

Tesis de Maestría

Influencia del efecto del volúmen parcial en la cuantificación de lesiones pequeñas: su efecto sobre el SUV

González Pérez, Guadalupe Itzel

2009

Tesis presentada para obtener el grado de Magister de la
Universidad de Buenos Aires en el área de Física Médica de
la Universidad de Buenos Aires

Este documento forma parte de la colección de tesis doctorales y de maestría de la Biblioteca Central Dr. Luis Federico Leloir, disponible en digital.bl.fcen.uba.ar. Su utilización debe ser acompañada por la cita bibliográfica con reconocimiento de la fuente.

This document is part of the Master's and Doctoral Theses Collection of the Central Library Dr. Luis Federico Leloir, available in digital.bl.fcen.uba.ar. It should be used accompanied by the corresponding citation acknowledging the source.

Cita tipo APA:

González Pérez, Guadalupe Itzel. (2009). Influencia del efecto del volúmen parcial en la cuantificación de lesiones pequeñas: su efecto sobre el SUV. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires.

http://hdl.handle.net/20.500.12110/tesis_n4456_GonzalezPerez

Cita tipo Chicago:

González Pérez, Guadalupe Itzel. "Influencia del efecto del volúmen parcial en la cuantificación de lesiones pequeñas: su efecto sobre el SUV". Tesis de Magister. Facultad de Ciencias Exactas y Naturales. Universidad de Buenos Aires. 2009.

http://hdl.handle.net/20.500.12110/tesis_n4456_GonzalezPerez

EXACTAS UBA

Facultad de Ciencias Exactas y Naturales



UBA

Universidad de Buenos Aires

UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE CIENCIAS EXACTAS Y NATURALES



**"INFLUENCIA DEL EFECTO DEL VOLUMEN PARCIAL EN LA
CUANTIFICACIÓN DE LESIONES PEQUEÑAS: SU EFECTO SOBRE EL SUV"**

Por:

Lic. Guadalupe Itzel González Pérez

Directora de tesis:

Lic. Mariana Levi de Cabrejas

78842

*Tesis presentada para optar al título de
Magíster de la Universidad de Buenos Aires en Física Médica
Abril 2009*

DEDICATORIA

A toda mi familia.

AGRADECIMIENTOS A:

- ❖ Dios, por permitirme cumplir esta meta.
- ❖ El programa de Excelencia Profesional, dirigido por la Secretaria Nacional de Ciencia, Tecnología e Innovación (SENACYT) y al Instituto para la Formación y Aprovechamiento de Recursos Humanos (IFARHU) de Panamá, por la oportunidad y el financiamiento para llevar a cabo los estudios de Maestría en Física Médica.
- ❖ Mariana Levi de Cabrejas, Raúl Cabrejas y Diana Feld por las enseñanzas y el tiempo que me brindaron.
- ❖ Al grupo de trabajo del Centro de Medicina Nuclear del Hospital Universitario Austral.
- ❖ A Samuel, Enrique, Gloria y familia por su aliento para seguir adelante.
- ❖ Mary, Marta y Marga por el tiempo compartido en Argentina.
- ❖ Mis compañeros y amigos de Argentina y Panamá.

A todos muchas gracias

ÍNDICE

	Pág.
1. INTRODUCCIÓN.....	6
2. FUNDAMENTOS TEÓRICOS.....	8
2.1 Tomografía por emisión de positrones (PET).....	8
2.1.1 Principios físicos del PET.....	9
2.1.1.1 Radiofármacos.....	9
2.1.1.2 Radionucleídos.....	9
2.1.1.2.1 Radionucleídos emisores de positrones.....	10
2.1.1.3 Interacción de la radiación con la materia.....	11
2.1.1.3.1 Interacción del los positrones.....	12
2.1.1.3.2 Interacción de los fotones de aniquilación.....	13
2.1.1.3.2.1 Efecto fotoeléctrico.....	14
2.1.1.3.2.2 Efecto Compton.....	15
2.1.1.4 Detección de la radiación.....	16
2.1.2 Adquisición de datos en PET.....	18
2.1.2.1 Tipos de eventos detectados.....	18
2.1.2.2 Modalidad de adquisición de datos.....	20
2.1.3 Formación de la imagen.....	21
2.1.3.1 Métodos de reconstrucción de imagen.....	22
2.1.3.1.1 Retroproyección filtrada.....	22
2.1.3.1.2 Métodos iterativos.....	23
2.1.4 Aplicaciones del PET.....	25
2.1.5 Evolución del PET.....	27
2.2 Valor de captación normalizado (SUV).....	29
2.3 Efecto del volumen parcial (EVP).....	30
2.3.1 Corrección del efecto del volumen parcial.....	33
3. MATERIALES Y MÉTODOS.....	35
3.1 Escáner PET.....	35
3.2 Control de calidad del PET.....	36
3.2.1 Control de calidad diario.....	36
3.2.2 Control de calidad de dos camillas.....	36
3.2.3 Resolución espacial.....	37
3.2.3.1 <i>Fantoma</i> para fuentes puntuales (G).....	37
3.2.3.2 Preparación del <i>fantoma</i> G.....	38
3.2.3.3 Protocolos de adquisición y método de reconstrucción.....	39
3.3 Coeficiente de recuperación.....	40
3.3.1 <i>Fantoma</i> de esferas (E).....	40
3.3.2 Preparación del <i>fantoma</i> E.....	41
3.3.3 Protocolos de adquisición y método de reconstrucción.....	44
3.4 Interfaz Gráfica (ControlPET).....	44
3.4.1 Descripción de ControlPET.....	45
3.4.2 Procesamiento de las imágenes con Control PET.....	46
3.3.2.1 Tamaño de píxel.....	47
3.3.2.1 Resolución espacial.....	46
3.3.2.1 Coeficiente de recuperación.....	46
3.4 Aplicaciones clínicas.....	48

4. RESULTADOS.....	49
4.1 Control de calidad PET.....	49
4.1.1 Control de calidad diario.....	49
4.1.2 Control de calidad de dos camillas.....	49
4.1.3 Resolución espacial y tamaño de píxel.....	50
4.1.3.1 Tamaño de píxel.....	50
4.1.3.2 Resolución espacial.....	51
4.2 Coeficiente de recuperación.....	52
4.3 Aplicaciones clínicas.....	57
5. DISCUSIÓN.....	60
5.1 Control de calidad PET.....	60
5.2 Coeficiente de recuperación.....	61
5.3 Aplicaciones clínicas.....	
6. CONCLUSIÓN.....	62
7. BIBLIOGRAFÍA.....	64
ANEXO. ALGORITMO EN MATLAB.....	66

1. INTRODUCCIÓN

La tomografía por emisión de positrones es una técnica de diagnóstico por imágenes que presenta la ventaja de ser cuantitativa, permitiendo mediciones de las concentraciones de trazador. Sin embargo, al medir la captación de un tumor muy pequeño ocurren desviaciones del valor real, y se dice que éstas se deben al *efecto de volumen parcial* (EVP). Este *efecto de volumen parcial* se observa al evaluar imágenes de lesiones que por su tamaño no pueden ser detectadas correctamente, por ejemplo en tomógrafos por emisión de positrones (PET), en Tomografía Computada por Emisión de Fotón Único, en Tomografía Computada y en Resonancia Magnética.

Este trabajo se centra en determinar la influencia del *efecto del volumen parcial* en las imágenes de lesiones pequeñas, obtenidas mediante equipos PET; en la corrección del efecto mediante el coeficiente de recuperación (CR). El desarrollo del trabajo se realizó en el centro de Medicina Nuclear del Hospital Universitario Austral, con la colaboración del grupo de Física Médica de la Comisión Nacional de Energía Atómica.

El objetivo de este trabajo es la elaboración de una metodología para corregir el EVP en la cuantificación de lesiones, a través de la curva de recuperación de contraste y la posterior corrección del valor de captación normalizado (SUV).

Para lograr el objetivo se construyeron dos *fantomas ad hoc* uno de fuentes puntuales y otro de lesiones calientes. Se diseñó un programa en MATLAB para procesar las imágenes de los *fantomas* adquiridas en el PET, para determinar la resolución espacial del equipo y el número de cuentas del *píxel* de máxima

actividad para las distintas lesiones, datos con los que se obtiene la curva de CR específica para el equipo. Conociendo el diámetro de las lesiones, se determina el CR para las diferentes lesiones y a partir de este dato, el valor verdadero normalizado de la captación de las lesiones.

La corrección EVP es de gran importancia, ya que una mala estimación del valor de captación normalizado de lesiones pequeñas puede conducir a un diagnóstico erróneo del paciente [9].

2. FUNDAMENTOS TEÓRICOS

En este capítulo se presentan los fundamentos teóricos concernientes al desarrollo de la tesis, se divide en tres partes: Tomografía por emisión de positrones, valor de captación normalizado y *efecto del volumen parcial*.

2.1 TOMOGRAFÍA POR EMISIÓN DE POSITRONES (PET)

La tomografía por emisión de positrones es una técnica de radiodiagnóstico que permite medir la concentración de emisores de positrones en un cuerpo tridimensional a través de mediciones externas. Una de las grandes ventajas del PET, radica en utilizar isótopos de elementos de gran importancia biológica, que integran los compuestos orgánicos: ^{15}O , ^{11}C , ^{13}N , o de elementos que los puedan sustituir (^{18}F que reemplaza al H). Este hecho permite el estudio de moléculas básicas (glucosa, ácidos grasos, CO_2 , etc.) con la misma estructura química sin la adición de otros componentes. De esta manera, pueden utilizarse en estudios para determinar el volumen sanguíneo regional, medición del metabolismo regional, etc. [5].

Para conocer el principio y funcionamiento de la Tomografía por Emisión de Positrones es necesario conocer: las características de los trazadores empleados, los aspectos físicos presentes en las interacciones radiación-materia, el proceso de adquisición de los datos y los métodos de reconstrucción de imágenes. Estos aspectos se tratan a continuación.

2.1.1 Principios físicos del PET

El PET se basa en la detección en coincidencia de fotones gamma en direcciones antiparalelas, los cuales son resultado de la aniquilación de un positrón con un electrón. Para conseguir que esta medición proporcione información útil del organismo en estudio, se utilizan radiofármacos de moléculas orgánicas marcadas con radionucleídos emisores de positrones (β^+) de periodo de semi-desintegración corto. Este trazador es inyectado al paciente donde se distribuye. Al realizar una medición de los fotones gamma con suficiente muestreo espacial y mediante el empleo de técnicas de reconstrucción de imágenes se pueden obtener imágenes de la distribución espacial del trazador dentro del organismo.

2.1.1.1 Radiofármaco

El radiofármaco FDG (^{18}F -Desoxi-D-Glucosa), es uno de los más utilizados en PET, es un análogo de la glucosa con un átomo de ^{18}F sustituyendo a un OH y un $T_{1/2}$ de 109,8 minutos. Esta molécula es transportada al interior de la célula, por la hexoquinasa fosforilada, quedando atrapada en su interior ya que no sigue los otros pasos de la ruta metabólica normal. Lo anterior justifica que la concentración de FDG sea usada como indicador de la cantidad de glucosa que utiliza un determinado tejido del organismo.

2.1.1.2 Radionucleidos

Los radionucleídos que se caracterizan por tener un núcleo inestable, tienden a alcanzar el estado de estabilidad emitiendo radiaciones, en un proceso que es denominado decaimiento radiactivo. Estas emisiones pueden ser

evidenciadas a través de equipos de detección específicos para el tipo de radiación que es emitida.

Los radionucleídos siguen la ley de decaimiento radiactivo:

$$A(t) = A_0 e^{-\lambda t} \quad 2-1$$

donde $\lambda = \ln 2 / T_{1/2}$ representa la constante de desintegración, $T_{1/2}$ es el periodo de semi-desintegración y A es la actividad del radionucleído.

La incorporación de radionucleidos al paciente para fines diagnóstico implica considerar un periodo de semi-desintegración físico y un periodo corto de semi- desaparición biológico.

2.1.1.2.1 Radionucleidos emisores de positrones

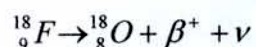
Los radionucleídos emisores de positrones fueron descubiertos en los años 30 del siglo XX, se caracterizan por tener un corto período de semi-desintegración y una actividad específica alta.

El positrón que es la antipartícula del electrón, tiene igual masa en reposo y carga opuesta a la del electrón; es el producto de la transformación nuclear de un núcleo con exceso de protones que busca ser más estable. Esto se logra mediante un reordenamiento en el núcleo y la emisión de un positrón (β^+) y un neutrino (ν):



donde X es el nucleido madre, Y es el nucleido hija, A es el número másico, Z el número atómico. En la ecuación 2.2 se observa que el número másico permanece igual pero el número de protones disminuye una unidad, por lo que representa una transformación isobárica.

Un ejemplo típico de emisor de positrones utilizado en PET es el radionucleido ^{18}F que decae a ^{18}O , como muestra la siguiente transformación:



En la Tabla 2-1 se presenta una lista de emisores de positrones que incluye sus características: energía, periodo de semi-desintegración; y la forma de producción.

Tabla 2-1. Nucleidos emisores de positrones que se usan para realizar imágenes en vivo [6].				
Radionucleido	Periodo de semi-desintegración	Fracción de β^+	Energía máxima (MeV)	Forma de Producción
^{11}C	20,4 min	0,99	0,960	Ciclotrón
^{13}N	9,96 min	1,00	1,19	Ciclotrón
^{15}O	123 s	1,00	1,72	Ciclotrón
^{18}F	110 min	0,97	0,635	Ciclotrón
^{62}Cu	9,74 min	0,98	2,94	Generador (^{62}Zn)
^{64}Cu	12,7 h	0,19	0,580	Ciclotrón
^{68}Ga	68,3 h	0,88	1,9	Ciclotrón (^{68}Ge)
^{76}Br	16,1 h	0,54	3,7	Ciclotrón
^{82}Rb	78 s	0,95	3,35	Ciclotrón (^{82}Sr)
^{124}I	4,18 d	0,22	1,5	Ciclotrón

Los radionucleidos emisores de positrones que se utilizan para adquirir imágenes PET en vivo son costosos, ya que algunos se producen con aceleradores de partículas, como los ciclotrones y otros a través de generadores. Como estos radionucleídos tienen periodo de semi-desintegración muy corto es necesario que el PET este próximo al centro de producción, que en la mayoría de los casos es un ciclotrón.

2.1.1.3 Interacción de la radiación con la materia

Se describe la interacción de los positrones y de los fotones con la materia.

2.1.1.3.1 Interacción del los positrones

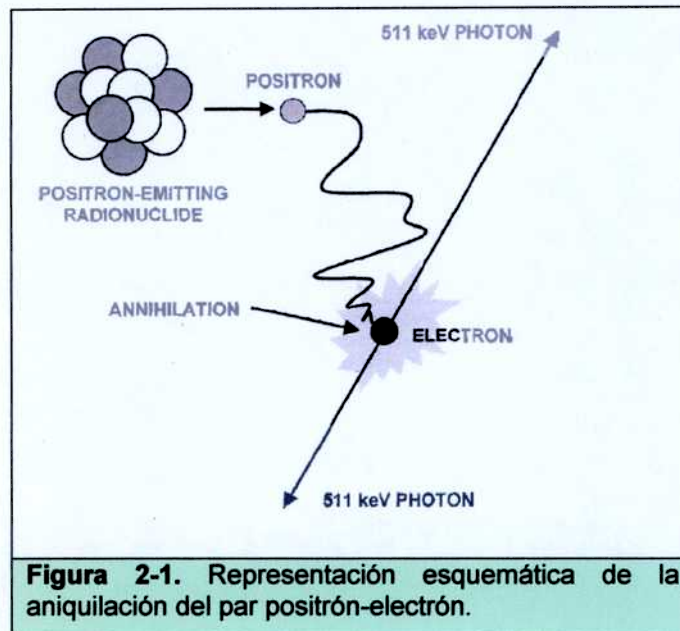
Luego de ser emitido el positrón tendrá una energía inicial, que toma un espectro continuo de valores hasta llegar a un máximo. Este positrón cede esta energía cinética al interactuar con la materia, lo cual puede ocurrir de cuatro formas:

- ❖ Por colisiones inelásticas con electrones atómicos, proceso predominante para la pérdida de energía cinética.
- ❖ Por colisiones elásticas con electrones atómicos.
- ❖ Por dispersión elástica con un núcleo, con deflexión del positrón y frecuentemente con la correspondiente emisión de radiación Bremsstrahlung.
- ❖ Por dispersión elástica con un núcleo.

Cuando un positrón pasa a través de la materia pierde energía cinética por eventos de ionización con otros átomos, o por radiación después de una dispersión inelástica. Ambas situaciones inducirán una deflexión en el paso del positrón, lo que hace pensar que el paso de éste por la materia es tortuoso. Por ello, es difícil estimar el rango de los positrones basándose sólo en la energía. El alcance promedio en agua de algunos positrones usados en PET se presenta en la Tabla 2.2.

Tabla 2-2 Alcance de positrones	
Radionucleído	Rango en agua (mm)
¹⁸ F	2,4
¹¹ C	5,0
¹³ N	5,4
¹⁵ O	8,2
⁶⁸ Ga	9,1

Luego de haber cedido toda su energía cinética el positrón se aniquila con un electrón, el resultado es la emisión de dos fotones con energías de 511 keV, que salen en direcciones opuestas y aproximadamente paralelas, como muestra la Figura 2-1.

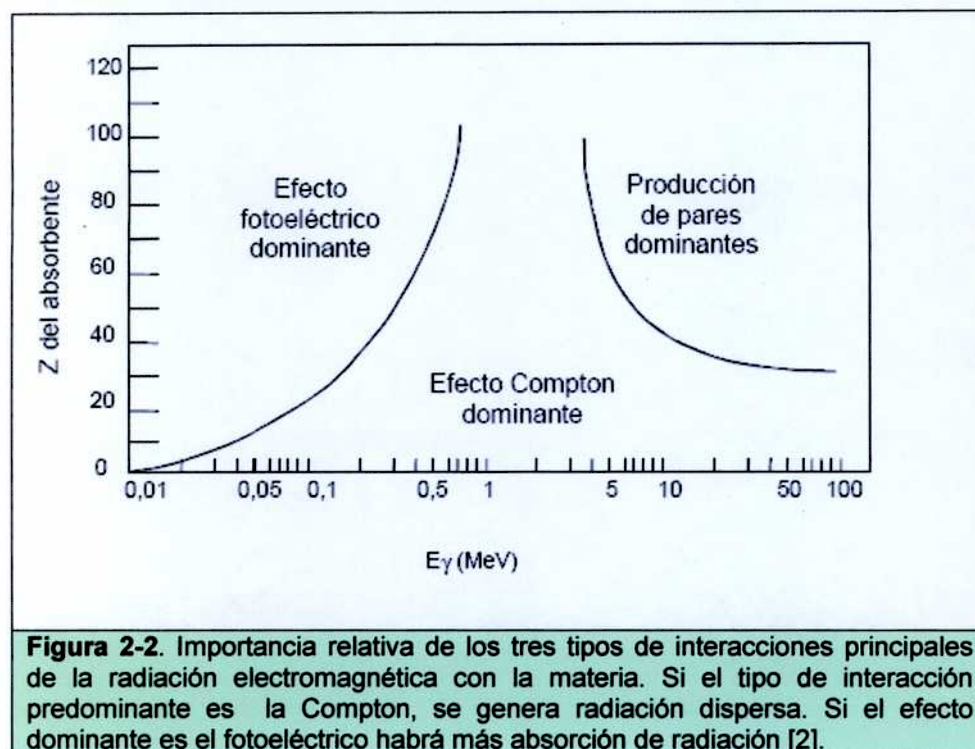


Cuando una partícula y su antipartícula inicialmente en reposo y cercanas entre sí, se unen, aniquilándose, la materia de la cual están formadas desaparece y se libera energía en forma de fotones. Debido a que las partículas están en reposo, el *impulso* inicial del sistema es cero y se debe conservar en el proceso; debe ocurrir entonces la generación de dos fotones que se muevan con impulsos iguales pero en direcciones opuestas. Este proceso es el inverso a la creación de pares y es muy interesante porque verifica la estrecha relación que existe entre los fotones y la materia.

2.1.1.3.2 Interacción de los fotones de aniquilación

Los fotones se caracterizan por no tener ni carga ni masa, tienen tres mecanismos principales de interacción con la materia: efecto fotoeléctrico, efecto

Compton y formación de pares. En la Figura 2-2 se muestra la predominancia de los efectos dependiendo de la energía y el número másico (Z).



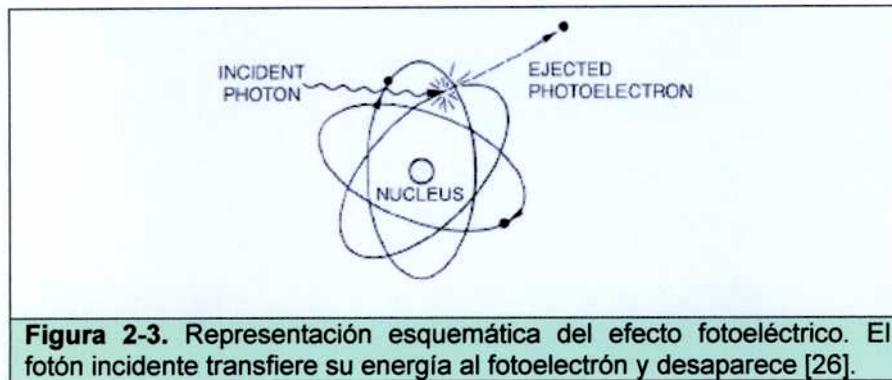
Los fotones provenientes de la aniquilación del par de partículas (positrón y electrón), pueden interactuar con la materia mediante efecto fotoeléctrico o efecto Compton, y no así mediante formación de pares ya que no tienen la energía suficiente para que se dé este mecanismo ($E > 1,022$ MeV). A continuación se describe estos efectos.

2.1.1.3.2.1 Efecto fotoeléctrico

En la interacción fotoeléctrica, la energía total del fotón se transfiere a un electrón del interior de un átomo. El electrón sale expulsado del átomo con energía cinética E_x , donde E_x es igual a la energía $h\nu$ del fotón menos la energía de ligadura E_B necesaria para remover el electrón del átomo:

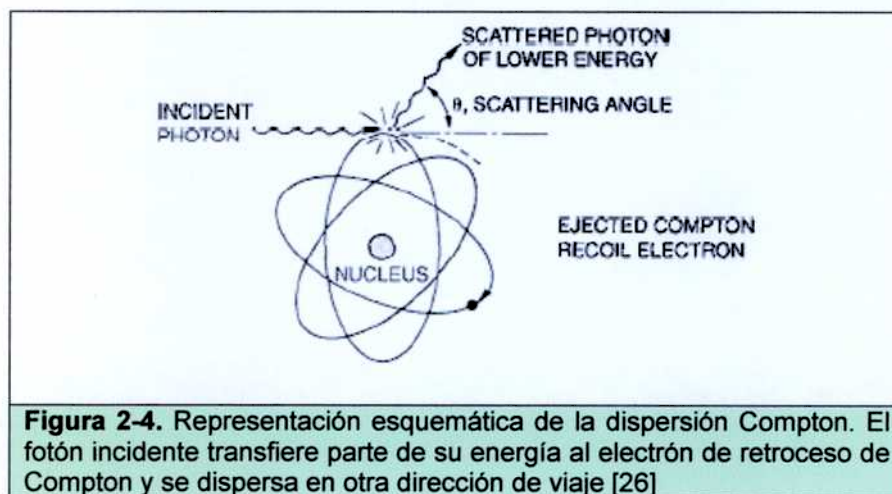
$$E_x = h\nu - E_B \quad 2-3$$

donde el electrón expulsado recibe el nombre de fotoelectrón, Figura 2-3.



2.1.1.3.2.2 Efecto Compton

En este proceso los fotones interactúan con electrones de las capas más externas, en este tipo de interacción parte de la energía del fotón incidente se transfiere a un electrón débilmente unido o libre dentro de un medio atenuante, Figura 2-4. La energía cinética del electrón de retroceso iguala la energía perdida por el fotón, asumiendo que la energía de ligadura del electrón es despreciable.



Durante una interacción Compton, el fotón incidente cede parte de su energía al electrón, la energía del fotón dispersado se expresa en la siguiente ecuación:

$$h\nu' = h\nu \frac{1}{1 + \frac{h\nu}{m_0 c^2} (1 - \cos \theta)} \quad 2-4$$

donde $h\nu$ es la energía del fotón incidente y θ es el ángulo de dispersión del fotón.

Es importante considerar el efecto Compton, ya que cuando un fotón es dispersado y cede parte de su energía, cambia de dirección y se detecta como proveniente de otra posición.

2.1.1.4 Detección de la radiación

Los fotones provenientes de la aniquilación del par electrón-positrón se detectan en coincidencia mediante equipos de detección gamma, Figura 2-5.

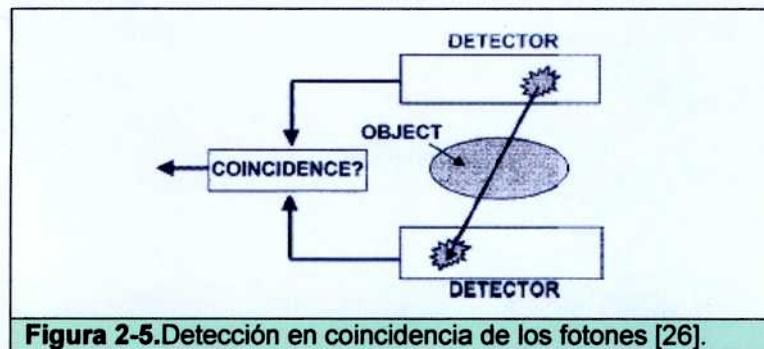


Figura 2-5.Detección en coincidencia de los fotones [26].

El PET aprovecha lo anterior para localizar el decaimiento del emisor de positrones en una línea que une dos detectores llamada línea de respuesta (LOR). Esta técnica de detección en coincidencia se traduce en una colimación electrónica que elimina la necesidad de usar colimación física como ocurre en la cámara gamma o SPECT.

Algunos aspectos propios de la detección de estos fotones limitan la precisión del PET, como son el rango del positrón y el hecho de que no todos los rayos gamma procedentes de la aniquilación son emitidos a 180° , Figura 2-6. Sin embargo, en la práctica la precisión de los PET está limitada principalmente por el tamaño, la geometría, el material y la respuesta temporal de los detectores, y en menor medida por las limitaciones físicas [5].

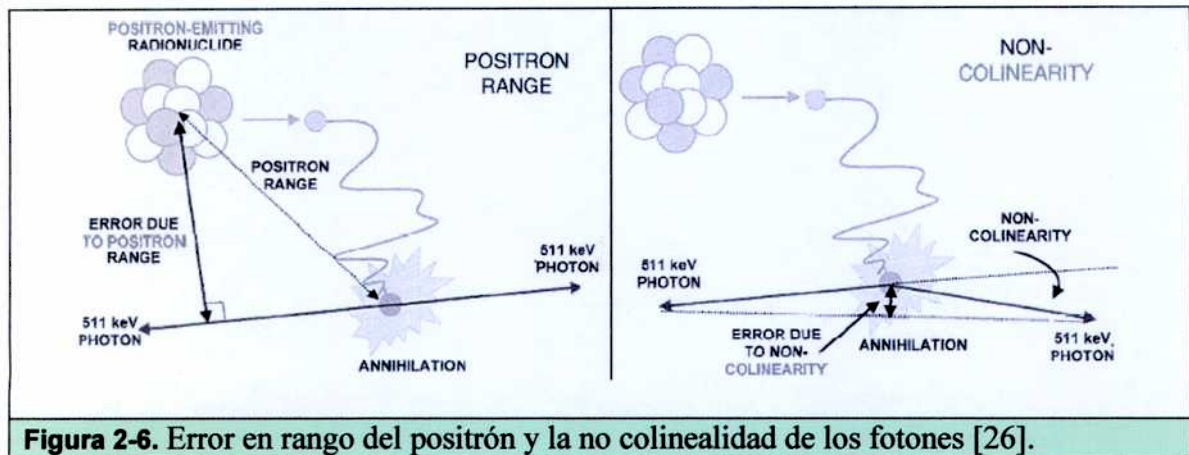


Figura 2-6. Error en rango del positrón y la no colinealidad de los fotones [26].

Un escáner PET se compone de un conjunto de dos o más detectores enfrentados, Figura 2-7.

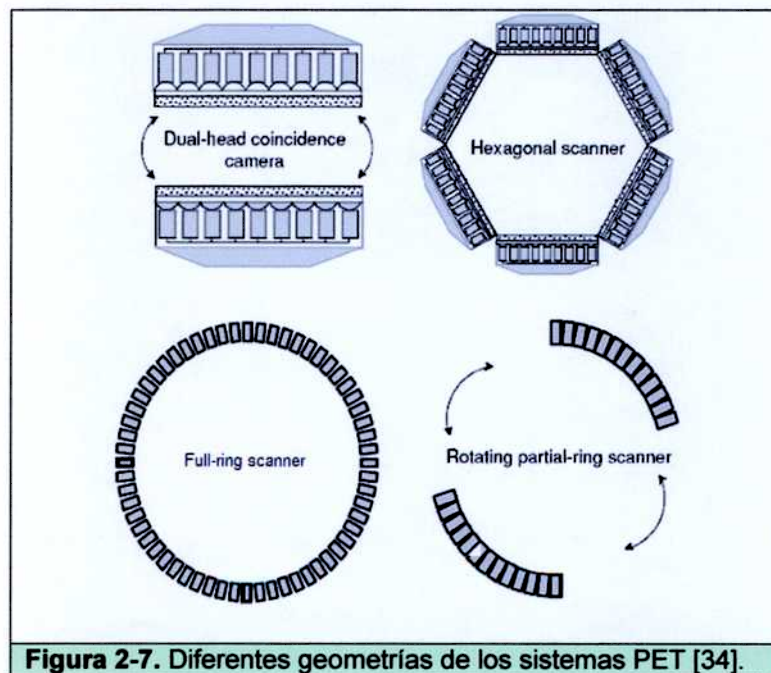


Figura 2-7. Diferentes geometrías de los sistemas PET [34].

