Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Centro de Estudios de Electrónica y Tecnologías de la Información

TRABAJO DE DIPLOMA

Herramienta para manipulación y análisis de imágenes de gammagrafía planar

Autor: Bradley Fernández Cabrera

Tutor: Ing. Iroel Miranda Castañeda

Santa Clara

2012

"Año 54 de La Revolución"

Universidad Central “Marta Abreu” de Las Villas

Facultad de Ingeniería Eléctrica

Centro de Estudios de Electrónica y Tecnologías de la Información

TRABAJO DE DIPLOMA

Herramienta para manipulación y análisis de imágenes de gammagrafía planar

Autor: Bradley Fernández Cabrera

bfernandez@uclv.edu.cu

Tutor: Ing. Iroel Miranda Castañeda

Facultad de Ingeniería Eléctrica, departamento de procesamiento de imágenes, miranda@uclv.edu.cu

Consultante: Lic. Reinaldo Roque Díaz

Santa Clara

2012

"Año 54 de La Revolución"

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

_____

Firma del Autor

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

_____ _____

Firma del Autor

Firma del Jefe de Departamento donde se defiende el trabajo

_____

Firma del Responsable de Información Científico-Técnica

PENSAMIENTO

No hay secretos para el éxito. Éste se alcanza preparándose, trabajando arduamente y aprendiendo del fracaso.

Colin Powell

DEDICATORIA

Dedicado a mi padre.

AGRADECIMIENTOS

Doy gracias a mi papá y a mi mamá por el apoyo que me brindaron en todo momento. A mi tutor, el Ing. Iroel Miranda Castañeda que estuvo siempre ayudándome y aconsejándome y a mi cotutor, el Lic. Reinaldo Roque que me ayudó y facilitó las imágenes para este trabajo.

A mis compañeros de aula, en especial a Asnel y Alexis por los momentos que compartimos.

Al Dr. Julián Cárdenas Barreras por su ayuda en la interpretación de las imágenes planares.

A todos aquellos que de una forma u otra ayudaron a que fuera posible la realización de este trabajo.

TAREA TÉCNICA

- A sugerencia de los especialistas médicos del hospital provincial "Celestino Hernández Robau" se diseña e implementa la herramienta para manipular las imágenes de la cámara Gamma presente en dicho hospital.
- Se implementan algoritmos para manipular las imágenes en el formato *Sopha* provenientes de la cámara Gamma, sobre la plataforma *Windows*.
- Como alternativas para los análisis se implementan funciones básicas para la manipulación de imágenes en general.
- Con el objetivo de que sea lo más similar posible al *software* de la cámara se incluyen opciones estrechamente relacionadas con los estudios que se realizan a las imágenes de Medicina Nuclear.
- Se incluye una ayuda para facilitar el uso de la herramienta desarrollada.

Firma del Autor

Firma del Tutor

RESUMEN

El análisis de las imágenes planares en medicina nuclear brinda grandes posibilidades a la hora de realizar un diagnóstico. Aunque en la ciudad de Santa Clara se cuenta con una cámara capaz de obtener este tipo de imágenes existen limitaciones técnicas que imposibilitan un diagnóstico totalmente eficiente, dado incompatibilidades que existen entre el formato de la cámara y la plataforma comúnmente utilizada en las instituciones hospitalarias, *Windows*. Hacer uso del software *Matlab*, específicamente el ambiente de desarrollo de interfaz de usuario y el *toolbox* de Procesamiento de Imágenes ofrece la posibilidad de crear una herramienta capaz de solucionar la desventaja de la incompatibilidad existente con las imágenes de gammagrafía planar. Con los resultados obtenidos en este trabajo se propone una herramienta capaz de manipular las imágenes provenientes de la cámara Gamma, además permitir realizar diferentes análisis de manera generales en las imágenes y algunos específicos de las imágenes de medicina nuclear. La herramienta obtenida tiene buena aceptación por parte del personal médico y puede ser usada en investigaciones y en la docencia en temas relacionados imágenes de medicina nuclear.

TABLA DE CONTENIDOS

INTRODUCCIÓN

El diagnóstico por imagen ha tenido grandes avances en las últimas décadas lo que favorece la detección de enfermedades en etapas relativamente tempranas y una mejor planificación del tratamiento. La medicina nuclear es una de las técnicas de obtención de imagen pioneras en la historia de las imágenes médicas. Su desarrollo ha sido relevante para realizar diagnósticos precisos a pacientes con diversas patologías. En la actualidad, en el Hospital "Celestino Hernández Robau", con sede en la ciudad de Santa Clara, se cuenta con una cámara Gamma de fabricación francesa, del año 1989, que brinda servicios a pacientes en la región central. Una de las imágenes que se pueden obtener son las gammagrafías planares, dichas imágenes presentan conflictos a la hora de su manipulación y estudio debido a que es muy limitada la disposición de medios para realizar su informe, producto de la incompatibilidad del sistema *Sopha* que presenta la cámara y la plataforma *Windows* que es lo comúnmente utilizado en la actualidad. Esto presenta un inconveniente para los especialistas médicos ya que sólo pueden realizar el análisis de las imágenes en la propia cámara y no disponen de una herramienta capaz de manipular este tipo de imagen.

Para dar respuesta a este problema se pretende desarrollar algoritmos de manipulación y tratamiento para las imágenes que se obtienen de la cámara Gamma. El desarrollo de herramientas puede ser útil a la hora de realizar la implementación de un *software* capaz de manipular este tipo de imágenes. Dicha herramienta debe cumplir con determinados requisitos propuestos por los especialistas médicos para realizar análisis y modificaciones sobre las imágenes y que éstas cumplan su propósito fundamental, el diagnóstico eficiente.

El hecho de contar con una aplicación que pueda obtener y visualizar en una pantalla este tipo de imágenes, se considera de gran ayuda para el personal encargado por la posibilidad de analizar las imágenes en cualquier ordenador. Las prestaciones que brinde la

herramienta deben estar cercanas al *software* que posee la cámara por dos motivos fundamentales;

- para que cumpla objetivo la utilización de la herramienta
- evitar posible resistencia al cambio por parte de los usuarios de la cámara.

En el momento en que se realiza la implementación de la herramienta, no existe ningún otro método ni mecanismo capaz de facilitar la tarea de manipular y analizar las imágenes en el formato *Sopha*, el mismo es propietario y el fabricante no brinda información sobre este tipo de imágenes.

OBJETIVOS DEL TRABAJO

Objetivos generales

El objetivo principal consiste en diseñar y construir una herramienta para manipular las imágenes que se obtienen de la cámara Gamma *Sophy Circular* 1000 del Hospital Provincial “Celestino Hernández Robau”.

Objetivos particulares

- Implementar algoritmo para manipular las imágenes en formato *Sopha* desde la plataforma *Windows*.
- Diseñar herramienta gráfica de fácil utilización para los especialistas médicos.
- Implementar los análisis necesarios sobre las imágenes a fin de establecer un diagnóstico eficiente.
- Utilizar las prestaciones del ambiente de desarrollo de interfaces gráficas y el *toolbox* de Procesamiento Digital de Imágenes (PDI) de *Matlab* para una implementación más eficiente de la herramienta.

CAPÍTULO 1. Estudio y manipulación de imágenes de medicina nuclear

Dentro de las diferentes técnicas de diagnóstico por imagen se encuentran imágenes de medicina nuclear las que son imprescindibles para el diagnóstico de determinadas patologías, lo que hace necesario incrementar las facilidades de su acceso y manipulación por parte del personal médico buscando un diagnóstico más eficiente.

Este capítulo presenta detalles sobre la gammagrafía planar, la cámara Gamma *Sophy Circular* 1000 que se encuentra en el hospital "Celestino Hernández Robau" de la ciudad de Santa Clara, y sobre el software *Matlab*, seleccionado como alternativa para el diseño de una herramienta capaz de manipular las imágenes de dicha cámara y facilitar su análisis por parte de los especialistas.

1.1 Las imágenes médicas en formato digital

Las imágenes médicas en la actualidad se consideran un importante método de diagnóstico. Una imagen no es más que la representación de un órgano, tejido o proceso fisiológico que se obtiene a través de procesos físicos ordenados con este fin. La representación de la información contenida en una imagen se utiliza para emitir diagnósticos o evaluar cambios que ocurren al paciente durante determinado tratamiento. Gracias a los avances logrados en la actualidad se puede obtener una imagen de casi cualquier parte del cuerpo mediante diferentes modalidades. En la Fig. 1 podemos apreciar ejemplos de diferentes tipos de imágenes obtenidas por algunas de las técnicas existentes [1].

(a) (b) (c) (d)

Fig. 1. Ejemplo de imágenes obtenidas mediante deferentes técnicas: (a) Resonancia Magnética, (b) Tomografía Axial Computarizada, (c) Radiografía y (d) Ultrasonido.

Matemáticamente, una imagen puede considerarse como una función bidimensional expresada de la siguiente manera:

$$I = f(x, y); \tag{1}$$

donde I representa el valor que adquiere la intensidad en la coordenada x, y de la matriz que representa la imagen. Cuando se cuantifican los valores de intensidad y sus respectivas coordenadas espaciales se obtiene una imagen digital, los valores de intensidad discretos son un número limitado de elementos denominados píxeles, los cuales tienen un valor único, que se asigna durante el proceso de adquisición y se considera el menor elemento en el que puede dividirse una imagen [1].

1.2 Gammagrafía planar

La gammagrafía planar es un procedimiento de diagnóstico en el que se obtienen imágenes morfológicas o funcionales tras introducir por vía intravenosa una sustancia marcada con un isotopo radiactivo, la sustancia aplicada es a fin con el órgano o estructura en estudio. La información de la imagen depende de la cantidad de conteos que posea cada píxel, de modo que la intensidad de cada uno es aumentada proporcionalmente con la cantidad de conteos según el tiempo de exposición del paciente. El número de conteos o intensidad del píxel se hace coincidir con tonos grises o de colores con lo que queda conformada la imagen [2].

Las imágenes planares convencionales producen una proyección de estructuras desde un plano tridimensional a un espacio en dos dimensiones. La suma de estructuras de distintos planos se traduce en la suma de intensidades que son difíciles de separar visualmente para extraer información cuantitativa. Lo que se convierte en la principal desventaja de este tipo de imágenes ya que no es posible eliminar la interferencia de planos adyacentes o vecinos, y se dificulta la ubicación correcta de una acumulación del radiotrazador esto pudiera ser tomado por algo inusual o patológico cuando se hace el diagnóstico [2].

La gammagrafía permite obtener imágenes patológicas por hipercaptación (nódulos calientes, con hiperactividad tiroidea) o hipocaptación (nódulos fríos), así como detectar metástasis de cáncer de tiroides en otras localizaciones. También se obtienen imágenes patológicas de infecciones, tumores primarios o metastásicos y enfermedades óseas metabólicas. Otra de las patologías que se pueden detectar son las zonas isquémicas por enfermedad coronaria.

A las imágenes planares se les pueden realizar diversos análisis para el diagnóstico de diferentes patologías. Dentro de los más utilizados se encuentran la medición de área y distancia sobre lesiones en el cuerpo con el fin de determinar el tamaño o volumen de determinadas regiones. Realizar acercamientos en la imagen así como seleccionar regiones que sean de interés para los especialistas. Aplicar mapas de falso color y rotar las imágenes según sea necesario para una mejor visualización, entre otras.

1.3 Cámara Gamma

El equipo encargado de la obtención de las imágenes se denomina cámara Gamma, es un dispositivo capaz de obtener imágenes mediante la detección y cuantificación de los fotones emitidos por una sustancia introducida en el organismo. Dicha imagen se obtiene mediante la detección simultánea de la radiación Gamma emitida por la sustancia que se encuentra en el órgano o tejido.

Básicamente una cámara Gamma se compone por los siguientes elementos:

Detector:

- Colimador (Constituyen la guía direccional de los rayos gamma)

- Cristal de Centelleo (Convierte los rayos gamma en destellos de luz)
- Arreglo de Tubos Fotomultiplicadores (TFM) (La salida de los TFM son sumadas para obtener la señal Z)
- Pre-amplificadores

Electrónica de procesamiento:

- Circuitos de Posicionamiento (Convierten las señales eléctricas en señales X, Y y Z)
- Analizador de Altura de Pulso (Analiza la amplitud de la señal de energía y rechaza cualquier señal fuera del rango de energías seleccionado)
- Sistema de Visualización o Procesamiento

Relacionado con la funcionabilidad y el tiempo de construcción del dispositivo algunos equipos son más complejos que otros [3] [4] [5] [6].

1.3.1 Principio de funcionamiento de la cámara Gamma

A diferencia de otras técnicas como la Tomografía Computarizada y la Radiografía convencional en los cuales los rayos X atraviesan al paciente y se mide la densidad del tejido basado en el grado de absorción, en la gammagrafía el paciente genera la radiación mediante una sustancia que se le introduce con un marcador a fin al órgano o tejido que se pretende estudiar. La radiación gamma emitida por el paciente llega directamente al colimador de la cámara. El fotón gamma incidente que pasa a través de los agujeros del colimador, interacciona con el cristal de centelleo y produce un impulso de luz proporcional a la energía depositada en el cristal. El pulso de luz atraviesa el cristal y se transmite, a través de la guía de luz, a los tubos fotomultiplicadores, que convierten la señal luminosa en señales eléctricas de amplitud proporcional a la luz incidente. El conjunto de fototubos que reciben la luz generan unas señales X, Y de posicionamiento y una señal Z que porta la amplitud del impulso originado en cada fototubo. Estas señales son preamplificadas y filtradas mediante un sistemas de preamplificadores especializados; obteniéndose como resultado dos señales de codificación espacial X, Y y una señal Z proporcional a la energía depositada en el cristal por el fotón. La señal Z será clasificada en el Analizador de Altura de Pulsos, si se encuentra dentro del rango de energías predefinido por el operador,

conocido como ventana energética, se digitalizará a través de un Conversor Análogo Digital y se ubicará en la matriz de adquisición en su posición X, Y correspondiente; formando una imagen en dos dimensiones en las unidades de procesamiento y presentación. En la Fig. 2 podemos ver un diagrama de bloques del funcionamiento descrito anteriormente [3] [6].

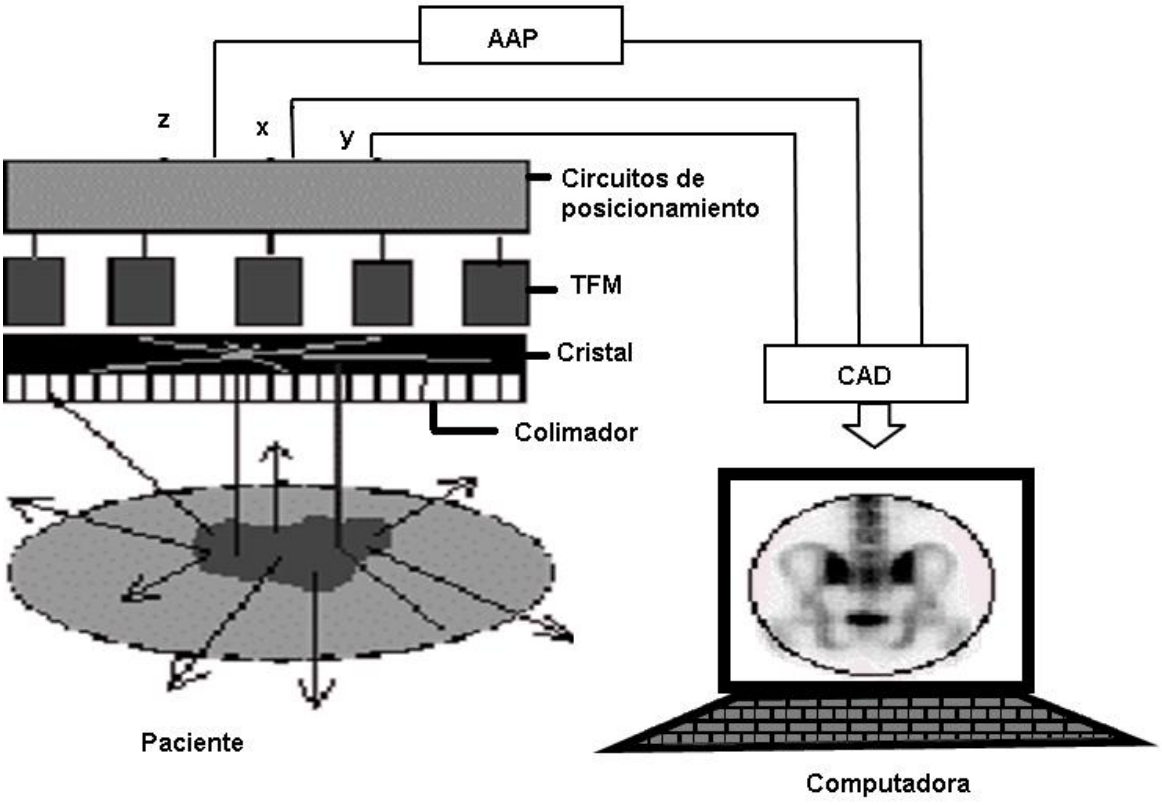


Fig. 2. Diagrama de bloques del principio de funcionamiento de la cámara Gamma.

1.4 La cámara Gamma *Sophy Circular* 1000

La cámara Gamma *Sophy Circular* 1000, fabricada en 1989, actualmente está instalada en el hospital "Celestino Hernández Robau", sede en la ciudad de Santa Clara, es considerada como un importante método de diagnóstico de enfermedades de diversos niveles de peligrosidad. Con más de veinte años de trabajo, ofrece diagnósticos con elevada precisión, dicha cámara presta servicio a pacientes atendidos en el área de Medicina Nuclear de este y

otros hospitales debido a que es el único equipo en su tipo para las provincias de la región central de Cuba. En la Fig. 3 podemos apreciar imágenes de la cámara.

(a) (b)

Fig. 3. Cámara Gamma *Sophy Circular* 1000, hospital "Celestino Hernández Robau", Santa Clara: (a) Gantri y (b) local de monitoreo e informe de las imágenes.

1.4.1 Especificaciones técnicas

Las especificaciones técnicas que presenta la cámara Gamma son las siguientes:

Fabricante: *Sopha Médical*

Equipo: *Sophycaméra série* 1000 circular

Detector: Cristal de Centelleo de Yoduro de Sodio activado con Talio, 9.5 mm de grosor, 61 fotomultiplicadores hexagonales.

Fabricado en Francia [7]

1.4.2 Bondades y limitaciones

La cámara Gamma *Sophy Circular* 1000 permite realizar imágenes planares para el estudio de diferentes patologías. Brinda buenos resultados en cuanto a las imágenes que se obtienen con ella y permite recopilar la información en un disco duro, además de exportar la información necesaria en discos de 5¼ pulgadas (Fig. 4) cuya capacidad varía entre los 100 KB y 1.2 MB los cuales, en la actualidad, se consideran obsoletos y su uso se limita a aplicaciones de las imágenes obtenidas con la cámara.

Debido a que de la época en la que fue desarrollada la cámara y que en actualidad las normas y dispositivos para el almacenamiento y transmisión de la información han evolucionado aceleradamente, presenta serias incompatibilidades, por ejemplo, solamente exporta las imágenes obtenidas mediante discos de 5¼ pulgadas, que dados su extenso tiempo de uso, no dan garantía a la hora de transportar y manipular las imágenes. Con el paso de los años el material magnético con el que están conformados tiende a perder sus propiedades lo que provoca fallos frecuentes. Por otra parte, con los avances tecnológicos actuales, ya no se fabrica este tipo de soporte, lo que hace que escaseen.

Fig. 4. Imagen de discos 5¼ pulgadas usados para la extracción de imágenes de la cámara Gamma *Sophy Circular* 1000.

El sistema operativo que utiliza la cámara permite exportar las imágenes en un formato llamado *Sopha*, que es propietario y no se brinda información acerca del mismo. Actualmente en Cuba algunas instalaciones médicas que brindan servicios utilizando cámaras Gamma cuentan con unidades computarizadas que permiten adquirir, procesar y archivar los estudios realizados, producto una actualización de los equipos con el sistema *Imagamma* (bloques de procesamiento y software). En la ciudad de Santa Clara aún no se ha llevado a cabo esta actualización lo cual imposibilita extraer las imágenes de la cámara, almacenarlas y realizar el diagnóstico en otro momento que no sea el del estudio con el paciente en el gantri [3].

