Universidad
Politécnica
de Cartagena

industriales
etsii UPCT

Segmentación de imágenes médicas en software libre

Titulación: Ingeniero Industrial
Intensificación: Sistemas Eléctricos
Alumno/a: Francisco Javier Marín Marín
Director/a/s: José Abad López
José Damián Catalá Galindo

Cartagena, 05 de Septiembre de 2014

Resumen.

En este proyecto se ha desarrollado un programa de segmentación de imágenes médicas en Octave, un software libre de programación. El programa es capaz de realizar la búsqueda de diferentes órganos cuando se le proporciona una serie de imágenes de rayos X del tórax de un paciente. En este trabajo se limitó a tres órganos principales (pulmones, corazón y médula). No sólo se describe dicho programa, sino toda la investigación que se ha desarrollado para alcanzarlo. Por ejemplo, se tratan conceptos generales de procesamiento de imágenes digitales y de segmentación, así como todo el tratamiento que se ha llevado a cabo con las imágenes para lograr el método más adecuado de segmentación. Inicialmente el programa está sujeto a una serie de condiciones y parámetros experimentales, estas condiciones pueden ser ampliadas hasta conseguir un programa completamente automatizado. Destacar que es un programa gratuito, y se puede utilizar en cualquier dispositivo.

Abstract.

On this project it has been developed a medical image segmentation software on Octave, a free programming software. This software is capable of making the search of different organs when it is given a set of X-ray images of a patient's torax. On this work it was limited to three main organs (lumbs, heart and spine). Not only it is described this software, but also all the investigation developed to achieve it. For instance, general concepts of processing and segmentation of digital images are discussed, moreover the whole treatment that was made for achieving the most suitable segmentation method. Firstly, the software is subject to a set of experimental conditions and parameters, these conditions can be extended up to aim a complete automated software. Note that it's a free software, and it can be used in any device.

Índice de contenidos

Índice de imágenes

Índice de gráficos

Índice de esquemas

1. Introducción.

A lo largo de la presente memoria se detallará el proyecto relativo a la segmentación de imágenes médicas en software libre, concretamente en el programa Octave, similar al programa comercial Matlab.

La segmentación de imágenes consiste en dividir una imagen en tantas regiones como objetos y fondo contenga, de forma que se cumplan una serie de condiciones:

- La imagen completa quede dividida en regiones, sin huecos.
- Estas regiones deben ser completas, continuas.
- Los píxeles de cada una de estas regiones cumplirán una propiedad común a toda la región.
- Los píxeles de dos regiones adyacentes no pueden cumplir la misma propiedad.

Las imágenes médicas se guardan en formato DICOM (Digital Imaging and Communication in Medicine), por sus siglas en inglés, y las que en este caso se estudian son imágenes de rayos X de la zona del tórax.

Estas imágenes son en escala de grises, lo cual puede facilitar el proceso de segmentación, sin embargo contienen numerosas similitudes de intensidades y diversidad de formas, que no lo hacen tan sencillo.

La problemática que se plantea es la siguiente. Un médico tiene que tratar a lo largo del día con numerosas imágenes procedentes de tomografía axial computerizada (TAC), realizando en cada una de ellas un proceso de segmentación manual, con ayuda del ratón, indicando cuales son los pulmones, el corazón, etc.

La idea es que un software realice, al menos, ese trabajo preliminar de segmentación, dejando bien indicado con colores qué órganos estamos tratando. Es cierto que esto ya ha sido desarrollado por algunas compañías, pero los softwares que lo realizan requieren un desembolso de dinero importante. Por ello, lo que se quiere plantear aquí es un software gratuito y de código abierto, y se plantea ello como la principal ventaja.

Asimismo, este proyecto puede servir de base, una vez realizada la segmentación, para detectar anomalías en los pacientes, comparando con las imágenes que el programa guarde en una base de datos.

1.1. Contenido de los capítulos.

A lo largo de estos apartados se desarrollarán los aspectos más destacados del proyecto.

En el capítulo 2 se centrará la atención en los objetivos que se quieren llevar a cabo con el programa, el alacance del mismo y en los aspectos de motivación que sirvieron para elegir este proyecto.

El capítulo 3 trata el estado del arte del procesamiento de imágenes digitales y la segmentación de las mismas.

En el capítulo 4 se incluye el desarrollo cronológico del proyecto, explicando los pasos experimentales que se llevaron a cabo.

En el capítulo 5 se contempla la estructura final del programa, es el capítulo que directamente puede consultar el que quiera ponerse a trabajar con el programa.

El capítulo 6 está dedicado a mostrar los resultados obtenidos para la segmentación de imágenes del tórax de un paciente concreto. Notar que los resultados se pueden mejorar en programas posteriores, aquí se ha hecho más hincapié en el trabajo de investigación previo a la obtención de resultados.

En el capítulo 7 se reflejan las metas futuras que se pueden llevar a cabo en próximas investigaciones.

El penúltimo capítulo está dedicado a las conclusiones que se obtienen de la realización de este proyecto.

Finalmente, el capítulo 9 recoge las referencias bibliográficas utilizadas para el desarrollo de este programa.

También se ha querido incluir dos anexos, con los códigos del programa y de las funciones que se han realizado, para que sean de consulta y utilidad para cualquiera que lo desee.

2. Objetivos y motivación.

Este proyecto trata de cubrir la necesidad de segmentar las imágenes, que originalmente es tarea del médico, con la ayuda de un simple *'mouse'*, con un software libre capaz de realizar dicha tarea en un tiempo que no sea relativamente largo.

Es cierto que hay programas que ya realizan este tipo de segmentaciones, pero debido a su alto coste económico, y a veces computacional, se ha querido desarrollar este software abierto a cualquier persona, con un simple ordenador personal.

Las imágenes de estudio han sido proporcionadas por el jefe de Radioterapia del Hospital de Santa Lucía, en Cartagena, con el interés exclusivo de realizar este programa.

Se realizará la segmentación de los numerosos cortes del tórax, buscando tres órganos fundamentales: pulmones, corazón y médula. Esta es la base del proyecto, ampliable a cualquier órgano, pero que se ha limitado a estos debido a la restricción de tiempo para la realización de un proyecto fin de carrera como el que concierne.

La meta es conseguir un programa que de forma automática, segmente una serie de cortes transversales del tórax, indicando y contorneando los pulmones, el corazón y la médula, y es ahí donde se pone punto final al proyecto. Si bien, se puede seguir avanzando en el futuro, y detectar más órganos y anomalías.

En opinión personal del autor, el área de la Medicina es siempre un lugar interesante en el que colaborar, desde la posición de ingenieros, para ayudar todo lo posible a la mejora de la calidad de vida de las personas.

Es un proyecto diferente al típico que se puede realizar en Ingeniería Industrial, y esto supone un entretenido reto que se ha querido afrontar.

Se podría enmarcar en el campo de la ingeniería biomédica. A continuación se cita la información de la Universidad Politécnica de Valencia al respecto:

"La ingeniería biomédica es la disciplina que aplica los principios y métodos propios de la ingeniería a la solución de problemas en biología y medicina, y a la mejora de los métodos de prevención, diagnóstico, tratamiento y rehabilitación.

La ingeniería biomédica es una de las ramas de la ingeniería que ha experimentado un mayor crecimiento en los últimos años. Es un área en continua expansión donde se está produciendo una importante demanda de profesionales capaces de integrarse en equipos interdisciplinares, junto con profesionales de la salud, biólogos y médicos, para abordar nuevos retos en la mejora de la tecnología sanitaria. " (1)

Por estas razones, es de la opinión del autor considerarlo proyecto interesante, y siempre ha tenido curiosidad de entrar en estos terrenos de medicina, eso sí, desde el punto de vista del ingeniero, para servir de ayuda al médico.

3. Estado del Arte.

La segmentación de imágenes no es algo nuevo, lleva muchos años investigándose, y se cuenta con bastante información al respecto. Se pueden consultar las referencias (2), (3), (4), (5), (6), (7), (8), (9), y (10), de las que se han obtenido fragmentos.

Antes de la segmentación, se hablará sobre otros temas previos.

3.1. Contenidos de este apartado.

En este punto, se discutirá sobre el estado del Arte en cuanto al procesamiento de imágenes digitales y su posterior segmentación. Es un repaso de conceptos generales, que en muchos casos pueden ser conocidos, básicos o predecibles, pero que son necesarios para el tratamiento preciso de las imágenes.

En primer lugar, se hablará de las características principales de las imágenes digitales. Seguidamente, se tratarán los procesos de mejora de las imágenes, si bien realmente se

refiere a la mejora de las características que sean de interés, en detrimento de otras. Se hablará en profundidad de la mejora del contraste, por ser el método más visual, de forma que se intentará plasmar la segmentación que realizan los propios ojos en el ordenador. Las operaciones de punto permiten hacer variaciones de intensidad en partes de la imagen o en la imagen completa, píxel a píxel, sin tener en cuenta el valor de los de su alrededor. Se describirán una serie de funciones típicas para las operaciones de punto. Se hablará de las transformaciones locales, tomando en cuenta un vecindario 3 x 3, 5 x 5,… alrededor de un píxel.

Otro apartado importante corresponde a los algoritmos de detección de bordes, pues es la parte central de este proyecto. Se detallarán los algoritmos típicos, y las máscaras que se utilizan en los mismos para facilitar el tratamiento de las matrices. La eliminación del ruido y el filtrado de las imágenes son puntos útiles también a la hora de destacar características de las imágenes.

Finalmente, se tratará la segmentación de imágenes, la búsqueda de las regiones que contiene la imagen, separando los procesos que parten de un "punto-semilla", y los que parten de la imagen completa. Se describirán los modelos típicos de segmentación de imágenes que se encuentran en la bibliografía. Se cerrará con un apartado sobre cómo describir las regiones que se han obtenido.

La mayor parte de la información se ha obtenido de la referencia (11), una colección de vídeos interesantes de un profesor de la universidad de la India.

3.2. El procesamiento de imágenes digitales.

Una imagen continua, se subdivide en cierta cantidad de elementos, llamados píxeles, para poder ser almacenada en forma de matriz. Los tamaños habituales son 256 x 256, o 512 x 512. A cada píxel se le asocia un valor de intensidad. De la misma forma, los valores de intensidad, que son continuos, se tienen que dividir en una cantidad de niveles concreta. Generalmente se divide en 256 niveles, pero también puede segmentarse en 65536 niveles, dependiendo de la calidad que se quiera conservar.

Las aplicaciones en las que se emplea el procesamiento de imágenes son, hoy en día, muy numerosas, prácticamente en cualquier momento que se toma una fotografía, ya que ahora todas las cámaras son digitales. Se puede destacar, entre las aplicaciones, la transmisión y almacenamiento de imágenes, las aplicaciones médicas y militares, y las aplicaciones en la industria para el control de calidad.

En nuestro caso, se centrará la atención en las aplicaciones médicas, donde el procesamiento de imágenes se puede utilizar para comparar dos imágenes y detectar enfermedades. Se puede comparar un órgano con un defecto con un órgano sin defecto. O se pueden comparar dos imágenes del mismo paciente en épocas distintas, para ver si ha sufrido algún defecto.

3.3. Tratamientos de mejora de las imágenes.

Si la imagen tiene un contraste malo, o ruido, se pueden aplicar algunas técnicas para mejorar la calidad de la imagen, o al menos adecuarla a lo que se va buscando, resaltando algunas de sus características. Se nombran algunas de estas técnicas a continuación:

- Mejora del contraste (ecualizando el histograma, por ejemplo).
- Afilamiento de los bordes.
- Reducción del ruido.
- Filtrado.

Se va a detallar en profundidad el primer punto, la mejora del contraste de la imagen. Se debe especificar qué cualidad de la imagen se está mejorando, porque la mejora no será de la imagen completa.

3.3.1. Mejora del contraste.

Se puede realizar una mejora del contraste en el dominio espacial. De forma que si se tiene inicialmente una función $f(x,y)$ que contiene las intensidades en todos los puntos (x,y), y se quiere obtener una $g(x,y)$, que contendrá las intensidades mejoradas.

$$g(x,y) = T[f(x,y), p(x,y), x, y]$$

Donde (x,y) son las coordenadas del píxel, $f(x,y)$, la intensidad de dicho píxel, $p(x,y)$ un vecindario $(3\times 3, 5\times 5, \dots)$ alrededor del punto (x,y).

Si la función T sólo depende de las intensidades $f(x,y)$, entonces las transformaciones se llaman "operaciones de punto" u "operaciones de memoria cero", ya que no se necesita almacenar la información de los puntos de alrededor.

$$g(x,y) = T[f(x,y)]$$

Por ejemplo, si se llama u a la función de las intensidades y $v = f(u)$ a la función de las nuevas intensidades, definidas ambas en un intervalo $[0, L] = [0, 255]$, se tiene:

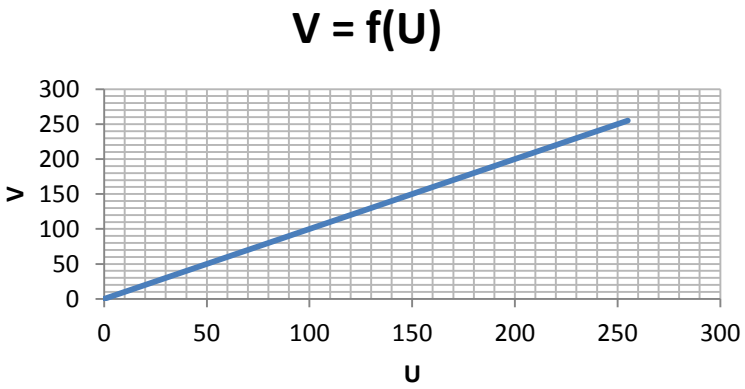


Gráfico 1. Operación de punto

Con esta transformación, lineal de pendiente 1, las intensidades conservarían el mismo valor, como si no se hubiese aplicado nada.

Otras funciones pueden tener tres regiones con diferente pendiente:

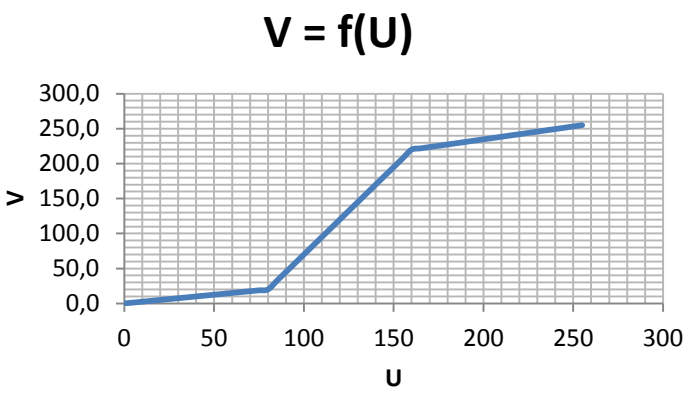


Gráfico 2. Función típica de operación de punto

Las pendientes de las tres zonas serán $\alpha, \beta,$ y γ, respectivamente. La función que define esta transformación es la siguiente:

$$V = f(U) = \begin{cases} \alpha U, & si\ U \leq A \\ A\alpha + \beta(U - A), & si\ A < U < B \\ A\alpha + \beta(B - A) + \gamma(U - B), & si\ U \geq B \end{cases}$$

Donde A y B son los puntos de cambio de pendiente.

A continuación, se muestran algunos casos típicos.

- **Función de recorte** ($\alpha = \gamma = 0$).

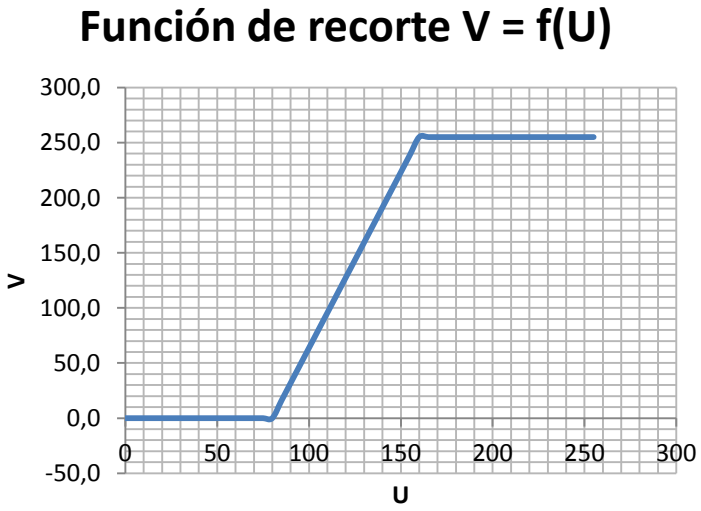


Gráfico 3. Función de recorte

- **Función de frontera.**

Si se acercan cada vez más A y B, se obtiene una función de frontera.

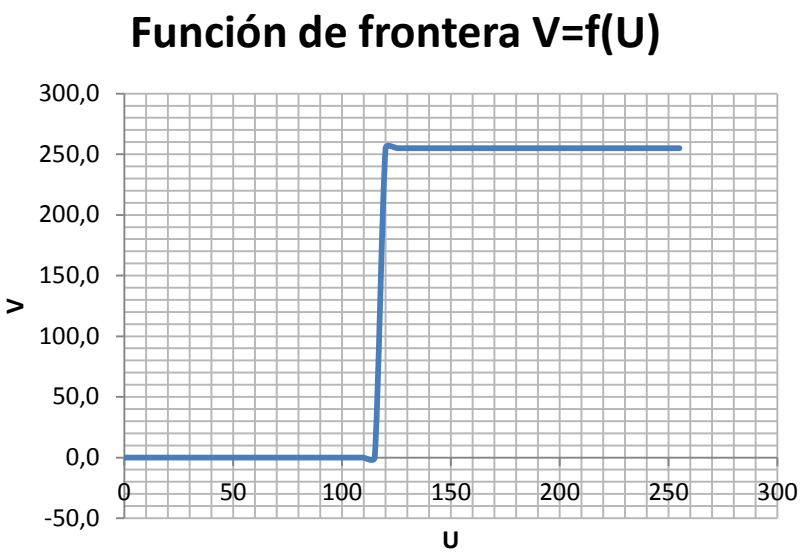


Gráfico 4. Función de frontera

- **Función inversa.**

También se puede obtener la función inversa:

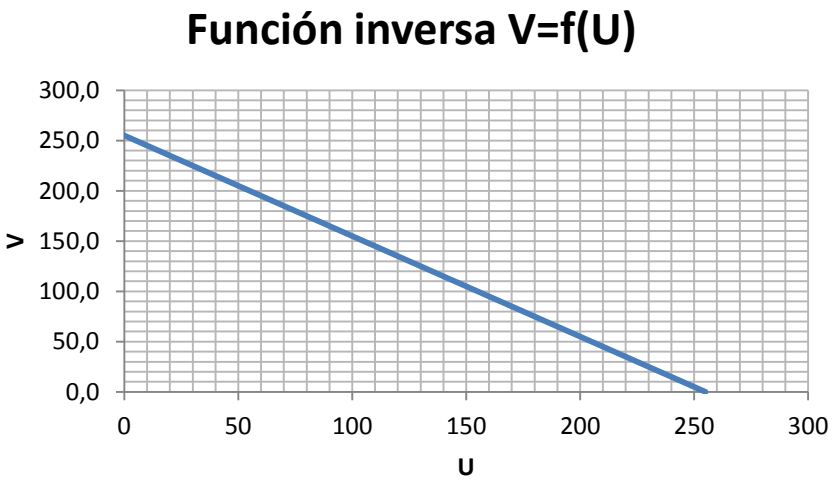


Gráfico 5. Función inversa

- **Función de resalto.**

Una función de resalto.

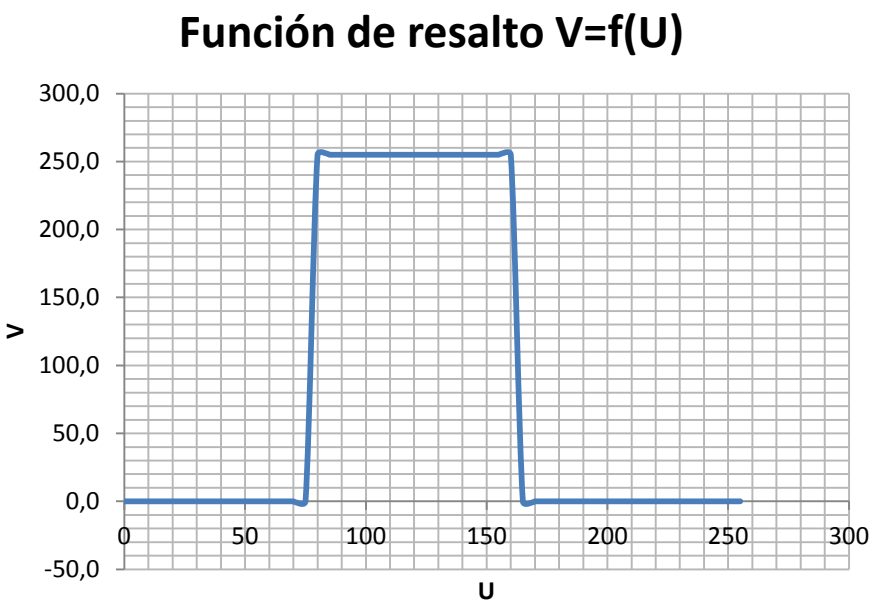


Gráfico 6. Función de resalto

- **Función de resalto manteniendo el fondo.**

Se puede mantener el fondo de la imagen.

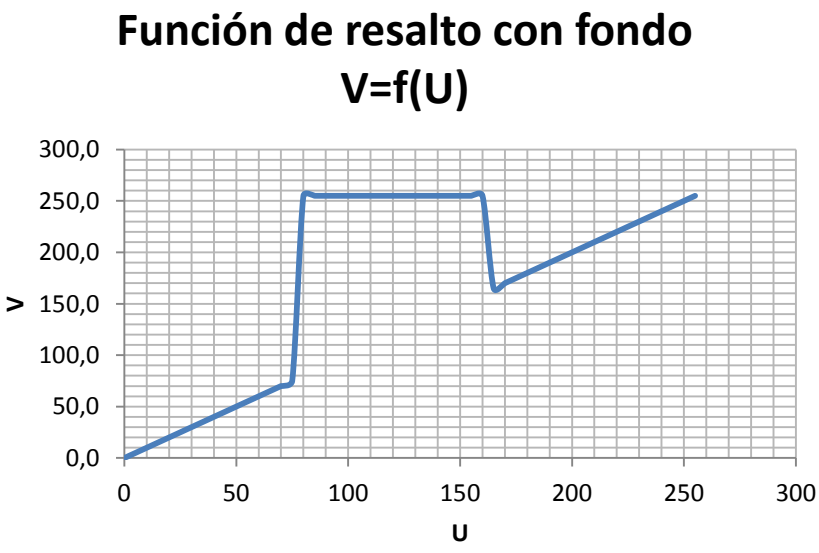


Gráfico 7. Función de resalto con fondo

Estas son las típicas funciones que se pueden aplicar para mejorar ciertas características de las imágenes.

- **Función de transformación logarítmica.**

También se pueden aplicar transformaciones no lineales, como la transformación logarítmica.

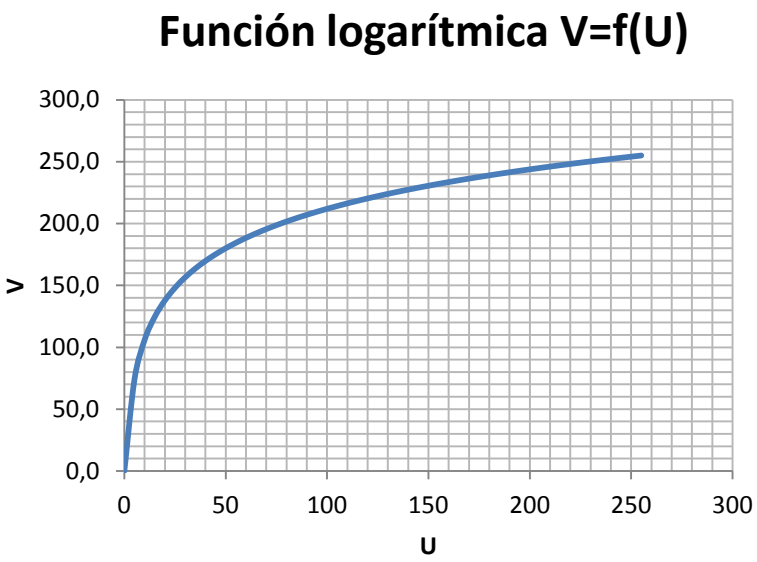


Gráfico 8. Función logarítmica

$$V = c * \ln(U)\,, donde\ c\ es\ la\ constante\ de\ escala$$

- **Transformaciones locales.**

Si en lugar de tomar una función T que dependa sólo del punto, se toma un vecindario, 2x2, 3x3, etc., se puede obtener otro tipo de transformaciones:

$$v(m,n) = T\begin{bmatrix} u(m-1,n-1) & u(m,n-1) & u(m+1,n-1) \\ u(m-1,n) & u(m,n) & u(m+1,n) \\ u(m-1,n+1) & u(m,n+1) & u(m+1,n+1) \end{bmatrix}$$

Se puede aplicar una media espacial, o una suma ponderada de los píxeles alrededor del punto.

Cuando el píxel está en una esquina, habrá valores de la matriz que no se pueden alcanzar. En ese caso, se suele considerar que tienen valor 0.

Si se incrementa el tamaño de la ventana, se pierden los bordes de los objetos, los cambios repentinos de intensidad. Pero si la imagen que se trata tiene mucho ruido, puede ser efectivo el tratamiento. Dependerá de cada imagen, no es un método de mejora para todas las imágenes.

Otro procedimiento, no lineal en este caso, puede ser aplicar la mediana de los puntos de la matriz.

3.3.2. Operadores de detección de bordes.

Los operadores son máscaras que se aplican a la matriz de puntos, de forma que se transforman en imágenes donde las zonas de cambio de intensidad (los bordes) se han resaltado, mientras que las zonas de intensidad constante (fondo e interior de los objetos) han disminuido su valor de intensidad. Con lo cual, aplicando una operación de punto con un valor umbral, finalmente resulta una imagen binaria, 0 (fondo) y 1 (para los bordes).

A continuación, se explican algunos de esos operadores.

- **Sobel.**

El principal operador que se utiliza para detectar bordes es el operador 'Sobel'. Consiste en aplicar la máscara siguiente:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Para detectar los gradientes en dirección horizontal.

Notar que los puntos de la máscara suman cero. La aplicación de la máscara a una matriz 512 x 512 es muy sencilla, consiste en tomar una matriz 3x3 centrada en cada punto, y multiplicarla elemento a elemento con esta máscara. A continuación se suman todos los valores obtenidos, y ese es el valor que se coloca en el espacio reservado al píxel. Cuando el valor sea bajo, significará que tanto a la izquierda como a la derecha, los valores de intensidad son prácticamente iguales, con lo cual se está dentro de un objeto. Si un valor es alto, significa que hay una variación de intensidad, y por tanto, se está en un borde.

Si se quiere detectar el gradiente en dirección vertical, utilizamos la máscara:

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Este operador es ampliamente utilizado porque se ha demostrado que es resistente a posibles ruidos que contenga la imagen.

- **Prewitts.**

Otro operador, muy similar, es el 'Prewitts', que utiliza las siguientes máscaras, para dirección horizontal y vertical, respectivamente:

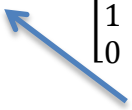
$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

El Prewitt no enfatiza sólo los gradientes horizontales y verticales, sino también los diagonales.

Asimismo, se pueden utilizar estos operadores en cualquiera de las 8 direcciones, simplemente rotando los elementos de la matriz en torno al centro.

Por ejemplo, si se quieren detectar los gradientes en dirección noroeste:



$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

Si se aplican las máscaras en las 8 direcciones y luego se suman todas esas matrices, habría que quedarse con los valores más altos. Se aplicaría cierto valor umbral para transformar en $[0, 1]$.

- **Filtro de alta frecuencia.**

Sólo deja pasar la imagen de alta frecuencia, mediante la siguiente máscara:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- **Operador Kirsch.**

$$\begin{bmatrix} 5 & 5 & 5 \\ -3 & 0 & -3 \\ -3 & -3 & -3 \end{bmatrix}$$

- **Operador Laplaciano.**

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Antes de aplicarlo, se debe eliminar el ruido de la imagen.

3.3.3. Eliminación del ruido.

Dependiendo del origen del ruido de la imagen, se utilizará un método u otro. Por ejemplo:

- **Ruido de "sal y pimienta".**

Se llama así porque son píxeles aleatorios con valor 0 (negros) o el máximo de la imagen (blancos) repartidos por toda la imagen, como consecuencia de defectos en la cámara fotográfica, o en el entorno al tomar la imagen. El efecto es similar al que se obtendría si se esparciera sal y pimienta a la fotografía en papel.

Un buen método de eliminar este ruido es tomar una media en cada píxel, de los valores de su alrededor, realizando una operación de transformación local.

El problema se puede encontrar en imágenes que tienen cambios suaves de intensidad, pues estos se perderán. Por ello, se recuerda que no todos los métodos son adecuados para todas las imágenes.

3.3.4. Filtrado de imágenes.

Se pueden eliminar partes de la imagen, eliminar zonas de intensidad por encima o por debajo de cierto valor, y más transformaciones de punto, locales o globales, como las que se han detallado.

Una vez la imagen ha sido tratada, y se han destacado las cualidades que son de interés, se pasa a la segmentación de imágenes como tal.

3.4. Segmentación de imágenes.

La segmentación de una imagen se define como: la extracción o división de forma semántica, es decir, en formas o zonas que tienen relación directa con un objeto o forma real. Los algoritmos de segmentación intentan buscar relaciones de semejanza u homogeneidad (regiones) o divergencia o heterogeneidad (contornos) dentro de las imágenes para determinar o clasificar las partes que las componen.

3.4.1. Modelos de segmentación de imágenes.

Hay bastantes modelos, pero se van a tomar los reflejados en la referencia. Simplemente, se dan unas breves pinceladas de las características destacables de los modelos, sin entrar en detalles.

- **Segmentación basada en regiones.**
  - Crecimiento de regiones por agregación de píxeles.

En este tipo de segmentación se trata de coger un *"punto-semilla"*, y observar los puntos de su alrededor. Si tienen características similares, se unen a la región. Si no, forman una región independiente. Sucesivamente se va pasando a los puntos de alrededor.

Se tiene que establecer un criterio de pertenencia a la región.

Por ejemplo, se tiene la siguiente matriz de píxeles. El criterio que se establece es que la diferencia de intensidad no sea mayor que 2 entre un píxel y los de su alrededor.

$$\begin{bmatrix} 1 & 0 & 1 & 6 & 6 & 8 \\ 2 & 0 & 2 & 7 & 7 & 6 \\ 1 & 2 & \mathbf{1} & 8 & 6 & 5 \\ 2 & 1 & 7 & 7 & 7 & 6 \\ 1 & 2 & 6 & 6 & 7 & 7 \\ 1 & 1 & 1 & 6 & 7 & 8 \end{bmatrix}$$

El "punto semilla" elegido es el que está marcado en amarillo. Se observa su entorno:

$$\begin{matrix} \mathbf{0} & \mathbf{2} & 7 \\ \mathbf{2} & \mathbf{1} & 8 \\ \mathbf{1} & 7 & 7 \end{matrix}$$

Los puntos cuya intensidad es similar a 1, se adhieren a la región de amarillo. Así, se continúa con los píxeles de alrededor.

$$\begin{matrix} 1 & 0 & 1 & 6 \\ 2 & 0 & 2 & 7 \\ 1 & 2 & 1 & 8 \\ 2 & 1 & 7 & 7 \\ 1 & 2 & 6 & 6 \end{matrix}$$

Una vez se completa la región, se toma otro *"punto-semilla"* fuera de ella, y se repite el proceso. Así se sigue mientras se tengan píxeles que no estén adjuntos a ninguna región. Se recuerda que un píxel no puede pertenecer a dos regiones, pues se estaría incumpliendo uno de los principios de la segmentación de imágenes, que los píxeles de dos regiones adyacentes no pueden satisfacer una misma propiedad.

$$\begin{bmatrix} 1 & 0 & 1 & 6 & 6 & 8 \\ 2 & 0 & 2 & 7 & 7 & 6 \\ 1 & 2 & 1 & 8 & 6 & 5 \\ 2 & 1 & 7 & 7 & 7 & 6 \\ 1 & 2 & 6 & 6 & 7 & 7 \\ 1 & 1 & 1 & 6 & 7 & 8 \end{bmatrix}$$

También puede ser que el criterio que se necesite aplicar no esté basado en la intensidad, sino, por ejemplo, en el reconocimiento de patrones.

$$\begin{bmatrix} \Delta & \blacksquare & \blacktriangle & \blacktriangle \\ \Delta & \blacksquare & \blacksquare & \blacktriangle \\ \Delta & \blacksquare & \blacksquare & \blacktriangle \\ \Delta & \blacksquare & \blacktriangle & \blacktriangle \end{bmatrix}$$

En este caso, las regiones se buscan por similitud con los patrones de formas.

La elección del *"punto-semilla"* y del criterio de pertenencia a la región es una tarea difícil, que en muchos casos se resuelve de forma experimental. Si bien, se puede ayudar uno del histograma para detectar la presencia de picos de intensidad.

- Separación y unión de regiones.

Este es otro método ampliamente utilizado. Se parte de una región R, que representa la imagen completa. Se observa si se cumple la propiedad que se ha establecido dentro de toda la región.

$$\begin{bmatrix} 1 & 0 & 1 & 2 & 7 & 7 & 7 & 7 \\ 2 & 1 & 2 & 2 & 1 & 6 & 8 & 6 \\ 1 & 1 & 1 & 0 & 1 & 6 & 6 & 8 \\ 2 & 2 & 2 & 0 & 2 & 7 & 7 & 6 \\ 1 & 1 & 1 & 2 & 1 & 8 & 6 & 7 \\ 2 & 1 & 2 & 1 & 7 & 7 & 7 & 6 \\ 1 & 2 & 1 & 2 & 6 & 6 & 7 & 7 \\ 1 & 2 & 1 & 1 & 1 & 6 & 7 & 8 \end{bmatrix}$$

En el caso de no cumplirse, que será lo habitual, se divide en 4 regiones iguales: R_1, R_2, R_3 y R_4. Se estudia en cada una de estas regiones si se cumple la propiedad citada en el entorno completo.

1	0	1	2	7	7	7	7
2	1	2	2	1	6	8	6
1	1	1	0	1	6	6	8
2	2	2	0	2	7	7	6
1	1	1	2	1	8	6	7
2	1	2	1	7	7	7	6
1	2	1	2	6	6	7	7
1	2	1	1	1	6	7	8

En este caso, la región superior izquierda, y la región inferior izquierda, cumplen la propiedad de no poseer píxeles internos con diferencias de intensidad mayores que 2, con lo cual se dejan como están, anotando su pertenencia. Las otras 2 regiones, de la parte derecha, se tienen que volver a dividir en 4 partes, y repetir el mismo proceso.

1	0	1	2	7	7	7	7
2	1	2	2	1	6	8	6
1	1	1	0	1	6	6	8
2	2	2	0	2	7	7	6
1	1	1	2	1	8	6	7
2	1	2	1	7	7	7	6
1	2	1	2	6	6	7	7
1	2	1	1	1	6	7	8

Se observan las regiones que están completas ya.

1	0	1	2	7	7	7	7
2	1	2	2	1	6	8	6
1	1	1	0	1	6	6	8
2	2	2	0	2	7	7	6
1	1	1	2	1	8	6	7
2	1	2	1	7	7	7	6
1	2	1	2	6	6	7	7
1	2	1	1	1	6	7	8

Y finalmente se obtiene la matriz segmentada en regiones, en este caso contiene sólo dos regiones.

$$\begin{bmatrix} 1 & 0 & 1 & 2 & 7 & 7 & 7 & 7 \\ 2 & 1 & 2 & 2 & 1 & 6 & 8 & 6 \\ 1 & 1 & 1 & 0 & 1 & 6 & 6 & 8 \\ 2 & 2 & 2 & 0 & 2 & 7 & 7 & 6 \\ 1 & 1 & 1 & 2 & 1 & 8 & 6 & 7 \\ 2 & 1 & 2 & 1 & 7 & 7 & 7 & 6 \\ 1 & 2 & 1 & 2 & 6 & 6 & 7 & 7 \\ 1 & 2 & 1 & 1 & 1 & 6 & 7 & 8 \end{bmatrix}$$

Este sistema computacionalmente es muy lento, pues se deben mantener todas las regiones en cada paso.

- **Modelo de distribución de puntos (Point Model Distribution).**

También es conocido como el modelo de contornos activos o *'snakes'*. Trata de parametrizar los objetos o regiones de interés con un conjunto de puntos que conforman un *spline* de continuidad controlada, capaz de deformarse y adaptarse para encontrar el contorno. El concepto físico en el que se basa es en la minimización de energía. La energía que controla el contorno está basada en tres factores:

- ❖ Energía de la imagen completa.
- ❖ Factores internos.
- ❖ Factores externos.

- **Etiquetado.**

Este sistema tiene dos pasos. Una primera etapa en la que se asocia una etiqueta a cada píxel, y una segunda etapa en la que se juntan los píxeles con etiquetas iguales.

- **Lógica difusa.**

Es una herramienta matemática muy potente que trata de simular la capacidad humana de extraer conclusiones con información incompleta o imprecisa.

- **Modelo Geométrico.**

El objetivo es parametrizar las regiones de interés con curvas asociadas a frentes de propagación. Estos frentes son capaces de evolucionar y segmentar el contorno del objeto u objetos de interés dentro de la imagen. El concepto físico subyacente es la evolución de curvas o frentes de propagación mediante la ecuación del calor.

- **Modelo Geodésico.**

Extiende el Modelo Geométrico al Modelo de Distribución de Puntos, para que al combinar ambos, resulte un modelo más estable y preciso.

- **Modelo de Regiones Activas Geodésicas.**

Tienen probada eficacia en segmentación de imágenes con independencia de la posición inicial, al contrario que los métodos anteriormente descritos.

- **Modelo de Múltiples Regiones Activas Geodésicas.**

Es una extensión del modelo anterior, solucionando el problema de los órganos compuestos por más de una región, y dotándolos de nuevas características.

La principal limitación de estos métodos de segmentación propuestos es la forma de determinar los parámetros adecuados para cada tipo y modalidad de imagen.

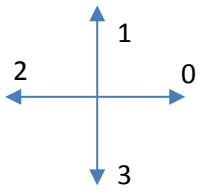
3.4.2. Representación de regiones.

Una vez la imagen está dividida en regiones, toca expresar dichas regiones. Se puede hacer por dos formas diferentes, según sus puntos externos o según sus puntos internos. Una condición indispensable es que la definición de las regiones no debe cambiar aunque se someta a la imagen a una transformación, tal como una traslación o rotación. Se detallan seguidamente los dos tipos de clasificación.

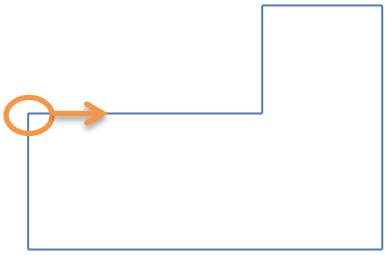
- **Características externas (fronteras de los objetos).**

  - Códigos de cadena.

Se puede utilizar un sencillo código de cadena de 4 direcciones, en el caso de que las fronteras de los objetos sólo estén compuestas por líneas horizontales y verticales.



Se empieza por un punto del borde, y se va indicando la dirección en que se mueve, con uno de esos 4 números.



El código de la cadena, empezando por el píxel marcado, y con movimiento en la dirección indicada, sería el siguiente: 010321.

¿Pero qué ocurre si se empieza por otro punto? El código cambiaría, aunque los giros serían los mismos. Por ejemplo: 032101, también sería un código de la cadena.

Para hacerlo invariable al punto de comienzo, pues, se debe establecer otro criterio, que será implantar el código que dé el número más pequeño. En este caso sería 010321, de entre las 6 posibilidades que se tienen.

Aun así, el código no es invariante a la rotación alrededor de un punto, con lo cual conviene aplicar otro código:

  - Cambios unidireccionales del código de cadena.