En la actualidad muchos de los escáneres PET utilizan detectores de cristales de centelleo, generalmente BGO (germanato de bismuto), acoplados a fotomultiplicadores. Los impulsos recibidos a la salida de los fotomultiplicadores se digitalizan y el software del equipo se diseña de manera que acepte solamente los eventos en coincidencias que se detectaron en detectores opuestos dentro de una ventana de tiempo determinada.

En la Tabla 2-3 se presenta una lista de materiales centelladores con su respectiva eficiencia de detección.

Tabla 2-3. Propiedades físicas de los detectores comunes de centelleo en PET [6].						
Property	NaI (Tl)	BGO	LSO	YSO	GSO	BaF ₂
Effective Z	50	74	66	34	59	52
Density (gm/cm ³)	3.7	7.1	7.4	4.5	6.7	4.9
Scintillation decay time (ns)	230	300	40	70	60	0.6
Photon yield per keV	38	6	29	46	10	2
Relative light output	100	15	75	118	25	5
Linear attenuation coefficient, μ (cm ⁻¹)	0.35	0.96	0.87	0.39	0.70	0.44
Energy resolution (% at 511 keV)	6.6	20	10	12.5	8.5	11.4
BGO: Bismuth Germanate, Bi ₄ Ge ₃ O ₁₂ . LSO: Lutetium oxyorthosilicate doped with cerium (Ce), Lu ₂ SiO ₅ :Ce. YSO: Yttrium oxyorthosilicate doped with Ce, Y ₂ SiO ₅ :Ce. GSO: Gadolinium oxyorthosilicate doped with Ce, Gd ₂ SiO ₅ :Ce. BaF ₂ : Barium fluoride.						

2.1.2 Adquisición de datos en PET

En esta sección se describe el proceso general de adquisición de datos y los tipos de eventos que se detectan, así como la modalidad de adquisición.

2.1.2.1 Tipos de eventos detectados

Un evento se considera como válido si dos fotones:

- ❖ Se detectan en una ventana de tiempo electrónico conocido como la ventana de coincidencia, esta ventana es el periodo de tiempo que utiliza como referencia un circuito electrónico para decidir si dos fotones detectados entran en coincidencia.
- ❖ la línea de respuesta formada entre ellos es válida dentro de un ángulo de aceptación del tomógrafo.

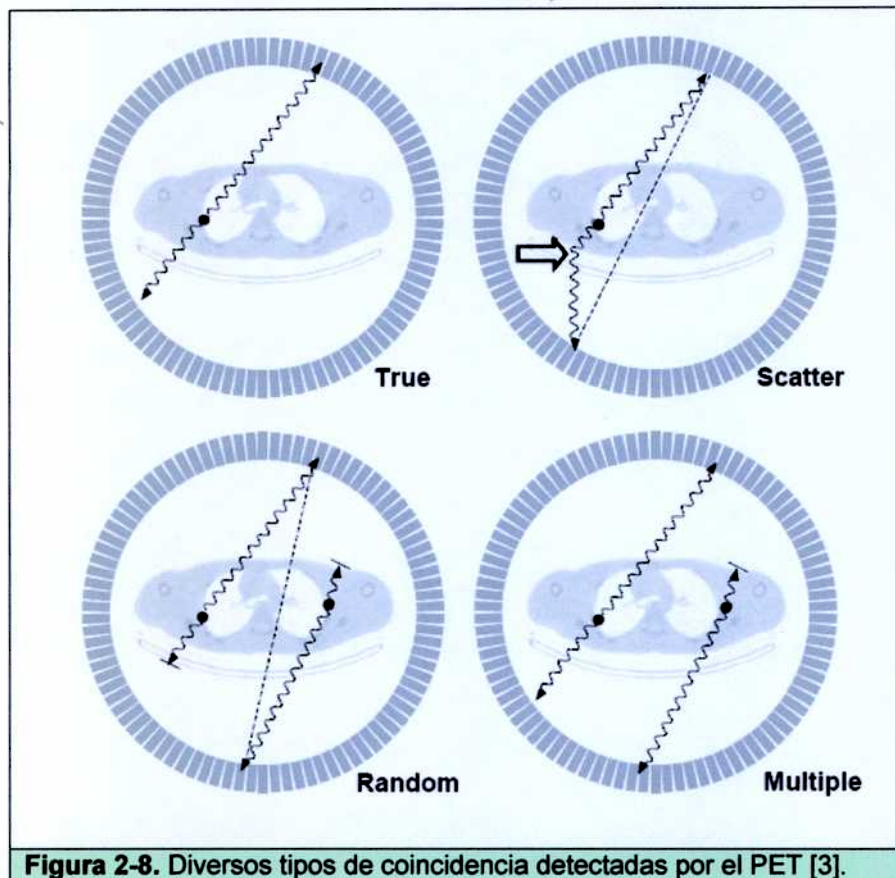
- ❖ la energía depositada en el cristal de ambos detectores está dentro de la ventana de energía seleccionada.

En la Figura 2-8 se muestran distintos tipos de coincidencias detectadas por un PET; se describen a continuación:

- ❖ coincidencias verdaderas: son aquellas en las que se detectan los fotones provenientes de una misma aniquilación sin que estos interactúen con el medio.
- ❖ coincidencias dispersadas: corresponden a un mismo proceso de aniquilación, pero uno o ambos fotones ha sido dispersado por efecto Compton, con lo que el fotón ha perdido parte de su energía y cambiado su dirección.
- ❖ coincidencias aleatorias: corresponden a aniquilaciones aproximadamente simultáneas, donde se emiten cuatro fotones. Dos de estos fotones de diferentes aniquilaciones se cuentan y se consideran provenientes de una misma aniquilación y los otros dos se pierden.
- ❖ coincidencias múltiples: son similares a los eventos aleatorios, excepto que tres eventos de dos aniquilaciones se detectan en la misma ventana de tiempo.

El sistema de adquisición de datos en PET desempeña un papel importante para disminuir estos efectos: los eventos dispersados se reducen con una buena resolución en energía; los eventos aleatorios se reducen con buena resolución temporal; los eventos múltiples se corrigen mediante el empleo de algoritmos estadísticos de admisión de interacciones múltiples y los eventos verdaderos se incrementan con tiempos muertos cortos, con lo que se han de utilizar cristales

con tiempo de respuesta menor, electrónica más rápida y mayor cantidad de pares de detectores en paralelo, etc.

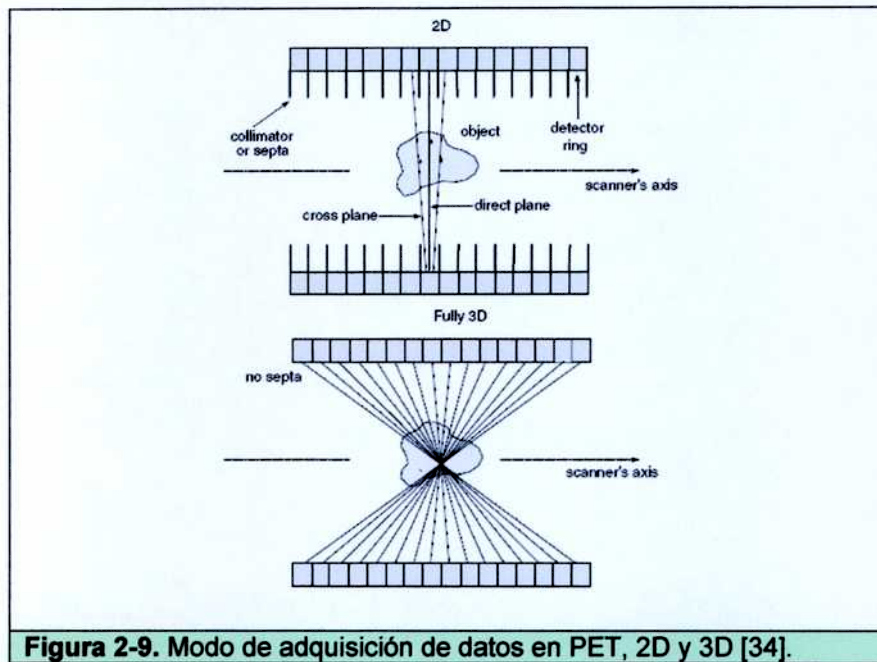


Debido a las coincidencias aleatorias y las dispersadas que se dan en la detección, es necesaria, en algunos casos, una colimación física para disminuir la cantidad de estos eventos que se pueden registrar.

2.1.2.2 Modalidad de adquisición de datos

En la mayoría de los escáneres PET se pueden utilizar modalidad de recopilación de datos en 2D y 3D, sin embargo los equipos modernos operan esencialmente en geometrías 3D. En el modo 2D, se colocan *septs* entre el detector y el paciente para limitar la detección de los fotones dispersos y reducir los eventos aleatorios en coincidencia. El uso de diafragmas también simplifica los algoritmos tomográficos necesarios para reconstruir los datos. Esto último es

importante ya que algunas técnicas requieren tomar en cuenta sólo las líneas de respuesta que están contenidos dentro de uno o varios cortes en una geometría 2D, en lugar de varios cortes como, ocurre al emplear una geometría 3D. La Figura 2-9 ilustra la diferencia conceptual entre la adquisición 2D y 3D [34].



2.1.3 Formación de la imagen

Los datos de proyección adquiridos en el modo 2D o el modo 3D se almacenan en *sinogramas*: filas y columnas que representan muestreos angulares y radiales, respectivamente. Los datos en 3D son algo más complejos que los datos en 2D. La reconstrucción de estas imágenes se logra mediante dos métodos: por retroproyección filtrada y por métodos iterativos [12]. Ambos métodos se describen a continuación.

2.1.3.1 Métodos de reconstrucción de imagen

2.1.3.1.1 Retroproyección filtrada

En la adquisición 2D, la actividad en una línea de respuesta dada de un *sinograma* es la suma de todas las actividades captadas por un par de detectores a lo largo de la línea, a través de la profundidad del objeto. El principio de retroproyección se emplea para reconstruir las imágenes a partir de estas líneas de respuestas adquiridas.

Dicho de otra manera la retroproyección simple, las cuentas a lo largo de una LOR en un *sinograma* captado por un par de detectores se retroproyectan a lo largo de la línea de la cual ellos provinieron. El proceso se repite para todas las LORs, por ejemplo para todos los pares de detectores del PET. Entonces las cuentas de cada LOR, subsecuente se suman a las cuentas de los datos retroproyectados precedentes, obteniéndose así una imagen retroproyectada del objeto original.

La retroproyección simple presenta el problema de artificios en forma de “estrella” (Figura 2-1A), que son producto del muestreo angular, este muestreo hace que la resolución en el centro de la imagen sea mayor que en la periferia, por eso es necesario aplicar un filtro $1/r$ (Figura 2-10C) para filtrar más en el centro de la imagen que en la periferia.

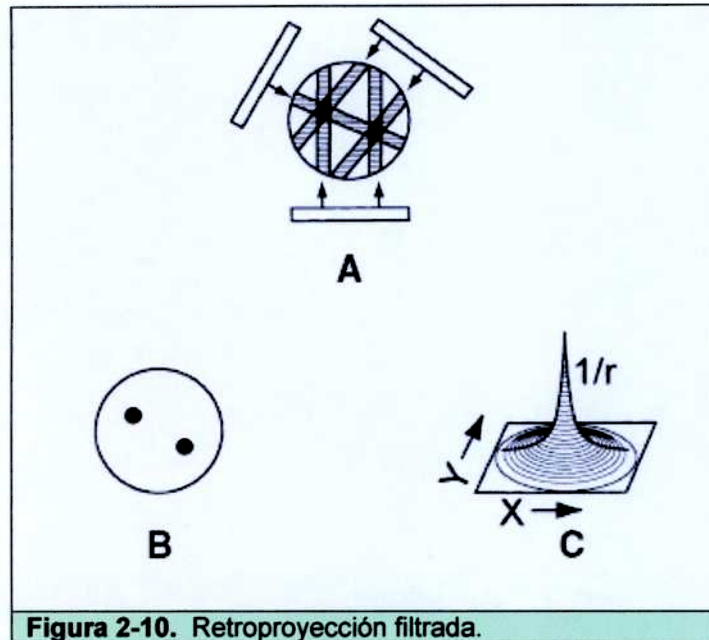
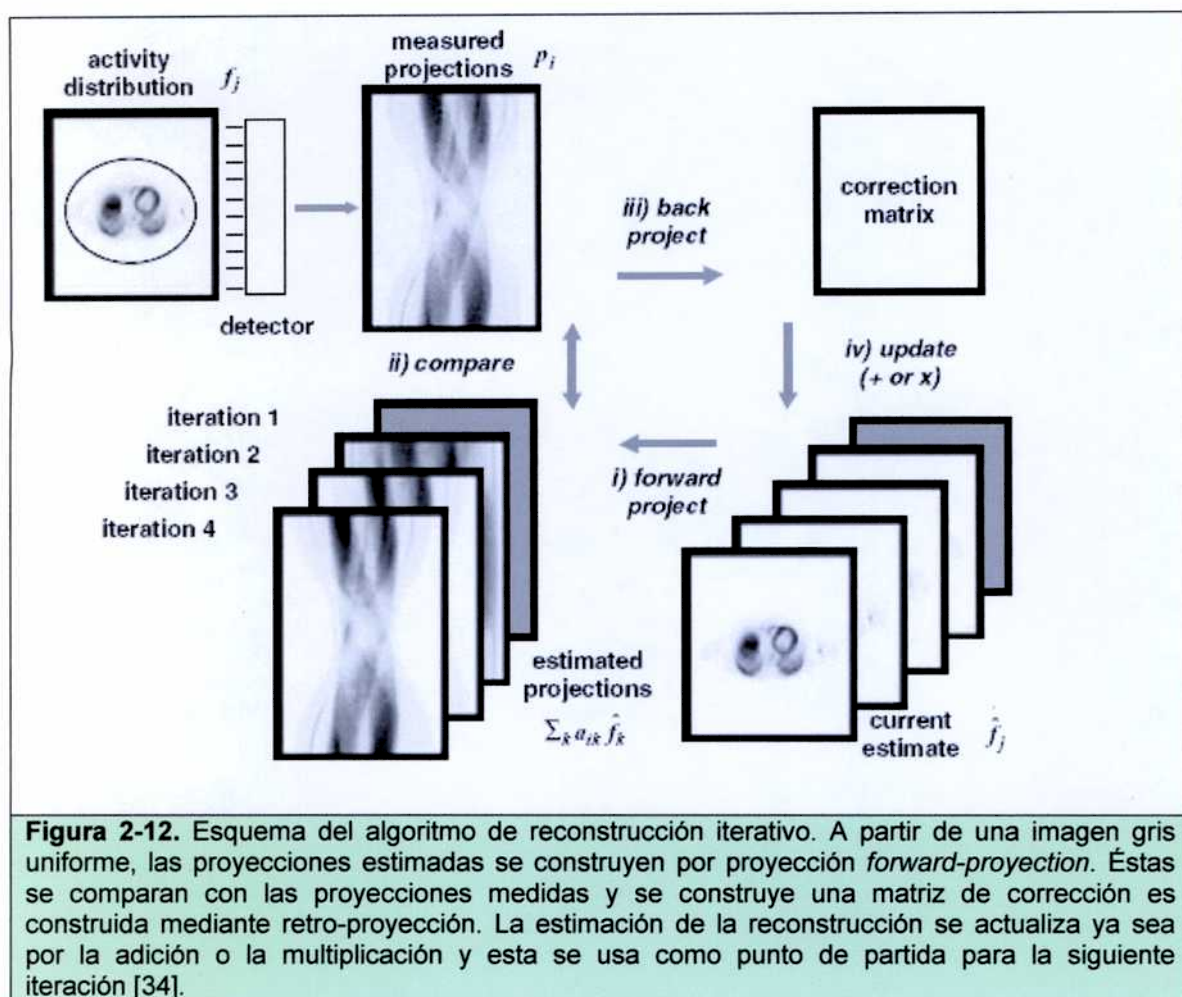


Figura 2-10. Retroproyección filtrada.

2.1.3.1.2 Métodos iterativos

Los métodos de reconstrucción iterativa se basan en la teoría de *Expectation Maximization* (EM), que convergen a una estimación de la imagen con máxima probabilidad [30]. La técnica de reconstrucción iterativa se ilustra en la Figura 2-12. A grandes rasgos el algoritmo iterativo consiste en los siguientes pasos [5]:

- Se parte de una estimación inicial de las imágenes reconstruidas.
- Se proyecta esta imagen estimada para obtener la estimación de datos o sinogramas.
- Se compara la diferencia entre los datos crudos reales y los estimados y se crea una imagen error mediante retroproyección.
- Esta imagen error se utiliza para actualizar la imagen estimada.
- Se repite los pasos b y d hasta cumplirse el criterio de terminación.



Se han desarrollado varios algoritmos para métodos iterativos. Éstos se diferencian principalmente en la manera en que se calculan las proyecciones a partir de la imagen estimada, en el orden en que se aplican las correcciones y en el tipo de corrección de errores que se apliquen a las proyecciones estimadas (corrección de atenuación, corrección de radiación dispersa, etc.).

El más utilizado de los algoritmos iterativos en PET es MLEM y el algoritmo OSEM. La característica principal del algoritmo MLEM es actualizar la imagen en cada iteración. Este método requiere de muchas iteraciones para lograr un acuerdo aceptable entre la imagen estimada y la imagen medida, lo que implica mucho tiempo de cálculo. Para eludir este problema, se emplea el algoritmo OSEM, que es una modificación de MLEM, en cual las proyecciones se agrupan

en subgrupos alrededor del objeto, separados por un de ángulo fijo. Para las proyecciones libres de ruido, se ha demostrado que el OSEM estima basándose sólo en un subconjunto de proyecciones y converge a una solución de máxima verosimilitud, tan cerca como una iteración completa del MLEM, utilizando todas las proyecciones [16]. En otras palabras, si existe n subconjuntos de proyecciones y, a continuación, una vez que todas las proyecciones se utilizan en una sola iteración de OSEM, la estimación resultante de la imagen es similar a la obtenida por n iteraciones de MLEM. Es esta característica del OSEM que acelera el proceso y en general el tiempo de cálculo disminuye con el aumento del número de subgrupos.

2.1.4 Aplicaciones del PET

La utilización en tomografía por emisión de positrones de radiofármacos ,con funciones específicas en el organismo, es una de las ventajas que presenta los PET sobre otros equipos de diagnósticos por imágenes. Además su capacidad de cuantificar es mayor debido a la colimación electrónica, que aumenta cerca de 10 veces la eficiencia de los equipos PET frente a los SPECT, con lo que se pueden coleccionar más cuentas en el mismo tiempo y las imágenes presentan menos ruido estadístico.

En forma general, el PET se puede usar para: diagnóstico, evaluación y estadificación, evaluación de recurrencia, evaluación de respuesta al tratamiento y pronóstico [10].

Algunas de las aplicaciones clínicas que nos brinda esta técnica están detalladas ampliamente en la literatura [1, 10, 11, 15, 23, 28, 32]. En la Tabla 2-1 se presentan un breve resumen.

Tablas 2-4. Aplicaciones clínicas del PET. [36]	
Oncología	Diagnostico del Nódulo pulmonar solitario: Benigno vs. maligno. Cáncer de Pulmón no microcítico Cáncer de Mama Cáncer de Colón Linfoma Cáncer de esófago Melanoma Tumores de cabeza y cuello
Neurología	Diagnostico precoz de la enfermedad de Alzheimer Diagnóstico diferencial de las demencias Evaluación prequirúrgica de las epilepsias Diagnostico diferencial del trastorno del movimiento
Cardiología	Estudio de viabilidad miocárdica previo a revascularización

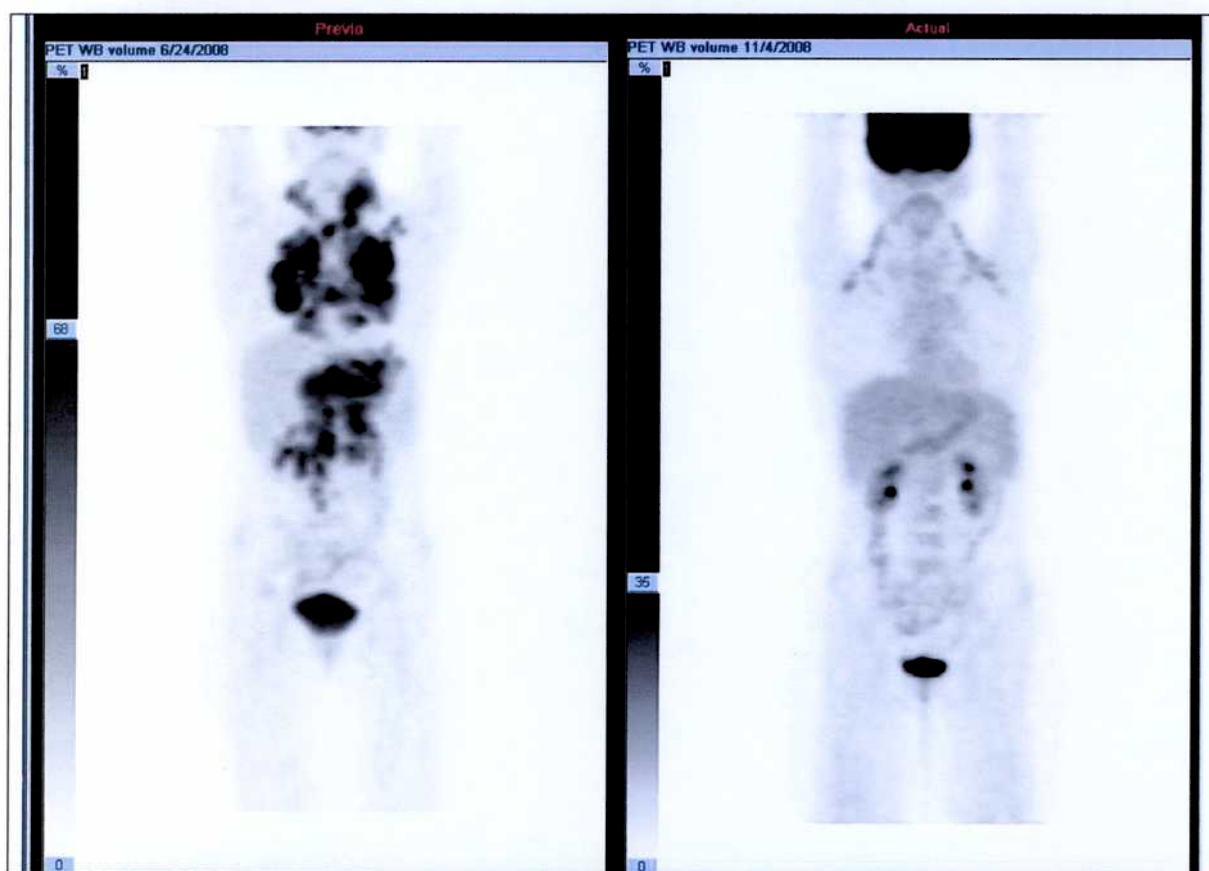


Figura 2-13. Aplicaciones de PET. Paciente con linfoma **Hodgkin** previo y posterior al tratamiento de quimioterapia. Las imágenes posteriores al tratamiento muestran una respuesta positiva a la terapia.

2.1.5 Evolución del PET

En 1950, se hicieron varias sugerencias con el fin de mejorar la calidad de las imágenes nucleares para la detección de tumores cerebrales y otras enfermedades. En particular, se sugirió que el uso de fotones proveniente de la aniquilación de un par positrón - electrón podría mejorar la calidad de las imágenes del cerebro mediante el aumento de la sensibilidad y resolución. A finales de ese año en el Laboratorio de Investigación Física en el Hospital General de Massachusetts se construyó el primer escáner utilizando dos detectores de Ioduro de Sodio activado con Talio (NaI (Tl)) enfrentados, donde la detección se realizaba con los movimientos mecánico en dos dimensiones y un mecanismo de impresión para formar una imagen bidimensional de la fuente de positrones.

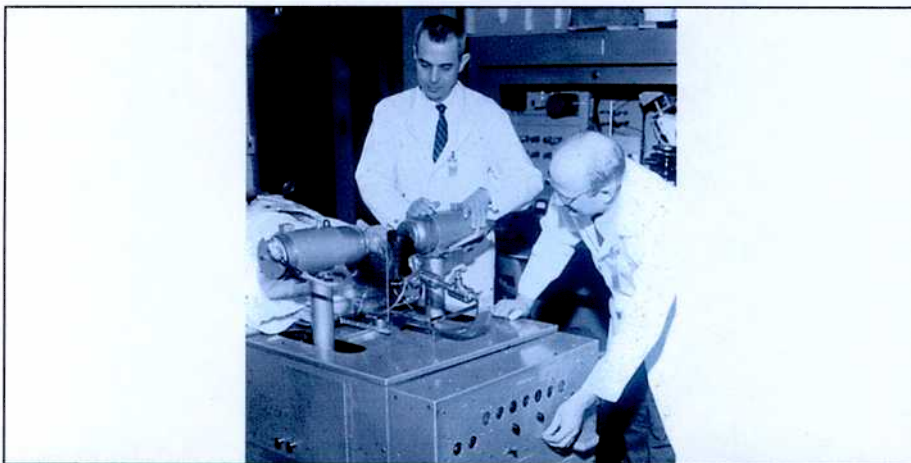
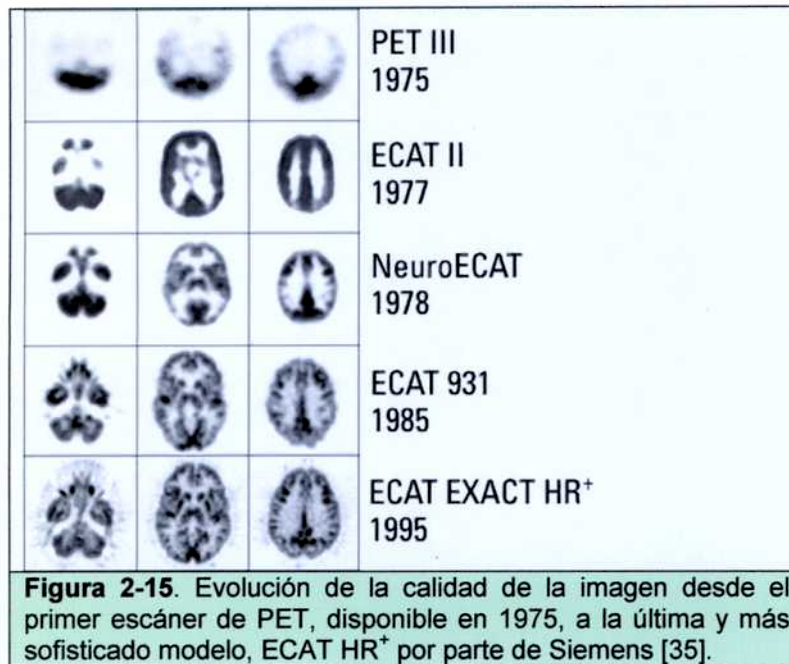


Figura 2-14. Primera clínica de positrones dispositivo de imágenes. Drs. Brownell (izquierda) y Aronow se muestran con el escáner (1953) [38].

En la Figura 2-15 se presenta la evolución de las estructuras internas del cerebro a medida que evolucionaban los PET. La mejora de este equipo se dio con el desarrollo de detectores y de los algoritmos de procesamiento.



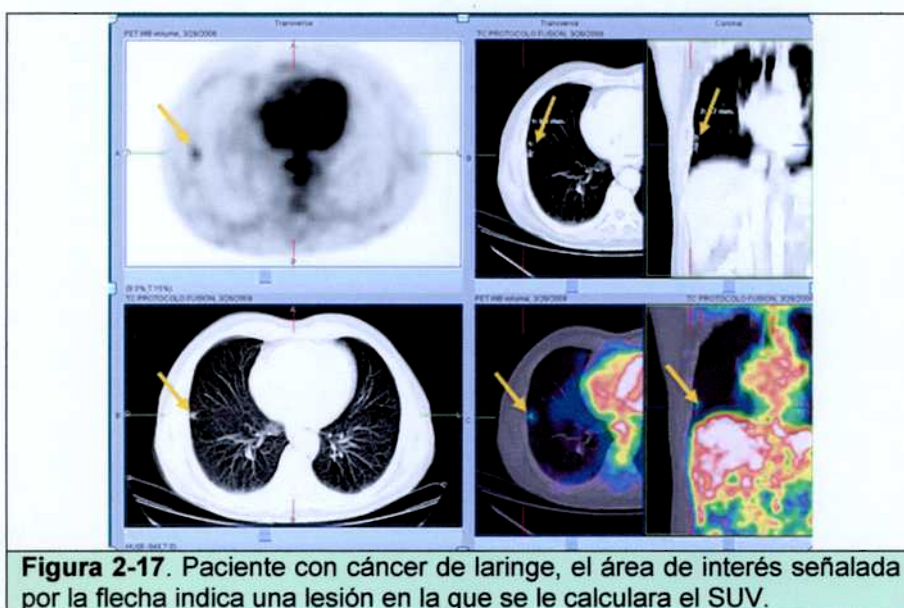
Los modernos PET en la actualidad son sofisticados y de alto costo, mucho más fáciles de instalar y de operar, y tienen nuevas capacidades que los médicos utilizan con ventaja para llevar a cabo muchas hazañas de diagnóstico.



