UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA POLITÉCNICA SUPERIOR DE GANDIA

GRADO EN ING. SIST. DE TELECOM., SONIDO E IMAGEN

“ Desarrollo de una GUI en MATLAB que simula un escáner de Tomografía Computarizada para pocas proyecciones. ”

TRABAJO FINAL DE GRADO

Autor/a:
Nuria Bosch Ventura

Tutor:
Vicente E. Vidal Gimeno

Director Experimental:
Mónica Chillarón Pérez

GANDIA, Septiembre 2020

Resumen

En este trabajo se presenta un diseño e implementación de una aplicación informática mediante el uso de un entorno de desarrollo (GUIDE), que se encarga de generar interfaces gráficas de usuario (GUI) en el lenguaje de programación de MATLAB.

Dicha aplicación realiza las simulaciones de sinogramas de baja dosis en fantomas de tomografía computarizada (TC) en 2D y la reconstrucción de las imágenes médicas de TC asociadas a dichos sinogramas.

La finalidad de este trabajo es, que con la ayuda de la aplicación que se ha llevado a cabo, los investigadores puedan generar una base de datos que servirá para entrenar una red neuronal donde, dado un sinograma, lo asocie a un TC y así facilitar la reconstrucción de imagen médica de TC algebraico iterativo.

Palabras Clave

Aplicación, simulación, fantoma, sinograma, reconstrucción.

Abstract

In this work, the creation of a computer application is presented. The application has been developed using a development environment (GUIDE), which is responsible for generating graphical interfaces of user (GUI) in the MATLAB programming language.

This application is in charge of carrying out the simulations of low-dose sinograms in 2D computed tomography (CT) phantoms and the reconstruction of the CT images associated with these sinograms.

The purpose of this work is that, with the help of the application that has been carried out, researchers can generate a database that could be used to train a neural network where a given sinogram could be associated with a set of CT reconstructions and thus it would facilitate the subsequent reconstructions by iterative algebraic methods.

Keywords

Application, simulation, phantom, sinogram, reconstruction.

Índice

Índice de figuras

1. Introducción

El uso de las tecnologías se vuelve cada día más importante y esencial en muchos aspectos de nuestra vida, sobre todo si hablamos de aquellas tecnologías que se encuentran presentes en el sector de la medicina, ya que éstas han permitido que mejore, notablemente, nuestra calidad de vida.

Desde las primeras vacunas hasta el uso radiografías y escáneres, la medicina sigue progresando con el fin de prevenir, curar y diagnosticar enfermedades, de forma que estas puedan ser tratadas a tiempo. La tecnología sigue avanzando, y está con la medicina, proporcionando mejoras e innovaciones en el sector.

Por ejemplo, en el caso de la realización de un TAC con un escáner portátil, pueden haber restricciones que no permitan captar todos los ángulos, por razones físicas, por ello se empezaron a usar métodos iterativos en este ámbito, con el fin de proporcionar reconstrucciones fiables con menos vistas.

1.1. Objetivo general

El objetivo general de este trabajo es la creación de una aplicación informática que ayude a la simulación de sinogramas de baja dosis de fantomas de tomografía computarizada (TC) y la reconstrucción de imágenes médicas de TC asociadas a dichos sinogramas, con las cuales, posteriormente, se podrá generar una base de datos que ayudará al entrenamiento de una red neuronal (IA), y así dado un sinograma poder asociarlo con una TC factible para poder aportar una solución inicial del proceso de la reconstrucción de imagen médica de TC mediante métodos iterativos algebraicos.

1.2. Objetivos específicos

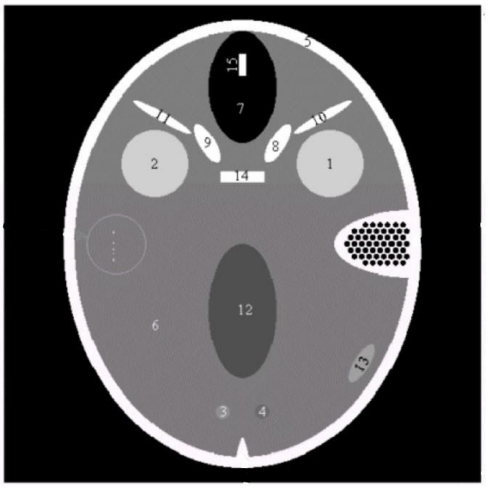
Este proyecto está estructurado en módulos para seguir uno por uno los objetivos específicos, que son en los que se divide la aplicación, lo cual hace más sencillo el desarrollo y funcionamiento del mismo.