Se trata de anotar en cada paso, el número de cambios de dirección que ha habido, en el sentido de las agujas del reloj. Por ejemplo, para pasar de la dirección 0 a la 3 hay 1

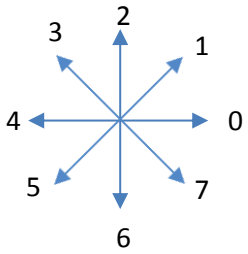
cambio de dirección, mientras que para pasar de la dirección 3 a la dirección 0 hay 3 cambios de dirección.

Y para hacerlo invariante al punto de comienzo, también se debe escoger el orden de forma que salga el número más pequeño.

Así, el código de la cadena que anteriormente se veía sería: 311111. Pero se tiene que ordenar de forma que se produzca el menor número, con lo cual será: 111113.

- Código de cadena de 8 direcciones.

Es el código más extendido, pues los bordes suelen estar compuestos también por líneas diagonales.

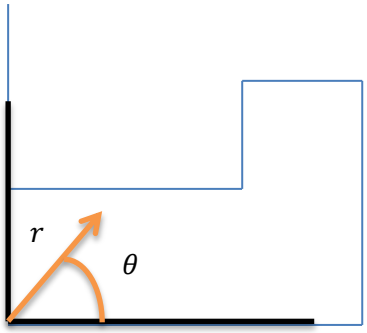


- Aproximación polígonal.

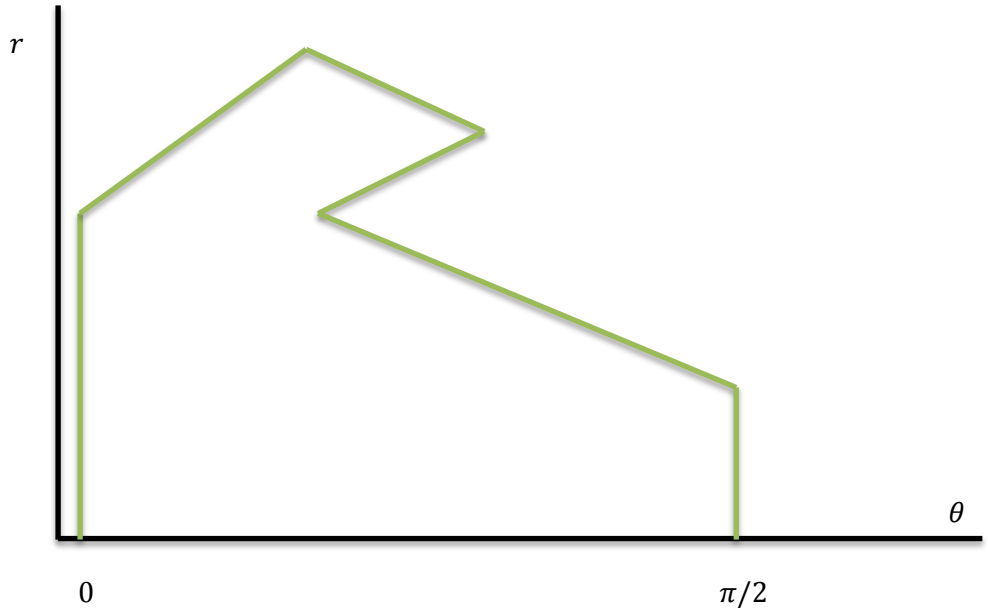
Conseguir el mínimo polinomio que describa la curva, cumpliendo unos criterios de proximidad respecto a la curva real.

- Sistema de coordenadas polares.

Se toma un sistema de referencia con coordenadas $[r, \theta]$, y comenzando por cualquier punto, se va representando el gráfico.



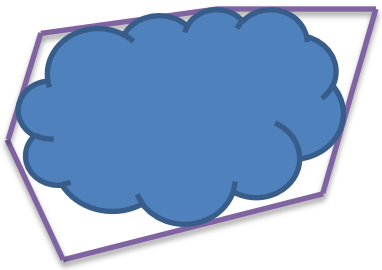
Con este sistema se representa, en función del ángulo que se va girando, el valor del radio.



Esta representación varía dependiendo del punto inicial. Por ello se tiene que tomar algún criterio. Se elegirá empezar por el punto más lejano del origen, y siempre girar en el sentido contrario a las agujas del reloj.
Para conseguir que sea invariante al escalado, se debe normalizar, pasando de una escala $[0, r]$ a una escala $[0, 1]$.

- Envolvente convexa.

Consiste en envolver a la curva con la figura convexa más pequeña posible.



Se anotan los puntos de inflexión.

- Esqueleto de una región.

Para cada punto del interior de la figura, se busca su punto de la frontera más cercano. Si hay dos puntos de la frontera a la misma distancia de él, dicho punto pertenece al esqueleto de la región. Por ejemplo:

Para la figura marcada en color naranja, el esqueleto está compuesto por las líneas de color rojo. También se conoce como el "eje medio". El algoritmo de 'thinning' (adelgazamiento) está ampliamente relacionado con mantener el esqueleto de las regiones. Se puede encontrar más detalles en la referencia [15].

- **Características internas (píxeles que componen la región).**

Son menos utilizadas las representaciones de regiones mediante puntos del interior de la región, por ello no se va a entrar en este tema.

3.4.3. Descriptores de la frontera.

Una vez se tiene la región descrita, se pueden obtener algunas características de la misma, como son:

a) Longitud: número de píxeles a lo largo del contorno.
b) Diámetro: máxima distancia entre dos puntos pertenecientes al contorno.
c) Curvatura: diferencia de pendientes entre los segmentos adyacentes en un punto.

3.4.4. Cierre de contornos: Algoritmo de Thinning.

El algoritmo de Thinning, desde un punto de vista descriptivo, consiste en realizar sucesivos "barridos" por todos los píxeles de la imagen con los elementos del conjunto B, y comparar aquellos píxeles que pertenezcan a un contorno (1's binarios) con sus píxeles vecinos, proceso que determinará la eliminación o no de dicho píxel, y después se rota la "Matriz de Thinning". Es importante no alterar el orden del proceso para obtener los objetivos que se pretenden. Si en lugar de analizar primero todos los píxeles, es decir, hacer un 'barrido" de toda la imagen, y a continuación rotar la "matriz de thinning" (cambiar Bi por Bi+1), se rota la matriz después de aplicarla sobre cada píxel, al mismo tiempo que los contornos se van adelgazando se producen roturas de los mismos.

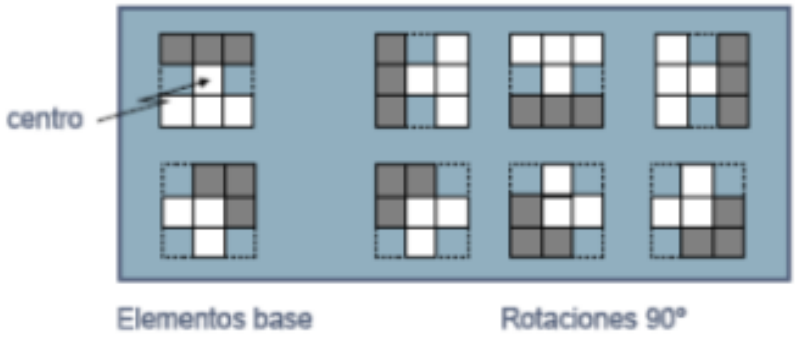


Imagen 1. Algoritmo de Thinning

Entonces, la idea del adelgazamiento de contornos es ir eliminando los píxeles del contorno que sobran, pero sin romperlos.

Observaciones:

1).-El thinning depende de la forma del conjunto y de la forma del elemento estructurante.

2).-La elección del elemento inicial y el orden de la secuencia de thinnings influyen en el resultado final.

3).-Estas transformaciones no son muy robustas. Esta falta de robustez se puede ver en la aparición de un gran número de ramas que dependen fuertemente de la elección del elemento estructurante.

A continuación se muestra un ejemplo de los resultados obtenido en cada paso del algoritmo. Se puede observar cómo se han adelgazado los contornos sin producirse rotura alguna.

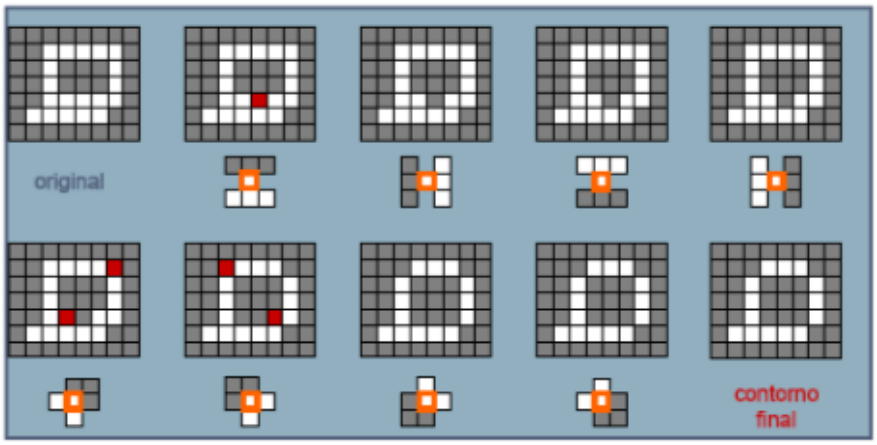


Imagen 2. Aplicación del algoritmo de Thinning

Se puede consultar más información al respecto en la referencia (12).

4. Desarrollo del proyecto.

En este apartado se va a detallar, tan cronológicamente como sea posible, lo que ha sido el desarrollo de este proyecto.

4.1. Contenidos de este apartado.

En primer lugar, se hablará del software libre que se decidió utilizar, descargando su versión de la web de desarrolladores. Seguidamente se contará el proceso de lectura de imágenes que se necesitó llevar a cabo con el programa. Los tipos de imágenes que se encuentran el programa. La primera idea de contorneo inicial, con ayuda de funciones ya desarrolladas. El cambio de contraste en la imagen como forma de mejora de la segmentación. Una vez se tuvo el contorno, se describe el proceso de coloreado y adición a la imagen original. El proceso completo con una biblioteca de imágenes y el empleo de información útil de anteriores imágenes contorneadas. Se entra en el detalle del proceso de estudio, empezando por una imagen en particular, que sirve de base, explicando todos los procesos que se le fueron aplicando. Un contorneo inicial, una mejora del contraste, estudios experimentales ensayo-error, etc. Para mejorar la claridad

del programa y comodidad, se desarrollaron una serie de funciones que también se detallan, y una mejora del proceso que se tuvo que realizar porque el método inicial no era lo suficientemente preciso cuando se aplicaba a una colección de imágenes completa.

4.2. Software empleado.

En primer lugar, se presenta un entorno del programa utilizado.

Se trata del GNU Octave, en este caso utilizamos la versión 3.6.4 para Windows.

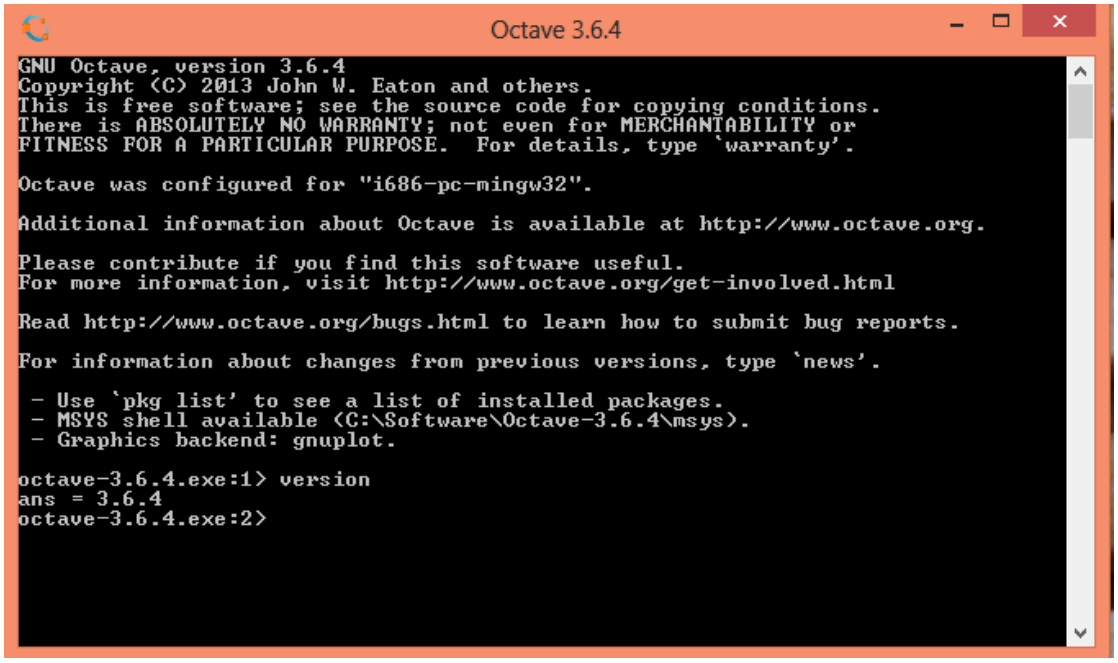


Imagen 3. Entorno de trabajo de Octave

Se observa que el entorno de trabajo es muy rudimentario, el de MS-DOS. Los desarrolladores de Octave prevén lanzar una plataforma similar a la de Matlab, en las versiones 4.0x, con lo cual el trabajo posterior será más fácil.

El programa se puede descargar de la siguiente dirección web:

http://www.gnu.org/software/octave/download.html (13)

La dirección web principal del programa es:

http://octave.sourceforge.net/ (14)

Cuenta con una serie de paquetes de funciones, que ayudarán al autor a la realización de su proyecto, que se pueden descargar en el siguiente enlace:

http://octave.sourceforge.net/packages.php (15)

En el caso de instalar el programa Octave en Windows 8, la ventana de comandos no funcionará correctamente de primeras. Pero tiene fácil solución, se debe hacer clic con

el botón derecho sobre el icono del acceso directo del escritorio de Octave. Pulsando en *"propiedades"* del menú desplegable, se añade la siguiente línea al apartado *"destino":*

C:\Software\Octave-3.6.4\bin\octave-3.6.4.exe **-i --line-editing**

Cuando se comienza a trabajar con Octave, se observa que las rutas de acceso por defecto son las siguientes:

C:\Software\Octave 3.6.4\share\octave\3.6.4\m para guardar las funciones.

C: \Software\Octave 3.6.4 para cargar las imágenes.

C:\Software\Octave-3.6.4\share\octave\packages para instalar los paquetes de funciones.

Se puede cambiar el directorio principal con el comando '*cd*'. Y se pueden añadir directorios de búsqueda de funciones con los comandos *'addpath(dir)'*. Para saber el directorio en el que se encuentra, se puede teclear el comando *'ls'*.

Para el desarrollo de los programas, se ha empleado un editor de texto comercial, el '*Notepad++*', que permite escribir en diversos lenguajes de programación de forma cómoda y sencilla.

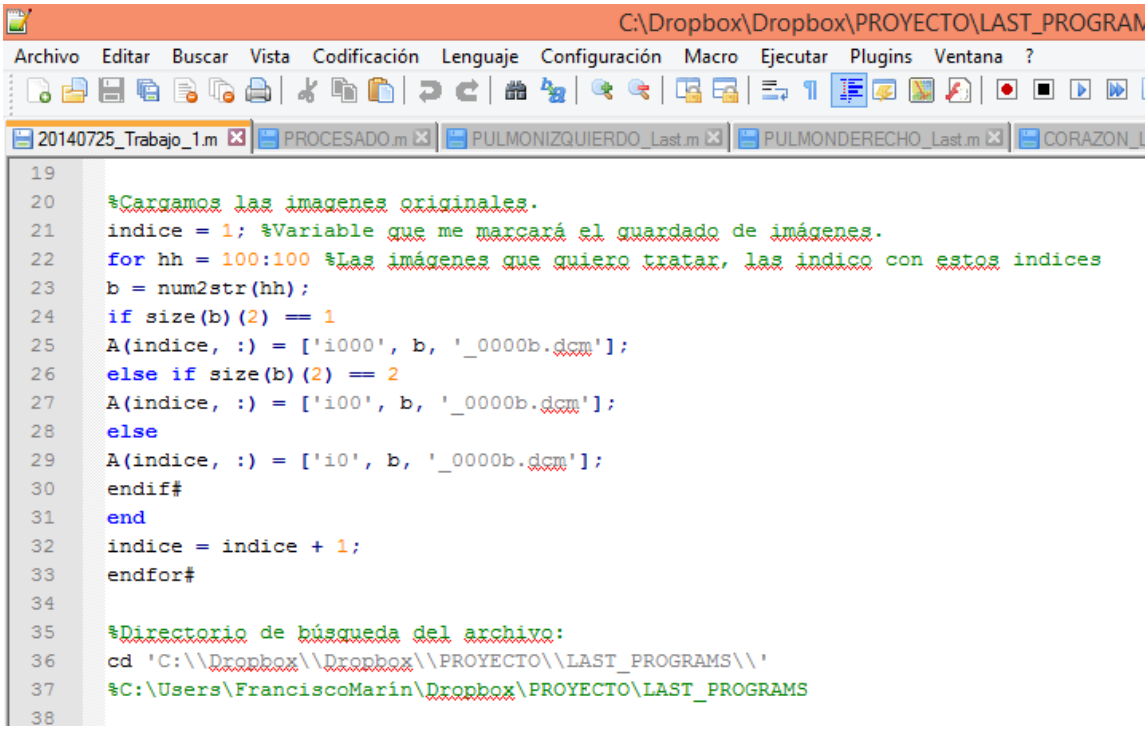


Imagen 4. Entorno de trabajo del editor de texto Notepad++

4.3. Lectura de imágenes.

Las imágenes con las se trabajó en este proyecto son de formato DICOM (*Digital Imaging and Communication in Medicine*). Este formato es el estándar reconocido y

utilizado en el ámbito de la medicina. Incluye los datos del paciente dentro del mismo archivo, los cuales será preciso conservar.

Para el procesado de estas imágenes, los desarrolladores de Octave han creado un paquete de funciones DICOM, que se puede descargar en la referencia (15).

Empleando las funciones de lectura de imágenes DICOM, se carga la imagen, y ya se puede tratar como si de una imagen normal se tratase.

Se trabaja con una serie de 200 imágenes de TAC. Para el desarrollo del programa primero se trabajó con una sola imagen, de la zona central del tórax, para posteriormente testear el programa con todas las imágenes.

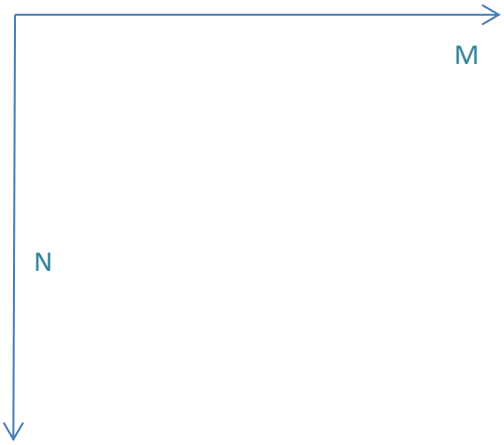
Para realizar el procesamiento de la imagen, será necesario también descargar el paquete de funciones IMAGE, descargable en la referencia (15).

Se observó que las características principales de esta imagen eran:

- Es una imagen en escala de grises.
- La matriz es de tamaño 512 x 512 (26144 píxeles).
- La tonalidad de cada píxel varía entre 0 (negro) y 3810 (blanco).

4.3.1. Tipos de imágenes.

- Las imágenes con las que se trabajan en la biblioteca de Octave pueden ser de diversas formas.
- Respecto al tamaño: *uint16* (valores de intensidad de 0 a 65535), *uint8* (0 a 255), *binary* (0 a 1),… En el caso de este proyecto, es una imagen de clase *uint16*.
- *En escala de grises:* son de tamaño M x N. Con M y N se dirige a las coordenadas de cualquier punto, y se le da un valor correspondiente a la intensidad. El 0 siempre es el negro y el valor más alto es el blanco.



- *En color:* las imágenes RGB son de tamaño M x N x 3. Con M y N se dirige a las coordenadas de un punto, y con la 3ª componente se indica:
  - $RGB(M, N, 1) = cantidad\ de\ rojo\ (red)$
  - $RGB(M, N, 2) = cantidad\ de\ verde\ (green)$

- $RGB(M, N, 3) = cantidad\ de\ azul\ (blue)$

  Mezclando una cantidad determinada de los 3 colores se puede sacar cualquier color. Si se le da a los 3 la misma cantidad, se entra en la escala de grises.

- *Imágenes binarias:* son de tamaño M x N. El 0 es negro y el 1 es blanco.

Se procedió a cargar la imagen original, con ayuda de la función *'dicomread'* del paquete *'dicom'*. La mostramos a continuación:

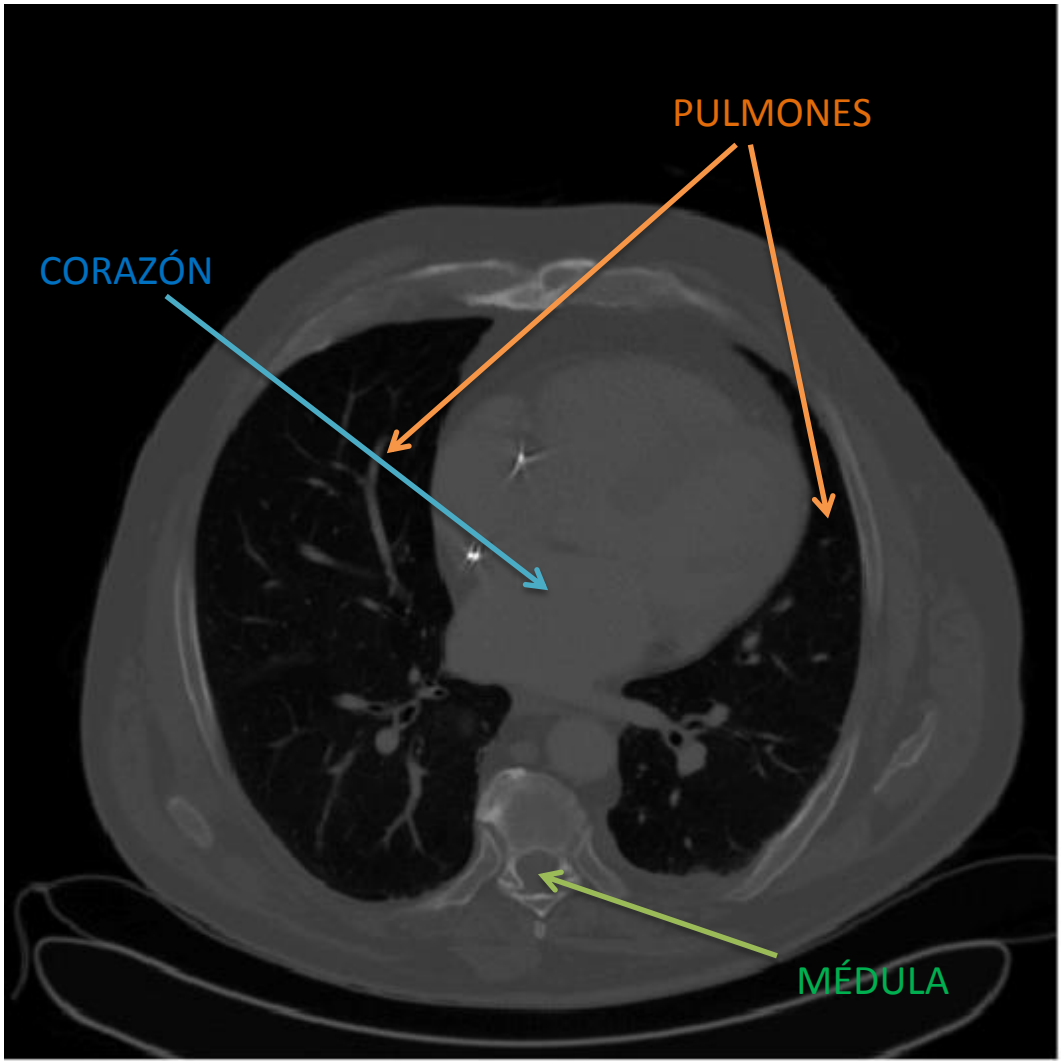


Imagen 5. Base para las pruebas con el programa

En esta imagen del tórax, se puede ver que se distinguen los órganos de interés, como son los pulmones, el corazón y la médula, que se señalan. El objetivo de este proyecto será que el programa realice el contorneo de estos órganos de forma automática.

De la imagen del paciente, también se puede extraer cierta información útil, que se quiere conservar. Se muestra un extracto de esa información:

```
ContrastBolusAgent = CONTRAST
BodyPartExamined = TC DE CUELLO CON CONTRASTE
ScanOptions = HELIX
SliceThickness =  3
KVP =  120
SpacingBetweenSlices =  1.5000
DataCollectionDiameter =  500
SoftwareVersions = 2.2.1
ProtocolName = TORAX 3 mm/Thorax/Hx
ReconstructionDiameter =  434
GantryDetectorTilt = 0
TableHeight =  139
RotationDirection = CW
XRayTubeCurrent =  260
Exposure =  173
FilterType = B
ConvolutionKernel = B
PatientPosition = FFS
CTExposureSequence =

  scalar structure containing the fields:

    Item_1 =

      scalar structure containing the fields:

        ExposureModulationType: 1x4 sq_string
        EstimatedDoseSaving: 1x1 scalar
        ExposureTimeInMs: 1x1 scalar
        XRayTubeCurrentInMA: 1x1 scalar
        ExposureInMAs: 1x1 scalar
        CTDIvol: 1x1 scalar


ExposureModulationType = DDOM
EstimatedDoseSaving =  14
CTDIvol =  13.840
StudyInstanceUID = 1.2.840.113704.1.111.12008.1323254492.1
SeriesInstanceUID = 1.2.840.113704.1.111.12008.1323254665.14
StudyID = 03
SeriesNumber =  2
InstanceNumber =  101
ImagePositionPatient =

   -218.09
   -159.51
   1523.00

ImageOrientationPatient =

   1
   0
```

Imagen 6. Extracto de la información que contiene un archivo dicom

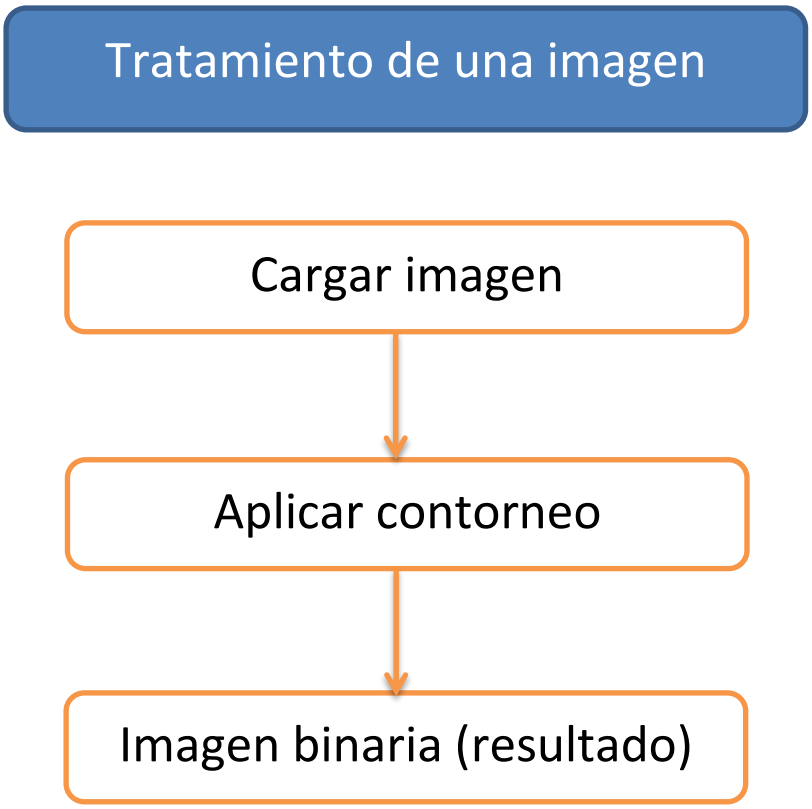
4.4. Planteamiento teórico inicial.

4.4.1. Contorneo inicial.

En el paquete de funciones IMAGE, se tienen algunas funciones que sirven para contornear. Se pudo aplicar alguna de ellas, por ejemplo la función edge, a la imagen original.

Se observó que el remarcado obtenido no era el que queríamos, ya que señalaba prácticamente todo lo que hay en la imagen, y no permitía diferenciar por zonas.

El esquema era el siguiente:



Esquema 1. Esquema inicial del proceso

Este esquema es el que se ocurrió realizar en primera instancia. El resultado es una imagen binaria, del mismo tamaño (512x512), donde los píxeles tienen un valor 0 (negro) o 1 (blanco).

¿Qué problemas hay?

La imagen obtenida no sirve. No remarca ningún órgano significativo, resalta las líneas de casi todas las partes interiores que tienen los órganos, las partes exteriores, etc.

Se muestra a continuación dicha imagen:

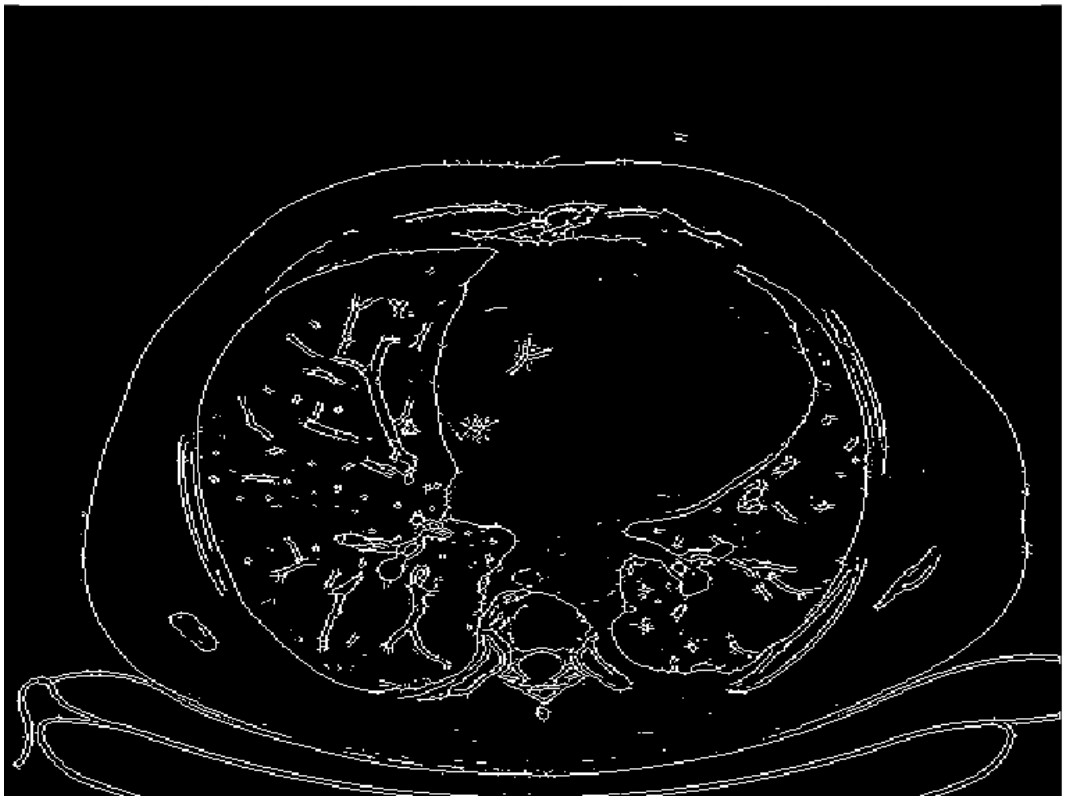


Imagen 7. Imagen original contorneada directamente

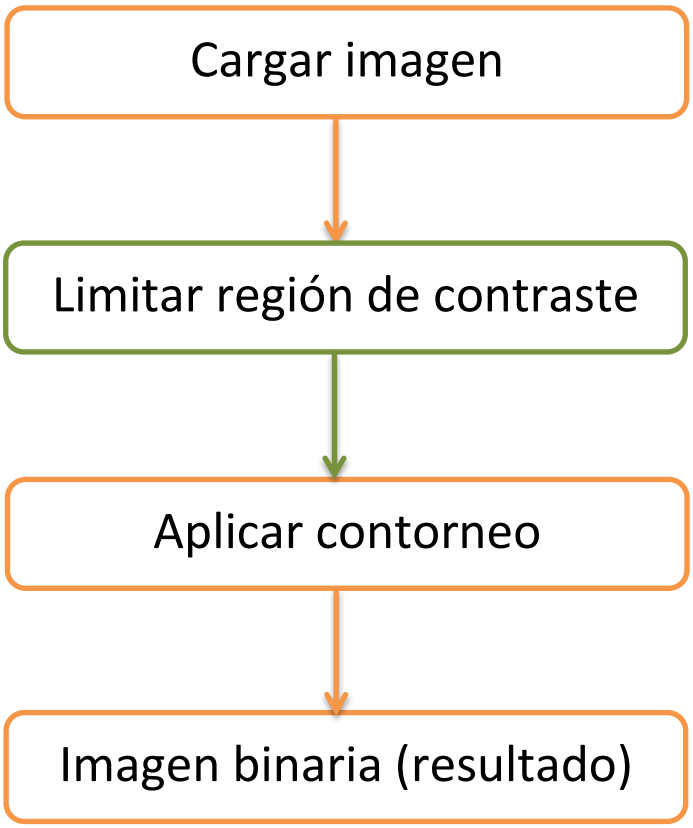
Se observa que se remarca el contorno de los pulmones, pero alrededor y en el interior se señalan muchas partes que no son de interés.

4.4.2. Cambio del contraste de la imagen como método de mejora de la segmentación.

Dado que el planteamiento inicial básico no segmenta la imagen como se desea, se plantea la siguiente idea.

Inicialmente el contraste de los píxeles varía entre 0 (negro) y 3810 (blanco). Se van a cambiar esos límites, buscando unos más adecuados para cada órgano que se quiera remarcar. Por ejemplo, si se quiere observar los pulmones, se acota el rango a [700, 800], donde 700 pasa a ser el valor de negro y 800 el valor de blanco. Los valores por debajo de 700 pasarán a valer 700, y los que estén por encima de 800, se quedarán en 800.

Así, añadiendo este detalle, se puede observar una gran mejora:



Esquema 2. Añadiendo región de contraste

La imagen binaria obtenida ahora, es considerablemente mejor que la obtenida en el anterior proceso. La dificultad está en el apartado:

Pues habrá que deducir cuáles son los mejores límites de contraste para cada órgano.

Para observar mejor los pulmones, se va a aplicar una transformación de punto, manteniendo los límites de intensidad entre 450 y 800, y convirtiendo en blanco todo lo demás. Se muestra a continuación el resultado.

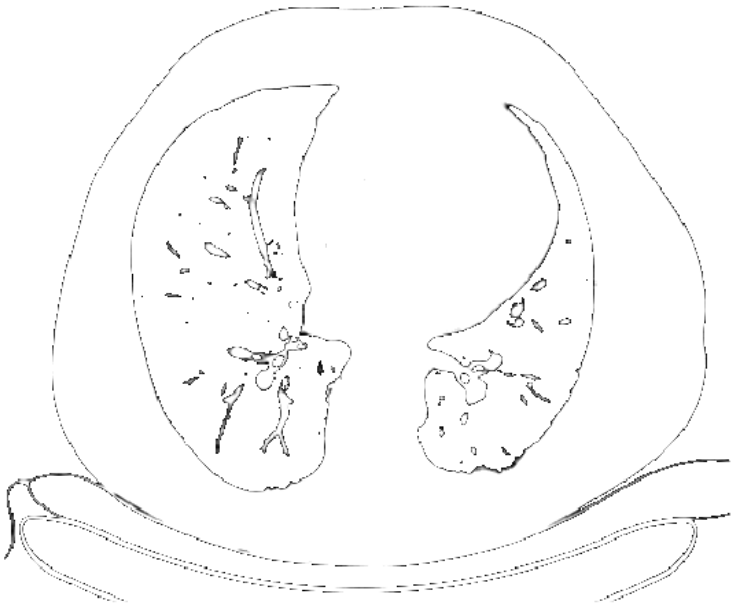


Imagen 8. Contraste mejorado

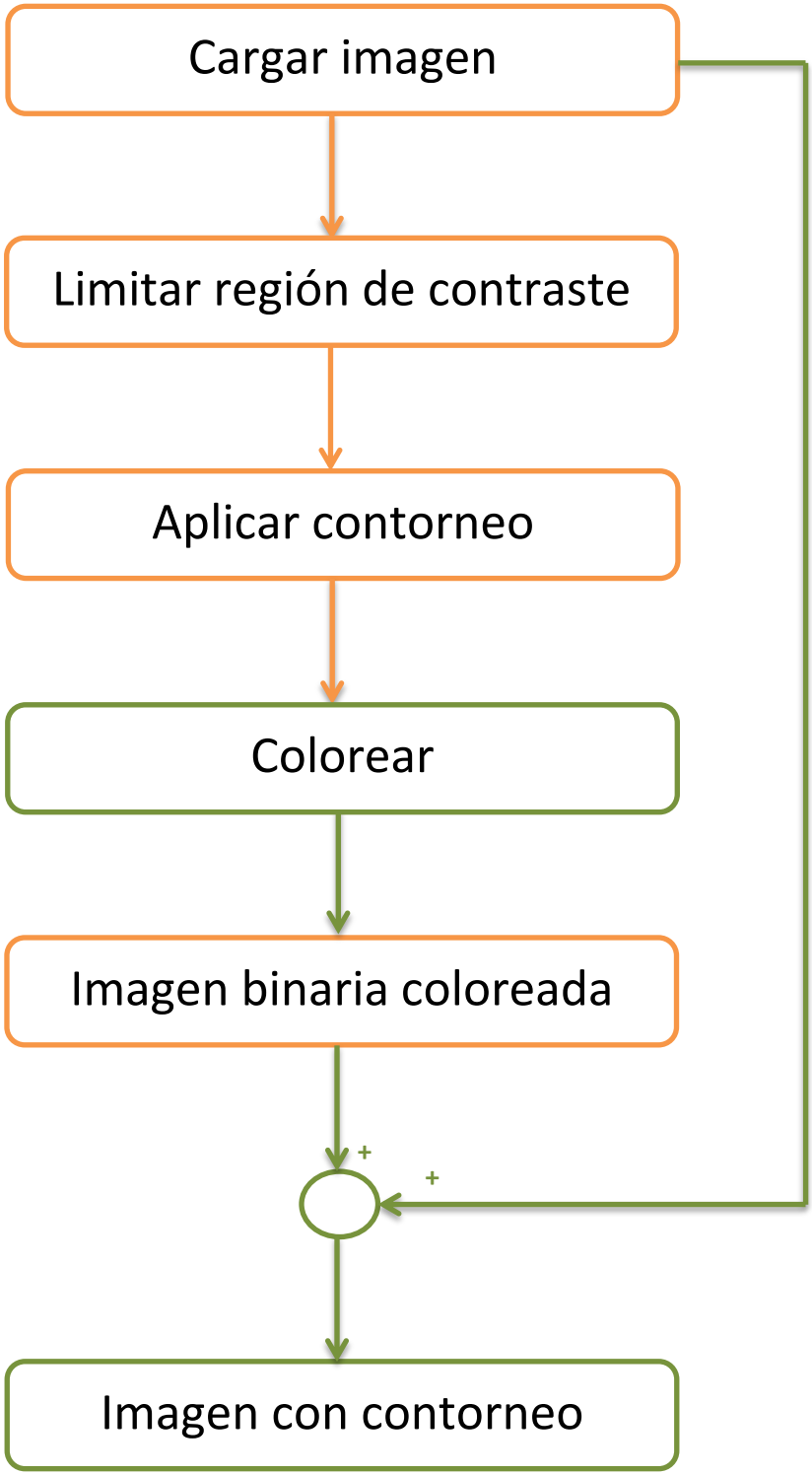
En esta imagen se observa que los pulmones se encuentran mucho más aislados, con lo cual si aplicamos contorneo se va a obtener un resultado bastante mejor.

Este tipo de juego con el contraste, se realizó experimentalmente, buscando el mejor intervalo.

4.4.3. Coloreado del contorno y adición a la imagen original.

La siguiente idea es colorear ese contorno. Es decir, se quiere transformar la imagen en escala de grises en una imagen a color. Con lo cual el tamaño de la imagen ya no será M x N, sino M x N x 3, siendo la 3ª dimensión de la matriz un indicador de la intensidad de cada uno de los 3 colores principales.