El avance tecnológico que se ha alcanzado en el diagnóstico por imagen ha proporcionado una herramienta importante como es el caso de la fusión de imágenes de PET con imágenes de tomografía computada (CT) y la Resonancia magnética, que dan la descripción anatómica del individuo, Figura 2-16.

2.2 VALOR DE CAPTACIÓN NORMALIZADO (SUV)

Hay dos formas de evaluar la captación de FDG en imágenes de PET [23]. Una es subjetiva-cualitativa (presencia o ausencia de captación, con muchos grados intermedios) y la otra es objetiva-semicuantitativa. Este segundo método se basa en la aplicación del parámetro “valor de captación normalizado” (SUV) que expresa la concentración de FDG en una lesión, normalizada a la actividad administrada y el peso corporal del paciente. El SUV comprende un rango de valores que va desde cero a un valor máximo indefinido. Como guía general, las lesiones con un SUV mayor que 2,5 se asocian con malignidad, y lesiones con un SUV menor que 2,5 están más relacionadas con benignidad. Ahora bien, el SUV es mayor en tejidos metabólicamente activos como el corazón, el cerebro o tumores.



La medición precisa y reproducible del SUV es un factor crítico a considerar para su uso confiable en la evaluación de una terapia. Las variables que se deben considerar al evaluar el SUV incluyen el radiofármacos utilizado, la medición de

captación del radiotrazador en el tejido, los factores biológicos, la cinética del trazador, los parámetros de adquisición y de reconstrucción [15].

En un estudio realizado por [28] se analizó la variabilidad del SUV en función del número de iteraciones empleadas por el algoritmo de reconstrucción de imágenes OSEM, variándose el número de iteraciones. Las diferencias entre los promedios del SUV_{prom} y del SUV_{max} fueron estadísticamente diferentes entre los distintos grupos de iteración.

En el siguiente punto se considera el *efecto del volumen parcial*, como otro factor que influye sobre el valor determinado de SUV.

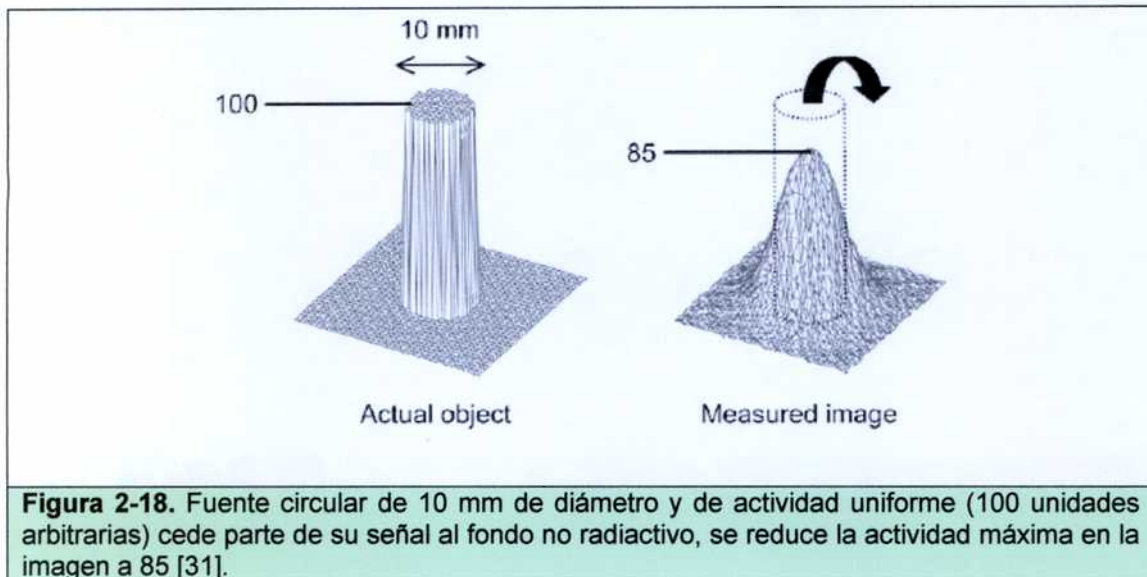
2.3 EFECTO DEL VOLUMEN PARCIAL (EVP)

A pesar de la gran capacidad cuantitativa de los equipos PET, la técnica de tomografía por emisión de positrones presenta una resolución espacial relativamente pobre, lo que influye en la localización espacial de las señales de radiactividad y que genera además el efecto conocido como del *volumen parcial*.

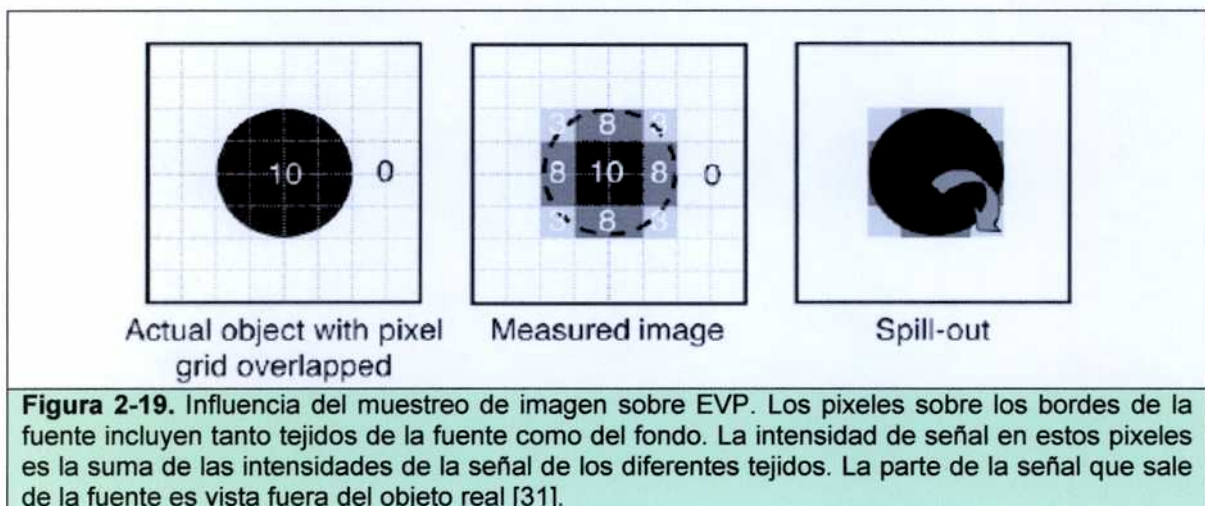
Las causas del *efecto del volumen parcial* son básicamente, el *spillover* (mala ubicación de los eventos por resolución espacial deficiente) de la radioactividad a las regiones aledañas y la inhomogeneidad de los tejidos en la región en estudio.

La “respuesta a una fuente puntual” en una imagen, Point Spread Function (PSF) brinda una idea del efecto de la resolución espacial finita que tiene como consecuencia el *spillover* entre regiones de una imagen de un tomógrafo [14, 20, 24]. Típicamente la PSF se aproxima por una distribución gaussiana en 3D con una FWHM del orden de unos pocos milímetros. Como consecuencia de la resolución espacial finita, la imagen de una fuente puntual es más ancha y

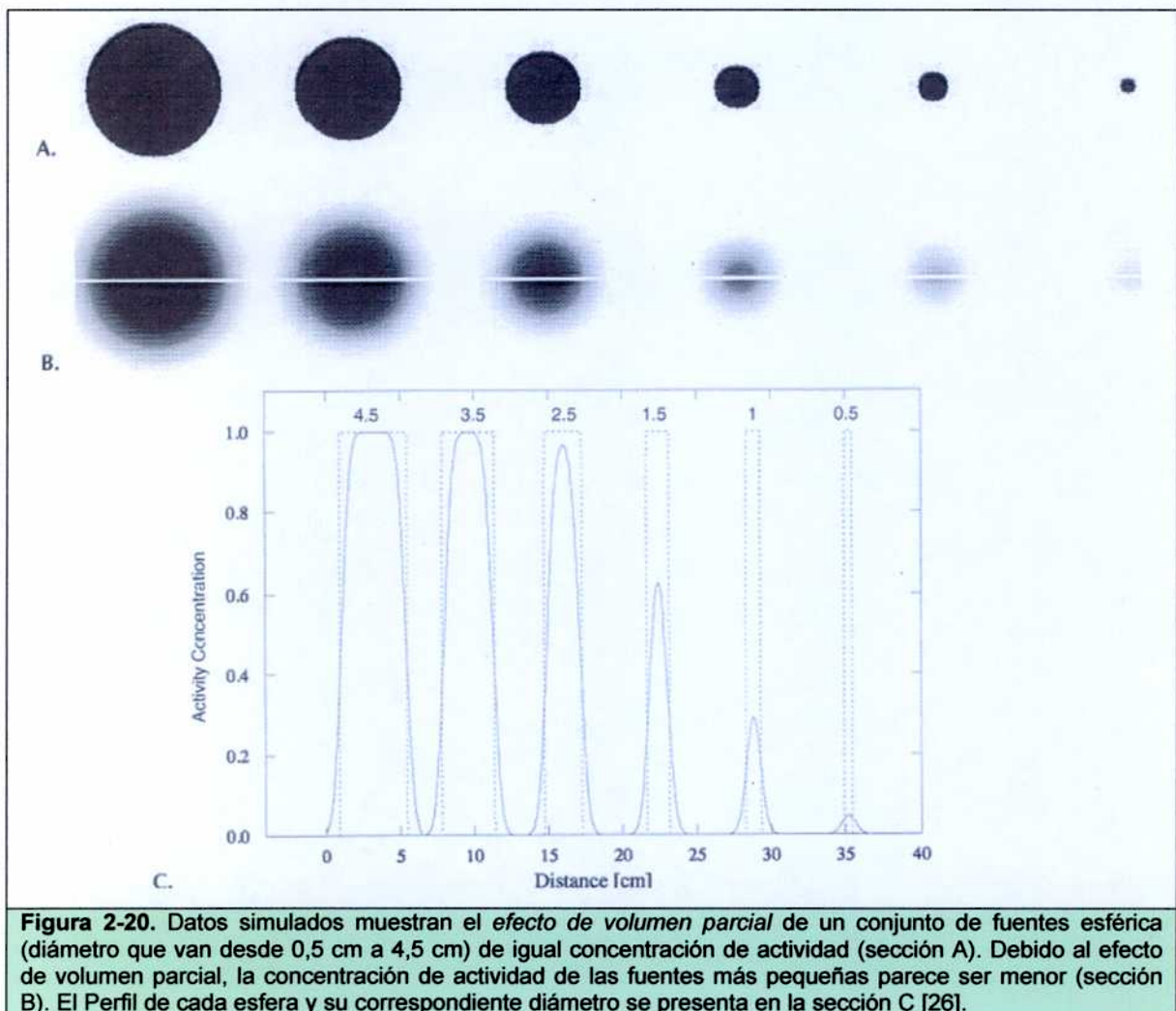
atenuada. Parte de la señal proveniente de la fuente se esparce y se ve fuera de la posición de la fuente real, Figura 2-18.



El otro efecto que causa el *efecto del volumen parcial* es el muestreo de las imágenes. En PET, la distribución del radiotrazador se muestrea en una serie de voxels. El contorno de los voxels suele no coincidir con la distribución real del trazador. Por ello, La mayoría de los voxels incluyen diferentes tipos de tejidos. La Figura 2-19 ilustra este efecto en dos dimensiones. La intensidad de la señal en cada voxel es el promedio de las intensidades de las señales de todos los tejidos que éste contiene [31].



El *efecto del volumen parcial* depende en gran medida del tamaño del tumor. Cuanto más pequeño es el tumor, mayor es la subestimación del valor de la captación. Como consecuencia de esto, se observa que tumores de tamaños diferentes pero igual captación, presentan imágenes con diferentes grados de brillo y por tanto diferentes valores estimados de captación [31]. En la Figura 2-20 se muestra un resultado de una simulación de lesiones de diferentes diámetros pero de igual concentración de actividad.

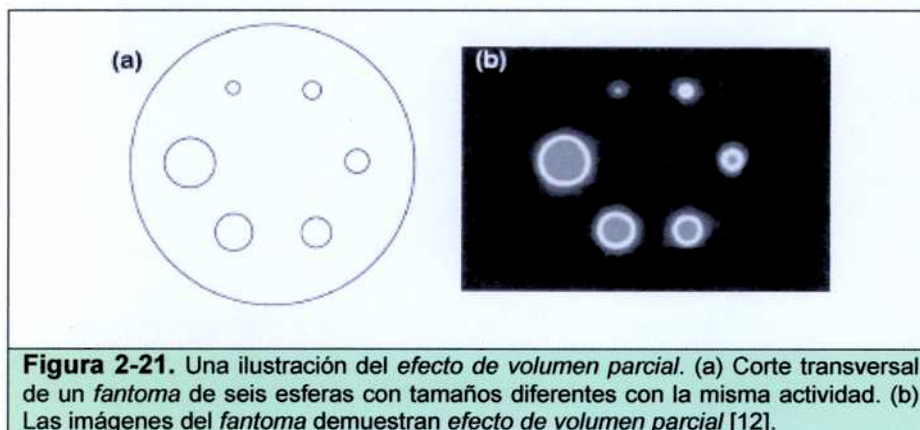


2.3.1 Corrección del efecto del volumen parcial

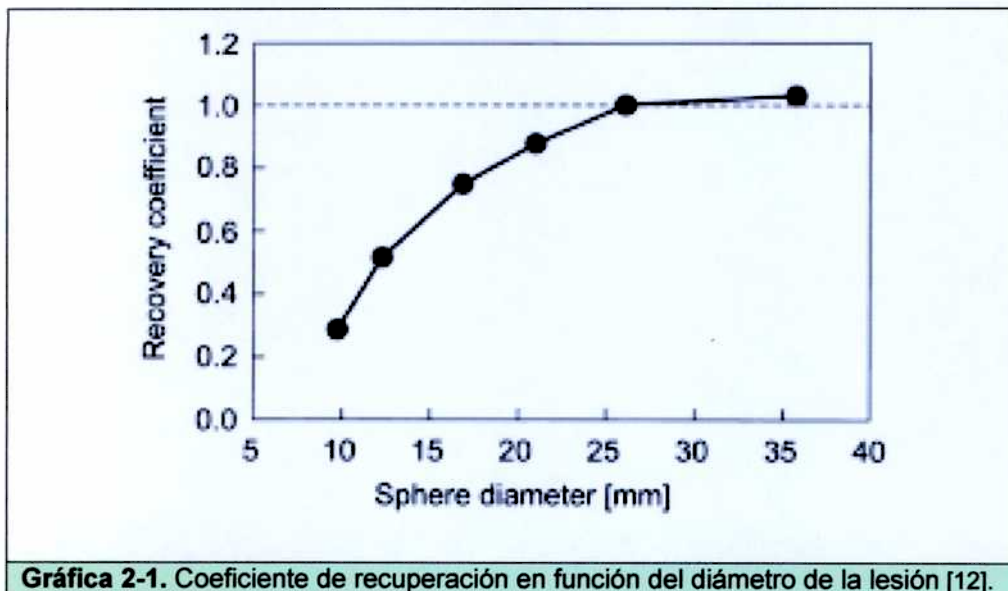
Se han desarrollado múltiples métodos para corregir el *efecto del volumen parcial*. Una clasificación divide en dos grupos, dependiendo de la corrección que se aplica a regiones de interés (ROI) o si esta se realiza a nivel de *voxel*, para mejorar la resolución de toda la imagen. Ahora bien, otra clasificación considera los métodos que usan la respuesta del sistema PET [19].

La primera clasificación incluye los métodos, que toman en consideración, y que por tanto corrigen, la PSF durante la reconstrucción. La segunda categoría incluye los métodos que realizan corrección a las imágenes PET luego de la reconstrucción. La mayoría de estos métodos calculan de una forma u otra el *spillover* de la señal PET de una estructura a otra [27]. Tales métodos usan información anatómica previa para definir los volúmenes con actividad elevada y controlar así la variación en el resto de la imagen [4, 33].

Para corregir el *efecto del volumen parcial* en imágenes reconstruidas con lesiones pequeñas, se puede emplear un factor de corrección llamado coeficiente de recuperación (CR) [12]. Este coeficiente se calcula a partir de simulaciones de lesiones calientes de tamaños variables dentro de un *fantoma*, adquirido con corrección de atención y radiación dispersa [5], Figura 2-21.



El coeficiente de recuperación se determina midiendo la densidad de cuentas de lesiones que contengan la misma concentración de actividad dividida por las cuentas de aquella lesión que no esta afectada por EVP, Gráfica 2-1.



Gráfica 2-1. Coeficiente de recuperación en función del diámetro de la lesión [12].

Normalmente, el coeficiente de recuperación es 1 para las lesiones más grandes, Gráfica 2-1. Los valores de CR así obtenidos se emplean para corregir el EVP en imágenes de lesiones pequeñas.

3. MATERIALES Y MÉTODOS

En esta sección se presenta los controles de calidad del PET y la metodología empleada para evaluar el efecto del volumen parcial en lesiones pequeñas, para esto se construyo un *fantoma* de fuentes puntuales (G) para medir la resolución espacial axial del PET y un *fantoma* de esferas (E) para conocer los coeficientes de recuperación de las lesiones. Se elaboró además un programa en MATLAB (ver Anexo 1) para procesar las imágenes de los *fantomas* y finalmente se presenta como se realiza la corrección del SUV de una serie de aplicaciones clínicas.

3.1 PET

El equipo utilizado es un *PET ECAT ACCEL* fabricado por *Siemens*, Figura 3-1. Este PET adquiere datos ya sea en modo 2D o 3D. Cuando la adquisición de datos es en modo 2D, el PET coloca septas delante del anillo del detector; en el modo de 3D se retraen las septas. Las especificaciones del sistema se presentan en la Tabla 3-1.



Figura 3-1. PET ECAT ACCEL de Siemens.

Tabla 3-1. Especificaciones del PET ECAT ACCEL [13].	
Material del detector	LSO
Dimensiones del cristal (mm ³)	6,45x6,75x25
Diámetro del anillo del detector (cm)	82,4
Campo de visión transversal (cm)	58,3
Campo de visión axial (mm)	162
Ventana de tiempo de coincidencia (ns)	6

Los datos de transmisión de una fuente de ⁶⁸Ge se emplean para realizar la corrección por atenuación. El escáner cuenta con los algoritmos OSEM y retroproyección filtrada para la reconstrucción de las imágenes. Según el protocolo empleado, la reconstrucción se realiza al adquirir los datos o posteriormente.

3.2 CONTROL DE CALIDAD DEL PET

3.2.1 Control de calidad diario

Para conocer si el equipo PET esta en condiciones de trabajo se realiza diariamente el "Daily QC", consiste en evaluar la respuesta de cada block a una fuente homogénea para todos los anillos de detección. El protocolo para este control esta programado para hacerse en forma automática todos los días a las primeras horas.

3.2.2 Control de calidad de dos camillas

Se realiza el siguiente control diario para chequear el sistema PET en su totalidad: comunicación entre *Advanced Computacional System* (ACS) y computador, movimientos de camilla, adquisición y procesamiento de imágenes de transmisión y de emisión, reconstrucción de la imagen y superposición de camillas. Se utiliza el *fantoma* del equipo (un cilindro sin insertos) el que se coloca

de forma tal que abarque dos camillas. El protocolo del equipo ejecutara una adquisición de cuerpo completo y se harán secuencias de emisión y transmisión.

3.2.3 Resolución espacial

La resolución espacial expresa la capacidad que tiene el sistema para discriminar dos fuentes cercanas después de la reconstrucción de la imagen. Una forma de medir este parámetro consiste en: obtener el perfil de una fuente puntual en aire, en diversos puntos del FOV; en determinar el ancho de la Point Spread Función (PSF) de la fuente, en la mitad conocido como *full width at half maximum* (FWHM) y a un décimo conocido por *full width at tenth maximum* (FWTM) del perfil de la fuente [25]. La resolución espacial, estará relacionada con el inverso del valor obtenido para el FWHM es decir, cuanto mayor sea su valor, menor será la resolución del sistema.

Para la determinación del FWHM (FWTM) se procede a contar el número de pixeles que hay entre dos puntos opuestos situados sobre el perfil, a la altura media (décima) y de la transformación a unidades de longitud de los pixeles contados.

3.2.3.1 Fantoma para fuentes puntuales (G)

Para medir el tamaño de píxel y la resolución del PET se diseñó un *fantoma* que contiene fuentes puntuales. Un esquema del diseño se presenta en la Figura 3-2. La base del *fantoma* G está fabricada en acrílico de 5 mm de espesor y contiene orificios realizados con láser para posicionar capilares de 1mm de diámetro.

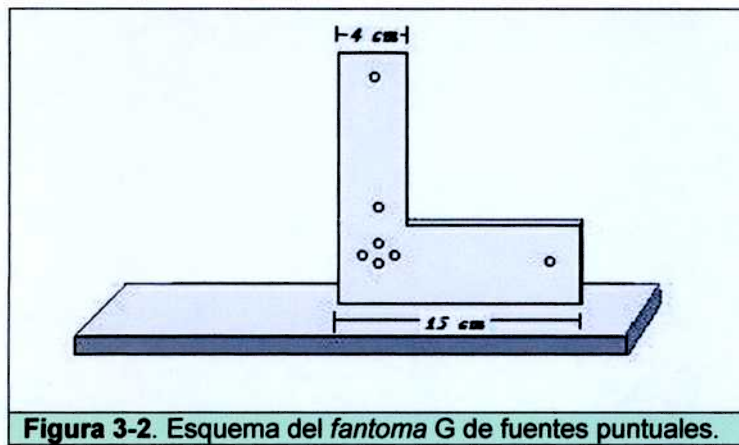


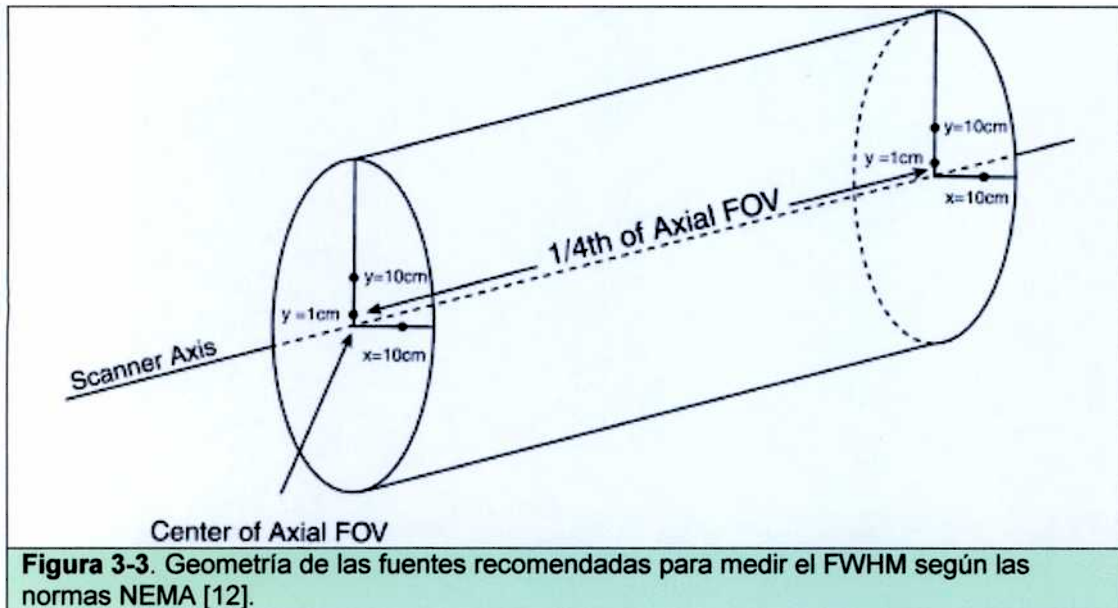
Figura 3-2. Esquema del *fantoma G* de fuentes puntuales.

3.2.3.2 Preparación del *fantoma G*

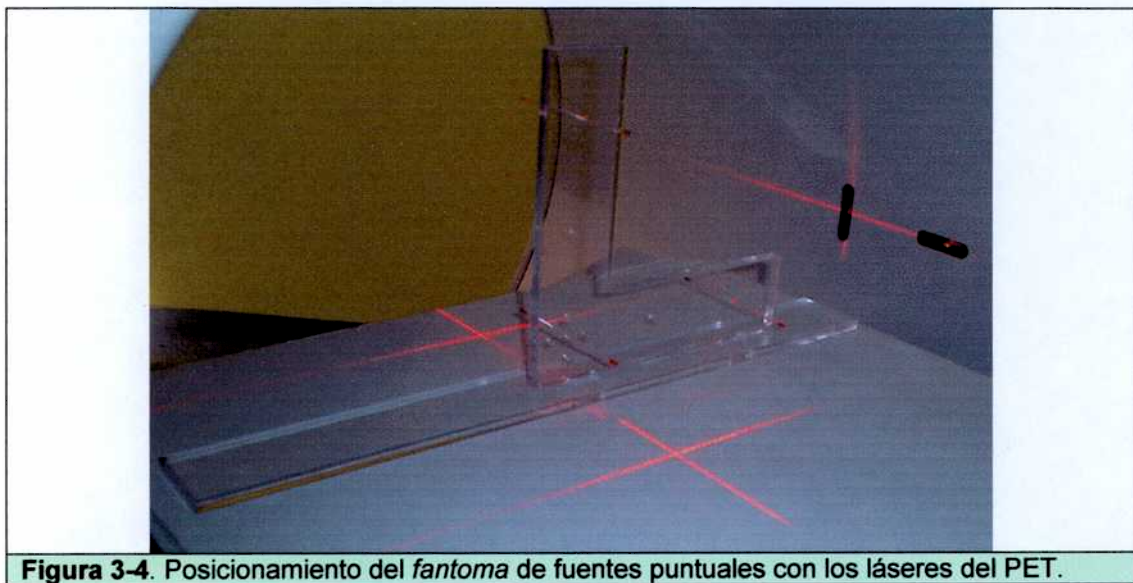
Para medir el tamaño de píxel y la resolución se utilizaron tres capilares cargados con $50 \mu Ci$ cada uno y con los extremos sellados con masilla, como fuente puntual.

Para determinar el tamaño del píxel se emplearon dos capilares. Estos se colocaron en el *fantoma G* (ubicado en centro del FOV) en la posición $x=-1$, $y=0$ y la posición $x=1$, $y=0$.

Para determinar el FWHM y FWTM se emplearon tres capilares. Las posiciones de las fuentes en el *fantoma G* son las recomendadas por National Electrical Manufacturers Association [25] y estas son: $x=0$, $y=1$; $x=10$, $y=0$; $x=0$, $y=10$. El *fantoma* se ubica en dos posiciones diferentes en el centro del FOV y a $\frac{1}{4}$ del FOV. En la Figura 3-3 se presenta esta geometría de fuentes.



El *fantoma* es centrado con los láseres de posición del PET para obtener las ubicaciones requeridas por NEMA (Figura 3-4)



3.2.3.3 Protocolos de adquisición y método de reconstrucción

Los datos se adquieren en las modalidades 2D y 3D, en una matriz de 512x512 píxeles y zoom de 2.5, siendo el tiempo de adquisición de 10 minutos.

Las imágenes se reconstruyen con los parámetros que se detallan en la Tabla 3-2.

Tabla 3-2. Parámetros de reconstrucción	
Método	FBP
Filtro	Rampa
Corrección scatter	Off
Corrección de atenuación	Off

Las imágenes adquiridas del *fantoma* G se transfieren de la consola de adquisición a la estación de trabajo, donde se realizan los cortes axiales, con el programa ECAT 7.2.2. Posteriormente las imágenes se procesan empleando el programa ControlPET.

3.3 COEFICIENTE DE RECUPERACIÓN

La corrección por efecto del volumen parcial permite recalcular el SUV a través de los coeficientes de recuperación. Esta metodología se realiza utilizando el valor del máximo píxel de las ROI en lesiones calientes simuladas con el *fantoma* E, en base al trabajo presentado por [32].

3.3.1 *Fantoma* de esferas (E)

Para evaluar la pérdida de contraste en lesiones pequeñas, y por lo tanto el “efecto del volumen parcial”, se diseñó el *fantoma* E de lesiones calientes (Figura 3-5). Un esquema del diseño se presenta en la Figura 3-6. El cilindro está hecho con acrílico de 5mm de espesor, mide 20 cm de diámetro y 30 cm de altura. Las esferas son de vidrio borosilicato con diámetros diferentes (8, 13.5, 18.4, 23.1, 28.2, 38.4) mm; y están unidas a la parte superior del cilindro mediante capilares, de manera tal que sus centros geométricos forman un plano circular paralelo a la base del cilindro. La longitud de los capilares hace necesario el

empleo de una aguja de punción lumbar para el proceso de suministro y retiro del radiofármacos en las esferas. El cilindro posee su orificio de llenado y vaciado, en la parte superior.