El formato *Sopha* no es compatible con el sistema operativo *Windows*, que es el sistema operativo más comúnmente usado en las instalaciones hospitalarias de nuestro país. Esto trae como consecuencia que el informe de los estudios realizados por parte de los especialistas se vea reducido únicamente a realizarlo en la propia cámara y no a una

computadora aparte con software especializados para determinados análisis y posibles manejos más eficientes; de ahí la motivación de crear una herramienta que pueda cargar las imágenes de esta cámara y realizarle diversos análisis, además de presentar una forma amigable y fácil acceso a sus funciones para que los especialistas médicos no necesiten conocimientos de programación. El propósito general de esta herramienta es que las imágenes puedan utilizarse sobre la plataforma del sistema operativo *Windows*.

1.5 Diseño de interfaces gráficas de usuario

El término Interfaz Gráfica de Usuario *GUI* del inglés (*Graphical User Interface*) se utiliza para llamar al conjunto de elementos visuales relacionados entre sí, que brinda un sistema o programa para que el usuario interactúe con él. El estudio y desarrollo del diseño de una interfaz requiere de un trabajo donde están inmersas viarias disciplinas en función a un mismo objetivo: cubrir la necesidad del hombre de transmitir y comunicar, en este caso, a través de un medio virtual. Las disciplinas que intervienen pueden variar pero las que se mantienen de alguna manera constante son: la ingeniería, la programación y el diseño. Una interfaz gráfica, de forma general, debe ser básicamente:

1. Sencilla. Los elementos están para apoyar, ayudar y guiar, no para confundir, hay que evitar la saturación y colocación innecesaria de los mismos.
2. Clara. La información debe ser fácilmente localizable, es decir, debe estar organizada ya sea de manera lógica, jerárquica o temática.
3. Predecible. A acciones iguales, resultados iguales.
4. Flexible. Pensar en botones que puedan modificar textos, realizar cambios en algunas secciones según convenga, etc.
5. Consistente. Aunque se realicen cambios en la programación, la representación gráfica de las funciones e imágenes debe permanecer igual.
6. Intuitiva. El usuario se siente más seguro en una aplicación en la que no tenga que adivinar ni pensar como ejecutar acciones.

7. Coherente. Tanto texto como gráficos, colores y demás elementos utilizados deben corresponder al contenido de la aplicación. Apoyados generalmente por una construcción de palabras, frases y elementos visuales [8].

1.6 Software *Matlab* como alternativa para el estudio y manipulación de imágenes obtenidas con la cámara Gamma *Sophy Circular* 1000

Matlab, cuyo nombre proviene del inglés (*MATrix LABoratory*) es un software matemático que tiene un lenguaje de programación propio (Lenguaje M) y es multiplataforma (*Unix, Windows y Apple Mac Os X*) (ver Fig. 5). Creado en 1984 por Cleve Moler con la idea de crear paquetes de subrutinas escritas en *Fortran* (del inglés *Formula Translating System*), un lenguaje orientado al cálculo numérico, diseñado en sus inicios para las computadoras *IBM* y usado en aplicaciones científicas y de ingeniería, se puede destacar que es el más antiguo de los lenguajes de alto nivel [9].

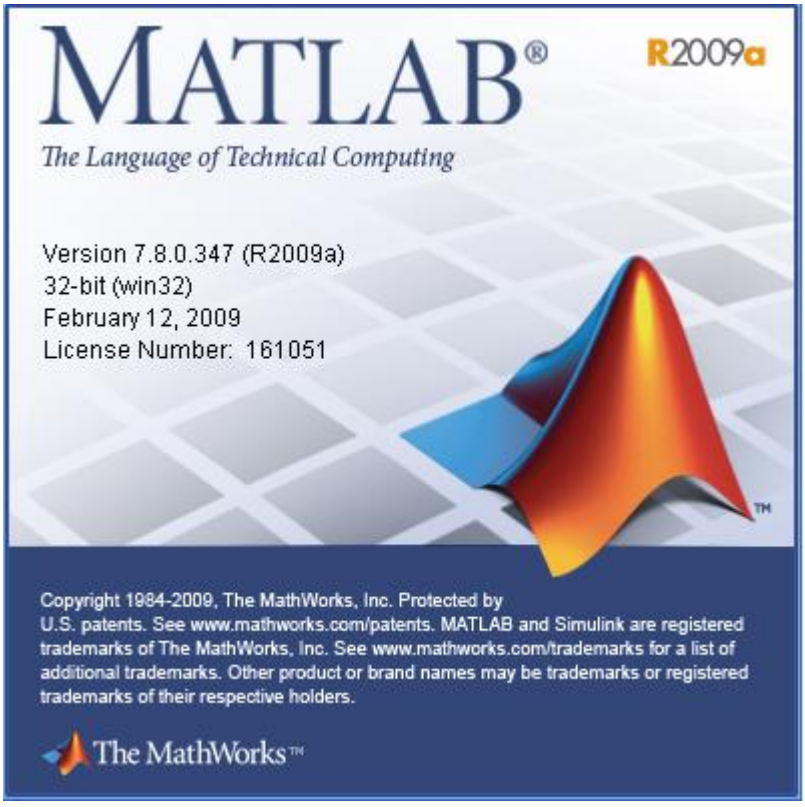


Fig. 5. Presentación de la vista del software *Matlab*.

El lenguaje de programación M se creó en 1970 proporcionando una acceso sencillo al software de matrices *LINPACK* (del inglés *Linear System Package*) y *EISPACK* (del inglés

Eigen System Package) sin tener que hacer uso del lenguaje *Fortran*. Ya en el año 2004 se apreciaba que *Matlab* era usado por aproximadamente más de un millón de personas, tanto académicos como empresarios.

Dentro de sus principales funciones se encuentran:

- Manipulación de matrices.
- Representación de datos y funciones.
- Implementación de algoritmos.
- Creación de interfaces de usuario (*GUI*).
- Comunicación con programas en otros lenguajes y con otros dispositivos *Hardware*.

Además, posee herramientas adicionales como el *Simulink* (Plataforma de simulación multidominio) y el Editor de interfaces de usuario, junto con *toolbox* de Procesamiento digital de imágenes [9].

El uso del *Matlab* ha posibilitado el desarrollo en áreas de investigación relacionadas con procesamiento digital de señales ya sea audio, imágenes o video, debido a la gran diversidad de funciones y herramientas que presenta destinadas para ese fin. Una de las aplicaciones más comunes que se le da a este *software* está relacionada con el tratamiento de las imágenes médicas, eso agregado a su fácil manejo y la amigable interpretación del lenguaje de programación hacen que sea seleccionado como una alternativa para el estudio y el manejo de las imágenes de medicina nuclear. Posee una ayuda al usuario con numerosos ejemplos que, hace posible a un usuario de menor experiencia realizar los algoritmos, funciones y herramientas en un intervalo corto de tiempo dependiendo de la complejidad del problema al que se enfrente [9].

1.7 GUIDE

Una de las herramientas que posee *Matlab* es el Editor de Interfaces de Usuario *GUIDE* (del inglés *Graphical User Interface Development Environment*), la cual presenta características que posibilitan el desarrollo de interfaces gráficas de usuario de manera sencilla e intuitiva. Para realizar la construcción de un *GUI* se utiliza el Editor de Diseño del *GUIDE* que permite de manera gráfica seleccionar y configurar los elementos y los

componentes del *GUI* que se desea desarrollar. Los componentes en el *GUIDE* más utilizados dentro del área de diseño son:

- *Push button* o botones de presión son los encargados de generar una acción cuando se le hace clic encima de ellos.
- *Slider* o barra de desplazamiento, acepta valores de entrada de posición dentro de un rango especificado que son establecidos moviendo la barra deslizante. La localización de la barra deslizante indica la localización que es el valor que tomará la variable asociada al elemento.
- *Check Box* o cajas verificación, pueden generar una acción una vez que se activen e indican su estado de verificado o no verificado o sea en alto o en bajo. Las cajas verificadoras son útiles para proporcionar opciones independientes al usuario, por ejemplo, cuando se tienen los textos de una imagen la opción de ser mostrados o no puede estar asociada a un elemento de este tipo.
- *Radio Button* o botón circular, son similares a los *check boxes* o botones de chequeo con la diferencia de que están relacionados entre sí de manera que, si uno está seleccionado el otro por definición esta deseleccionado.
- *Edit Text* o texto editable, son campos que le permiten a los usuarios insertar o modificar cadenas de caracteres. Es importante destacar que este objeto manipula cadenas de caracteres por lo tanto cuando se insertan números para poder ser usados deben convertirse a sus equivalentes numéricos.
- *Static Text* o textos estáticos, es un objeto que permite mostrar líneas de texto que a diferencia del *Edit Text* se muestran fijas y no pueden ser modificadas por el usuario. Este tipo de objeto puede ser usado para etiquetar controles, proporciona las direcciones al usuario, o indica valores resultantes de algún cálculo.
- *Pop-Up menú* o menú desplegable, este objeto despliega una lista de opciones.
- *Axes*, permiten al *GUI* desplegar gráficos o imágenes. Tienen propiedades que pueden ser modificadas en función de los requerimientos de la aplicación.

- *Panel* o tablero, se usa para agrupar elementos del *GUI* en grupos según las funciones que realicen o a gusto del programador, lo cual hace más amigable el diseño.

La Fig. 6 muestra la herramienta *GUIDE* inicializada lista para comenzar un diseño [8][14].

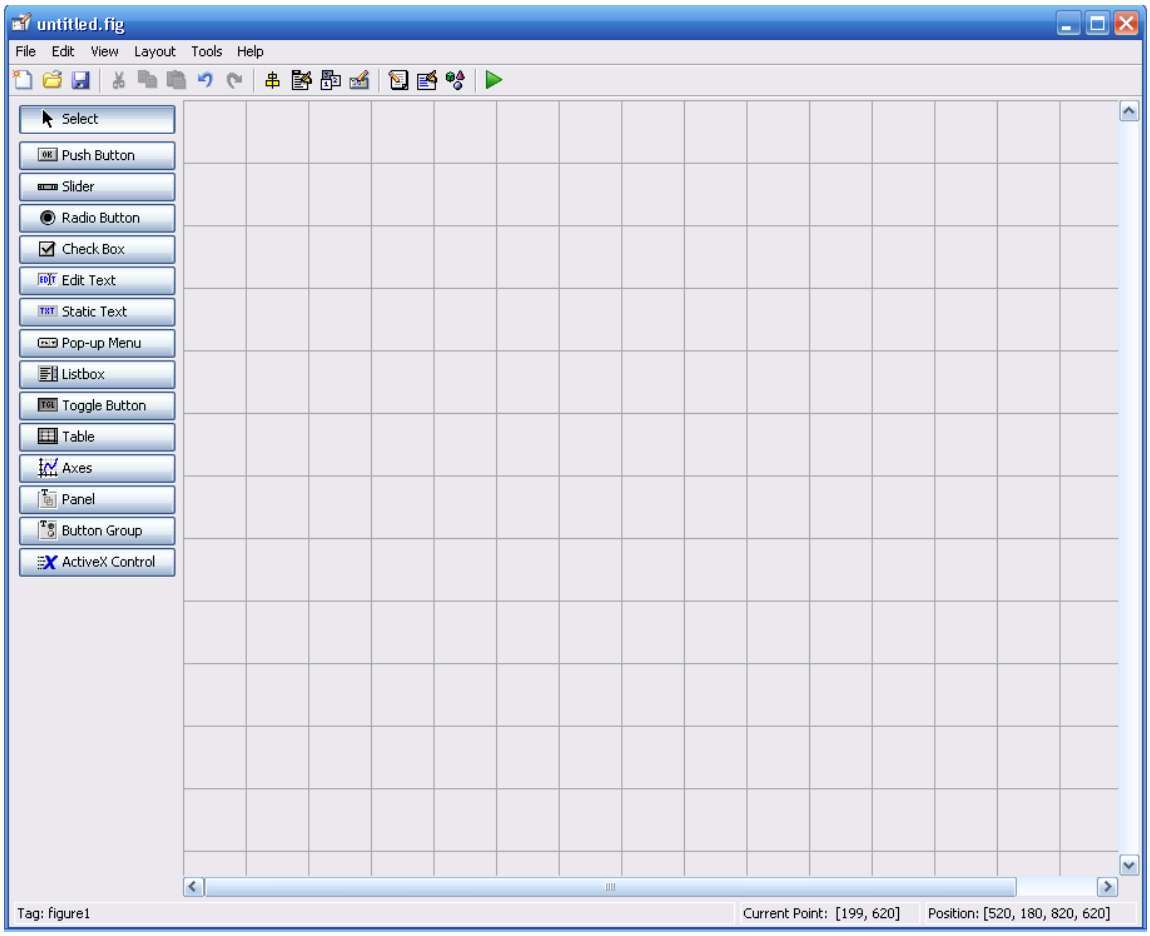


Fig. 6. Herramienta *GUIDE* lista para comenzar el diseño.

Una de las bondades que presenta el desarrollo de un *GUI* utilizando *Matlab* es la interactividad que presenta con el desarrollador y el amigable lenguaje de programación, por otro lado la propia herramienta genera de forma automática el código en un archivo **.m* asociado a cada uno de los elementos, dentro de este fichero se reservan espacios específicos donde el programador introduce segmentos de código correspondiente a las instrucciones específicas para cada elemento que compone la herramienta en desarrollo.

Cuando se desarrolla un *GUI* queda almacenado en dos archivos

- Un archivo con extensión **.m*, llamado *M-File* (archivo **.m*), mencionado anteriormente, el cual contiene el código que corresponde a cada elemento del *GUI*.
- Un archivo con extensión **.fig*, llamado *FIG-File* (archivo **.fig*), el cual contiene la estructura gráfica del diseño y de la configuración de los componentes del *GUI*.

De modo que cuando se trabaja en la componente gráfica, es almacenado en el archivo **.fig* y cuando se modifica el código de programación de los elementos del *GUI* es almacenado en el archivo **.m* [9] [10].

1.8 *Toolbox* de Procesamiento de imágenes

El procesamiento digital de imágenes es ampliamente utilizado hoy día, con varios fines dentro de los que se encuentra filtrado, realce de bordes, segmentación, extracción de patrones, entre otros. La caja de herramientas (en inglés *Toolbox*) de Procesamiento Digital de Imágenes que presenta el *software Matlab*, proporciona un conjunto completo de algoritmos de referencia-estándar y herramientas gráficas para el procesamiento, el análisis y la visualización de imágenes y el desarrollo de algoritmos. Puede realizar mejoras de imágenes, enfoque de imágenes borrosas, detección de funciones, reducción de ruidos, segmentación de imágenes, transformaciones geométricas y registro de imágenes. Muchas de las funciones del *toolbox* son multiproceso, para aprovechar los ordenadores con varios núcleos o procesadores [9].

Este *toolbox* soporta un conjunto diverso de tipos de imágenes, tales como las de alto rango dinámico, las de resolución de gigapíxeles, las de perfiles ICC embebidos y las tomográficas. Las herramientas gráficas le permiten explorar una imagen, examinar una región de píxeles, ajustar el contraste, crear contornos o histogramas y manipular regiones de interés (*ROIs*). Con los algoritmos de la *toolbox* puede restaurar imágenes degradadas, detectar y medir funciones, analizar formas y texturas y ajustar el balance de color [9].

1.9 Conclusiones parciales

El presente capítulo expone información sobre las imágenes médicas en formato digital. Realiza una explicación sobre las cámaras Gamma, y el principio de funcionamiento de las mismas e introduce el tema de la gammagrafía planar, su concepto y aplicación en la

medicina. Hace referencia además, a las imágenes obtenidas con la cámara Gamma *Sophy Circular* 1000, a las especificaciones técnicas, bondades y limitaciones de la misma.

También hace referencia a la utilización del software *Matlab*, sus orígenes, funciones y herramientas, y sus ventajas para el desarrollo del procesamiento digital de imágenes, lo que lo convierte una alternativa para el diseño de una interfaz gráfica para el tratamiento de imágenes médicas, aplicación que tendrá como futuro su implementación en lenguajes de programación de alto nivel para un mejor desarrollo y utilización por parte del personal médico.

CAPÍTULO 2. MATERIALES Y MÉTODOS

Con el objetivo de visualizar las imágenes provenientes de la cámara Gamma en una computadora con plataforma *Windows* se realizan una serie de experimentos de diseño con el software *Matlab*.

- Implementación de una herramienta para manipular las imágenes en formato *Sopha*
- Programación para el cálculo de medidas, área y distancia sobre la imagen
- Programación de funciones útiles para realizar un diagnóstico de la manera más similar posible a las funciones y métodos usados por la propia cámara
- Implantación de funciones específicas que permitan mejorar el diagnóstico, como son rotar las imágenes, mapas de falso color, selección de regiones de interés, impresión de imágenes, entre otras.

2.1 Software *Matlab*

El software *Matlab* por sus prestaciones y gran cantidad de funciones incorporadas relacionadas con el tratamiento de imágenes es utilizado como material fundamental para el diseño y construcción de la herramienta para manipulación y estudio de las imágenes planares. En su totalidad las funciones utilizadas para el manejo y análisis de las imágenes pertenecen al *software Matlab* y algunas específicamente al *toolbox* de Procesamiento de imágenes con el fin de obtener un mejor rendimiento en el producto final. Haciendo uso de la herramienta *GUIDE*, que brinda la posibilidad de crear interfaces gráficas, se diseña tanto la interfaz principal de la herramienta como sus complementarias.

2.2 Conjunto de imágenes de prueba

Las imágenes usadas en el presente trabajo fueron obtenidas de la cámara Gamma *Sophy Circular* 1000 del hospital "Celestino Hernández Robau". Dichas imágenes, mostradas en la Fig. 7, fueron extraídas mediante discos de 5¼ pulgadas, presentan una resolución de 128 x 128 píxeles y se encuentran en el formato *Sopha*, con extensión **.DAT*. El formato de las imágenes extraídas es *Sopha* y es en entero de 16 bits sin signo.

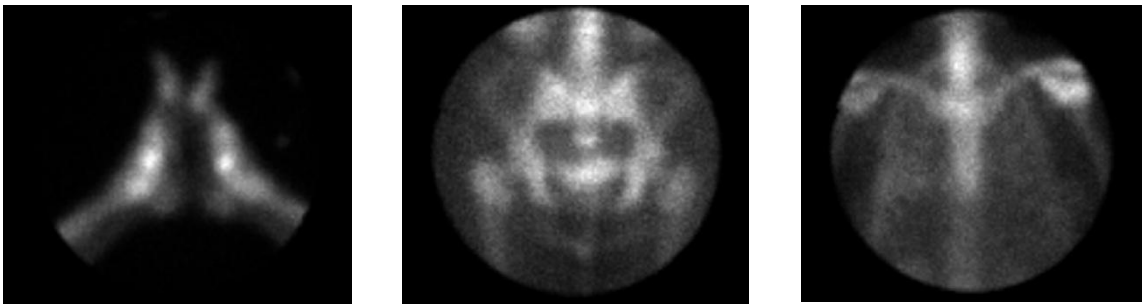


Fig. 7. Imágenes extraídas de la cámara Gamma.

2.3 Diseño de la herramienta

En el desarrollo de las *GUI* intervienen múltiples factores, tales como: usuario a quien va dirigida la aplicación, tema a tratar, medio en el que se utilizará, además de considerar aspectos de usabilidad que faciliten el manejo de las aplicaciones desarrolladas.

El procedimiento seguido para el diseño de la interfaz gráfica se muestra en el esquema a continuación en la Fig. 8.

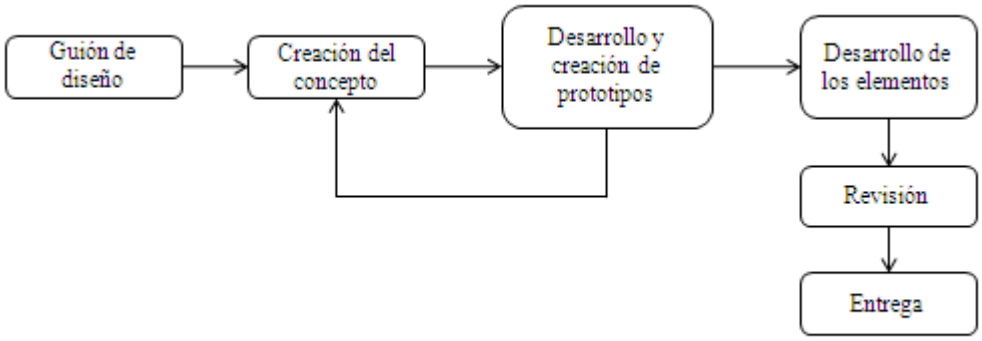


Fig. 8. Flujo de procesos para la elaboración de la interfaz gráfica de usuario

El proceso de creación de las interfaces gráficas de usuario se comienza a partir de un guión de diseño, en el que se define el contenido de cada elemento y el esquema que conformará la aplicación [11].