El esquema será:



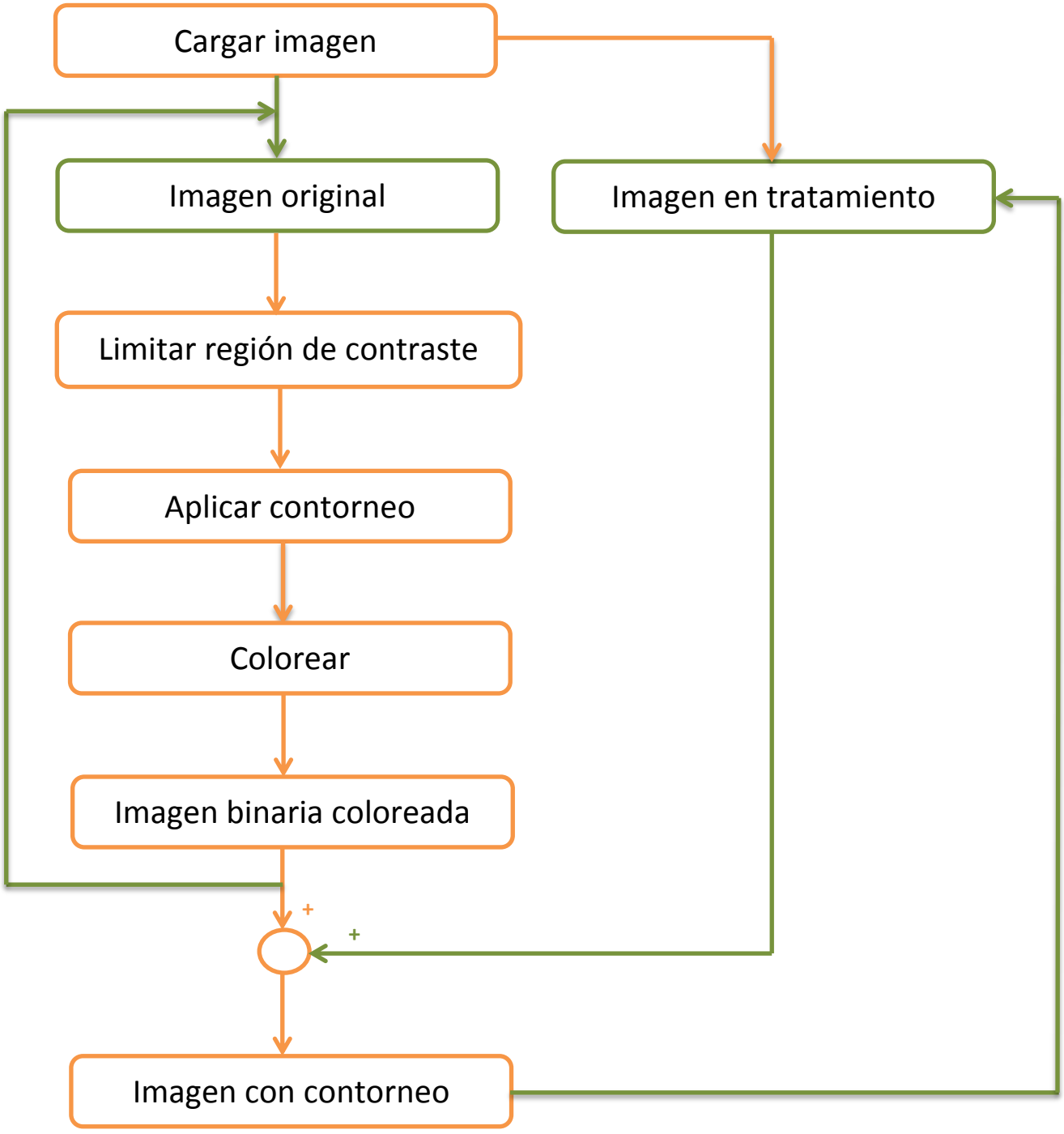
Esquema 3. Suma de imagen original y coloreada

La imagen obtenida tiene un órgano determinado contorneado. Es una imagen de tamaño 512x512x3.

4.4.4. Iteración sucesiva para obtener la imagen contorneada completa.

Seguidamente, se repite el procedimiento para diferentes regiones de contraste. Así swe podría contornear los pulmones, la médula espinal, el corazón, etc.

El esquema es el siguiente:

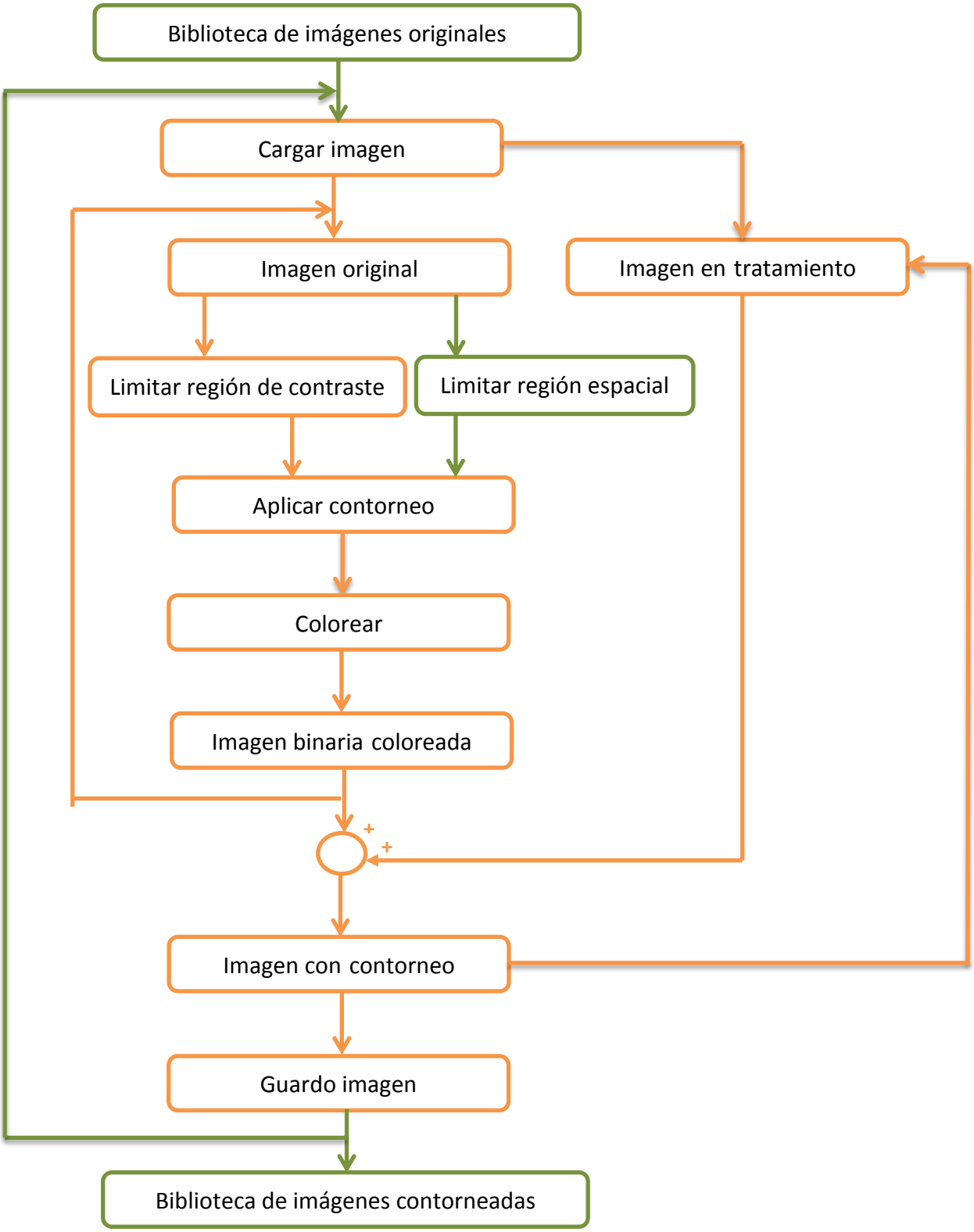


Esquema 4. Realimentación del proceso de contorneo de una imagen

Como se puede observar, se trata de contornear la imagen original de diferentes formas e ir sumándolas a la imagen en tratamiento.

4.4.5. Aplicación a una colección de imágenes.

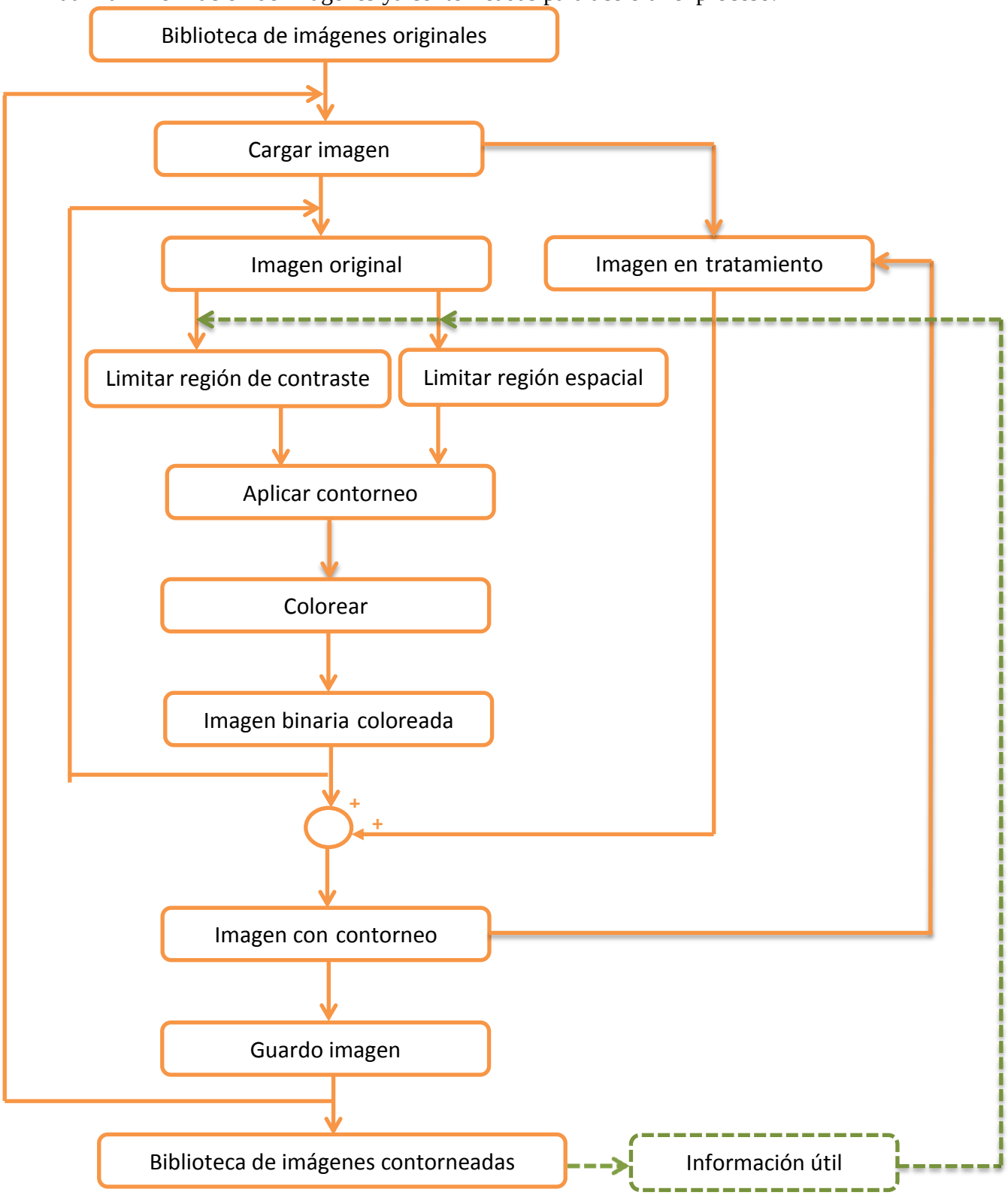
Además, se añade la selección de una región espacial, ya que cada órgano va a tener una zona de aparición más probable, y zonas en las que seguro que no aparece. Así se ahorra tiempo de procesamiento.



Esquema 5. Procesado de imágenes.

4.4.6. Empleo de información útil de las imágenes previamente contorneadas.

Parece obvio que, dada una serie de imágenes realizadas con la misma máquina, sin cambiar la posición, todas van a ser muy parecidas. Por tanto, el programa puede utilizar información de imágenes ya contorneadas para acelerar el proceso.



Esquema 6. Empleo de información útil

Este es el planteamiento inicial que se tenía, pero ahora se procede a ver cómo ha sido desarrollado.

4.5. Estudio de una imagen en particular.

Se tiene la imagen original cargada, que tiene las siguientes características:

- Tamaño 512 x 512. En total 262.144 píxeles.
- La imagen es de clase '*uint16*', es decir, sus valores de intensidad pueden variar entre 0 y 65.535, necesita 65.536 niveles disponibles.
- Esto supone un tamaño de almacenamiento considerable, en total:

$$2^{16} = 65536\ niveles$$

Necesita 16 bits, o lo que es lo mismo 2 bytes, para almacenar cada valor de intensidad. Teniendo en cuenta que hay 512 x 512 píxeles, se tiene:

$$Almacenamiento\ requerido = 524.488\ Bytes = 0{,}5\ MB$$

Se necesita 0,5 MB para almacenar una imagen.

Una vez estudiadas las características de la imagen, se procedió a tratar de contornearlas, con las funciones de contorneado que vienen ya desarrolladas.

4.5.1. Contorneo de la imagen.

- ***Función 'edge' (del paquete imagen).***

Esta función permite contornear por varios métodos: Sobel, Prewitt, Roberts, Kirsch, Canny,…

Se emplea la opción 'Sobel', por ser el método más ampliamente utilizado.

Al aplicar el contorneo se tiene una imagen binaria, donde el 0 corresponde al negro, o fondo, y el 1 corresponde al blanco, que son las fronteras de los objetos. Como el contorneo obtenido contiene demasiadas líneas, se realizará un filtrado.

- ***Filtrado experimental.***

Observando el histograma de la función, y tras determinadas pruebas, se obtuvo que para que se observaran mejor los pulmones, se debía realizar un filtrado, de forma que se dejasen únicamente los valores entre 700 y 800, de la siguiente forma:

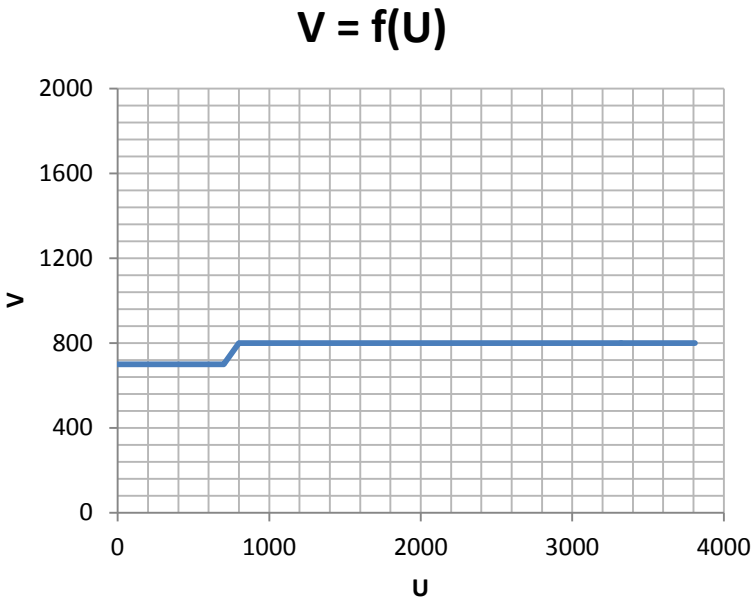


Gráfico 9. Operación de punto entre 700 y 800

Donde U corresponde a los valores de intensidad iniciales, y V a los valores de intensidad finales. Matemáticamente la función que se ha realizado se representa así:

$$V = f(U) = \begin{cases} V = 700, si\ U \leq 700 \\ V = U, si\ 700 < U < 800 \\ V = 800, si\ V \geq 800 \end{cases}$$

Se trata de una operación de punto, explicada en el apartado teórico, que trata de conseguir que se observen mejor los pulmones, eliminando el fondo.

Así, se contornea y se obtiene una imagen mejor, en la que se han eliminado líneas que sobraban. Pero se tiene un problema añadido, y es que la línea que rodea los pulmones no es continua.

Imagen 9. Imagen filtrada contorneada.

También se puede tomar una imagen con un filtrado entre 450 y 800:

Imagen 10. Imagen filtrada en [450, 800] contorneada

4.5.2. Proceso de comparación de imágenes.

Para paliar esta situación, se plantea una función, que recorre los puntos de la imagen filtrada contorneada, y en los puntos blancos, se va a la imagen original y copia los de alrededor.

Por ejemplo:

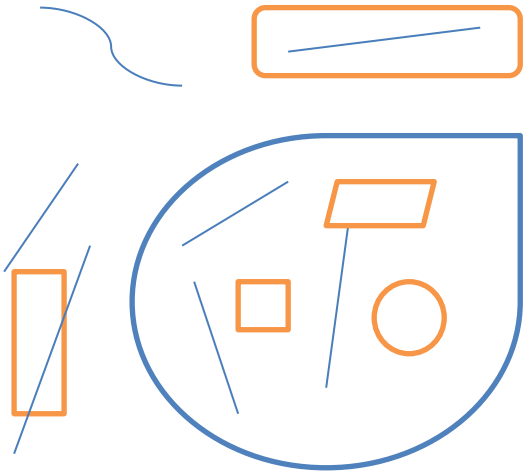


Imagen 11. Ejemplo de figuras

Imagine que el objeto azul es el contorno que se quiere destacar, pero han salido marcados elementos interiores y elementos exteriores que no son de interés.

Se realiza el filtrado a la imagen, y al contornear se obtiene lo siguiente:

Imagen 12. Ejemplo de contorneo

En cada punto contorneado, se va a la imagen original, y copia el valor de todos los de alrededor.

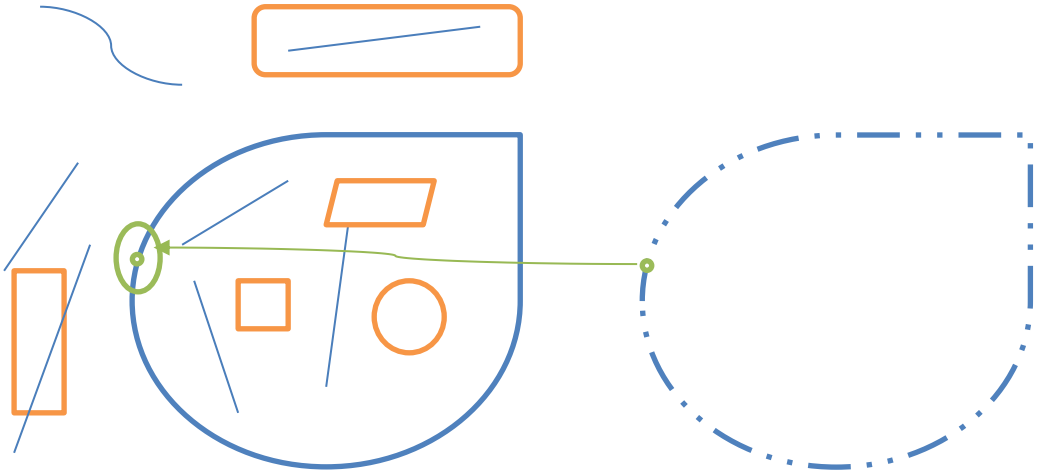


Imagen 13. Ejemplo de comparación.

Así se obtendría el contorno válido:

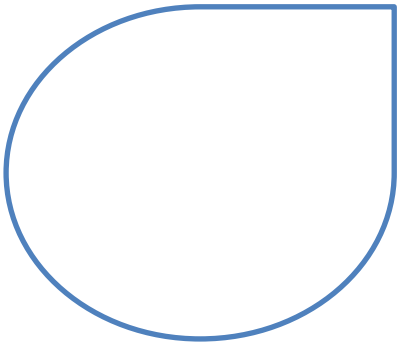


Imagen 14. Ejemplo de contorno válido.

El problema es que el tiempo de computación es alto, ya que hay que definir un radio para el círculo (o rectángulo) alrededor del punto, y acceder a todas esas posiciones de los píxeles, con el consiguiente derroche de memoria.

De todas formas, la primera prioridad no era el tiempo, sino que saliese bien, con lo cual se dio por válido para esta imagen.

Imagen 15. Contorneo aproximado

4.5.3. Coloreado del contorno.

El siguiente paso que se nos plantea es colorear ese contorno y añadirlo a la imagen original, de forma que se quede resaltado.

Para colorearlo, se vale de una serie de funciones del paquete imagen, que realizan transformaciones. Se transforma la imagen binaria en formato de imagen en escala de grises. Posteriormente se transforma esta imagen en escala de grises a imagen a color, dando por ejemplo el valor máximo al rojo, y cero al verde y al azul. Así, se remarcarían los pulmones de color rojo, como se observa en la siguiente imagen.

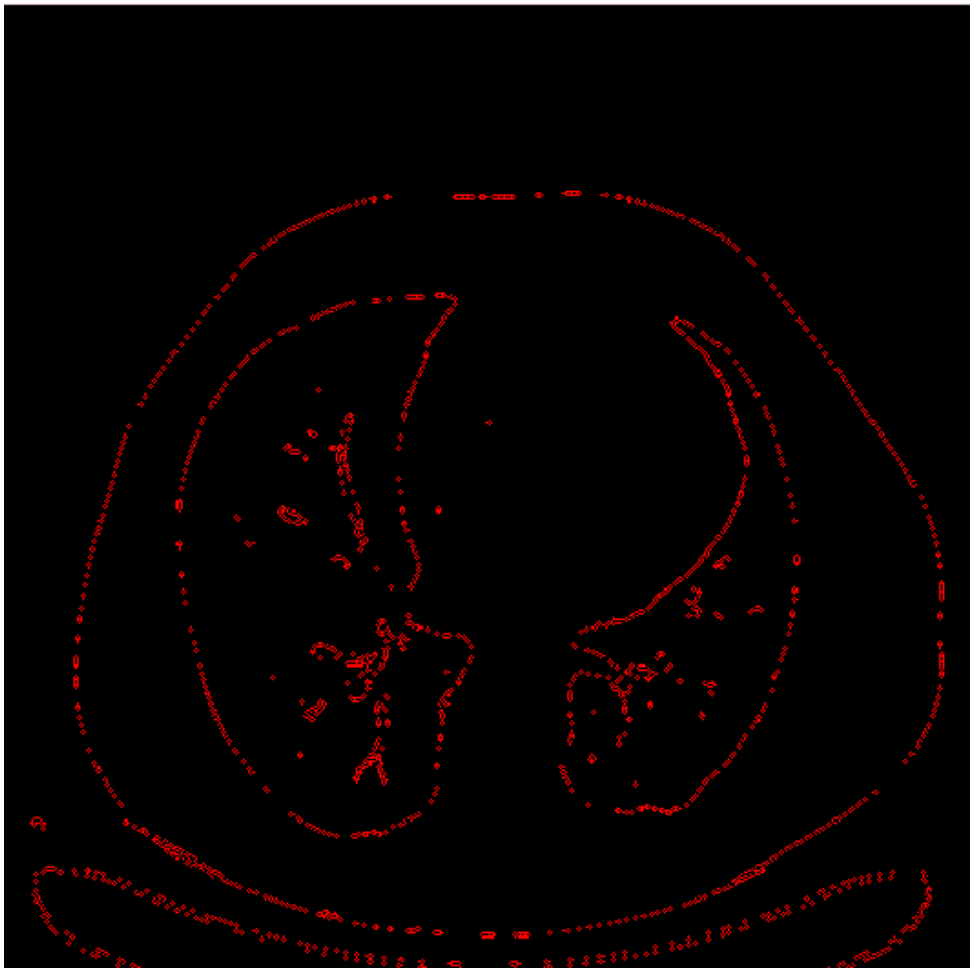


Imagen 16. Ejemplo de contorneo coloreado.

4.5.4. Adición de la imagen coloreada a la imagen original.

Una vez se tiene la imagen coloreada, se procede a añadirla a la imagen original. Para ello se realiza un proceso de barrido por todos los píxeles. Cuando se trata de un píxel que tiene color, los valores de R, G y B se mantienen en la imagen final. Cuando se trata de un píxel sin color, se toma R = G = B = el valor que tengan en la imagen original. Así, se obtiene la siguiente imagen:

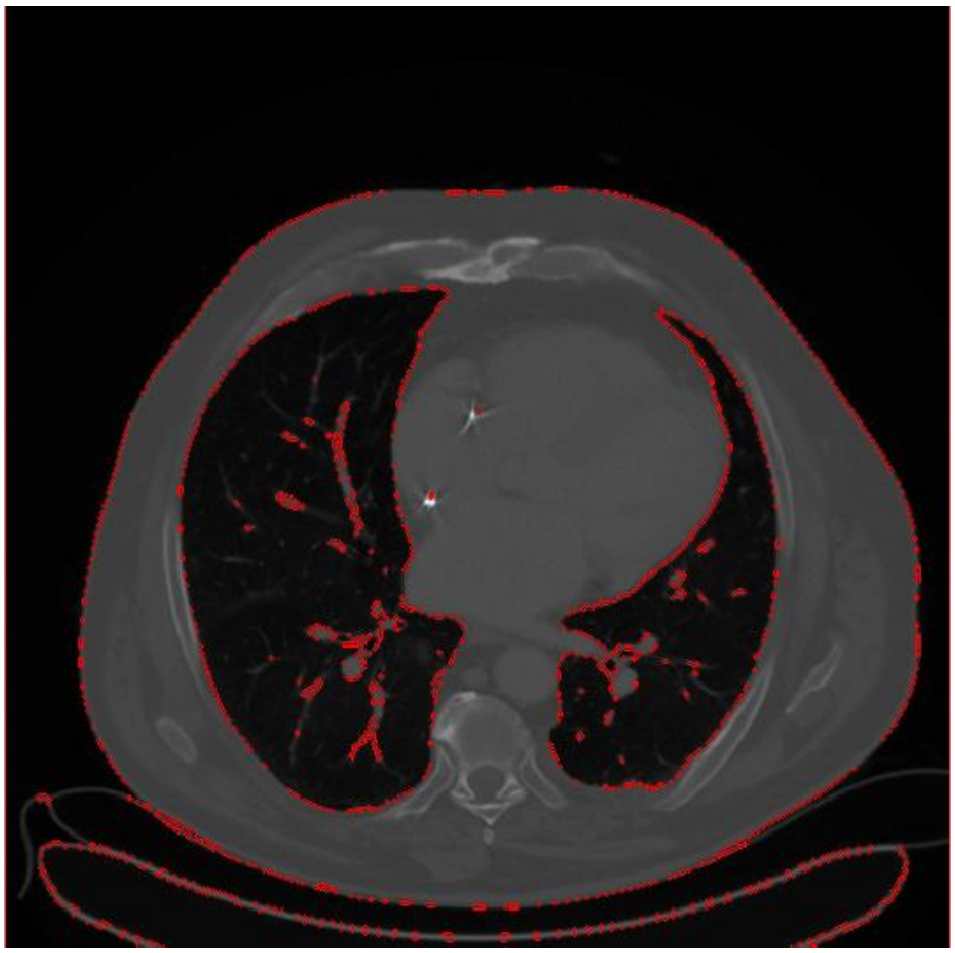


Imagen 17. Ejemplo de adición del contorneo coloreado a la imagen original.

De esta forma, se tiene contorneados los pulmones de la imagen ejemplo. Pero esta imagen se puede mejorar bastante. Realizando otro filtrado, se obtiene el contorneo que se muestra en la siguiente imagen, que tampoco es perfecto.

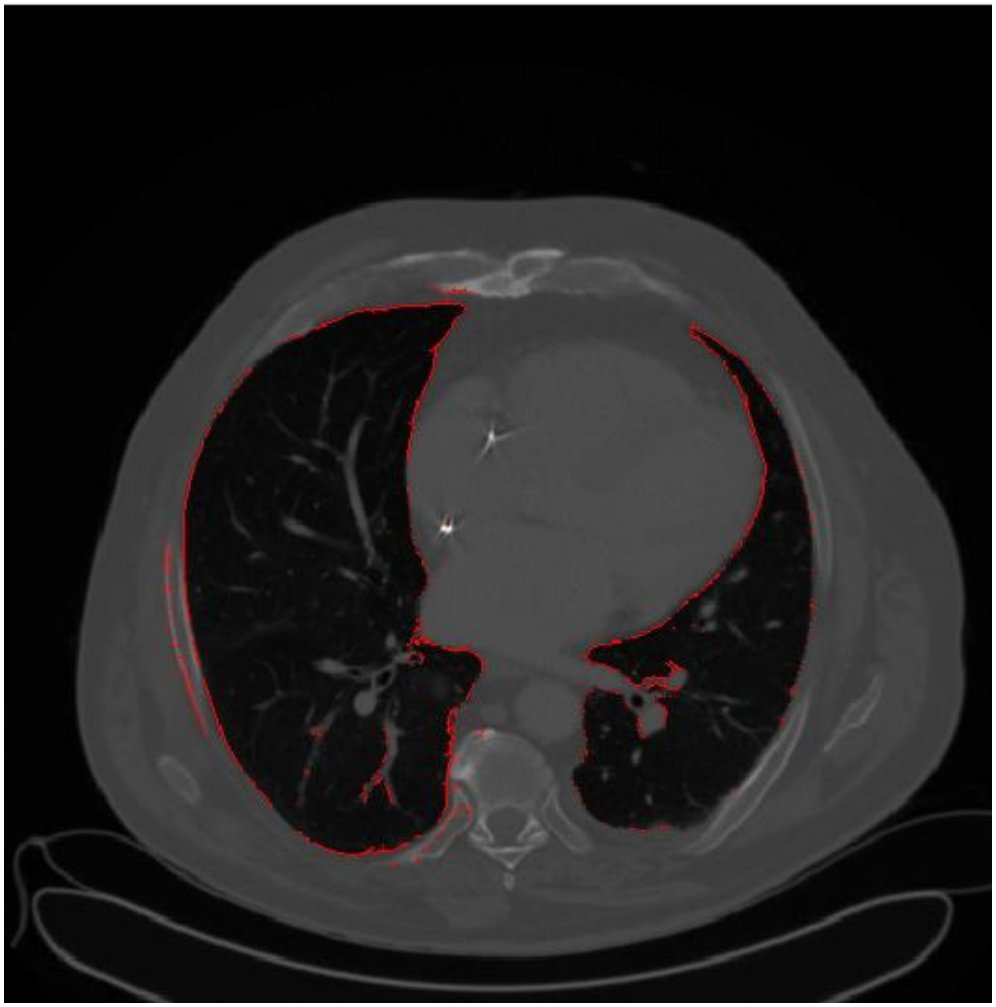


Imagen 18. Filtrado y coloreado

4.5.5. Filtro de posición.

Además del filtro de intensidad, se aplicó un filtro de posición, sabiendo que determinadas zonas no pueden contener las líneas que se buscan, y por ello se eliminan.

4.5.6. Propiedades de los objetos de una imagen.

Se encuentra el autor con funciones ya definidas en el paquete imagen, que permiten hallar los objetos que contienen las imágenes, y obtener de ellos ciertas propiedades, como por ejemplo el área.

Los objetos dentro de una imagen son zonas que tienen una intensidad similar, sin cambios bruscos, y que cumplen propiedades de continuidad, etc.

Aplicando la función '*bwlabel'* a la imagen original contorneada, se obtiene como resultado una matriz que contiene para cada píxel el número de objeto al que pertenece dicho píxel. Si no pertenece a ningún objeto, aparece el valor 0. Y con la función '*regionprops'* se obtiene las propiedades de los objetos.

Combinando esta información, se puede decidir si un píxel es de interés, si pertenece a un objeto de gran área (pues se está buscando los pulmones, que tienen un área considerable), y así se descartan pequeños objetos.

También se obtiene la posición del 'centroide' de los objetos, y si no se encuentra dentro del rango deseado, se descarta.

Con esto, se consigue una imagen bastante limpia de los pulmones.

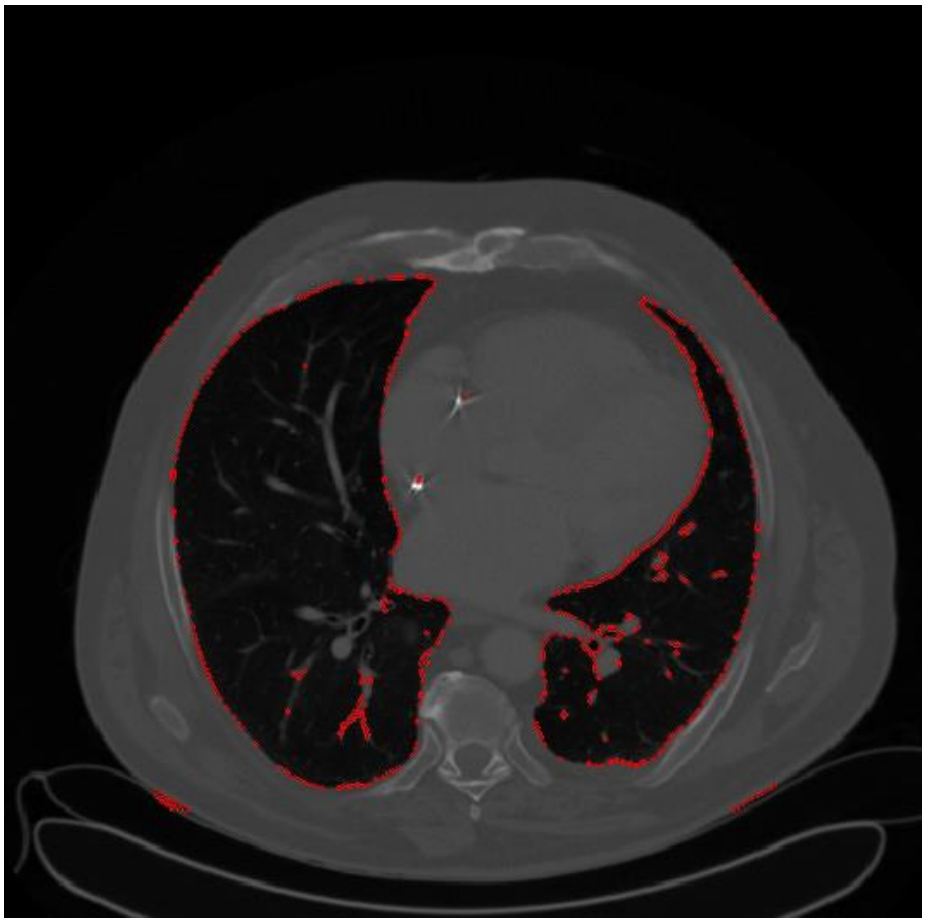


Imagen 19. Aplicación de propiedades y coloreado.

El siguiente paso, sería aplicarlo a todas las imágenes. Pero aparecen problemas, no en todas las imágenes es válido este proceso, pues en algunas aparecen líneas de más o líneas sin remarcar. Con lo cual, se creó necesario cambiar el método.

Pero previamente, dada la gran cantidad de líneas de código, se connotó necesario crear una serie de funciones que contengan parte de los algoritmos.

4.5.7. Definición de funciones.

Conforme se avanza en el trabajo, la cantidad de información dentro del programa es considerable, y comienza a ser difícil entender un programa que contiene 300 líneas. Por ello, se procedió a resumir el contenido de varias líneas que realizan una acción sencilla, en una función. Por ejemplo, la función colorear, simplemente recibe una

imagen y devuelve la coloreada. Es suficiente una línea de código para ello, ocultando las numerosas líneas de código que el programa necesita para realizar el proceso.

Siguiendo el código general de definición de funciones en Octave, similar al de Matlab, se crean funciones guardándolas en un archivo *'nombrefuncion.m'*, que contiene la siguiente estructura:

##Ayuda.

Function *nombrefuncion*

Body;

Endfunction

Se puede utilizar el comando '*return'* para que el programa salga de la función en cualquier instante.

Octave guarda automáticamente una variable '*nargin'* con el número de argumentos que se le envía a la función, y el autor puede ayudarse de ella en el cuerpo de la función.

En primer lugar, se creó la función colorear. Esta función recibe una imagen binaria, y debe devolver una imagen coloreada, en formato RGB. Por ello, hay que indicarle también los colores que se quieren.

Ejemplo:

Imagen_coloreada = colorear (imagen_binaria, red, green, blue)

En las variables *red*, *green*, *blue* se introduciría la cantidad que se desea de cada uno de los tres colores.

La función queda así definida:

```
function img_coloured = colorear(img_orig, R, G, B)

%Primero se transforma de matriz(imagen binaria) a imagen gris:
I = mat2gray (img_orig);

%Luego se transforma la imagen binaria a una imagen indexada:
[img,map] = gray2ind(I, 2);
%Luego se transforma la imagen indexada a una imagen RGB:
RGB = ind2rgb(img, map);
%Transformamos a clase uint16 (valores 0-65535):
img_coloured = im2uint16(RGB);

if nargin == 1
error("You must specify a colour for the image")
endif#

if nargin == 2
switch(R)
case "red"
R = 65535;
```

```
G = 0;
B = 0;
case "green"
R = 0;
G = 65535;
B = 0;
case "blue"
R = 0;
G = 0;
B = 65535;
case "yellow"
R = 65535;
G = 65535;
B = 0;
case "pink"
R = 65535;
G = 0;
B = 65535;
case "orange"
R = 65535;
G = 165 / 255 * 65535;
B = 0;
case "violet"
R = 128 / 255 * 65535;
G = 0;
B = 128 / 255 * 65535;
case "white"
R = 65535;
G = 65535;
B = 65535;
case black
R = 0;
G = 0;
B = 0;
otherwise error("non defined valid colour")
endswitch#
endif#

img_aux = img_coloured;
colores = img_coloured; %Creo una matriz de colores.
colores(:, :, 1) = R;
colores(:, :, 2) = G;
colores(:, :, 3) = B;

img_coloured(img_aux > 0) = colores(img_aux > 0);

endfunction
```

Se ha dado la posibilidad de introducir la función de la siguiente forma:

Imagen_coloreada = colorear (imagen_original, color)

Donde *color* puede ser cualquiera de estos: "*green*", "*yellow*", "*red*", "*blue*", "*pink*", "*gray*", "*orange*", etc. Con esto, se facilita la labor del usuario.

También se incluye un archivo de ayuda en la función, que se pone en el encabezado del archivo:

```
## -*- texinfo -*-
## @deftypefn  {Function File} {@var{img_coloured} =} colorear
(@var{img_orig}, @var{colour})
## @deftypefnx {Function File} {@var{img_coloured} =} colorear
(@var{img_orig}, @var{R}, @var{G}, @var{B})
##
## Esta funcion colorea la imagen binaria @var{img_orig},
transformandola
## en una imagen @var{img_coloured} de clase uint16 (valores de 0 a
65535)
## y formato RGB.
##
## Podemos especificar el color que queremos de los siguientes: "red",
"green",
## "blue", "yellow", "pink", "orange", "violet", "white", "black". O
bien
## especificamos la cantidad que queremos de Red, Blue y Green. Notar
que los
## valores deben ir de 0 a 65535.
##
## Variables de entrada:
## @var{img_orig} -> imagen binaria original.
## @var{colour} -> el color que queremos ("red", "green",
## "blue", "yellow", "pink", "orange", "violet", "white", "black").
## @var{R} -> valor de rojo que queremos (0-65535).
## @var{G} -> valor de verde que queremos (0-65535).
## @var{B} -> valor de azul que queremos (0-65535).
##
## Variables de salida:
## @var{img_coloured} -> imagen coloreada (de clase uint16: 0-65535).
##
## For example:
##
## @example
## @group
##
## %Si queremos colorear la imagen de amarillo:
## img_coloured = colorear(img_orig, "yellow")
## max(max(img_coloured ))
##    @result{} ans(:,:,1) = 65535
##    @result{} ans(:,:,1) = 65535
##    @result{} ans(:,:,1) = 65535
## @end group
##
## @group
## %Si queremos especificar los valores de rojo, verde y azul:
## img_coloured = colorear(img_orig, 35480, 20000, 15756)
## max(max(img_coloured ))
##    @result{} ans(:,:,1) = 35480
##    @result{} ans(:,:,1) = 20000
##    @result{} ans(:,:,1) = 15756
## @end group
##
## @end example
##
## @seealso{}
## @end deftypefn

## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
## Created: Jun 13, 2014
## Last modified: Aug 29, 2014
```

De forma que si en el entorno de Octave se escribe 'help colorear', se recibe la ayuda aquí descrita, como se puede ver en la siguiente imagen:

Imagen 20. Salida por pantalla de la ayuda de una función.

