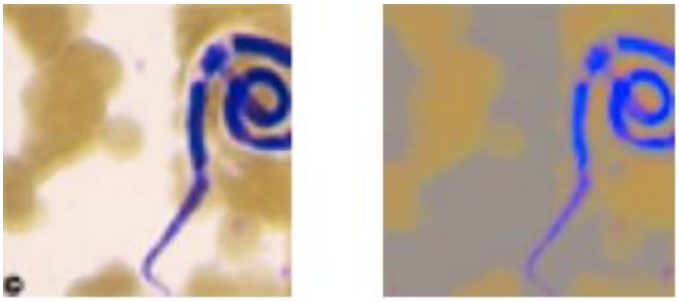
En la siguiente aplicación se realiza la normalización de una imagen médica RGB, respecto a la norma-L2. Además se realiza la normalización entre el valor mínimo y el valor máximo deseado de una imagen en escala de grises. En la aplicación se utiliza las funciones de la *toolbox* MImg:

- `minormalize1`
- `minormalize2`

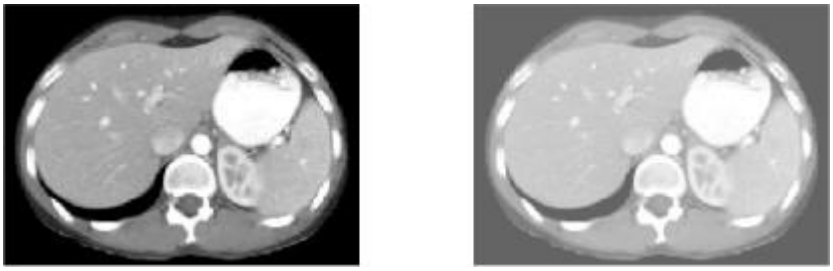
La aplicación consta de los siguientes pasos:

1. Se carga la imagen médica “filariasis.jpg”.
2. Se normaliza la imagen respecto a la norma-L2.
3. Se visualizan los resultados.
4. Se carga la imagen “CT_AbdomenLiver”.
5. Se normaliza la imagen tomando como valor mínimo 100 y como valor máximo 255.

6. Se visualizan los resultados.



(a)



(b)

Resultado de la normalización de las imágenes: (a) Imagen RGB normalizada respecto a la norma L2 y (b) Imagen en escala de grises normalizada entre 100 y 255.

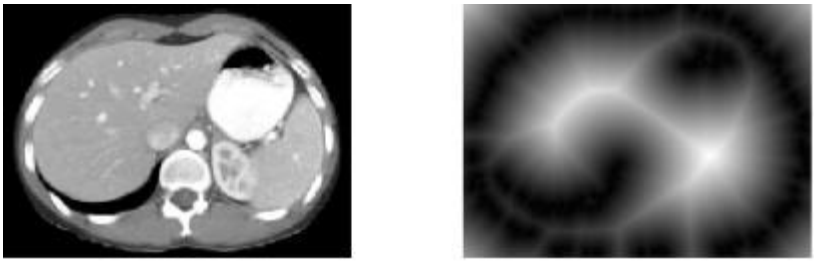
3.2.3 Cálculo de la transformada de distancia en imágenes en escala de grises

En la siguiente aplicación se calcula la transformada de distancia de una imagen en escala de grises. En la aplicación se utiliza las funciones de la *toolbox* MImg:

- `midistance1`

La aplicación consta de los siguientes pasos:

1. Se carga la imagen médica “CT_AbdomenLiver”.
2. Se calcula la transformada de distancia.
3. Se visualizan los resultados.



Cálculo de la transformada de distancia.

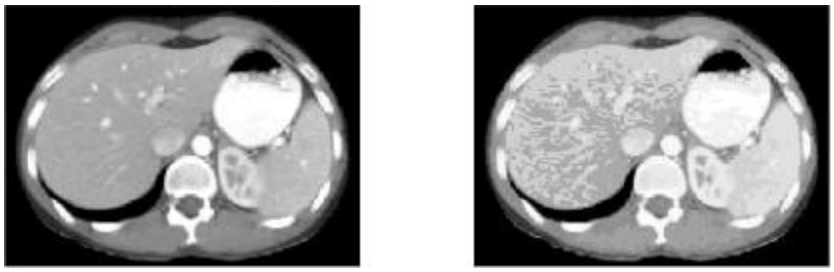
3.2.4 Ecualización del histograma en imágenes en escala de grises

En la siguiente aplicación se realiza la ecualización del histograma de una imagen médica en escala de grises. En la aplicación se utiliza las funciones de la *toolbox* MImg:

- `Miequalize`

La aplicación consta de los siguientes pasos:

1. Se carga la imagen médica “CT_AbdomenLiver”.
2. Se le ecualiza el histograma a la imagen, con 10 niveles de histograma y tomando como valor mínimo 100 y como valor máximo 255.
3. Se visualizan los resultados.



Ecualización del histograma con 10 niveles de histograma y tomando como valor mínimo 100 y como valor máximo 255.

3.2.5 Redimensionamiento de imágenes en escala de grises

En la siguiente aplicación se realiza el redimensionamiento de una imagen médica en escala de grises. En la aplicación se utiliza la función de la *toolbox* MImg:

- `miresize`

La aplicación consta de los siguientes pasos:

1. Se carga la imagen médica “CT_AbdomenLiver”.
2. Se redimensiona la imagen a un tamaño de 250x300 pixeles, utilizando interpolación bicúbica.
3. Se visualizan los resultados.