Las funciones de reconstrucción del TC empleadas en este trabajo son propiedad de los investigadores Vicente Vidal y Mónica Chillarón del grupo de Computación de Altas Prestaciones para Aplicaciones Médicas de la Universidad Politècnica de València.

1.2.1. Generador de la imagen de referencia, del sinograma y de la matriz del sistema

Para entender la parte del generador primero se explicará que son las imágenes de referencia, también llamadas fantomas, y los sinogramas.

Un fantoma consiste en una representación simple de las estructuras anatómicas que son importantes para evaluar la calidad de una imagen de la TC. El fantoma utilizado como ejemplo en este trabajo es el Forbild Head Phantom, que está definido por objetos geométricos simples como esferas, cilindros, elipsoides o conos [1], que son una representación aproximada de los elementos presentes en la cabeza humana, como se puede observar en la *Figura 1*:



Etiqueta	Geometría	Número CT	Relación anatómica
1	Esfera	60	Ojo
2	Esfera	60	Ojo
3	Esfera	52.5	
4	Esfera	47.5	
5	Elipse	800	Cúpula
6	Elipse	50	Materia cerebral homogénea
7	Elipse	-1000	Senos frontales
8	Elipse	800	Hueso alrededor del seno frontal
9	Elipse	800	Hueso alrededor del seno frontal
10	Cilindro Elíptico	800	Hueso alrededor del seno frontal
11	Cilindro Elíptico	800	Hueso alrededor del seno frontal
12	Elipsoide	45	Ventrículo
13	Elipsoide	55	Hematoma subdural
14	Cilindro Elíptico	800	Hueso alrededor del seno frontal
15	Cilindro Elíptico	800	Hueso alrededor del seno frontal

Figura 1: Ejemplo de una imagen de referencia, o fantoma.

Por otra parte el sinograma, que lo denotaremos como *b,* representa la intensidad de todos los rayos en su llegada a los detectores, o lo que es lo mismo, la atenuación que han sufrido al atravesar el objeto, la cual será diferente para cada material (hueso, sangre, grasa, etc.) según su densidad [2].

Para proyectar una imagen, se puede utilizar un salto angular equidistante, o bien seguir un esquema con desplazamientos como el que se muestra en la *Figura 2.1* [3].

$$\Theta_i = \begin{cases} (360/vistas) * (i-1) & \text{si } 1 <= i <= (vistas/2) \\ 1{,}5 + (360/vistas) * (i-1) & \text{si } (vistas/2) < i(vistas) \\ (360-1) & \text{si } i = vistas \end{cases}$$

Figura 2.1: Ecuación de los ángulos de disparo.