Se puede introducir ejemplos en la ayuda, como se observa. Asimismo, se puede recomendar que se consulten otras funciones, en el apartado '*see also*'.

La siguiente función que se creó es una función que se llama *'juntarimg'*, que sirve para unir una imagen en escala de grises con una imagen coloreada. Lo que hace es recorrer todos los píxeles, y donde hay color, deja el color. Si no hay color, pone los valores de la imagen original en escala de grises, R = G = B.

Se guardaron todas las funciones en una carpeta '*myfunctions*', en el directorio de los paquetes de funciones, de forma que el programa Octave pueda acceder a ellas. Si se

quiere dejarla en otro lugar, se tendrá que añadir el directorio a los de búsqueda de funciones, por ejemplo con ayuda de la función '*addpath(dir)*'.

La siguiente función que se creó fue una función que se llama *'contornear'*, y que permite contornear variando fácilmente los parámetros del gradiente de la imagen y el método. Es cierto que lo único que hace es llamar a la función '*edge'* y a la función '*imgradient'*, pero resulta más sencillo, y se obtiene la imagen contorneada con una sola línea de código.

Otras funciones útiles que se han desarrollado son *'eliminar_X'*, *'eliminar_Y'*, *'eliminar_Square'*, que sirven para eliminar franjas de la función en el eje X, en el eje Y, o para eliminar un cuadrado que no interese. De esta forma se reducen líneas de código del programa principal. El concepto de estas funciones es sencillo, se le envía información con la imagen y los extremos de la franja o franjas que queremos eliminar, y se devuelve la imagen ya con el área sustraída (color negro).

Otra función que mejora el proceso de contorneado es *'comparar_negro'*. Esta función lo que hace es que recibe dos imágenes, y lo que es negro en una, lo transforma en negro en la otra imagen. Esto puede servir cuando se tiene una imagen contorneada que se sabe que contiene seguro las líneas buscadas, aunque contenga líneas de más. Mientras que se tiene otra imagen que tiene menos líneas. Así, con este barrido de la imagen se mejora el contorneo, eliminando líneas innecesarias.

El proceso de eliminación de píxeles que no pertenecen a objetos de área considerable, también se puede reflejar en una función que llamamos *'excluir_areas'*. Esta función recibe la imagen original, el área de exclusión mínima, y el área de exclusión máxima.

Se desarrollaron las funciones *'area_escogida'* y *'area_devuelta'*, para recortar una imagen y después del proceso devolverla a la original, respectivamente.

4.5.8. Contorneo de los órganos por separado.

A continuación, este programa se centra en contornear por separado pulmones, médula y corazón, y juntarlos todo en uno, con la imagen base de las investigaciones.

Para guardar las imágenes en una carpeta, a la hora de trabajar con muchas, se emplea la función *'imwrite'*. Y se emplea el comando '*save'* para guardar la info de la imagen DICOM. Aunque posteriormente se sustituye por el comando '*fopen'*, que proporciona más facilidad de formatos.

Se utiliza los comandos:

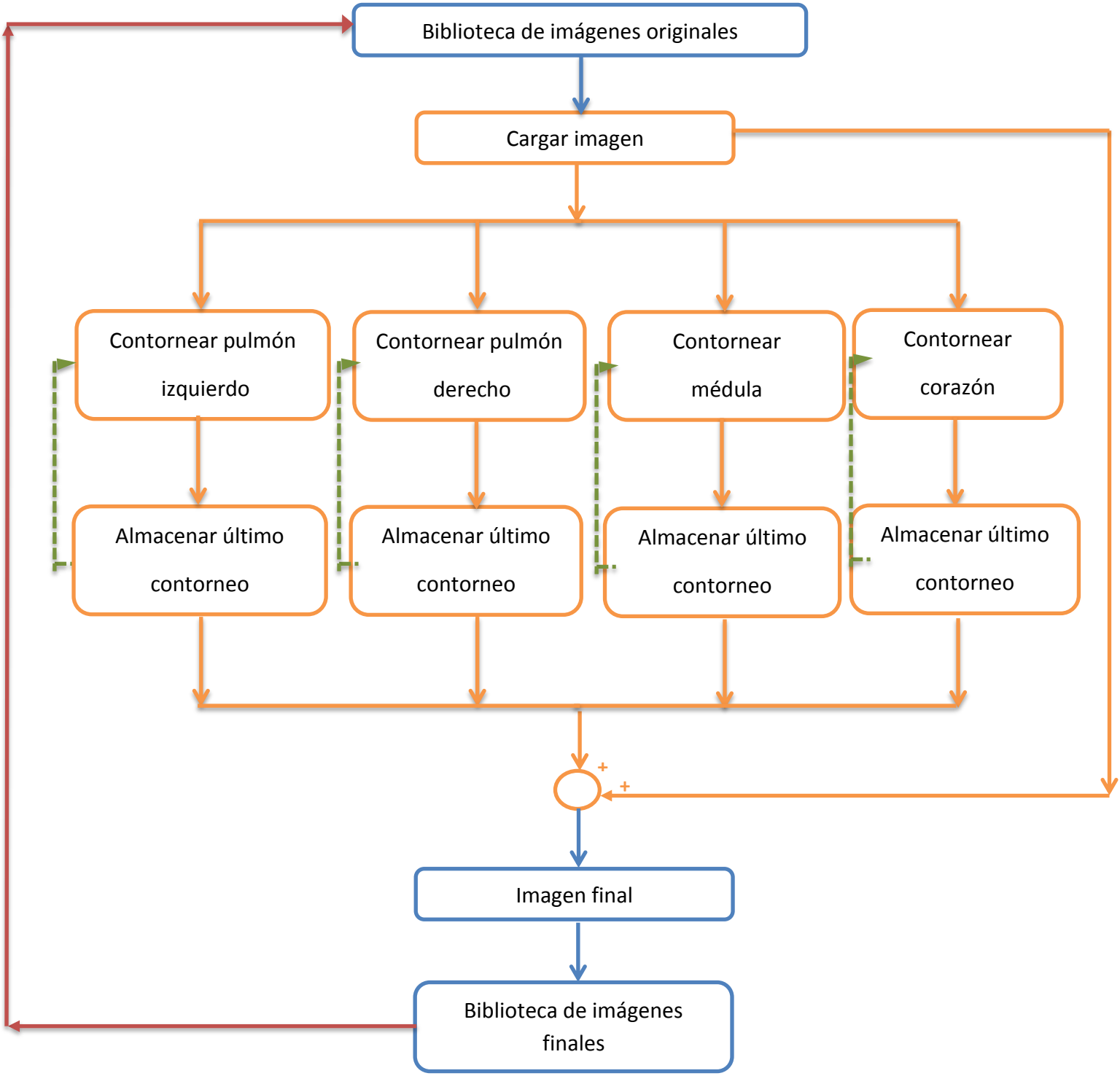
- '*cd'*: para cambiar de directorio cuando así se requiere.
- '*source'*: para cargar los archivos .m. Simplemente se ejecutan las líneas de código como si se escribiesen directamente en la ventana de comandos.

El proceso de trabajo es el siguiente:

❖ Se comienza un bucle de procesamiento de imágenes.

- Se llama al programa que contornea los pulmones.
- Se llama al programa que contornea la médula.
- Se llama al programa que contornea el corazón.
- Se Obtiene la imagen que se quiere, y se guarda en una carpeta.

Y así, se repite con todas las imágenes.



Esquema 7. Esquema de trabajo del programa

Aunque en el esquema aparecen los 4 contorneos de forma simultánea, se realizarán uno a uno, pues el derroche de memoria es grande.

¿Qué problemas se encuentran?

El programa era demasiado lento, del orden de cinco minutos por imagen, con lo cual se tuvo que retocar las funciones y simplificar los algoritmos. Pero finalmente se consiguió que tardase unos 10 segundos por imagen.

El contorneo de los pulmones falla en algunas imágenes. La mejora de esto se ocurrió que fuese separando pulmón izquierdo y pulmón derecho, de esta forma el contorneo sale bastante mejor.

Inicialmente se contorneaban los pulmones y la vértebra que envuelve la médula, obteniendo la siguiente imagen:

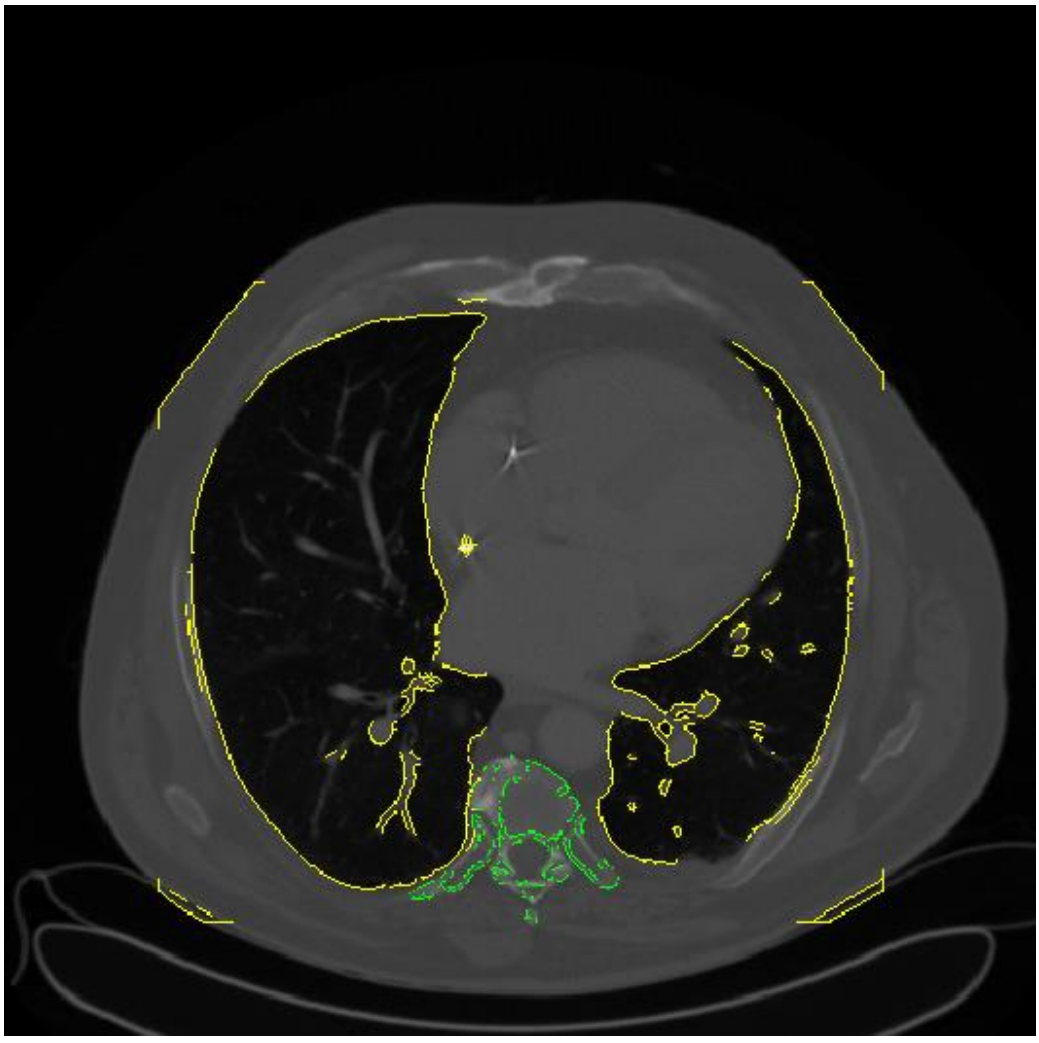


Imagen 21. Contorneo de dos órganos.

Pero se observa que el contorneo era bastante pobre, pues las líneas eran discontinuas, y además contenía líneas interiores innecesarias.

Se analizó el histograma de una imagen, para saber mejor qué límites se tenía que escoger para cada pulmón, para la médula y para el corazón.

Se decidió cambiar el método de contorneo de los pulmones, para mejorarlo.

4.5.9. Concepto de "pequeños cambios" de una imagen a otra.

Es cierto que, a partir de la imagen base, que era la de un corte intermedio del tórax, si se va avanzando tanto hacia un lado como a otro, los cambios de una imagen a la inmediata siguiente no son muy significativos. Ello hizo ver al autor que interesa conservar el "último contorneo", pues el contorneo de la imagen tiene que estar cercano a él, en su entorno.

Para ello puede servir la función *'comparar'*, que inicialmente se había desarrollado con otra idea, para buscar en los entornos de un punto y quedarse con esos puntos.

4.5.10. Incorporación de una imagen leyenda.

A nivel académico, para que una persona que no entiende de medicina sepa los órganos que estamos remarcando, se añade una leyenda indicando de qué órganos se trata. Sin embargo, a la hora de ser utilizado el programa por médicos, se debería quitar esa opción, para no estropear las imágenes.

4.5.11. Aceleración de los bucles.

Para hacer que el programa funcione más rápido, la última novedad que se introdujo fue acelerar todos los bucles, reescribiéndolos en la forma de tratamiento de matrices más efectiva. De esta forma se redujo el tiempo considerablemente, pues de tardar 3-5 minutos por imagen, se pasó a 10 segundos, un tiempo que comienza a ser competitivo.

5. Estructura del programa definitivo.

5.1. Contenidos de este capítulo.

En este apartado se va a tratar de desarrollar el programa como finalmente ha quedado. De esta forma, el usuario que no desee conocer cómo se ha desarrollado el programa, sino que busque simplemente entender su estructura final, puede consultar exclusivamente este apartado.

Se describirá los programas de contorneo uno a uno, y luego la biblioteca de funciones. Se pueden consultar los programas completos en el anexo I.

5.2. Estructura del programa.

El programa final queda dividido en varios bloques. La base es el programa de trabajo, y se le pueden anidar programas de contorneo de órganos. Se resume de la siguiente forma.

- ✓ Programa de trabajo.
  - ❖ Contorneo del pulmón izquierdo.
  - ❖ Contorneo del pulmón derecho.

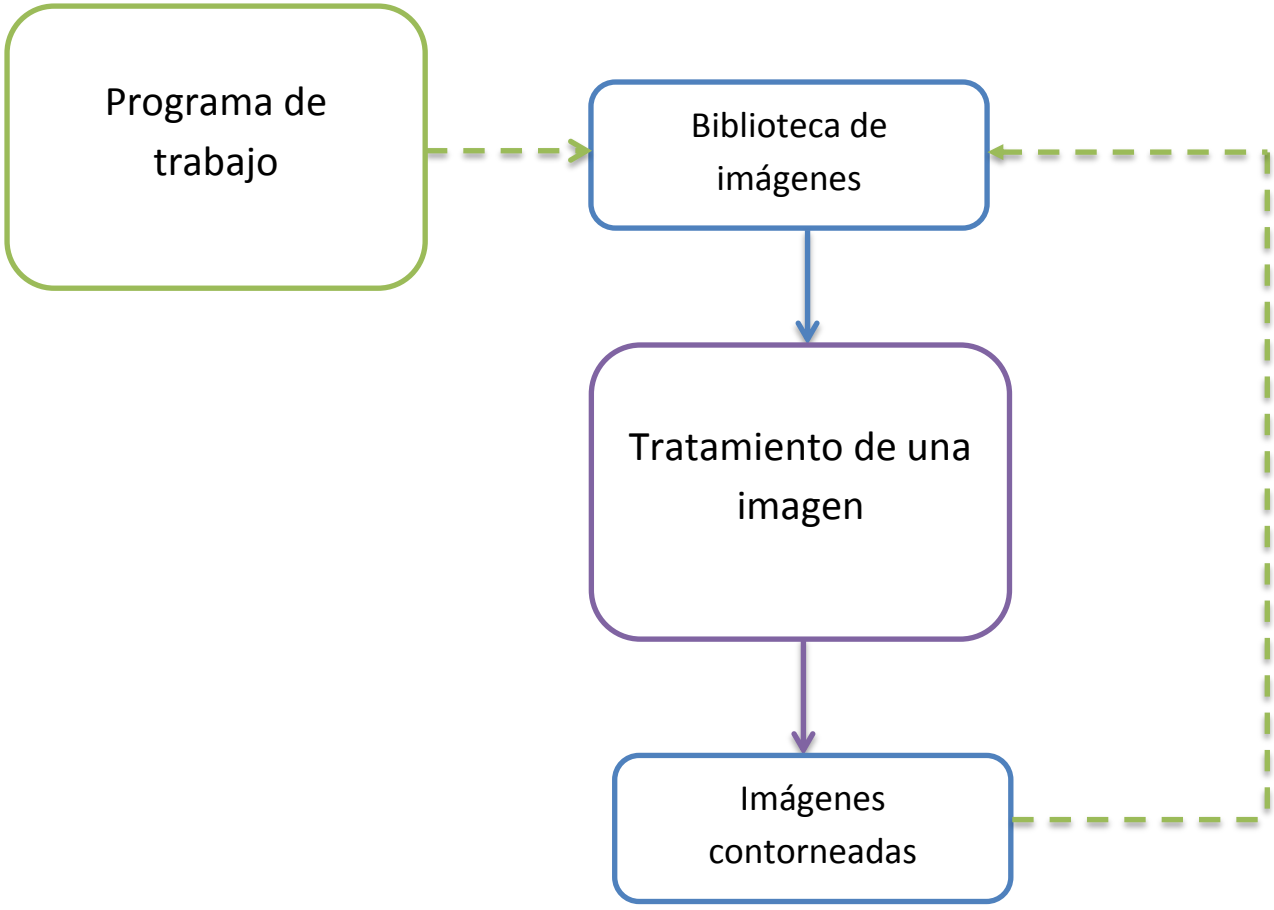
- Contorneo de la médula.
- Contorneo del corazón.

Dentro del programa de trabajo, se puede elegir las imágenes que se quiere contornear, indicando con números. La máquina registra 199 imágenes, de la 0 a la 198. La número 100 es la imagen que se toma como base. Por ello se realiza la segmentación de la 100 a la 198, y posteriormente de la 100 a la 0 en sentido decreciente. Si se quiere segmentar menos imágenes, se indica con los números las que se quiere.

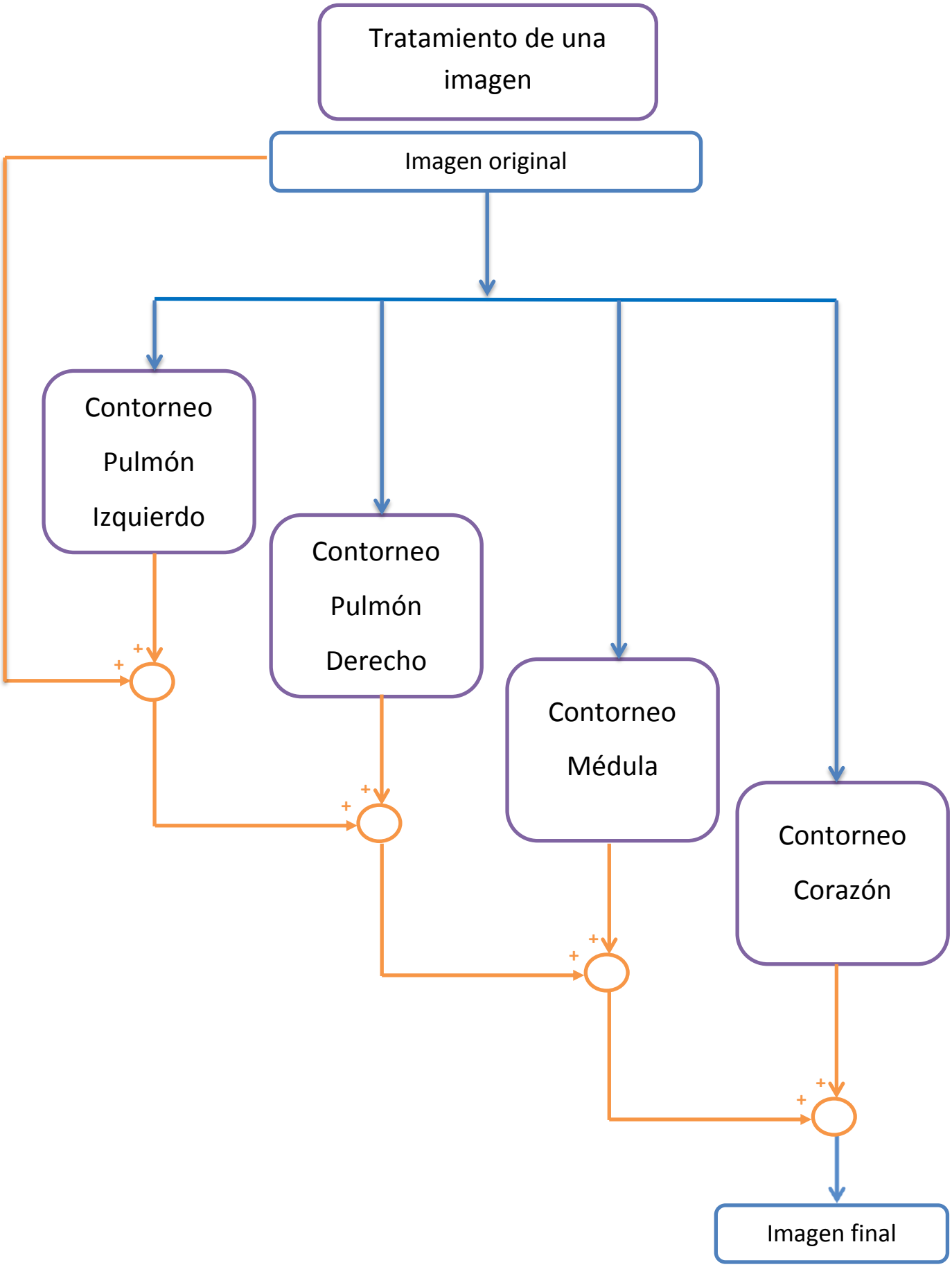
El programa por defecto llama a los 4 programas de contorneo, pero se le puede indicar sólo algunos órganos.

En el siguiente esquema se representa el funcionamiento del programa. El programa de trabajo indica a la biblioteca cuáles son las imágenes que requieren tratamiento. Se van tratando una a una y almacenando en una biblioteca de imágenes finales.

A continuación, se detalla el apartado de tratamiento de una imagen. En principio consiste en contornear los cuatro órganos: pulmón izquierdo, pulmón derecho, médula y corazón, en ese orden. Pero se puede abrir cualquiera de los caminos, y contornear menos órganos.



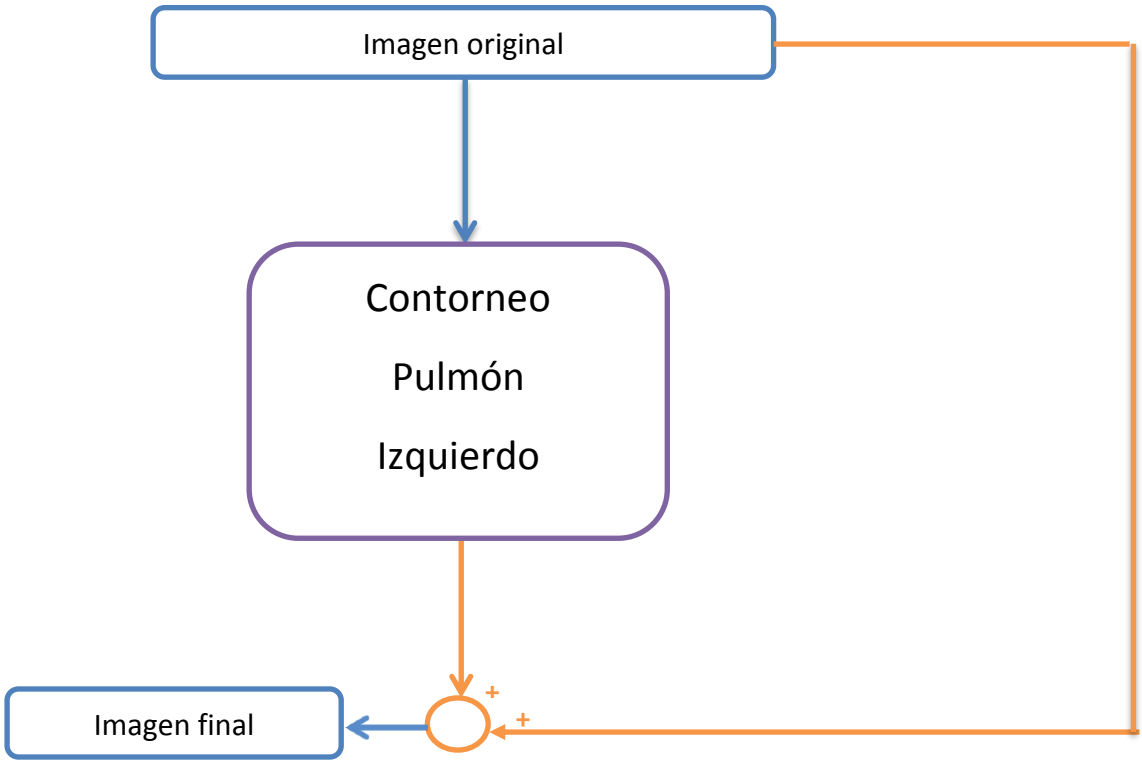
Esquema 8. Programa de trabajo



Esquema 9. Tratamiento de una imagen

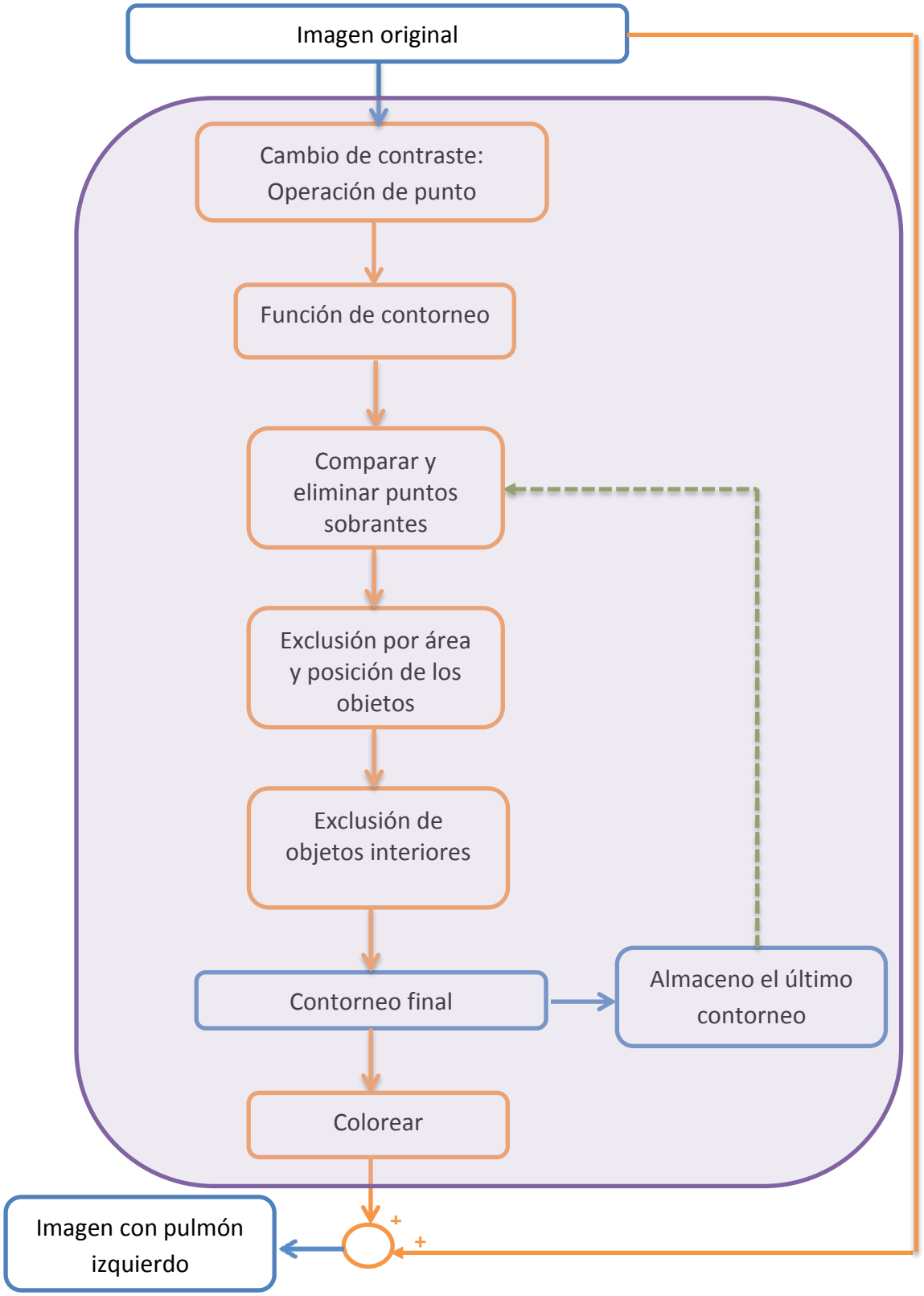
5.3. Contorneo del pulmón izquierdo.

El primer órgano al que se le realiza la operación de tratamiento es al pulmón izquierdo.



Esquema 10. Contorneo del pulmón izquierdo.

En este esquema se puede ver el proceso sencillo que se quiere realizar. Sin embargo, seguidamente se detallará el interior de ese proceso llamado "contorneo pulmón izquierdo", con el siguiente esquema:



Esquema 11. Contorneo del pulmón izquierdo desarrollado.

El contorneo del pulmón izquierdo se realiza de la siguiente forma: se coge la imagen original y se le aplica una operación de punto con función de resalto, para diferenciar las intensidades bajas (pulmones) de las demás intensidades (resto del tórax). El umbral lse establece en $[30,\ 400]$, según los experimentos de ensayo-error del autor. El resultado no es aceptable del todo, los pulmones contienen ciertos huecos.

Imagen 22. Pulmones tras operación de cambio de contraste.

A continuación, se puede aplicar un contorneo con alguna de las funciones que se conoce, por ejemplo, Sobel.

Imagen 23. Contorneo de los pulmones con contraste.

Este contorneo se almacena, pues se empleará a la hora de contornear las siguientes imágenes. Se utiliza una función de comparar, que no permite que los puntos del nuevo contorno se hayan alejado respecto del antiguo una cierta cantidad, marcada por la variable "radio de acción".

También se aplica a la imagen original una detección de objetos, y un análisis de sus propiedades, para descartar los objetos que no interesan, los que tienen un área más pequeña, los que no están situados en la zona de interés, etc.

Finalmente se aplica un criterio para eliminar los objetos interiores.

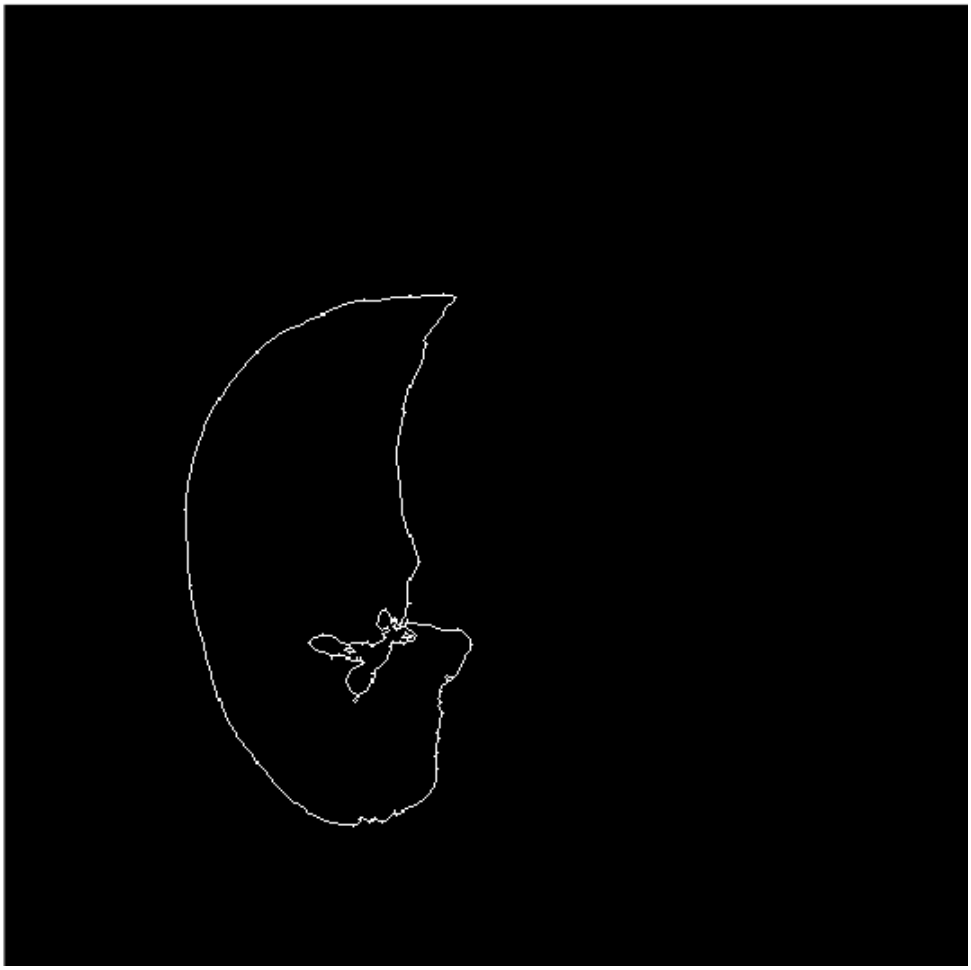


Imagen 24. Pulmón izquierdo bueno.

El paso de colorear este contorneo y añadirlo a la imagen original es inmediato.

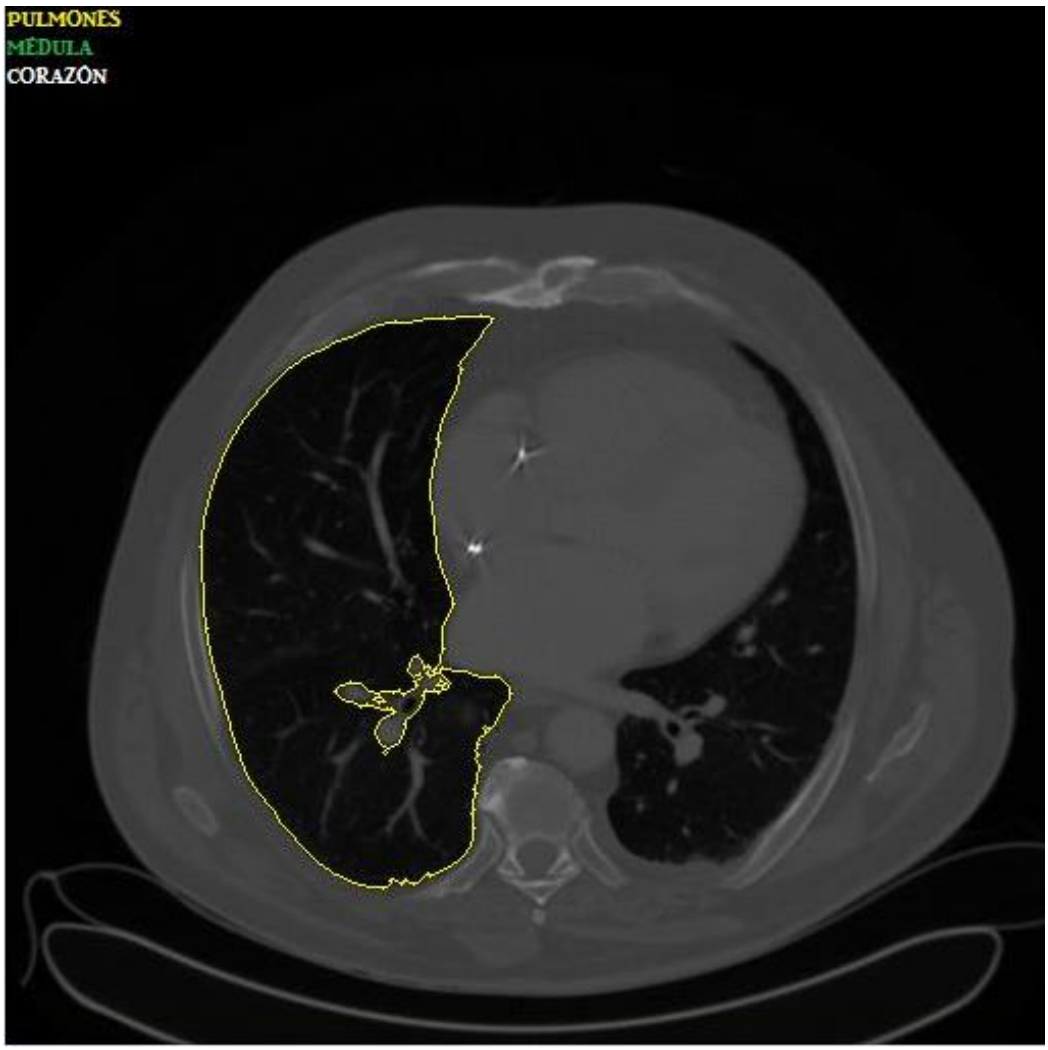


Imagen 25. Imagen con pulmón izquierdo señalado.

5.4. Contorneo del pulmón derecho.

Es muy similar el proceso al contorneo del pulmón izquierdo, simplemente se cambia el área de localización.

5.5. Contorneo de la médula.

En este caso, observando todas las imágenes, se ve que la médula siempre está situada en la misma zona aproximadamente, por ello se recorta la imagen, en un área localizada alrededor de la vértebra.

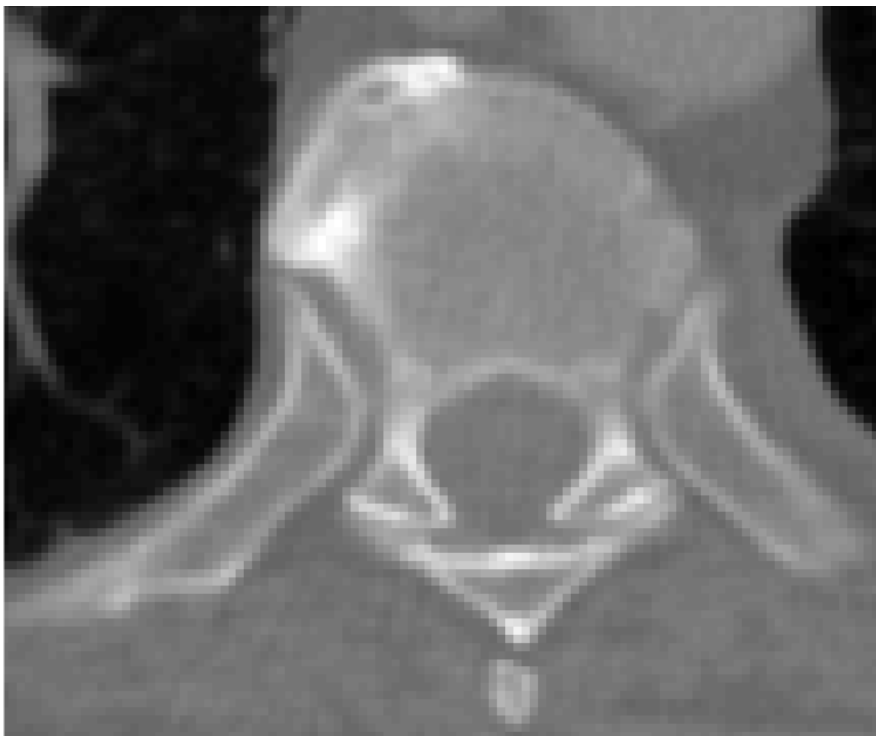


Imagen 26. Zona de la vértebra.

Se aplica una operación de punto para anular las intensidades fuera del rango de interés, en este caso, experimentalmente se obtuvo el rango $[900, 1100]$ como el rango ideal.

Imagen 27. Zona de la vertebra tras filtrado de contraste.