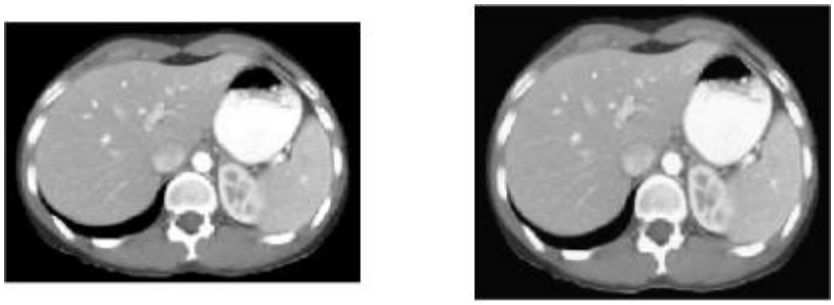


Imagen redimensionada a un tamaño de 250x300 pixeles utilizando interpolación bicúbica.

3.2.6 Contaminación de imágenes con ruido aditivo aleatorio

En la siguiente aplicación se realiza la contaminación de una imagen médica en escala de grises, añadiendo ruido aditivo aleatorio a los valores de la imagen. En la aplicación se utiliza la función de la *toolbox* MImg:

- `minoise`

La aplicación consta de los siguientes pasos:

1. Se carga la imagen médica “CT_AbdomenLiver”.
1. Se contamina la imagen con ruido aditivo gaussiano con dispersión igual 30.
2. Se visualizan los resultados.

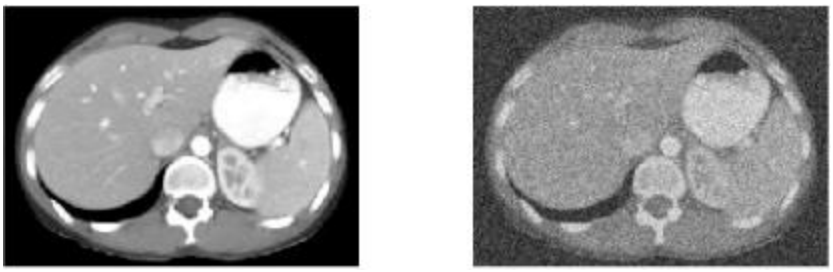


Imagen contaminada con ruido aditivo gaussiano con dispersión igual 30.

3.2.7 Eliminación de ruido en imágenes

La siguiente aplicación ilustra el uso de varios métodos implementados en la *toolbox* MImg que pueden ser usados en la eliminación de ruido. En esta aplicación se usan las siguientes funciones de la *toolbox* MImg:

- `minoise`
- `miblur`

La aplicación consta de los siguientes pasos:

1. Se carga la imagen médica "CT_AbdomenLiver".
2. Se contamina la imagen utilizando un ruido blanco gaussiano con dispersión igual 30.
3. Se realiza la eliminación del ruido introducido utilizando la técnica difusión de Canny-Deriche.
4. Se visualizan los resultados.

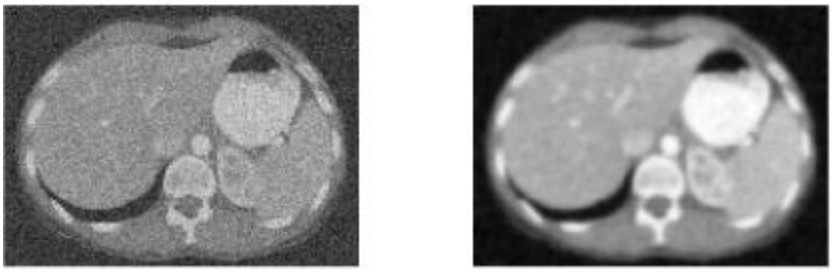


Imagen con el ruido eliminado utilizando la técnica difusión de Canny-Deriche.

3.3 Ejercicios de PDI

3.3.1 Detección de imagen por tamaño

El siguiente ejercicio se emplea distintas funciones de la *toolbox* MImg para la realización de operaciones morfológicas. Con el objetivo de eliminar todos los cuadrados, excepto los tres cuadrados más grandes que se observan en la imagen. Para la realización de este ejercicio se utilizan las siguientes funciones de la *toolbox* MImg:

- `mierode`
- `midilate`

El ejercicio consta de los siguientes pasos:

1. Se carga la imagen "cuadrados".
2. Se erosiona la imagen utilizando un elemento estructurante cuadrado de lado 13.
3. Se dilata la imagen utilizando un elemento estructurante cuadrado de lado 13.
4. Se visualizan los resultados.

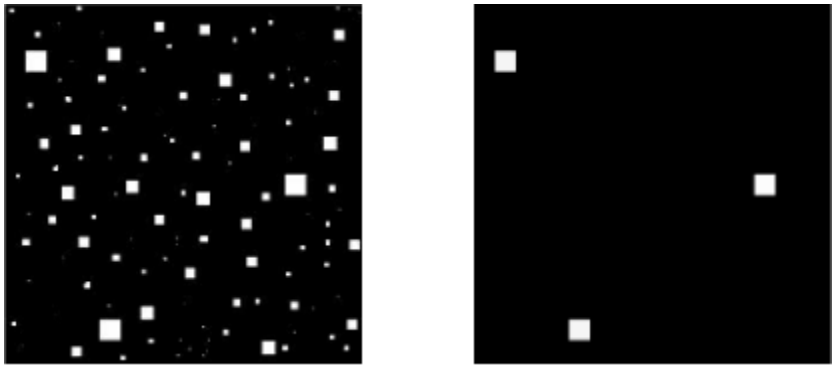


Imagen que muestra los 3 cuadrados más grandes

3.3.2 Procesamiento de la imagen circuito electrónico para eliminar las conexiones más finas

El siguiente ejercicio se emplea distintas funciones de la *toolbox* MImg para la realización de operaciones morfológicas. Con el objetivo de eliminar en el gráfico sólo las líneas de conexión más delgadas, y restituir al resto del circuito su tamaño original. Para la realización de este ejercicio se utilizan las siguientes funciones de la *toolbox* MImg:

- `mierode`
- `midilate`

El ejercicio consta de los siguientes pasos:

1. Se carga la imagen “circuito”.
2. Se erosiona la imagen utilizando un elemento estructurante cuadrado de lado 4.
3. Se dilata la imagen utilizando un elemento estructurante cuadrado de lado 4.
4. Se visualizan los resultados.

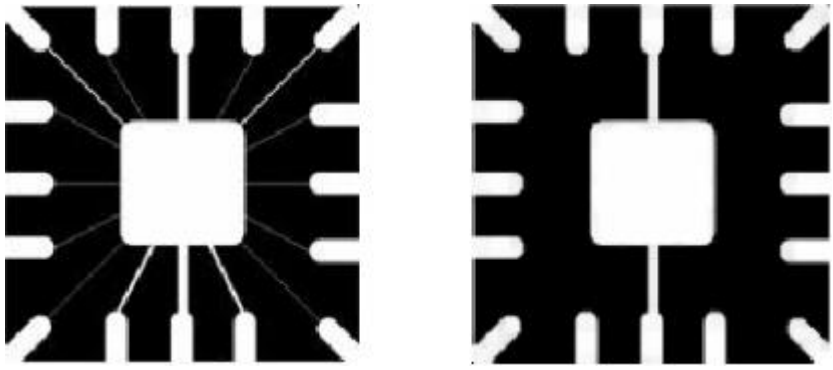


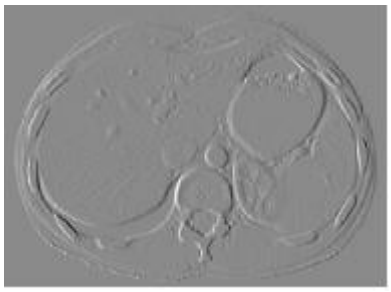
Imagen que muestra las conexiones más finas eliminadas

3.3.3 Detección de bordes mediante el cálculo del gradiente en imagen contaminada con ruido Poisson

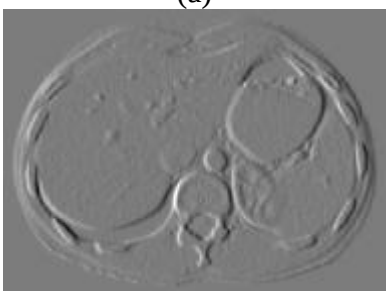
En el siguiente ejercicio se utiliza distintas funciones de la *toolbox* MImg. Estas funciones se utilizan para detectar los bordes en una imagen previamente redimensionada y contaminada con ruido de Poisson Para la realización de este ejercicio se utilizan las siguientes funciones de la *toolbox* MImg:

- `miresize`
- `minoise`
- `miblur`
- `migetgradient`

1. Se carga la imagen "CT_AbdomenLiver".
2. Se redimensiona la imagen a un tamaño de 512x695 pixeles, utilizando interpolación bicúbica.
3. Se contamina la imagen con ruido de Poisson.
4. Se realiza la eliminación del ruido introducido utilizando la técnica difusión de Canny-Deriche.
5. Se realiza la detección de bordes mediante el cálculo del gradiente utilizando máscara de Sobel.
6. Se visualizan los resultados.



(a)



(b)

Resultado de la detección de bordes: (a) sobre la imagen original y (b) sobre la imagen resultante del filtrado.

3.4 Interfaz para la utilización de las funciones desde MATLAB

La interfaz que se muestra en la figura 3.1 implementa un conjunto de funciones de la *toolbox* MImg, así como algunos ejemplos del funcionamiento de las mismas.