Figura 3-5. *Fantoma E* de esferas calientes.

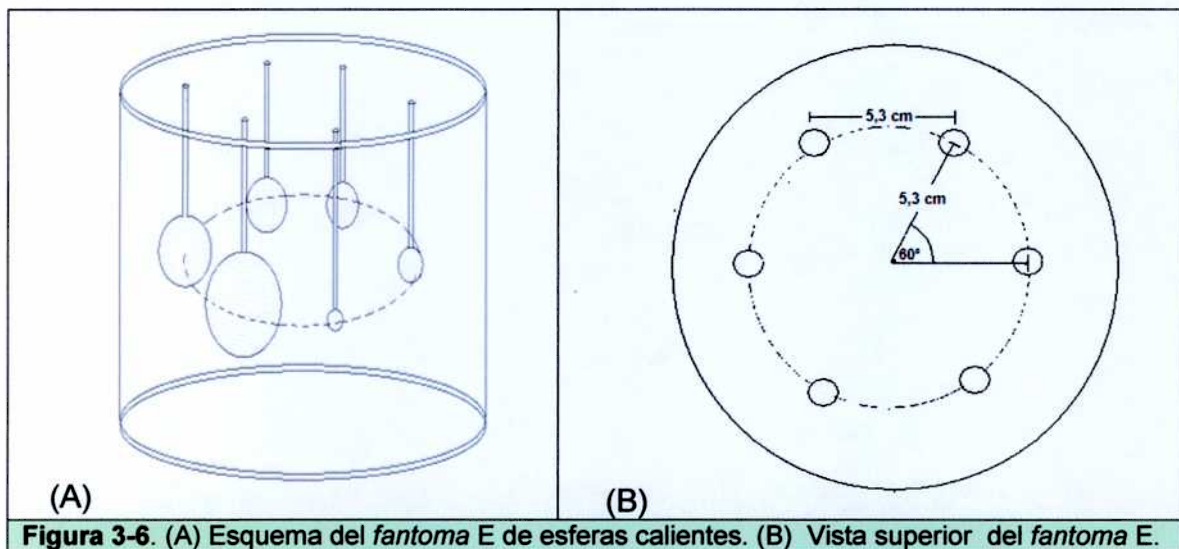


Figura 3-6. (A) Esquema del *fantoma E* de esferas calientes. (B) Vista superior del *fantoma E*.

3.3.2 Preparación del *fantoma E*

Se administran concentraciones conocidas de FDG a las esferas y al cilindro. La razón de concentración de actividad de las esferas con respecto al fondo fueron 3.7:1, 5.8:1 y 10.9:1. Es necesario agitar brevemente el *fantoma E* para homogenizar la distribución de concentración de actividad.

El *fantoma* E se coloca en el PET de tal manera que el centro de las esferas quede ubicado en el centro del FOV, Figura 3-7.



3.3.3 Protocolos de adquisición y método de reconstrucción

Los datos se adquieren en las modalidades 2D y 3D, en una matriz de 128x128 píxeles, siendo el tiempo de adquisición de 30 minutos. Las imágenes se reconstruyen con los parámetros que se detallan en la Tabla 3-3.

Tabla 3-3. Parámetros de reconstrucción	
Método	Iterativo
Iteraciones	4
Subset	8
Zoom	1
Filtro axial	Gaussian
Corrección scatter	On

Las imágenes adquiridas del *fantoma* E se transfieren de la consola de adquisición a la estación de trabajo, donde se realizan los cortes axiales con el programa ECAT 7.2.2. Posteriormente las imágenes del *fantoma* E se procesan empleando el programa ControlPET.

3.4 INTERFAZ GRÁFICA (CONTROLPET)

Se ha realizado una interfaz gráfica de usuario (GUI) en entorno de programación MATLAB para procesar y analizar las imágenes de los *fantomas*,

previamente reconstruidas. La interfaz está diseñada básicamente para: abrir imágenes DICOM, seleccionar y analizar regiones de interés (ROI) y calcular el coeficiente de recuperación del efecto de volumen parcial en función del diámetro de una lesión.

La GUI se desarrolla en MATLAB, entre otras razones, por la disponibilidad del programa en la Facultad de Ciencias Exactas y Naturales de la UBA; por la funciones incorporadas para el tratamiento de imágenes DICOM; porque el programa permite realizar una interfaz gráfica de usuario, lo que facilita integrar en una ventana tanto controles como gráficos e imágenes y porque se puede crear un paquete ejecutable para distribuir a terceros que no posean el programa MATLAB.

Los requerimientos mínimos para usar el ejecutable ControlPet.exe se presentan en la Tabla 3-4 [37]:

Tabla 3-4. Requerimientos mínimos para usar el programa ControlPET.	
PC	Macintosh
❖ Procesador Intel Pentium III o superior ❖ Windows XP, Windows 2000 (con Service Pack 3 o 4) o Windows NT 4.0 (con Service Pack 5 o 6a) ❖ Unidad de CD-ROM o conexión en red para realizar la instalación ❖ 512 MB de RAM	❖ Mac OS X 10.3.2 (Panther) ❖ Unidad de CD-ROM o conexión en red para realizar la instalación ❖ 512 MB de RAM ❖ 64 MB de espacio de intercambio recomendado

3.4.1 Descripción de ControlPET

Las funciones que se han incorporado a ControlPet son:

- ❖ Manipulación de imágenes DICOM: apertura, selección y análisis de ROIs y almacenamiento.
- ❖ Perfiles y cálculo del tamaño de píxel de las imágenes generadas por el PET, cálculo del FWHM y FWTM de una fuente puntual.

- ❖ Determinación del coeficiente de recuperación, mediante regresión por mínimos cuadrados.

3.4.2 Procesamiento de las imágenes con Control PET

Al teclear dos veces sobre el icono del programa ControlPET, se abre el ejecutable. A continuación se presenta la ventana **buscar carpeta**, que requerirá seleccionar la carpeta donde están guardados las imágenes a utilizar, Figura 3-8. Una vez seleccionada esta carpeta, teclear el botón **aceptar**

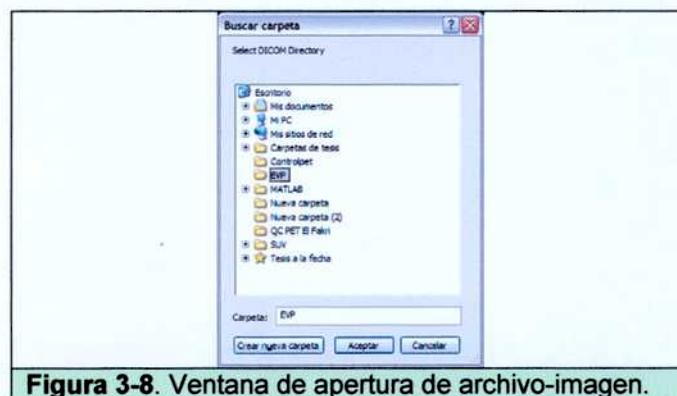


Figura 3-8. Ventana de apertura de archivo-imagen.

Se abrirá la ventana principal de Control PET. En esta, se puede ver la imagen seleccionada, así como los datos de la imagen. También se pueden observar grupos de controles con funciones que se especifican por sus nombres: tamaño de píxel, resolución y coeficiente de recuperación, como muestra la Figura 3-9.

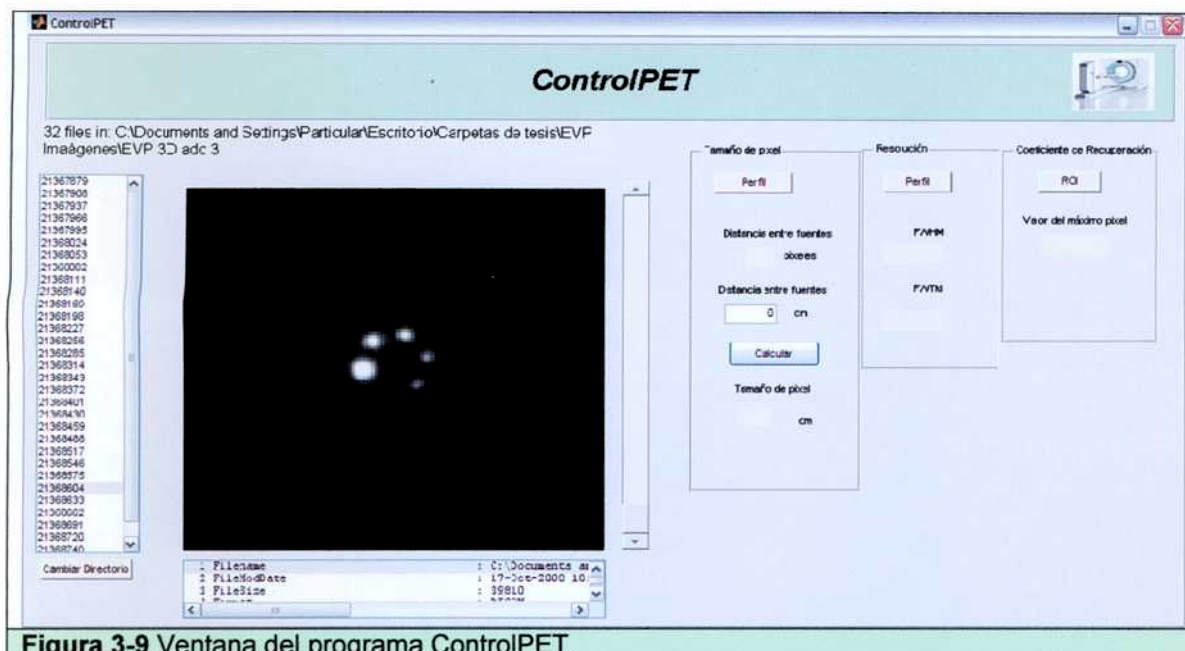


Figura 3-9. Ventana del programa ControlPET.

3.4.2.1 Tamaño de píxel

Se emplea para determinar la equivalencia en medidas de longitud de un píxel en una imagen. Se tecldea sobre el botón *perfil* y se traza una línea que pase sobre dos fuentes puntuales (Figura 3-10), de las cuales se conoce previamente su separación en unidades de longitud; el programa calcula la distancia de separación entre fuentes en pixeles y calcula el tamaño de píxel en unidades de longitud.

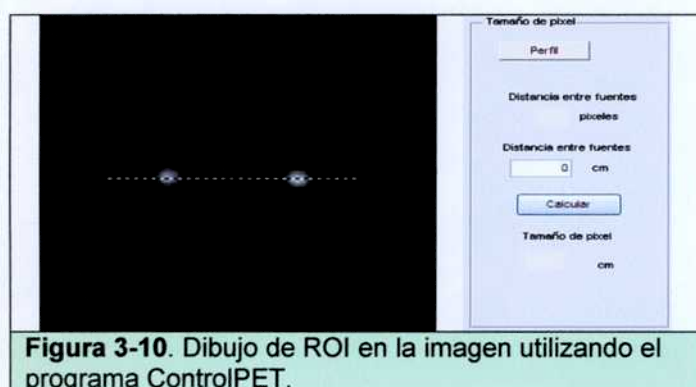
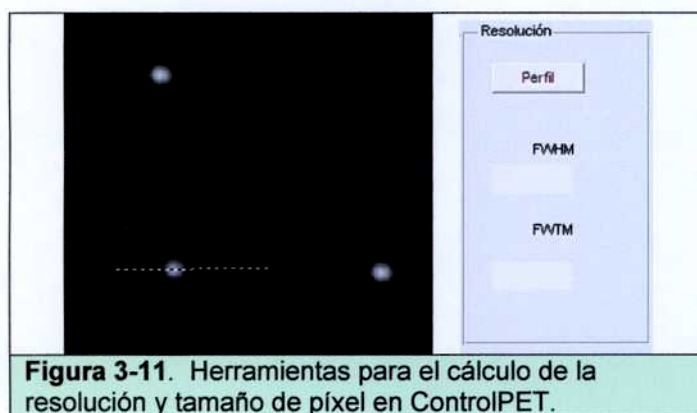


Figura 3-10. Dibujo de ROI en la imagen utilizando el programa ControlPET.

3.4.2.2 Resolución espacial

Esta función calcula el FWHM y FWTM. Es un proceso semiautomático, en el que el usuario traza una línea de perfil sobre la imagen de una fuente puntual (Figura 3-11), y obtiene los valores en las casillas correspondientes en píxeles. Posteriormente estos valores deberán ser multiplicados por el tamaño de píxel para conocer su valor en unidades de longitud.



3.4.2.3 Coeficiente de recuperación

Para encontrar el Coeficiente de recuperación (CR) se deben seleccionar primero las ROIs en cada uno de las lesiones calientes simulados por el *fantoma E* y que se ven en la imagen como círculos brillantes. Esto se logra mediante el botón ROI . En general se tendrá que:

- ❖ Crear ROIs que se ajusten al tamaño de la lesión en la imagen. La ROI se puede desplazar mediante la acción del *mouse*.
- ❖ Abrir el menú contextual de estadística de la ROI, presionando dos veces el botón derecho del *mouse* sobre la circunferencia de la ROI. El programa muestra el valor del máximo píxel, para cada lesión, Figura 3-12.

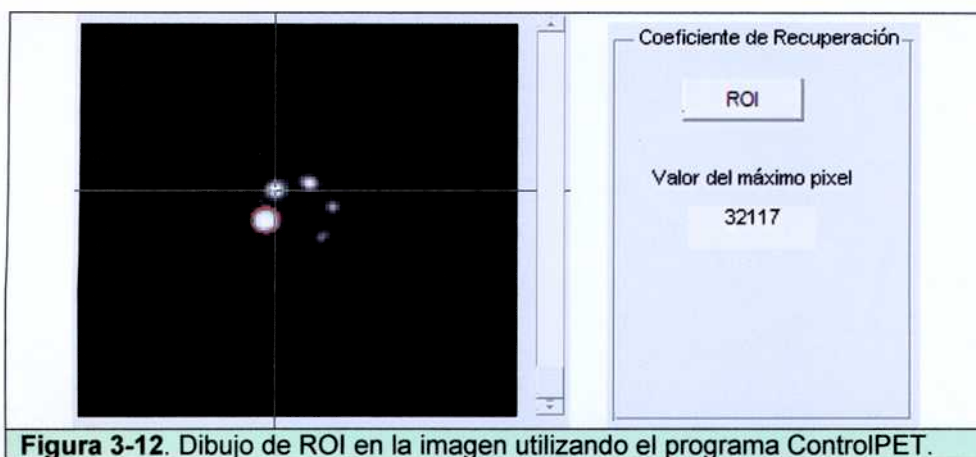


Figura 3-12. Dibujo de ROI en la imagen utilizando el programa ControlPET.

El programa EVP, que forma parte de ControlPET, obtiene la gráfica del coeficiente de recuperación a partir del diámetro de las esferas del *fantoma* E y del valor del máximo píxel en cada punto caliente de la imagen. Es necesario ingresar los datos del diámetro de las lesiones y del valor del máximo píxel asociado a éstas. Al presionar el botón *graficar* el programa interpola y genera la gráfica del coeficiente de recuperación (Figura 3-13).

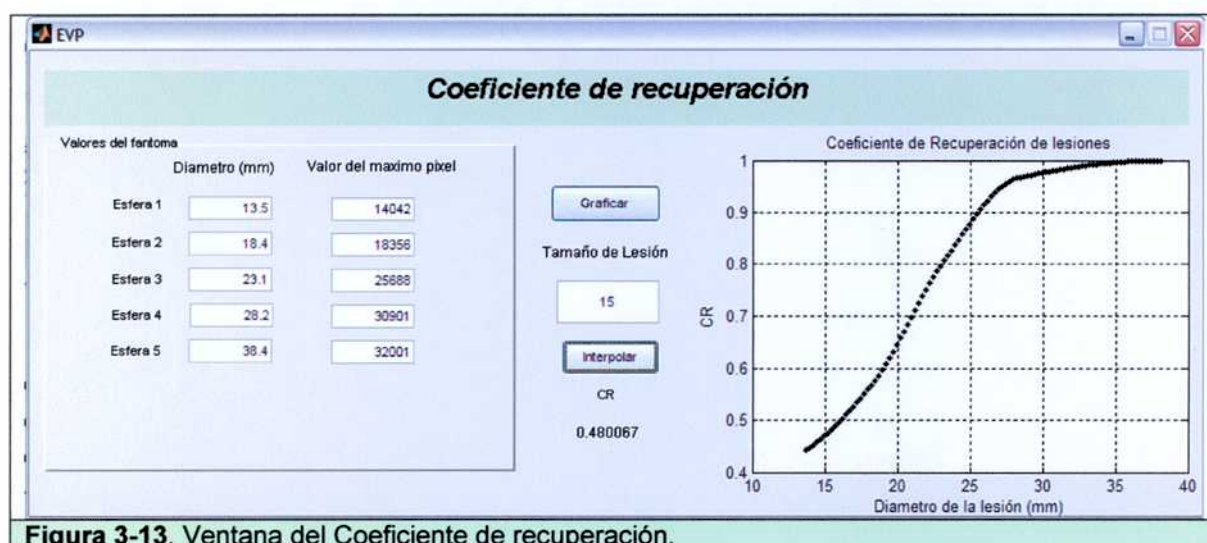


Figura 3-13. Ventana del Coeficiente de recuperación.

La grafica generada por el programa corresponde a los valores de CR, donde el valor de máximo píxel de cada esfera es dividido por el valor de máximo

píxel de la esfera de mayor diámetro, de esta manera el valor 1 corresponde a la esfera que esta menos afectada por el efecto del volumen parcial.

Para obtener el CR de una lesión, se ingresa el diámetro de la lesión en la casilla *tamaño de lesión* y se tecléa sobre el botón interpolar.

3.5 APLICACIONES CLÍNICAS

El coeficiente de recuperación se utiliza para la corrección del SUV en imágenes de lesiones de una serie de pacientes, para esto es necesario conocer previamente el tamaño de la lesión y el valor de SUV. El tamaño de la lesión es medido en la imagen de fusión PET-CT y el SUV es calculado con el programa del equipo PET. El tomógrafo computado CT brinda imágenes de mejor resolución que el PET por lo que brinda una forma mas precisa para medir el tamaño de lesión. Conocido el tamaño se ingresaron los valores al programa EVP para calcular el valor del coeficiente de recuperación propio de la lesión.

Sólo las imágenes de pacientes con protocolo de adquisición y parámetros de reconstrucción iguales a las imágenes del *fantoma* E de esferas se corrigen. El SUV corregido es el cociente que resulta de dividir el SUV original por el CR.

4. RESULTADOS

4.1 CONTROL DE CALIDAD DEL PET

4.1.1 Control de calidad diario

El control diario del equipo dio valores de varianza promedio entre barridos menores a 2.5, lo que indicaba que el equipo estaba en condiciones de trabajo.

4.1.2 Control de calidad de dos camillas

El control de calidad fue aceptable ya que el volumen reconstruido se aprecio completo y continuo lo que indica que la superposición de la camilla es satisfactoria.

4.1.3 Resolución espacial y tamaño de píxel

La Figura 4-1 muestra imágenes de corte axial del *fantoma* de fuentes puntuales, adquiridas en un equipo *PET ECAT* de *Siemens*. Estas imágenes se analizaron con *ControlPet* para determinar el FWHM del equipo a través del análisis del perfil de una fuente puntual y con éste, la resolución espacial del PET. Además, dado que se conoce la distancia entre 2 fuentes de la imagen de la derecha, se la empleó para determinar el tamaño de píxel.

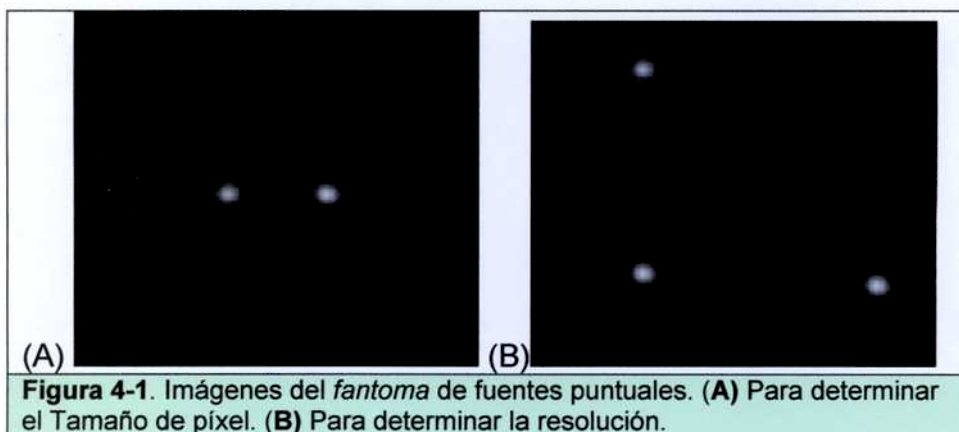
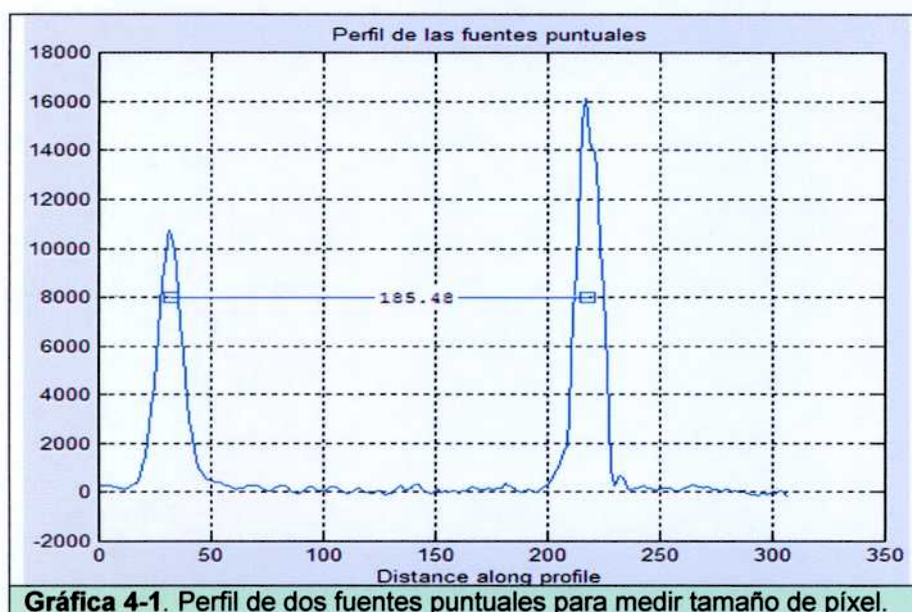


Figura 4-1. Imágenes del *fantoma* de fuentes puntuales. (A) Para determinar el Tamaño de píxel. (B) Para determinar la resolución.

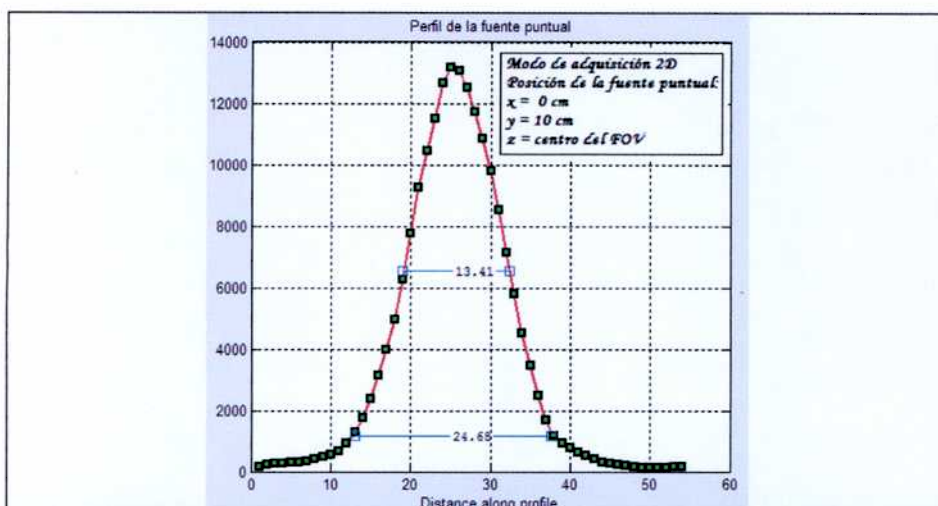
4.1.3.1 Tamaño del píxel

La Gráfica 4-1 resulta al trazar el perfil de las fuentes puntuales de la Figura 4-1A. La posición relativa de los picos mayores de este perfil se emplearon para determinar el tamaño del píxel (0.467 ± 0.019) mm. Este valor está en conformidad con el valor teórico del tamaño del píxel= 0.455 mm.



4.1.3. 2 Resolución espacial

La Gráfica 4-2 se obtuvo al trazar el perfil sobre una sola fuente del *fantoma*. Este perfil presenta una forma de campana que es el esperado de la actividad de una fuente puntual. A partir de esta curva se determinó el FWHM y FWTM. La Tabla 4-1 muestran los valores obtenidos tanto en imágenes 2D como en imágenes 3D y en distintas posiciones del FOV.



Gráfica 4-2. Perfil de la fuente puntual. Se observa que el FWHM es de 13.41 píxel y el FWTM es de 24.68 píxel.

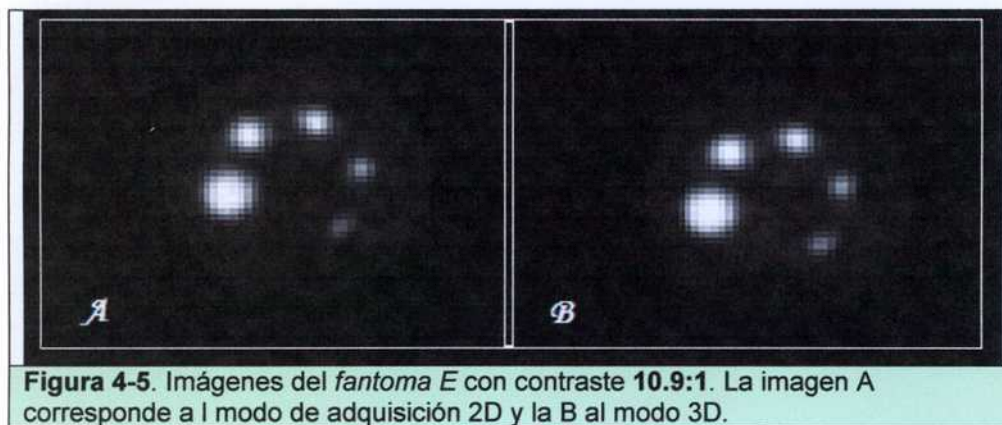
Tabla 4-1. Resolución axial en adquisición 2D.				
A 1 cm de radio	Posición	FWHM	FWTM	
	$x = 0, y = 0, z = \text{centro FOV}$	11.07	19.24	píxel
	$x = 0, y = 0, z = 1/4\text{FOV}$	12.85	19.91	píxel
	Promedio	11.96	19.57	píxel
		5.80*	9.14*	mm
A 10 cm de radio	Posición	FWHM	FWTM	
	$x = 10, y = 0, z = \text{centro FOV}$	13.41	24.68	píxel
	$x = 0, y = 10, z = \text{centro FOV}$	13.20	28.46	píxel
	$x = 10, y = 0, z = 1/4\text{FOV}$	12.86	23.09	píxel
	$x = 0, y = 10, z = 1/4\text{FOV}$	13.11	28.45	píxel
	Promedio	13.14	26.17	píxel
		6.14*	12.22*	mm

Tabla 4-2. Resolución axial en adquisición 3D.				
A 1 cm de radio	Posición	FWHM	FWTM	
	$x = 0, y = 0, z = \text{centro FOV}$	12.82	20.32	píxel
	$x = 0, y = 0, z = 1/4\text{FOV}$	12.62	19.34	píxel
	Promedio	12.72	19.83	píxel
		5.94*	9.26*	mm
A 10 cm de radio	Posición	FWHM	FWTM	
	$x = 10, y = 0, z = \text{centro FOV}$	12.82	20.32	píxel
	$x = 0, y = 10, z = \text{centro FOV}$	13.37	24.83	píxel
	$x = 10, y = 0, z = 1/4\text{FOV}$	13.29	24.25	píxel
	$x = 0, y = 10, z = 1/4\text{FOV}$	13.57	30.88	píxel
	Promedio	13.26	26.07	píxel
		6.19*	11.71*	mm

* Valores de FWHM y FWTM expresados en mm usando la relación 1 píxel = 0.467 mm.