El resultado obtenido es una zona remarcada de la vértebra, y una zona interior aislada, que es la médula que se busca.

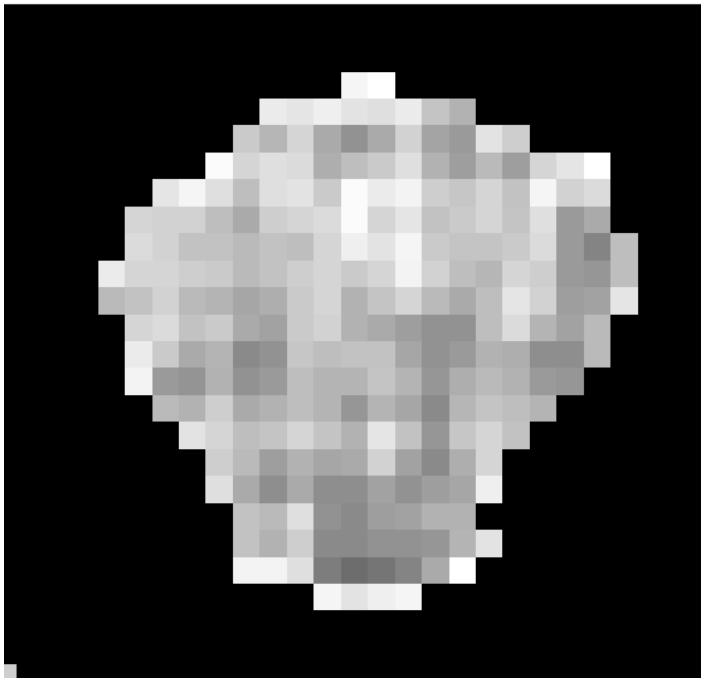


Imagen 28. Zona de la médula.

Se recorta la imagen alrededor de la médula.

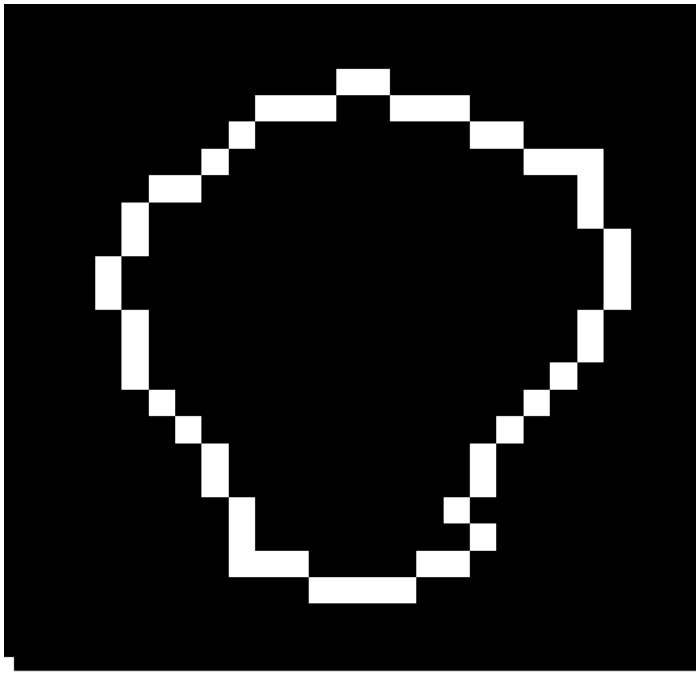


Imagen 29. Médula tras la aplicación de la función "bordes".

Se aplica una función que se queda con los píxeles más cercanos a los extremos, que se desarrolla. El resultado es el mostrado en la imagen.

Se aplica a la imagen general.

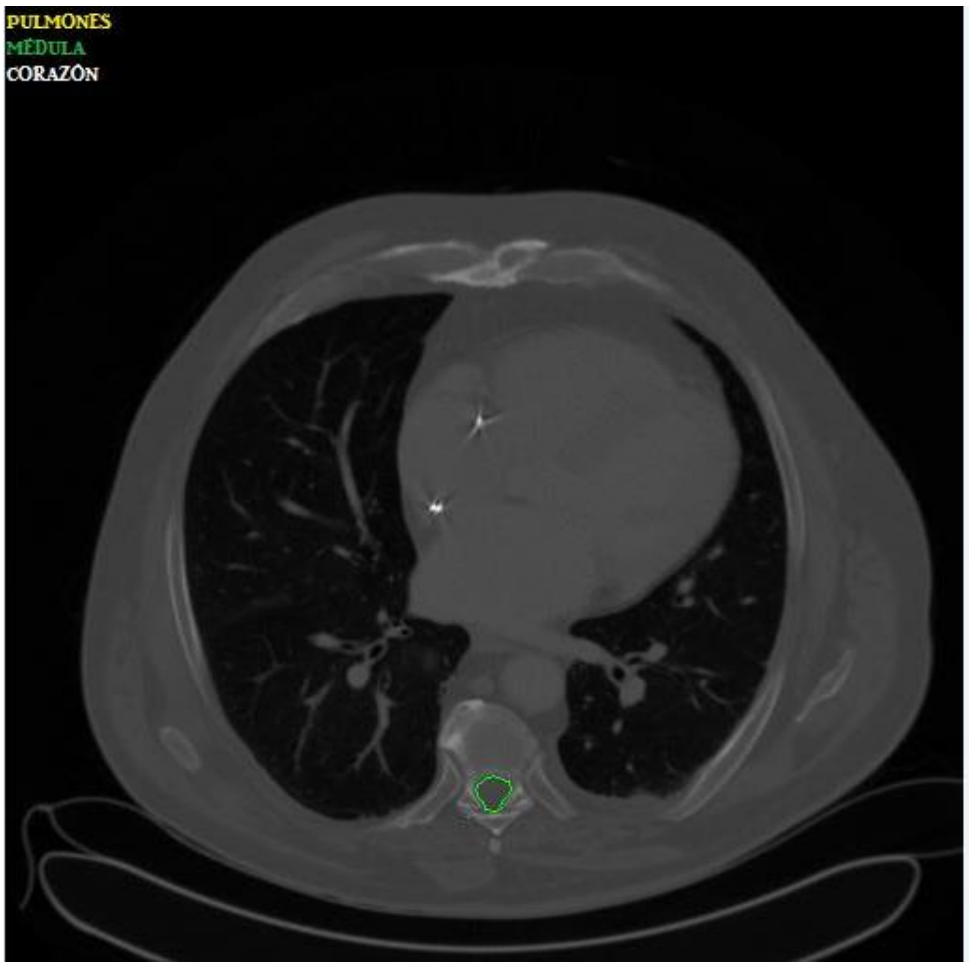


Imagen 30. Imagen con la médula señalada.

Seguidamente, a la hora de segmentar las siguientes imágenes, se va a tener en cuenta que los puntos de una imagen a otra apenas se van a mover, y se utiliza la función comparar para eliminar los puntos que se hayan alejado más de un cierto radio de acción.

Bien es cierto, que la zona de la médula se moverá un poco en dirección vertical, y por ello se tiene que modificar los extremos de recorte.

5.6. Contorneo del corazón.

Para este órgano, primero se recorta la imagen, para centrarse en la zona de interés.

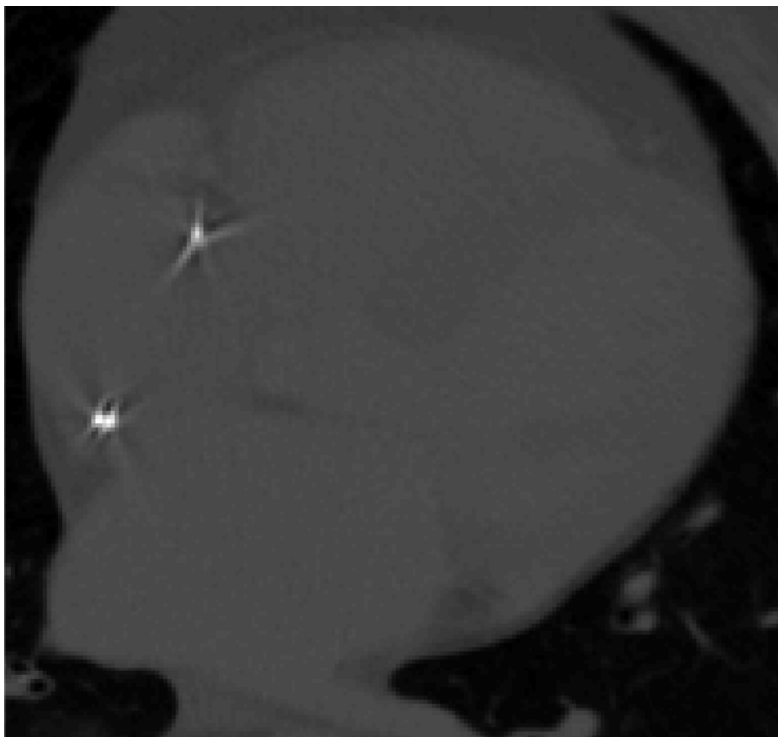


Imagen 31. Zona del corazón.

Posteriormente se aplica una operación de punto a la imagen, de forma que se quede con un rango de intensidad interesante.

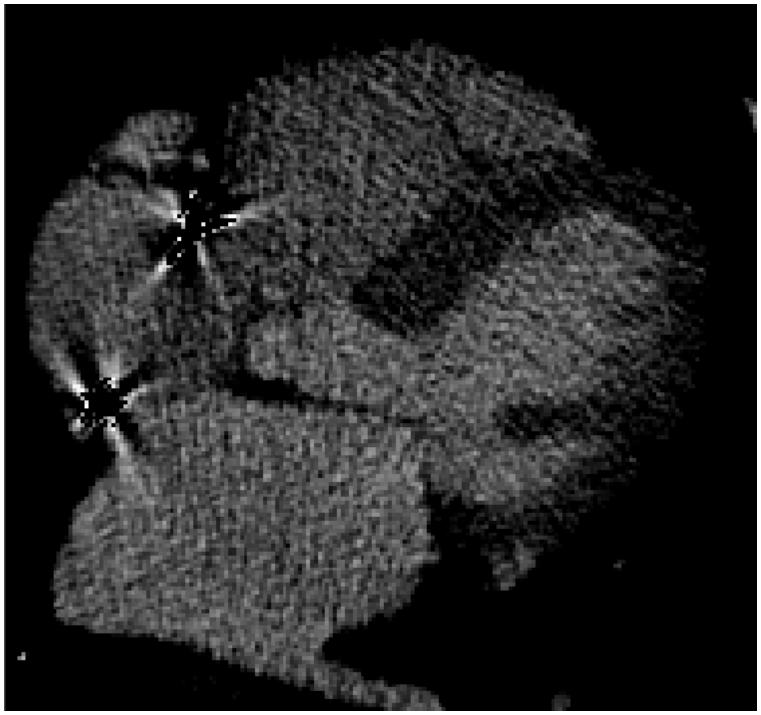


Imagen 32. Zona del corazón filtrada.

A continuación, tras prestar atención a la imagen, se observó que los bordes estaban delimitados, simplemente aplicando una función que busca desde los extremos, de

forma similar a como se hacía para detectar la médula. Cuando encuentra un punto, barriendo por filas y columnas, se pasa a la siguiente.

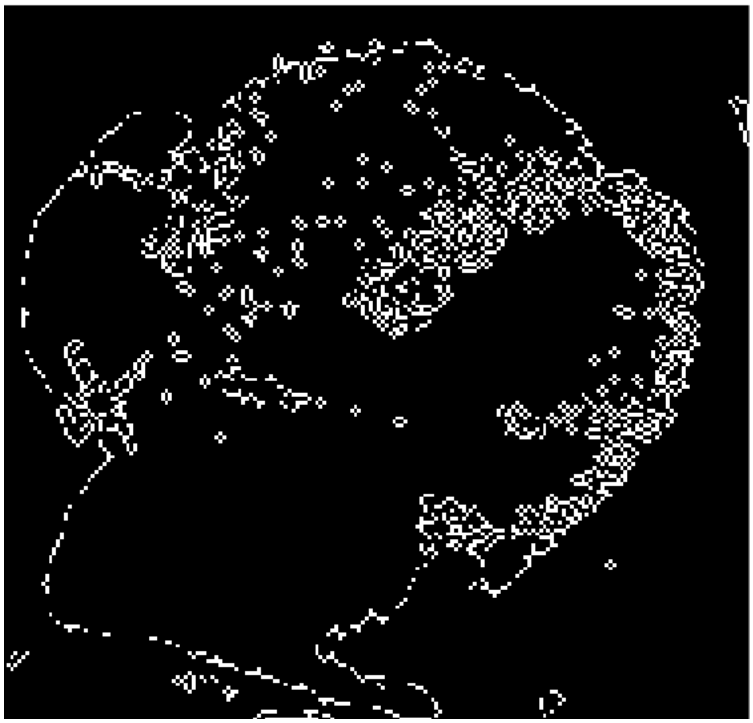


Imagen 33. Zona del corazón contorneada.

La aplicación a diversas imágenes se realiza de la misma forma, utilizando siempre la función "comparar" para detectar que los puntos no se hayan alejado demasiado de una imagen a la inmediata siguiente.

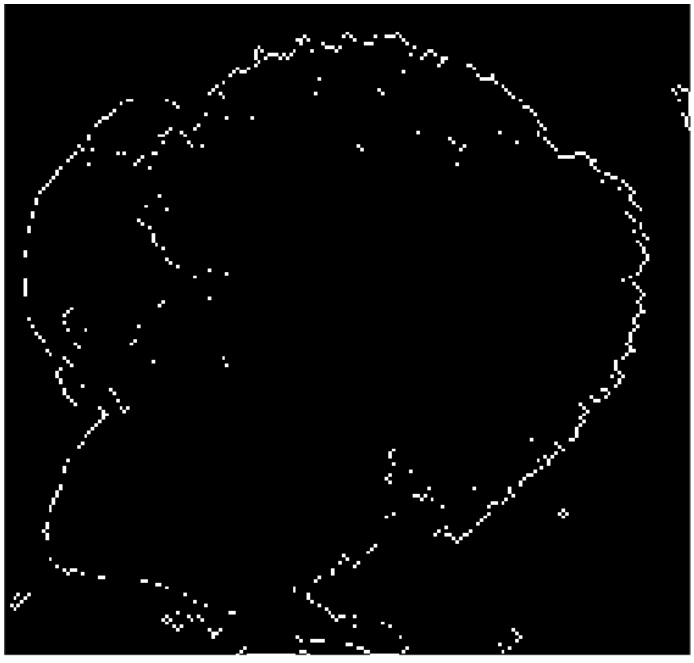


Imagen 34. Zona del corazón tras la aplicación de bordes,

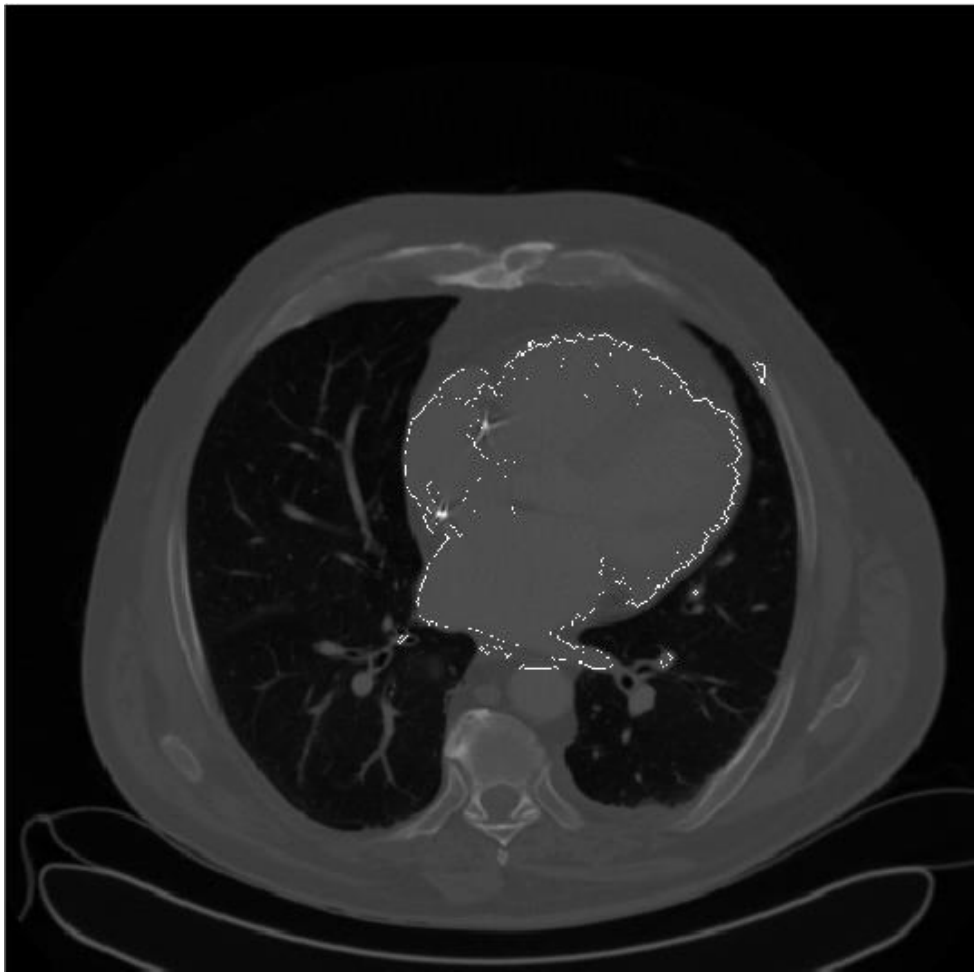


Imagen 35. Imagen del tórax con el corazón señalado.

Como el contorneo queda un poco discontinuo, se aplica una función que se llama "repaso", basada en la "táctica del punto gordo". Se colocan píxeles blancos alrededor de los píxeles blancos.

5.7. Paquete de funciones "myfunctions".

El paquete de funciones personales que ha creado el autor, contiene las funciones que se van a listar a continuación[1].

- **"área_devuelta"**

Esta función coloca una parte de una imagen en otra imagen, o a una imagen en negro del tamaño que se especifique.

- **"área_escogida"**

Esta función recorta un área rectangular de una imagen.

[1] Notar que las remarcadas en negrita son las funciones más importantes.

- "áreas_objetos"

Esta función recoge las áreas de los objetos que contiene la imagen en un vector.

- **"bordes"**

Se queda con los bordes extremos de una imagen binaria.

- "bounding_box"

Recoge las características de las "bounding box" de los objetos que contiene la imagen en vectores.

- "centroide_objetos"

Almacena las coordenadas del "centroide" de los objetos en dos vectores.

- **"colorear"**

Colorea una imagen binaria del color que queramos.

- **"comparar"**

Compara una imagen binaria con otra, y va copiando el valor de todos los puntos de alrededor de los píxeles de valor 1, según un radio de acción.

- **"comparar_eliminar"**

Compara dos imágenes binarias y elimina los puntos de valor 1 que se han alejado demasiado, según un radio de acción.

- **"comparar_negro"**

Compara dos imágenes binarias, y pone a 0 los valores que en la imagen principal son 0.

- **"contornear"**

Contornea una imagen llamando a la función "edge", según los valores que se le indiquen del gradiente, y el método.

- "dif_areas_objects"

Calcula las áreas rellenas de los objetos que contiene una imagen, y las almacena.

- "eliminar_Square"

Elimina un rectángulo de la imagen.

- "eliminar_X"

Elimina una o varias franjas de la imagen en el eje X.

- “eliminar_Y”

Elimina una o varias franjas de la imagen en el eje Y.

- “encontrar”

Busca los píxeles de la imagen que tienen una intensidad comprendida entre un límite inferior y un límite superior, y los almacena en coordenadas.

- **“excluir_areas”**

Excluye los objetos de una imagen que tengan un área fuera del rango que se establezca.

- “excluir_BoBox_xpos”

Excluye los objetos de una imagen cuya “bounding box” tenga una coordenada X fuera del rango que se establezca.

- “excluir_BoBox_ypos”

Excluye los objetos de una imagen cuya “bounding box” tenga una coordenada X fuera del rango que se establezca.

- “excluir_centroide_xpos”

Excluye los objetos de una imagen cuyo centroide tenga una coordenada X fuera del rango que se establezca.

- “excluir_centroide_ypos”

Excluye los objetos de una imagen cuyo centroide tenga una coordenada X fuera del rango que se establezca.

- “excluir_dif_areas”

Excluye los objetos de una imagen cuya área rellena esté fuera del rango que se establezca.

- “excluir_mean_intensity”

Excluye los objetos de una imagen cuya intensidad media esté fuera del que se establezca.

- “excluir_perimeter”

Excluye los objetos de una imagen cuyo perímetro tenga una longitud fuera del rango que se establezca.

- "excluir_posicioncentroide"

Excluye los objetos de una imagen cuyo centroide está situado dentro de un rectángulo que se establezca.

- "excluir_puntosfrontera"

Excluye los objetos de una imagen según el número de puntos de la frontera.

- "fecha"

Devuelve la fecha en formato *"AAAAMMDD_hh.mm.ss"*.

- "filled_Area_objects"

Calcula las áreas rellenas de los objetos.

- **"filtro"**

Realiza una operación de punto a la imagen, dejando los valores comprendidos entre un límite inferior y un límite superior. A los valores fuera del rango, les da el valor del límite inferior.

- "img_inversa"

Calcula la imagen inversa de una imagen.

- **"juntarimg"**

Junta una imagen en escala de grises o color con otra imagen en color, sustituyendo los puntos de la primera por los puntos que tengan color en la segunda.

- "lectura_img"

Almacena los nombres de las imágenes que le indiquemos según números para su posterior lectura.

- "limpieza"

Esta función realiza una limpieza de una imagen en escala de grises, según unos parámetros que se introducen.

- "max_intensity_objects"

Almacena las intensidades máximas de los objetos de una imagen.

- "mean_intensity_objects"

Almacena las intensidades medias de los objetos de una imagen.

- “mostrar_imagen”

Muestra una imagen, llamando a la función “imshow”.

- “perimeter_objects”

Almacena el perímetro de los objetos de una imagen.

- “repaso”

Utiliza la llamada “técnica del punto gordo”: coloca píxeles blancos alrededor de los píxeles blancos de una imagen binaria.

6. Resultados del proyecto.

Como fruto de este desarrollo se tiene un programa capaz de segmentar imágenes médicas. Esto evita la tarea de segmentar las imágenes a mano, con un ‘mouse’, o utilizar un software de alto coste económico. Se va a mostrar como ejemplo las imágenes obtenidas para un paciente.

En este caso se ha trabajado con la segmentación de los cortes transversales de un tórax, distinguiendo tres órganos: pulmones, corazón y médula.

- **Resultados de la segmentación de un tórax.**

Se muestra como resultado varias imágenes del paciente que se ha proporcionado para la investigación.

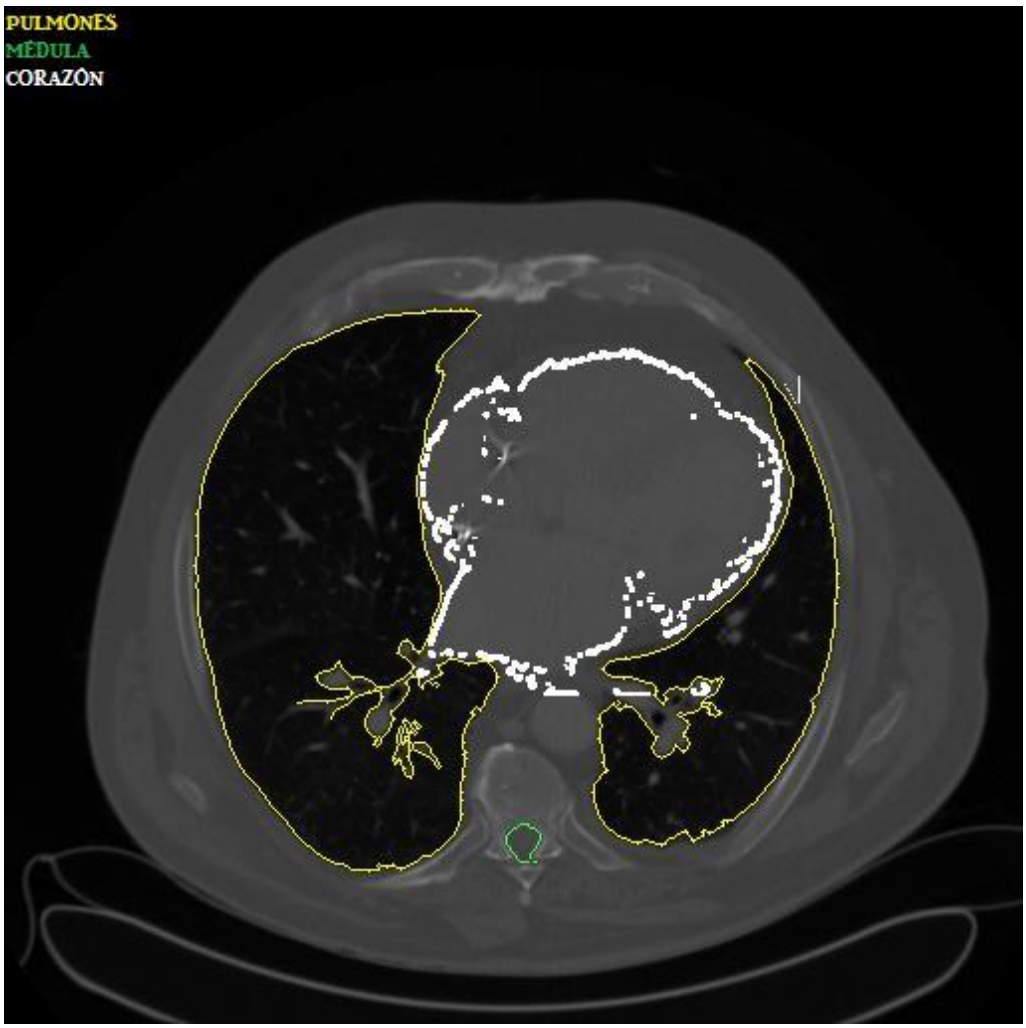


Imagen 36. Imagen del tórax segmentado. (nº102)

Ésta es la imagen que se ha empleado como base.

Conforme se avanza hacia la parte superior del tórax, se deja de observar los pulmones.

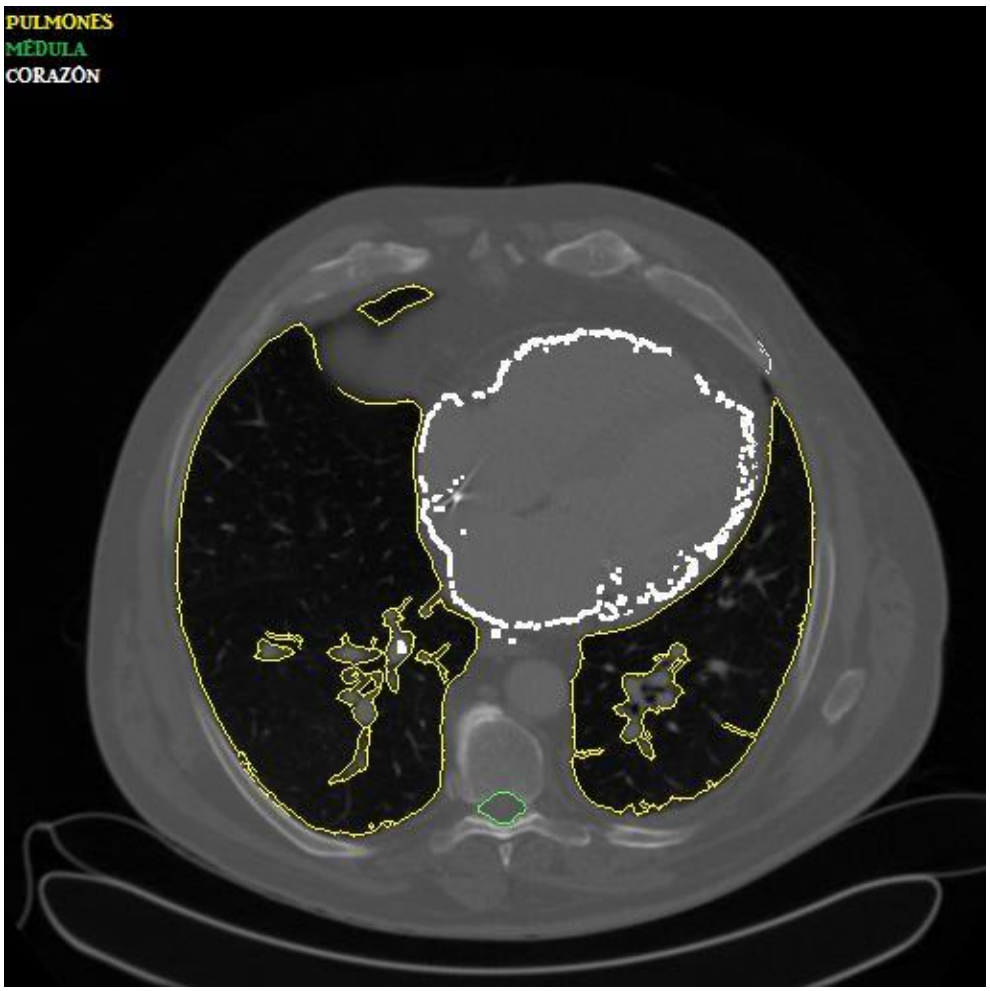


Imagen 37. Imagen del tórax segmentado (nº112).

Esta imagen representa una zona superior. En ella se puede observar que los pulmones disminuyen su tamaño respecto de la anterior imagen.

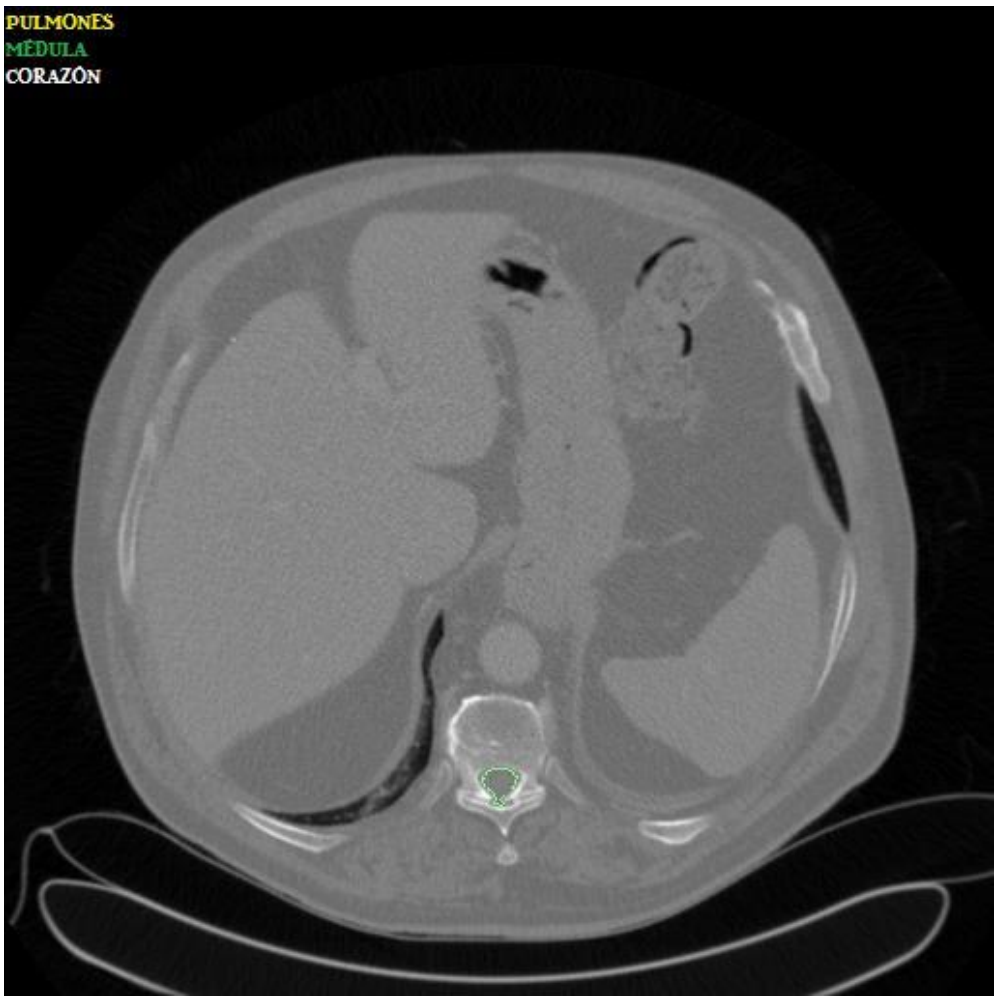


Imagen 38. Imagen del tórax segmentado (nº160).

En esta imagen ya no se distinguen los pulmones ni el corazón. El único órgano destacado en la imagen es la médula.

Las imágenes posteriores son muy similares, por ello pasamos a los cortes inferiores.

En los cortes inferiores, no se observa el corazón:

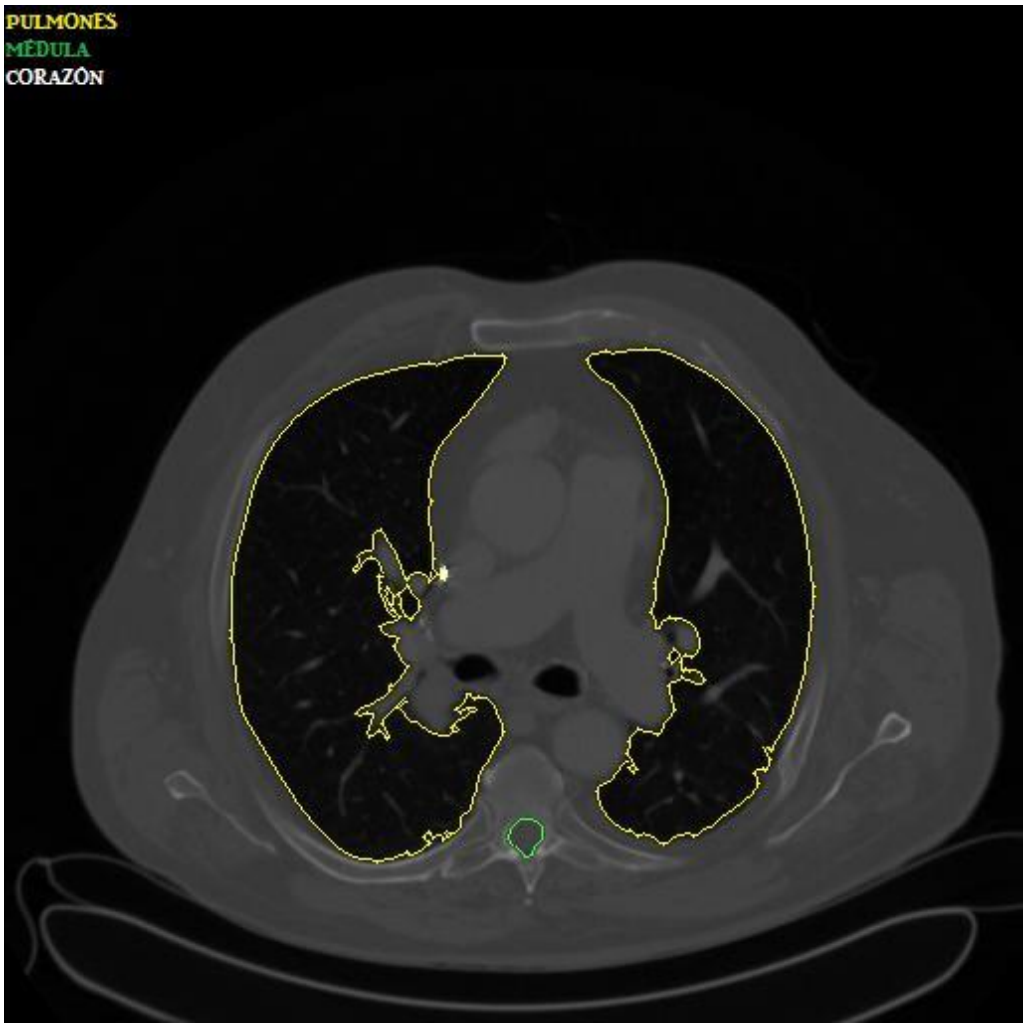


Imagen 39. Imagen del tórax segmentado (nº59).

Los pulmones reducen su tamaño progresivamente, como se puede apreciar en la imagen.

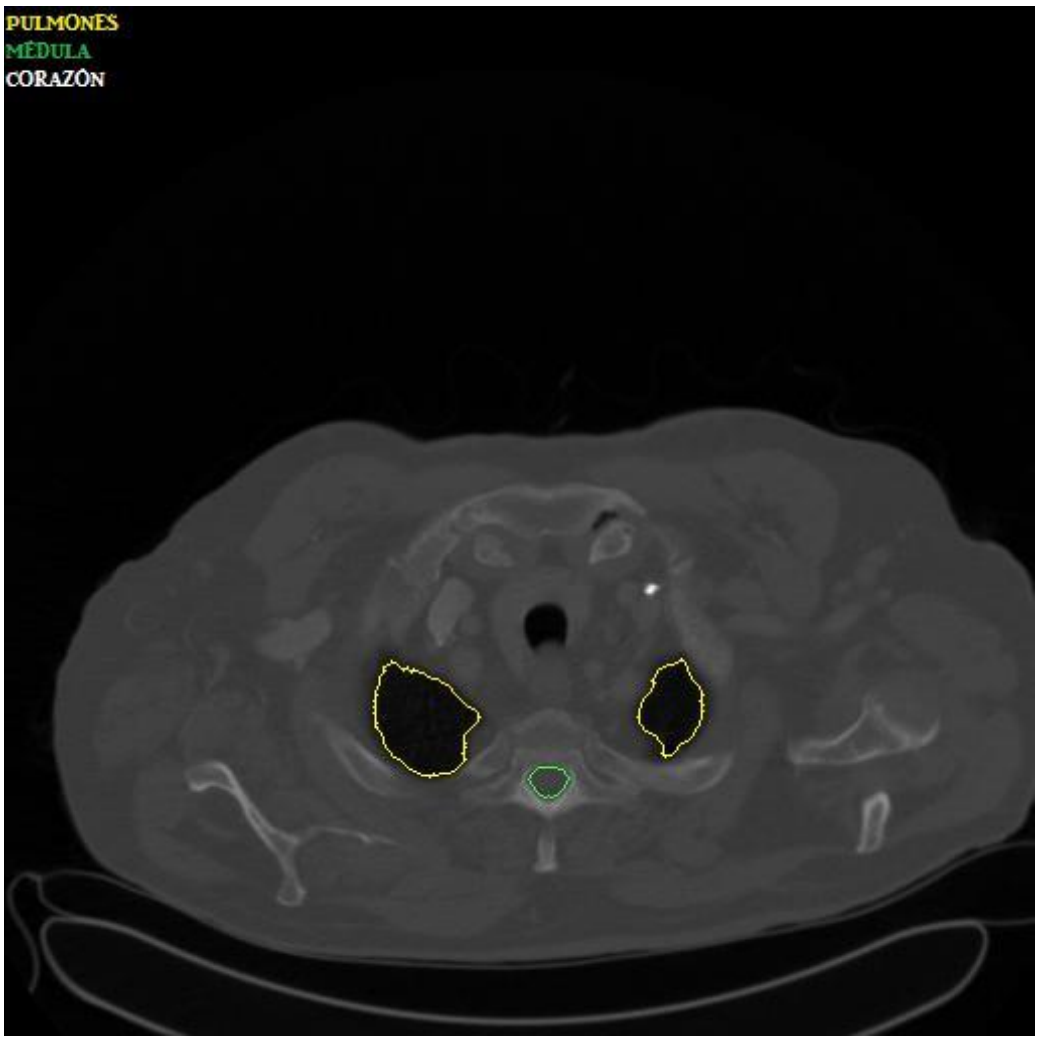


Imagen 40. Imagen del tórax segmentado (nº14).

En esta imagen se observa la parte inferior de los órganos, de tamaño muy reducido.

7. Metas futuras.

El autor ha querido que este proyecto sirva como base para estudios interesantes en el futuro, que puede continuar cualquier persona, siguiendo las indicaciones reflejadas. No hay ningún interés económico en este proyecto, sino el interés del beneficio público.