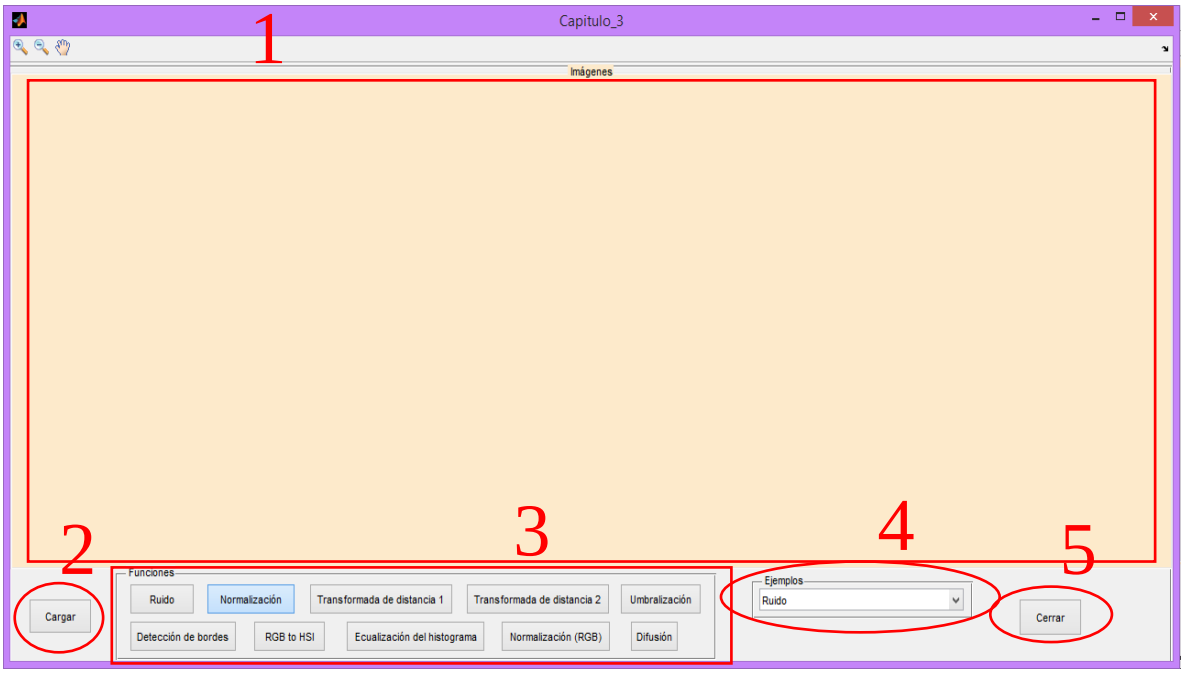


Fig. 3.1 Interfaz para la utilización de las funciones desde MATLAB

3.4.1 Estructura de la interfaz

La interfaz consta de cinco partes fundamentales, como se observa en la figura 3.1. El funcionamiento de estas partes se describe continuación:

Partes de la interfaz	Nombre	Breve descripción
1	Imágenes	Se muestran las imágenes cargadas e imágenes procesadas.
2	Botón cargar	Carga las imágenes a procesar.
3	Funciones	Conjunto de funciones de la *toolbox* MImg

4	Ejemplos	Conjunto de ejemplos del funcionamiento de algunas funciones
5	Botón cerrar	Carga las imágenes a procesar.

3.5 Conclusiones del capítulo

En este capítulo se presentaron los resultados obtenidos de desarrollar el *toolbox* MImg, haciendo énfasis en su posible utilización para de desarrollo de las investigaciones, docencia e introducción de resultados del CEETI, teniendo en cuenta sus posibilidades y el costo computacional de las funciones implementadas. La utilización de bibliotecas desarrolladas en C/C++ mejora el desempeño de aplicaciones en MATLAB. Estas bibliotecas incorporan nuevas características, además de poseer un costo computacional menor. Las herramientas implementadas tienen aplicación directa e inmediata en las investigaciones que desarrolla el CEETI así como en la docencia de pregrado y posgrado.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

A partir de los resultados del presente trabajo se llegó a las siguientes conclusiones:

- De las bibliotecas disponibles CImg presenta las características deseables debido a su portabilidad y su fácil utilización desde MATLAB.
- La utilización de la biblioteca CImg desarrollada en C/C++ mejora el desempeño de aplicaciones en MATLAB.
- Las herramientas implementadas tiene aplicación directa e inmediata en las investigaciones que desarrolla el CEETI así como en la docencia de pregrado y posgrado.

Recomendaciones

Para dar continuidad al presente trabajo se dan las siguientes recomendaciones:

1. Utilizar la herramienta creada en la docencia e investigación del Centro de Estudios de Electrónica y Tecnologías de la Información.
2. Aumentar el alcance de la investigación a otras herramientas de software que se puedan utilizar para PDI como puede ser Python.

REFERENCIAS BIBLIOGRÁFICAS

1. Tschumperlé, D., *The CImg Library C++ Termplate Image Processing Toolkit.* GREYC Laboratory (CNRS UMR 6072), Image Team, 6 Bd Marechal Juin, 14050 Caen/France.
2. Tschumperlé, D., *The CImg Library and G'MIC Open-Source Toolboxes for Image Processing at Different Levels.* October 2009.
3. Selvam, V., *An Introduction to the CImg Library A project conducted with the aid of Professor Todd Wittman.*
4. *The CImg Library Reference Manual 1.2.7 Generated by Doxygen 1.5.3.* Jan 25, 2008.
5. Tinel C., F.D., de Boissezon H., Grizonnet M., Michel J., *The ORFEO acompaniment program and ORFEO ToolBox.* 2012: p. 7102 – 7105.
6. *The Orfeo ToolBox Cookbook, a guide for non-developers Updated for OTB-5.0 OTB Development Team.* May 27, 2015.
7. Gil, L.I.J., *Desarrollo de nuevos algoritmos para procesamiento de imágenes hiperespectrales en ORFEO Toolbox.*
8. May S., I.J., *Urban area detection and segmentation using OTB.* 2009. **4**: p. 928-931.
9. Inglada J., C.E., *The Orfeo Toolbox remote sensing image processing software.* 2009. **4**: p. 733-736.
10. *The OpenCV Tutorials. Release 2.3.3.* March 11, 2012.
11. Laganière, R., *OpenCV 2 Computer Vision Application Programming Cookbook.* May 2011.
12. Culjak I., A.D., Pribanic T., Dzapo H., Cifrek M., *A brief introduction to OpenCV.* 2012 p. 1725-1730.
13. Kari Pulli, A.B., Kirill Kornyakov, Victor Eruhimov, *Real-Time Computer Vision with OpenCV.* june 2012.
14. Jayneil Dalal, S.P., *Instant OpenCV Starter.* May, 2013.
15. Gary Bradski, A.K., *Learning OpenCV* October 2013: p. 16-22.
16. Oscar Deniz Suarez, M.d.M.F.C., Noelia Vállez Enano, Gloria Bueno García,

Ismael Serrano Gracia, Julio Alberto Patón Incertis, Jesus Salido Tercero, *OpenCV Essentials.* p. Chapter No.4.