4.2 COEFICIENTE DE RECUPERACIÓN

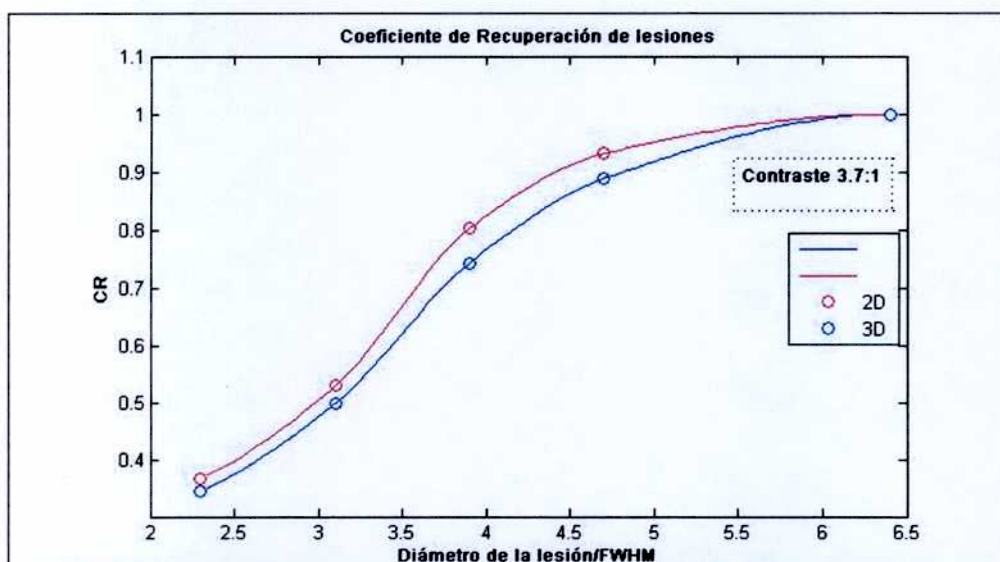
Las imágenes del *Fantoma E* de lesiones calientes correspondientes a diferentes contrastes y a diferentes modos de adquisición, se muestran en la Figuras 4-3 a 4-5.



En las Tablas 4-3 a 4-5 se presenta el valor de máximo píxel dentro de cada ROI de las imágenes adquiridas con el *Fantoma E*. Las razones de contrastes empleadas son 3.7:1, 5.8:1, 10.9:1 para los modos de adquisición 2D y 3D.

En as Gráficas 4-3 a 4-5 se presenta el *coeficiente de recuperación* en función el *diámetro de la lesión / FWHM*, considerando un FWHM promedio de 6 mm.

Tabla 4-3. Datos de adquisición para una relación de contraste 3.7:1				
Diámetro de lesión (mm)	Adquisición 2D		Adquisición 3D	
	Valor del máximo píxel	CR	Valor del máximo píxel	CR
38.4	32767	1.0000	32715	1.0000
28.2	30542	0.9321	29065	0.8884
23.1	26303	0.8027	24319	0.7434
18.4	17357	0.5297	16300	0.4982
13.5	12046	0.3676	11274	0.3446



Gráfica 4-3. Comparación de modos de adquisición para la relación de contraste 3.7:1.

Tabla 4-4. Datos de adquisición para una relación de contraste 5.8:1				
Diámetro de lesión (mm)	Adquisición 2D		Adquisición 3D	
	Valor del máximo píxel	CR	Valor del máximo píxel	CR
38.4	32001	1.0000	32640	1.0000
28.2	30901	0.9656	29528	0.9047
23.1	26184	0.8182	25138	0.7702
18.4	18253	0.5704	16358	0.5011
13.5	14042	0.4388	11710	0.3588

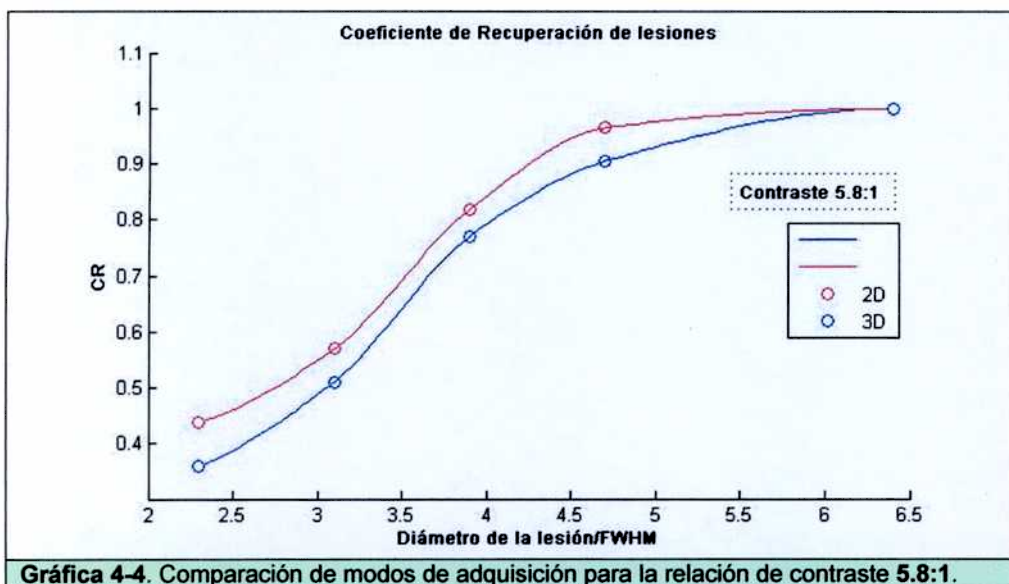
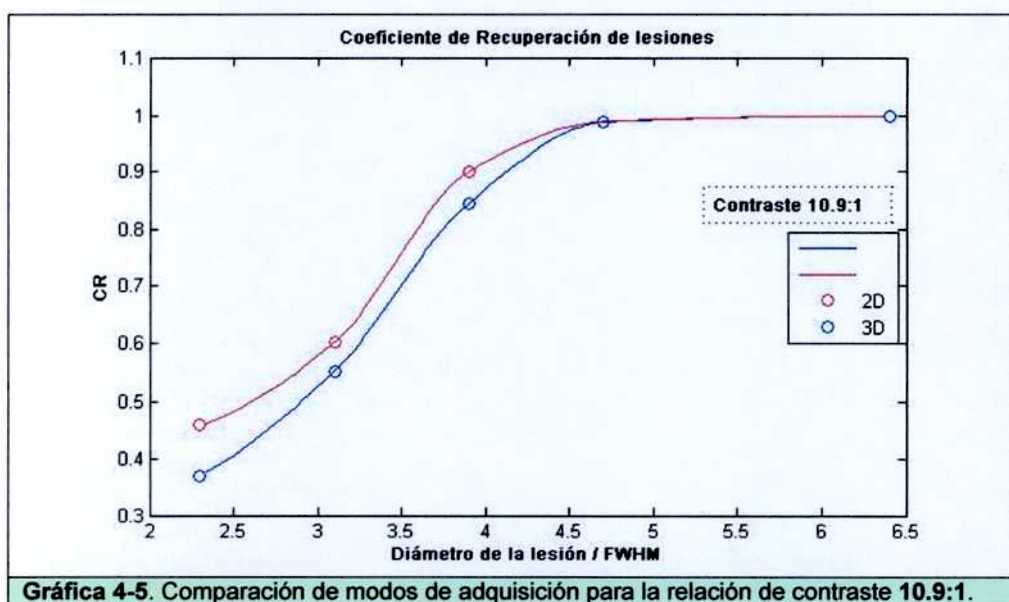
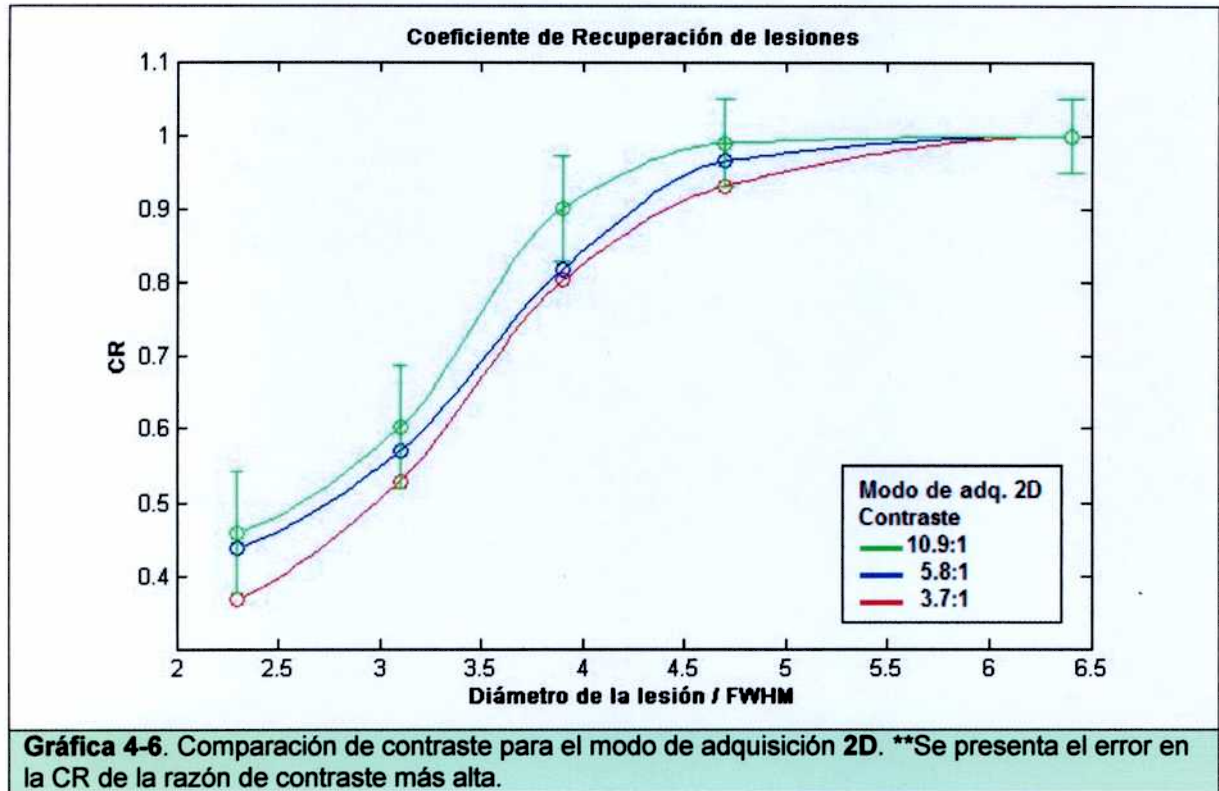


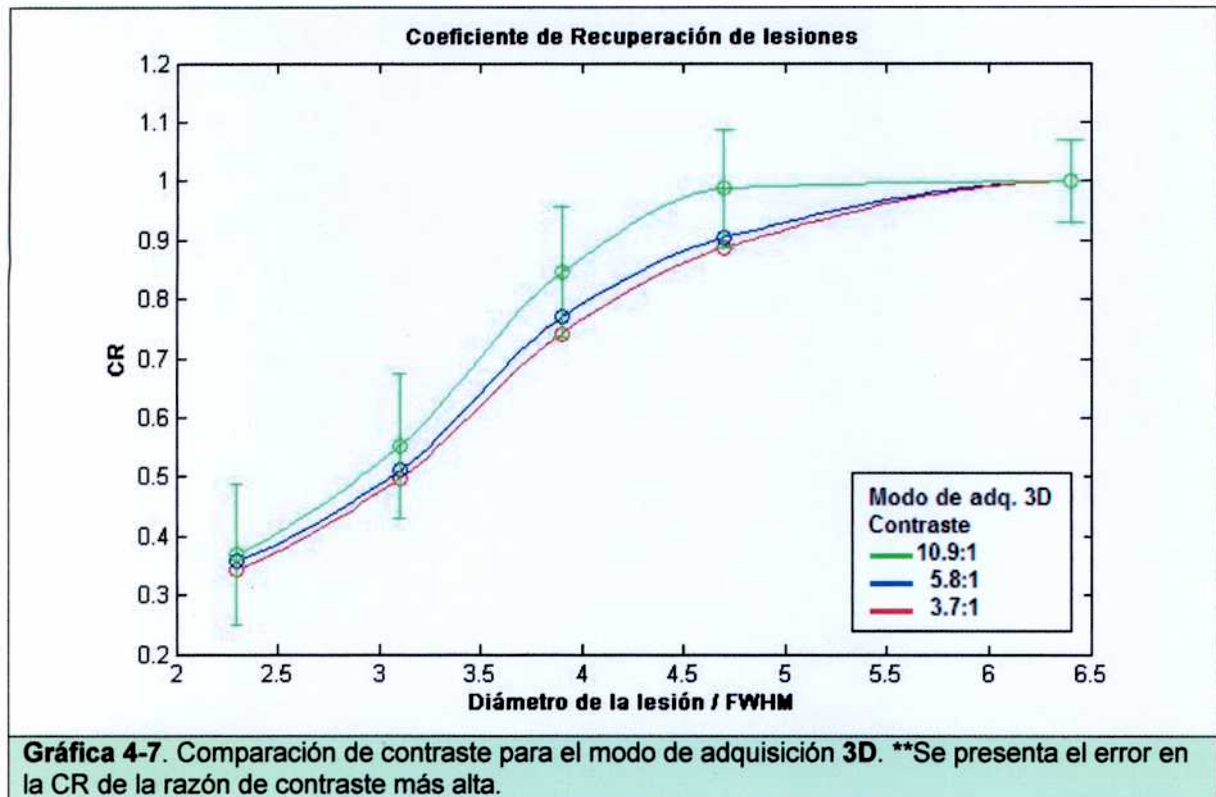
Tabla 4-5. Datos de adquisición para una relación de contraste 10.9:1

Diámetro de lesión (mm)	Adquisición 2D		Adquisición 3D	
	Valor del máximo píxel	CR	Valor del máximo píxel	CR
38.4	32117	1.000	31767	1.000
28.2	31803	0.9902	31413	0.9889
23.1	28956	0.9016	26873	0.8459
18.4	18423	0.5736	15983	0.5031
13.5	14757	0.4595	11749	0.3698



Para evaluar la variación del CR en función del contraste se representó gráficamente, el CR para los tres contrastes, y un solo modo de adquisición. Los resultados se presentan en las Gráficas 4-6 y 4-7.





** El porcentaje de ruido en la imagen [6] se calcula = $100\% / SNR_{PIXEL}$

Donde $SNR_{PIXEL} \approx \sqrt{\frac{12N_{image}}{\pi^2 (D/d)^3}}$

N_{image} = número de cuentas de la esfera.

D = diámetro de cada esfera

d = distancia de muestreo lineal (4.67mm) correspondiente al tamaño de matriz.

Como el CR es un cociente, la propagación del error = error del denominador + error en el numerador.

4.3 APLICACIONES CLÍNICAS

En la Tabla 4-6 se presentan los valores obtenidos al realizar la corrección de SUV a imágenes de lesiones en distintos tejidos de doce pacientes. Los valores de coeficiente de recuperación usados para corregir el SUV se obtuvieron a partir de imágenes adquiridas con los mismos parámetros de reconstrucción que las imágenes de los pacientes y con la mayor razón de contraste, 10.9:1.

Tabla 4-6. Tamaño de lesiones, SUV y coeficiente de recuperación de imágenes de pacientes adquiridos en PET.							
Paciente	Patología	Área	Adq.	Tamaño de lesión(mm)	SUV	CR	SUV corregido
1	CA Pulmón	Ganglio Retroperitoneal	2D	12.4	2.4	0.409	5.86
2	CA Pulmón	Ganglio Hiliar	3D	12.5	2.77	0.324	8.55
3	CA Pulmón	Nod Pleural	3D	17.7	2.01	0.484	4.15
4	CA Pulmón	Nod Pulmonar	2D	10.8	1.57	0.360	4.37
5	CA Laringe	Nod Pulmonar	3D	10.8	1.53	0.268	5.71
6	Rabdomiosarcoma	Pleural	2D	14.0	1.12	0.471	2.38
7	Rabdomiosarcoma	Pleural 2	2D	12.5	2.88	0.413	6.98
8	Rabdomiosarcoma	Pleural 3	2D	10.0	4.23	0.338	12.52
9	Coriocarcinoma	Ovario	2D	23.7	4.1	0.912	4.50
10	Coriocarcinoma	Promontorio	2D	17.2	1.63	0.546	2.99
11	CA Rectosigmoideo	Hígado	3D	22.2	5.45	0.780	6.98
12	Tiroides	Cuello (Resto)	3D	19.8	3.52	0.605	5.82

En la Figura 4-6 y 4-7 se presentan dos de los casos estudiados: el primero corresponde a una lesión en el pulmón y el segundo a una lesión en el hígado.

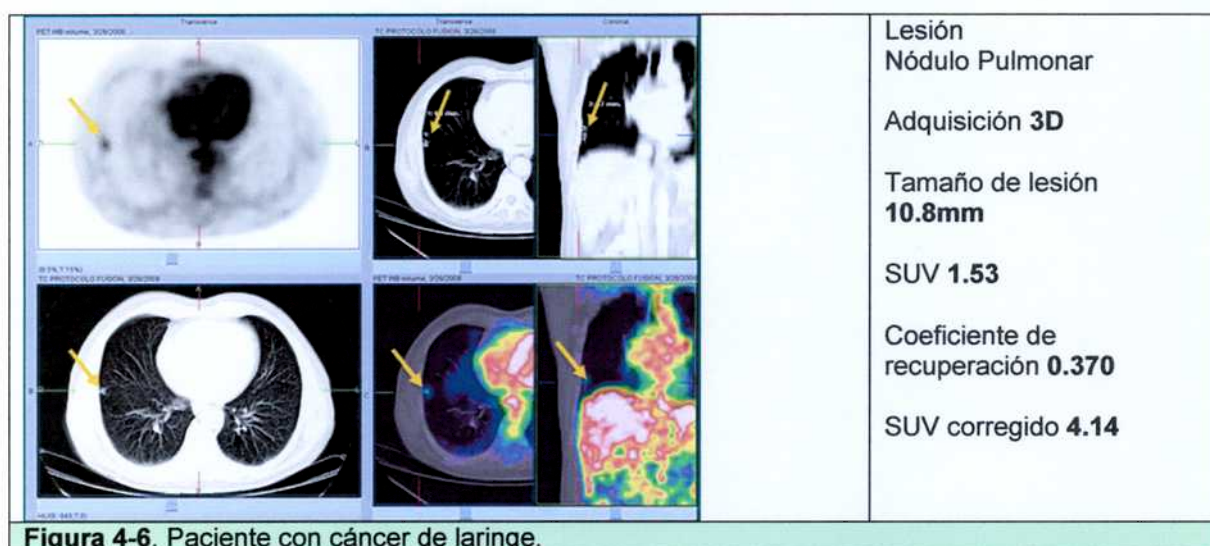


Figura 4-6. Paciente con cáncer de laringe.

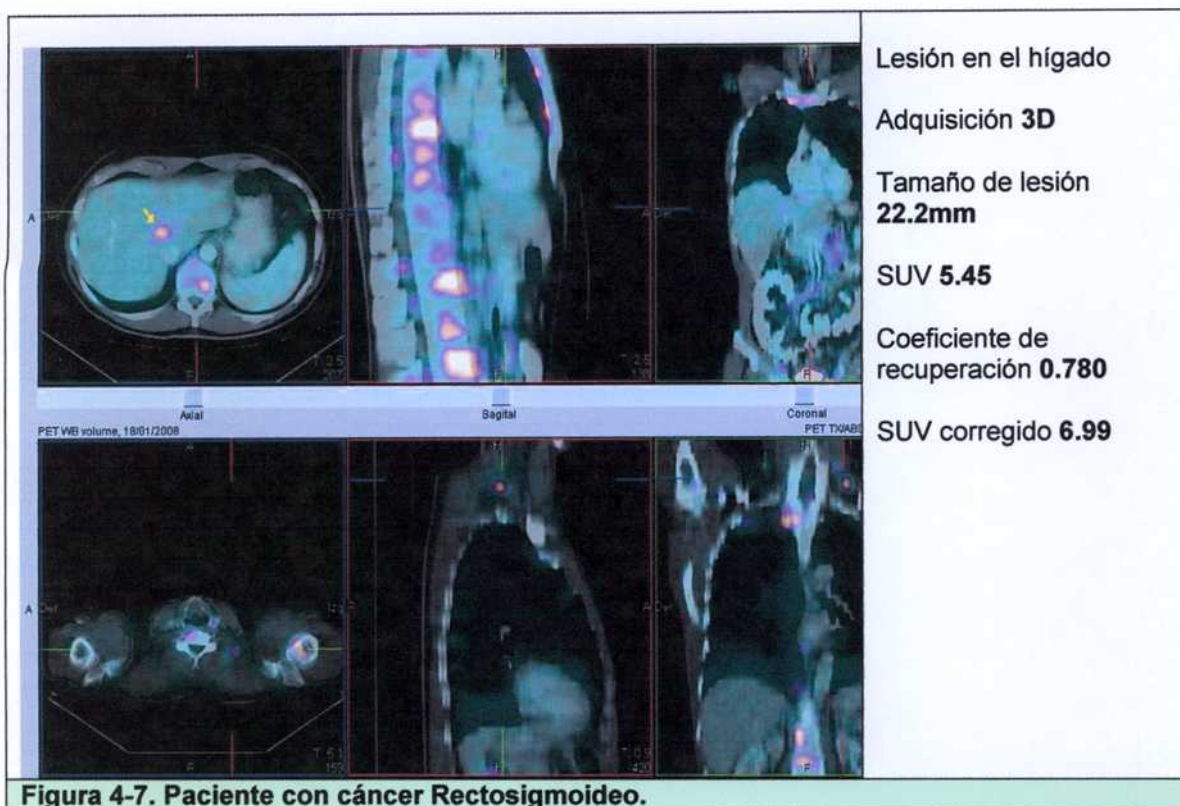
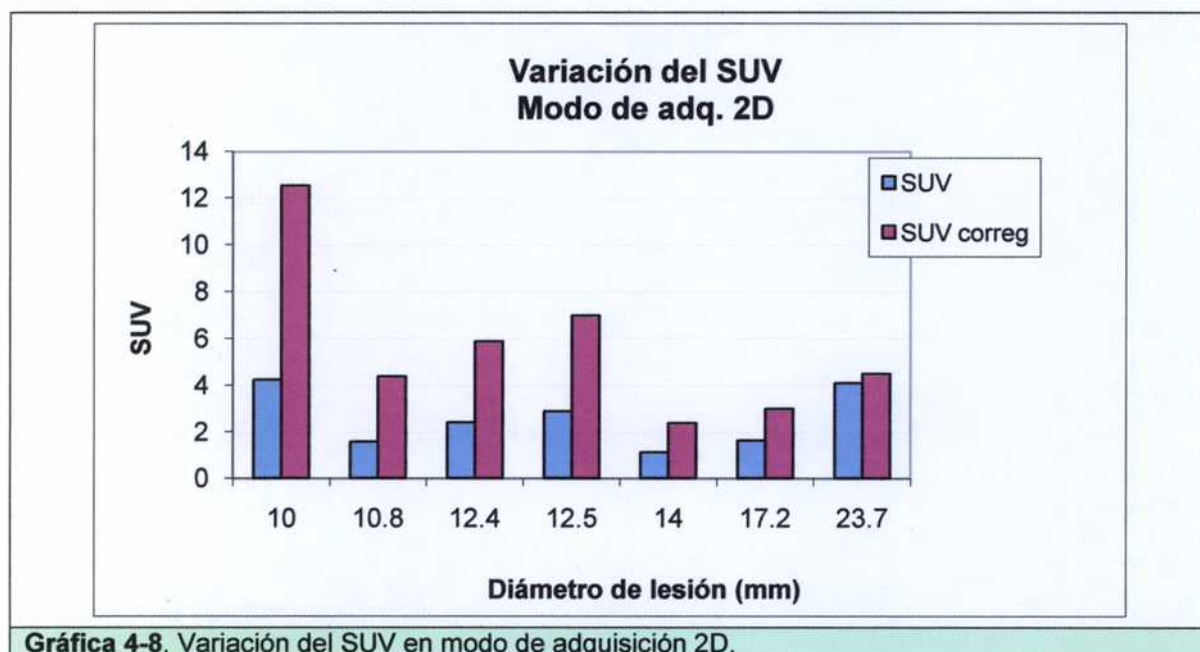
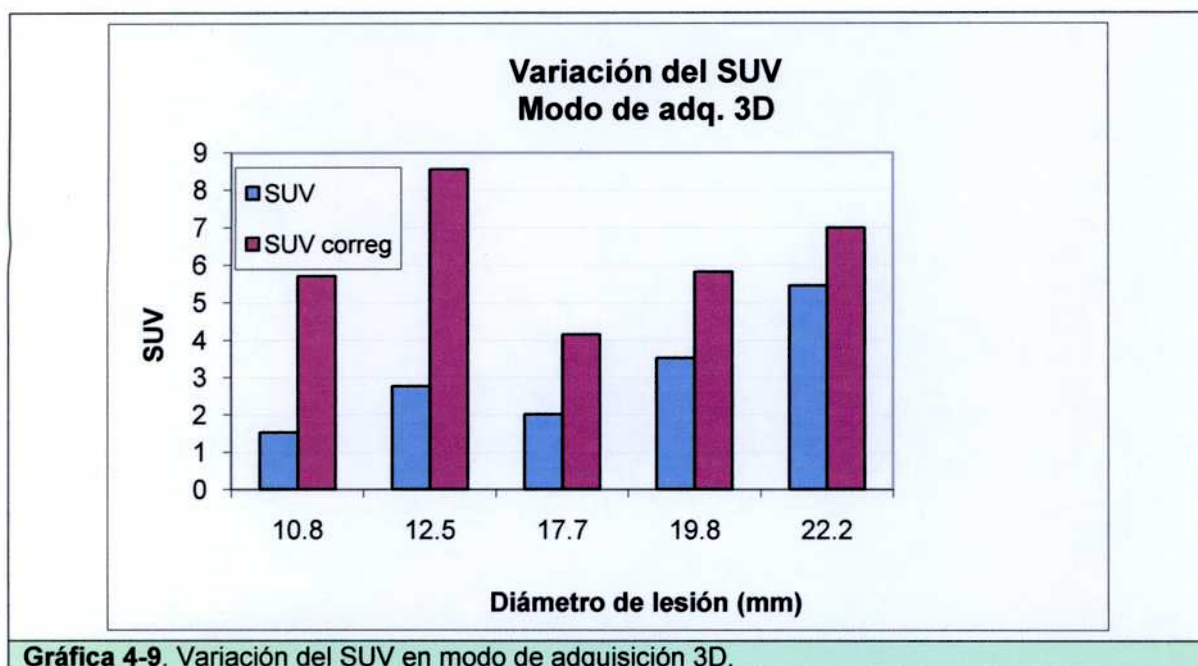


Figura 4-7. Paciente con cáncer Rectosigmoideo.

En la Gráfica 4-8 y 4-9 se presentan las magnitudes del SUV antes y después de la corrección, correspondiente al modo de adquisición 2D y 3D.

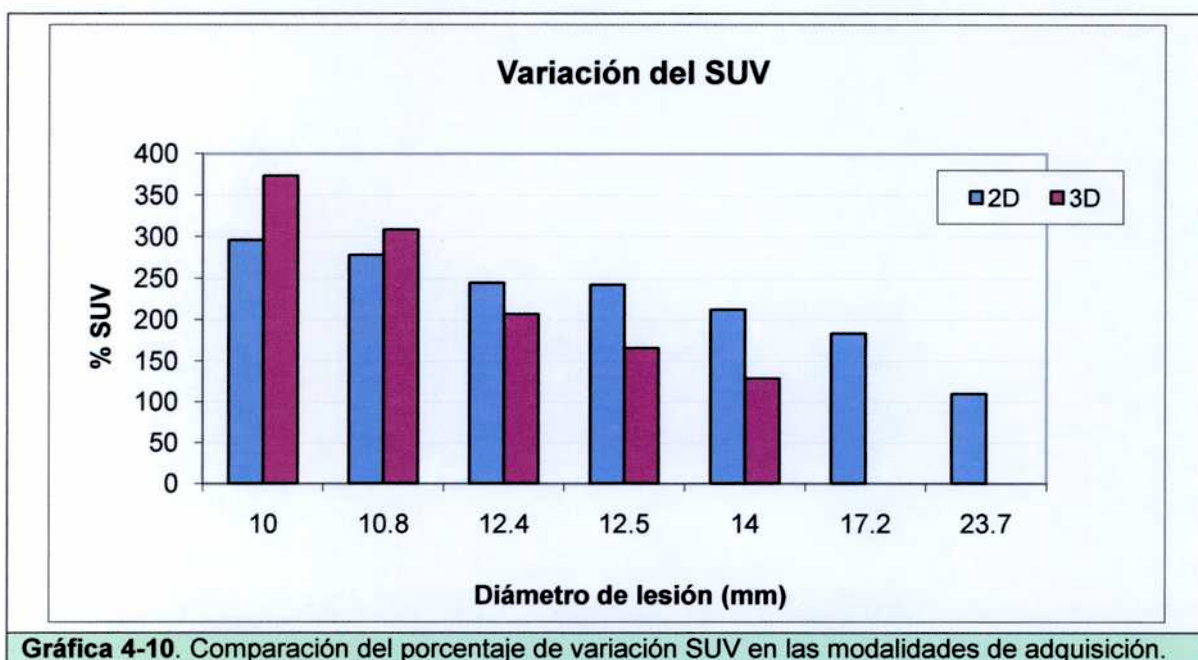


Gráfica 4-8. Variación del SUV en modo de adquisición 2D.



Gráfica 4-9. Variación del SUV en modo de adquisición 3D.

En la Gráfica 4-10 se presenta la variación porcentual del SUV correspondiente al modo de adquisición 2D y 3D.