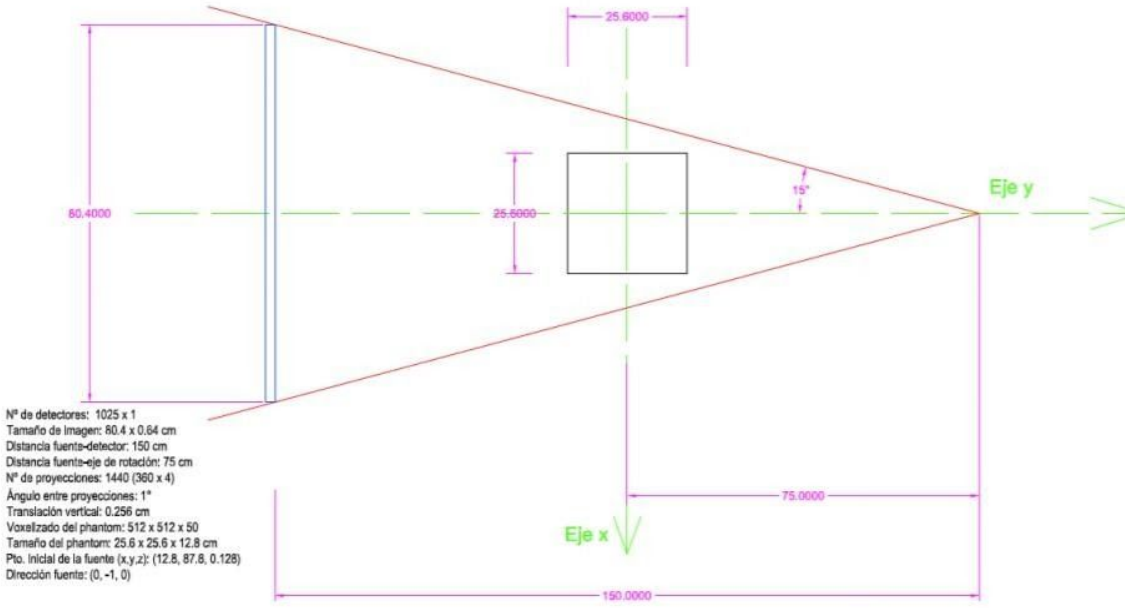


Figura 2.2: Característica del escáner.

En este módulo se generará el fantoma, el sinograma y la matriz del sistema, permitiendo al usuario el cambio de todos los valores de los parámetros del escáner, como son las dimensiones de la imagen, el número de detectores o el número de vistas entre otros (*Figura 2.2*). La matriz del sistema la denotaremos como A.

1.2.2. Reconstrucción de imagen

El módulo de reconstrucción de imagen contiene las funciones que resuelven el sistema de ecuaciones, Ax=b, donde A es la matriz asociada al escaner, b es el sinograma y x es la imagen obtenida. Para la resolución de imagen se utiliza el método iterativo LSQR [4], que puede combinarse con las técnicas de regularización (WTD-STF) [5], la de aceleración FISTA [6], y el filtro bilateral [7]. Si el usuario quiere también podrá utilizar los filtros del entorno de MATLAB, como el filtro bilateral.

1.2.3. Métricas de calidad

El módulo de métricas se utiliza para analizar la calidad de la imagen reconstruida comparada con la imagen de referencia. Se utilizan varias métricas ya que cada una analiza unas propiedades de la imagen.

1.2.4. Visualizador de imágenes y sinogramas

Tras crear fantomas, sinogramas e imágenes reconstruidas el usuario podrá visualizar dichas imágenes, tanto en el apartado del generador, como en el de reconstrucción de imagen. Para ello nos ayudaremos de la opción *axes* disponible en GUI.

1.2.5. Etiquetado y almacenamiento de los datos

Con el etiquetado lo que se consigue es facilitar el uso de la aplicación, pues el usuario sabrá en todo momento el significado de cada una de las variables.
Y la parte del almacenamiento se ha implementado para poder guardar cada fantoma, sinograma, matriz o imagen reconstruida que el usuario haya generado. Esta parte es fundamental para, en el futuro, crear la base de datos.

1.2.6. Herramienta: añadir ruido

Al disminuir la radiación empleada para la generación de un TAC real, también tendremos una menor definición de estas imágenes, provocando también la aparición de ruido.
Con este apartado lo que se pretende hacer es añadir distintos tipos de ruido para analizar qué método, de los disponibles en el apartado de reconstrucción de imagen, lo elimina mejor [8].

2. Metodología

A continuación, se va a abordar uno por uno los módulos descritos en el apartado anterior “Objetivos específicos”, siguiendo el esquema de la *Figura 3*.