17. Gloria Bueno Garcia, O.D.S., Jose Luis Espinosa Aranda, *Learning Image Processing with OpenCV.* 2015.
18. Howse, J., *OpenCV Computer Vision with Python.* April 2013.
19. *EmguCV: OpenCV in .NET (C#, VB, C++ and more). Disponible en* http://ww.emgu.com
20. S.Rivas Rivas, I.T.G., J.D.Baena Carrasco, *Librerías para el Procesamiento y Síntesis de Imágenes. Una Revisión Actual.*
21. Beucher, S., *Algorithmic description of erosions and dilations in Mamba.*
22. Nicolas Beucher, S.B., *The MAMBA IMAGE Library.*
23. Wurflinger K., K.P., Maurutschek P., *Mamba-a programmable broadband multimedia interface board. Performance, Computing, and Communications.* 2001: p. 281-286.
24. Nicolas Beucher, S.B., *Hierarchical Queues: general description and implementation in MAMBA Image library.* April 2011.
25. *Mamba RealTime Api Reference Manual. Automatically generated using doxygen* February 18, 2012.
26. Chadwick G.A., M.S.W., *Mamba: A scalable communication centric multi-threaded processor architecture.* 2012: p. 277-283.
27. I. Bitter, R.V.U., I. Wolf, L. Ibañez, J.M. Kuhnigk, *Comparison of four freely available frameworks for image processing and visualization that use ITK.* 2007: p. 483–493.
28. Luis Ibañez, W.S., Lydia Ng, Josh Cates and the Insight Software Consortium, *The ITK Software Guide Second Edition Updated for ITK version 2.4.*
29. Hui Dong, L.X., Jin Zhang, Andong Cai, *Medical Image Reconstruction Based on ITK and VTK.* 2013: p. 642-645.
30. Trezise, M., *SimITK: Model Driven Engineering for Medical Imaging.* August, 2013.
31. Matthias K. Wolf, S.J.R., Julien J. Florian, LinkHeinz-Otto P., *Embedding VTK and ITK into a visual programming and rapid prototyping platform.* 2006.
32. David G Gobbi, P.M., Andrew W L Dickinson, Purang Abolmaesumi, *SimITK: Visual Programming of the ITK Image-Processing Library within Simulink.* 2014.
33. Gianluca Paladini, F.S.A., *An extensible imaging platform for optical imaging applications.* 2009.
34. Perez M.E.M., H.A.D., Thorn S.A., Parker K.H., *Improvement of a retinal blood vessel segmentation method using the Insight Segmentation and Registration Toolkit (ITK).* 2007: p. 892-895.
35. Albert, A., *A computation model with data flow sequencing.*
36. *Java 2D API. Disponible en* http://java.sun.com/products/java-media/2D/index.jsp

37. *AForge.NET. Disponible en* http://www.aforgenet.com/

ANEXOS

Anexo I Funcionalidades de la biblioteca CImg

```
1  #include "CImg.h"
2  using namespace cimg_library;
3
4  int main(int argc, char **argv){
5
6      CImg<unsigned char> img(300,200,1,3);
7      img.fill(32);
8      img.noise(128);
9      img.blur(2,0,0);
10     const unsigned char white[] = {255,255,255};
11     img.draw_text("Hello World",80,80,white,0,32);
12     img.display();
13
14     return 0;
15 }
```

Las partes de este código son las siguientes:

- Líneas 1 y 2: Encabezado necesario para usar la biblioteca CImg.
- Línea 6: Creación de una imagen (variable img) de dimensión 300x200, con una profundidad de 1 y con 3 canales.
- Línea 7: La imagen se rellena con el color 32. En este punto la imagen contiene en cada píxel el valor (32, 32, 32).
- Línea 8: La imagen es contaminada con un ruido blanco gaussiano de amplitud 128.
- Línea 9: La imagen es difuminada usando el filtro de Canny-Deriche con amplitud 2, 0 y 0 en los ejes “x”, “y” y “z”, respectivamente.
- Línea 10 y 11: En la imagen se pega una etiqueta titulada “*Hello World*”, en la posición (80,80), de color blanco, fondo transparente y sin opacidad.
- Línea 12: La imagen se muestra en pantalla. El resultado mostrado es como el que se muestra en la figura 1.6.

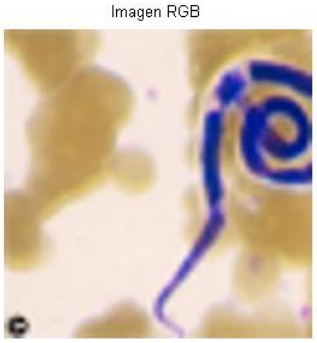
Anexo II Manual de ayuda para la *toolbox* MImg

Contents

IMÁGENES

```
i=imread('CT_AbdomenLiver.bmp');
i2=imread('filariasis.jpg');
```

```
figure(1);imshow(i);title('Imagen en escala de grises');
figure(2);imshow(i2);title('Imagen RGB');
```