Gráfica 4-10. Comparación del porcentaje de variación SUV en las modalidades de adquisición.

5. DISCUSIÓN

5.1 CONTROL DE CALIDAD DEL PET

La resolución espacial axial del equipo PET, del orden de 6 mm, se determinó a partir del FWHM de una fuente puntual y de la medida experimental del tamaño de píxel. Este valor brinda una estimación del tamaño de lesiones que se pueden detectar con el PET ($1.75 \times \text{FWHM}$) [17] y es congruente con las especificaciones del fabricante y con publicaciones previas [13].

Hay que tener presente que la resolución espacial fue medida en condiciones que no representa exactamente la situación en que se adquieren normalmente las imágenes de lesiones: con mayor posibilidad de eventos de dispersión y el número limitado de eventos adquiridos que implica el uso de filtros de reconstrucción. De esta manera la resolución espacial así medida representa un caso ideal, indicando más alto desempeño que se puede alcanzar en un sistema PET [25].

5.2 COEFICIENTE DE RECUPERACIÓN

En las imágenes del *fantoma* E (Figura 4-3 a 4-5) solo se ven cinco esferas, la lesión de 8 mm no se observa, esto se debe al *efecto del volumen parcial*, ya que el tamaño de la lesión está en el orden de la resolución espacial del equipo.

Las Gráficas 4-3 a 4-5 muestran que para un mismo contraste, la recuperación en el modo de adquisición 2D es mayor que en el modo de 3D. Esto se debe a que la resolución espacial del equipo es diferente para cada caso, principalmente por empleo de septas en el modo de adquisición 2D la resolución espacial es menor que para modo 3D.

La Gráfica 4-6 y 4-7 muestran que para una misma modalidad de adquisición y diferentes contrastes, ocurre una mayor recuperación de contraste a medida que se aumenta la razón de contraste de la lesión con respecto al fondo. Sin embargo el error calculado para la razón de contraste 10.9:1 es alto y cubre los CR de los otros contrastes.

5.3 APLICACIONES CLÍNICAS

Las correcciones del SUV en lesiones de pacientes se hicieron con imágenes que seguían el mismo modo de adquisición y parámetros de reconstrucción, ya que estudios presentados por [28] ha demostrado que existe diferencia al variar el número de iteraciones en el parámetro de SUV. Por lo tanto las correcciones son confiables desde el punto de vista del método de reconstrucción.

Desde el punto de vista del contraste, los valores de SUV fueron corregidos con la razón de contraste más alta, dado que a mayor contraste existe una mejor recuperación, el valor de SUV corregido no fue sobrestimado.

La Gráfica 4-8 y 4-9 muestra que a medida que aumenta el tamaño de la lesión la magnitud de SUV sin corregir se aproxima más al valor de la magnitud del SUV corregido. Esto demuestra que para lesiones de mayor tamaño el *efecto del volumen parcial* disminuye.

La Gráfica 4-10 presenta el porcentaje de variación del SUV, se demuestra que existe una mayor variación de SUV en la modalidad 3D con respecto a la modalidad 2D.

6. CONCLUSIÓN

Se corrigió la influencia del *efecto del volumen parcial* en imágenes PET, a través de la determinación del SUV de lesiones de pacientes y el coeficiente de recuperación en imágenes de un fantoma de esferas calientes. Esta metodología es válida: cuando las lesiones que se quieran corregir se aproximan a una geometría esférica; cuando las imágenes del fantoma y de las lesiones se hayan adquirido con la misma modalidad de adquisición y con los mismos parámetros de reconstrucción.

El SUV es un parámetro importante para cuantificar una lesión y su valor demostró que está influenciado por EVP, como ya se establecido en otros estudios. Dado que el SUV se utiliza como un valor frontera para clasificar una lesión como benigna o patológica (en caso de ser patológica se caracteriza por un $SUV > 2.5$), la determinación correcta de este parámetro (CR) sirve para corregir el SUV y de esta manera obtener un valor de captación más cercano al valor verdadero.

Para lesiones aproximadamente menores a 23.1 mm, el CR es menor a 90%, lo que indica que se hace necesario corregir el SUV para obtener un valor más cercano al valor verdadero de captación de la lesión.

El programa desarrollado en MATLAB permitió utilizar las imágenes en formato DICOM, determinar la resolución del equipo PET, calcular el CR de una lesión en función de su diámetro. Este programa es una base para futuros desarrollos en este campo.

El trabajo realizado requiere profundización en aspectos como: estudio con distintas concentraciones de actividad, tiempos de adquisición y diferentes niveles de contraste; el impacto del ruido de la imagen sobre el cálculo del CR y

considerar el contraste de la lesión del paciente para utilizar el valor del CR que sea más apropiado para un tejido, cuando se usa FDG y otros radiofármacos como radiotrazadores.

7. BIBLIOGRAFÍA

- [1] Adler L., Blair H., Makley J., et al. Noninvasive grading of musculoskeletal tumors using PET. *J Nucl Med* 1991; 32:1508-1512.
- [2] Autoridad Regulatoria Nuclear. Curso de protección radiológica-nivel técnico. 2007.
- [3] Bailey D. Positron emission tomography: basic sciences. Springer Science+Business Media, 2005.
- [4] Boussion N., Hatt M., Lamare F., et al. A multiresolution image based approach for correction of partial volume effects in emission tomography. *Phys. Med. Biol.* 51(2006) 1857–1876.
- [5] Cabrejas M. Tomografía en Medicina Nuclear. Editado por Comité de Instrumentación y Garantía de calidad de ALASBIMN (1999).
- [6] Cherry S., Sorenson J., Phelps M. *Physics in Nuclear Medicine*. Third edition. Elsevier, 2003.
- [7] Daube-Witherspoon M., Karp J., Casey M., Frank P., et al. PET Performance Measurement using NEMA NU 2-2001 Standard. *Journal of Nuclear Medicine* Vol. 43 No. 10 1398-1409. October 2002.
- [8] Erdi Y., Nehmeh S., Mulnix T., et al. PET Performance Measurements for an LSO-Based Combined PET/CT Scanner Using the National Electrical Manufacturers Association NU 2-2001 Standard. *Journal of Nuclear Medicine* Vol. 45 No. 5 813-821.
- [9] González G., Cabrejas R., Cabrejas M. Influencia del efecto del volumen parcial (EVP) en la cuantificación de lesiones pequeñas: Su efecto sobre el SUV. *Física y Radioprotección: XVI Congreso Argentino de Biología y Medicina Nuclear*, 2008.
- [10] González P., Massardo T., Canessa J., et al. Aplicaciones clínicas del PET. *Rev. méd. Chile* v.130 n.5 Santiago mayo 2002.
- [11] Gjedde A. Calculation of cerebral glucose phosphorylation from brain uptake of glucose analogs in vivo: a re-examination. *Brain Res Rev* 1982; 4: 237-274.
- [12] Gopal S. *Basics of PET Imaging: Physics, Chemistry, and Regulations*. Springer, 2005.
- [13] Herzog H., Tellmann L., Hocke C., et al. NEMA NU2-2001 guided performance evaluation of four Siemens ECAT PET-scanners. *Nuclear Science Symposium Conference Record, 2003 IEEE Volume 4, Issue 19-25 Oct. 2003 Page(s): 2836 - 2838 Vol.4*.
- [14] Hoffman E., Huang S., Phelps M. Quantitation in positron emission computed tomography, 1: effect of object size. *J Comput Assist Tomogr.* 1979 Jun; 3(3):299-308.
- [15] Huang SC. Anatomy of SUV. Standardized uptake value. *Nucl Med Biol* 2000; 27: 643- 646.
- [16] Hudson HM, Larkin RS. Accelerated image reconstruction using ordered subsets of projection data. *IEEE Trans Med Imaging.* 1994;13: 601.
- [17] IAEA. Control de calidad de los instrumentos en medicina nuclear, 1991. TECDOC 602/S, Diciembre de 1996.
- [18] Kessler R., Ellis J., Eden M. Analysis of emission tomographic scan data: limitations imposed by resolution and background. *J Comput Assist Tomogr.* 1984; 8:514–522.

- [19] Kirov A., J Z Piao J., Schmidlein C. Partial volume effect correction in PET using regularized iterative deconvolution with variance control based on local topology. *Phys. Med. Biol.* 53 (2008) 2577–2591.
- [20] Leahy, R. M. and Qi, J. Statistical approaches in quantitative positron emission tomography. *Statistics and Computing*, Volume 10, Number 2, April 2000, pp. 147-165(19).
- [21] Lewellen TK, Kohlmeyer S, Miyaoka R, Schubert S, Stearns C. Investigation of the count rate performance of the General Electric Advance positron emission tomograph. *IEEE Trans Nucl Sci* 1995;42:1051-1057.
- [22] Marchand, P. Graphics and GUIs with MATLAB. Chapman & Hall/CRC, Florida: 2003.
- [23] Márquez I., Díez R., Álvarez J. Positron Emission Tomography for the Study of Solitary Pulmonary Nodules. *ScienceDirect, Archivos de Bronconeumología*, Volume 44, Issue 9, 2008, Pages 493-498.
- [24] Mazziotta, J., Phelps, M., Plummer D., Kuhl, D. Quantitation in positron emission computed tomography: 5. physical– anatomical effects. *J Comput Assist Tomogr* (1981), 5(5):734–43.
- [25] National Electrical Manufacturers Association. NEMA Standards Publication NU 2-2001: Performance Measurements of Positron Emission Tomographs. Rosslyn, VA: National Electrical Manufacturers Association; 2001.
- [26] Phelps M, PET Physics, Instrumentation, and Scanners. Springer Science+Business Media, 2006.
- [27] Rousset O., Ma Y., Evans A. Correction for partial volume effects in PET: principle and validation. *The Journal of Nuclear Medicine* Vol. 39 (1998) No. 5 904-911.
- [28] Sánchez A., Barandela J., Garrido M., et al. Utilidad de la tomografía por emisión de positrones (PET) con ^{18}F -FDG en la estadificación del carcinoma de pulmón. *Pneuma* 2006; 6: 143 - 150
- [29] Sharma S., Prasad R., Shetye B., et al. Whole-body PET acceptance test in 2D and 3D using NEMA NU 2-2001 protocol. *Journal of Nuclear Medicine* Vol. 32 No. 4, 2007.
- [30] Shep L. Maximum likelihood reconstruction for emission tomography. *IEEE Trans Med Imag*, 1: 113, 1982.
- [31] Soret M., Bacharach S., Buvat I. Partial-Volume Effect in PET Tumor Imaging. *JNM*, June 2007, Volume 48, No. 6.
- [32] Tsujikawa T., Otsuka H., Morita N., et al. Does partial volume corrected maximum SUV based on count recovery coefficient in 3D-PET/CT correlate with clinical aggressiveness of non-Hodgkin's lymphoma?. *Ann Nucl Med* (2008) 22:23-30.
- [33] Turkheimer F., Boussion N, Anderson A., et al. PET image denoising using a synergistic multiresolution analysis of structural (MRI/CT) and functional datasets. *J Nucl Med*. 2008 Apr;49(4):657-66.
- [34] Zaidi H. Quantitative Analysis in Nuclear Medicine Imaging. Editado por Springer Science+Business Media (2006).
- [35] <http://www.cerebromente.org.br/>
- [36] <http://www.fundacionannavazquez.wordpress.com/2007/10/17/aplicaciones-clinicas-de-la-pet/>
- [37] <http://www.mathworks.es/services/training/policies/requirements.html>
- [38] <http://www.pethistory.us/>

ANEXO. ALGORITMO EN MATLAB

Se presentan los algoritmos en MATLAB del programa ControlPET y programas de respaldo. Para generar este programa se debe guardar los algoritmos en el editor de MATLAB con sus respectivos nombres.

- ❖ ControlPET
- ❖ Fwhm y fwtm (Patrick Egan, Rev 1.2, April 2006)***
- ❖ Draggable (Francois Bouffard, 20 Nov 2003 (Updated 01 Aug 2006))***
- ❖ Roitool (Laszlo Balkay, 29 Nov 2005 (Updated 25 Feb 2008))***
- ❖ EVP

*** Archivo de descarga libre bajado de Matlab Central
(<http://www.mathworks.com/matlabcentral/>)

ControlPET

```
function varargout = ControlPET(varargin)
% ControlPET M-file for ControlPET.fig
% ControlPET, by itself, creates a new ControlPET or
% raises the existing
% singleton*.
%
% H = ControlPET returns the handle to a new ControlPET
% or the handle to
% the existing singleton*.
%
%
% ControlPET('CALLBACK',hObject,eventData,handles,...)
% calls the local
% function named CALLBACK in ControlPET.M with the
% given input arguments.
%
% ControlPET('Property','Value',...) creates a new
% ControlPET or raises the
% existing singleton*. Starting from the left, property
% value pairs are
% applied to the GUI before
% DICOMViewer_OpeningFunction gets called. An
% unrecognized property name or invalid value makes
% property application
% stop. All inputs are passed to ControlPET_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
% DICOMFILES

% Last Modified by GUIDE v2.5 27-Nov-2008 10:46:55

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

end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
    varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% End initialization code - DO NOT EDIT

% --- Executes just before DICOMFiles is made visible.
function ControlPET_OpeningFcn(hObject, eventdata,
handles, varargin)

% Choose default command line output for DICOMFiles
handles.output = hObject;
handles.hidden = [];
guidata(hObject, handles);
colormap gray(256)

%zoom and opengl does not work well with some video card
openglinfo = opengl('data');
handles.openglinfo.Renderer = openglinfo.Renderer;
handles.openglinfo.UseGenericOpenGL = 0;
handles.openglinfo.miafigure_useOpenGL = 0;

if strcmp(openglinfo.Renderer,'GeForce4 Ti 4200 with
AGP8X/AGP/SSE2')| ...
    strcmp(openglinfo.Renderer,'Savage4')
    handles.openglinfo.UseGenericOpenGL = 1;
    feature('UseGenericOpenGL',1); % turn off the hardware
version of opengl
else
    handles.openglinfo.UseGenericOpenGL = 0;
end

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

handles.ColorRes = 128; % used color resolution in color
maps
handles.ROINumOfColor = 8; % number of color used for
ROI coloring
handles.ROIColorStrings = ['y','m','c','r','g','b','w','k']; % color
codes for ROI colors
```

```
handles.ROIColorStringsLong = cellstr(['yellow
';magenta';cyan ';red '; ...
';green ';blue ';white ';black ']); % long names for ROI
colors
```

```
% initiate variables for image handlers of resliced maps
handles.haxial = []; handles.hcoronal= []; handles.hsagital= [];
handles.haxial2 = []; handles.hcoronal2= [];
handles.hsagital2= [];
```

```
% initiate variables for image handlers of sliceomatic_mia and
improfile tools
```

```
handles.sliceomatic_haxmain = [];
handles.hf_improfile = 0;
```

```
% use push-button callback
```

```
if length(varargin) & ischar(varargin{1})
```

```
handles.dfolder = varargin{1};
```

```
SetFolder(handles);
```

```
Listbox_Callback(hObject, eventdata, handles);
```

```
else
```

```
newFolder_Callback(hObject, eventdata, handles)
```

```
end
```

```
% UIWAIT makes DICOMFiles wait for user response (see
UIRESUME)
```

```
% uiwait(handles.figure1);
```

```
% Código para que aparezcan las imagenes en la ventana
```

```
A=imread('foto2.jpg');
```

```
Img=image(A,'Parent',handles.logo);
```

```
set(handles.logo,'Visible','off','YDir','reverse','XLim',get(Img,'
XData'),'Ylim',get(Img,'YData'));
```

```
% --- Outputs from this function are returned to the command
line.
```

```
function varargout = ControlPET_OutputFcn(hObject,
eventdata, handles)
```

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

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

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

```
P = fileparts(mfilename('fullpath'));
```

```
nfolder=uigetdir(P,'Select DICOM Directory');
```

```
if ~ischar(nfolder)
```

```
disp('no valid Directory selected.')
```

```
return;
```

```
end
```

```
handles.dfolder=nfolder;
```

```
guidata(hObject, handles);
```

```
SetFolder(handles);
```

```
Listbox_Callback(hObject, eventdata, handles)
```

```
%%%%%%%%%%%%%%
```

```
function SetFolder (handles)
```

```
%%%%%%%%%%%%%%
```

```
dfiles=dir(handles.dfolder);
```

```
dfiles=dfiles(3:end); % avoid . and ..
```

```
nfiles=length(dfiles);
```

```
if nfiles<1
```

```
disp('no files available.')
```

```
return;
```

```
end
```

```
set(handles.ListBox,'String',char(dfiles.name),'value',1);
```

```
s = [num2str(nfiles) ' files in: ' handles.dfolder];
```

```
set(handles.NoffFiles,'String', s);
```

```
guidata(handles.figure1, handles);
```

```
% --- Executes during object creation, after setting all
properties.
```

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

```
if ispc
```

```
set(hObject,'BackgroundColor','white');
```

```
else
```

```
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgro
undColor'));
```

```
end
```

```
% --- Executes on selection change in Listbox.
```

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

```
fname = get(handles.ListBox,'String');
```

```
fname = fname(get(handles.ListBox,'value'),:);
```

```
try
```

```
metadata = dicominfo([handles.dfolder '\ ' fname]);
```

```
catch
```

```
disp ('apparently not a DICOM file');
```

```
return
```

```
end
```

```
img = dicomread([handles.dfolder '\ ' fname]);
```

```
imagesc(img);
```

```
axis off
```

```
ch = get(handles.HeaderList, 'value');
```

```
fields=char(fieldnames(metadata));
```

```
len = setdiff (1:size(fields,1), handles.hidden);
```

```
id=0;
```

```
for k=len,
```

```
estr=eval(['metadata.' fields(k,:)]);
```

```
if ischar(estr)
```

```
str=[fields(k,:) ' : ' estr];
```

```
elseif isnumeric(estr)
```

```
str=[fields(k,:) ' : ' num2str(estr(1:min(3,end)))];
```

```
else
```

```
str=[fields(k,:) ' : ...'];
```

```
end
```

```
id = id+1;
```

```
cstr{id}=sprintf('%3d %s',k,str);
```

```
end
```

```
set(handles.HeaderList,'Value',ch);
```

```
set(handles.HeaderList,'String',cstr);
```

```
guidata(hObject, handles);
```

```
return;
```

```
% --- Executes during object creation, after setting all
properties.
```

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

```
if ispc
```

```
set(hObject,'BackgroundColor','white');
```

```
else
```

```
set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgro
undColor'));
```

```
end
```

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

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

```
st = get(handles.HeaderList, 'string');
```

```
hide = get(handles.HeaderList, 'value');
```

```
if length(hide)~=length(st)
```

```
disp ('WARNING: at least one field must be shown');
```

```
return
```

```
end
```

```
hidev=[];
```

```
for id=hide
```

```
hidev = [hidev str2num(st{id}(1:3))];
```

```
end
```

```
handles.hidden = union(handles.hidden, hidev);
```

```
set(handles.HeaderList,'Value',1);
```

```
guidata(hObject, handles);
```

```
Listbox_Callback(hObject, eventdata, handles);
```

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

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

```
handles.hidden = [];
```

```
guidata(hObject, handles);
```

```

ListBox_Callback(hObject,eventdata,handles);
%imcontrast(gca)

% --- Executes on selection change in HeaderList.
function HeaderList_Callback(hObject,eventdata,handles)
% hObject handle to HeaderList (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 HeaderList
contents as cell array
% contents{get(hObject,'Value')} returns selected item
from HeaderList

% --- Executes on mouse press over axes background.
function Imagen_ButtonDownFcn(hObject,eventdata,
handles)
% hObject handle to Imagen (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject,eventdata,handles)
% hObject handle to pushbutton4 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)
%handles.hidden = [];
%guidata(hObject,handles);
%ListBox_Callback(hObject,eventdata,handles);
y=improfile;
%h=imdistline
x=[1:size(y)];
%plot(y)

plot(x,y,'-rs','LineWidth',2,...
'MarkerEdgeColor','k',...
'MarkerFaceColor','g',...
'MarkerSize',5)
title('Perfil de la fuente puntual')
xlabel('Distance along profile');
ylabel('Count');
grid;
imdistline
saveas(gcf,'filename.pdf')
guidata(hObject,handles);

%[pks,locs] = findpeaks(y)
%Distancia = locs(2) - locs(1)
%Para la mitad de la altura
handles.fhwm=fwhm(x,y)
set(handles.text1,'String',handles.fhwm)

% Para la decima parte
handles.fhwm=fwtm(x,y)
set(handles.text27,'String',handles.fhwm)
guidata(hObject,handles);

% --- If Enable == 'on', executes on mouse press in 5 pixel
border.
% --- Otherwise, executes on mouse press in 5 pixel border or
over text20.
function text20_ButtonDownFcn(hObject,eventdata,handles)
% hObject handle to text20 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)

```

```

% -----
function Untitled_1_Callback(hObject,eventdata,handles)
% hObject handle to Untitled_1 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)

function edit17_Callback(hObject,eventdata,handles)
% hObject handle to edit17 (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 edit17 as text
% str2double(get(hObject,'String')) returns contents of
edit17 as a double

% --- Executes during object creation, after setting all
properties.
function edit17_CreateFcn(hObject,eventdata,handles)
% hObject handle to edit17 (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 pushbutton5.
function pushbutton5_Callback(hObject,eventdata,handles)
% hObject handle to pushbutton5 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)
x=handles.x;
y=handles.y;
fwhm(x,y)
set(handles.FWHM,'String',FWHM);

% --- Executes during object creation, after setting all
properties.
function text1_CreateFcn(hObject,eventdata,handles)
% hObject handle to text1 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles empty - handles not created until after all
CreateFcns called

function edit2_Callback(hObject,eventdata,handles)
% hObject handle to edit2 (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 edit2 as text
% str2double(get(hObject,'String')) returns contents of
edit2 as a double

% --- Executes during object creation, after setting all
properties.
function edit2_CreateFcn(hObject,eventdata,handles)
% hObject handle to edit2 (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 mouse press over axes background.
function axes6_ButtonDownFcn(hObject, eventdata, handles)
% hObject handle to axes6 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)

% --- Executes on button press in roi.
function roi_Callback(hObject, eventdata, handles)
% hObject handle to roi (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)
%roitoool('circle')
[nada,roimax]=roitoool('circle');
handles.roimax=roimax
set(handles.maxpixel,'String', roimax);

guidata(hObject, handles);

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

% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton7 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)
winopen(strcat(EVP.m));

% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton9 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)

% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton11 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)

function longitud_Callback(hObject, eventdata, handles)
% hObject handle to longitud (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 longitud as
text
% str2double(get(hObject,'String')) returns contents of
longitud as a double
longitud = str2double(get(hObject,'String'));
if isnan(longitud)
    set(hObject,'String', 0);
    errordlg('Input must be a number','Error');
end

% Save the new density value
handles.longitud = longitud;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function longitud_CreateFcn(hObject, eventdata, handles)
% hObject handle to longitud (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 pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton12 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)
y=improfile;
%h=imdistline
x=[1:size(y)];
plot(y)
title('Perfil de las fuentes puntuales')
xlabel('Distance along profile');
ylabel('Count');
grid;
imdistline

%saveas(h,'filename.ext')
%[filename, user_canceled] = imsave(y)

[pks,locs] = findpeaks(y)
data={'amplitudes','localizacion';pks(1) locs(1);pks(2)
locs(2);pks(3) locs(3)}
xlswrite('tempdata_03.xls', data, 'pixeles_4', 'E1');

Distancia = locs(2) - locs(1)
handles.Distancia = Distancia;
set(handles.text28,'String',handles.Distancia)
guidata(hObject, handles);

function edit21_Callback(hObject, eventdata, handles)
% hObject handle to edit21 (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 edit21 as text
% str2double(get(hObject,'String')) returns contents of
edit21 as a double

```

```

% --- Executes during object creation, after setting all
properties.
function edit21_CreateFcn(hObject, eventdata, handles)
% hObject handle to edit21 (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 pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton14 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)
%A=longitud./Distancia;
Valorpixel = handles.longitud./handles.Distancia;
handles.Valorpixel = Valorpixel
set(handles.text31,'String',handles.Valorpixel)

% --- If Enable == 'on', executes on mouse press in 5 pixel
border.
% --- Otherwise, executes on mouse press in 5 pixel border or
over newFolder.
function newFolder_ButtonDownFcn(hObject, eventdata,
handles)
% hObject handle to newFolder (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)

% --- Executes on button press in pushbutton15.
%function pushbutton15_Callback(hObject, eventdata,
handles)
% hObject handle to pushbutton15 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)

% --- If Enable == 'on', executes on mouse press in 5 pixel
border.
% --- Otherwise, executes on mouse press in 5 pixel border or
over pushbutton15.

```

```

%function pushbutton15_ButtonDownFcn(hObject, eventdata,
handles)
% hObject handle to pushbutton15 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles structure with handles and user data (see
GUIDATA)

```

```

% --- Executes on key press with focus on pushbutton15 and
none of its controls.
%function pushbutton15_KeyPressFcn(hObject, eventdata,
handles)
% hObject handle to pushbutton15 (see GCBO)
% eventdata structure with the following fields (see
UICONTROL)
% Key: name of the key that was pressed, in lower case
% Character: character interpretation of the key(s) that was
pressed
% Modifier: name(s) of the modifier key(s) (i.e., control,
shift) pressed
% handles structure with handles and user data (see
GUIDATA)

```

```

% --- Executes on slider movement.
function slider1_Callback(hObject, eventdata, handles)
% hObject handle to slider1 (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,'Value') returns position of slider
% get(hObject,'Min') and get(hObject,'Max') to determine
range of slider
a=0.02*get(hObject,'Value');

```

```

if handles.direccion==0
return
else

```

```

%b=rgb2gray(imread(handles.direccion));
b=(dicomread(handles.direccion));
c=double(b); d=a*c; imshow(d)
end

```

```

% --- Executes during object creation, after setting all
properties.
function slider1_CreateFcn(hObject, eventdata, handles)
% hObject handle to slider1 (see GCBO)
% eventdata reserved - to be defined in a future version of
MATLAB
% handles empty - handles not created until after all
CreateFcns called

```

```

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor',[.9 .9 .9]);
end

```

fwfm

```

function width = fwfm(x,y)

% function width = fwfm(x,y)
%
```

```

% Full-Width at Half-Maximum (FWHM) of the waveform
y(x)
% and its polarity.
% The FWHM result in 'width' will be in units of 'x'
% Rev 1.2, April 2006 (Patrick Egan)

```

```

y = y / max(y);
N = length(y);
lev50 = 0.5;
if y(1) < lev50 % find index of center (max or min)
of pulse
[garbage,centerindex]=max(y);
Pol = +1;
disp('Pulse Polarity = Positive')
else
[garbage,centerindex]=min(y);
Pol = -1;
disp('Pulse Polarity = Negative')
end
i = 2;
while sign(y(i)-lev50) == sign(y(i-1)-lev50)
i = i+1;
end %first crossing is between v(i-1) &
v(i)

```

```

interp = (lev50-y(i-1)) / (y(i)-y(i-1));
tlead = x(i-1) + interp*(x(i)-x(i-1));
i = centerindex+1; %start search for next crossing
at center
while ((sign(y(i)-lev50) == sign(y(i-1)-lev50)) & (i <= N-1))
i = i+1;
end
if i ~= N
Ptype = 1;
disp('Pulse is Impulse or Rectangular with 2 edges')
interp = (lev50-y(i-1)) / (y(i)-y(i-1));
ttrail = x(i-1) + interp*(x(i)-x(i-1));
width = ttrail - tlead;
else
Ptype = 2;
disp('Step-Like Pulse, no second edge')
ttrail = NaN;
width = NaN;
end

```

fwtm

```

function width = fwhm(x,y)

% function width = fwhm(x,y)
%
% Full-Width at Half-Maximum (FWHM) of the waveform
y(x)
% and its polarity.
% The FWHM result in 'width' will be in units of 'x'
% Rev 1.2, April 2006 (Patrick Egan)

y = y / max(y);
N = length(y);
lev50 = 0.1;
if y(1) < lev50 % find index of center (max or min)
of pulse
[garbage,centerindex]=max(y);
Pol = +1;
disp('Pulse Polarity = Positive')
else
[garbage,centerindex]=min(y);
Pol = -1;
disp('Pulse Polarity = Negative')
end

```

```

i = 2;
while sign(y(i)-lev50) == sign(y(i-1)-lev50)
i = i+1;
end %first crossing is between v(i-1) &
v(i)
interp = (lev50-y(i-1)) / (y(i)-y(i-1));
tlead = x(i-1) + interp*(x(i)-x(i-1));
i = centerindex+1; %start search for next crossing
at center
while ((sign(y(i)-lev50) == sign(y(i-1)-lev50)) & (i <= N-1))
i = i+1;
end
if i ~= N
Ptype = 1;
disp('Pulse is Impulse or Rectangular with 2 edges')
interp = (lev50-y(i-1)) / (y(i)-y(i-1));
ttrail = x(i-1) + interp*(x(i)-x(i-1));
width = ttrail - tlead;
else
Ptype = 2;
disp('Step-Like Pulse, no second edge')
ttrail = NaN;
width = NaN;
end

```

draggable

```

function draggable(h,varargin)
% DRAGGABLE - Make it so that a graphics object can be
dragged in a figure.
% This function makes an object interactive by allowing it to
be dragged
% across a set of axes, following or not certain constraints.
This
% allows for intuitive control elements which are not buttons
or other
% standard GUI objects, and which reside inside an axis.
Typical use
% involve markers on an axis, whose position alters the
output of a
% computation or display
%
% >> draggable(h);
%
% makes the object with handle "h" draggable. Use the
"Position" property
% of the object to retrieve its position, by issuing a
get(h,'Position')

```

```

% command.
%
% >> draggable(h,...,motionfcn)
%
% where "motionfcn" is a function handle, executes the given
function
% while the object is dragged. Handle h is passed to
motionfcn as an
% argument. Argument "motionfcn" can be put anywhere
after handle "h".
%
% >> draggable(h,...,constraint,p);
%
% enables the object with handle "h" to be dragged, with a
constraint.
% Arguments "constraint" (a string) and "p" (a vector) can be
put
% anywhere after handle "h".
%
% The argument "constraint" may be one of the following
strings:

```

```

%
% 'n' or 'none': The object is unconstrained (default).
% 'h' or 'horizontal': The object can only be moved
% horizontally.
% 'v' or 'vertical': The object can only be moved
% vertically.
%
% The argument "p" is an optional parameter which depends
% upon the
% constraint type:
%
% Constraint p Description
% -----
% 'none' [x1 x2 y1 y2] Drag range (for the object's
% outer
% limits, from x1 to x2 on the x-axis
% and from y1 to y2 on the y-axis).
% Default is the current axes range.
% Use "inf" if no limit is desired.
%
% 'horizontal' [xmin xmax] Drag range (for the
% object's outer
% limits). Default is the x-axis
% range. Use "inf" if no limit is
% desired.
%
% 'vertical' [ymin ymax] Drag range (for the object's
% outer
% limits). Default is the y-axis
% range. Use "inf" if no limit is
% desired.
% -----
% >> draggable(h,...,renderer);
%
% where renderer is one of 'painters', 'zbuffer' or 'opengl',
% uses the
% corresponding renderer for the figure while the graphical
% object whose
% handle is h is being dragged. By default, zbuffer is used
% since it is
% the only renderer that offers both acceptable performance
% and a
% guaranteed correct behavior with draggable. The 'painters'
% renderer is
% too slow, while the 'opengl' renderer's behavior with
% draggable depends
% on the graphics driver used and may differ from what is
% expected.
%
% >> draggable(h,'off')
%
% returns object h to its original, non-draggable state.
%
% See the source code (e.g. by issuing "type draggable" at
% the Matlab
% prompt) for implementation notes and the copyright
% notice.