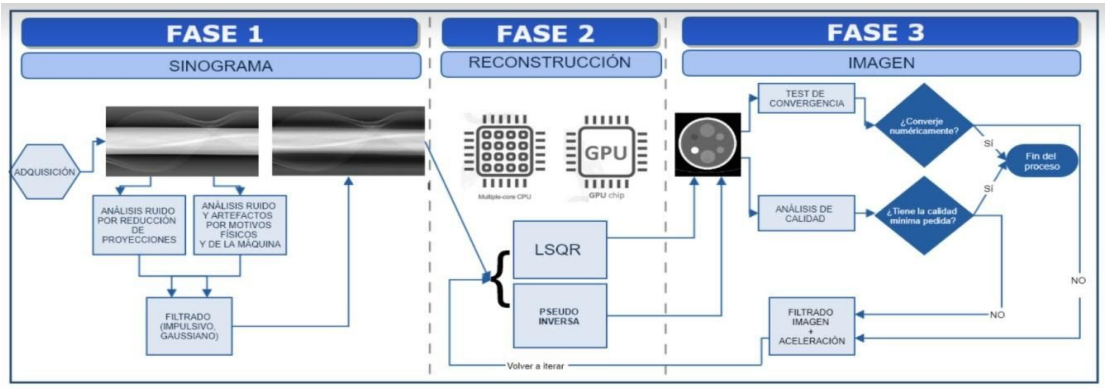


Figura 3: Esquema de las fases del procesado de reconstrucción de una imagen TC [8].

En el esquema se observan las distintas fases: en la fase 1 se genera el sinograma, como se podrá ver en el apartado 2.1, en la fase 2, se lleva a cabo la reconstrucción de imagen, apartado 2.2, y finalmente en la fase 3 se analizará la calidad de la imagen obtenida, que se verá en el apartado 2.3.

2.1. Generador de la imagen de referencia, del sinograma y de la matriz del sistema

Este apartado es el primer módulo con el que nos encontramos al abrir el menú principal de la aplicación informática *(Figura 4)*, en el cual también podemos ver el módulo de reconstrucción de imagen, que veremos más adelante.

Figura 4: Menú Principal

En esta ventana de “Menú Principal” podemos encontrar dos botones de pulsado, el cual si es pulsado hace funcionar el siguiente código:

```
function pushbutton1_Callback(hObject, eventdata, handles)

    close(proyecto); % Cerramos el Menú Principal
    Modulo1 % Abrimos el Generador
```

Dicho código se encarga de cerrar la ventana en la que se encuentra el usuario y abrir la ventana seleccionada, en este caso la del “Generador de sinogramas, phantom y matriz del sistema” *(Figura 5)*.