FUNCIONES

* miblur

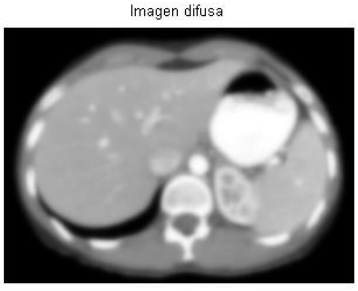
Realiza una difusión de la imagen usando el filtro de Canny-Deriche.

```
% filteredim = miblur(im,sigmax,sigmay,sigmaz)

% PARÁMETROS DE ENTRADA:
%  sigmax : Difusión a lo largo del eje-X
%  sigmay : Difusión a lo largo del eje-Y
%  sigmaz : Difusión a lo largo del eje-Z

% PARÁMETROS DE SALIDA:
% filteredim : Imagen difusa.

difusión = miblur(i,2,2);
figure(3);imshow(i);title('Imagen original');
figure(4);imshow(difusión,[]);title('Imagen difusa');
```



* minoise

Añade ruido aditivo aleatorio a los valores de la imagen.

```
% filteredim = minoise(im,sigma,noiset)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.
%  sigma : Amplitud del ruido aditivo aleatorio. Si sigma < 0,
%          representa un porcentaje del rango de valores globales.
%  noiset : Tipo de ruido aditivo.  Los valores de esta variable pueden
ser:
%         0 = Gausiano.
%         1 = Uniforme.
%         2 = Sal y Pimienta.
%         3 = Poisson.
%         4 = Rician.

% PARÁMETROS DE SALIDA:
% filteredim : Imagen contaminada con ruido aditivo aleatorio.

ruido = minoise(i,40,1);
figure(5);imshow(i);title('Imagen original');
figure(6);imshow(ruido,[]);title('Imagen contaminada con ruido');
```

* midistance1

Calcula la transformada de la distancia de acuerdo a un valor especificado.

```
% filteredim = midistance(im,const_T)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.
%  const_T : Constante T.

% PARÁMETROS DE SALIDA:
% filteredim : Transformada de distancia de la imagen.

dist1 = midistance1(i,1);
figure(7);imshow(i,[]);title('Imagen original');
figure(8);imshow(dist1,[]);title('Transformada de distancia');
```

* midistance2

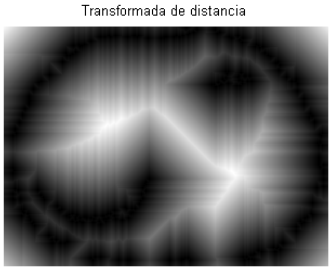
Calcula la transformada de la distancia según el valor especificado, con una métrica personalizada.

```
% filteredim = midistance2(im,const_T,metric_mask)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.
%  const_T : Constante T.
%  metric_mask : Métrica personalizada.

% PARÁMETROS DE SALIDA:
% filteredim : Transformada de distancia de la imagen.

dist2 = midistance2(i,1,1);
figure(9);imshow(i);title('Imagen original');
figure(10);imshow(dist2,[]);title('Transformada de distancia');
```



* miequalize

Realiza la ecualización del histograma.

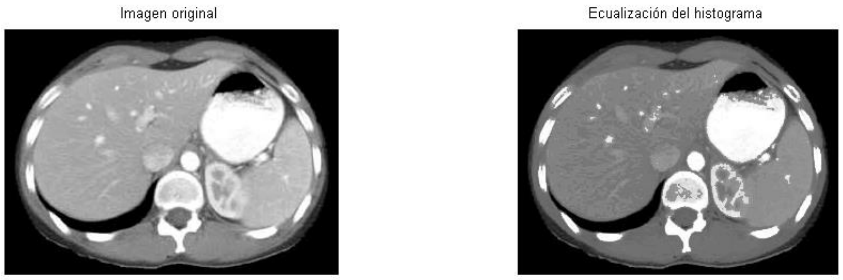
```
% filteredim = miequalize(im,nb_levels,value_min,value_max)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.
```

```
%  nb_levels : Número de niveles de histograma usado para la
ecualización.
%  value_min : Valor mínimo del píxel, considerado para el cálculo del
histograma.
%  value_max : Valor máximo del píxel, considerado para el cálculo del
histograma.

% PARÁMETROS DE SALIDA:
% filteredim : Imgen con el histograma ecualizado.

ecualizacion = miequalize(i,10,100,200);
figure(11);imshow(i);title('Imagen original');
figure(12);imshow(ecualizacion,[]);title('Ecualización del histograma');
```



* mierode

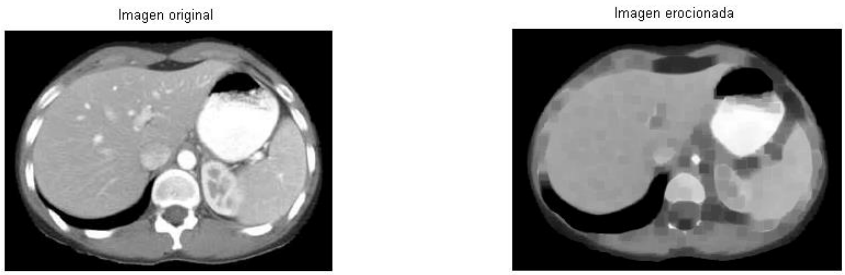
Realiza una erosión de la imagen con un elemento estructurante.

```
% filteredim = mierode(im,mask)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.
%  mask : Elemento estructurante.

% PARÁMETROS DE SALIDA:
% filteredim : Imagen erosionada.

erocion = mierode(i,8);
figure(13);imshow(i);title('Imagen original');
figure(14);imshow(erocion,[]);title('Imagen erocionada');
```



* midilate

Realiza una dilatación de la imagen con un elemento estructurante.

```
% filteredim = midilate(im,mask)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.
%  mask : Elemento estructurante.

% PARÁMETROS DE SALIDA:
% filteredim : Imagen dilatada.

dilatacion = midilate(i,5);
figure(15);imshow(i);title('Imagen original');
figure(16);imshow(dilatacion,[]);title('Imagen dilatada');
```

* migetgradient

Calcula una lista de imágenes que se corresponden con sus gradientes en los ejes X y Y..