El objetivo de este proceso es la creación de un concepto visual acorde a la temática de la aplicación y el usuario final. Después de la revisión del guión de diseño, se hace necesario tener conocimiento general acerca del contenido de la aplicación, temática, público a quien va dirigido [11].

Sobre la base del concepto desarrollado, se realizan bocetos o propuestas de diseño para la aplicación. Estas propuestas son evaluadas por el departamento que utilizará la aplicación, a partir de lo que se determina una solución que satisface los requerimientos del material [11].

Una vez desarrollados los elementos de la herramienta, se envía al departamento de medicina nuclear en el hospital donde es evaluado que todos los elementos cumplan con los requerimientos de la aplicación [11].

2.3.1 Estructura de la herramienta

Una buena práctica para el desarrollo de interfaces de gráficas, es hacer un diseño estructural que consiste en realizar un esquema previo de cómo se van a visualizar cada una de las funciones determinando componentes comunes y singulares en cada de ellas.

Entre los elementos que se incluyen en la herramienta se encuentran:

- Encabezado. Ubicado en la parte superior de la herramienta, contiene el logo que identifica la aplicación y el nombre de la misma.
- Menú. Ubicado debajo del encabezado, para facilitar una navegabilidad rápida.
- Componentes generales. Panel de acciones para un acceso directo a las funciones más representativas, pantalla de visualización, entre otras.
- Notificaciones. Ventanas que muestran resultados y mensajes de alerta. [12].

2.3.2 Personalización de los botones

Debido a que los futuros usuarios de la herramienta serán el personal médico se hace necesario que el acceso a las funciones de la herramienta sea lo más intuitivo posible, para

ello cada botón se personaliza de manera que los usuarios tengan una apreciación visual de la función que realizan. El algoritmo general para fijar los íconos de los botones se muestra en la Fig. 9. La función *imread* permite cargar el ícono que se encuentra en formato *JPG*. Con *size* se hace referencia al tamaño que tendrá el ícono y con *ceil* se fijan valores que tendrá e ícono para las coordenadas verticales y horizontales. Por último se fija la imagen al botón deseado mediante la función *set* [9] [10].

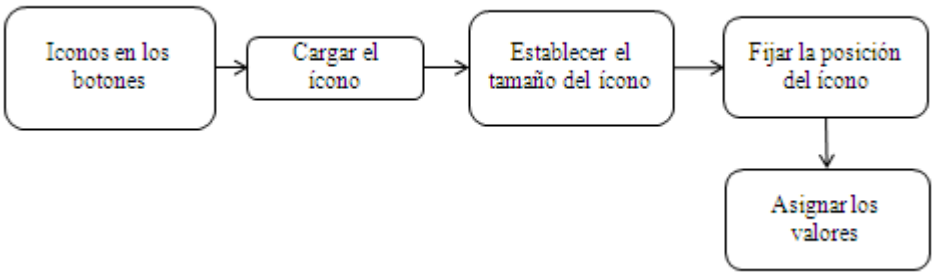


Fig. 9. Algoritmo para personalización de los botones.

2.4 Interpretación de las imágenes en formato *Sopha*

Para leer las imágenes en formato *Sopha* el primer detalle a destacar es que el fichero tiene formato *BIG-ENDIAN*, esto está relacionado con el orden en que se almacenan los datos y quiere decir ¨Del más grande al final¨. Los formatos en que se almacenan los datos se les conoce como Endianess, siendo este formato donde se comienza almacenando del byte más al menos significativo, exactamente igual que como solemos representar los números, siendo el primer byte el que tiene más peso en la secuencia.

Mediante la función *fopen* se obtiene el identificador del fichero y se le pasa a la función *fread* que es capaz de leer datos de un fichero binario hacia el espacio de trabajo de *Matlab*. Dado que el dato leído es un fichero binario lo que se obtiene es una cadena de valores, estos valores hay que reorganizarlos y obtener la matriz que representa la imagen mediante la función *reshape*. Finalmente los valores de la matriz obtenida se convierten a una imagen de intensidad de grises con valores entre 0 (negro) y 1 (blanco) mediante la función de *Matlab mat2gray*.

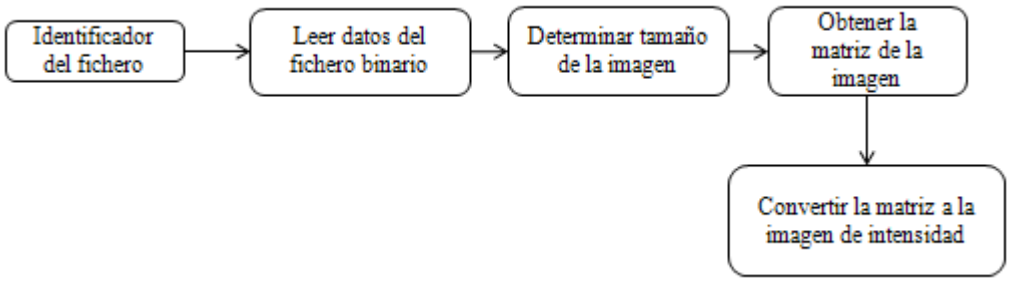


Fig. 10. Algoritmo de interpretación de las imágenes en formato *Sopha*.

2.5 Funciones básicas sobre las imágenes

Teniendo en cuenta que la herramienta en desarrollo será utilizada para la manipulación de imágenes debe incluir un conjunto de funciones básicas a realizar sobre las imágenes en estudio, que no son más que las comúnmente utilizadas en programas de visualización de imágenes: cargar la imagen, guardar la imagen en varios formatos, rotar la imagen, acercar la imagen (*zoom*), entre otras.

2.5.1 Cargar la imagen

El proceso de cargar la imagen en la herramienta se realiza por diferentes vías dependiendo del formato en que se encuentre para lo que se implementan diferentes mecanismos. El principal formato, el de la imagen obtenida de la cámara Gamma **.DAT*, resulta ser el de mayor complejidad por no ser compatible el formato *Sopha* con la plataforma *Windows*. En la herramienta se incluye un botón, para cargar archivos **.DAT*, el mismo activa la lectura de la imagen proveniente de la cámara directamente. Para ésta y para los otros formatos de imágenes que pueden cargarse como JPG y DICOM, se implementa un explorador que facilita la búsqueda del fichero contenedor de la imagen, también se incluye un filtro que ayuda a buscar por formatos de imágenes.

2.5.2 Guardar la imagen

Para guardar las imágenes se implementa un algoritmo que permite la escritura del fichero de la imagen en el formato más comúnmente usado, JPG. Mediante la función *uiputfile* se llama un explorador para designar la dirección de la imagen a guardar.

El algoritmo de escritura de la imagen quedaría del siguiente modo, se obtiene la imagen actual en pantalla con la función *getimage* y el rango dinámico de la imagen se lleva a valores entre 0 y 255. Mediante la función *fullfile* se define la dirección y el nombre del fichero contenedor de la imagen y luego usando la función *imwrite* queda guardado dicho archivo con el formato definido respectivamente. Para las imágenes en formato JPG y DICOM es muy similar. La Fig. 11 muestra un diagrama en bloques del procedimiento general de la escritura de la imagen en los diferentes formatos [9] [10].

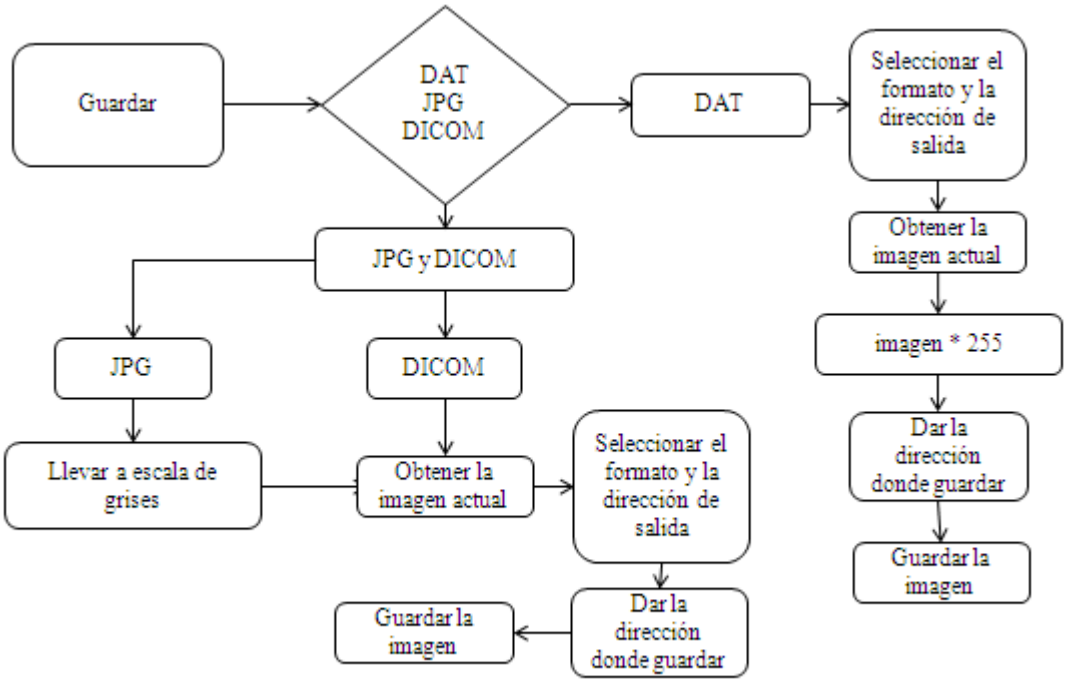


Fig. 11. Diagrama en bloques que representa el algoritmo para guardar una imagen.

2.5.3 Rotar la imagen

El proceso de rotar las imágenes en sentido horario y anti horario es una acción que apropiadamente se ubica en el panel de Acciones o en el Menú Herramientas. La Fig. 12 muestra un diagrama en bloques que resume este algoritmo. Primeramente su usa la función *getimage* para obtener la imagen actual que se está visualizando en pantalla y la función *imrotate* para rotar la imagen, a la cual se le especifican como parámetros de entrada la imagen a rotar y el ángulo de rotación. Si se especifica el ángulo de rotación de 90 grados la imagen rotará en sentido anti horario y viceversa si se le especifica el ángulo

de rotación -90 grados para rotar a la derecha. Una vez que la imagen ha sido rotada se usa la función *imshow*, la cual se encarga de mostrar la imagen resultante. Uno de los parámetros que es necesario especificar a la función *imshow* es el mapa de colores que se está usando en ese momento para garantizar el modo de visualización del usuario, esto será explicado en epígrafes posteriores [9] [10].

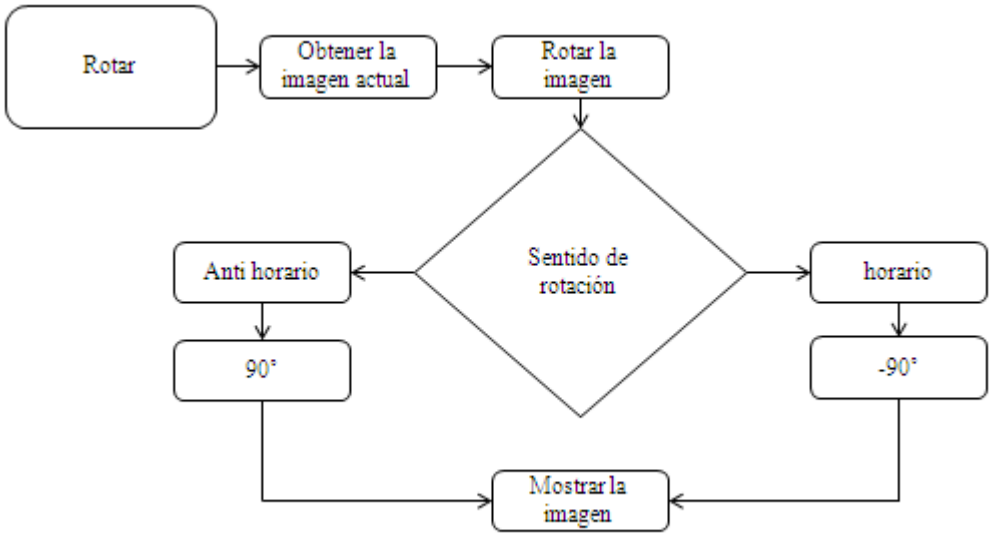


Fig. 12. Diagrama en bloques que representa el algoritmo para rotar las imágenes.

2.5.4 Acercamiento (*zoom*)

El acercamiento comúnmente conocido como *zoom* por su traducción del inglés se realiza usando un *Toggle Button (botón de alternar)*, que brinda la posibilidad de activar y desactivar la función a través de un solo botón dependiendo si está en alto en bajo. Primeramente se obtiene la imagen actual que se está visualizando en pantalla del mismo modo que se explica en epígrafes anteriores. Al activar el acercamiento o *zoom* mediante la función de *Matlab* llamada propiamente *zoom*. Por último se fijan los parámetros de la función *zoom* que permitieran realizar los cambios a la imagen como muestra la Fig. 13.

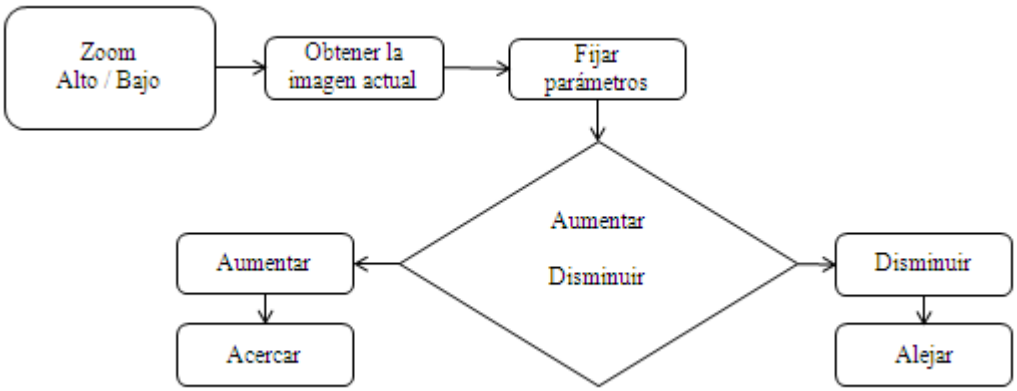


Fig. 13. Representación del zoom mediante un diagrama en bloques.

Los parámetros que se le deben especificar a la función de acercamiento son la movilidad de los ejes de coordenadas mediante la función *Motion* y la dirección del acercamiento, ya sea aumentar o disminuir. Para aumentar se establece en *in* y *out* para disminuirlo [9] [10].

2.5.5 Desplazamiento o *Pan*

La acción de realizarle el desplazamiento del inglés *Pan* a una imagen está estrechamente relacionada con el acercamiento. Lo cual no es más que mover la imagen en cualquier dirección, una vez que esté tan aumentado el tamaño que sus bordes se salgan de la pantalla. Haciendo uso de un *Toggle Button* para activar o desactivar la función de desplazamiento de la imagen basta con establecer el parámetro activo (*Enable*) en *on* para activarla o en *off* para desactivarla. La Fig. 14 muestra una explicación mediante un diagrama en bloques [9] [10].

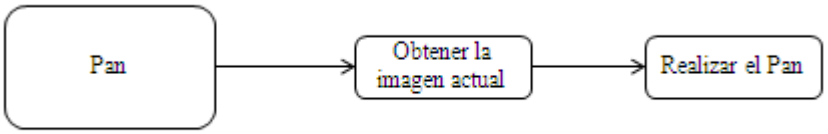


Fig. 14. Representación de la acción Pan.

2.5.6 Mapa de colores

A petición de los especialistas se incluye la opción de incluirle falso color a las imágenes de modo que la visualización se realice lo más parecido posible a los mapas de color que

brinda el *software* de la cámara Gamma. Se utilizan la mayoría de los mapas de colores que brinda el *toolbox* de Procesamiento de imágenes *Matlab*, de acuerdo con lo especificado por los especialistas médicos. Para la selección del mapa de colores se hace uso un menú desplegable o *Pop-up Menú* el cual brinda la posibilidad de desplegar una lista con las diferentes opciones a seleccionar por el usuario. La Fig. 15 muestra un diagrama en bloques que representa el algoritmo de funcionamiento. Para generar el mapa de falso color se utiliza la sintaxis *switch* que es un código de ejecución condicional que permite salidas alternativas en dependencia del valor que tome la variable de entrada, o sea que ejecuta una de las de las declaraciones seleccionadas desde un número arbitrario de alternativas. Cada una de estas alternativas de entrada es denominada mediante el comando *case* y a cada alternativa se le asigna un mapa de falso color que posteriormente es asignado a la imagen. Primero se obtiene la imagen actual que se muestra en pantalla del mismo modo que se explica en epígrafes anteriores, luego se iguala el mapa de color que se obtiene mediante la función *colormap* (y aquí se pone el nombre del mapa de color deseado) a una variable. A la hora de usar el mapa de colores basta con hacer referencia a la variable que lo contiene [9] [10].

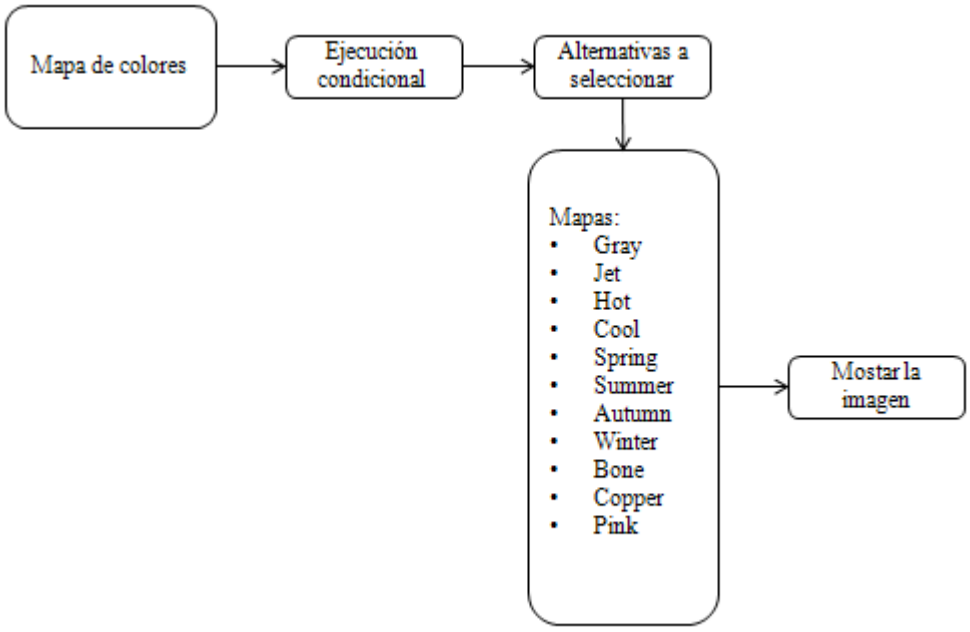


Fig. 15. Diagrama en bloques que representa el algoritmo del mapa de colores.

2.6 Funciones más específicas y otras aplicaciones

Además de las funciones básicas que se le incluyen a la herramienta, similares a las de cualquier herramienta de manipulación de imágenes convencional, también se le agrega otro conjunto de funciones más específicas a fin a la medicina nuclear. Muchas de estas funciones fueron incluidas teniendo en cuenta las necesidades de los especialistas médicos.