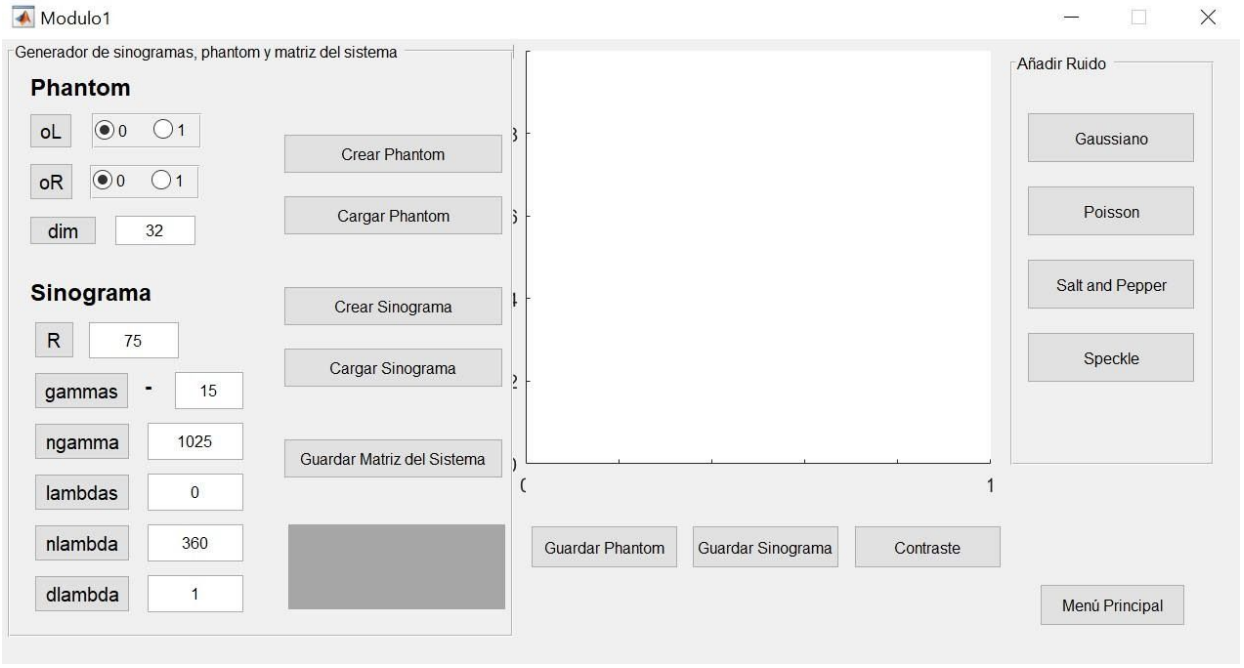


Figura 5: Menú del módulo “Generador de sinogramas, phantom y matriz del sistema”

Como se puede observar en la *Figura 5*, este módulo consta de distintas partes:

2.1.1. Phantom (fantoma):

Figura 6: Parte del Fantoma.

En esta primera parte (*Figura 6*) es donde creamos los fantomas o imágenes de referencia, en este caso de cabeza.

El usuario puede escoger si activa o no (‘0’ desactivado, ‘1’ activado) la vista, tanto de la oreja izquierda, como de la oreja derecha, y también se le da a escoger las dimensiones de la imagen, cuyo valor por defecto es de 32x32 píxels, aunque dicho valor puede ser de 64, 128, 256, 512, etc.

Para finalmente crear este fantoma, se tendrá que presionar el botón llamado “Crear Phantom”, aunque también existe la posibilidad que el fantoma que se quiere utilizar, o visualizar, ya esté creado, con lo cual el botón que se presione será el de “Cargar Phantom”, que abrirá un buscador para que el usuario escoja el que quiera cargar *(Figura 7)*.

En la *Figura 8* se muestra un fantoma para diferentes resoluciones, nótese que dependiendo de la resolución se podrán detectar diferentes cuerpos del fantoma.

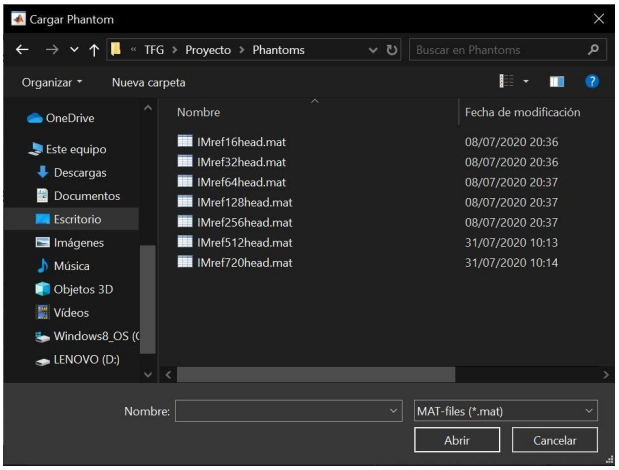
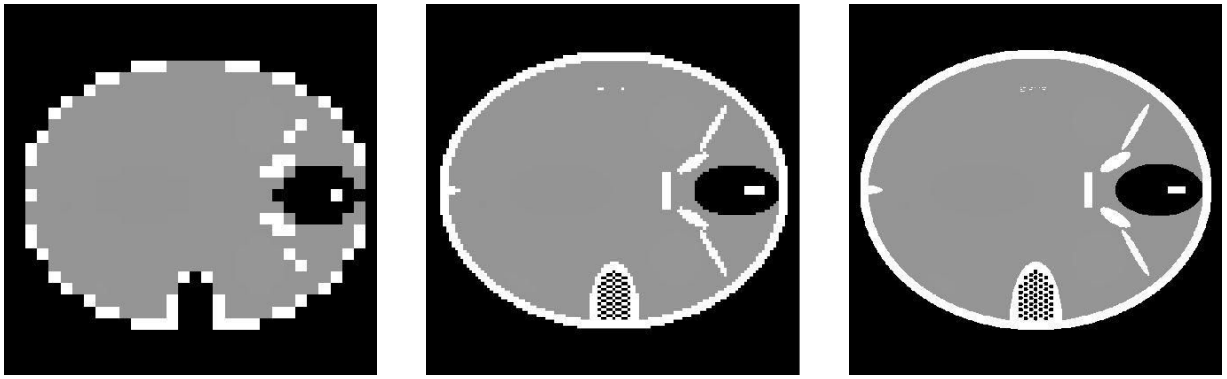


Figura 7: Ventana para cargar un fantoma.