Algunas de las metas que se propone alcanzar, en opinión del autor, son:

- Reconstrucción tridimensional de los órganos a partir de los cortes bidimensionales de los mismos.
- Reconocer y segmentar otros órganos característicos.
- Detección de anomalías, tumores, etc., mediante dos comparaciones:
  - Una imagen de un órgano con la biblioteca de imágenes sobre dicho órgano.
  - Una imagen de un órgano de un paciente con las imágenes del mismo que se hayan tomado con anterioridad.
- Implementar nuevas subrutinas que aceleren el proceso, y alcanzar el mismo nivel de velocidad de los programas comerciales actuales.
- Convertir el programa en interactivo. Permitir a un usuario con conocimientos básicos de medicina y de programación, saber pedir al programa qué órganos quiere contornear, contestar con la opinión sobre el contorneo, cambiar parámetros, etc.
- Aumentar el grado de automatización del programa, pues actualmente necesita de la inserción de algunos parámetros manualmente.

8. Conclusiones.

Se reflejan a continuación las principales conclusiones de este proyecto:

- Se ha logrado un software libre, capaz de segmentar una imagen médica del tórax, adaptado en este caso al reconocimiento de tres órganos principales, como son los pulmones, el corazón y la médula.
- Se ha previsto una posible reconstrucción tridimensional a partir de estos cortes bidimensionales.
- Este software es gratuito, operable en cualquier dispostivo con una capacidad de procesamiento normal.
- Se proporciona todo el código, dando la facilidad a futuros investigadores para que sigan desarrollando este programa.
- El programa es muy versátil, dando la posibilidad de estudiar cualquier otra región del cuerpo humano, e implementando nuevos factores, para estudios no sólo de segmentación, sino para detección de anomalías en los órganos tratados.

Referencias bibliográficas

1. Universidad Politécnica de Valencia. [En línea] 2014. www.upv.es/titulaciones/GIB/.

2. **Barhoumi, W. y Zagrouba, E.** Towards a Standard Approach for Medical Images Segmentation. [En línea] 2005. http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=1387119&url=http%3A%2F%2Fieeexplore.ieee.org%2Fiel5%2F9525%2F30191%2F01387119.

3. **Brinkkmann, D. H. y Piper, J.** Image registration, deformation, and enhanced contouring for radiotherapy with MIM MaestroTM. [En línea] http://goo.gl/fs6qD8.

4. **Das, B. y Banerjee, S.** Parametric Contour Model in Medical Image Segmentation. [En línea] http://goo.gl/BvJW1W.

5. **Dong, B., Chien, A. y Shen, Z.** Frame based segmentation for medical images. [En línea] http://goo.gl/eDLdsv.

6. **Gao, M., y otros, y otros.** Simplified Labeling Process for Medical Image Segmentation. [En línea]

7. **Huang, X. y Tsechpenakis, G.** Medical Image Segmentation. [En línea] http://goo.gl/Ewsz2K.

8. **Pham, D. L., Xu, C. y Prince, J. L.** A survey of current methods in medical image segmentation. [En línea]

9. **Yang, D., y otros, y otros.** Techniques and software tool for 3D multimodality medical image segmentation. [En línea] 2009. http://jroi.org/index.php/jroi/article/view/4/4.

10. **Zhang, T. Y. y Suen, C. Y.** *"A fast parallel algorithm for thinning digital patterns".* s.l. : Communications of tha ACM, Vol. 27, No.3, 1984.

11. **Sengupta, Prof. Somnath.** Digital Image Processing. [En línea] 1996. https://www.youtube.com/watch?v=q0AnFKYl7sg.

12. **Hervella Azouzi, S.** PFC "Editor de Imágenes basado en Regiones. Aplicación en entorno Matlab.". [En línea] 2006. http://goo.gl/Wl8JUM.

13. Web de descarga de Octave. [En línea] http://www.gnu.org/software/octave/download.html.

14. Web principal de Octave. [En línea] http://octave.sourceforge.net/.

15. Paquetes de Octave. [En línea] http://octave.sourceforge.net/packages.php .

16. Foro de Octave. [En línea] http://octave.1599824.n4.nabble.com/.

Anexo I. Código del programa.

Programa de orden. ("Proy_DEF.m").

```
%Programa que llama a los programas de trabajo.

#####Debemos escribir el directorio##########
%Directorio de búsqueda del archivo:
cd 'C:\\Dropbox\\Dropbox\\PROYECTO\\LAST_PROGRAMS\\'
###################

%Primero borramos todo lo que hay anteriormente en pantalla
clear all
close all
clc

########Seleccionamos las imagenes:##########
%Imagenes que quiero leer:
%Sentido directo:
img_inicial_directo = 100; %Aquí elijo los numeros de las imagenes
que quiero leer.
img_final_directo = 198;
%Sentido inverso:
img_inicial_inverso = 100;
img_final_inverso = 0;
##############################################

%Banderas: (cuando pongo una bandera a 1, no se realiza)
bandera_directa = 0;
bandera_inversa = 0;

%Nos vamos al procesado de las imagenes:
source("20140903_TRABAJO_DEF.m")
```

Programa de trabajo ("20140903_TRABAJO_DEF.m").

```
%Programa que llama a los programas de trabajo.

%Cargamos los paquetes de tratamiento de imágenes.
pkg load dicom;
pkg load image;

if bandera_directa == 0
%Cargamos las imagenes originales.
[A, Num] = lectura_img(img_inicial_directo, img_final_directo);
valor_inicial = 1;
%Directorio de búsqueda del archivo:
cd 'C:\\Dropbox\\Dropbox\\PROYECTO\\LAST_PROGRAMS\\'

%Nos vamos al procesado de las imagenes:
source("PROCESADO_DEF.m")

valor_inicial = size(Num)(1) + 1;
```

```
18
19 else
20 valor_inicial = 1;
21 endif#
22
23 %Cargamos más imágenes:
24
25 if bandera_inversa == 0
26
27 if bandera_directa == 0
28 [A, Num] = lectura_img(img_inicial_inverso, img_final_inverso, A,
   valor_inicial, Num);
29 else
30 [A, Num] = lectura_img(img_inicial_inverso, img_final_inverso);
31 endif#
32
33 %Directorio de búsqueda del archivo:
34 cd 'C:\\Dropbox\\Dropbox\\PROYECTO\\LAST_PROGRAMS\\'
35
36 %Nos vamos al procesado de las imagenes:
37 source("PROCESADO_DEF.m")
38
39 endif#
```

Programa de procesado ("PROCESADO_DEF.m").

```
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %VALORES INICIALES
3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4
5  %Banderas inicializadas:
6  bandera_corazon = 0;
7  bandera_pulmonizq = 0;
8  bandera_pulmondch = 0;
9  bandera_medula = 0;
10 bandera_leyenda = 0;
11 bandera_info = 1;
12 bandera_imagen = 0;
13
14 %Limites del corazón iniciales (experimentales):
15 xmin_cor = 206;
16 xmax_cor = 400;
17 ymin_cor = 165;
18 ymax_cor = 350;
19 %Límites de la médula:
20 xmin_med = 200;
21 xmax_med = 307;
22 ymin_med = 365;
23 ymax_med = 455;
24
25 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26 %BUCLE DE PROCESADO DE LAS IMÁGENES
27 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
28
29 for hh = valor_inicial:size(A)(1)
30
31 %Directorio de búsqueda de imágenes:
32 cd 'C:\\Dropbox\\Dropbox\\PROYECTO\\DICOM_images\\im_2\\'
33 %Cargamos las imágenes:
```

```
34 im_orig = dicomread(A(hh,:));
35 im_cont_orig = contornear(im_orig, 'Sobel', 4, 'both');
36
37 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
38 %LLAMAMOS A LOS PROGRAMAS DE CONTORNEO DE LAS IMÁGENES
39 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
40
41 %Antes de empezar a contornear, inicializamos la imagen final:
42 im_final = im_orig;
43 %Directorio de búsqueda del archivo:
44 cd 'C:\\Dropbox\\Dropbox\\PROYECTO\\LAST_PROGRAMS\\'
45 if (Num(hh) == 125 || Num(hh) == 83)
46 bandera_corazon = 1;
47 endif#
48
49 %Cargamos el archivo de contorneo del corazón:
50 if bandera_corazon == 0
51 source("CORAZON_DEF.m")
52 endif#
53
54 if (Num(hh) == 13 || Num(hh) == 145)
55 bandera_pulmonizq = 1;
56 bandera_pulmondch = 1;
57 endif#
58 %Cargamos el archivo de contorneo de pulmon izquierdo:
59 if bandera_pulmonizq == 0
60 source("PULMONIZQUIERDO_DEF.m")
61 endif#
62 %Cargamos el archivo de contorneo de pulmon derecho:
63 if bandera_pulmondch == 0
64 source("PULMONDERECHO_DEF.m")
65 endif#
66
67 %Cargamos el archivo de contorneo de la médula:
68 if bandera_medula == 0
69 source("MEDULA_DEF.m")
70 endif#
71
72 %Cargo la imagen de leyenda:
73 legend = imread('Legend7.bmp');
74 legend = im2uint16(legend);
75
76 %Coloco la leyenda en la esquina superior izquierda:
77 if bandera_leyenda == 0
78 im_final(1:size(legend)(1), 1:size(legend)(2), :) = legend(:, :,
   :);
79 endif#
80
81 %Directorio de búsqueda de imágenes:
82 cd 'C:\\Dropbox\\Dropbox\\PROYECTO\\DICOM_images\\im_2\\'
83
84 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
85 %MOSTRAR O GUARDAR LAS IMÁGENES
86 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
87
88 %Guardaremos el nombre de archivo con la hora:
89 fecha = fecha();
90
91 xx = num2str(Num(hh)); %Esto me sirve para definir el nombre de
   archivo posteriormente.
92
```

```
93 %Necesito guardar la informacion del paciente:
94 B = dicominfo(A(hh,:));
95 C = disp(B);%Esto es lo que se mostraria por pantalla al llamar a
   B.
96
97 if size(xx)(2) == 3
98 aux = ['Imagen_', 'PMC_', xx];
99 else if size(xx)(2) == 2
100   aux = ['Imagen_', 'PMC_0', xx];
101   else
102   aux = ['Imagen_', 'PMC_00', xx];
103   endif#
104   end
105
106   texto = [aux, '_', fecha];
107
108   %Directorio de guardado de la informacion del archivo:
109   cd 'C:\\Dropbox\\Dropbox\\PROYECTO\\CONTOURNED_IMAGES\\'
110
111   if bandera_info == 0
112   outfile = fopen([texto, '.txt'], "wt");
113   fprintf(outfile, "%s\n", C);
114   fflush(outfile);
115   fclose(outfile);
116   endif#
117
118   if bandera_imagen == 0
119   imwrite(im_final,
   ['C:\\Dropbox\\Dropbox\\PROYECTO\\CONTOURNED_IMAGES\\', texto,
   '.jpg']);
120   endif#
121
122   clear fecha
123
124   endfor#
125
126   %Volvemos al directorio principal de trabajo:
127   cd 'C:\\Dropbox\\Dropbox\\PROYECTO\\LAST_PROGRAMS\\'
```

Programa de contorneo del pulmón izquierdo ("PULMONIZQUIERDO_DEF.m").

```
1  %Prueba:
2  im_prueba = im_orig;


3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4  %BÚSQUEDA DEL PULMÓN IZQUIERDO
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

6  %Operacion de punto para remarcar los pulmones:
7  im_prueba(im_prueba > 30 & im_prueba < 400) = max(max(im_orig));

8  im_cont_prueba = contornear(im_prueba, 'Sobel', 2, 'both');

9  if hh > 1
10 r_acc = 12;
```

```
var_aux = 1; %Para que se elimine el contorneo de la imagen anterior, y no se superpongan.
im_cont_prueba = comparar(contorneo_ultimo_izq, im_cont_prueba, r_acc, var_aux);


%Hago una exclusion por area:
im_cont_prueba = mat2gray(im_cont_prueba);
im_cont_prueba = im2bw(im_cont_prueba, 0.5); %Transformo para que sea una imagen logica.

%Excluyo objetos segun el area:
Area_exc_min = 40;
Area_exc_max = "default";
im_cont_prueba = excluir_areas(im_cont_prueba, Area_exc_min, Area_exc_max);


%Me guardo el contorneo:
contorneo_ultimo_izq = im_cont_prueba;


else

%Defino Áreas de exclusión:
Area_exc_min = 25; ###%Puedo cambiar estas áreas.
Area_exc_max = 1500; ###%

%Defino zonas de exclusion del centroide:
Centroide_xmin = 150;
Centroide_xmax = 440;
Centroide_ymin = 90;
Centroide_ymax = 255;

%Defino limites de la BoundingBox:
Width_min = 1;
Width_max = 2000;
Height_min = 1;
Height_max = 2000;
Coordenada_xmin = 130;
Coordenada_xmax = 440;
Coordenada_ymin = 70;
Coordenada_ymax = 255;

%Defino limites para el perímetro:
Perimeter_min = 25;
Perimeter_max = 1500;

%Excluyo objetos segun el area:
im_nuev = excluir_areas(im_cont_prueba, Area_exc_min, Area_exc_max);

%Excluyo objetos según la posición de la Bounding Box:
im_nuev = excluir_BoBox_xpos(im_nuev, Coordenada_xmin, Coordenada_xmax, im_cont_prueba);
im_nuev = excluir_BoBox_ypos(im_nuev, Coordenada_ymin, Coordenada_ymax, im_cont_prueba);

%Excluyo objetos según su perímetro:
im_nuev = excluir_perimeter(im_nuev, Perimeter_min, Perimeter_max, im_cont_prueba);
```

```
%Excluyo según la posición del centroide:
im_nuev = excluir_centroide_xpos(im_nuev, Centroide_xmin,
Centroide_xmax, im_cont_prueba);
im_nuev = excluir_centroide_ypos(im_nuev, Centroide_ymin,
Centroide_ymax, im_cont_prueba);



%Vuelvo a hallar los objetos:
%[Obj2, num2] = bwlabel(im_nuev);

%Hallo las areas de los objetos:
Area = areas_objetos(im_nuev);

%Ordeno dichas areas:
Area_ordered = sort(Area, 'descend');

%Calculamos la bounding_box:
[Coordenadas, Width, Height] = bounding_box(im_nuev);

for ii = 1:size(Area(Area > 400))(1)
%Posición del objeto más grande:
xh = Area_ordered(ii);
xk = ind2sub(size(Area), find(ismember(Area, xh))); %Me da el
numero de objeto.

%Me voy al objeto de área más grande y defino limites de la
bounding box:
Coordenada_xmin = Coordenadas(xk, 1) + 1;
Coordenada_xmax = Coordenadas(xk, 1) + Width(xk, 1) - 1;
Coordenada_ymin = Coordenadas(xk, 2) + 1;
Coordenada_ymax = Coordenadas(xk, 2) + Height(xk, 1) - 1;

%Realizo una exclusión:
%Excluyo objetos según la posición de la Bounding Box:
im_proceso = excluir_BoBox_xpos(im_nuev, Coordenada_xmin,
Coordenada_xmax);
if max(max(im_proceso)) != 0
im_proceso = excluir_BoBox_ypos(im_proceso, Coordenada_ymin,
Coordenada_ymax);
endif#

%Resultado final:
im_result = im_nuev - im_proceso;

im_nuev = im_result;
endfor#



%Me guardo el contorneo:
contorneo_ultimo_izq = im_nuev;

endif#


if (Num(hh) == 60)
contorneo_ultimo_izq = eliminar_Square(contorneo_ultimo_izq, 234,
280, 320, 350);
endif#
if (Num(hh) < 40 && Num(hh) > 31)
```

```
87 contorneo_ultimo_izq = eliminar_Square(contorneo_ultimo_izq, 246,
   270, 318, 346);
88 endif#


89 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
90 %COLOREAR IMAGEN
91 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

92 RGB_2 = colorear(contorneo_ultimo_izq, "yellow");

93 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
94 %AÑADIR A LA IMAGEN ORIGINAL
95 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

96 %Creo una imagen final:
97 im_final = juntarimg(im_final, RGB_2);
```

Programa de contorneo del pulmón derecho ("PULMONDERECHO_DEF.m").

```
1  %Prueba:
2  im_prueba = im_orig;
3
4
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6  %BÚSQUEDA DEL PULMÓN DERECHO
7  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
8
9  %Operacion de punto para remarcar los pulmones:
10 im_prueba(im_prueba > 30 & im_prueba < 400) = max(max(im_orig));
11
12 im_cont_prueba = contornear(im_prueba, 'Sobel', 2, 'both');
13
14 if hh > 1
15 r_acc = 12;
16 var_aux = 1; %Para que se elimine el contorneo de la imagen
   anterior, y no se superpongan.
17 im_cont_prueba = comparar(contorneo_ultimo_der, im_cont_prueba,
   r_acc, var_aux);
18
19 %Hago una exclusion por area:
20 im_cont_prueba = mat2gray(im_cont_prueba);
21 im_cont_prueba = im2bw(im_cont_prueba, 0.5); %Transformo para que
   sea una imagen logica.
22
23 %Excluyo objetos segun el area:
24 Area_exc_min = 100;
25 Area_exc_max = "default";
26 if max(Area) < Area_exc_max
27 bandera_pulmondch = 1;
28 break
29 endif#
30 im_cont_prueba = excluir_areas(im_cont_prueba, Area_exc_min,
   Area_exc_max);
31
32 %Me guardo el contorneo:
```

```
33 contorneo_ultimo_der = im_cont_prueba;
34
35 else
36
37 %Defino Áreas de exclusión:
38 Area_exc_min = 90; ###%Puedo cambiar estas áreas.
39 Area_exc_max = 1200; ###%
40
41 %Defino zonas de exclusion del centroide:
42 Centroide_xmin = 150;
43 Centroide_xmax = 440;
44 Centroide_ymin = 290;
45 Centroide_ymax = 430;
46
47 %Defino limites de la BoundingBox:
48 Width_min = 1;
49 Width_max = 2000;
50 Height_min = 1;
51 Height_max = 2000;
52 Coordenada_xmin = 130;
53 Coordenada_xmax = 420;
54 Coordenada_ymin = 270;
55 Coordenada_ymax = 420;
56
57 %Defino limites para el perímetro:
58 Perimeter_min = 25;
59 Perimeter_max = 1500;
60
61 %Excluyo objetos segun el area:
62 im_nuev = excluir_areas(im_cont_prueba, Area_exc_min,
   Area_exc_max);
63
64 %Excluyo objetos según la posición de la Bounding Box:
65 im_nuev = excluir_BoBox_xpos(im_nuev, Coordenada_xmin,
   Coordenada_xmax, im_cont_prueba);
66 im_nuev = excluir_BoBox_ypos(im_nuev, Coordenada_ymin,
   Coordenada_ymax, im_cont_prueba);
67
68 %Excluyo objetos según su perímetro:
69 im_nuev = excluir_perimeter(im_nuev, Perimeter_min, Perimeter_max,
   im_cont_prueba);
70
71 %Excluyo según la posición del centroide:
72 im_nuev = excluir_centroide_xpos(im_nuev, Centroide_xmin,
   Centroide_xmax, im_cont_prueba);
73 im_nuev = excluir_centroide_ypos(im_nuev, Centroide_ymin,
   Centroide_ymax, im_cont_prueba);
74
75 if max(max(im_nuev)) == 0
76 bandera_pulmondch = 1;
77 else
78
79 %Hallo las areas de los objetos:
80 Area = areas_objetos(im_nuev);
81
82 %Ordeno dichas areas:
83 Area_ordered = sort(Area, 'descend');
84
85 %Calculamos la bounding_box:
86 [Coordenadas, Width, Height] = bounding_box(im_nuev);
87
```

```
88 for ii = 1:size(Area(Area > 400))(1)
89 %Posición del objeto más grande:
90 xh = Area_ordered(ii);
91 xk = ind2sub(size(Area), find(ismember(Area, xh))); %Me da el
   numero de objeto.
92
93 %Me voy al objeto de área más grande y defino limites de la
   bounding box:
94 Coordenada_xmin = Coordenadas(xk, 1) + 1;
95 Coordenada_xmax = Coordenadas(xk, 1) + Width(xk, 1) - 1;
96 Coordenada_ymin = Coordenadas(xk, 2) + 1;
97 Coordenada_ymax = Coordenadas(xk, 2) + Height(xk, 1) - 1;
98
99 %Realizo una exclusión:
100   %Excluyo objetos según la posición de la Bounding Box:
101   im_proceso = excluir_BoBox_xpos(im_nuev, Coordenada_xmin,
   Coordenada_xmax);
102   if max(max(im_proceso)) != 0
103   im_proceso = excluir_BoBox_ypos(im_proceso, Coordenada_ymin,
   Coordenada_ymax);
104   endif#
105
106   %Resultado final:
107   im_result = im_nuev - im_proceso;
108
109   im_nuev = im_result;
110   endfor#
  111
112   %Me guardo el contorneo:
113   contorneo_ultimo_der = im_nuev;
114
115   endif#
116   endif#
117
118   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
119   %COLOREAR IMAGEN
120   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
121
122   RGB_2 = colorear(contorneo_ultimo_der, "yellow");
123
124   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
125   %AÑADIR A LA IMAGEN ORIGINAL
126   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
127
128   %Creo una imagen final:
129   im_final = juntarimg(im_final, RGB_2);
```

Programa de contorneo de la médula (“MEDULA_DEF.m”).

```
1  %Límites de la médula:
2  if (Num(hh) == 100)
3  xmin_med = 200;
4  xmax_med = 307;
5  ymin_med = 365;
6  ymax_med = 455;
7  endif#
```

```
8
9  %Ajustes de los límites según la imagen (Valores
   experimentales):#################
10 if (Num(hh) == 66 || Num(hh) == 34)
11 ymin_med = ymin_med - 4;
12 ymax_med = ymax_med - 4;
13 xmin_med = xmin_med + 4;
14 xmax_med = xmax_med + 4;
15 endif#
16
17 if (((Num(hh) == 45 || Num(hh) == 30) || (Num(hh) == 25 || Num(hh)
   == 20)) || ((Num(hh) == 55 || Num(hh) == 50) || (Num(hh) == 15 ||
   Num(hh) == 10)))
18 ymin_med = ymin_med - 4;
19 ymax_med = ymax_med - 4;
20 endif#
21
22 if (Num(hh) == 108 || Num(hh) == 124)
23 ymin_med = ymin_med - 3;
24 ymax_med = ymax_med - 3;
25 endif#
26
27 if (((Num(hh) == 138 || Num(hh) == 139) || Num(hh) == 188) ||
   ((Num(hh) == 156 || Num(hh) == 166) || (Num(hh) == 173 || Num(hh)
   == 180)))
28 ymin_med = ymin_med - 4;
29 ymax_med = ymax_med - 4;
30 endif#
31
32 if Num(hh) == 190
33 ymin_med = ymin_med - 4;
34 ymax_med = ymax_med - 4;
35 xmin_med = xmin_med - 3;
36 xmax_med = xmax_med - 3;
37 endif#
38 ##############################################
39
40
41
42 %Seleccion de un area:
43 im_orig_recortada = area_escogida(im_orig, xmin_med, xmax_med,
   ymin_med, ymax_med);
44
45
46 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
47 %APLICAR FILTRO A LA IMAGEN ORIGINAL
48 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
49
50 %Selection of region of interest (respect to INTENSITY)
51 lim_inf = 900;%%%%SE PUEDEN TOCAR ESTOS LÍMITES%%%%%%%%%
52 lim_sup = 1100;%%%%SE PUEDEN TOCAR ESTOS LÍMITES%%%%%%%%%
53 im = filtro(im_orig_recortada, lim_inf, lim_sup);
54
55 %Nuevos límites de recorte:
56 x_min1 = 49;
57 x_max1 = 75;
58 y_min1 = 45;
59 y_max1 = 70;
60
61 im_recortada = area_escogida(im, x_min1,x_max1,y_min1,y_max1);
62
```

```
63
64 %Prueba: Rellanar la imagen filtrada:
65 im_probando = bordes(im_recortada); %Me quedo con los bordes de la
   imagen.
66
67
68 if hh > 1
69 %Eliminación de puntos que no pertenezcan al contorno:
70 r_acc = 3;
71 im_probando = comparar_eliminar(im_probando, ultimo_contorneo_med,
   r_acc);
72
73
74 endif#
75
76 %Me guardo el ultimo contorneo:
77 ultimo_contorneo_med = im_probando;
78
79
80 im_nuev = ultimo_contorneo_med;
81
82
83 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
84 %ADJUNTAR EL RECORTE A LA IMAGEN
85 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
86
87 im = area_devuelta(ultimo_contorneo_med, size(im)(1), size(im)(2),
   x_min1,x_max1,y_min1,y_max1);
88
89 im_nuev = area_devuelta(im, size(im_orig)(1), size(im_orig)(2),
   xmin_med, xmax_med, ymin_med, ymax_med);
90
91 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
92 %COLOREAR IMAGEN
93 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
94
95 im_cont = im_nuev; % Machacamos la antigua imagen contorneada.
96
97 RGB_2 = colorear(im_cont, "green");
98
99 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
100  %AÑADIR A LA IMAGEN ORIGINAL
101  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
102
103  %Creo una imagen final:
104  im_final = juntarimg(im_final, RGB_2);
105
```

Contorneo del corazón (“CORAZON_DEF.m”).

```
1  #Limites del corazón iniciales:
2  if Num(hh) == 100
3  xmin_cor = 206;
4  xmax_cor = 400;
5  ymin_cor = 165;
6  ymax_cor = 350;
7  endif#

8  %Reajuste de límites:
```

```
9   if (((Num(hh) == 102 || Num(hh) == 103) || (Num(hh) == 107 ||
    Num(hh) == 108)) || (Num(hh) == 86 || Num(hh) == 88))
10  ymin_cor = ymin_cor - 3;
11  ymax_cor = ymax_cor - 3;
12  endif#

13  %Seleccion de un area:
14  im_orig_recortada = area_escogida(im_orig, xmin_cor, xmax_cor,
    ymin_cor, ymax_cor);

15  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
16  %APLICAR FILTRO A LA IMAGEN ORIGINAL
17  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

18  %Selection of region of interest (respect to INTENSITY)
19  lim_inf = 1100;%%%%SE PUEDEN TOCAR ESTOS LÍMITES%%%%%%%%%
20  lim_sup = 1300;%%%%SE PUEDEN TOCAR ESTOS LÍMITES%%%%%%%%%
21  im = filtro(im_orig_recortada, lim_inf, lim_sup);

22  %Reestructuro la imagen im, de forma que sea binaria:
23  maximo = max(max(im));

24  im(im > 0) = maximo;

25  im_nuev = contornear(im);

26  %Me quedo con los bordes de la imagen:
27  im = bordes(im_nuev);
28  im = repaso(im);

29  %mostrar_imagen(im);

30  if hh > 1
31  %Eliminación de puntos que no pertenezcan al contorno:
32  r_acc = 3;
33  im = comparar_eliminar(im, ultimo_contorneo_cor, r_acc);

34  endif#

35  ultimo_contorneo_cor = im;

36  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
37  %ADJUNTAR EL RECORTE A LA IMAGEN
38  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

39  im = area_devuelta(im, size(im_orig)(1), size(im_orig)(2) ,
    xmin_cor, xmax_cor, ymin_cor, ymax_cor);

40  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
41  %COLOREAR IMAGEN
42  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

43  RGB_2 = colorear(im, "white");

44  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
45  %AÑADIR A LA IMAGEN ORIGINAL
46  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

47  %Creo una imagen final:
48  im_final = juntarimg(im_final, RGB_2);
```

Anexo II. Código de la biblioteca de funciones desarrollada.

Función "área_devuelta.m"

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_result} =} area_devuelta
   (@var{img}, @var{size_1}, @var{size_2}, @var{xmin}, @var{xmax},
   @var{ymin}, @var{ymax})
3  ## @deftypefnx {Function File} {@var{img_result} =} area_devuelta
   (@var{img}, @var{size_1}, @var{size_2}, @var{xmin}, @var{xmax},
   @var{ymin}, @var{ymax}, @var{img_main})
4  ##
5  ## Esta funcion adjunta una parte de imagen, @var{img}, a una
   imagen completamente
6  ## negra, de dimensiones (@var{size_1} x @var{size_2}). En el caso
   de introducir la
7  ## variable opcional @var{img_main}, se le adjuntaria a esta misma.
   En ese caso las
8  ## variables @var{size_1} y @var{size_2} no se utilizaran.
9  ##
10 ## Variables de entrada:
11 ## @var{img} -> imagen que contiene el area que queremos adjuntar.
12 ## @var{size_1} -> dimension en el eje X de la imagen que vamos a
   crear.
13 ## @var{size_2} -> dimension en el eje Y de la imagen que vamos a
   crear.
14 ## @var{xmin} -> minimo valor de x del rectangulo de colocacion que
   vamos a formar.
15 ## @var{xmax} -> maximo valor de x del rectangulo de colocacion que
   vamos a formar.
16 ## @var{ymin} -> minimo valor de y del rectangulo de colocacion que
   vamos a formar.
17 ## @var{ymax} -> maximo valor de y del rectangulo de colocacion que
   vamos a formar.
18 ## @var{img_main} -> en el caso de introducir esta variable
   opcional, sera la imagen
19 ## a la cual le adjuntaremos el area.
20 ##
21 ## Variables de salida:
22 ## @var{img_result} -> nueva imagen resultado de adjuntar el area
   que hemos introducido
23 ## en una imagen negra, o en la @var{img_main}, en el caso de
   haberse utilizado esta
24 ## variable opcional.
25 ##
26 ## For example:
27 ##
28 ## @example
29 ##
30 ## %Si queremos introducir una imagen recortada IMG en el
   rectangulo de extremos (250, 285),
31 ## %(320, 285), (250, 510), (320, 510) de una imagen IMG_ORIG,
   escribimos:
32 ## img_result = area_devuelta(IMG, "default", "default", 250, 320,
   285, 510, IMG_ORIG);
33 ##
34 ## %Si lo que queremos es introducirlo en una imagen en negro, de
   dimensiones 512 x 580:
35 ## img_result = area_devuelta(IMG, 512, 580, 250, 320, 285, 510);
```

```
36 ##
37 ## @end example
38 ##
39 ## @seealso{area_escogida}
40 ## @end deftypefn
41
42 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
43 ## Created: Jun 17, 2014
44 ## Last modified: Sep 02, 2014
45
46 function img_result = area_devuelta(img, size_1 = "default", size_2
   = "default" , xmin = "default", xmax = "default", ymin = "default",
   ymax = "default", img_main)
47
48 %Inicializacion en caso de meter valores por defecto:
49 if xmin == "default"
50 xmin = 1;
51 endif#
52 if ymin == "default"
53 ymin = 1;
54 endif#
55 if xmax == "default"
56 xmax = size_1;
57 endif#
58 if ymax == "default"
59 ymax = size_2;
60 endif#
61
62 %Caso segun el numero de argumentos de entrada:
63 if nargin < 7
64 error("not adequate number of input arguments")
65 else if nargin ==7
66 %Creo una nueva matriz y le meto el area que indico:
67 img_result = zeros(size_1, size_2);
68 else
69 img_result = img_main;
70 endif#
71 end
72
73 %Algoritmo de colocacion de area:
74 img_result(ymin:ymax, xmin:xmax) = img(1:(ymax - ymin + 1), 1:(xmax
   - xmin + 1));
75
76
77 endfunction
```

Función “área_escogida.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{new_matrix} =} area_escogida
   (@var{img}, @var{xmin}, @var{xmax}, @var{ymin}, @var{ymax})
3  ##
4  ## Esta funcion recorta una imagen @var{img}, y se queda solo con
   un rectangulo
5  ## indicado por los extremos @var{xmin}, @var{xmax}, @var{ymin} e
   @var{ymax}.
```

```
6  ## El resultado es una matriz @var{new_matrix} de menores
   dimensiones.
7  ## Si en alguno de los valores @var{xmin}, @var{xmax}, @var{ymin} e
   @var{ymax}
8  ## introducimos "default", tomaran el valor minimo o maximo de la
   imagen.
9  ##
10 ## Variables de entrada:
11 ## @var{img} -> imagen de la cual queremos extraer un area.
12 ## @var{xmin} -> minimo valor de x del rectangulo de extraccion que
   vamos a formar.
13 ## @var{xmax} -> maximo valor de x del rectangulo de extraccion que
   vamos a formar.
14 ## @var{ymin} -> minimo valor de y del rectangulo de extraccion que
   vamos a formar.
15 ## @var{ymax} -> maximo valor de y del rectangulo de extraccion que
   vamos a formar.
16 ##
17 ## Variables de salida:
18 ## @var{new_matrix} -> nueva matriz que solo contiene el rectangulo
   que hemos
19 ## seleccionado.
20 ##
21 ## For example:
22 ##
23 ## @example
24 ##
25 ## %Si quiero seleccionar en IMG el rectangulo de extremos (250,
   285), (320, 285),
26 ## %(250, 510), (320, 510), escribimos:
27 ## new_matrix = area_escogida(IMG, 250, 320, 285, 510);
28 ##
29 ## @end example
30 ##
31 ## @seealso{area_devuelta}
32 ## @end deftypefn
33
34 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
35 ## Created: Jun 15, 2014
36 ## Last modified: Sep 03, 2014
37
38 function new_matrix = area_escogida(img, xmin = "default", xmax =
   "default", ymin = "default", ymax = "default");
39
40 %Inicializacion en caso de meter valores por defecto:
41 if xmin == "default"
42 xmin = 1;
43 endif#
44 if ymin == "default"
45 ymin = 1;
46 endif#
47 if xmax == "default"
48 xmax = size(img)(1);
49 endif#
50 if ymax == "default"
51 ymax = size(img)(2);
52 endif#
53
54 if nargin != 5
55 error("not adequate number of input arguments")
56 else
```

```
57 %Creo una nueva matriz con el area que indico:
58 new_matrix = zeros((ymax - ymin + 1), (xmax - xmin + 1));
59 new_matrix = img(ymin:ymax, xmin:xmax);
60 endif#
61
62 endfunction
```

Función "áreas_objetos.m"

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{Area} =} areas_objetos
   (@var{img})
3  ##
4  ## Esta funcion almacena las areas de los objetos que contiene la
   imagen @var{img}
5  ## en una matriz @var{Area}.
6  ##
7  ## Variables de entrada:
8  ## @var{img} -> imagen de cuyos objetos queremos conocer el area.
9  ##
10 ## Variables de salida:
11 ## @var{Area} -> matriz que contiene las areas de los objetos.
12 ##
13 ## For example:
14 ##
15 ## @example
16 ##
17 ## %Para almacenar las areas de los objetos que contiene una imagen
   IMG:
18 ## Area = areas_objetos(img);
19 ##
20 ## @end example
21 ##
22 ## @seealso{regionprops, bwlabel, centroide_objetos}
23 ## @end deftypefn
24
25 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
26 ## Created: Jun 15, 2014
27 ## Last modified: Sep 02, 2014
28
29 function Area = areas_objetos(img)
30
31 props = regionprops(img, 'Area');
32 A = struct2cell(props);
33 B = cell2mat(A(1,1,:));
34 Area(:, 1)= B(1, 1, :);
35
36 endfunction
```

Función "bordes.m"

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_result} =} bordes
   (@var{img_binary})
```

```
3  ## @deftypefnx  {Function File} {@var{img_result} =} bordes
   (@var{img_binary}, @var{option})
4  ##
5  ## Esta funcion se queda con los bordes de la imagen binaria
   @var{img_binary}, creando una
6  ## nueva imagen @var{img_result}. Con el parametro @var{option}
   podemos indicar si realizar la
7  ## busqueda de bordes en sentido vertical, horizontal, o ambos (por
   defecto).
8  ##
9  ## La imagen final @var{img_result} binaria sera el resultado
   final.
10 ##
11 ##
12 ## Variables de entrada:
13 ## @var{img_binary} -> imagen binaria a la que queremos buscarle
   los bordes.
14 ## @var{option} -> parametro que me permite indicar el sentido de
   la busqueda de bordes: "horizontal",
15 ## "vertical", "both" (default).
16 ##
17 ##
18 ## Variables de salida:
19 ## @var{img_result} -> imagen final binaria que contiene los
   bordes.
20 ##
21 ## For example:
22 ##
23 ## @example
24 ##
25 ## %Queremos quedarnos con los bordes de una imagen IMG_ORIG:
26 ## IMG_RESULT = bordes(IMG_ORIG);
27 ##
28 ## @end example
29 ##
30 ## @seealso{}
31 ## @end deftypefn
32
33 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
34 ## Created: Sep 03, 2014
35
36 function img_result = bordes(img_binary, option = "both")
37
38 img_result = zeros(size(img_binary));
39
40 if (option == 'both' || option == 'horizontal')
41
42 %Busco por filas empezando por la izquierda:
43 for ii = 1:size(img_binary)(1)
44 for jj = 1:size(img_binary)(2)
45 if img_binary(ii, jj) > 0
46 img_result(ii, jj) = 1;
47 break
48 endif#
49 endfor#
50 endfor#
51
52 %Busco por filas empezando por la derecha:
53 for ii = 1:size(img_binary)(1)
54 for jj = size(img_binary)(2):-1:1
55 if img_binary(ii, jj) > 0
```

```
56 img_result(ii, jj) = 1;
57 break
58 endif#
59 endfor#
60 endfor#
61
62 endif#
63
64 if (option == 'both' || option == 'vertical')
65 %Busco por columnas empezando por arriba:
66 for jj = 1:size(img_binary)(2)
67 for ii = 1:size(img_binary)(1)
68 if img_binary(ii, jj) > 0
69 img_result(ii, jj) = 1;
70 break
71 endif#
72 endfor#
73 endfor#
74
75 %Busco por columnas empezando por abajo:
76 for jj = 1:size(img_binary)(2)
77 for ii = size(img_binary)(1):-1:1
78 if img_binary(ii, jj) > 0
79 img_result(ii, jj) = 1;
80 break
81 endif#
82 endfor#
83 endfor#
84
85 endif#
86
87 endfunction
```

Función “bounding_box.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {[@var{Coordenadas}, @var{Width},
   @var{Height}] =} bounding_box (@var{img})
3  ##
4  ## Esta funcion almacena la BoundingBox de los objetos de la imagen
5  ## @var{img} en unas variables: @var{Coordenadas} para las
   coordenadas del
6  ## extremo izquierdo superior, @var{Width} para la anchura de la
   caja, @var{Height}
7  ## para la altura de la caja.
8  ##
9  ## Variables de entrada:
10 ## @var{img} -> imagen de cuyos objetos queremos conocer la
   BoundingBox.
11 ##
12 ## Variables de salida:
13 ## @var{Coordenadas} -> matriz que contiene los extremos superiores
   izquierdos de
14 ## las cajas. Its size is (num_obj x 2). Guarda las dos
   coordenadas.
15 ## @var{Width} -> Guarda la anchura de los objetos. Its size is
   (num_obj x 1).
```

```
16 ## @var{Height} -> Guarda la altura de los objetos. Its size is
   (num_obj x 1).
17 ##
18 ## For example:
19 ##
20 ## @example
21 ##
22 ## %Para almacenar las BoundingBoxes de los objetos que contiene
   una imagen IMG:
23 ## [Coordenadas, Width, Height] = bounding_box(IMG);
24 ##
25 ## @end example
26 ##
27 ## @seealso{regionprops, bwlabel, areas_objetos, centroide}
28 ## @end deftypefn
29
30 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
31 ## Created: Jul 06, 2014
32 ## Last modified: Sep 02, 2014
33
34 function [Coordenadas, Width, Height] = bounding_box(img)
35
36 props = regionprops(img, "BoundingBox");
37 A = struct2cell(props);
38 B = cell2mat(A(1,1,:));
39
40 Coordenadas(:, 1)= B(1, 1, :);%Coordenadas del extremo izquierdo
   superior.
41 Coordenadas(:, 2)= B(1, 2, :);
42 Width(:, 1)= B(1, 3, :);%Anchura.
43 Height(:, 1)= B(1, 4, :);%Altura.
44
45 endfunction
```

Función “centroide_objetos.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{Centroid} =} centroide_objetos
   (@var{img})
3  ##
4  ## Esta funcion almacena los centroides de los objetos que contiene
   la imagen
5  ## @var{img} en una matriz @var{Centroid}.
6  ##
7  ## Variables de entrada:
8  ## @var{img} -> imagen de cuyos objetos queremos conocer el area.
9  ##
10 ## Variables de salida:
11 ## @var{Centroid} -> matriz que contiene los centroides de los
   objetos. Its size
12 ## is (num_obj x 2). Guarda las dos coordenadas
13 ##
14 ## For example:
15 ##
16 ## @example
17 ##
18 ## %Para almacenar los centroides de los objetos que contiene una
   imagen IMG:
```

```
19 ## Centroid = centroide_objetos(img);
20 ##
21 ## @end example
22 ##
23 ## @seealso{regionprops, bwlabel, areas_objetos}
24 ## @end deftypefn
25
26 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
27 ## Created: Jun 15, 2014
28 ## Last modified: Sep 02, 2014
29
30 function Centroid = centroide_objetos(img)
31
32 props = regionprops(img, 'Centroid');
33 A = struct2cell(props);
34 B = cell2mat(A(1,1,:));
35
36 Centroid(:,1)= B(1, 1, :);
37 Centroid(:,2)= B(1, 2, :);
38
39 endfunction
```