2.6.1 Medición de distancia sobre la imagen

Uno de los procedimientos más comúnmente utilizado en el diagnóstico médico por imagen es la medición de distancia. *Matlab* provee varias alternativas para medir distancia entre dos puntos, la alternativa que se utiliza es la implementada en la función *imdistline* porque que es el método más similar al utilizado por las herramientas de diagnóstico médico, consiste en una línea que se le ajusta la posición de los extremos en las posiciones que se quiere medir la distancia y ésta devuelve el resultado automáticamente en una etiqueta. En la Fig. 16 se muestra el diagrama general de la implementación de la opción de medir distancia, debido a que la función utilizada muestra una etiqueta con la distancia en píxeles se hace necesario la conversión a milímetros (mm), que es la medida convencionalmente utilizada en el diagnóstico, a esto se le agrega que no siempre las imágenes pueden ser obtenidas con iguales resoluciones, por lo que la relación píxel mm no siempre será la misma. Se necesita de algún modo facilitar la tarea de convertir de píxeles a mm para ello se incluye una opción donde el especialista introduce la resolución con la que se obtuvo la imagen y devuelve la distancia en milímetros mediante dos (2) casillas llamadas Dist Píxeles y Conv Dist respectivamente que son en las que los usuarios introducen los valores y finalmente obtener el valor real de la distancia en milímetros [9].

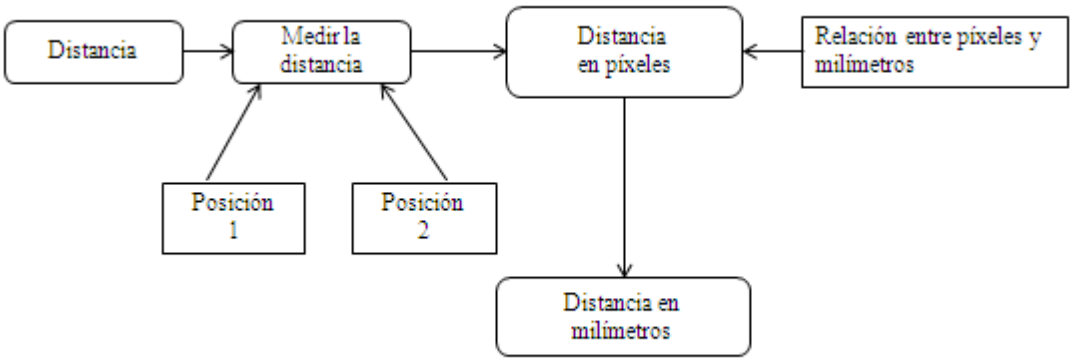


Fig. 16. Diagrama en bloques de la medición de distancia.

2.6.2 Medición de área sobre la imagen

La medición de área es otra de las acciones comúnmente utilizadas en el diagnóstico por imagen, a petición de los especialistas, se incluye esta opción. En la Fig. 17 se muestra el diagrama en bloques del algoritmo que permite medición del área. Para realizar este proceso se obtiene la imagen actual que está en pantalla del modo explicado en epígrafes anteriores, mediante la función del *toolbox* de PDI de *Matlab* nombrada *roipoly* se crea un polígono cuyos puntos son seleccionados y modificados por el usuario, de modo que la lesión queda encerrada dentro del mismo y representa el área a medir, esta función devuelve una máscara binaria en la que los píxeles dentro del polígono son de valor 1 (blanco) y los que están fuera 0 (negro). Teniendo la máscara binaria se usa la función *bwarea*, se calcula el área total de todos los píxeles que se encuentran en 1 (blanco). Se introduce el valor equivalente para convertirlo en milímetros en la casilla denominada Conv Ar, el cual es multiplicado por el valor obtenido de la función de cálculo de área según la región seleccionada [9] [10].

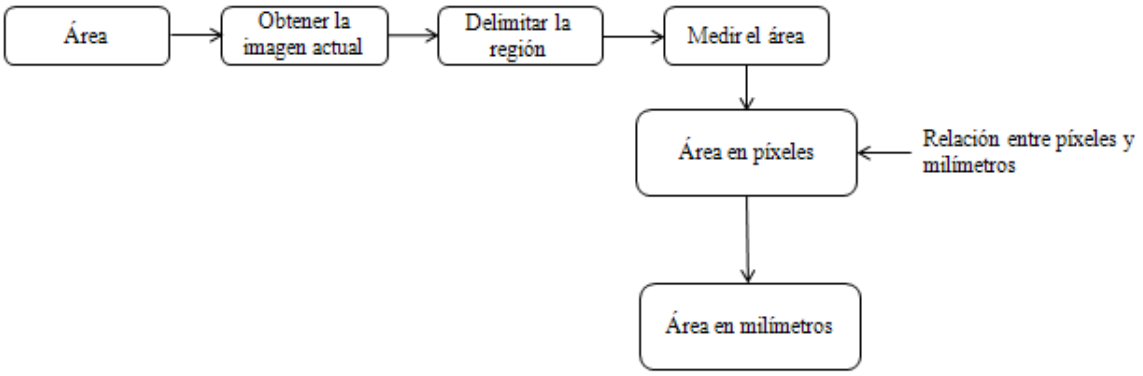


Fig. 17. Diagrama en bloques de la medición del área.

2.6.3 Inclusión de los datos del paciente

Una de las funciones más útiles y necesarias para el diagnóstico médico es la inclusión de texto en la imagen, ya que este texto puede estar relacionado con los datos del paciente, la institución que realiza el diagnóstico, datos del especialista que realiza el examen, etc. Para incluir el texto se implementa el algoritmo mostrado en la Fig. 18.

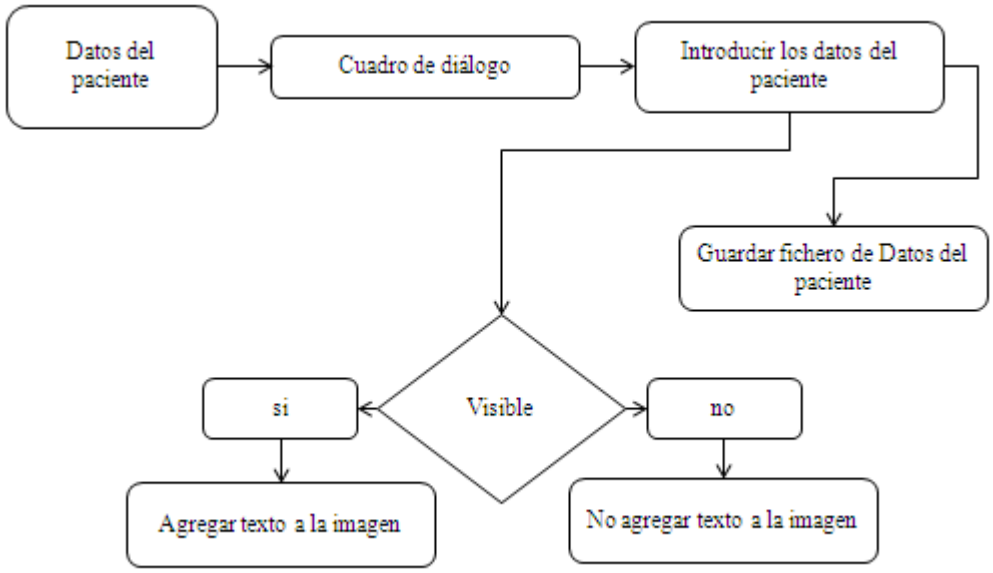


Fig. 18. Diagrama en bloques de la inclusión de los datos del paciente.

Para que el usuario introduzca los datos se crea un cuadro de diálogo mediante la función *inputdlg* de *Matlab* que permite al usuario introducir una determinada cantidad de datos

según la configuración realizada por el programador, una vez se introducen todos los datos la función *text* de *Matlab* permite introducir los datos en la imagen, con lo cual se pueden configurar opciones como tamaño de letra, posición y color [9] [10].

2.6.4 Intensidad del píxel

En la gammagrafía esta opción tiene gran importancia ya que permite conocer de manera más exacta la cantidad de conteo en las regiones que se estudian según la intensidad de los píxeles. El cálculo del valor de la intensidad de los píxeles se realiza como se demuestra en el diagrama de la Fig. 19. Mediante la función *imdistline*, se obtiene la intensidad del píxel en un intervalo entre cero y uno y su ubicación en la imagen, correspondiendo el mayor valor de intensidad, al píxel con mayor brillo [9] [10].

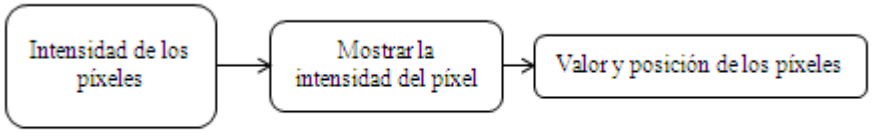


Fig. 19. Diagrama en bloques que representa la obtención de la intensidad de los píxeles.

2.6.5 Selección de región de interés

Mediante la selección de regiones de interés se puede hacer comparaciones entre regiones específicas. También es posible exportar la región de interés en formatos conocidos del mismo modo que se explica en epígrafes anteriores El algoritmo para exportar una región de interés es el siguiente: primero se obtiene la imagen actual que está en pantalla del modo explicado en epígrafes anteriores, se usa la función *rect*, para extraer de la matriz de la imagen original seleccionada por el usuario, después mediante la función *imshow* se muestra la imagen resultante de la selección en una ventana diferente. La Fig. 20 hace una representación del algoritmo [9] [10].

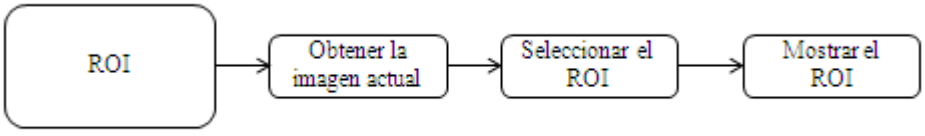


Fig. 20. Diagrama en bloques que representa la obtención de la región de interés.

2.6.6 Impresión de imágenes

A fin de tener a disposición de especialistas la información en formato duro de las imágenes de un paciente se implementa la impresión de las imágenes como un método alternativo para el diagnóstico. Para imprimir las imágenes se emplea el algoritmo descrito en la Fig. 21. Se obtiene la imagen actual que se muestra en pantalla como se explica en epígrafes anteriores, y con la función *printpreview* se hace uso de un cuadro de opciones de impresión, que brinda la posibilidad de cambiar o ajustar la imagen a imprimir con varias opciones como son el formato de impresión, el tamaño de la imagen, color y fondo de la imagen [9] [10].

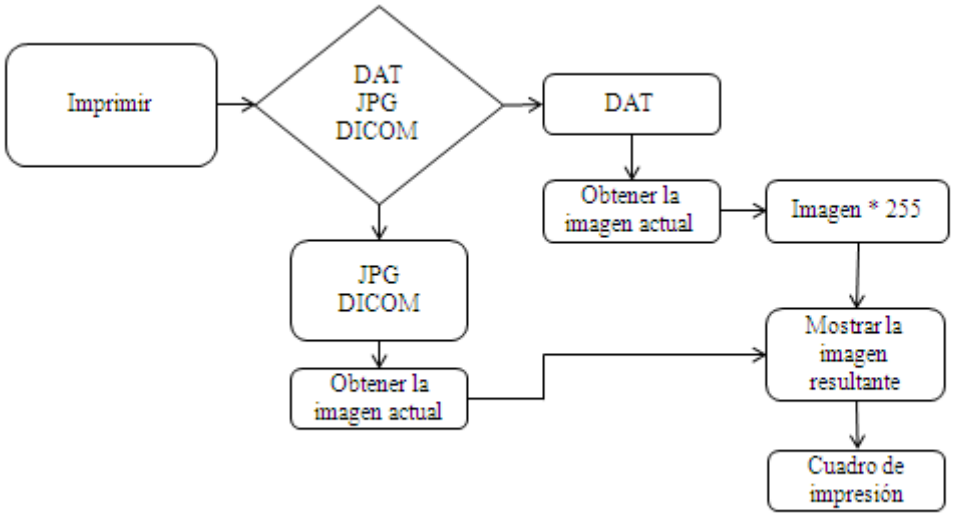


Fig. 21. Diagrama en bloques que representa la obtención de la región de interés.

2.6.7 La ayuda

La ayuda del programa está dirigida al usuario, en ella se describe el uso detallado de la herramienta, las funciones que posee y el modo de operación. Brinda los requerimientos del sistema, es decir, que espacio y requerimientos de la computadora son necesarios para poder usar la herramienta. Se realiza en formato *. *htm* y se usa para llamarla la función *open* [9] [10].

2.7 Conclusiones parciales

En este capítulo se han presentado los materiales y métodos que se utilizan para diseñar una interfaz gráfica con vista a manipular las imágenes provenientes de la cámara Gamma *Sophy Circular* 1000. Se han presentado las imágenes que se utilizan y sus principales características, el conjunto de funciones que realiza la herramienta y se ha descrito los métodos de programación desarrollados.

CAPÍTULO 3. RESULTADOS Y DISCUSIÓN

3.1 La herramienta

A continuación se muestra el ejemplo de la herramienta la cual se nombra *VGamma* de las palabras Visor Gamma propiamente seleccionadas para ser asociadas con las imágenes para la que fue desarrollada. En la Fig. 22 se puede apreciar que la herramienta posee una interfaz amigable y sencilla. Este es el principal resultado de la investigación, diseñar la herramienta *VGamma* que es capaz de cargar las imágenes de la cámara Gamma y permitirle a los especialistas realizarle análisis.

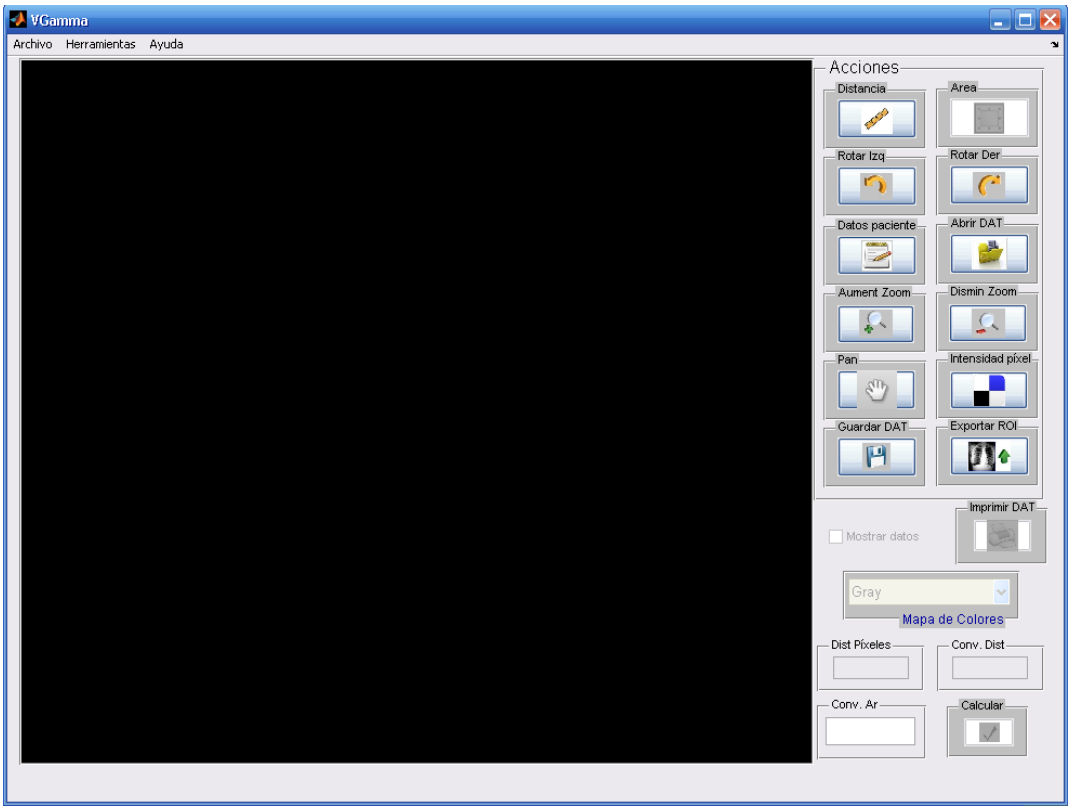


Fig. 22. Herramienta *VGamma*.

3.2 Funciones básicas que realiza la herramienta VGamma

A continuación se ejemplifica con detalles las acciones básicas que posee la herramienta *VGamma*. Las acciones básicas consisten en cargar y guardar las imágenes, rotarlas, hacerles zoom y pan y aplicarles mapas de falso color.

3.2.1 Cargar las imágenes

La herramienta posee un botón ubicado en la parte derecha en el panel de Acciones, llamado Abrir DAT, que realiza la función de cargar las imágenes provenientes de la cámara Gamma. Para ésta y para los otros formatos de imágenes que pueden cargarse como JPG y DICOM se hace uso de un explorador que facilita la búsqueda por formatos de imágenes. Para cada tipo de imagen se usa un explorador específico según su formato. Una muestra del explorador para cargar las imágenes provenientes de la cámara se puede apreciar en la Fig. 23. De manera similar es para los formatos JPG y DICOM.

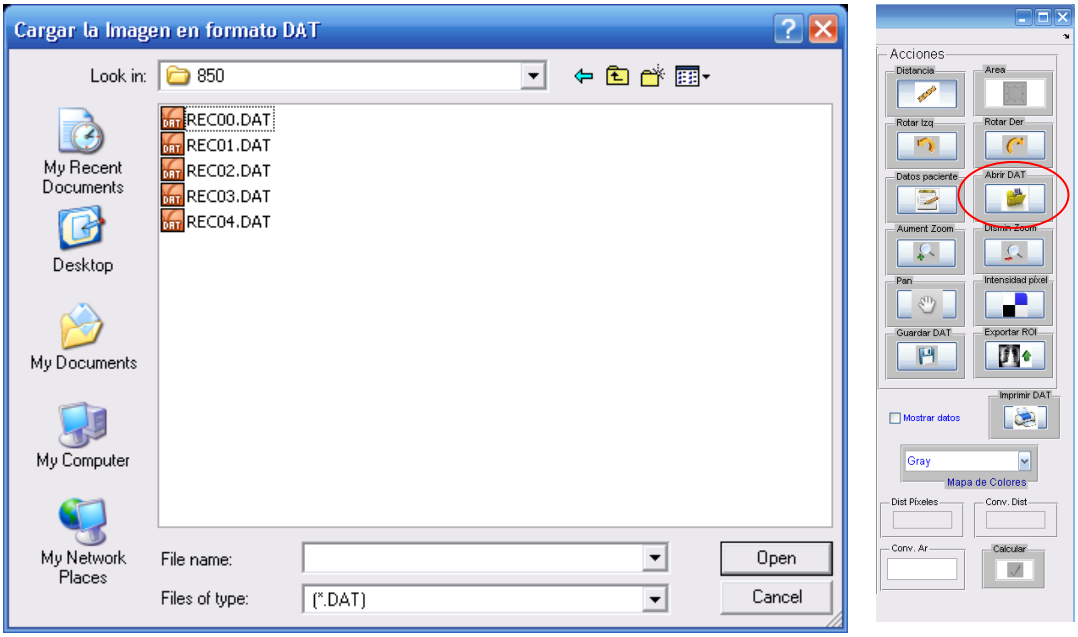


Fig. 23. Explorador para cargar las imágenes **.DAT*.

Para cargar imágenes en los formatos *JPG* y *DICOM*, se usa el submenú *Abrir .JPG* y *Abrir DICOM* respectivamente, que se encuentra en el menú *Archivo* (ver Fig. 24).

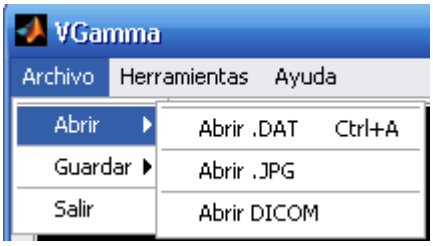


Fig. 24. Submenú Abrir.

3.2.2 Guardar las imágenes

Guardar las imágenes previamente cargadas y/o modificadas es algo que resulta necesario en muchos casi todos los casos. El objetivo de guardar la imagen es exportarla a otros formatos brindando la posibilidad de que sean cargadas por otros visores de imágenes compatibles con los formatos estándares *Windows*. Se implementaron diferentes formas de guardar las imágenes, en la Fig. 25 podemos ver el menú creado para guardar las imágenes. Al igual que para cargar las imágenes se desarrolló un explorador, que permite guardar las imágenes en diferentes formatos, JPG y JPEG (ver Fig. 26) de modo que el usuario puede especificar de una manera relativamente sencilla la dirección del fichero a guardar.