```

% VERSION INFORMATION:

```

% 2003-11-20: Initially submitted to MatlabCentral.Com
% 2004-01-06: Addition of the renderer option, as proposed
% by Ohad Gal
% as a feedback on MatlabCentral.Com.
% 2004-02-18: Bugfix: now works with 1-element plots and
% line objects
% 2004-03-04: Bugfix: sanitized the way the object's new
% position is
% computed; it now always follow the mouse even
% after the
% mouse pointer was out of the axes.
% 2004-03-05: Bugfix: movement when mouse is out of the
% axes is now
% definitely correct ;)

```

% IMPLEMENTATION NOTES:

```

%
% This function uses the dragged object's "ButtonDownFcn"
% function and set it
% so that the object becomes draggable. Any previous
% "ButtonDownFcn" is thus
% lost during operation, but is retrieved after issuing the
% draggable(h,'off')
% command.
%
% Information about the object's behavior is also stored in the
% object's
% 'UserData' property, using setappdata() and getappdata().
% The original
% 'UserData' property is restored after issuing the
% draggable(h,'off')
% command.
%
% The corresponding figure's "WindowButtonDownFcn",
% "WindowButtonUpFcn" and
% "WindowButtonMotionFcn" functions. During operation,
% those functions are
% set by DRAGGABLE; however, the original ones are
% restored after the user
% stops dragging the object.
%
% By default, DRAGGABLE also switches the figure's
% renderer to 'zbuffer'
% during operation: 'painters' is not fast enough and 'opengl'
% sometimes
% produce curious results. However there may be a need to
% switch to another
% renderer, so the user can now specify a specific figure
% renderer during
% object drag (thanks to Ohad Gal for the suggestion).
%
% The "motionfcn" function handle is called at each
% displacement, after the
% object's position is updated, using "feval(motionfcn,h)",
% where h is the
% object's handle.
%
% TO DO:
%
% 1 - For now, DRAGGABLE allows only one object at a
% time to be draggable. In
% the future, draggable(h), where h is a vector of handles, will
% create a
% group of objects that will all be dragged when one of them
% is selected.
%
% =====
% Copyright (C) 2003, 2004
% Francois Bouffard
% fbouffar@gel.ulaval.ca
% Université Laval, Québec City
% =====
%
% Input arguments management
% =====

```

% Initialization of some default arguments

```

user_renderer = 'zbuffer';
user_movefcn = [];
constraint = 'none';
p = [];

% At least the handle to the object must be given
Narg = nargin;
if Narg == 0
    error('Not enough input arguments');
elseif prod(size(h))>1
    error('Only one object at a time can be made draggable');
end;

```

% Fetching informations about the parent axes

```

axh = get(h,'Parent');
if iscell(axh)

```

```

    axh = axh{1};
end;
fgh = get(axh,'Parent');
ax_xlim = get(axh,'XLim');
ax_ylim = get(axh,'YLim');

% Assigning optional arguments
Noptarg = Narg - 1;
for k = 1:Noptarg
    current_arg = varargin{k};
    if isa(current_arg,'function_handle');
        user_movefcn = current_arg;
    end;
    if ischar(current_arg);
        switch lower(current_arg)
            case {'off'}
                set_initial_state(h);
                return;
            case {'painters','zbuffer','opengl'}
                user_renderer = current_arg;
            otherwise
                constraint = current_arg;
        end;
    end;
    if isnumeric(current_arg);
        p = current_arg;
    end;
end;

% Assigning defaults for constraint
switch lower(constraint)
    case {'n','none'}
        if isempty(p); p = [ax_xlim ax_ylim]; end;
    case {'h','horizontal'}
        if isempty(p); p = ax_xlim; end;
    case {'v','vertical'}
        if isempty(p); p = ax_ylim; end;
    otherwise
        error('Unknown constraint type');
end;

% =====
% Saving initial state and parameters, setting up the object
callback
% =====

% Saving object's and parent figure's initial state
setappdata(h,'initial_userdata',get(h,'UserData'));
setappdata(h,'initial_objbdfcn',get(h,'ButtonDownFcn'));
setappdata(h,'initial_renderer',get(fgh,'Renderer'));
setappdata(h,'initial_wbdfcn',get(fgh,'WindowButtonDownFcn'));
setappdata(h,'initial_wbufcn',get(fgh,'WindowButtonUpFcn'));
setappdata(h,'initial_wbmfcn',get(fgh,'WindowButtonMotionFcn'));

% Saving parameters
setappdata(h,'constraint_type',constraint);
setappdata(h,'constraint_parameters',p);
setappdata(h,'user_movefcn',user_movefcn);
setappdata(h,'user_renderer',user_renderer);

% Detecting if object's position is specified through the
% 'Position' or 'XData' and 'YData' properties
h_properties = get(h);
if isfield(h_properties,'Position')
    setappdata(h,'position_type','rect');
else
    setappdata(h,'position_type','xydata');
end;

% Setting the object's ButtonDownFcn
set(h,'ButtonDownFcn',@click_object);

% =====
% FUNCTION click_object

```

```

% Executed when the object is clicked
% =====

function click_object(obj,eventdata);
% obj here is the object to be dragged and gcf is the object's
parent
% figure since the user clicked on the object
h = obj;
position_type = getappdata(h,'position_type');
if strcmp(position_type,'xydata')
    setappdata(h,'initial_xdata',get(h,'XData'));
    setappdata(h,'initial_ydata',get(h,'YData'));
else
    setappdata(h,'initial_position',get(h,'Position'));
end;
setappdata(h,'initial_point',get(gcf,'CurrentPoint'));
set(gcf,'WindowButtonDownFcn',{@activate_movefcn,h});
set(gcf,'WindowButtonUpFcn',{@deactivate_movefcn,h});
activate_movefcn(gcf,eventdata,h);

% set the pointer shape to close hand /LB UDEB PET Center
setappdata(h,'current_figure_pointer',get(gcf,'pointer'));
setptr(gcf,'closedhand');

% =====
% FUNCTION activate_movefcn
% Activates the WindowButtonMotionFcn for the figure
% =====

function activate_movefcn(obj,eventdata,h);
% obj here is the figure containing the object
set(obj,'Renderer',getappdata(h,'user_renderer'));
set(obj,'WindowButtonMotionFcn',{@movefcn,h});

% =====
% FUNCTION deactivate_movefcn
% Deactivates the WindowButtonMotionFcn for the figure
% =====

function deactivate_movefcn(obj,eventdata,h);
% obj here is the figure containing the object

% set the pointer shape to the original /LB UDEB PET Center
setptr(gcf,getappdata(h,'current_figure_pointer'));

set(obj,'WindowButtonMotionFcn',getappdata(h,'initial_wbmfcn'));
set(obj,'WindowButtonDownFcn',getappdata(h,'initial_wbdfcn'));
set(obj,'WindowButtonUpFcn',getappdata(h,'initial_wbufcn'));
set(obj,'Renderer',getappdata(h,'initial_renderer'));

% =====
% FUNCTION set_initial_state
% Returns the object to its initial state
% =====

function set_initial_state(h);
initial_objbdfcn = getappdata(h,'initial_objbdfcn');
initial_userdata = getappdata(h,'initial_userdata');
set(h,'ButtonDownFcn',initial_objbdfcn);
set(h,'UserData',initial_userdata);

% =====
% FUNCTION movefcn
% Actual code for dragging the object
% =====

function movefcn(obj,eventdata,h);
% obj here is the figure containing the object

% Retrieving data saved in the figure
position_type = getappdata(h,'position_type');
if strcmp(position_type,'xydata')
    initial_xdata = getappdata(h,'initial_xdata');
    initial_ydata = getappdata(h,'initial_ydata');

```

```

else
    %initial_position = getappdata(h,'initial_position');
end;
initial_point = getappdata(h,'initial_point');
constraint = getappdata(h,'constraint_type');
p = getappdata(h,'constraint_parameters');
user_movefcn = getappdata(h,'user_movefcn');

% Getting current point
current_point = get(gca,'CurrentPoint');

% Retrieving (x,y) couple for current and initial points
cpt = current_point(1,1:2);
ipt = initial_point(1,1:2);

% Computing movement
dpt = cpt - ipt;

% Computing movement range and imposing movement constraints
% (p is always [xmin xmax ymin ymax])
switch lower(constraint)
case {'n','none'};
    range = p;
case {'h','horizontal'}
    dpt(2) = 0;
    range = [p -inf inf];
case {'v','vertical'}
    dpt(1) = 0;
    range = [-inf inf p];
end;

% Computing new position.
% What we want is actually a bit complex: we want the object
to adopt the new
% position, unless it gets out of range. If it gets out of range in
a direction,
% we want it to stick to the limit in that direction. Also, if the
object is out
% of range at the beginning of the movement, we want to be
able to move it back
% into range, so that movement must then be allowed.

switch lower(position_type)

% Objects with rectangle-type position information
case 'rect'

    % Retrieving various quantities:
    % initial_position is the position when the object was
clicked
    % oldpos is the current position, which will be updated
    % newpos is the new (proposed) position
    % old_obj_extent and new_pos extent are the extent
occupied by
    % the old and new objects
    initial_position = getappdata(h,'initial_position');
    oldpos = get(h,'Position');
    old_obj_extent = [oldpos(1) oldpos(1)+oldpos(3)
oldpos(2) oldpos(2)+oldpos(4)];
    newpos = initial_position + [dpt 0 0];
    new_obj_extent = [newpos(1) newpos(1)+newpos(3)
newpos(2) newpos(2)+newpos(4)];

    % Verifying if old and new objects breach the allowed
range in any
    % direction (see the function is_inside_range below)
    old_inrange = is_inside_range(old_obj_extent,range);
    new_inrange = is_inside_range(new_obj_extent,range);

    % Modifying dpt to stick to range limit if range violation
occured,
    % but the movement won't get restricted if the object was
out of range
    % to begin with
    if old_inrange(1) & ~new_inrange(1)
        dpt(1) = range(1) - min(initial_xdata);
    end
    if old_inrange(2) & ~new_inrange(2)
        dpt(1) = range(2) - max(initial_xdata);
    end;
    if old_inrange(3) & ~new_inrange(3)
        dpt(2) = range(3) - min(initial_ydata);
    end;
    if old_inrange(4) & ~new_inrange(4)
        dpt(2) = range(4) - max(initial_ydata);
    end

    % Computing the final new position and setting it
    newpos = initial_position + dpt*ones(1,size(oldpos,2));
    set(h,'XData',newpos(1,:),'YData',newpos(2,:));

end;

% Calling user-provided function handle
if ~isempty(user_movefcn)
    feval(user_movefcn,h);
end;

```

```

dpt(1) = range(1) - initial_position(1);
end
if old_inrange(2) & ~new_inrange(2)
    dpt(1) = range(2) - initial_position(1) -
initial_position(3);
end;
if old_inrange(3) & ~new_inrange(3)
    dpt(2) = range(3) - initial_position(2);
end;
if old_inrange(4) & ~new_inrange(4)
    dpt(2) = range(4) - initial_position(2) -
initial_position(4);
end

% Computing the final new position and setting it
newpos = initial_position + [dpt 0 0];
set(h,'Position',newpos);

% Objects with line-type position information
case 'xydata'

    % Retrieving various quantities:
    % initial_position is the position when the object was
clicked
    % oldpos is the current position, which will be updated
    % newpos is the new (proposed) position
    % old_obj_extent and new_pos extent are the extent
occupied by
    % the old and new objects
    initial_xdata = getappdata(h,'initial_xdata');
    initial_ydata = getappdata(h,'initial_ydata');
    initial_position = [initial_xdata(:) ; initial_ydata(:)];
    xdata = get(h,'XData');
    ydata = get(h,'YData');
    oldpos = [xdata(:) ; ydata(:)];
    old_obj_extent = [min(xdata) max(xdata) min(ydata)
max(ydata)];
    newpos = initial_position + dpt*ones(1,size(oldpos,2));
    new_obj_extent = [min(newpos(1,:)) max(newpos(1,:))
min(newpos(2,:)) max(newpos(2,:))];

    % Verifying if old and new objects breach the allowed
range in any
    % direction (see the function is_inside_range below)
    old_inrange = is_inside_range(old_obj_extent,range);
    new_inrange = is_inside_range(new_obj_extent,range);

    % Modifying dpt to stick to range limit if range violation
occured,
    % but the movement won't get restricted if the object was
out of range
    % to begin with
    if old_inrange(1) & ~new_inrange(1)
        dpt(1) = range(1) - min(initial_xdata);
    end
    if old_inrange(2) & ~new_inrange(2)
        dpt(1) = range(2) - max(initial_xdata);
    end;
    if old_inrange(3) & ~new_inrange(3)
        dpt(2) = range(3) - min(initial_ydata);
    end;
    if old_inrange(4) & ~new_inrange(4)
        dpt(2) = range(4) - max(initial_ydata);
    end

    % Computing the final new position and setting it
    newpos = initial_position + dpt*ones(1,size(oldpos,2));
    set(h,'XData',newpos(1,:),'YData',newpos(2,:));

end;

% Calling user-provided function handle
if ~isempty(user_movefcn)
    feval(user_movefcn,h);
end;

```

```

%
%
% FUNCTION is_inside_range
% Checks if a rectangular object is entirely inside a
rectangular range

function [roi_handler,roimax] = roitool(arg1, arg2, arg3)
%global roimax
%handles.roimax=roimax
% ROITool enables to define different type of rois on
intensity images
% by mouse. The supported types are 'polygon', 'circle' and
'rectangle'
% using the ROITool('circle'), ROITool('rectangle') and
ROITool('polygon'),
% respectively. ROIPOLY returns the handler of the line
object relating
% to the roi. The created roi can be dragged and moved to any
position of
% the image with the help of DRAGGABLE function
(developed by Francois
% Bouffard, see the help of draggable).
%
% Definition rules (displayed always in the Matlab command
window):
% circle: Select the center of the circle by normal button
click then
% define the desired radius moving the mouse and
finally
% double click to finish the ROI.
% rectangle: Use the mouse to click and drag the desired
rectangle. To
% constrain the rectangle to be a square, use a shift- or
% right-click to begin the drag.
% polygon: Use normal button clicks to add vertices to the
polygon and
% right-click close the vertex.
%
% ROITool generates a context menu to the roi making
possible the
% selection of the following commands:(MATLAB displays
the context
% menu whenever you right-click over the object)
% - change the roi color
% - change the size of the line width
% - switch on/off the "statistical bar", which displays
% the main stat. results of the roi (mean, std, min, max,
roi area)
% - switch on/off the "histogram figure", which shows the
intensity
% histogram of the pixels belonging to the roi
% - switch on/off the "detailed rectangle" figure, which
plots the
% zoomed image segment defined by the rectangle roi
% - resize the roi (in case of circle or rectangle roi)
% - delete the roi
%
% roihandler = ROITool('roitype') command returns the
handler of the
% roi (the line object). The handler has an associate data
structure
% containing the parameters of the roi. This structure can be
evaluated
% issuing the
% roidata = get(roihandler,'userdata');
% command. The roidata structure includes the following
fields (among
% others) - roimean, roistd, roimax, roimin, pixelrange (image
pixel
% indexes relating to the roi), roiarea, handlers of the joint

```

roitool

```

%
%
function inrange = is_inside_range(extent,range)
inrange = [extent(1)>=range(1) extent(2)<=range(2) ...
extent(3)>=range(3) extent(4)<=range(4)];

% object (histogram figure, stat. bar, ...).
%
% ROITool('circledemo'), ROITool('rectangledemo') or
ROITool('polygondemo')
% start the roi definition using the MATLAB 'MRI' data set.

if nargin == 0
    roitype = 'circle';
else
    roitype = arg1;
end

if ischar(roitype) % execute a roi-definition callback
    % test the roitype input
    if ~strcmp(roitype,'circle') & ~strcmp(roitype,'rectangle') &
~strcmp(roitype,'polygon') & ...
~strcmp(roitype,'circledemo') &
~strcmp(roitype,'rectangledemo') &
~strcmp(roitype,'polygondemo')
        error('RoiError:inputtest','Roitype ',upper(roitype),' is not
supported. \n ', ...
'Only CIRCLE, RECTANGLE and POLYGON type is
currently enabled. ');
        return;
    end
    % test whether the roitype contains a 'DEMO' string
    if ~isempty(strfind(roitype,'demo'))
        mridata = load('mri');
        imaVOL = squeeze(mridata.D);
        figure('name','MRI data for ROITool
test','numbertitle','off');
        imagesc(imaVOL(:,:,17));
        map = colormap(bone(256));
        axis square;
    end
    % test the current figure existence
    currentfigure_handle = get(0,'CurrentFigure');
    if isempty(currentfigure_handle)
        msg = sprintf('%s expects a current figure containing an
image.', ...
upper(mfilename));
        error(msg);
        return;
    end
    % test the imagetype. Only intensity image is supported.
    imModel =
getimagemodel(imhandles(currentfigure_handle));
    if (~(isequal(getImageType(imModel),'intensity') || ...
isequal(getImageType(imModel),'truecolor'))
        error('Only intensity and truecolor images are supported. ');
        return;
    end
elseif ishandle(roitype) % execute a context menu defined
callback
    %test the roi_handler
    roi_handler = roitype;
    if ~strcmp(get(roi_handler,'tag'),'roi_handler');
        msg = sprintf('%s expects arg1 only as real roi_handler.',
...
upper(mfilename));
        error(msg);
        return;
    end
    % if the roi_handle OK

```

```

switch lower(arg2)
case 'roi_delete'
    roi_delete(roi_handler);
case 'roi_resize'
    roi_resize(roi_handler);
case 'roi_drag'
    roi_drag(roi_handler);
case 'roi_showstat'
    onoff = arg3;
    roi_showstat(roi_handler, onoff);
case 'roi_showhist'
    onoff = arg3;
    roi_showhist(roi_handler, onoff);
case 'roi_showdetrect'
    onoff = arg3;
    roi_showdetrect(roi_handler, onoff);
end
return;
end
%
% start the ROI definition
%
set(gca,'units','pixels');
image_h = findobj(gca,'type','image');
userdata.currentimage = get(image_h,'cdata');
mainfigure = gcf;
if ~isempty(strfind(roitype,'circle'))
    roitype = 'circle';
    disp(' '); disp('Select the center of the circle!');
    drawnow;
    [xcenter, ycenter] = ginput(1);
    x0 = xcenter; y0 = ycenter; R0 = 5;
    t = 0:pi/20:2*pi;
    xi = R0*cos(t)+xcenter;
    yi = R0*sin(t)+ycenter;
    roi_handler = line(xi,yi,'LineWidth',2,'Color','red');
    userdata.t = t; userdata.xcenter = xcenter; userdata.ycenter =
ycenter; userdata.R0 = R0;
    set(roi_handler,'userdata',userdata);
    % set the roisize
    disp(' '); disp('Define the desired radius moving the
mouse. ');
    disp('Double click to finalize the ROI!');disp(' ');

    set(gcf,'WindowButtonMotionFcn',[ 'change_radius(' num2str(r
oi_handler,20),')']);
    set(gcf,'WindowButtonDownFcn','complete_radius');
    waitfor(gcf,'WindowButtonMotionFcn');
    drawnow;
elseif ~isempty(strfind(roitype,'polygon'))
    roitype = 'polygon';
    disp('Use the mouse to select the vertices of polygon. ');
    disp('Double-click adds a final vertex and draws the
roi');disp(' ');
    roi_x=[]; roi_y=[];
    ButtonPressed = 1; i = 1;
    drawnow; pause(1);
    while ButtonPressed == 1
        [xtmp, ytmp, ButtonPressed] = ginput(1);
        roi_x = [roi_x; xtmp];
        roi_y = [roi_y; ytmp];
        line_h(i) = line(roi_x,roi_y,'color','red','LineWidth',2);
        i = i + 1;
    end;
    % delete the temporary lines and create the final polyroi
    delete(line_h);
    roi_x(length(roi_x)+1)=roi_x(1);
    roi_y(length(roi_y)+1)=roi_y(1);
    roi_handler = line(roi_x,roi_y,'color','red','linewidth',2);
elseif ~isempty(strfind(roitype,'rectangle'))
    roitype = 'rectangle';
    disp(' '); disp('Use the mouse to click and DRAG the desired
rectangle. ');disp(' ');
    rectpos = [0 0 0 0];
    while rectpos(3) == 0;
        rectpos = round(getrect(gca));

        if rectpos(3) == 0;
            mbh = msgbox('Use the mouse to click and DRAG the
desired rectangle.','Roi Info');
            uiwait(mbh);
        end
        end
        % define the coordinate range and data for detailed
rectangle
        dry = round(rectpos(1)):round(rectpos(1)) +
round(rectpos(3)-1);
        drx = round(rectpos(2)):round(rectpos(2)) +
round(rectpos(4)-1);
        cimg = userdata.currentimage(drx,dry);
        userdata.cimgsize = size(cimg);
        userdata.cmap = colormap;
        ScreenSize = get(0,'ScreenSize');
        Pos = [0.6*ScreenSize(3) 0.1*ScreenSize(4)
0.4*ScreenSize(3) 0.35*ScreenSize(4)];
        DetailRectangleFig = figure('name','Detail Rectangle', ...
'NumberTitle','off','menubar','none','Position',Pos);
        colormap(userdata.cmap);
        set(DetailRectangleFig,'doubleBuffer','on');
        userdata.DetailRectangleFig = DetailRectangleFig;
        imh = imagesc(cimg);
        axis image; set(gca,'tag','DetailRectangleAx');
        set(imh,'tag','DetailRectangleImg');
        set(imh,'EraseMode','none');
        axis off;
        set(DetailRectangleFig,'visible','off');
        figure(mainfigure);
        % define the points for the line to be drawn
        roi_x = [rectpos(1), rectpos(1)+rectpos(3),
rectpos(1)+rectpos(3), ...
rectpos(1), rectpos(1)];
        roi_y = [rectpos(2), rectpos(2), rectpos(2)+rectpos(4), ...
rectpos(2)+rectpos(4), rectpos(2)];
        roi_handler = line(roi_x,roi_y,'color','red','linewidth',2);
    end
    userdata.roitype = roitype;
    set(roi_handler,'tag','roi_handler');
    %
    % Define the context menu
    %
    cmenu = uicontextmenu;
    menuitem1 = uimenu(cmenu, 'Label', 'Color', 'Callback', ...
[ 'set(' num2str(roi_handler,20),','color','uisetcolor')']);

    menuitem2 = uimenu(cmenu, 'Label', 'LineWidth', 'Callback',
...
[ 'set(' num2str(roi_handler,20), ...
','LineWidth', str2num(cell2mat(inputdlg('Enter the line
width','ROI setup',1,{ '3' }))))]);

    menuitem3 = uimenu(cmenu, 'Label', 'Show Statistics: +',
'Callback', ...
[roitool(' num2str(roi_handler,20),'roi_showstat','on');'],
...
'tag','ShowStatMenu');

    menuitem4 = uimenu(cmenu, 'Label', 'Show Histogram: -',
'Callback', ...
[roitool(' num2str(roi_handler,20),'roi_showhist','on');'],
...
'tag','ShowHistMenu');

    if strcmp(roitype,'rectangle')
        menuitem6 = uimenu(cmenu, 'Label', 'Show DetailedRect: -
', 'Callback', ...
[roitool(' num2str(roi_handler,20),'roi_showdetrect','on');'],
...
'tag','ShowDetRectMenu');
    end

    if ~strcmp(roitype,'polygon')
        menuitem5 = uimenu(cmenu, 'Label', 'Resize', 'Callback', ...