Función “colorear.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_coloured} =} colorear
   (@var{img_orig}, @var{colour})
3  ## @deftypefnx {Function File} {@var{img_coloured} =} colorear
   (@var{img_orig}, @var{R}, @var{G}, @var{B})
4  ##
5  ## Esta funcion colorea la imagen binaria @var{img_orig},
   transformandola
6  ## en una imagen @var{img_coloured} de clase uint16 (valores de 0 a
   65535)
7  ## y formato RGB.
8  ##
9  ## Podemos especificar el color que queremos de los siguientes:
   "red", "green",
10 ## "blue", "yellow", "pink", "orange", "violet", "white", "black".
   O bien
11 ## especificamos la cantidad que queremos de Red, Blue y Green.
   Notar que los
12 ## valores deben ir de 0 a 65535.
13 ##
14 ## Variables de entrada:
15 ## @var{img_orig} -> imagen binaria original.
16 ## @var{colour} -> el color que queremos ("red", "green",
17 ## "blue", "yellow", "pink", "orange", "violet", "white", "black").
18 ## @var{R} -> valor de rojo que queremos (0-65535).
19 ## @var{G} -> valor de verde que queremos (0-65535).
20 ## @var{B} -> valor de azul que queremos (0-65535).
21 ##
22 ## Variables de salida:
23 ## @var{img_coloured} -> imagen coloreada (de clase uint16: 0-
   65535).
24 ##
25 ## For example:
26 ##
```

```
27 ## @example
28 ## @group
29 ##
30 ## %Si queremos colorear la imagen de amarillo:
31 ## img_coloured = colorear(img_orig, "yellow")
32 ## max(max(img_coloured ))
33 ##    @result{} ans(:,:,1) = 65535
34 ##    @result{} ans(:,:,1) = 65535
35 ##    @result{} ans(:,:,1) = 65535
36 ## @end group
37 ##
38 ## @group
39 ## %Si queremos especificar los valores de rojo, verde y azul:
40 ## img_coloured = colorear(img_orig, 35480, 20000, 15756)
41 ## max(max(img_coloured ))
42 ##    @result{} ans(:,:,1) = 35480
43 ##    @result{} ans(:,:,1) = 20000
44 ##    @result{} ans(:,:,1) = 15756
45 ## @end group
46 ##
47 ## @end example
48 ##
49 ## @seealso{}
50 ## @end deftypefn
51
52 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
53 ## Created: Jun 13, 2014
54 ## Last modified: Aug 29, 2014
55
56 function img_coloured = colorear(img_orig, R, G, B)
57
58 %Primero se transforma de matriz(imagen binaria) a imagen gris:
59 I = mat2gray (img_orig);
60
61 %Luego se transforma la imagen binaria a una imagen indexada:
62 [img,map] = gray2ind(I, 2);
63 %Luego se transforma la imagen indexada a una imagen RGB:
64 RGB = ind2rgb(img, map);
65 %Transformamos a clase uint16 (valores 0-65535):
66 img_coloured = im2uint16(RGB);
67
68 if nargin == 1
69 error("You must specify a colour for the image")
70 endif#
71
72 if nargin == 2
73 switch(R)
74 case "red"
75 R = 65535;
76 G = 0;
77 B = 0;
78 case "green"
79 R = 0;
80 G = 65535;
81 B = 0;
82 case "blue"
83 R = 0;
84 G = 0;
85 B = 65535;
86 case "yellow"
87 R = 65535;
```

```
88 G = 65535;
89 B = 0;
90 case "pink"
91 R = 65535;
92 G = 0;
93 B = 65535;
94 case "orange"
95 R = 65535;
96 G = 165 / 255 * 65535;
97 B = 0;
98 case "violet"
99 R = 128 / 255 * 65535;
100   G = 0;
101   B = 128 / 255 * 65535;
102   case "white"
103   R = 65535;
104   G = 65535;
105   B = 65535;
106   case black
107   R = 0;
108   G = 0;
109   B = 0;
110   otherwise error("non defined valid colour")
111   endswitch#
112   endif#
113
114   img_aux = img_coloured;
115   colores = img_coloured; %Creo una matriz de colores.
116   colores(:, :, 1) = R;
117   colores(:, :, 2) = G;
118   colores(:, :, 3) = B;
119
120   img_coloured(img_aux > 0) = colores(img_aux > 0);
121
122 Endfunction
```

Función “comparar.m”

```
1   ## -*- texinfo -*-
2   ## @deftypefn  {Function File} {@var{img_final} =} comparar
    (@var{img_cont}, @var{img_cont_orig}, @var{r_acc})
3   ## @deftypefnx {Function File} {@var{img_final} =} comparar
    (@var{img_cont}, @var{img_cont_orig}, @var{r_acc}, @var{var_aux})
4   ##
5   ## Esta funcion se fija en los puntos blancos de la imagen binaria
    contorneada
6   ## @var{img_cont}. En cada punto contorneado, se va a la imagen
    original contorneada
7   ## binaria @var{img_cont_orig} y copia el valor de todos los puntos
    de su alrededor
8   ## (indicamos cuanto radio de cuadrado queremos con @var{r_acc}).
    Si le damos a la variable
9   ## auxiliar @var{var_aux} el valor 1, entonces creara una imagen
    nueva para guardar los
10  ## puntos de contorneo.
11  ##
12  ## La imagen final binaria @var{img_final} sera el resultado final.
13  ##
```

```
14  ##
15  ## Variables de entrada:
16  ## @var{img_cont} -> imagen contorneada binaria.
17  ## @var{img_cont_orig} -> imagen original contorneada binaria.
18  ## @var{r_acc} -> radio de la comparacion que queremos realizar.
19  ## @var{var_aux} -> si se le da el valor 1, crea una imagen nueva.
    Por defecto vale 0.
20  ##
21  ## Variables de salida:
22  ## @var{img_final} -> imagen final resultado de la comparacion.
23  ##
24  ## For example:
25  ##
26  ## @example
27  ##
28  ## %Las imagenes que tenemos inicialmente tienen un numero de
    puntos contorneados de:
29  ## size(img_cont(img_cont == 1))(1)
30  ##      @result{} ans = 2189
31  ## size(img_cont(img_cont_orig == 1))(1)
32  ##      @result{} ans = 30413
33  ##
34  ## %Ahora realizamos el proceso de comparacion:
35  ## r_acc = 5;
36  ## img_final = comparar(img_cont, img_cont_orig, r_acc)
37  ##
38  ## size(img_final(img_final == 1))(1)
39  ##      @result{} ans = 7536
40  ## %Tiene un numero de puntos contorneados intermedio entre las
    dos.
41  ##
42  ## @end example
43  ##
44  ## @seealso{}
45  ## @end deftypefn
46
47  ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
48  ## Created: Jun 13, 2014
49  ## Last modified: Jul 14, 2014
50
51  function img_final = comparar(img_cont, img_cont_orig, r_acc,
    var_aux = 0)
52
53  img_final = img_cont;
54
55  if var_aux == 1
56  img_final(:, :) = 0;
57  endif#
58
59  for ii = 1:size(img_cont_orig)(1)
60  for jj = 1:size(img_cont_orig)(2)
61  if img_cont(ii, jj) == 1
62  %Defino los limites del cuadrado:
63  min_x = max(1, ii - r_acc);
64  max_x = min(size(img_cont)(1), ii + r_acc);
65  min_y = max(1, jj - r_acc);
66  max_y = min(size(img_cont)(2), jj + r_acc);
67  %Sustituyo los puntos del cuadrado:
68  img_final(min_x:max_x, min_y:max_y) = img_cont_orig(min_x:max_x,
    min_y:max_y);
69  endif#
```

```
70 endfor#
71 endfor#
72
73 endfunction
```

Función “comparar_eliminar.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_final} =} comparar
   (@var{img_cont}, @var{img_cont_orig}, @var{r_acc})
3  ## @deftypefnx {Function File} {@var{img_final} =} comparar
   (@var{img_cont}, @var{img_cont_orig}, @var{r_acc}, @var{precision})
4  ##
5  ## Esta funcion se fija en los puntos blancos de la imagen binaria
   contorneada
6  ## @var{img_cont}. En cada punto contorneado, se va a la imagen
   original contorneada
7  ## binaria @var{img_cont_orig} y busca los puntos de alrededor que
   tienen el valor 1
8  ## (indicamos cuanto radio de cuadrado queremos con @var{r_acc}).
   Si el numero de puntos
9  ## que tienen un valor 1 es menor que el indicado por
   @var{precision}, entonces, ese punto
10 ## se elimina. La precision por defecto permite 3 puntos.
11 ##
12 ## La imagen final binaria @var{img_final} sera el resultado final.
13 ##
14 ## ## Variables de entrada:
15 ## @var{img_cont} -> imagen contorneada binaria.
16 ## @var{img_cont_orig} -> imagen original contorneada binaria.
17 ## @var{r_acc} -> radio de la comparacion que queremos realizar.
18 ## @var{precision} -> numero de puntos minimo que tienen que valer
   1.
19 ##
20 ## Variables de salida:
21 ## @var{img_final} -> imagen final resultado de la comparacion.
22 ##
23 ## For example:
24 ##
25 ## @example
26 ##
27 ## %Las imagenes que tenemos inicialmente tienen un numero de
   puntos contorneados de:
28 ## size(img_cont(img_cont == 1))(1)
29 ##      @result{} ans = 2189
30 ## size(img_cont(img_cont_orig == 1))(1)
31 ##      @result{} ans = 30413
32 ##
33 ## %Ahora realizamos el proceso de comparacion:
34 ## r_acc = 5;
35 ## img_final = comparar(img_cont, img_cont_orig, r_acc)
36 ##
37 ## size(img_final(img_final == 1))(1)
38 ##      @result{} ans = 7536
39 ## %Tiene un numero de puntos contorneados intermedio entre las
   dos.
```

```
40 ##
41 ## @end example
42 ##
43 ## @seealso{}
44 ## @end deftypefn
45
46 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
47 ## Created: Jul 13, 2014
48 ## Last modified: Sep 03, 2014
49
50 function img_final = comparar_eliminar(img_cont, img_cont_orig,
   r_acc, precision = 3)
51
52 img_final = img_cont;
53
54 for ii = 1:size(img_cont_orig)(1)
55 for jj = 1:size(img_cont_orig)(2)
56 if img_cont(ii, jj) == 1
57 min_x = max(1, ii - r_acc);
58 max_x = min(size(img_cont)(1), ii + r_acc);
59 min_y = max(1, jj - r_acc);
60 max_y = min(size(img_cont)(2), jj + r_acc);
61 suma = sum(sum(img_cont_orig(min_x:max_x, min_y:max_y)));
62 if suma < precision
63 img_final(ii, jj) = 0;
64 endif#
65 endif#
66 endfor#
67 endfor#
68
69 endfunction
```

Función “comparar_negro.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_result} =} comparar_negro
   (@var{img_main}, @var{img_second})
3  ##
4  ## Esta funcion coge los valores 0 (negro) de @var{img_main} y se
   los coloca a @var{img_second},
5  ## dando como resultado @var{img_result}. Images must have the same
   dimension.
6  ##
7  ## Input variables:
8  ## @var{img_main} -> blackwhite or binary image taken as main
   image.
9  ## @var{img_second} -> blackwhite or binary image taken as
   secondary image.
10 ##
11 ## Variables de salida:
12 ## @var{img_result} -> imagen resultado tras colocar los pixeles
   negros de @var{img_main}
13 ## en @var{img_second}.
14 ##
15 ## For example:
16 ##
17 ## @example
18 ##
```

```
19 ## %Para sustituir los puntos negros de una imagen A en una imagen
   B:
20 ## img_result = comparar_negro(A, B)
21 ##
22 ## %El resultado sera la antigua imagen B, pero con los pixeles
   negros de A incorporados.
23 ##
24 ## @end example
25 ##
26 ## @seealso{}
27 ## @end deftypefn

28 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
29 ## Created: Jun 14, 2014
30 ## Last modified: Sep 04, 2014

31 function img_result = comparar_negro(img_main, img_second)

32 img_result = img_second;

33 if (size(img_main) != size(img_second))
34 error("img_main and img_second must have the same dimension")
35 else

36 img_result(img_main == 0) = 0;

37 endif#

38 endfunction
```

Función “contornear.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_cont} =} contornear
   (@var{img_orig})
3  ## @deftypefnx {Function File} {@var{img_cont} =} contornear
   (@var{img_orig}, @var{method})
4  ## @deftypefnx {Function File} {@var{img_cont} =} contornear
   (@var{img_orig}, @var{method}, @var{factor_gradiente})
5  ## @deftypefnx {Function File} {@var{img_cont} =} contornear
   (@var{img_orig}, @var{method}, @var{factor_gradiente},
   @var{orientation})
6  ##
7  ## Esta funcion calcula el gradiente medio, y llama a la funcion
   EDGE para contornear
8  ## la imagen original en escala de grises @var{img_orig},
   devolviendo la imagen binaria
9  ## contorneada @var{img_cont}. Los 3 argumentos @var{method},
   @var{factor_gradiente}
10 ## y @var{orientation} son opcionales.
11 ## Si el metodo que metemeos no utiliza el gradiente como
   threshold, debemos utilizar
12 ## la funcion EDGE.
13 ##
14 ## Variables de entrada:
15 ## @var{img_orig} -> imagen en escala de grises que queremos
   contornear.
```

```
16 ## @var{method} -> metodo de contorneo (mirar la funcion EDGE). Por
   defecto utiliza 'Sobel'.
17 ## @var{factor_gradiente} -> factor por el que multiplicamos el
   gradiente. Por defecto 4.
18 ## @var{orientation} -> orientacion en la que miramos el gradiente.
   Por defecto 'Both'.
19 ##
20 ## Variables de salida:
21 ## @var{img_cont} -> imagen contorneada binaria.
22 ##
23 ## For example:
24 ##
25 ## @example
26 ##
27 ## %Si queremos contornear con un factor de gradiente de 8:
28 ## img_cont = contornear(img_orig, 'Sobel', 8);
29 ##
30 ## %Es lo mismo que poner:
31 ## img_cont = edge(img_orig, 'Sobel', 8 * gradMed);
32 ## %Pero ese gradiente medio tenemos que tenerlo previamente
   calculado, que es lo que hace esta funcion.
33 ##
34 ## %Si queremos contornear con el metodo Canny: utilizar la funcion
   EDGE.
35 ##
36 ## @end example
37 ##
38 ## @seealso{edge}
39 ## @end deftypefn

40 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
41 ## Created: Jun 14, 2014

42 function img_cont = contornear(img_orig, method = 'Sobel',
   factor_gradiente = 4, orientation = 'both')%Valores por defecto

43 %Calculo del gradiente medio:
44 [gradX, gradY] = imgradientxy (img_orig);
45 grad = sqrt(gradX .^2 + gradY .^ 2);
46 gradMed = mean(mean(grad));

47 [img_cont,thresh] = edge(img_orig, method, factor_gradiente *
   gradMed, orientation);%AHORA CONTORNEAMOS LA IMAGEN FILTRADA.


48 endfunction
```

Función “dif_Areas_objects.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{dif_Areas} =}
   dif_areas_objects (@var{img})
3  ##
4  ## Esta funcion calcula las diferencias de las areas rellenas de
   los objetos y las
5  ## areas normales de @var{img} y las almacena en @var{dif_Areas}.
6  ##
7  ## Variables de entrada:
```

```
8  ## @var{img} -> imagen de cuyos objetos queremos conocer la
   diferencia de areas.
9  ##
10 ## Variables de salida:
11 ## @var{dif_Areas} -> matriz que contiene la diferencia de areas de
   los objetos.
12 ## Its size is (num_obj x 1).
13 ##
14 ## For example:
15 ##
16 ## @example
17 ##
18 ## %Para almacenar las diferencias de areas de los objetos que
   contiene una imagen IMG:
19 ## dif_Areas = dif_areas_objects(IMG);
20 ##
21 ## @end example
22 ##
23 ## @seealso{regionprops, bwlabel, areas_objetos, centroide,
   bounding_box}
24 ## @end deftypefn

25 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
26 ## Created: Jul 06, 2014

27 function dif_Areas = dif_areas_objects(img)

28 dif_Areas = filled_Area_objects(img) - areas_objetos(img);

29 endfunction
```

Función “eliminar_Square.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_correg} =} eliminar_Square
   (@var{img_orig}, @var{min_x}, @var{max_x}, @var{min_y},
   @var{max_y})
3  ##
4  ## Esta funcion permite eliminar 1 cuadrado de una imagen binaria o
   en
5  ## escala de grises. A los pixeles contenidos en el cuadrado les da
   el
6  ## valor 0 (negro).
7  ## Los valores minimo y maximo del eje X y del eje Y del cuadrado
   que queremos
8  ## eliminar, respectivamente, se indican con @var{min_x} y
   @var{max_x}, para el eje X,
9  ## y @var{min_y}y @var{max_y}, para el eje Y.
10 ##
11 ## Variables de entrada:
12 ## @var{img_orig} -> imagen en escala de grises o binaria a la cual
   queremos eliminarle
13 ## un cuadrado.
14 ## @var{min_x} -> valor minimo del eje X del cuadrado que queremos
   eliminar.
15 ## @var{max_x} -> valor maximo del eje X del cuadrado que queremos
   eliminar.
```

```
16 ## @var{min_y} -> valor minimo del eje Y del cuadrado que queremos
   eliminar.
17 ## @var{max_y} -> valor maximo del eje Y del cuadrado que queremos
   eliminar.
18 ##
19 ## Variables de salida:
20 ## @var{img_correg} -> imagen resultado tras la eliminacion del
   cuadrado.
21 ##
22 ## For example:
23 ##
24 ## @example
25 ##
26 ## %Para eliminar el cuadrado de extremos (285,250), (350, 250),
   (285, 470), (350, 470):
27 ## img_correg = eliminar_Square(img_orig, 285, 350, 250, 470)
28 ##
29 ## @end example
30 ##
31 ## @seealso{eliminar_X, eliminar_Y}
32 ## @end deftypefn

33 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
34 ## Created: Jun 14, 2014
35 ## Last modified: Sep 04, 2014

36 function img_correg = eliminar_Square(img_orig, min_x, max_x,
   min_y, max_y)

37 img_correg = img_orig;

38 if nargin != 5
39 error("inadequate number of input arguments")
40 endif#

41 img_correg(min_y:max_y,min_x:max_x) = 0;

42 endfunction
```

Función “eliminar_X.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_correg} =} eliminar_X
   (@var{img_orig}, @var{min_x1}, @var{max_x1})
3  ## @deftypefnx {Function File} {@var{img_correg} =} eliminar_X
   (@var{img_orig}, @var{min_x1}, @var{max_x1}, @var{min_x2},
   @var{max_x2})
4  ## @deftypefnx {Function File} {@var{img_correg} =} eliminar_X
   (@var{img_orig}, @var{min_x1}, @var{max_x1}, @var{min_x2},
   @var{max_x2}, @var{min_x3}, @var{max_x3})
5  ##
6  ## Esta funcion permite eliminar hasta 3 franjas diferentes de una
   imagen binaria o en
7  ## escala de grises, en el eje X. A los pixeles contenidos en
   dichas franjas les da el
8  ## valor 0 (negro).
```

```
9   ## Los valores minimo y maximo de la franja que queremos eliminar
    se indican con @var{min_x1}
10  ## y @var{max_x1}, respectivamente. Si queremos eliminar mas
    franjas, se emplean los
11  ## argumentos @var{min_x2} y @var{max_x2}, para la franja 2, y
    @var{min_x3} y @var{max_x3}
12  ## para la franja 3.
13  ##
14  ## Variables de entrada:
15  ## @var{img_orig} -> imagen en escala de grises o binaria a la cual
    queremos eliminarle
16  ## una franja en el eje X.
17  ## @var{min_x1} -> valor minimo de X de la franja 1 que queremos
    eliminar.
18  ## @var{max_x1} -> valor maximo de X de la franja 1 que queremos
    eliminar.
19  ## @var{min_x2} -> valor minimo de X de la franja 2 que queremos
    eliminar.
20  ## @var{max_x2} -> valor maximo de X de la franja 2 que queremos
    eliminar.
21  ## @var{min_x3} -> valor minimo de X de la franja 3 que queremos
    eliminar.
22  ## @var{max_x3} -> valor maximo de X de la franja 3 que queremos
    eliminar.
23  ##
24  ## Variables de salida:
25  ## @var{img_correg} -> imagen resultado tras la eliminacion de
    franjas.
26  ##
27  ## For example:
28  ##
29  ## @example
30  ##
31  ## %Para eliminar la franja izquierda de una imagen, por ejemplo,
    entre 1 y 85:
32  ## img_correg = eliminar_X(img_orig, 1, 85);
33  ##
34  ## %Si queremos solamente dejar la franja central, entre 250 y 300:
35  ## img_correg = eliminar_X(img_orig, 1, 249, 301, 512)
36  ##
37  ##
38  ## @end example
39  ##
40  ## @seealso{eliminar_Y, eliminar_Square}
41  ## @end deftypefn

42  ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
43  ## Created: Jun 14, 2014
44  ## Last modified: Sep 04, 2014

45  function img_correg = eliminar_X(img_orig, min_x1, max_x1, min_x2,
    max_x2, min_x3, max_x3)%Valores por defecto

46  img_correg = img_orig;

47  switch nargin
48  case 3
49  img_correg(:, min_x1:max_x1) = 0;
50  case 5
51  img_correg(:, min_x1:max_x1) = 0;
52  img_correg(:, min_x2:max_x2) = 0;
```

```
53  case 7
54  img_correg(:, min_x1:max_x1) = 0;
55  img_correg(:, min_x2:max_x2) = 0;
56  img_correg(:, min_x3:max_x3) = 0;
57  otherwise
58  error("inadequate number of input arguments")
59  endswitch#

60  endfunction
```

Función “eliminar_Y.m”

```
1   ## -*- texinfo -*-
2   ## @deftypefn  {Function File} {@var{img_correg} =} eliminar_Y
    (@var{img_orig}, @var{min_y1}, @var{max_y1})
3   ## @deftypefnx {Function File} {@var{img_correg} =} eliminar_Y
    (@var{img_orig}, @var{min_y1}, @var{max_y1}, @var{min_y2},
    @var{max_y2})
4   ## @deftypefnx {Function File} {@var{img_correg} =} eliminar_Y
    (@var{img_orig}, @var{min_y1}, @var{max_y1}, @var{min_y2},
    @var{max_y2}, @var{min_y3}, @var{max_y3})
5   ##
6   ## Esta funcion permite eliminar hasta 3 franjas diferentes de una
    imagen binaria o en
7   ## escala de grises, en el eje Y. A los pixeles contenidos en
    dichas franjas les da el
8   ## valor 0 (negro).
9   ## Los valores minimo y maximo de la franja que queremos eliminar
    se indican con @var{min_y1}
10  ## y @var{max_y1}, respectivamente. Si queremos eliminar mas
    franjas, se emplean los
11  ## argumentos @var{min_y2} y @var{max_y2}, para la franja 2, y
    @var{min_y3} y @var{max_y3}
12  ## para la franja 3.
13  ##
14  ## Variables de entrada:
15  ## @var{img_orig} -> imagen en escala de grises o binaria a la cual
    queremos eliminarle
16  ## una franja en el eje Y.
17  ## @var{min_y1} -> valor minimo de Y de la franja 1 que queremos
    eliminar.
18  ## @var{max_y1} -> valor maximo de Y de la franja 1 que queremos
    eliminar.
19  ## @var{min_y2} -> valor minimo de Y de la franja 2 que queremos
    eliminar.
20  ## @var{max_y2} -> valor maximo de Y de la franja 2 que queremos
    eliminar.
21  ## @var{min_y3} -> valor minimo de Y de la franja 3 que queremos
    eliminar.
22  ## @var{max_y3} -> valor maximo de Y de la franja 3 que queremos
    eliminar.
23  ##
24  ## Variables de salida:
25  ## @var{img_correg} -> imagen resultado tras la eliminacion de
    franjas.
26  ##
```

```
27 ## For example:
28 ##
29 ## @example
30 ##
31 ## %Para eliminar la franja superior de una imagen, por ejemplo,
   entre 1 y 85:
32 ## img_correg = eliminar_Y(img_orig, 1, 85);
33 ##
34 ## %Si queremos solamente dejar la franja central, entre 250 y 300:
35 ## img_correg = eliminar_Y(img_orig, 1, 249, 301, 512)
36 ##
37 ##
38 ## @end example
39 ##
40 ## @seealso{eliminar_X, eliminar_Square}
41 ## @end deftypefn

42 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
43 ## Created: Jun 14, 2014
44 ## Last modified: Sep 04, 2014

45 function img_correg = eliminar_Y(img_orig, min_y1, max_y1, min_y2,
   max_y2, min_y3, max_y3)%Valores por defecto

46 img_correg = img_orig;

47 switch nargin
48 case 3
49 img_correg(kk, min_y1:max_y1) = 0;
50 case 5
51 img_correg(kk, min_y1:max_y1) = 0;
52 img_correg(kk, min_y2:max_y2) = 0;
53 case 7
54 img_correg(kk, min_y1:max_y1) = 0;
55 img_correg(kk, min_y2:max_y2) = 0;
56 img_correg(kk, min_y3:max_y3) = 0;
57 otherwise
58 error("inadequate number of input arguments")
59 endswitch#


60 endfunction
```

Función “encontrar.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{puntos} =} encontrar
   (@var{img}, @var{lim_inf}, @var{lim_sup})
3  ## @deftypefnx {Function File} {[@var{puntos}, @var{num}] =}
   encontrar (@var{img}, @var{lim_inf}, @var{lim_sup})
4  ##
5  ## Esta funcion busca en la imagen black&white o binaria @var{img},
   los pixeles
6  ## cuyos valores de intensidad esten comprendidos entre
   @var{lim_inf} y @var{lim_sup}.
7  ## Si escribimos "default" en alguno de los limites, tomara el
   valor minimo o maximo.
```

```
8  ## Almacena las coordenadas en la matriz @var{puntos}, donde la
   primera coordenada se
9  ## almacena en la primera columna, y la segunda coordenada en la
   segunda. La variable
10 ## @var{num} opcional devuelve el numero de puntos encontrados.
11 ##
12 ## Variables de entrada:
13 ## @var{img} -> imagen en la que queremos buscar los puntos.
14 ## @var{lim_inf} -> limite inferior de la intensidad buscada.
15 ## @var{lim_sup} -> limite superior de la intensidad buscada.
16 ##
17 ## Variables de salida:
18 ## @var{puntos} -> matriz que almacena las coordenadas de los
   puntos en dos columnas.
19 ## @var{num} -> variable que registra el numero de puntos
   obtenidos.
20 ##
21 ## For example:
22 ##
23 ## @example
24 ##
25 ## %Para hallar los puntos de IMG cuya intensidad se encuentra
   entre 100 y 200:
26 ## [puntos, num] = encontrar(IMG, 100, 200);
27 ##
28 ## %Podemos comprobar la dimension de la nueva matriz generada:
29 ## size(puntos)
30 ##      @result{} ans = 96547      2
31 ## %Esto quiere decir que hay 96547 puntos cuyo valor de intensidad
   se encuentra
32 ## %entre 100 y 200, y sus coordenadas se han almacenado en esa
   matriz.
33 ## %Podemos comprobar que obtenemos el mismo numero de puntos si
   llamamos a la
34 ## %variable num:
35 ## num
36 ##      @result{} ans = 96547
37 ##
38 ## @end example
39 ##
40 ## @seealso{}
41 ## @end deftypefn

42 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
43 ## Created: Jun 20, 2014

44 function [puntos, num] = encontrar(img, lim_inf = "default",
   lim_sup = "default")

45 if lim_inf == "default"
46 lim_inf = 0;
47 endif#
48 if lim_sup == "default"
49 lim_sup = max(max(img));
50 endif#

51 k = 0;
52 for ii = 1:size(img)(1)
53 for jj = 1:size(img)(2)
54 if (img(ii, jj) >= lim_inf && img(ii, jj) <= lim_sup)
55 k = k + 1;
```

```
56 puntos(k, 1) = ii;
57 puntos(k, 2) = jj;
58 else
59 endif#
60 endfor#
61 endfor#

62 num = size(puntos)(1);

63 endfunction
```

Función "excluir_areas.m"

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_result} =} excluir_areas
   (@var{img_main}, @var{Area_exc_min}, @var{Area_exc_max})
3  ## @deftypefnx {Function File} {@var{img_result} =} excluir_areas
   (@var{img_main}, @var{Area_exc_min}, @var{Area_exc_max},
   @var{img_second})
4  ##
5  ## Esta funcion excluye los objetos de la imagen @var{img_main} que
   tienen un
6  ## area fuera del rango @var{Area_exc_min} - @var{Area_exc_max}. Si
   en alguno
7  ## de los valores escribimos "default", lo tomara por el valor
   minimo o maximo.
8  ## En el caso de introducir una imagen @var{img_second}, los
   objetos se buscaran
9  ## en ella, pero se seguiran eliminando en @var{img_main}. El
   resultado es la
10 ## imagen @var{img_result}. All images must have the same
   dimension.
11 ##
12 ## Variables de entrada:
13 ## @var{img_main} -> imagen que queremos limpiar.
14 ## @var{Area_exc_min} -> area de exclusion minima.
15 ## @var{Area_exc_max} -> area de exclusion maxima.
16 ## @var{img_second} -> imagen secundaria. En el caso de
   introducirla, los objetos
17 ## se buscaran en ella.
18 ##
19 ## Variables de salida:
20 ## @var{img_result} -> imagen resultado de la limpieza de objetos.
21 ##
22 ## For example:
23 ##
24 ## @example
25 ##
26 ## %Para eliminar los objetos que tengan un area menor que 300:
27 ## img_result = excluir_areas(img_main, 300, 'default');
28 ##
29 ## @end example
30 ##
31 ## @seealso{excluir_puntosfrontera, regionprops, bwlabel,
   areas_objetos}
32 ## @end deftypefn

33 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
```

```
34 ## Created: Jun 15, 2014
35 ## Last modified: Sep 02, 2014

36 function img_result = excluir_areas(img_main, Area_exc_min =
   "default", Area_exc_max = "default", img_second)

37 %Inicializo:
38 img_result = img_main;

39 %Calculo los objetos de la imagen:
40 if nargin == 3
41 [Obj, num] = bwlabel(img_main);
42 %Calculo las areas de los objetos:
43 Area = areas_objetos(img_main);
44 else
45 [Obj, num] = bwlabel(img_second);
46 Area = areas_objetos(img_second);
47 endif#

48 %;Me guardo las areas en una matriz:
49 Area_mod(2:(size(Area)(1) + 1)) = Area(1:size(Area)(1));
50 Area_mod(1) = 0;

51 if Area_exc_min == "default"
52 Area_exc_min = 1;
53 else
54 endif#
55 if Area_exc_max == "default"
56 Area_exc_max = max(Area);
57 else
58 endif#

59 Areas = Area_mod(Obj + 1);

60 if Area_exc_min >= Area_exc_max
61 error("Area_exc_min must be lower than Area_exc_max")
62 endif#

63 img_result(Areas < Area_exc_min | Areas > Area_exc_max) = 0;

64 endfunction
```

Función “excluir_BoBox_xpos.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_result} =}
   excluir_BoBox_xpos (@var{img_main}, @var{BoBox_xmin},
   @var{BoBox_xmax})
3  ## @deftypefnx {Function File} {@var{img_result} =}
   excluir_BoBox_xpos (@var{img_main}, @var{BoBox_xmin},
   @var{BoBox_xmax}, @var{img_second})
4  ##
5  ## Esta funcion excluye los objetos de la imagen @var{img_main} que
   tienen un
6  ## extremo superior izquierdo de la BoundingBox con una coordenada
   X fuera del rango
```

```
7  ## [@var{BoBox_xmin}, @var{BoBox_xmax}]. Si en alguno de los
   valores escribimos "default",
8  ## lo tomara por el valor minimo o maximo. En el caso de introducir
   una imagen
9  ## @var{img_second}, los objetos se buscaran en ella, pero se
   seguiran eliminando en
10 ## @var{img_main}. El resultado es la  imagen @var{img_result}. All
   images must have
11 ## the same dimension in order not to get a wrong result.
12 ##
13 ## Variables de entrada:
14 ## @var{img_main} -> imagen que queremos limpiar.
15 ## @var{BoBox_xmin} -> posicion x del extremo de la BoundingBox
   minima.
16 ## @var{BoBox_xmax} -> posicion x del extremo de la BoundingBox
   maxima.
17 ## @var{img_second} -> imagen secundaria. En el caso de
   introducirla, los objetos
18 ## se buscaran en ella.
19 ##
20 ## Variables de salida:
21 ## @var{img_result} -> imagen resultado de la limpieza de objetos.
22 ##
23 ## For example:
24 ##
25 ## @example
26 ##
27 ## %Para eliminar los objetos que tengan un extremo de la
   BoundingBox situado fuera del
28 ## %rango [0, 300]:
29 ## img_result = excluir_BoBox_xpos(img_main, 0, 300);
30 ##
31 ## @end example
32 ##
33 ## @seealso{excluir_puntosfrontera, regionprops, bwlabel,
   excluir_areas, bounding_box}
34 ## @end deftypefn

35 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
36 ## Created: Jul 06, 2014
37 ## Last modified: Sep 02, 2014

38 function img_result = excluir_BoBox_xpos(img_main, BoBox_xmin =
   "default", BoBox_xmax = "default", img_second)

39 %Inicializo:
40 img_result = img_main;

41 %Calculo los objetos de la imagen:
42 if nargin == 3
43 [Obj, num] = bwlabel(img_main);
44 %Calculo las areas de los objetos:
45 [Coordenadas, Width, Height] = bounding_box(img_main);
46 else
47 [Obj, num] = bwlabel(img_second);
48 [Coordenadas, Width, Height] = bounding_box(img_second);
49 endif#

50 for ii = 1:size(Coordenadas)(1)
51 CoorX_mod(ii + 1) = Coordenadas(ii, 1);
52 endfor#
```

```
53 CoorX_mod(1) = 0; %Las correspondientes al objeto 0.

54 if BoBox_xmin == "default"
55 BoBox_xmin = 1;
56 else
57 endif#
58 if BoBox_xmax == "default"
59 BoBox_xmax = size(img_main)(1);
60 else
61 endif#

62 Coors = CoorX_mod(Obj + 1);

63 if BoBox_xmin >= BoBox_xmax
64 error("BoBox_xmin must be lower than BoBox_xmax")
65 endif#

66 img_result(Coors < BoBox_xmin | Coors > BoBox_xmax) = 0;

67 endfunction
```

Función “excluir_BoBox_ypos.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_result} =}
   excluir_BoBox_ypos (@var{img_main}, @var{BoBox_ymin},
   @var{BoBox_ymax})
3  ## @deftypefnx {Function File} {@var{img_result} =}
   excluir_BoBox_ypos (@var{img_main}, @var{BoBox_ymin},
   @var{BoBox_ymax}, @var{img_second})
4  ##
5  ## Esta funcion excluye los objetos de la imagen @var{img_main} que
   tienen un
6  ## extremo superior izquierdo de la BoundingBox con una coordenada
   Y fuera del rango
7  ## [@var{BoBox_ymin}, @var{BoBox_ymax}]. Si en alguno de los
   valores escribimos "default",
8  ## lo tomara por el valor minimo o maximo. En el caso de introducir
   una imagen
9  ## @var{img_second}, los objetos se buscaran en ella, pero se
   seguiran eliminando en
10 ## @var{img_main}. El resultado es la  imagen @var{img_result}. All
   images must have
11 ## the same dimension in order not to get a wrong result.
12 ##
13 ## Variables de entrada:
14 ## @var{img_main} -> imagen que queremos limpiar.
15 ## @var{BoBox_ymin} -> posicion y del extremo de la BoundingBox
   minima.
16 ## @var{BoBox_ymax} -> posicion y del extremo de la BoundingBox
   maxima.
17 ## @var{img_second} -> imagen secundaria. En el caso de
   introducirla, los objetos
18 ## se buscaran en ella.
19 ##
20 ## Variables de salida:
21 ## @var{img_result} -> imagen resultado de la limpieza de objetos.
22 ##
```

```
23  ## For example:
24  ##
25  ## @example
26  ##
27  ## %Para eliminar los objetos que tengan un extremo de la
    BoundingBox situado fuera del
28  ## %rango [0, 300]:
29  ## img_result = excluir_BoBox_ypos(img_main, 0, 300);
30  ##
31  ## @end example
32  ##
33  ## @seealso{excluir_BoBox_xpos, bounding_box,
    excluir_puntosfrontera, regionprops, bwlabel, excluir_areas}
34  ## @end deftypefn

35  ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
36  ## Created: Jul 06, 2014

37  function img_result = excluir_BoBox_ypos(img_main, BoBox_ymin =
    "default", BoBox_ymax = "default", img_second)

38  %Inicializo:
39  img_result = img_main;

40  %Calculo los objetos de la imagen:
41  if nargin == 3
42  [Obj, num] = bwlabel(img_main);
43  %Calculo las areas de los objetos:
44  [Coordenadas, Width, Height] = bounding_box(img_main);
45  else
46  [Obj, num] = bwlabel(img_second);
47  [Coordenadas, Width, Height] = bounding_box(img_second);
48  endif#

49  for ii = 1:size(Coordenadas)(1)
50  CoorY_mod(ii + 1) = Coordenadas(ii, 2);
51  endfor#
52  CoorY_mod(1) = 0; %Las correspondientes al objeto 0.