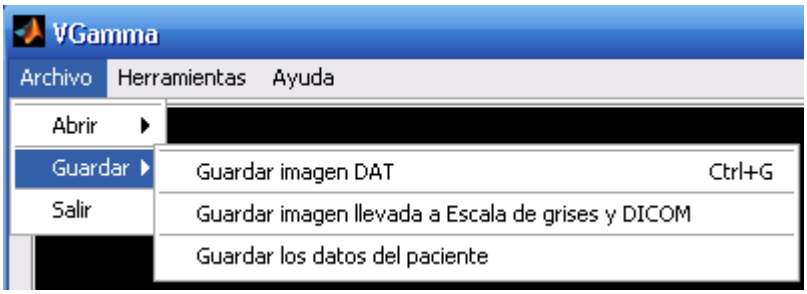


Fig. 25. Submenú Guardar.

La opción de guardar se configura de modo que las imágenes guardadas conservaran el mismo mapa de falso color que se estaba visualizando en el momento en que se guardó.

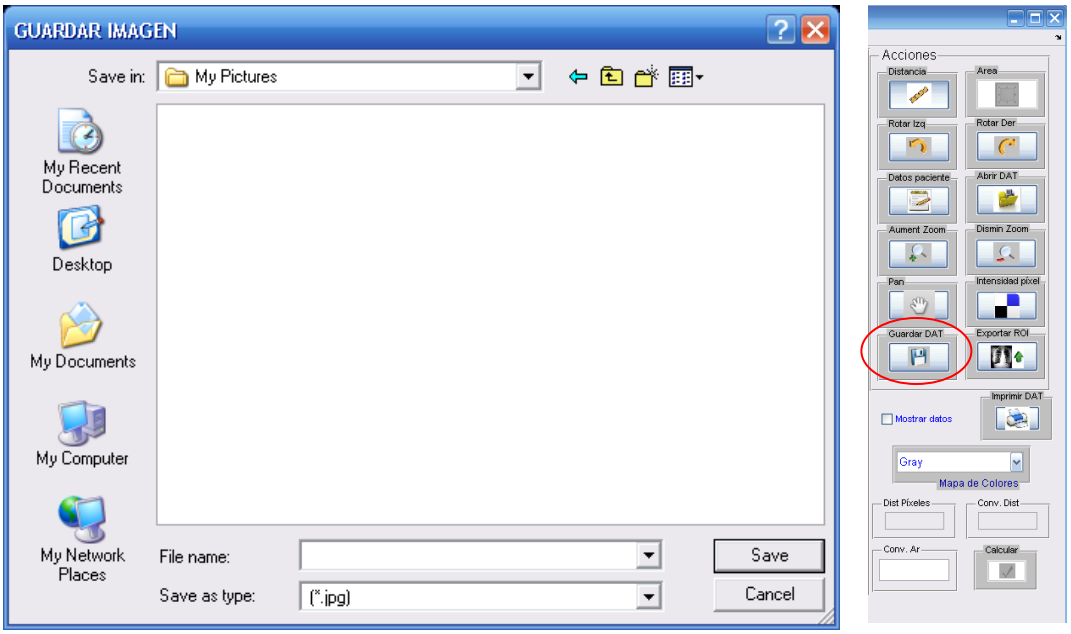


Fig. 26. Explorador para guardar las imágenes.

3.2.3 Rotar las imágenes

La opción de rotar la imagen en sentido horario y anti horario son acciones que se realizan directamente desde el panel de Acciones o desde el menú Herramientas. Las imágenes rotan hacia la derecha o hacia la izquierda según se desee siempre en ángulos de 90 grados. La Fig. 27 muestra el submenú Rotar del menú Herramientas. Si previamente no se carga una imagen esta función no realizará ninguna acción.

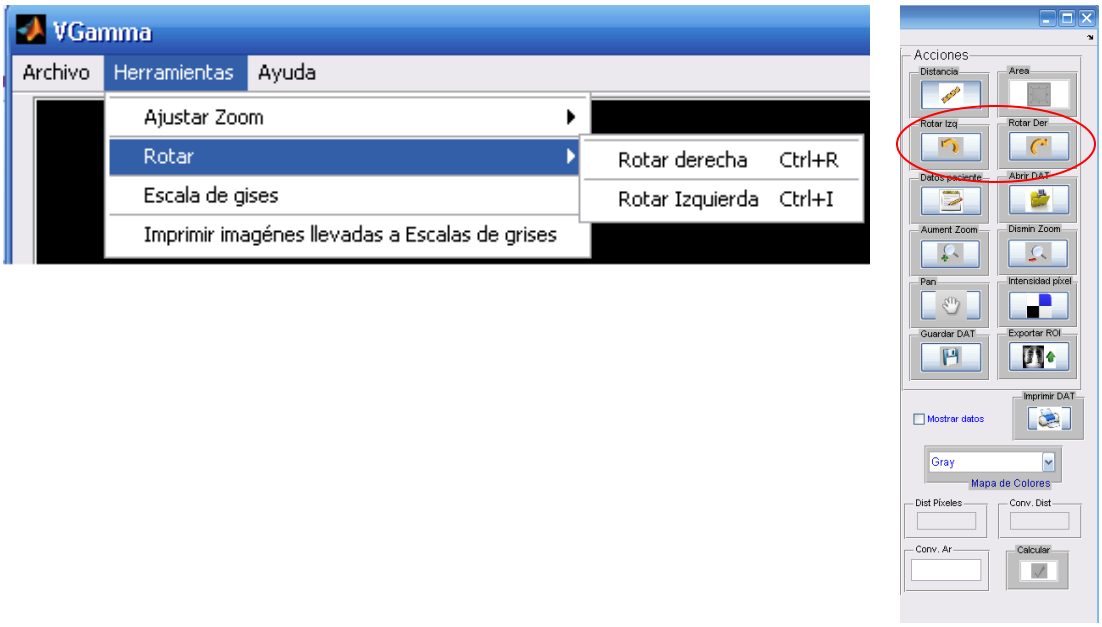


Fig. 27. Submenú Rotar en el menú Herramientas y en el panel Acciones.

3.2.4 Acercamiento o *Zoom*

Hacer un aumento de la imagen es una acción que permite visualizar con mejor facilidad los detalles de la misma. *VGamma* posee acceso a esta acción mediante botones que se encuentran en el panel de Acciones o bien, en una opción del submenú Ajustar *Zoom*, que se encuentra en el menú Herramientas. Para aumentar una porción de una imagen, la función de *zoom* creará un rectángulo que delimitará la región a aumentar. También, se puede hacer uso del botón central del mouse, con la cual se puede aumentar y disminuir el *zoom* si está activa la acción. Si la acción se realiza desde el menú Herramientas, estarán disponibles tres submenús, uno para aumentar, otro para disminuir y otro para desactivar. En la Fig. 28 se pueden apreciar las opciones de activar el *zoom*.

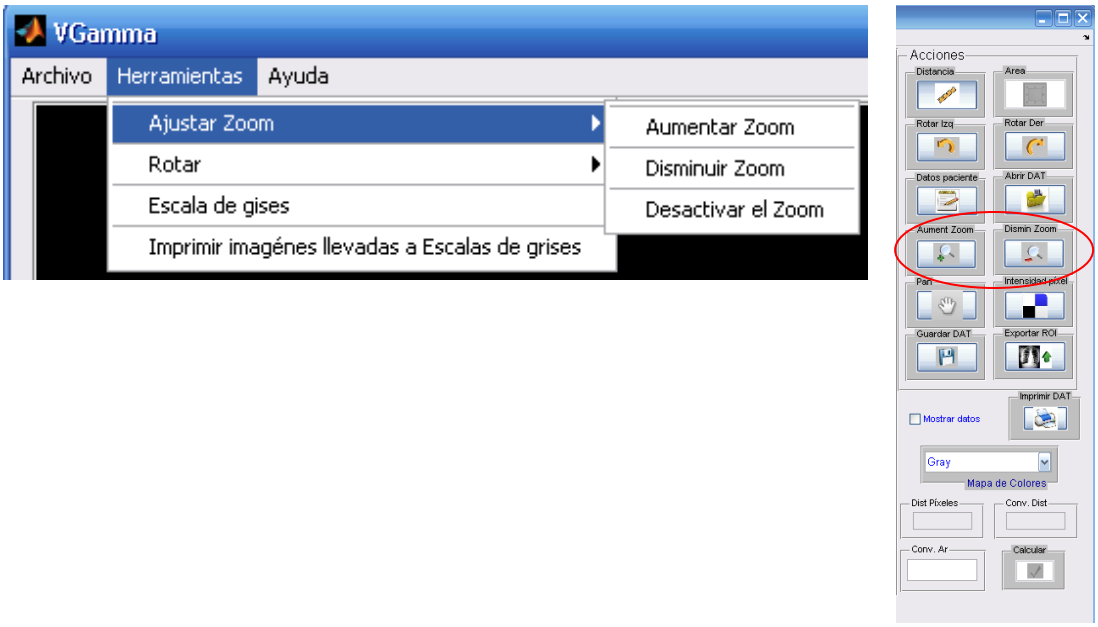


Fig. 28. Submenú Ajustar Zoom.

3.2.5 Desplazamiento o *Pan* a las imágenes

Esta acción solo se realiza mediante un botón en el panel de Acciones (ver Fig. 29) y se simboliza con una mano para asociarlo con el agarrar la imagen y desplazarla. Una vez ejecutada, el puntero se convierte en una mano con la cual se puede controlar el desplazamiento en la dirección que se desee. Para desactivar la acción se usa el mismo botón y la imagen mantiene la posición que se estableció.

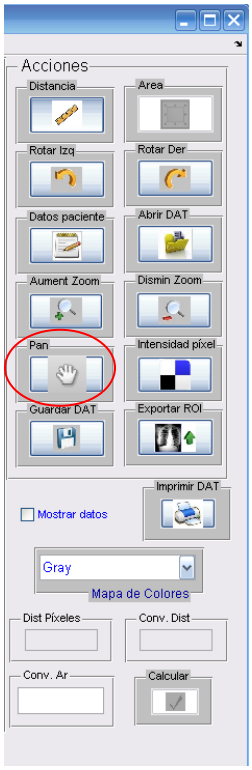
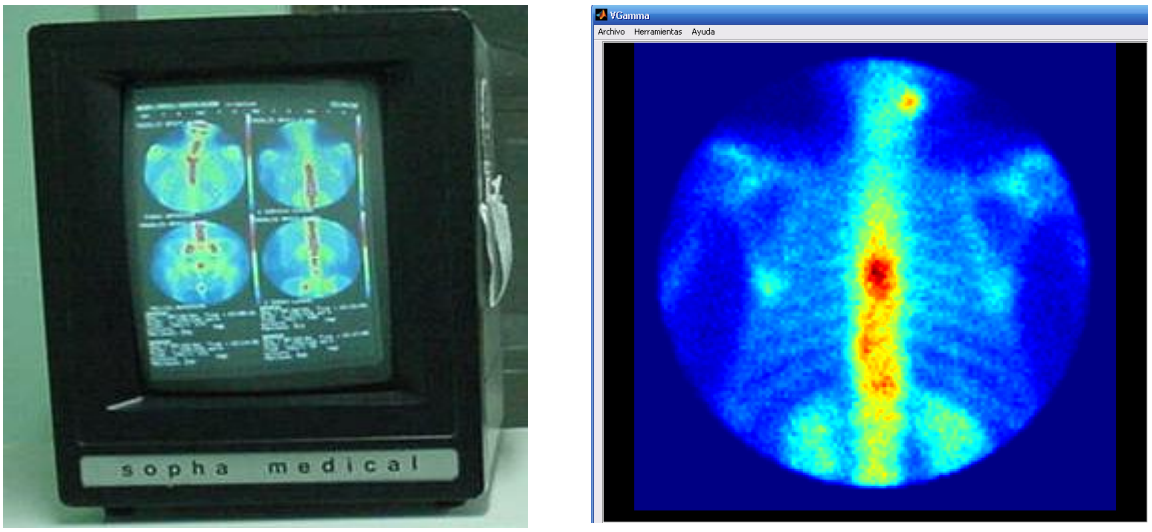


Fig. 29. Ubicación del botón Pan en el panel de Acciones.

3.2.6 Mapas de falso color en las imágenes

La acción de incluir falso color a la imagen brinda la posibilidad de incluir varios mapas según los requerimientos de los especialistas médicos. Esta opción se encuentra disponible debajo del panel de Acciones y se activará únicamente cuando una imagen es cargada, si es una imagen proveniente de la cámara Gamma o en formato DICOM. Si la imagen se encuentra inicialmente en formato JPG, deberá llevarse primeramente a escala de grises. La opción de color falso es uno de los detalles que tributa a que la herramienta se asemeje a el modo en que opera el *software* de la cámara Gamma. A continuación se muestra en la Fig. 30 una comparación entre la cámara y *VGamma* donde se visualizan imágenes con falso color, ambas con el mapa de color *Jet*.



(a) (b)

Fig. 30. Comparación de mapa de falso color del color Jet. a) en la cámara Gamma *Sophy Circular* 1000, b) herramienta *VGamma*.

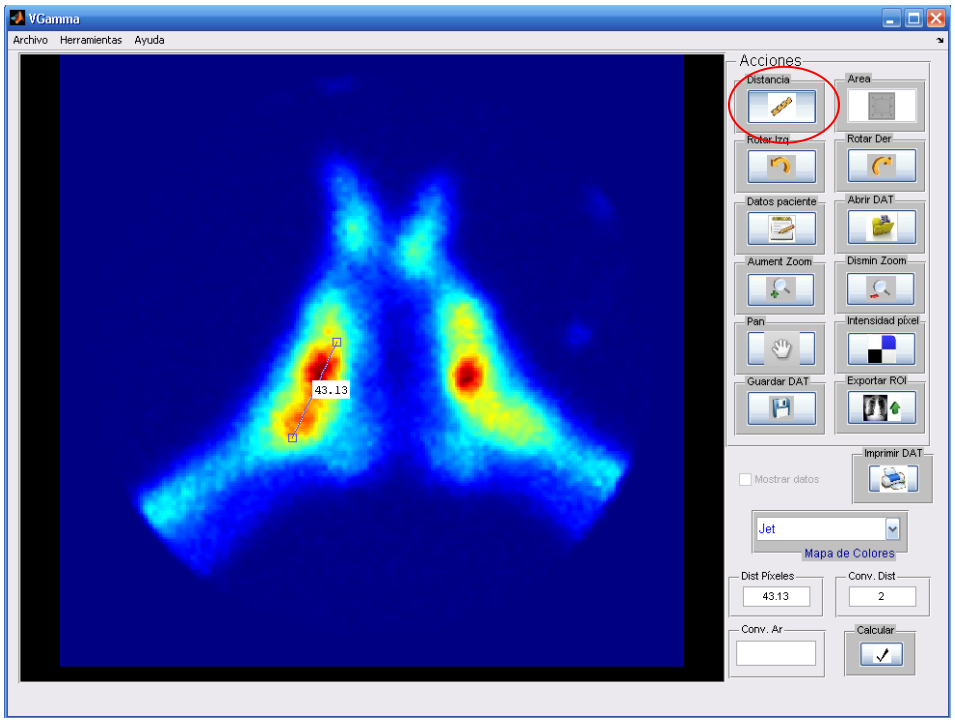
3.3 Funciones más específicas

VGamma posee un conjunto de herramientas de mayor complejidad en función de análisis más específicos. Estas funciones se realizan de manera que fuera amigable y cómoda a la hora de hacer los análisis.

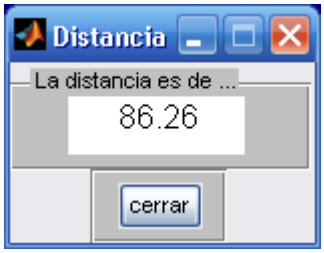
3.3.1 Medir Distancia

Para medir distancia se incluye un botón en el panel de Acciones el cual se simboliza apropiadamente para una fácil localización. Cuando se activa esta acción aparecerá sobre la imagen una línea horizontal por defecto con la distancia de 128 píxeles, que es mostrada en una etiqueta, moviendo lo extremos en cualquier dirección se ajusta la distancia que quiere ser medida. A petición de los especialistas se incluye la conversión de las distancias de píxeles a mm de manera independiente, por el hecho de que las imágenes no siempre se adquieren con la misma resolución. Para la conversión existe un panel en la parte de inferior derecha de la herramienta que permite, introducir la distancia mostrada en la etiqueta de la línea de medición y la relación entre los píxeles y los milímetros en la imagen

y devuelve en un cuadro de mensaje con la distancia en milímetros. . La Fig. 31 muestra un ejemplo del uso de esta aplicación.



(a)



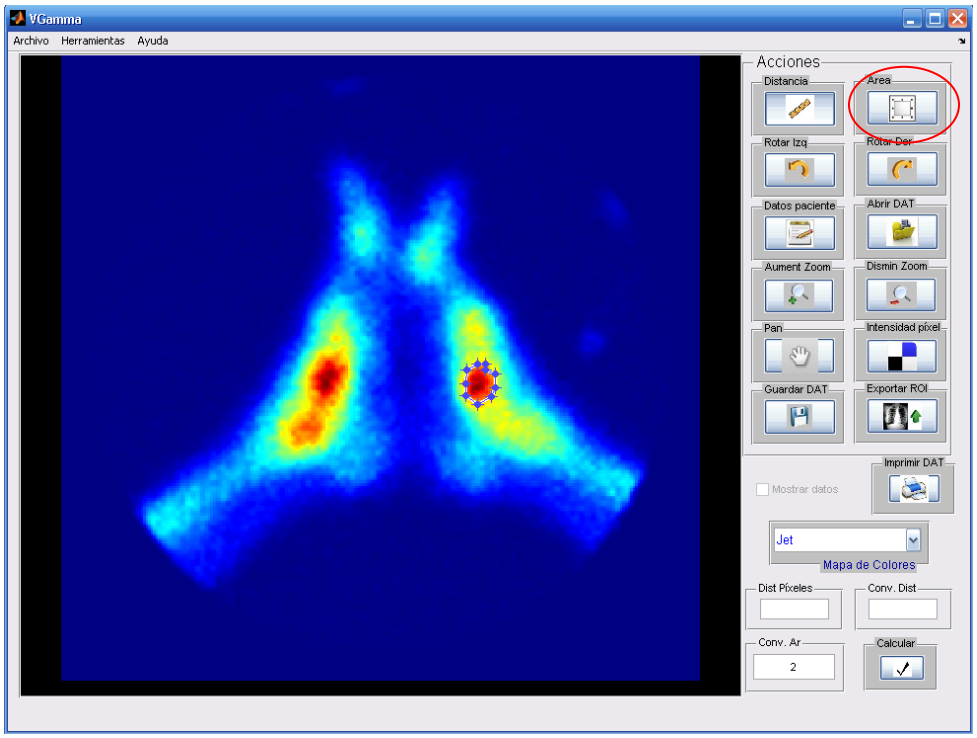
(b)

Fig. 31. Ejemplo de medición de distancia: (a) Uso de la herramienta para medir la distancia en píxeles y (b) ventana con la información de la distancia final en milímetros.

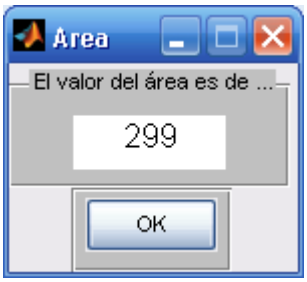
3.3.2 Medir el Área

Para realizar la medición de área en una región en una imagen se incluye un botón en el panel de acciones. La opción de realizar la medición de área mediante un polígono tuvo buena aceptación por parte de los especialistas porque brinda exactitud y se parece al método utilizado en el software de la cámara. La relación entre píxeles y milímetros es

calculada para el área igual que en la distancia. La Fig. 32 muestra un ejemplo del uso de la herramienta para hacer la medición de área.



(a)



(b)

Fig. 32. Ejemplo de medición de área: (a) Uso de la herramienta para realizar la medición de área a una región de la imagen y (b) ventana con la información del valor del área en milímetros.

3.3.3 Inclusión de los datos del paciente

A modo de introducir los datos pertenecientes al estudio que se realiza dígase, nombre del paciente, carnet de identidad (CI), edad, sexo, institución hospitalaria, patología y especialista que realiza el estudio, se incluye la opción de incluir datos, los cuales pueden ser mostrados en la imagen o no, además la herramienta brinda la opción de guardar estos

datos en un documento Excel o un Bloc de Notas. Para entrar los datos la herramienta cuenta con un cuadro de dialogo como muestra la Fig. 33. Mediante la casilla llamada Mostrar Datos se controla el mostrar u ocultar los datos en la imagen ver Fig. 34. En la Fig. 35 se puede apreciar la opción de guardar los datos en formato Excel y Bloc de notas.