Dim: 32 *Dim: 128* *Dim: 512*

Figura 8: Ejemplos de fantomas de cabeza de distintas dimensiones.

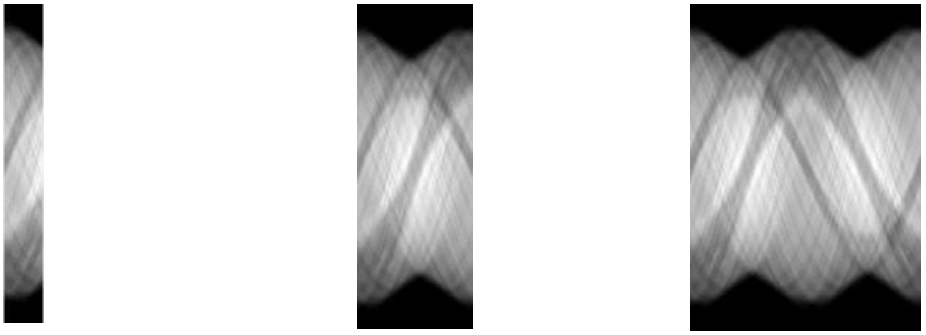
2.1.2. Sinograma:

Figura 9: Parte del Sinograma.

En esta parte, como se puede observar en la *Figura 9*, el usuario dispone de más variables que puede modificar según lo necesite, a las cuales se les ha asignado un valor por defecto, permitiendo cambiar los parámetros del escáner (*Figura 2.2*). Dichos parámetros son los siguientes:

- **R:** Distancia desde la fuente de rayos X hasta el centro ISO (Radio del escáner).
- **Gammas:** Posición del primer detector.
- **Ngamma:** Número de detectores.
- **Lambdas:** Posición angular de la primera fuente de rayos X (respecto al centro de una imagen).
- **Nlambda:** Número de vistas.
- **Dlambda:** Incremento angular para la trayectoria de los rayos X.

Como en el apartado del fantoma, en este también disponemos de un botón para crear los sinogramas y cargar los previamente creados para poder visualizarlos.



Nlambda: 60 *Nlambda: 180* *Nlambda: 360*

Figura 10: Sinogramas con distintos Nlambda (número de vistas).

2.2. Reconstrucción de imagen

A este apartado se accede de la misma manera que el primero, a través del menú principal *(Figura 11)* y pulsando el botón de “Reconstrucción de imagen”, el cual nos abre una nueva ventana *(Figura 12)*.

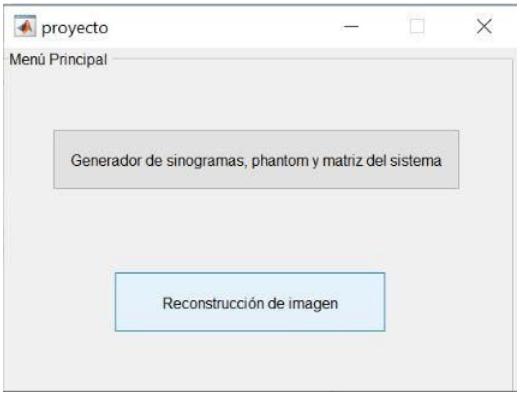


Figura 11: Menú Principal.