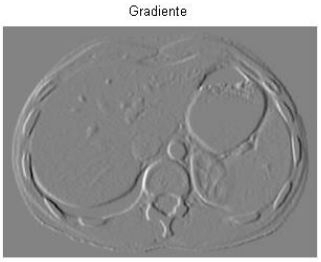
```
% filteredim = migetgradient(im,scheme)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.
%  scheme : Esquema numérico utilizado para el cálculo del gradiente. Los
valores de esta variable pueden ser:
%        -1 = Diferencias finitas hacia atrás.
%         0 = Diferencias finitas centradas.
%         1 = Diferencias finitas hacia adelante.
%         2 = Usando la máscara de Sobel.
%         3 = Utilizando una máscara invariante a la rotación (por
defecto).
%         4 = Usando el filtro recursivo de Deriche.

% PARÁMETROS DE SALIDA:
% filteredim : Gradiente de la imagen.

gradiente = migetgradient(i,2);
figure(17);imshow(i);title('Imagen original');
```

```
figure(18);imshow(gradiente,[]);title('Gradiente');
```



* mihaar

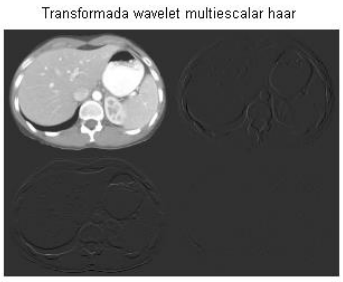
Calcula la transformada wavelet multiescalar Haar.

```
% filteredim = mihaar(im,axis,invert,nb_scales)

% PARÁMETROS DE ENTRADA
%  im : Imagen de entrada.
%  axis : Ejes considerados para la transformada.
%  invert : Inverso de la transformada directa.
%  nb_scales : Número de escalas utilizadas en la transformada.

% PARÁMETROS DE SALIDA:
% filteredim : Transformada wavelet multiescalar haar de la imagen.

haar=mihaar(i,'x',false,1);
haar1=mihaar(haar,'y',false,1);
figure(19);imshow(i);title('Imagen original');
figure(20);imshow(haar1,[]);title('Transformada wavelet multiescalar
haar');
```



* minormalize1

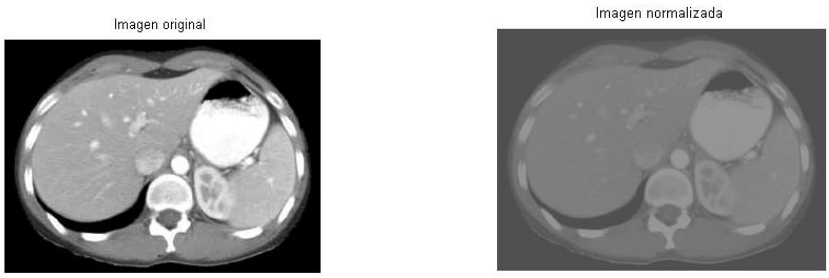
Se obtienen valores linealmente normalizados de un instante de la imagen, entre el valor mínimo y el valor máximo deseado.

```
% filteredim = minormalize1(im,value_min,value_max)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.
%  min : Valor mínimo deseado de la imagen resultante.
%  max : Valor máximo deseado de la imagen resultante.

% PARÁMETROS DE SALIDA:
% filteredim : Imagen normalizada.

norm1=minormalize1(i,80,150);
figure(21);imshow(i);title('Imagen original');
figure(22);imshow(uint8(norm1));title('Imagen normalizada');
```



* minormalize2

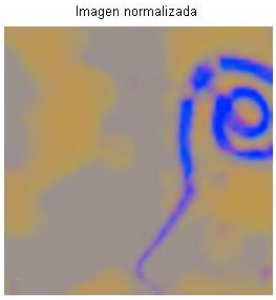
Normaliza múltiples valores de pixeles de un instante de la imagen. con respecto a su norma L2

```
% filteredim = minormalize2(im)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.

% PARÁMETROS DE SALIDA:
% filteredim : Imagen normalizada.

norm2 = minormalize2(i2);
figure(23);imshow(i2);title('Imagen original');
figure(24);imshow(norm2,[]);title('Imagen normalizada');
```