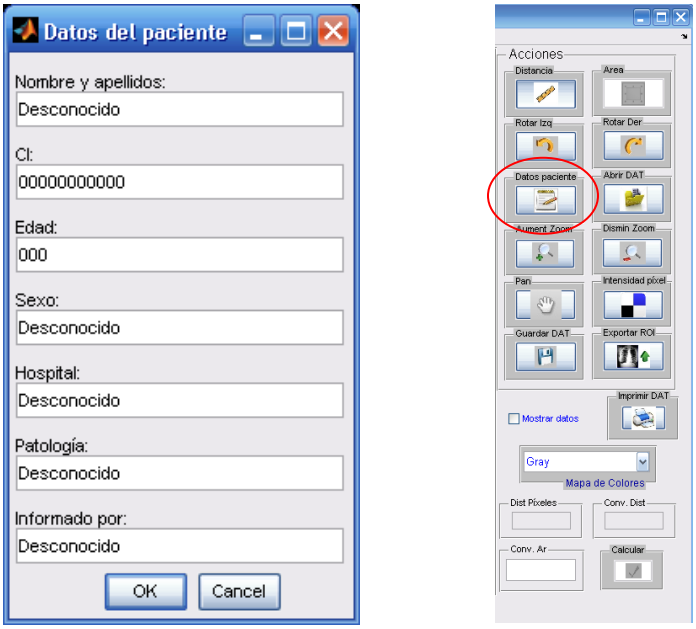


Fig. 33. Ventana para llenar los datos del paciente.

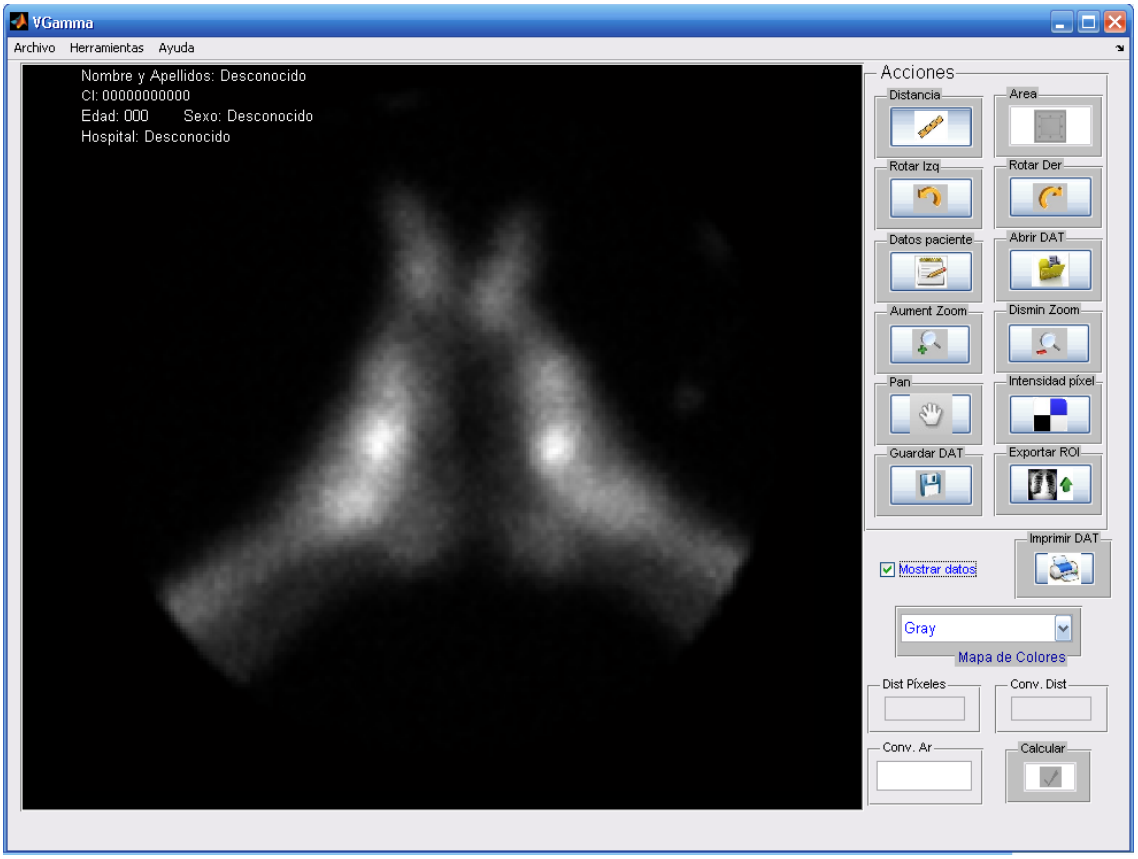


Fig. 34. Imagen con los datos del paciente .

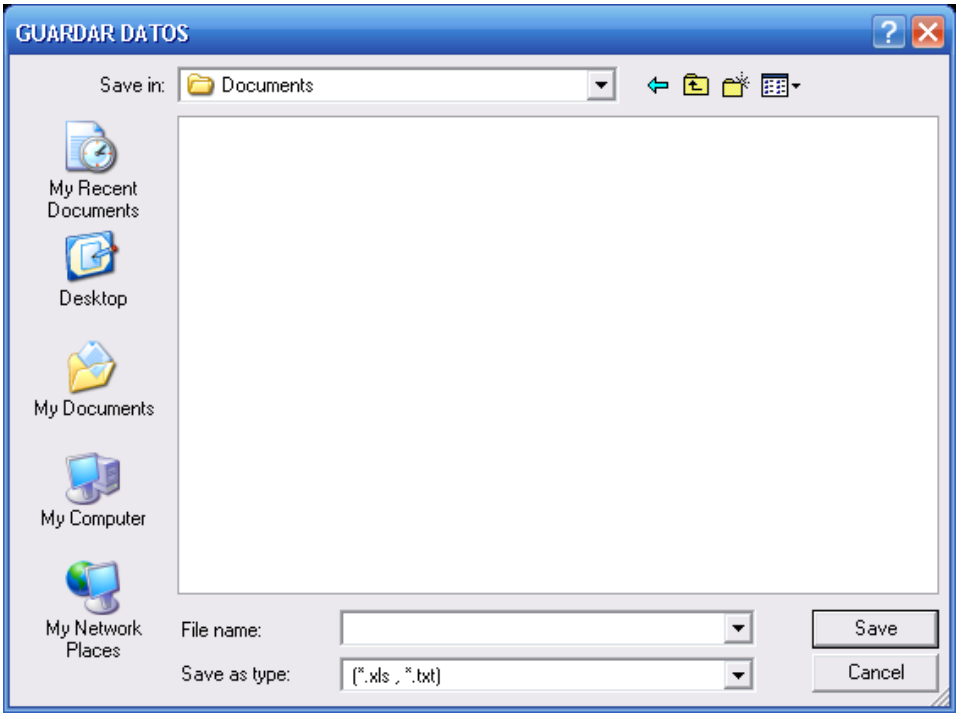


Fig. 35. Explorador para guardar los datos del paciente.

3.3.4 Determinar la intensidad de los píxeles

Al activarse la opción determinar la intensidad de los píxeles en la parte inferior izquierda de la herramienta se mostrará una etiqueta con los valores de coordenadas e intensidad del píxel señalado con el puntero, ver figura a continuación.

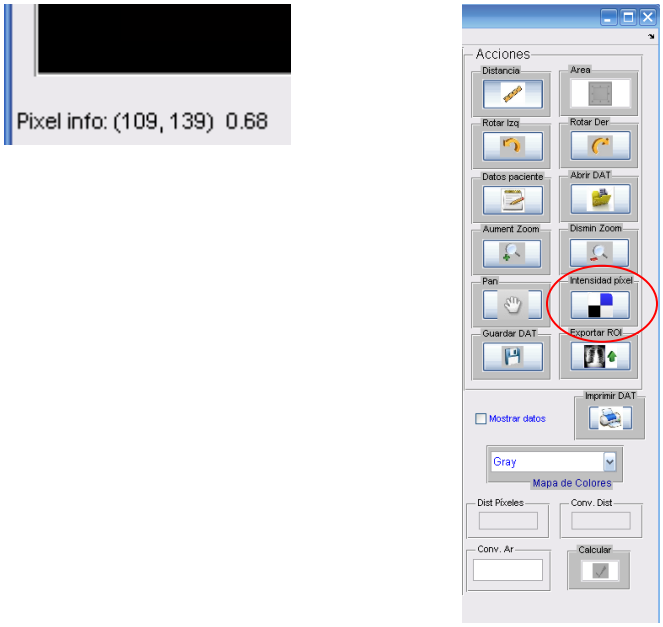


Fig. 36. Información de un píxel señalado en la imagen.

3.3.5 Seleccionar una región de interés

La opción Exportar ROI del panel de Acciones permite seleccionar una región de la imagen mostrada y la muestra en una ventana independiente como se muestra en la Fig. 37, se puede destacar que para la imagen mostrada que corresponde a la región de interés seleccionada se puede utilizar diferente falso color.

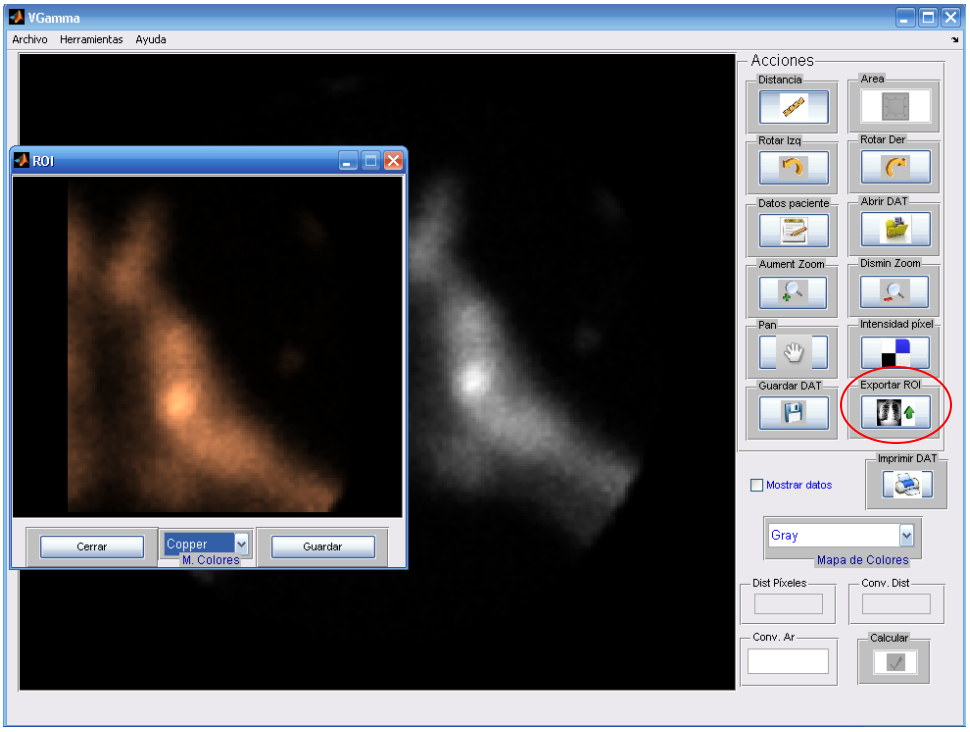


Fig. 37. Selección de una región de interés.

3.3.6 Imprimir distintos tipos de imágenes

La opción de imprimir las imágenes Fig. 38 muestra el cuadro de opciones para la impresión de la imagen que se está visualizando.

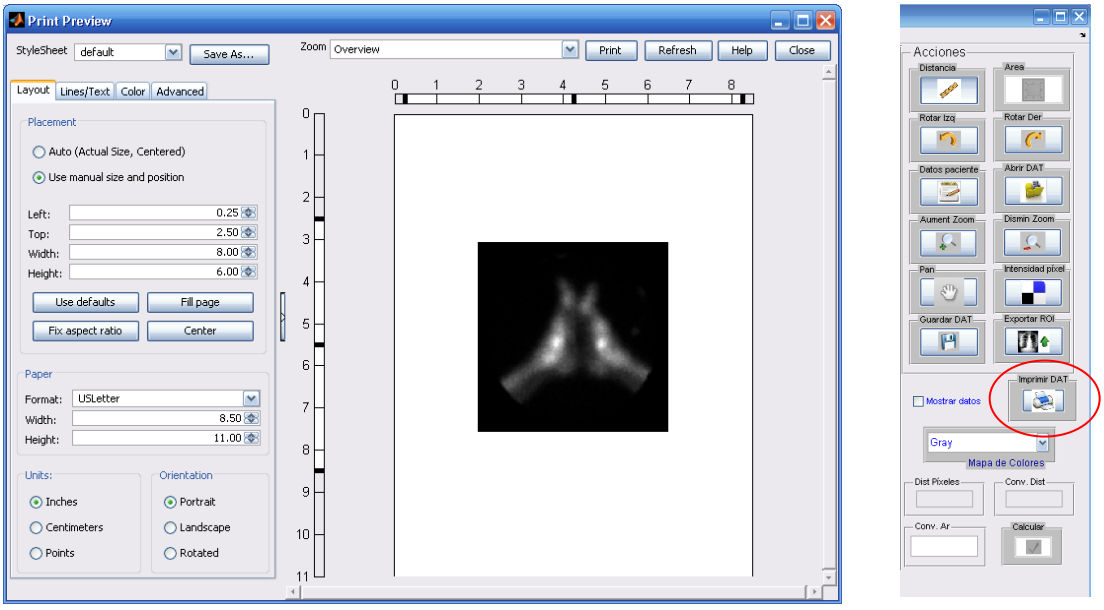


Fig. 38. Ventana de impresión.

3.3.7 Acceso a la ayuda

VGamma cuenta con una ayuda que ofrece información acerca del uso de sus funciones, además de ofrecer información acerca del *Software*.

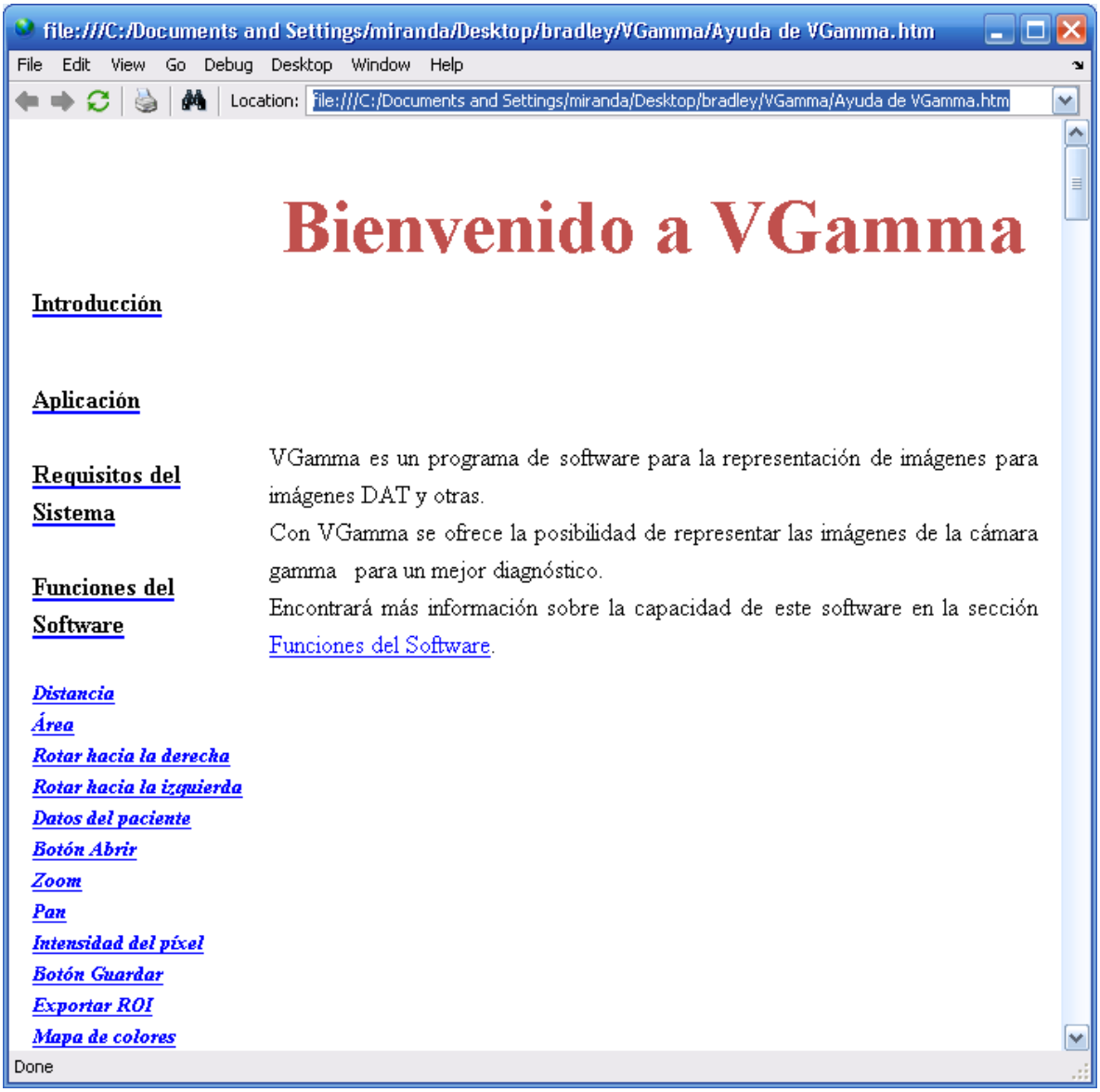


Fig. 39. Ayuda de VGamma.

3.4 Conclusiones parciales

La utilización de las funciones del *toolbox* de procesamiento de imágenes de *Matlab* dio como resultado la creación de la herramienta *VGamma*. Se logra cargar las imágenes de la

cámara Gamma y realizar de forma amigable funciones básicas para el análisis de las imágenes, por parte de especialistas médicos, tales como cargar, guardar, rotar, zoom, mapas de falso color, entre otros.

También se obtiene como resultado la programación de funciones más específicas como medir la distancia entre dos puntos, medir el área de una región, incluir los datos de los pacientes a las imágenes y guardarlos como archivos de *Microsoft* Excel o Bloc de notas y medir la intensidad de los píxeles en las imágenes. Otra de las funciones que se obtuvo con la creación de la herramienta es la selección de regiones de interés en las imágenes y la posibilidad de impresión.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Con el resultado obtenido se aprecia una herramienta capaz de manipular las imágenes provenientes de la cámara Gamma *Sophy Circular* 1000 sobre ordenadores con el sistema operativo *Windows*. *VGamma* tuvo buena aceptación por el personal médico del departamento de medicina nuclear independientemente de que algunas funciones no son por completo iguales a las del software original de la cámara.

La herramienta obtenida brinda la posibilidad de manipular las imágenes, así como realizar mediciones de área y distancia, seleccionar regiones de interés para los usuarios, aplicar mapas de falso color y otras funciones específicas comúnmente utilizadas en las imágenes de medicina nuclear.

Se usa el *toolbox* de Procesamiento de Imágenes que provee el software *Matlab* para desarrollar los algoritmos para las opciones que brinda la herramienta. La herramienta cuenta con una interfaz de usuario que fue desarrollada con el *GUIDE* de *Matlab*, la misma presenta un ambiente amigable e intuitivo para evitar confusiones y mejorar el acceso a las opciones.

La herramienta desarrollada posee una ayuda al usuario donde se explica de forma detallada el funcionamiento de cada botón y el modo usar las funciones de la herramienta, característica que da la posibilidad de tener una mejor interacción entre el *software* y el usuario.

Recomendaciones

1 Implementar *VGamma* en un lenguaje de alto nivel a fin de hacerla independiente del *software Matlab*

2 Aprovechar las bondades del *software Matlab* para incluirle opciones de filtrado y algunas mejoras en general de la imagen.

3 Utilizar las bondades de la herramienta desarrollada para utilizar estas imágenes en la docencia de la carrera de ingeniería biomédica y en investigaciones relacionadas con el Procesamiento Digital de Imágenes.