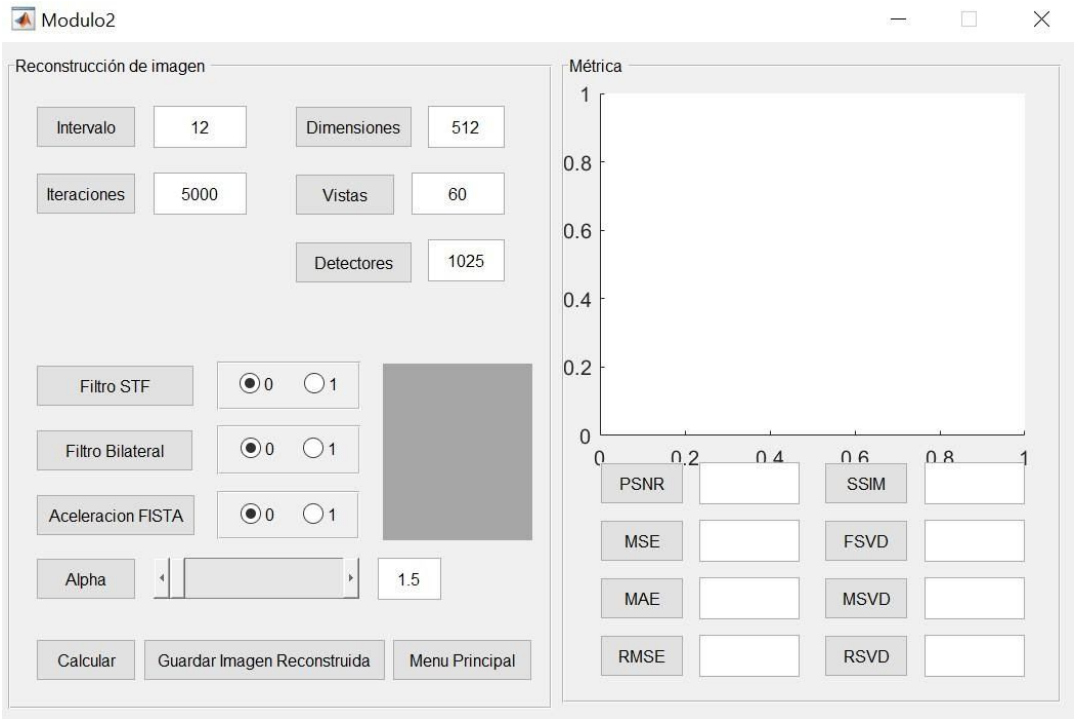


Figura 12: Menú del módulo “Reconstrucción de imagen”.

Este módulo también se separa por partes, como se puede ver en la *Figura 12*, pero este apartado se centra solo en la parte de la Reconstrucción de imagen.
Al igual que en el módulo anterior, en este el usuario también puede modificar ciertos parámetros a la hora de reconstruir una imagen:

- **Intervalo:** Representa cuántas iteraciones del método LSQR se hacen antes de realizar una aplicación del filtro STF, bilateral y FISTA. Toma valores entre 4 y 30, por defecto 12.
- **Iteraciones:** Máximo número de iteraciones totales.
- **Dimensiones:** Resolución de la imagen a reconstruir, se pueden tomar los siguientes valores: 32, 64, 128, 256, 512.
- **Vistas:** Número de vistas a utilizar, toma valores de 30 a 360.
- **Detectores:** Número de detectores físicos del escáner TAC que se está simulando.
- **Filtro STF:** La función de este filtro es regularizar la solución obtenida. Mejora la aproximación y elimina ruido de la imagen, conservando los bordes, ya que tiene en cuenta los gradientes en todas las direcciones [5].
- **Filtro Bilateral:** Es un filtro preservador de bordes y de reducción de ruido para el suavizado de imágenes [7].
- **Aceleración FISTA:** Ayuda a que el proceso total de reconstrucción sea más rápido. Si se aplica, el método LSQR convergerá en menos iteraciones [6].
- **Alpha:** Parámetro que se utiliza en el filtro STF.

Una vez el usuario ha escogido qué parámetros quiere utilizar para la reconstrucción de imagen se pasa a realizar el cálculo de la imagen reconstruida, para ello utilizamos el método LSQR, que es un método analíticamente equivalente al método estándar de gradientes conjugados, pero posee propiedades numéricas más favorables [4].

Una vez se tiene la imagen reconstruida, se analizará y comparará con la imagen de referencia correspondiente en el siguiente módulo, “Métricas de Calidad”.