* miresize

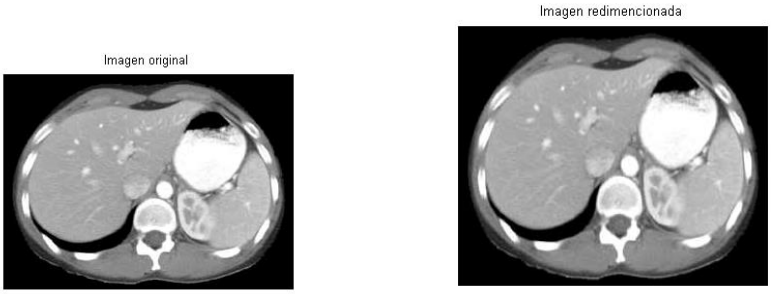
Realiza un redimecionamiento de la imagen.

```
% filteredim =
miresize(im,sizex,sizey,sizez,sizec,interpolation_type,border_conditions)

% PARÁMETROS DE ENTRADA::
%  im : Imagen de entrada.
%  sizex : Tamaño a lo largo del eje X.
%  sizey : Tamaño a lo largo del eje Y.
%  sizez : Tamaño a lo largo del eje Z.
%  sizec : Tamaño a lo largo del eje C.
%  interpolation_type : Método de interpolación.
%        -1 = Sin interpolación: cambio de tamaño de memoria en bruto
%         0 = Sin interpolación.
%         1 = Interpolación vecino más cercano.
%         2 = Interpolación promedio móvil.
%         3 = Interpolación linear.
%         4 = Interpolación de rejilla.
%         5 = Interpolación bicúbica.
%         6 = Interpolación lanczos.
%  border_conditions: Condición de borde.
%         centrado_x Establecer el tipo de centrado (solo si
interpolation_type=0).
%         centrado_y Establecer el tipo de centrado (solo si
interpolation_type=0).
%         centrado_z Establecer el tipo de centrado (solo si
interpolation_type=0).
%         centrado_c Establecer el tipo de centrado (solo si
interpolation_type=0).

% PARÁMETROS DE SALIDA:
% filteredim : Imagen redimencionada.

resize = miresize(i,300,250,100,2,2,0);
figure(25);imshow(i);title('Imagen original');
figure(26);imshow(resize,[]);title('Imagen redimencionada');
```



* miRGBtoHSI

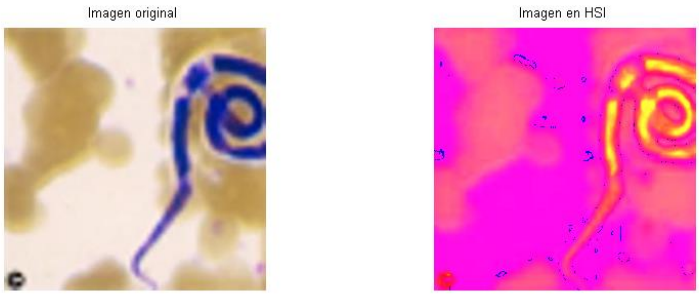
Convierte el color de los pixeles de (R,G,B) a (H,S,I).

```
% filteredim = miRGBtoHSI(im)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.

% PARÁMETROS DE SALIDA:
% filteredim : Imagen en HSI.

HSI = miRGBtoHSI(i2);
figure(27);imshow(i2);title('Imagen original');
figure(28);imshow(HSI,[]);title('Imagen en HSI');
```



* mirotate

Realiza un rotación de la imagen.

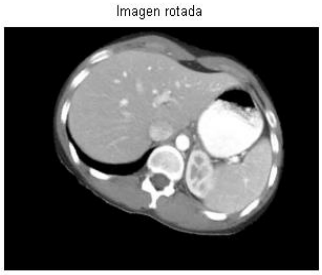
```
% filteredim = mirotate(im,angle,cx,cy,zoom,border_conditions,cond)

% PARÁMETROS DE ENTRADA::
%  im : Imagen de entrada.
%  angle : Angulo de rotación(en grados).
%  cx : Centro de rotación por las X.
%  cy : Centro de rotación por las Y.
```

```
%  zoom : ampliación.
%  border_conditions : Condición de borde.
%      0 = derichlet.
%      1 = neumann.
%      2 = cíclico.
%  cond : Tipo de rotación:
%      0 = valor cero en los bordes.
%      1 = píxel cercano.
%      2 = cíclico.

% PARÁMETROS DE SALIDA:
% filteredim : Imagen rotada.

rotar = mirotate(i,20,200,100,0.8,1,1);
figure(29);imshow(i);title('Imagen original');
figure(30);imshow(rotar,[]);title('Imagen rotada');
```



* mishift

Realiza un traslado de la imagen.

```
% filteredim = mishift(im,deltax,deltay,deltaz,deltac,border_conditions)

% PARÁMETROS DE ENTRADA:
%  im : Imagen de entrada.
%  deltax : Traslado a lo largo del eje-X.
%  deltay : Traslado a lo largo del eje-Y.
%  deltaz : Traslado a lo largo del eje-Z.
%  deltac : Traslado a lo largo del eje-C.
%  border_conditions : Condición de borde.
%      0 = derichlet.
%      1 = neumann.
%      2 = fourier.

% PARÁMETROS DE SALIDA:
% filteredim : Imagen trasladada.

shift = mishift(i,30,30,0,0,1);
figure(31);imshow(i);title('Imagen original');
figure(32);imshow(shift,[]);title('Imagen trasladada');
```

* mithreshold

Realiza una ubralización de la imagen.

```
% filteredim = mithreshold(im,value,soft_threshold,strict_threshold)

% PARÁMETROS DE ENTRADA:
% im : Imagen de entrada.
% value : Valor del humbral.
% soft_threshold : Indica si la umbralización es suave.
% strict_threshold : Indica si la umbralización es estricta.

% PARÁMETROS DE SALIDA:
% filteredim : Imagen umbralada.

i3=im2double(i);
umbralizacion=mithreshold(i3,0.3,true,false);
figure(33);imshow(i);title('Imagen original');
figure(34);imshow(umbralizacion,[]);title('Imagen umbralada');
```