REFERENCIAS BIBLIOGRÁFICAS

1. Miranda Castañeda I. , "Efectos de la compresión con pérdidas sobre mediciones de distancia en imágenes de TAC", Tesis, 2009
2. Casas Cardoso M. del Carmen, "Reducción de ruido en imágenes planares de Medicina Nuclear con el empleo de la Trasformada Wavelet", Tesis, 2008
3. Colectivo de autores, "Protocolo Nacional para el Control de Calidad de la Instrumentación en Medicina Nuclear", Libro
4. Sopha Médical. "*Console: Drawings and Schematics*", Paris, 1991
5. Sopha Médical. "*Detector and Gantry: Drawings and Schematics*", Paris, 1991
6. Cherry R., S. Sorenson, A. Phelps. "*Physic in Nuclaer Medicine*", Libro, 2003
7. Sopha Médical. "*Sophycamera operator's manual*", Paris, 1991
8. Luna González L., "El diseño de interfaz gráfica de usuario para publicaciones digitales", Artículo, Agosto 2004
9. www.mathworks.com, "*MATLAB The Language of Technical Computing*", Libro, 2004
10. Rafael C. Gonzalez, Richard E. Woods, Steven L. Eddins, "*Digital Image processing using Matlab*", Libro, 2004.
11. Morales Velasco A., Mata Rodríguez S., "Diseño de interfaces gráficas de usuario", Artículo
12. Báez A., Castañeda C., Castañeda D., "Metodología para el diseño y desarrollo de interfaces de usuario", Artículo, 2005
13. Humeres P., "Medicina nuclear: Aplicación en patología osteoarticular", Revista Chilena de Radiología, 2002

14. Colectivo de Autores, "*Quality control of nuclear medicine instruments*", Libro, 1991
15. Fraxedas R., "Estado evolutivo de la Instrumentación en Medicina Nuclear", Libro, Diciembre 2010
16. Sopha Médical. "*Address Assignment and memory map for the Sophycamera*", Paris, 1991
17. Durán L. S. "Ampliación y reestructuración de la Unidad de Imagenología: Unidad de Imagenología para pacientes externos y Clínica de Radiología para la mujer", Tesis, Julio 2007
18. Paz Viera J. E. , "Evaluación de la calidad en imágenes de Resonancia Magnética compactadas con pérdidas", Tesis, 2008
19. Eberl S. , "Programa Asistido de Capacitación a Distancia para Tecnólogos en Medicina Nuclear, Instrumentación - Parte 2 Cámara Gamma", Libro del Curso de la OIEA de Medicina Nuclear, Módulo 3
20. Duch Renom J., Fuster Pelfort D., "Utilidad de la gammagrafía ósea". JANO, medicina y humanidades¨ 2008

ANEXOS

Anexo I Script de *Matlab* para VGamma

```
function varargout = VGamma(varargin)
% VGAMMA M-file for VGamma.fig
%      VGAMMA, by itself, creates a new VGAMMA or raises the existing
%      singleton*.
%
%      H = VGAMMA returns the handle to a new VGAMMA or the handle to
%      the existing singleton*.
%
%      VGAMMA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in VGAMMA.M with the given input
arguments.
%
%      VGAMMA('Property','Value',...) creates a new VGAMMA or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before VGamma_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to VGamma_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help VGamma

% Last Modified by GUIDE v2.5 31-May-2012 14:22:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @VGamma_OpeningFcn, ...
                   'gui_OutputFcn',  @VGamma_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
```

```
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before VGamma is made visible.
function VGamma_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to VGamma (see VARARGIN)

% Choose default command line output for VGamma
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes VGamma wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% Poner un ícono para cada botón
%--------------------------------------------------------------------------
--

[a,map]=imread('Area.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonArea,'CData',g);

[a,map]=imread('rotar Izq.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonRotIzq,'CData',g);

[a,map]=imread('rotar derecha.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonRotDer,'CData',g);
```

```
[a,map]=imread('Texto.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.DatosPaciente,'CData',g);

[a,map]=imread('Aumentar Zoom.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/50);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonAumZoom,'CData',g);

[a,map]=imread('Dism Zoom.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/50);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonDismZoom,'CData',g);

[a,map]=imread('exportar Roig.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/40);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonExportROI,'CData',g);

[a,map]=imread('Abrir.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/40);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonAbrirDAT,'CData',g);

[a,map]=imread('salvar.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonGuardarDAT,'CData',g);

[a,map]=imread('Pan.jpg');
[r,c,d]=size(a);
x=ceil(r/90);
y=ceil(c/90);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonPan,'CData',g);
```

```
[a,map]=imread('distancia.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonDistancia,'CData',g);


[a,map]=imread('checkM.jpg');
[r,c,d]=size(a);
x=ceil(r/20);
y=ceil(c/20);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonCalcular,'CData',g);


[a,map]=imread('contraste.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonIntesPixel,'CData',g);


[a,map]=imread('imprimir.jpg');
[r,c,d]=size(a);
x=ceil(r/30);
y=ceil(c/30);
g=a(1:x:end,1:y:end,:);
g(g==255)=5.5*255;
set(handles.BotonImprimir,'CData',g);
%--------------------------------------------------------------------
handles.cmp = [];
guidata(hObject,handles);

% --- Outputs from this function are returned to the command line.
function varargout = VGamma_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in BotonDistancia.
function BotonDistancia_Callback(hObject, eventdata, handles)
% hObject    handle to BotonDistancia (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of BotonDistancia

% para hallar la distancia en píxeles
%--------------------------------------------------------------------

imdistline

set(handles.EditDistPix, 'Enable', 'On')
set(handles.EditConvDist, 'Enable', 'On')
set(handles.BotonCalcular, 'Enable', 'On')
guidata(hObject,handles);

%--------------------------------------------------------------------

% --- Executes on button press in BotonArea.
function BotonArea_Callback(hObject, eventdata, handles)
% hObject    handle to BotonArea (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para calcular el área
%--------------------------------------------------------------------


Img = getimage (handles.axes1);

BW = roipoly;
% BW = imfreehand;
handles.area = bwarea(BW);
guidata(hObject,handles);


ARM=handles.area;

areaR = get(handles.EditConvArea,'String');
areaR=str2double(areaR);



AreaT = areaR * handles.area;
guidata(hObject,handles);

global TT

TT=AreaT;
Area

%--------------------------------------------------------------------

% --- Executes on button press in BotonRotIzq.
```

```
function BotonRotIzq_Callback(hObject, eventdata, handles)
% hObject    handle to BotonRotIzq (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para rotar las imágenes hacia la izquierda
%--------------------------------------------------------------------

Img= getimage (handles.axes1);
rot = imrotate(Img,90);
imshow(rot,[min(Img(:)) max(Img(:))], 'colormap', handles.cmp)

%--------------------------------------------------------------------
% --- Executes on button press in BotonRotDer.
function BotonRotDer_Callback(hObject, eventdata, handles)
% hObject    handle to BotonRotDer (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para rotar las imágenes hacia la derecha
%--------------------------------------------------------------------
Img= getimage (handles.axes1);
rot = imrotate(Img,-90);
imshow(rot,[min(Img(:)) max(Img(:))], 'colormap', handles.cmp)


%--------------------------------------------------------------------


% --- Executes on button press in DatosPaciente.
function DatosPaciente_Callback(hObject, eventdata, handles)
% hObject    handle to DatosPaciente (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para introducir los datos del paciente
%--------------------------------------------------------------------

set(handles.MostrarDatosCheckB, 'Enable', 'on')

options.Resize='on';
options.WindowStyle='normal';
options.Interpreter='tex';
prompt = {'Nombre y apellidos:','CI:','Edad:', 'Sexo:','Hospital:',
'Patología:', 'Informado por:'};
dlg_title = 'Datos del paciente';
num_lines = 1;
def = {'Desconocido','00000000000', '000', 'Desconocido', 'Desconocido',
'Desconocido', 'Desconocido'};
handles.datos = [];
handles.datos = inputdlg(prompt,dlg_title,num_lines,def, options);
guidata(hObject,handles);

consulta = isempty(handles.datos);
```

```
if consulta == 0
nombre = handles.datos{1,1};
CI = handles.datos{2,1};
edad = handles.datos{3,1};
sexo = handles.datos{4,1};
hosp = handles.datos{5,1};
patologia = handles.datos{6,1};
informado = handles.datos{7,1};


hosp = ['Hospital: ' hosp];
edad = ['Edad: ' edad];
nombre = ['Nombre y Apellidos: ' nombre];
CI = ['CI: ' CI];
sexo = ['Sexo: ' sexo];
informado = ['Informado por: ' informado];
patologia = ['Patología: ' patologia];

handles.t1 = text(5,10,nombre,'FontSize',10, 'color','w');
handles.t2 = text(5,17,CI,'FontSize',9, 'color','w');
handles.t3 = text(5,24,edad,'FontSize',10, 'color','w');
handles.t4 = text(40,24,sexo,'FontSize',10, 'color','w');
handles.t5 = text(5,31,hosp,'FontSize',10, 'color','w');
handles.t6 = text(5,37,patologia, 'FontSize', 10);
handles.t7 = text(5,43,informado, 'FontSize', 10);
guidata(hObject,handles);

set(handles.t1, 'Visible', 'off')
set(handles.t2, 'Visible', 'off')
set(handles.t3, 'Visible', 'off')
set(handles.t4, 'Visible', 'off')
set(handles.t5, 'Visible', 'off')

guidata(hObject,handles);
end
%--------------------------------------------------------------------
% --- Executes on button press in BotonAbrirDAT.
function BotonAbrirDAT_Callback(hObject, eventdata, handles)
% hObject    handle to BotonAbrirDAT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para cargar la imagen en formato DAT
%--------------------------------------------------------------------


[nam,path]=uigetfile('*.DAT','Cargar la Imagen en formato DAT');

file=[path nam];

if nam ~= 0

handles.cmp = [];
```

```
guidata(hObject,handles);

% leer la imagen. viene en formato BIG-ENDIAN
fid=fopen(file,'rb','b');
% el formato es en entero de 16 bits sin signo
data=fread(fid,inf,'uint16');
fclose(fid);

% Determinar el tamaño de la imagen. Por lo que vi, no tiene
% encabezamiento, sino cola
[f,e]=log2(length(data));
data_len = 2^(e-1);
im_values = data(1:data_len);

% convertor a matriz. Supongo resoluciones por la Horizontal y la
vertical
% iguales
M = sqrt(data_len);
N = M;
im_matrix = reshape (im_values, M, N);

%convertirla a niveles de grises. Supongo que la intensidad máxima
depende
%le máximo valor. (no necesariamente tiene que ser así. Más abajo te
%muestro otra opcion

[f,e] = log2(max(im_values));
num_of_gray_levels = 2^e;
im_grayimage = mat2gray(im_matrix, [0 num_of_gray_levels-1]);

%figure; imshow (im_grayimage);
%title([ num2str(num_of_gray_levels, 'Imagen con %d niveles de grises ')
...
   % num2str(e,'(%d bits)')])

%también puedo considerar que el margen dinámico de la escala de grises
es
%tal que el máximo valor representa la máxima intensidad

im_grayimage_highContrast = mat2gray(im_matrix);
max_gray_level = max (im_values);

%figure; imshow (im_grayimage_highContrast);
%title(['Nivel del blanco = ' num2str(max_gray_level)])




axes= imshow(im_grayimage_highContrast);
Img=image(axes,'Parent',handles.axes1);
set(handles.axes1,'Visible','off')% Quitar los ejes de coordenadas


Redim = imresize(im_grayimage,[256 256],'bilinear');
```

```
imagesc (Redim);
set(handles.figure1);
handles.cmp = [];

set(handles.MapaColores, 'Value', 1.0)
end

set(handles.MapaColores, 'Enable', 'on')
set(handles.MostrarDatosCheckB, 'Enable', 'off')
set(handles.BotonArea, 'Enable', 'off')
set(handles.EditDistPix, 'Enable', 'off')
set(handles.EditConvDist, 'Enable', 'off')
set(handles.BotonCalcular, 'Enable', 'off')
set(handles.EditDistPix, 'Value', 0.0)
set(handles.EditConvDist, 'Value', 0.0)
set(handles.EditConvArea, 'Value', 0.0)
set(handles.BotonImprimir, 'Enable', 'on')
guidata(hObject,handles);
%--------------------------------------------------------------------------


% --- Executes on button press in BotonDismZoom.
function BotonDismZoom_Callback(hObject, eventdata, handles)
% hObject    handle to BotonDismZoom (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of BotonDismZoom

% Para disminuir el Zoom
%--------------------------------------------------------------------------

button_state = get(hObject,'Value');
if button_state == get(hObject,'Max')

Img=getimage (handles.axes1);
Z=zoom
set(Z,'Motion','Both','Direction','out','Enable','on');

elseif button_state == get(hObject,'Min')


Img=getimage (handles.axes1);
Z=zoom
set(Z,'Motion','Both','Enable','off');
end

%--------------------------------------------------------------------------

% --- Executes on button press in BotonPan.
function BotonPan_Callback(hObject, eventdata, handles)
% hObject    handle to BotonPan (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hint: get(hObject,'Value') returns toggle state of BotonPan

% Para realizar un PAn a  una imagen aumentada
%--------------------------------------------------------------------------
-

button_state = get(hObject,'Value');
if button_state == get(hObject,'Max')
pan on

elseif button_state == get(hObject,'Min')

Im=getimage (handles.axes1);
p=pan
set(p,'Enable','off');
end

%--------------------------------------------------------------------

% --- Executes on button press in BotonIntesPixel.
function BotonIntesPixel_Callback(hObject, eventdata, handles)
% hObject    handle to BotonIntesPixel (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para ver la intensidad de los píxeles
%--------------------------------------------------------------------------

impixelinfo;

%--------------------------------------------------------------------------

% --- Executes on button press in BotonGuardarDAT.
function BotonGuardarDAT_Callback(hObject, eventdata, handles)
% hObject    handle to BotonGuardarDAT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para guardar los archivos DAT
%--------------------------------------------------------------------

[name,ruta] = uiputfile({'*.jpg';'*.jpeg'},'GUARDAR IMAGEN');
if name ~= 0

if name==0,  end
Img = getimage (handles.axes1);
Img_G = Img.*255;

fName = fullfile(ruta,name);
imwrite(Img_G, handles.cmp, fName);

end
```

```
%--------------------------------------------------------------------

% --- Executes on button press in BotonExportROI.
function BotonExportROI_Callback(hObject, eventdata, handles)
% hObject    handle to BotonExportROI (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para exportar un ROI
%--------------------------------------------------------------------

Img = getimage (handles.axes1);
handles.rect = round(getrect(handles.axes1));
handles.xmin = round(handles.rect(1));
handles.ymin = round(handles.rect(2));
handles.xmax = round(handles.rect(3)) + handles.xmin;
handles.ymax = handles.rect(4) + handles.ymin;
handles.imgrect =
Img(handles.ymin:handles.ymax,handles.xmin:handles.xmax);
guidata(hObject,handles);

global ROIM

ROIM = handles.imgrect;

guidata(hObject,handles);

ROI
%--------------------------------------------------------------------

% --- Executes on selection change in MapaColores.
function MapaColores_Callback(hObject, eventdata, handles)
% hObject    handle to MapaColores (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns MapaColores contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from
MapaColores

% Para generar un mapa de falso color
%--------------------------------------------------------------------
-

guidata(hObject,handles);
val=get(hObject,'Value');
str=get(hObject,'String');

switch str{val}
    case  'Normal'
         C1=getimage(handles.axes1);
         handles.cmp = colormap(gray);
```

```
    case  'Jet'
        C1=getimage(handles.axes1);
        handles.cmp =  colormap(jet);
    case  'Hot'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(hot);
    case  'Cool'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(cool);
    case  'Spring'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(spring);
    case  'Summer'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(summer);
    case  'Autumn'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(autumn);
    case  'Winter'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(winter);
    case  'Gray'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(gray);
    case  'Bone'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(bone);
    case  'Copper'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(copper);
    case  'Pink'
        C1=getimage(handles.axes1);
        handles.cmp = colormap(pink);

 end
guidata(hObject,handles);

%--------------------------------------------------------------------

% --- Executes during object creation, after setting all properties.
function MapaColores_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MapaColores (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function EditDistPix_Callback(hObject, eventdata, handles)
% hObject    handle to EditDistPix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of EditDistPix as text
%        str2double(get(hObject,'String')) returns contents of
EditDistPix as a double

% para poner la distancia en píxeles
%--------------------------------------------------------------------
textS=get(handles.EditDistPix,'String');

global T

T = textS

%--------------------------------------------------------------------

% --- Executes during object creation, after setting all properties.
function EditDistPix_CreateFcn(hObject, eventdata, handles)
% hObject    handle to EditDistPix (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function EditConvDist_Callback(hObject, eventdata, handles)
% hObject    handle to EditConvDist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of EditConvDist as text
%        str2double(get(hObject,'String')) returns contents of
EditConvDist as a double

% para poner la relación de píxeles y mm para la distancia
%--------------------------------------------------------------------

textStr=get(handles.EditConvDist,'String');

global C

C= textStr

%--------------------------------------------------------------------
```

```
% --- Executes during object creation, after setting all properties.
function EditConvDist_CreateFcn(hObject, eventdata, handles)
% hObject    handle to EditConvDist (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function EditConvArea_Callback(hObject, eventdata, handles)
% hObject    handle to EditConvArea (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of EditConvArea as text
%        str2double(get(hObject,'String')) returns contents of
EditConvArea as a double

% para activar el botón de área
%--------------------------------------------------------------------

set(handles.BotonArea, 'Enable', 'On')
guidata(hObject,handles);

%--------------------------------------------------------------------
% --- Executes during object creation, after setting all properties.
function EditConvArea_CreateFcn(hObject, eventdata, handles)
% hObject    handle to EditConvArea (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in BotonCalcular.
function BotonCalcular_Callback(hObject, eventdata, handles)
% hObject    handle to BotonCalcular (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% para calcular la distancia
```

```
%--------------------------------------------------------------------

global T

x = T

x1 = str2num(x);
guidata(hObject,handles);
global C


y = C
y2 = str2num(y);

guidata(hObject,handles);

Result = x1 * y2;

global R

R = Result
guidata(hObject,handles);

Distancia

%--------------------------------------------------------------------

% --- Executes on button press in MostrarDatosCheckB.
function MostrarDatosCheckB_Callback(hObject, eventdata, handles)


% Para mostrar los datos del paciente en la imagen
%--------------------------------------------------------------------

val = get(hObject, 'Value');

if val == 0
    set(handles.t1, 'Visible', 'off')
    set(handles.t2, 'Visible', 'off')
    set(handles.t3, 'Visible', 'off')
    set(handles.t4, 'Visible', 'off')
    set(handles.t5, 'Visible', 'off')

elseif val == 1
    set(handles.t1, 'Visible', 'on')
    set(handles.t2, 'Visible', 'on')
    set(handles.t3, 'Visible', 'on')
    set(handles.t4, 'Visible', 'on')
    set(handles.t5, 'Visible', 'on')
end
```

```
% Hint: get(hObject,'Value') returns toggle state of MostrarDatosCheckB



%--------------------------------------------------------------------

% Códigos para crear los menús

% --------------------------------------------------------------------
function Archivo_Callback(hObject, eventdata, handles)
% hObject    handle to Archivo (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Abrir_Callback(hObject, eventdata, handles)
% hObject    handle to Abrir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function GuardarDAT_Callback(hObject, eventdata, handles)
% hObject    handle to GuardarDAT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para guardar los archivos DAT
%--------------------------------------------------------------------

[name,ruta] = uiputfile({'*.jpg';'*.jpeg'},'GUARDAR IMAGEN');
if name ~= 0

if name==0,  end
Img = getimage (handles.axes1);
Img_G = Img.*255;

fName = fullfile(ruta,name);
imwrite(Img_G, handles.cmp, fName);

end

%--------------------------------------------------------------------

% --------------------------------------------------------------------
function AbrirDAT_Callback(hObject, eventdata, handles)
% hObject    handle to AbrirDAT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% Para abrir los archivos DAT
```

```
%-----------------------------------------------------------------------
[nam,path]=uigetfile('*.DAT','Cargar la Imagen en formato DAT');
if nam ~= 0

file=[path nam];




% leer la imagen. viene en formato BIG-ENDIAN
fid=fopen(file,'rb','b');
% el formato es en entero de 16 bits sin signo
data=fread(fid,inf,'uint16');
fclose(fid);

% Determinar el tamaño de la imagen. Por lo que vi, no tiene
% encabezamiento, sino cola
[f,e]=log2(length(data));
data_len = 2^(e-1);
im_values = data(1:data_len);

% convertor a matriz. Supongo resoluciones por la Horizontal y la
vertical
% iguales
M = sqrt(data_len);
N = M;
im_matrix = reshape (im_values, M, N);

%convertirla a niveles de grises. Supongo que la intensidad máxima
depende
%le máximo valor. (no necesariamente tiene que ser así. Más abajo te
%muestro otra opcion

[f,e] = log2(max(im_values));
num_of_gray_levels = 2^e;
im_grayimage = mat2gray(im_matrix, [0 num_of_gray_levels-1]);

%figure; imshow (im_grayimage);
%title([ num2str(num_of_gray_levels, 'Imagen con %d niveles de grises ')
...
   % num2str(e,'(%d bits)')])

%también puedo considerar que el margen dinámico de la escala de grises
es
%tal que el máximo valor representa la máxima intensidad

im_grayimage_highContrast = mat2gray(im_matrix);
max_gray_level = max (im_values);

%figure; imshow (im_grayimage_highContrast);
%title(['Nivel del blanco = ' num2str(max_gray_level)])
```

```
axes= imshow(im_grayimage_highContrast);
Img=image(axes,'Parent',handles.axes1);
set(handles.axes1,'Visible','off')% Supuestamente debe quitar los ejes de
coordenadas


Redim = imresize(im_grayimage,[256 256],'bilinear');
imagesc (Redim);
set(handles.figure1);
handles.cmp = [];

set(handles.MapaColores, 'Value', 1.0)
set(handles.EditDistPix, 'Value', 0.0)
set(handles.EditConvDist, 'Value', 0.0)
set(handles.EditConvArea, 'Value', 0.0)
end
set(handles.MapaColores, 'Enable', 'on')
set(handles.MostrarDatosCheckB, 'Enable', 'off')
set(handles.BotonArea, 'Enable', 'off')
set(handles.EditDistPix, 'Enable', 'off')
set(handles.EditConvDist, 'Enable', 'off')
set(handles.BotonCalcular, 'Enable', 'off')
set(handles.BotonImprimir, 'Enable', 'on')


guidata(hObject,handles);
%--------------------------------------------------------------------

% --------------------------------------------------------------------
function AbrirJPG_Callback(hObject, eventdata, handles)
% hObject    handle to AbrirJPG (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para abrir JPG
%--------------------------------------------------------------------
[nam,path]=uigetfile({'*.JPG ; *.jpg'},'Cargar la Imagen en formato
JPG');

if nam ~= 0
file=[path nam];

I=imread(file );

imshow(I,[]);
set(handles.MapaColores, 'Value', 1.0)
end
set(handles.MapaColores, 'Enable', 'off')
set(handles.MostrarDatosCheckB, 'Enable', 'off')
set(handles.BotonArea, 'Enable', 'off')
set(handles.EditDistPix, 'Enable', 'off')
set(handles.EditConvDist, 'Enable', 'off')
```

```
set(handles.BotonCalcular, 'Enable', 'off')
set(handles.EditDistPix, 'Value', 0.0)
set(handles.EditConvDist, 'Value', 0.0)
set(handles.EditConvArea, 'Value', 0.0)
set(handles.BotonImprimir, 'Enable', 'off')
guidata(hObject,handles);
%--------------------------------------------------------------------




% --------------------------------------------------------------------
function AbrirDICOM_Callback(hObject, eventdata, handles)
% hObject    handle to AbrirDICOM (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para abrir archivos DICOM
[nam,path]=uigetfile({'*.dcm'},'Cargar la Imagen en formato DICOM');

if nam ~= 0
file=[path nam];

I=dicomread(file);


imshow (I,[]);

end
set(handles.MapaColores, 'Enable', 'on')
set(handles.MostrarDatosCheckB, 'Enable', 'off')
set(handles.BotonArea, 'Enable', 'off')
set(handles.EditDistPix, 'Enable', 'off')
set(handles.EditConvDist, 'Enable', 'off')
set(handles.BotonCalcular, 'Enable', 'off')
set(handles.EditDistPix, 'Value', 0.0)
set(handles.EditConvDist, 'Value', 0.0)
set(handles.EditConvArea, 'Value', 0.0)
set(handles.BotonImprimir, 'Enable', 'off')
guidata(hObject,handles);
%--------------------------------------------------------------------

% --------------------------------------------------------------------
function GuardarEscGrises_Callback(hObject, eventdata, handles)
% hObject    handle to GuardarEscGrises (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para guardar los archivos llevados a escalas de grises y DICOM
%--------------------------------------------------------------------

% Obtener imagen del axes
Img = getimage(handles.axes1);
if isempty(Img), return, end
% Guardar archivo
```

```
[name,ruta] = uiputfile({'*.jpg';'*.jpeg'},'GUARDAR IMAGEN');


if name ~= 0

if name==0,  end
%Img = getimage (handles.axes1);


fName = fullfile(ruta,name);
imwrite(Img, handles.cmp, fName);

end


%--------------------------------------------------------------------

% --------------------------------------------------------------------
function Salir_Callback(hObject, eventdata, handles)
% hObject    handle to Salir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para salir del programa
%--------------------------------------------------------------------
ans=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
if strcmp(ans,'No')
return;
end

close VGamma

%--------------------------------------------------------------------
% --------------------------------------------------------------------
function AjustZoom_Callback(hObject, eventdata, handles)
% hObject    handle to AjustZoom (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Rotar_Callback(hObject, eventdata, handles)
% hObject    handle to Rotar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function rgb_a_gis_Callback(hObject, eventdata, handles)
% hObject    handle to rgb_a_gis (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Para llevar imágenes de rgb a gris
%--------------------------------------------------------------------

ans=questdlg('Se perderá la información de color. ¿Desea
continuar?','VGamma','Si','No','No');
if strcmp(ans,'No')
return;
end

Im=getimage (handles.axes1);
Im_g = rgb2gray(Im);
imshow(Im_g,[]);
set(handles.MapaColores, 'Enable', 'on')
set(handles.MapaColores, 'Value', 1.0)
%--------------------------------------------------------------------
--



% --------------------------------------------------------------------
function Herramientas_Callback(hObject, eventdata, handles)
% hObject    handle to Herramientas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function Ayuda_Callback(hObject, eventdata, handles)
% hObject    handle to Ayuda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function AumZoom_Callback(hObject, eventdata, handles)
% hObject    handle to AumZoom (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para aumentar el zoom
%--------------------------------------------------------------------
Img=getimage (handles.axes1);

Z=zoom
set(Z,'Motion','Both','Direction','in','Enable','on');
%--------------------------------------------------------------------

% --------------------------------------------------------------------
function DismZoom_Callback(hObject, eventdata, handles)
% hObject    handle to DismZoom (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para disminuir el zoom
%--------------------------------------------------------------------
```

```
Img=getimage (handles.axes1);
Z=zoom
set(Z,'Motion','Both','Direction','out','Enable','on');
%--------------------------------------------------------------------

% --------------------------------------------------------------------
function DesctZoom_Callback(hObject, eventdata, handles)
% hObject    handle to DesctZoom (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para desactivar el zoom
%--------------------------------------------------------------------
Img=getimage (handles.axes1);
Z=zoom
set(Z,'Motion','Both','Enable','off');
%--------------------------------------------------------------------

% --------------------------------------------------------------------
function Rotarderecha_Callback(hObject, eventdata, handles)
% hObject    handle to Rotarderecha (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para rotar a la derecha
%--------------------------------------------------------------------
Img= getimage (handles.axes1);
rot = imrotate(Img,-90);
imshow(rot,[min(Img(:)) max(Img(:))], 'colormap', handles.cmp)




%--------------------------------------------------------------------

% --------------------------------------------------------------------
function RotarIzquierda_Callback(hObject, eventdata, handles)
% hObject    handle to RotarIzquierda (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para rotar a la izquierda
%--------------------------------------------------------------------

Img= getimage (handles.axes1);
rot = imrotate(Img,90);
imshow(rot,[min(Img(:)) max(Img(:))], 'colormap', handles.cmp)



%--------------------------------------------------------------------

% --------------------------------------------------------------------
function AyudaProduct_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to AyudaProduct (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para conocer acerca de la ayuda del programa
%--------------------------------------------------------------------
open 'Ayuda de VGamma.htm'
%--------------------------------------------------------------------
% --------------------------------------------------------------------
function AcercadeProduct_Callback(hObject, eventdata, handles)
% hObject    handle to AcercadeProduct (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Para conocer acerca del producto
% --------------------------------------------------------------------
msgbox({'Realizado por Bradley Fernández Cabrera','Universidad Central
Marta Abreu de Las Villas',...
    'Para el Hospital Provincial Celestino Hernández','Versión del
producto: 1.0','2012'},'Acerca de ');
%--------------------------------------------------------------------------
--




% --- Executes on button press in BotonAumZoom.
function BotonAumZoom_Callback(hObject, eventdata, handles)
% hObject    handle to BotonAumZoom (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of BotonAumZoom

%Para aumentar el zoom
%--------------------------------------------------------------------
% Para aumentar el Zoom
%--------------------------------------------------------------------

button_state = get(hObject,'Value');
if button_state == get(hObject,'Max')

Img=getimage (handles.axes1);
Z=zoom
set(Z,'Motion','Both','Direction','in','Enable','on');

elseif button_state == get(hObject,'Min')

Img=getimage (handles.axes1);
Z=zoom
set(Z,'Motion','Both','Enable','off');
end
%--------------------------------------------------------------------------
```

```
% --- Executes on button press in BotonImprimir.
function BotonImprimir_Callback(hObject, eventdata, handles)
% hObject    handle to BotonImprimir (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Para imprimir las imágenes
%--------------------------------------------------------------------------
Img = getImage(handles.axes1);
h = figure('Visible', 'off')
Img = Img.*255;
imshow(Img,[min(Img(:)) max(Img(:))], 'colormap', handles.cmp)

printpreview(h)



%--------------------------------------------------------------------------




% --------------------------------------------------------------------
function GuardarPaciente_Callback(hObject, eventdata, handles)
% hObject    handle to GuardarPaciente (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% para guardar los datos de los pacientes
%--------------------------------------------------------------------------
-

dato.nombre  =  get(handles.t1,'String');
dato.numeroIdentidad = get(handles.t2,'String');
dato.edad = get(handles.t3,'String');
dato.sexo = get(handles.t4,'String');
dato.hosp = get(handles.t5 ,'String');
dato.patologia = get(handles.t6,'String');
dato.informado = get(handles.t7,'String');


guidata(hObject,handles);

[name,ruta] = uiputfile({'*.xls ; *.txt'},'GUARDAR DATOS');
if name ~= 0
if name==0,  end

fName = fullfile(ruta,name);


dlmwrite(fName, dato.nombre,  ...
    'roffset', 1,'coffset' , 1,'delimiter', '', 'newline', 'pc');
```

```
dlmwrite(fName, dato.numeroIdentidad, '-append', ...
  'roffset', 1, 'coffset' , 1,'delimiter', '', 'newline', 'pc');

dlmwrite(fName, dato.edad, '-append', ...
  'roffset', 1, 'coffset' , 1,'delimiter', '', 'newline', 'pc');

dlmwrite(fName, dato.sexo, '-append', ...
  'roffset', 1, 'coffset' , 1,'delimiter', '', 'newline', 'pc');

dlmwrite(fName, dato.hosp, '-append', ...
  'roffset', 1, 'coffset' , 1,'delimiter', '', 'newline', 'pc');

dlmwrite(fName, dato.patologia, '-append', ...
  'roffset', 1, 'coffset' , 1,'delimiter', '', 'newline', 'pc');

dlmwrite(fName, dato.informado, '-append', ...
  'roffset', 1, 'coffset' , 1,'delimiter', '', 'newline', 'pc');
type text
end

% --------------------------------------------------------------------
function Guardar_Callback(hObject, eventdata, handles)
% hObject    handle to Guardar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --------------------------------------------------------------------
function ImprimirGrises_Callback(hObject, eventdata, handles)
% hObject    handle to ImprimirGrises (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%Para imprimir las imágenes llevadas a escala de grises
%--------------------------------------------------------------------
Img = getImage(handles.axes1);
h = figure('Visible', 'off')
% Img = Img.*255;
imshow(Img,[min(Img(:)) max(Img(:))], 'colormap', handles.cmp)

printpreview(h)



%--------------------------------------------------------------------
```