53  if BoBox_ymin == "default"
54  BoBox_ymin = 1;
55  else
56  endif#
57  if BoBox_ymax == "default"
58  BoBox_ymax = size(img_main)(2);
59  else
60  endif#

61  Coors = CoorY_mod(Obj + 1);

62  if BoBox_ymin >= BoBox_ymax
63  error("BoBox_ymin must be lower than BoBox_ymax")
64  endif#

65  img_result(Coors < BoBox_ymin | Coors > BoBox_ymax) = 0;

66  endfunction
```

Función "excluir_centroide_xpos.m"

```
1   ## -*- texinfo -*-
2   ## @deftypefn  {Function File} {@var{img_result} =}
    excluir_centroide_xpos (@var{img_main}, @var{centroide_xmin},
    @var{centroide_xmax})
3   ## @deftypefnx {Function File} {@var{img_result} =}
    excluir_centroide_xpos (@var{img_main}, @var{centroide_xmin},
    @var{centroide_xmax}, @var{img_second})
4   ##
5   ## Esta funcion excluye los objetos de la imagen @var{img_main} que
    tienen un
6   ## centroide con una coordenada X fuera del rango
7   ## [@var{centroide_xmin}, @var{centroide_xmax}]. Si en alguno de
    los valores escribimos "default",
8   ## lo tomara por el valor minimo o maximo. En el caso de introducir
    una imagen
9   ## @var{img_second}, los objetos se buscaran en ella, pero se
    seguiran eliminando en
10  ## @var{img_main}. El resultado es la  imagen @var{img_result}. All
    images must have
11  ## the same dimension in order not to get a wrong result.
12  ##
13  ## Variables de entrada:
14  ## @var{img_main} -> imagen que queremos limpiar.
15  ## @var{centroide_xmin} -> posicion x del centroide minima.
16  ## @var{centroide_xmax} -> posicion x del extremo de la BoundingBox
    maxima.
17  ## @var{img_second} -> imagen secundaria. En el caso de
    introducirla, los objetos
18  ## se buscaran en ella.
19  ##
20  ## Variables de salida:
21  ## @var{img_result} -> imagen resultado de la limpieza de objetos.
22  ##
23  ## For example:
24  ##
25  ## @example
26  ##
27  ## %Para eliminar los objetos que tengan un centroide situado fuera
    del
28  ## %rango [0, 300]:
29  ## img_result = excluir_centroide_xpos(img_main, 0, 300);
30  ##
31  ## @end example
32  ##
33  ## @seealso{excluir_puntosfrontera, regionprops, bwlabel,
    excluir_areas, bounding_box}
34  ## @end deftypefn

35  ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
36  ## Created: Jul 06, 2014
37  ## Last modified: Sep 02, 2014

38  function img_result = excluir_centroide_xpos(img_main,
    centroide_xmin = "default", centroide_xmax = "default", img_second)

39  %Inicializo:
40  img_result = img_main;
```

```
41 %Calculo los objetos de la imagen:
42 if nargin == 3
43 [Obj, num] = bwlabel(img_main);
44 %Calculo las areas de los objetos:
45 Centroide = centroide_objetos(img_main);
46 else
47 [Obj, num] = bwlabel(img_second);
48 Centroide = centroide_objetos(img_second);
49 endif#

50 %Guardo los centroides:
51 Centroid_mod(2:(size(Centroide)(1) + 1)) =
   Centroide(1:size(Centroide)(1), 1);
52 Centroid_mod(1) = 0;

53 if centroide_xmin == "default"
54 centroide_xmin = 1;
55 else
56 endif#
57 if centroide_xmax == "default"
58 centroide_xmax = size(img_main)(1);
59 else
60 endif#

61 if centroide_xmin >= centroide_xmax
62 error("centroide_xmin must be lower than centroide_xmax")
63 endif#

64 %Matriz de centroides:
65 Centroids = Centroid_mod(Obj + 1);

66 img_result(Centroids < centroide_xmin | Centroids > centroide_xmax)
   = 0;

67 endfunction
```

Función “excluir_centroide_ypos.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_result} =}
   excluir_centroide_ypos (@var{img_main}, @var{centroide_ymin},
   @var{centroide_ymax})
3  ## @deftypefnx {Function File} {@var{img_result} =}
   excluir_centroide_ypos (@var{img_main}, @var{centroide_ymin},
   @var{centroide_ymax}, @var{img_second})
4  ##
5  ## Esta funcion excluye los objetos de la imagen @var{img_main} que
   tienen un
6  ## centroide con una coordenada Y fuera del rango
7  ## [@var{centroide_ymin}, @var{centroide_ymax}]. Si en alguno de
   los valores escribimos "default",
8  ## lo tomara por el valor minimo o maximo. En el caso de introducir
   una imagen
9  ## @var{img_second}, los objetos se buscaran en ella, pero se
   seguiran eliminando en
10 ## @var{img_main}. El resultado es la  imagen @var{img_result}. All
   images must have
```

```
11  ## the same dimension in order not to get a wrong result.
12  ##
13  ## Variables de entrada:
14  ## @var{img_main} -> imagen que queremos limpiar.
15  ## @var{centroide_ymin} -> posicion y del centroide minima.
16  ## @var{centroide_ymax} -> posicion y del extremo de la BoundingBox
    maxima.
17  ## @var{img_second} -> imagen secundaria. En el caso de
    introducirla, los objetos
18  ## se buscaran en ella.
19  ##
20  ## Variables de salida:
21  ## @var{img_result} -> imagen resultado de la limpieza de objetos.
22  ##
23  ## For example:
24  ##
25  ## @example
26  ##
27  ## %Para eliminar los objetos que tengan un centroide situado fuera
    del
28  ## %rango [0, 300]:
29  ## img_result = excluir_centroide_ypos(img_main, 0, 300);
30  ##
31  ## @end example
32  ##
33  ## @seealso{excluir_puntosfrontera, regionprops, bwlabel,
    excluir_areas, bounding_box}
34  ## @end deftypefn

35  ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
36  ## Created: Jul 06, 2014
37  ## Last modified: Sep 02, 2014

38  function img_result = excluir_centroide_ypos(img_main,
    centroide_ymin = "default", centroide_ymax = "default", img_second)

39  %Inicializo:
40  img_result = img_main;

41  %Calculo los objetos de la imagen:
42  if nargin == 3
43  [Obj, num] = bwlabel(img_main);
44  %Calculo las areas de los objetos:
45  Centroide = centroide_objetos(img_main);
46  else
47  [Obj, num] = bwlabel(img_second);
48  Centroide = centroide_objetos(img_second);
49  endif#

50  %Guardo los centroides:
51  Centroid_mod(2:(size(Centroide)(1) + 1)) =
    Centroide(1:size(Centroide)(1), 2);
52  Centroid_mod(1) = 0;

53  if centroide_ymin == "default"
54  centroide_ymin = 1;
55  else
56  endif#
57  if centroide_ymax == "default"
58  centroide_ymax = size(img_main)(2);
59  else
```

```
60  endif#

61  if centroide_ymin >= centroide_ymax
62  error("centroide_ymin must be lower than centroide_ymax")
63  endif#

64  %Matriz de centroides:
65  Centroids = Centroid_mod(Obj + 1);

66  img_result(Centroids < centroide_ymin | Centroids > centroide_ymax)
    = 0;

67  endfunction
```

Función “excluir_dif_areas.m”

```
1   ## -*- texinfo -*-
2   ## @deftypefn  {Function File} {@var{img_result} =}
    excluir_dif_areas (@var{img_main}, @var{dif_areas_min},
    @var{dif_areas_max})
3   ## @deftypefnx {Function File} {@var{img_result} =}
    excluir_dif_areas (@var{img_main}, @var{dif_areas_min},
    @var{dif_areas_max}, @var{img_second})
4   ##
5   ## Esta funcion excluye los objetos de la imagen @var{img_main} que
    tienen una diferencia
6   ## de area entre el objeto normal y relleno que esta fuera del
    rango [@var{dif_areas_min},
7   ## @var{dif_areas_max}]. Si en alguno de los valores escribimos
    "default",
8   ## lo tomara por el valor minimo o maximo. En el caso de introducir
    una imagen
9   ## @var{img_second}, los objetos se buscaran en ella, pero se
    seguiran eliminando en
10  ## @var{img_main}. El resultado es la  imagen @var{img_result}. All
    images must have
11  ## the same dimension in order not to get a wrong result.
12  ##
13  ## Variables de entrada:
14  ## @var{img_main} -> imagen que queremos limpiar.
15  ## @var{dif_areas_min} -> minimo valor de diferencia de areas
    permitido.
16  ## @var{dif_areas_max} -> maximo valor de diferencia de areas
    permitido.
17  ## @var{img_second} -> imagen secundaria. En el caso de
    introducirla, los objetos
18  ## se buscaran en ella.
19  ##
20  ## Variables de salida:
21  ## @var{img_result} -> imagen resultado de la limpieza de objetos.
22  ##
23  ## For example:
24  ##
25  ## @example
26  ##
27  ## @end example
```

```
28 ##
29 ## @seealso{excluir_BoBox_xpos, bounding_box,
   excluir_puntosfrontera, regionprops, bwlabel, excluir_areas}
30 ## @end deftypefn

31 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
32 ## Created: Jul 06, 2014
33 ## Last modified: Sep 05, 2014

34 function img_result = excluir_dif_areas(img_main, dif_areas_min =
   "default", dif_areas_max = "default", img_second)

35 %Inicializo:
36 img_result = img_main;

37 %Calculo los objetos de la imagen:
38 if nargin == 3
39 [Obj, num] = bwlabel(img_main);
40 %Calculo las areas de los objetos:
41 dif_Areas = dif_areas_objects(img_main);
42 else
43 [Obj, num] = bwlabel(img_second);
44 dif_Areas = dif_areas_objects(img_second);
45 endif#

46 if dif_areas_min == "default"
47 dif_areas_min = 1;
48 else
49 endif#
50 if dif_areas_max == "default"
51 dif_areas_max = max(dif_Areas);
52 else
53 endif#

54 if dif_areas_min >= dif_areas_max
55 error("dif_areas_min must be lower than dif_areas_max")
56 endif#
57 for ii = 1:size(img_main)(1)
58 for jj = 1:size(img_main)(2)
59 Aux = Obj(ii, jj); %Me guardo el numero de objeto y lo voy
   machacando.
60 if Aux == 0
61 img_result(ii, jj) = 0;
62 else if (dif_Areas(Aux) < dif_areas_min || dif_Areas(Aux) >
   dif_areas_max)
63 img_result(ii, jj) = 0;
64 else
65 %Si no se cumple esto:
66 %El objeto puede ser útil.
67 %No hago nada.
68 end##################No se por que hay que poner este "end".
69 endif#
70 endfor#
71 endfor#

72 endfunction
```

Función "excluir_mean_intensity.m"

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_result} =}
   excluir_mean_intensity (@var{img_main}, @var{img_bw},
   @var{mean_intensity_min}, @var{mean_intensity_max})
3  ## @deftypefnx {Function File} {@var{img_result} =}
   excluir_mean_intensity (@var{img_main}, @var{img_bw},
   @var{mean_intensity_min}, @var{mean_intensity_max},
   @var{img_second})
4  ##
5  ## Esta funcion excluye los objetos de la imagen @var{img_main} que
   tienen una
6  ## intensidad media en la @var{img_bw} fuera del rango
7  ## [@var{mean_intensity_min}, @var{mean_intensity_max}]. Si en
   alguno de los valores
8  ## escribimos "default", lo tomara por el valor minimo o maximo. En
   el caso de introducir
9  ## una imagen @var{img_second}, los objetos se buscaran  en ella,
   pero se seguiran eliminando
10 ## en @var{img_main}. El resultado es la imagen @var{img_result}.
11 ## All images must have the same dimension.
12 ##
13 ## Variables de entrada:
14 ## @var{img_main} -> imagen que queremos limpiar.
15 ## @var{img_bw} -> imagen en la que buscamos la intensidad.
16 ## @var{mean_intensity_min} -> valor de intensidad media minima
   permitida.
17 ## @var{mean_intensity_max} -> valor de intensidad media maxima
   permitida.
18 ## @var{img_second} -> imagen secundaria. En el caso de
   introducirla, los objetos
19 ## se buscaran en ella.
20 ##
21 ## Variables de salida:
22 ## @var{img_result} -> imagen resultado de la limpieza de objetos.
23 ##
24 ## For example:
25 ##
26 ## @example
27 ##
28 ## %Para eliminar los objetos de IMG que tengan una intensidad
   menor que 300 en IMG_BW:
29 ## img_result = excluir_areas(IMG, IMG_BW, 300, 'default');
30 ##
31 ## @end example
32 ##
33 ## @seealso{mean_intensity_objects, excluir_puntosfrontera,
   regionprops, bwlabel, areas_objetos}
34 ## @end deftypefn

35 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
36 ## Created: Jun 15, 2014
37 ## Last modified: Sep 05, 2014
```

```
38 function img_result = excluir_mean_intensity(img_main, img_bw,
   mean_intensity_min = "default", mean_intensity_max = "default",
   img_second)

39 %Inicializo:
40 img_result = img_main;

41 %Calculo los objetos de la imagen:
42 if nargin == 3
43 [Obj, num] = bwlabel(img_main);
44 %Calculo las areas de los objetos:
45 mean_intensity = mean_intensity_objects(img_main, img_bw);
46 else
47 [Obj, num] = bwlabel(img_second);
48 mean_intensity = mean_intensity_objects(img_second, img_bw);
49 endif#

50 if mean_intensity_min == "default"
51 mean_intensity_min = 1;
52 else
53 endif#
54 if mean_intensity_max == "default"
55 mean_intensity_max = max(Area);
56 else
57 endif#

58 if mean_intensity_min >= mean_intensity_max
59 error("mean_intensity_min must be lower than mean_intensity_max")
60 endif#
61 for ii = 1:size(img_main)(1)
62 for jj = 1:size(img_main)(2)
63 Aux = Obj(ii, jj); %Me guardo el numero de objeto y lo voy
   machacando.
64 if Aux == 0
65 img_result(ii, jj) = 0;
66 else if (mean_intensity(Aux) < mean_intensity_min ||
   mean_intensity(Aux) > mean_intensity_max)
67 img_result(ii, jj) = 0;
68 else
69 %Si no se cumple esto:
70 %El objeto puede ser útil.
71 %No hago nada.
72 end#################No se por que hay que poner este "end".
73 endif#
74 endfor#
75 endfor#

76 endfunction
```

Función “excluir_perimeter.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{img_result} =}
   excluir_perimeter (@var{img_main}, @var{Perimeter_min},
   @var{Perimeter_max})
```

```
3  ## @deftypefnx {Function File} {@var{img_result} =}
   excluir_perimeter (@var{img_main}, @var{Perimeter_min},
   @var{Perimeter_max}, @var{img_second})
4  ##
5  ## Esta funcion excluye los objetos de la imagen @var{img_main} que
   tienen un
6  ## perimetro fuera del rango[@var{Perimeter_min},
   @var{Perimeter_max}].
7  ## Si en alguno de los valores escribimos "default",  lo tomara por
   el valor minimo
8  ## o maximo. En el caso de introducir una imagen @var{img_second},
   los objetos se
9  ## buscaran en ella, pero se seguiran eliminando en @var{img_main}.
   El resultado es
10 ## la  imagen @var{img_result}. All images must have  the same
   dimension in order not
11 ## to get a wrong result.
12 ##
13 ## Variables de entrada:
14 ## @var{img_main} -> imagen que queremos limpiar.
15 ## @var{Perimeter_min} -> valor minimo de perimetro permitido.
16 ## @var{Perimeter_max} -> valor maximo de perimetro permitido.
17 ## @var{img_second} -> imagen secundaria. En el caso de
   introducirla, los objetos
18 ## se buscaran en ella.
19 ##
20 ## Variables de salida:
21 ## @var{img_result} -> imagen resultado de la limpieza de objetos.
22 ##
23 ## For example:
24 ##
25 ## @example
26 ##
27 ## @end example
28 ##
29 ## @seealso{excluir_BoBox_xpos, bounding_box,
   excluir_puntosfrontera, regionprops, bwlabel, excluir_areas}
30 ## @end deftypefn

31 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
32 ## Created: Jul 06, 2014
33 ## Last modified: Sep 05, 2014

34 function img_result = excluir_perimeter(img_main, Perimeter_min =
   "default", Perimeter_max = "default", img_second)

35 %Inicializo:
36 img_result = img_main;

37 %Calculo los objetos de la imagen:
38 if nargin == 3
39 [Obj, num] = bwlabel(img_main);
40 %Calculo las areas de los objetos:
41 perimeter = perimeter_objects(img_main);
42 else
43 [Obj, num] = bwlabel(img_second);
44 perimeter = perimeter_objects(img_second);
45 endif#

46 %Me guardo los perimetros en una matriz:
```

```
47 Perim_mod(2:(size(perimeter)(1) + 1)) =
   perimeter(1:size(perimeter)(1));
48 Perim_mod(1) = 0;

49 if Perimeter_min == "default"
50 Perimeter_min = 1;
51 else
52 endif#
53 if Perimeter_max == "default"
54 Perimeter_max = max(perimeter);
55 else
56 endif#

57 if Perimeter_min >= Perimeter_max
58 error("Perimeter_min must be lower than Perimeter_max")
59 endif#

60 %Matriz de perimetros:
61 Perims = Perim_mod(Obj + 1);

62 img_result(Perims < Perimeter_min | Perims > Perimeter_max) = 0;

63 endfunction
```

Función “fecha.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{fecha} =} fecha ()
3  ##
4  ## Esta funcion devuelve la fecha actual en formato de salida:
5  ## "AAAAMMDD_hh.mm.ss"
6  ##
7  ## La fecha @var{fecha} sera el resultado final.
8  ##
9  ##
10 ## Variables de entrada:
11 ##
12 ## Variables de salida:
13 ## @var{fecha} -> fecha actual.
14 ##
15 ## For example:
16 ##
17 ## @example
18 ##
19 ## %Queremos obtener la fecha actual:
20 ## fecha = fecha()
21 ##
22 ## @end example
23 ##
24 ## @seealso{}
25 ## @end deftypefn

26 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
27 ## Created: Sep 03, 2014

28 function fecha = fecha()
```

```
29 %Guardaremos el nombre de archivo con la hora:
30 B = localtime(time());
31 C = struct2cell(B);
32 year = cell2mat(C(7, 1));
33 year = year + 1900;
34 year = num2str(year);
35 month = cell2mat(C(6, 1));
36 month = month + 1;
37 month = num2str(month);
38 if size(month)(2) == 1
39 month = ['0', month];
40 endif#
41 day = cell2mat(C(5, 1));
42 day = num2str(day);
43 if size(day)(2) == 1
44 day = ['0', day];
45 endif#
46 hour = cell2mat(C(4, 1));
47 hour = num2str(hour);
48 if size(hour)(2) == 1
49 hour = ['0', hour];
50 endif#
51 minute = cell2mat(C(3, 1));
52 minute = num2str(minute);
53 if size(minute)(2) == 1
54 minute = ['0', minute];
55 endif#
56 second = cell2mat(C(2, 1));
57 second = num2str(second);
58 if size(second)(2) == 1
59 second = ['0', second];
60 endif#

61 fecha = [year, month, day, '_', hour, '.', minute, '.', second];

62 endfunction
```

Función “filled_Area_objects.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{filled_Area} =}
   filled_Area_objects (@var{img})
3  ##
4  ## Esta funcion calcula las areas rellenas de los objetos de la
   imagen
5  ## @var{img} y las almacena en @var{filled_Area}.
6  ##
7  ## Variables de entrada:
8  ## @var{img} -> imagen de cuyos objetos queremos conocer las areas
   rellenas.
9  ##
10 ## Variables de salida:
11 ## @var{filled_Area} -> matriz que contiene las areas rellenas de
   los objetos.
12 ## Its size is (num_obj x 1).
```

```
13 ##
14 ## For example:
15 ##
16 ## @example
17 ##
18 ## %Para almacenar las areas rellenas de los objetos que contiene
   una imagen IMG:
19 ## filled_Area = filled_Area_objects(IMG);
20 ##
21 ## @end example
22 ##
23 ## @seealso{regionprops, bwlabel, areas_objetos, centroide,
   bounding_box}
24 ## @end deftypefn

25 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
26 ## Created: Jul 06, 2014
27 ## Last modified: Sep 05, 2014

28 function filled_Area = filled_Area_objects(img)

29 props = regionprops(img, "filledArea");
30 A = struct2cell(props);
31 B = cell2mat(A(1,1,:));

32 filled_Area(:, 1) = B(1, 1, :);%Area rellena del objeto.

33 endfunction
```

Función “filtro.m”

```
1   ## -*- texinfo -*-
2   ## @deftypefn  {Function File} {@var{img_filt} =} mpoles
    (@var{img}, @var{lim_inf}, @var{lim_sup})
3   ##
4   ## Esta funcion filtra la imagen @var{img} segun la intensidad,
    dejando
5   ## solo los valores comprendidos entre lim_inf y lim_sup de
    intensidad.
6   ## A los demas les da el valor del lim_inf.
7   ##
8   ## Variables de entrada:
9   ## @var{img} -> imagen original.
10  ## @var{lim_inf} -> valor de intensidad minimo permitido.
11  ## @var{lim_sup} -> valor de intensidad maxima permitido.
12  ##
13  ## Variables de salida:
14  ## @var{img_filt} -> imagen filtrada segun los valores de
    intensidad.
15  ##
16  ## For example:
17  ##
18  ## @example
19  ##
20  ## max(max(img))
21  ##    @result{} max(max(img)) = 3810
```

```
22  ## lim_inf = 700;
23  ## lim_sup = 800;
24  ## img_filt = filtro(img, lim_inf, lim_sup);
25  ## max(max(img_filt))
26  ##    @result{} max(max(img_filt)) = 100
27  ##
28  ##
29  ## @end example
30  ##
31  ## @seealso{}
32  ## @end deftypefn

33  ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
34  ## Created: Jun 13, 2014
35  ## Last modified: Sep 03, 2014

36  function img_filt = filtro(img, lim_inf, lim_sup)

37  img_filt = img;
38  img_filt(img < lim_inf) = lim_inf;
39  img_filt(img > lim_sup) = lim_inf;
40  img_filt = img_filt - lim_inf;

41  endfunction
```

Función "juntarimg.m"

```
1   ## -*- texinfo -*-
2   ## @deftypefn  {Function File} {@var{img_final} =} juntarimg
    (@var{img_blackwhite}, @var{img_coloured})
3   ## @deftypefnx {Function File} {@var{img_final} =} juntarimg
    (@var{img_coloured_main}, @var{img_coloured_second})
4   ##
5   ## This function adds to @var{img_blackwhite} o
    @var{img_coloured_main}
6   ## (dependiendo de si la imagen original es blanco y negro o ya es
    una imagen RGB),
7   ## la imagen en color @var{img_coloured} o
    @var{img_coloured_second}.
8   ## Los pixeles coloreados de @var{img_coloured} o
    @var{img_coloured_second}
9   ## sustituiran el valor inicial de los pixeles en la imagen
    original
10  ## @var{img_blackwhite} o @var{img_coloured_main}.
11  ##
12  ##
13  ## Variables de entrada:
14  ## @var{img_blackwhite} -> imagen original en blanco y negro.
15  ## @var{img_coloured} -> imagen coloreada para sustituir.
16  ## @var{img_coloured_main} -> imagen original en color.
17  ## @var{img_coloured_second} -> imagen coloreada para sustituir.
18  ##
19  ## Variables de salida:
20  ## @var{img_final} -> imagen final resultado de unir las dos
    iniciales.
21  ##
```

```
22 ## For example:
23 ##
24 ## @example
25 ##
26 ## %Si queremos juntar una imagen en blanco y negro con un contorno
   en color:
27 ## img_final = juntarimg(img_blackwhite, img_coloured)
28 #
29 ## %Si queremos juntar una imagen en color con un contorno en
   color:
30 ## img_final = juntarimg(img_coloured_main, img_coloured_second)
31 ##
32 ## @end example
33 ##
34 ## @seealso{}
35 ## @end deftypefn

36 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
37 ## Created: Jun 13, 2014
38 ## Last modified: Jul 14, 2014

39 function img_final = juntarimg(img_blackwhite, img_coloured)

40 if isrgb(img_blackwhite) == 0 %Si la imagen es en blanco y negro,
   la transformo a una imagen a color.
41 r_exp = 65535 / max(max(img_blackwhite));
42 for ii = 1:3
43 img_final(:, :, ii) = img_blackwhite(:, :);
44 endfor#
45 img_final = img_final * r_exp;

46 else %Si la imagen es a color, la dejo como está:
47 img_final = img_blackwhite;

48 endif#

49 %Modifico los puntos donde la imagen en color secundaria valga más
   de 0:
50 img_final(img_coloured > 0) = img_coloured(img_coloured > 0);

51 endfunction
```

Función “lectura_img.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {[@var{A}, @var{Num}] =} lectura_img
   (@var{num_inicial}, @var{num_final})
3  ## @deftypefnx  {Function File} {[@var{A}, @var{Num}] =}
   lectura_img (@var{num_inicial}, @var{num_final},
   @var{vector_almacen})
4  ## @deftypefnx  {Function File} {[@var{A}, @var{Num}] =}
   lectura_img (@var{num_inicial}, @var{num_final},
   @var{vector_almacen}, @var{indice}, @var{vector_Num})
5  ##
6  ## Esta funcion almacena los nombres de las imagenes que queremos
   cargar posteriormente, desde
```

```
7   ## un numero inicial @var{num_inicial} a un numero final
    @var{num_final}, guardandose los nombres
8   ## de la forma "i0***_0000b.dcm", donde "***" representa el numero
    de la imagen que le indicamos.
9   ## Si proporcionamos un vector de almacenamiento
    @var{vector_almacen} con su posible indice
10  ## @var{indice}, se utilizarian para almacenar en el vector de
    almacenamiento a partir de un índice.
11  ##
12  ## El vector @var{A} contendra los nombres de las imagenes,
    mientras que @var{Num} contendra los numeros
13  ## reales de las imagenes.
14  ##
15  ##
16  ## Variables de entrada:
17  ## @var{num_inicial} -> numero inicial de imagen que queremos leer.
18  ## @var{num_final} -> numero final de imagen que queremos leer.
19  ## @var{vector_almacen} -> vector donde queremos almacenar los
    nombres (opcional).
20  ## @var{indice} -> indice a partir del cual queremos almacenar los
    nombres es @var{vector_almacen} (opcional).
21  ## @var{vector_Num} -> vector "Num" almacenado de anteriores
    llamadas.
22  ##
23  ## Variables de salida:
24  ## @var{A} -> vector que contiene los nombres de las imagenes.
25  ## @var{Num} -> vector que contiene los numeros de las imagenes.
26  ##
27  ## For example:
28  ##
29  ## @example
30  ##
31  ## %Para leer las imagenes de la 115 a la 125:
32  ## [A, Num] = lectura_img(115, 125);
33  ##
34  ## @end example
35  ##
36  ## @seealso{}
37  ## @end deftypefn

38  ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
39  ## Created: Sep 03, 2014

40  function [A, Num] = lectura_img(num_inicial, num_final,
    vector_almacen = 0, indice = 1, vector_Num = 0)

41  if vector_almacen > 0
42  A = vector_almacen;
43  endif#
44  if vector_Num > 0
45  Num = vector_Num;
46  endif#

47  %Leo las imagenes originales.
48  if num_inicial < num_final

49  for hh = num_inicial:num_final %Las imágenes que quiero tratar, las
    indico con estos indices
50  b = num2str(hh);
51  if size(b)(2) == 1
52  A(indice, :) = ['i000', b, '_0000b.dcm'];
```

```
53  else if size(b)(2) == 2
54  A(indice, :) = ['i00', b, '_0000b.dcm'];
55  else
56  A(indice, :) = ['i0', b, '_0000b.dcm'];
57  endif#
58  end
59  Num(indice) = hh; %Aqui almacenamos el numero real.
60  indice = indice + 1;
61  endfor#

62  else

63  for hh = num_inicial:-1:num_final %Las imágenes que quiero tratar,
    las indico con estos indices
64  b = num2str(hh);
65  if size(b)(2) == 1
66  A(indice, :) = ['i000', b, '_0000b.dcm'];
67  else if size(b)(2) == 2
68  A(indice, :) = ['i00', b, '_0000b.dcm'];
69  else
70  A(indice, :) = ['i0', b, '_0000b.dcm'];
71  endif#
72  end
73  Num(indice) = hh; %Aqui almacenamos el numero real.
74  indice = indice + 1;
75  endfor#

76  endif#

77  endfunction
```

Función “limpieza.m”

```
1   ## -*- texinfo -*-
2   ## @deftypefn  {Function File} {@var{img_final} =} limpieza
    (@var{img_main}, @var{wide}, @var{n_points}, @var{threshold})
3   ## @deftypefnx {Function File} {@var{img_final} =} limpieza
    (@var{img_main}, @var{wide}, @var{n_points}, @var{threshold},
    @var{value})
4   ## @deftypefnx {Function File} {@var{img_final} =} limpieza
    (@var{img_main}, @var{wide}, @var{n_points}, @var{threshold},
    @var{value}, @var{orientation})
5   ## @deftypefnx {Function File} {@var{img_final} =} limpieza
    (@var{img_main}, @var{wide}, @var{n_points}, @var{threshold},
    @var{value}, @var{orientation}, @var{xmin}, @var{xmax}, @var{ymin},
    @var{ymax})
6   ##
7   ## Esta funcion hace una limpieza de la imagen en escala de grises
    @var{img_main}.
8   ## El parametro @var{wide} marca la mitad de la anchura del
    intervalo que tomaremos.
9   ## Lo que hace esta funcion es fijarse en el numero de puntos
    dentro de ese intervalo
10  ## cuyas intensidades son menores que el umbral @var{threshold}.
    Establecemos el numero
```

```
11 ## maximo de puntos permitido con la variable @var{n_points}. En el
   caso de cumplirse
12 ## que el intervalo de un punto xi, yi hay un numero de puntos
   menor que @var{n_points}
13 ## que tiene una intensidad menor que @var{threshold}, se le
   asignara un valor de intensidad
14 ## @var{value} a ese punto xi, yi. Podemos indicar la orientacion
   en que queremos realizar
15 ## esta comparacion con el parametro @var{orientation}, que por
   defecto toma el valor
16 ## "consecutive", es decir horizontal y vertical consecutivamente.
   Tambien podemos indicar
17 ## los valores minimos, maximos de x e y con los parametros
   @var{xmin}, @var{xmax}, @var{ymin},
18 ## @var{ymax}.
19 ##
20 ## La imagen final en escala de grises @var{img_final} sera el
   resultado final.
21 ##
22 ##
23 ## Variables de entrada:
24 ## @var{img_main} -> imagen principal a la que queremos aplicar
   limpieza.
25 ## @var{wide} -> semi-anchura del intervalo que tomaremos alrededor
   de cada punto.
26 ## @var{n_points} -> numero maximo de puntos en un intervalo
   alrededor de un punto xi, yi,
27 ## con una intensidad menor que @var{threshold} permitidos.
28 ## @var{threshold} -> intensidad maxima permitida para los puntos.
29 ## @var{value} -> valor que se le asignara a los puntos cuyo
   intervalo de alrededor no haya cumplido
30 ## la condicion impuesta por @var{n_points} y @var{threshold}. Por
   defecto se asigna la media de
31 ## los valores de la imagen menores que @var{threshold}.
32 ## @var{orientation} -> Puede ser "consecutive", "horizontal",
   "vertical" or "both".
33 ## "consecutive" significa que primero se hace en horizontal, luego
   en vertical a partir de la nueva
34 ## imagen (valor por defecto).
35 ## "horizontal" solo se hace en horizontal.
36 ## "vertical" solo se hace en vertical.
37 ## "both" se hace en ambas direcciones pero con independencia.
38 ## @var{xmin} -> minimo valor de x del recuadro en el que
   realizaremos limpieza.
39 ## @var{xmax} -> maximo valor de x del recuadro en el que
   realizaremos limpieza.
40 ## @var{ymin} -> minimo valor de y del recuadro en el que
   realizaremos limpieza.
41 ## @var{ymax} -> maximo valor de y del recuadro en el que
   realizaremos limpieza.
42 ##
43 ##
44 ## Variables de salida:
45 ## @var{img_final} -> imagen final en escala de grises resultado de
   la limpieza.
46 ##
47 ## For example:
48 ##
49 ## @example
50 ##
```

```
51 ## %Queremos realizar una limpieza de IMG con una anchura de
   intervalo de 15, un numero
52 ## %de puntos maximos permitidos de 10, un umbral de 500 de
   intensidad y una orientacion
53 ## %horizontal. El resultado sera IMG_FINAL:
54 ## img_final = limpieza(IMG, 15, 10, 500, "default", 'horizontal');
55 ##
56 ## @end example
57 ##
58 ## @seealso{}
59 ## @end deftypefn

60 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
61 ## Created: Jul 02, 2014

62 function img_final = limpieza(img_main, wide = 10, n_points = 8,
   threshold = 500, value = "default", orientation = "consecutive",
   xmin = "default", xmax = "default", ymin = "default", ymax =
   "default")

63 img_final = img_main;

64 if xmin == "default"
65 xmin = 1;
66 endif#
67 if xmax == "default"
68 xmax = size(img_main)(1);
69 endif#
70 if ymin == "default"
71 ymin = 1;
72 endif#
73 if ymax == "default"
74 ymax = size(img_main)(2);
75 endif#

76 if ((orientation == 'both' || orientation == 'vertical') ||
   orientation == 'consecutive')
77 %Por columnas:
78 for ii = xmin:xmax
79 for kk = ymin:ymax
80 numero = 0;
81 for kkh = kk - wide:kk + wide
82 if (kkh >= 1 && kkh <= size(img_main)(2))
83 if (img_main(kkh, ii) >= threshold )
84 numero += 1;
85 endif#
86 endif#
87 endfor#
88 if numero < n_points
89 if value == "default"
90 value = mean(img_main(img_main < threshold))
91 endif#
92 img_final(kk, ii) = value;
93 endif#
94 endfor#
95 endfor#
96 endif#

97 if (orientation == 'consecutive')
98 img_main = img_final;
99 endif#
```

```
100   if((orientation == 'both' || orientation == 'horizontal') ||
    orientation == 'consecutive')
101   %Por filas:
102   for ii = xmin:xmax
103   for kk = ymin:ymax
104   numero = 0;
105   for kkh = kk - wide:kk + wide
106   if (kkh >= 1 && kkh <= size(img_main)(1))
107   if (img_main(ii, kkh) >= threshold )
108   numero += 1;
109   endif#
110   endif#
111   endfor#
112   if numero < n_points
113   if value == "default"
114   value = mean(img_main(img_main < threshold))
115   endif#
116   img_final(ii, kk) = value;
117   endif#
118   endfor#
119   endfor#
120   endif#

121 endfunction
```

Función “max_intensity_objects.m”

```
1   ## -*- texinfo -*-
2   ## @deftypefn  {Function File} {@var{max_intensity} =}
    max_intensity_objects (@var{img})
3   ##
4   ## Esta funcion calcula la maxima intensidad dentro de los objetos
    de la imagen
5   ## @var{img} y las almacena en @var{max_intensity}.
6   ##
7   ## Variables de entrada:
8   ## @var{img} -> imagen de cuyos objetos queremos conocer la maxima
    intensidad.
9   ##
10  ## Variables de salida:
11  ## @var{max_intensity} -> matriz que contiene la intensidad maxima
    de los objetos.
12  ## Its size is (num_obj x 1).
13  ##
14  ## For example:
15  ##
16  ## @example
17  ##
18  ## %Para almacenar las maximas intensidades de los objetos que
    contiene una imagen IMG:
19  ## max_intensity = max_intensity_objects(IMG);
20  ##
```

```
21 ## @end example
22 ##
23 ## @seealso{regionprops, bwlabel, areas_objetos, centroide,
   bounding_box}
24 ## @end deftypefn

25 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
26 ## Created: Jul 06, 2014
27 ## Last modified: Sep 05, 2014

28 function max_intensity = max_intensity_objects(img)

29 props = regionprops(img, "max_intensity");
30 A = struct2cell(props);
31 B = cell2mat(A(1,1,:));

32 max_intensity(:, 1) = B(1, 1, :);%Maxima intensidad del objeto.

33 endfunction
```

Función “mean_intensity_objects.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{mean_intensity} =}
   mean_intensity_objects (@var{img_binary}, @var{img_main})
3  ##
4  ## Esta funcion calcula las intensidades medias dentro de los
   objetos de la imagen
5  ## @var{img} y las almacena en @var{max_intensity}.
6  ##
7  ## Variables de entrada:
8  ## @var{img} -> imagen de cuyos objetos queremos conocer la
   intensidad media.
9  ##
10 ## Variables de salida:
11 ## @var{mean_intensity} -> matriz que contiene la intensidad media
   de los objetos.
12 ## Its size is (num_obj x 1).
13 ##
14 ## For example:
15 ##
16 ## @example
17 ##
18 ## %Para almacenar las intensidades medias de los objetos que
   contiene una imagen IMG:
19 ## mean_intensity = mean_intensity_objects(IMG);
20 ##
21 ## @end example
22 ##
23 ## @seealso{regionprops, bwlabel, areas_objetos, centroide,
   bounding_box}
24 ## @end deftypefn
```

```
25 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
26 ## Created: Jul 06, 2014
27 ## Last modified: Sep 05, 2014

28 function mean_intensity = mean_intensity_objects(img_binary,
   img_main)

29 props = regionprops(img_binary, img_main, "MeanIntensity");
30 A = struct2cell(props);
31 B = cell2mat(A(1,1,:));

32 mean_intensity(:, 1) = B(1, 1, :);%Intensidad media del objeto.

33 endfunction
```

Función “mostrar_imagen.m”

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} mostrar_imagen (@var{img})
3  ## @deftypefnx {Function File} mostrar_imagen (@var{img},
   @var{titulo})
4  ##
5  ## Esta funcion muestra una imagen @var{img} con un titulo
   @var{titulo}.
6  ## En el caso de que @var{img} sea una imagen binaria, la
   transformando a escala de
7  ## grises con la funcion MAT2GRAY y posteriormente llama a la
   funcion IMSHOW. En la
8  ## variable @var{titulo} podemos indicar un titulo de la  imagen
   con la forma 'titulo'.
9  ##
10 ## Variables de entrada:
11 ## @var{img} -> imagen que queremos mostrar.
12 ## @var{titulo} -> titulo de la imagen. Por defecto se muestra
   vacio.
13 ##
14 ## Variables de salida:
15 ##
16 ## For example:
17 ##
18 ## @example
19 ##
20 ## %Para mostrar una imagen A que se llame "hueso":
21 ## mostrar_imagen(A, 'Hueso');
22 ##
23 ## @end example
24 ##
25 ## @seealso{imshow, title}
26 ## @end deftypefn

27 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
28 ## Created: Jun 14, 2014

29 function mostrar_imagen(img_binary, titulo = '')%Valores por
   defecto
```

```
30 if (isbw(img_binary) == 1)
31 I = mat2gray(img_binary);
32 else
33 I = img_binary;
34 endif#
35 figure, imshow(I, []), title(titulo);

36 endfunction
```

Función "perimeter.m"

```
1  ## -*- texinfo -*-
2  ## @deftypefn  {Function File} {@var{perimeter} =}
   perimeter_objects (@var{img})
3  ##
4  ## Esta funcion calcula el perimetro de los objetos de la imagen
5  ## @var{img} y lss almacena en @var{perimeter}.
6  ##
7  ## Variables de entrada:
8  ## @var{img} -> imagen de cuyos objetos queremos conocer la maxima
   intensidad.
9  ##
10 ## Variables de salida:
11 ## @var{perimeter} -> matriz que contiene los perimetros de los
   objetos.
12 ## Its size is (num_obj x 1).
13 ##
14 ## For example:
15 ##
16 ## @example
17 ##
18 ## %Para almacenar los perimetros de los objetos que contiene una
   imagen IMG:
19 ## perimeter = perimeter_objects(IMG);
20 ##
21 ## @end example
22 ##
23 ## @seealso{regionprops, bwlabel, areas_objetos, centroide,
   bounding_box}
24 ## @end deftypefn

25 ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
26 ## Created: Jul 06, 2014
27 ## Last modified: Sep 02, 2014

28 function perimeter = perimeter_objects(img)

29 props = regionprops(img, "Perimeter");
30 A = struct2cell(props);
31 B = cell2mat(A(1,1,:));

32 perimeter(:, 1) = B(1, 1, :);%Perimetro del objeto.

33 endfunction
```

Función “repaso.m”

```
1   ## -*- texinfo -*-
2   ## @deftypefn  {Function File} {@var{img_result} =} repaso
    (@var{img_binary})
3   ##
4   ## Esta funcion emplea la coloquialmente llamada "tactica del punto
    gordo" sobre la imagen
5   ## binaria @var{img_binary}, creando una nueva imagen
    @var{img_result}. Consiste en colocar
6   ## puntos blancos alrededor de los puntos blancos de la imagen
    binaria @var{img_binary}.
7   ##
8   ## La imagen final @var{img_result} binaria sera el resultado
    final.
9   ##
10  ## Variables de entrada:
11  ## @var{img_binary} -> imagen binaria a la que queremos realizar el
    repaso.
12  ##
13  ## Variables de salida:
14  ## @var{img_result} -> imagen final binaria despues del repaso.
15  ##
16  ## For example:
17  ##
18  ## @example
19  ##
20  ## %Queremos repasar una imagen binaria IMG_ORIG:
21  ## IMG_RESULT = repaso(IMG_ORIG);
22  ##
23  ## @end example
24  ##
25  ## @seealso{}
26  ## @end deftypefn

27  ## Author: Francisco J. Marin <fcoj.marin@hotmail.com>
28  ## Created: Sep 03, 2014
29  ## Last modified: Sep 05, 2014

30  function img_result = repaso(img_binary)

31  img_result = img_binary;

32  for ii = 1:size(img_binary)(1)
33  for jj = 1:size(img_binary)(2)
34  if img_binary(ii, jj) > 0
35  min_x = max(1, ii - 1);
36  max_x = min(size(img_binary)(1), ii + 1);
37  min_y = max(1, jj - 1);
38  max_y = min(size(img_binary)(1), jj + 1);
39  img_result(min_x:max_x, min_y:max_y) = 1;
40  endif#
41  endfor#
42  endfor#

43  endfunction
```

Autor: **Francisco Javier Marín Marín.**

Titulación: **Ingeniero Industrial.**

Fecha: **08 de Septiembre de 2014.**