```

```

        ['roitool','num2str(roi_handler,20),'roi_resize');]);
end

menuitem7 = uimenu(cmenu, 'Label', 'Delete', 'Callback', ...
    ['roitool','num2str(roi_handler,20),'roi_delete']);

set(roi_handler, 'UIContextMenu', cmenu);
%
% create the StatValue display bar
%
StatFigure = findobj('name', 'Roi Stat. Figure');
if isempty(StatFigure)
    ScreenSize = get(0, 'ScreenSize');
    Pos = [0.5*ScreenSize(3) 0.8*ScreenSize(4)
        0.1*ScreenSize(3) 0.1*ScreenSize(4)];
    StatFigure = figure('name', 'Roi Stat. Figure', 'NumberTitle',
        'off', 'MenuBar', 'none', ...
        'visible', 'on', 'Position', Pos);
    visible_on = 'off';
    FigPos = get(gcf, 'position');
    % StatValuesBar0 = uicontrol(gcf, 'Style', 'text', ...
    %     'Units', 'pixels', ...
    %     'Position', [0 FigPos(4)-20 FigPos(3) 15], ...
    %     'Foreground', [1 1 .5], ...
    %     'Background', [0 0 0], ...
    %     'Horiz', 'left', ...
    %     'Tag', 'RoiStatValuesBar1', ...
    %     'String', '', ...
    %     'fontname', 'Helvetica', ...
    %     'FontSize', 8, ...
    %     'BusyAction', 'queue', ...
    %     'enable', 'inactive', ...
    %     'Interruptible', 'off');
    StatValuesBar1 = uicontrol(gcf, 'Style', 'text', ...
        'Units', 'pixels', ...
        'Position', [0 FigPos(4)-20 FigPos(3) 15], ...
        'Foreground', [1 1 .5], ...
        'Background', [0 0 0], ...
        'Horiz', 'left', ...
        'Tag', 'RoiStatValuesBar1', ...
        'String', '', ...
        'fontname', 'Helvetica', ...
        'FontSize', 8, ...
        'BusyAction', 'queue', ...
        'enable', 'inactive', ...
        'Interruptible', 'off');
    StatValuesBar2 = uicontrol(gcf, 'Style', 'text', ...
        'Units', 'pixels', ...
        'Position', [0 FigPos(4)-40 FigPos(3) 15], ...
        'Foreground', [1 1 .5], ...
        'Background', [0 0 0], ...
        'Horiz', 'left', ...
        'Tag', 'RoiStatValuesBar2', ...
        'String', '', ...
        'fontname', 'Helvetica', ...
        'FontSize', 8, ...
        'BusyAction', 'queue', ...
        'enable', 'inactive', ...
        'Interruptible', 'off');
    set(StatFigure, 'visible', 'on');
end
%userdata.StatValuesBar1 = StatValuesBar;
userdata.cmenu = cmenu;
%
% create histogram figure
%
imagesize = size(userdata.currentimage);
RoiHistogram = findobj('name', 'Roi Histogram');
if isempty(RoiHistogram)
    ScreenSize = get(0, 'ScreenSize');
    Pos = [0.6*ScreenSize(3) 0.3*ScreenSize(4)
        0.4*ScreenSize(3) 0.35*ScreenSize(4)];
    RoiHistogram = figure('name', 'Roi
Histogram', 'NumberTitle', ...
        'off', 'visible', 'on', 'Position', Pos, 'doubleBuffer', 'on');
    visible_on = 'off';

```

```

else
    visible_on = get(RoiHistogram, 'visible');
end
set(RoiHistogram, 'visible', 'on');
figure(RoiHistogram);
colors = {'r-', 'g-', 'b-'};
histnames =
{'roiHistogramR', 'roiHistogramG', 'roiHistogramB'};
hold on;
if size(imagesize) == [1 2]
    [n, xbin] = hist(double(userdata.currentimage(:)), 256);
    userdata.pixelmax = max(userdata.currentimage(:));
    hist_h = plot(xbin, n, 'k-');
    set(hist_h, 'tag', 'roiHistogram');
    for i=1:3
        hist_h = plot(0, 0, colors{i});
        set(hist_h, 'tag', histnames{i});
    end
else
    for i=1:3
        imagetmp = userdata.currentimage(:, i);
        if max(imagetmp(:)) ~= 0
            [n, xbin] = hist(double(imagetmp(:)), 256);
            hist_h = plot(xbin, n, colors{i});
            set(hist_h, 'tag', histnames{i});
        else
            hist_h = plot(0, 0, colors{i});
            set(hist_h, 'tag', histnames{i});
        end
    end
    hist_h = plot(0, 0, 'k-');
    set(hist_h, 'tag', 'roiHistogram');
end
userdata.hist_axsish = gca;
userdata.pixelmax = max(userdata.currentimage(:));
xlabel('Pixel intensity'); ylabel('Number of events');
set(RoiHistogram, 'visible', visible_on);
figure(mainfigure);
userdata.RoiHistogram = RoiHistogram;
set(roi_handler, 'userdata', userdata);
%
% calc. roi stat.
%
xi = get(roi_handler, 'Xdata'); yi = get(roi_handler, 'Ydata');
roimask = poly2mask(xi, yi, imagesize(1), imagesize(2));
pixel_range = find(roimask);
imagesize_ = size(imagesize);
if size(imagesize) == [1 2]
    % intensity image case
    roimean =
    sprintf('%0.5g', mean(userdata.currentimage(pixel_range)));
    roistd =
    sprintf('%0.5g', std(double(userdata.currentimage(pixel_range)
    )));
    roimin =
    num2str(min(userdata.currentimage(pixel_range)));
    roimax =
    num2str(max(userdata.currentimage(pixel_range)));
    handles.roimax=roimax;
else
    % truecolor (RGB) image case
    roimean = []; roistd=[]; roimin=[]; roimax=[];
    for i=1:3
        if i==1; sepchar=''; else sepchar=' '; end
        imagetmp= userdata.currentimage(:, i);
        roimean = [roimean, sepchar,
        sprintf('%0.5g', mean(imagetmp(pixel_range)))];
        roistd =
        [roistd, sepchar, sprintf('%0.5g', std(double(imagetmp(pixel_ran
        ge))))];
        roimin = [roimin, sepchar,
        num2str(min(imagetmp(pixel_range)))];
        roimax = [roimax, sepchar,
        num2str(max(imagetmp(pixel_range)))];
    end
end
end

```

```

%
% set the buttowndown event to "draggable"
%
userdata = get(roi_handler,'userdata');
userdata.roimean = roimean;
userdata.roistd = roistd;
userdata.roimin = roimin;
userdata.roimax = roimax;
userdata.roipixel_range = pixel_range;
userdata.roiarea = length(userdata.roipixel_range);
userdata.lastclickedpoint = [1, 1];
set(roi_handler,'userdata',userdata);
fun_h = @(x)roitool(x,'roi_drag');
draggable(roi_handler, fun_h);

%
% Sub-function - change_radius
% -----
function change_radius(roi_handler)

userdata = get(roi_handler,'userdata');
t = userdata.t; xcenter = userdata.xcenter; ycenter =
userdata.ycenter;

current_pts = get(gca,'CurrentPoint');
current_pt = current_pts(1,1:2);
R0 = sqrt((current_pt(1)-xcenter)^2+(current_pt(2)-
ycenter)^2);
xi = R0*cos(t)+ xcenter;
yi = R0*sin(t)+ ycenter;
userdata.R0 = R0;
set(roi_handler,'Xdata',xi,'Ydata',yi);
set(roi_handler,'userdata',userdata);
drawnow;
mouseclick = get(gcf,'SelectionType');

%
% Sub-function - complete_radius
% -----
function complete_radius
mouseclick = get(gcf,'SelectionType');
if strcmp(mouseclick,'open')
    set(gcf,'WindowButtonMotionFcn','');
    set(gcf,'WindowButtonDownFcn','');
end

%
% Sub-function - roi_delete
% -----
function roi_delete(roi_handler)

draggable(roi_handler,'off');
userdata = get(roi_handler,'userdata');
delete(userdata.cmenu);
roi_handlers = findobj('tag','roi_handler');
% if no additional rois, delete the
% StatValuesBar and the RoiHistogram also
if length(roi_handlers) == 1
    % if ishandle(userdata.StatValuesBar)
    %     delete(userdata.StatValuesBar);
    % end
    % if ishandle(userdata.RoiHistogram)
    %     delete(userdata.RoiHistogram);
    % end
    DetailRectangleFig = findobj('name','Detail Rectangle');
    if ishandle(DetailRectangleFig)
        delete(DetailRectangleFig);
    end
end
delete(roi_handler);

%
% Sub-function - roi_drag
% -----
function roi_drag(roi_handler)

```

```

userdata = get(roi_handler,'userdata');
imagesize = size(userdata.currentimage);
xi = get(roi_handler,'Xdata'); yi = get(roi_handler,'Ydata');
roimask = poly2mask(xi,yi, imagesize(1),imagesize(2));
pixel_range = find(roimask);
if size(imagesize) == [1 2]
    % intensity image case
    roimean =
    sprintf('%0.5g',mean(userdata.currentimage(pixel_range)));
    roistd =
    sprintf('%0.5g',std(double(userdata.currentimage(pixel_range)
    )));
    roimin =
    num2str(min(userdata.currentimage(pixel_range)));
    roimax =
    num2str(max(userdata.currentimage(pixel_range)));
else
    % truecolor (RGB) image case
    roimean = []; roistd=[]; roimin=[]; roimax=[];
    for i=1:3
        if i==1;sepchar='';else sepchar='';end
        imagetmp = userdata.currentimage(:,i);
        roimean = [roimean,sepchar,
        sprintf('%0.5g',mean(imagetmp(pixel_range)))];
        roistd =
        [roistd,sepchar,sprintf('%0.5g',std(double(imagetmp(pixel_ran
        ge))))];
        roimin = [roimin,sepchar,
        num2str(min(imagetmp(pixel_range)))];
        roimax = [roimax,sepchar,
        num2str(max(imagetmp(pixel_range)))];
    end
end
userdata.roimean = roimean;
userdata.roistd = roistd;
userdata.roimin = roimin;
userdata.roimax = roimax;
userdata.roipixel_range = pixel_range;
userdata.lastclickedpoint = get(0,'PointerLocation');
set(roi_handler,'userdata',userdata);

histFigure = findobj('name','Roi Histogram');
StatFigure = findobj('name','Roi Stat. Figure');
StatValuesBar1 = findobj('Tag','RoiStatValuesBar1');
StatValuesBar2 = findobj('Tag','RoiStatValuesBar2');
if strcmp(get(StatFigure,'visible'),'on')
    StatString1 = [' avg= ',userdata.roimean,' std=
    ',userdata.roistd];
    StatString2 = [' max= ',userdata.roimax,'
    min= ',userdata.roimin, ...
    ' area= ',num2str(userdata.roiarea)];
    set(StatValuesBar1,'String',StatString1);
    set(StatValuesBar2,'String',StatString2);
end
if strcmp(get(histFigure,'visible'),'on')
    histnames =
    {'roiHistogramR','roiHistogramG','roiHistogramB'};
    if size(imagesize) == [1 2]
        % intensity image case
        hist_h = findobj('tag','roiHistogram');
        [n,x] =
        hist(double(userdata.currentimage(pixel_range)),256);
        set(hist_h,'YData',n,'Xdata',x);
        for i=1:3
            hist_h = findobj('tag',histnames{i});
            set(hist_h,'YData',0,'Xdata',0);
        end
    else
        % truecolor (RGB) image case
        for i=1:3
            imagetmp = userdata.currentimage(:,i);
            if max(imagetmp(:)) ~= 0
                [n, x] = hist(double(imagetmp(pixel_range)),256);
                hist_h = findobj('tag',histnames{i});
                set(hist_h,'YData',n,'Xdata',x);
            end
        end
    end
end

```

```

        end
    end
    hist_h = findobj('tag','roiHistogram');
    set(hist_h,'YData',0,'Xdata',0);
end
set(userdata.hist_axisish,'xlim',[0 userdata.pixelmax]);
end
if strcmp(userdata.roitype,'rectangle')
    DetailRectangleFig = findobj('name','Detail Rectangle');
    if strcmp(get(DetailRectangleFig,'visible'),'on')
        drih = findobj('tag','DetailRectangleImg');
        cimg = userdata.currentimage(pixel_range);
        set(drih,'CData',reshape(cimg,userdata.cimgsize));
    end
end
end

%
% Sub-function - roi_resize
% -----
function roi_resize(roi_handler)

userdata = get(roi_handler,'userdata');
lastclickedpoint = userdata.lastclickedpoint;
draggable(roi_handler,'off');
userdata = get(roi_handler,'userdata');
mainfigure = gcf;
if strcmp(userdata.roitype,'circle')
    disp(' ');
    % calculating the center of the circle based on
    % R0 and 2 arbitrary points on the current circle
    userdata = get(roi_handler,'userdata');
    xi = get(roi_handler,'Xdata'); yi = get(roi_handler,'Ydata');
    x1 = xi(4); x2 = xi(9); % select the 4th and the 9th points
    (arbitrary selection)
    y1 = yi(4); y2 = yi(9);
    p = ((x1-x2)^2 + y2^2 - y1^2)/(2*(y1-y2));
    q = (x1-x2)/(y1-y2);
    a = (1+q^2);
    b = 2*(p-q^2*y1);
    c = p^2 - q^2*userdata.R0^2 + q^2*y1^2;
    userdata.ycenter = (-b - sqrt(b^2-4*a*c))/(2*a);
    userdata.xcenter = x2 - sqrt(userdata.R0^2 - (y2-
    userdata.ycenter)^2);
    userdata.isstartresize = 1;
    userdata.lastclickedpoint = lastclickedpoint;
    set(0,'PointerLocation',userdata.lastclickedpoint);
    set(roi_handler,'userdata',userdata);

    disp('Double click to finalize the ROI!');

set(gcf,'WindowButtonDownFcn','change_radius_onresize',
num2str(roi_handler,20),'');
set(gcf,'WindowButtonDownFcn','complete_radius');
waitfor(gcf,'WindowButtonDownFcn');
drawnow;
elseif strcmp(userdata.roitype,'rectangle')
    disp(' ');disp('Use the mouse to click and DRAG the desired
rectangle. ');disp(' ');
    rectpos = [0 0 0 0];
    while rectpos(3) == 0;
        rectpos = round(getrect(gca));
        if rectpos(3) == 0;
            mbh = msgbox('Use the mouse to click and DRAG the
desired rectangle. ','Roi Info');
            uiwait(mbh);
        end
    end
    % refresh the coordinate range and data for detailed
rectangle
    dry = round(rectpos(1)):round(rectpos(1)) +
round(rectpos(3)-1);
    drx = round(rectpos(2)):round(rectpos(2)) +
round(rectpos(4)-1);
    cimg = userdata.currentimage(drx,dry);
    userdata.cimgsize = size(cimg);
    DetailRectangleFig = findobj('name','Detail Rectangle');

```

```

    visible_onoff = get(DetailRectangleFig,'visible');
    drih = findobj('tag','DetailRectangleImg');
    set(drih,'CData',cimg);
    AxHandler = findobj('tag','DetailRectangleAx');
    axes(AxHandler); axis image;
    DetailRectangleFig = findobj('name','Detail Rectangle');
    if strcmp(visible_onoff,'off')
        set(DetailRectangleFig,'visible','off');
    end
    figure(mainfigure);
    % define the points for the line to be drawn
    roi_x = [rectpos(1), rectpos(1)+rectpos(3),
rectpos(1)+rectpos(3), ...
rectpos(1), rectpos(1)];
    roi_y = [rectpos(2), rectpos(2), rectpos(2)+rectpos(4), ...
rectpos(2)+rectpos(4), rectpos(2)];
    set(roi_handler,'Xdata',roi_x,'Ydata',roi_y);
    set(roi_handler,'userdata',userdata);
end

% calc. roi stat.
userdata = get(roi_handler,'userdata');
imagesize = size(userdata.currentimage);
xi = get(roi_handler,'Xdata'); yi = get(roi_handler,'Ydata');
roimask = poly2mask(xi,yi, imagesize(1),imagesize(2));
pixel_range = find(roimask);
if size(imagesize) == [1 2]
    % intensity image case
    roimean =
sprintf('%0.5g',mean(userdata.currentimage(pixel_range)));
    roistd =
sprintf('%0.5g',std(double(userdata.currentimage(pixel_range)
)));
    roimin =
num2str(min(userdata.currentimage(pixel_range)));
    roimax =
num2str(max(userdata.currentimage(pixel_range)));
else
    % truecolor (RGB) image case
    roimean = []; roistd=[]; roimin=[]; roimax=[];
    for i=1:3
        if i==1;sepchar="";else sepchar=" ";end
        imagetmp = userdata.currentimage(:,i);
        roimean = [roimean,sepchar,
sprintf('%0.5g',mean(imagetmp(pixel_range)))];
        roistd =
[roistd,sepchar,sprintf('%0.5g',std(double(imagetmp(pixel_ran
ge))))];
        roimin = [roimin,sepchar,
num2str(min(imagetmp(pixel_range)))];
        roimax = [roimax,sepchar,
num2str(max(imagetmp(pixel_range)))];
    end
end

% set the buttowndown event to "draggable"
userdata.roimean = roimean;
userdata.roistd = roistd;
userdata.roimin = roimin;
userdata.roimax = roimax;
userdata.roipixel_range = pixel_range;
userdata.lastclickedpoint = get(0,'PointerLocation');
set(roi_handler,'userdata',userdata);
fun_h = @(x)roitool(x,'roi_drag');
draggable(roi_handler, fun_h);

```

```

%
% Sub-function - change_radius_onresize
% -----
function change_radius_onresize(roi_handler)

```

```

userdata = get(roi_handler,'userdata');
t = userdata.t; xcenter = userdata.xcenter; ycenter =
userdata.ycenter;

```

```

if userdata.isstartresize
    R0 = userdata.R0;
    userdata.isstartresize = 0;
    set(roi_handler,'userdata',userdata);
else
    current_pts = get(gca,'CurrentPoint');
    current_pt = current_pts(1,1:2);
    R0 = sqrt((current_pt(1)-xcenter)^2+(current_pt(2)-
ycenter)^2);
end
xi = R0*cos(t) + xcenter;
yi = R0*sin(t) + ycenter;
set(roi_handler,'Xdata',xi,'Ydata',yi);

StatValuesBar1 = findobj('Tag','RoiStatValuesBar1');
StatValuesBar2 = findobj('Tag','RoiStatValuesBar2');
StatFigure = findobj('name','Roi Stat. Figure');
if strcmp(get(StatFigure,'visible'),'on')
    roimask = poly2mask(xi,yi, size(userdata.currentimage,1),
size(userdata.currentimage,2));
    imagesize = size(userdata.currentimage);
    xi = get(roi_handler,'Xdata'); yi = get(roi_handler,'Ydata');
    pixel_range = find(roimask);
    if size(imagesize) == [1 2]
        % intensity image case
        roimean =
sprintf('%0.5g',mean(userdata.currentimage(pixel_range)));
        roistd =
sprintf('%0.5g',std(double(userdata.currentimage(pixel_range)
)));
        roimin =
num2str(min(userdata.currentimage(pixel_range)));
        roimax =
num2str(max(userdata.currentimage(pixel_range)));
    else
        % truecolor (RGB) image case
        roimean = []; roistd=[]; roimin=[]; roimax=[];
        for i=1:3
            if i==1;sepchar='';else sepchar=';';end
            imagetmp= userdata.currentimage(:,:,i);
            roimean = [roimean,sepchar,
sprintf('%0.5g',mean(imagetmp(pixel_range)))];
            roistd =
[roistd,sepchar,sprintf('%0.5g',std(double(imagetmp(pixel_ran
ge))))];
            roimin = [roimin,sepchar,
num2str(min(imagetmp(pixel_range)))];
            roimax = [roimax,sepchar,
num2str(max(imagetmp(pixel_range)))];
        end
    end
    userdata.roimean = roimean;
    userdata.roistd = roistd;
    userdata.roipixel_range = pixel_range;
    userdata.roiarea = length(userdata.roipixel_range);
    set(roi_handler,'userdata',userdata);
    StatString1 = [' avg= ',userdata.roimean,' std= ',
userdata.roistd];
    StatString2 = [' max= ',userdata.roimax,'
min=',userdata.roimin, ...
' area= ',num2str(userdata.roiarea)];
    set(StatValuesBar1,'String',StatString1);
    set(StatValuesBar2,'String',StatString2);
end
drawnow;
mouseclick = get(gcf,'SelectionType');

%
% Sub-function - roi_showstat
% -----
function roi_showstat(roi_handler, onoff)

StatFigure = findobj('name','Roi Stat. Figure');
if strcmp(onoff,'on')
    StatValuesBar1 = findobj('Tag','RoiStatValuesBar1');

```

```

StatValuesBar2 = findobj('Tag','RoiStatValuesBar2');
set(StatFigure,'visible','on');
ShowStatMenu =
findobj(get(roi_handler,'UIContextMenu'),'tag','ShowStatMen
u');
set(ShowStatMenu,'Label','Show Statistics: +', 'Callback',
...

[roitool('num2str(roi_handler,20),'roi_showstat',"off");]);
% put the current stats. values to the StatValBar
userdata = get(roi_handler,'userdata');
% StatString1 = [' avg= ',userdata.roimean,' std= ',
userdata.roistd];
% StatString2 = [' maxpixel= ',userdata.roimax,'
min=',userdata.roimin, ...
' area= ',num2str(userdata.roiarea)];
StatString2 = [' maxpixel= ',userdata.roimax];
%set(StatValuesBar1,'String',StatString1);
set(StatValuesBar2,'String',StatString2);

else
    set(StatFigure,'visible','off');

    ShowStatMenu =
findobj(get(roi_handler,'UIContextMenu'),'tag','ShowStatMen
u');
    set(ShowStatMenu,'Label','Show Statistics: -', 'Callback', ...

[roitool('num2str(roi_handler,20),'roi_showstat',"on");]);
end

%
% Sub-function - roi_showhist
% -----
function roi_showhist(roi_handler, onoff)

if strcmp(onoff,'on')
    histFigure = findobj('name','Roi Histogram');
    set(histFigure,'visible','on');
    ShowHistMenu =
findobj(get(roi_handler,'UIContextMenu'),'tag','ShowHistMen
u');
    set(ShowHistMenu,'Label','Show Histogram: +', 'Callback',
...

[roitool('num2str(roi_handler,20),'roi_showhist',"off");]);
else
    histFigure = findobj('name','Roi Histogram');
    set(histFigure,'visible','off');
    ShowHistMenu =
findobj(get(roi_handler,'UIContextMenu'),'tag','ShowHistMen
u');
    set(ShowHistMenu,'Label','Show Histogram: -', 'Callback',
...

[roitool('num2str(roi_handler,20),'roi_showhist',"on");]);
end

%
% Sub-function - roi_showdetrect
% -----
function roi_showdetrect(roi_handler, onoff)

if strcmp(onoff,'on')
    DetailRectangleFig = findobj('name','Detail Rectangle');
    set(DetailRectangleFig,'visible','on');
    ShowDetRectMenu =
findobj(get(roi_handler,'UIContextMenu'),'tag','ShowDetRect
Menu');
    set(ShowDetRectMenu,'Label','Show DetailedRect: +',
'Callback', ...

[roitool('num2str(roi_handler,20),'roi_showdetrect',"off");]);
;
else
    DetailRectangleFig = findobj('name','Detail Rectangle');

```

```

set(DetailRectangleFig,'visible','off');
ShowDetRectMenu =
findobj(get(roi_handler,'UIContextMenu'),'tag','ShowDetRect
Menu');
set(ShowDetRectMenu,'Label','Show DetailedRect: -',
'Callback',...

```

```

[roitool('num2str(roi_handler,20),'roi_showdetrect',"on");])
;
end

```

EVP

```
function varargout = EVP(varargin)
```

```

%Función que se usa para generar la gráfica de recuperación.
También
%permite obtener el valor interpolado para corregir una lesión
%Entradas

```

```

% X: Vector arreglo que contiene los diámetros de la
lesiones
% Y: Vector arreglo que contiene cociente del máximo
píxel de cada
% esfera entre el máximo píxel de la esfera mayor.
% Begin initialization code - DO NOT EDIT

```

```
gui_Singleton = 1;
```

```

gui_State = struct('gui_Name', mfilename, ...
'gui_Singleton', gui_Singleton, ...
'gui_OpeningFcn', @EVP_OpeningFcn, ...
'gui_OutputFcn', @EVP_OutputFcn, ...
'gui_LayoutFcn', [], ...
'gui_Callback', []);

```

```

if nargin && ischar(varargin{1})
gui_State.gui_Callback = str2func(varargin{1});
end

```

```

if nargin
[varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else

```

```
gui_mainfcn(gui_State, varargin{:});
```

```
end
```

```
% End initialization code - DO NOT EDIT
```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Crea imagen de fondo
de la interface %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
function EVP_OpeningFcn(hObject, eventdata, handles,
varargin)
```

```

handles.output = hObject;
guidata(hObject, handles);
A=imread('esferas.jpg');
Img=image(A,'Parent',handles.axes1);
set(handles.axes1,'Visible','off','YDir','reverse','XLim',get(Img,
'XData'),'Ylim',get(Img,'YData'));

```

```

% --- Outputs from this function are returned to the command
line.

```

```

function varargout = EVP_OutputFcn(hObject, eventdata,
handles)
varargout{1} = handles.output;

```

```
function edit5_Callback(hObject, eventdata, handles)
```

```

% --- Executes during object creation, after setting all
properties.

```

```

function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

```

```
function edit6_Callback(hObject, eventdata, handles)
```

```

% --- Executes during object creation, after setting all
properties.

```

```

function edit6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

```

```

% --- If Enable == 'on', executes on mouse press in 5 pixel
border.

```

```

% --- Otherwise, executes on mouse press in 5 pixel border or
over text23.

```

```
function text23_ButtonDownFcn(hObject, eventdata, handles)
```

```
function diametro1_Callback(hObject, eventdata, handles)
```

```

diametro1 = str2double(get(hObject,'String'));
if isnan(diametro1)
set(hObject,'String',0);
errordlg('Ingresa un número','Error');
end

```

```
%salvar valores
```

```

handles.metricdata.diametro1 = diametro1;
guidata(hObject,handles)

```

```

% --- Executes during object creation, after setting all
properties.

```

```
function diametro1_CreateFcn(hObject, eventdata, handles)
```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

```

```
function diametro2_Callback(hObject, eventdata, handles)
```

```

diametro2 = str2double(get(hObject,'String'));
if isnan(diametro2)
set(hObject,'String',0);
errordlg('Input must be a number','Error');
end

```

```

handles.metricdata.diametro2 = diametro2;
guidata(hObject,handles)

```

```

% --- Executes during object creation, after setting all
properties.

```

```
function diametro2_CreateFcn(hObject, eventdata, handles)
```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
set(hObject,'BackgroundColor','white');
end

```

```
function diametro3_Callback(hObject, eventdata, handles)
```

```

diametro3 = str2double(get(hObject,'String'));
if isnan(diametro3)
set(hObject,'String',0);
errordlg('Input must be a number','Error');
end

```

```
% Salvar valor ingresado diametro3
```

```

handles.metricdata.diametro3 = diametro3;
guidata(hObject,handles)

```

```

% --- Executes during object creation, after setting all
properties.

```

```
function diametro3_CreateFcn(hObject, eventdata, handles)
```

```

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function diametro4_Callback(hObject, eventdata, handles)
diametro4 = str2double(get(hObject, 'String'));
if isnan(diametro4)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% Salvar valor ingresado diametro4
handles.metricdata.diametro4 = diametro4;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function diametro4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function diametro5_Callback(hObject, eventdata, handles)
diametro5 = str2double(get(hObject, 'String'));
if isnan(diametro5)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% % Salvar valor ingresado diametro5
handles.metricdata.diametro5 = diametro5;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function diametro5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function diametro6_Callback(hObject, eventdata, handles)
diametro6 = str2double(get(hObject, 'String'));
if isnan(diametro6)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% Salvar valor ingresado diametro6
handles.metricdata.diametro6 = diametro6;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function diametro6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Pixelfondo_Callback(hObject, eventdata, handles)
Pixelfondo = str2double(get(hObject, 'String'));
if isnan(Pixelfondo)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% Salvar valor ingresado maxpixel1
handles.metricdata.Pixelfondo = Pixelfondo;
guidata(hObject,handles)

```

```

% --- Executes during object creation, after setting all
properties.
function Pixelfondo_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Pixel1_Callback(hObject, eventdata, handles)
Pixel1 = str2double(get(hObject, 'String'));
if isnan(Pixel1)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Salvar valor
ingresado maxpixel1 %%%%%%%%%%
handles.metricdata.Pixel1 = Pixel1;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function Pixel1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Pixel2_Callback(hObject, eventdata, handles)
Pixel2 = str2double(get(hObject, 'String'));
if isnan(Pixel2)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% Salvar valor ingresado maxpixel2
handles.metricdata.Pixel2 = Pixel2;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function Pixel2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Pixel3_Callback(hObject, eventdata, handles)

Pixel3 = str2double(get(hObject, 'String'));
if isnan(Pixel3)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% Salvar valor ingresado maxpixel3
handles.metricdata.Pixel3 = Pixel3;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function Pixel3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Pixel3 (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 Pixel4_Callback(hObject, eventdata, handles)

Pixel4 = str2double(get(hObject, 'String'));
if isnan(Pixel4)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
% Salvar valor ingresado maxpixel4
handles.metricdata.Pixel4 = Pixel4;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function Pixel4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Pixel5_Callback(hObject, eventdata, handles)
Pixel5 = str2double(get(hObject, 'String'));
if isnan(Pixel5)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
% Salvar valor ingresado maxpixel5
handles.metricdata.Pixel5 = Pixel5;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function Pixel5_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function Pixel6_Callback(hObject, eventdata, handles)

Pixel6 = str2double(get(hObject, 'String'));
if isnan(Pixel6)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end
% Salvar valor ingresado maxpixel6
handles.metricdata.Pixel6 = Pixel6;
guidata(hObject,handles)

% --- Executes during object creation, after setting all
properties.
function Pixel6_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Genera la gráfica por interpolación
function pushbutton1_Callback(hObject, eventdata, handles)

x1= handles.metricdata.diametro1;
x2= handles.metricdata.diametro2;
x3= handles.metricdata.diametro3;
x4= handles.metricdata.diametro4;
x5= handles.metricdata.diametro5;

y1= handles.metricdata.Pixel1/handles.metricdata.Pixel5;
y2= handles.metricdata.Pixel2 / handles.metricdata.Pixel5;
y3= handles.metricdata.Pixel3 / handles.metricdata.Pixel5;
y4= handles.metricdata.Pixel4 / handles.metricdata.Pixel5;
y5= handles.metricdata.Pixel5 / handles.metricdata.Pixel5;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Definición de los valores eje horizontal para interpolación.

```

```

init= x1;%valor minimo de diametro
final=x5;%máximo diametro
const=100;%constante para definir el numero de divisiones
horizontales
rango = (final-init);
pr=rango/const;%Incrementos del eje horizontal
a=x1;

%Generar los elementos del eje horizontal
for i=1:99
    a = a + pr;
    b(i)=a;
    xi(i)=b(i);
end

%Crear vector de diametros
x=[x1,x2,x3,x4,x5];
%crear vector de cocientes de maxpíxeles.
y=[y1,y2,y3,y4,y5];

%Cálculo de interpolación

yi = interp1(x,y,xi,'cubic');

plot(x,y,'-b',xi,yi,'ok','LineWidth',1,...
'MarkerFaceColor','g',...
'MarkerSize',5)
title('Coeficiente de Recuperación de lesiones')
xlabel('Diámetro de la lesión (mm)');
ylabel('CR');
handles.metricdata.vectorX=x;
handles.metricdata.vectorY=y;
guidata(hObject,handles)
grid;

% --- Executes during object creation, after setting all
properties.
function axes1_CreateFcn(hObject, eventdata, handles)

function dia_Callback(hObject, eventdata, handles)

dia = str2double(get(hObject, 'String'));
if isnan(dia)
    set(hObject, 'String', 0);
    errordlg('Input must be a number','Error');
end

% Save the new density value
handles.dia = dia;
guidata(hObject,handles)

guidata(hObject,handles)
% --- Executes during object creation, after setting all
properties.
function dia_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes during object creation, after setting all
properties.
function val_CreateFcn(hObject, eventdata, handles)

%Calculo del valor interpolado.
function pushbutton4_Callback(hObject, eventdata, handles)

diametro=handles.metricdata.vectorX;
NormValue=handles.metricdata.vectorY;

LesionAInt=handles.dia
Yi = interp1(diametro,NormValue,LesionAInt,'linear');
set(handles.text26,'string',Yi);
guidata(hObject,handles)

```