Anexo II Script de Matlab para el Área

```
function varargout = Area(varargin)
% AREA M-file for Area.fig
%      AREA, by itself, creates a new AREA or raises the existing
%      singleton*.
%
%      H = AREA returns the handle to a new AREA or the handle to
%      the existing singleton*.
%
%      AREA('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in AREA.M with the given input arguments.
%
%      AREA('Property','Value',...) creates a new AREA or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Area_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Area_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Area

% Last Modified by GUIDE v2.5 25-Feb-2012 11:32:27

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Area_OpeningFcn, ...
                   'gui_OutputFcn',  @Area_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Area is made visible.
function Area_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Area (see VARARGIN)

% Choose default command line output for Area
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Area wait for user response (see UIRESUME)
% uiwait(handles.figure1);

global TT


valor = TT;




set(handles.text1, 'String', num2str(valor))


% --- Outputs from this function are returned to the command line.
function varargout = Area_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in botonOK.
function botonOK_Callback(hObject, eventdata, handles)
% hObject    handle to botonOK (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close Area
```

Anexo III Script de Matlab para los Datos del paciente

```
function varargout = DatosdelPaciente(varargin)
% DATOSDELPACIENTE M-file for DatosdelPaciente.fig
%      DATOSDELPACIENTE, by itself, creates a new DATOSDELPACIENTE or
raises the existing
%      singleton*.
%
```

```
%      H = DATOSDELPACIENTE returns the handle to a new DATOSDELPACIENTE
or the handle to
%      the existing singleton*.
%
%      DATOSDELPACIENTE('CALLBACK',hObject,eventData,handles,...) calls
the local
%      function named CALLBACK in DATOSDELPACIENTE.M with the given input
arguments.
%
%      DATOSDELPACIENTE('Property','Value',...) creates a new
DATOSDELPACIENTE or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before DatosdelPaciente_OpeningFcn gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to DatosdelPaciente_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help DatosdelPaciente

% Last Modified by GUIDE v2.5 28-May-2012 23:41:52

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @DatosdelPaciente_OpeningFcn, ...
                   'gui_OutputFcn',  @DatosdelPaciente_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before DatosdelPaciente is made visible.
function DatosdelPaciente_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to DatosdelPaciente (see VARARGIN)

% Choose default command line output for DatosdelPaciente
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes DatosdelPaciente wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = DatosdelPaciente_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)

% tomar los datos y pasarlos a la interfaz principal
%_____________________________________________________________________

global nom CI edad sexo hosp

nom = get(handles.edit1, 'String');
CI = get(handles.edit6, 'String');
edad = get(handles.edit4, 'String');
sexo = get(handles.edit5, 'String');
hosp = get(handles.edit7, 'String');
espe = get(handles.edit10, 'String');
pat = get(handles.edit11, 'String');

close DatosdelPaciente

%_____________________________________________________________________


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
function edit7_Callback(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit7 as text
%        str2double(get(hObject,'String')) returns contents of edit7 as a
double


% --- Executes during object creation, after setting all properties.
function edit7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit10_Callback(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit10 as text
%        str2double(get(hObject,'String')) returns contents of edit10 as
a double


% --- Executes during object creation, after setting all properties.
function edit10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit11_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit11 as text
%        str2double(get(hObject,'String')) returns contents of edit11 as
a double


% --- Executes during object creation, after setting all properties.
function edit11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a
double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a
double


% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit5_Callback(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit5 as text
%        str2double(get(hObject,'String')) returns contents of edit5 as a
double


% --- Executes during object creation, after setting all properties.
function edit5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit6_Callback(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of edit6 as text
%        str2double(get(hObject,'String')) returns contents of edit6 as a
double


% --- Executes during object creation, after setting all properties.
function edit6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Anexo IV Script de Matlab para la Distancia

```
function varargout = Distancia(varargin)
% DISTANCIA M-file for Distancia.fig
%      DISTANCIA, by itself, creates a new DISTANCIA or raises the
existing
%      singleton*.
%
%      H = DISTANCIA returns the handle to a new DISTANCIA or the handle
to
%      the existing singleton*.
%
%      DISTANCIA('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in DISTANCIA.M with the given input
arguments.
%
%      DISTANCIA('Property','Value',...) creates a new DISTANCIA or
raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before Distancia_OpeningFunction gets called.
An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to Distancia_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help Distancia
```

```
% Last Modified by GUIDE v2.5 02-Mar-2012 15:24:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Distancia_OpeningFcn, ...
                   'gui_OutputFcn',  @Distancia_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before Distancia is made visible.
function Distancia_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to Distancia (see VARARGIN)

% Choose default command line output for Distancia
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Distancia wait for user response (see UIRESUME)
% uiwait(handles.figure1);
%

global R

Valor = R

set(handles.text4, 'String', num2str(Valor))



% --- Outputs from this function are returned to the command line.
function varargout = Distancia_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in botoncerrar.
function botoncerrar_Callback(hObject, eventdata, handles)
% hObject    handle to botoncerrar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close Distancia
```

Anexo V Script de Matlab para la selección de ROI

```
function varargout = ROI(varargin)
% ROI M-file for ROI.fig
%      ROI, by itself, creates a new ROI or raises the existing
%      singleton*.
%
%      H = ROI returns the handle to a new ROI or the handle to
%      the existing singleton*.
%
%      ROI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in ROI.M with the given input arguments.
%
%      ROI('Property','Value',...) creates a new ROI or raises the
%      existing singleton*.  Starting from the left, property value pairs
are
%      applied to the GUI before ROI_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property
application
%      stop.  All inputs are passed to ROI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only
one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ROI

% Last Modified by GUIDE v2.5 05-Feb-2011 18:53:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ROI_OpeningFcn, ...
                   'gui_OutputFcn',  @ROI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
```

```
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ROI is made visible.
function ROI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ROI (see VARARGIN)

% Choose default command line output for ROI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ROI wait for user response (see UIRESUME)
% uiwait(handles.figure1);
global ROIM

img=ROIM;



axes(handles.axes1)
%imshow(img)
global color
imshow(img,[min(img(:)) max(img(:))])



% --- Outputs from this function are returned to the command line.
function varargout = ROI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in BotonOK.
function BotonOK_Callback(hObject, eventdata, handles)
% hObject    handle to BotonOK (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close ROI

% --- Executes on button press in BotonGuardar.
function BotonGuardar_Callback(hObject, eventdata, handles)
% hObject    handle to BotonGuardar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

rgb = getimage(handles.axes1);
if isempty(rgb), return, end
% Guardar archivo

[name,ruta] = uiputfile({'*.jpg';'*.bmp';'*.jpeg'},'GUARDAR IMAGEN DEL
ROI');
if name ~= 0

if name==0,  end

Img_G = Img.*255;

fName = fullfile(ruta,name);
imwrite(Img_G, handles.cmp, fName);

end

% --- Executes on selection change in MapdeColores.
function MapdeColores_Callback(hObject, eventdata, handles)
% hObject    handle to MapdeColores (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns MapdeColores contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from
MapdeColores
val=get(hObject,'Value');
str=get(hObject,'String');

switch str{val}
    case  'Normal'
         C1=getimage(handles.axes1);
         colormap(gray)
    case  'Jet'
        C1=getimage(handles.axes1);
        colormap(jet)
    case  'Hot'
        C1=getimage(handles.axes1);
        colormap(hot)
    case  'Cool'
        C1=getimage(handles.axes1);
        colormap(cool)
    case  'Spring'
```

```
        C1=getimage(handles.axes1);
        colormap(spring)
    case  'Summer'
        C1=getimage(handles.axes1);
        colormap(summer)
    case  'Autumn'
        C1=getimage(handles.axes1);
        colormap(autumn)
    case  'Winter'
        C1=getimage(handles.axes1);
        colormap(winter)
    case  'Gray'
        C1=getimage(handles.axes1);
        colormap(gray)
    case  'Bone'
        C1=getimage(handles.axes1);
        colormap(bone)
    case  'Copper'
        C1=getimage(handles.axes1);
        colormap(copper)
    case  'Pink'
        C1=getimage(handles.axes1);
        colormap(pink)

 end
guidata(hObject,handles);

% --- Executes during object creation, after setting all properties.
function MapdeColores_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MapdeColores (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```