2.3. Métricas de calidad

La calidad de imagen puede degradarse debido a distorsiones durante la adquisición y el procesamiento de la imagen. Para ello las métricas realizan un seguimiento de los errores no percibidos a medida que se propagan a través de una canalización de procesamiento de imágenes [9].

En este apartado, el cual se sitúa en el mismo módulo que el anterior, como se puede ver en la *Figura 12*, se va a analizar la calidad de la imagen reconstruida, en comparación con la imagen de referencia (fantoma). Para ello se procederá al cálculo de los siguientes parámetros:

- **PSNR:** se deriva del error cuadrado medio, e indica la relación de la intensidad máxima de píxeles a la potencia de la distorsión [10].
- **MSE:** Error cuadrático medio, mide la diferencia cuadrada promedio entre los valores de píxel reales e ideales [10].
- **MAE:** Error absoluto medio, mide la diferencia entre dos variables continuas [11].
- **RMSE:** Raíz del error cuadrático medio, diferencia entre los valores de un modelo, y los valores observados [11].
- **SSIM:** Esta métrica combina la estructura de imagen local, la luminancia y el contraste en una única puntuación de calidad local [10], nos indica el índice de similitud.
- **FSVD:** Esta función calcula el error entre dos imágenes utilizando la información de las métricas MSVD y RSVD. Realiza un promedio de ambas métricas para diferentes tamaños de bloques [12].
- **MSVD:** Esta función calcula la métrica de calidad M-SVD entre dos imágenes con el mismo formato, con una sola capa (grises) [13]. Esta métrica utiliza la información de los valores singulares que están asociados al contraste de la imagen.
- **RSVD:** Esta función calcula el error entre dos imágenes utilizando la información de los vectores singulares ya que estos deben tener más información estructural que los valores singulares [14].

2.4. Visualizador de imágenes y sinogramas

Para la visualización de las imágenes, como las imágenes de referencia o las reconstruidas, y los sinogramas, en ambos módulos se ha añadido una pequeña ventana, como se puede ver en la *Figura 13*. Esta ha sido creada con la opción de *axes*.

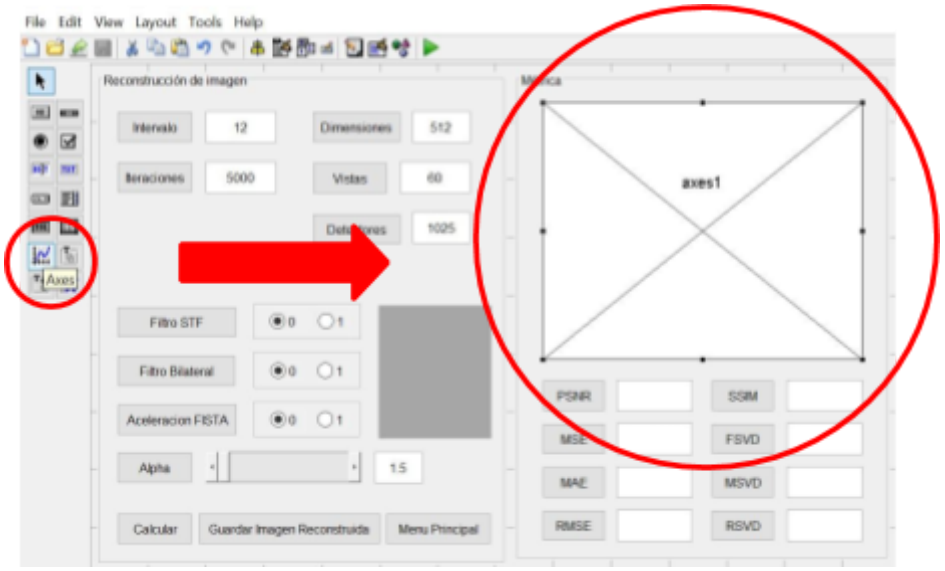


Figura 13: "Axes" para la visualización de imágenes.

La función utilizada para enviar las imágenes al apartado de *axes1* es *imshow(),* que lo que hace es mostrar la imagen que se desea visualizar, en una escala de grises (*Figura 14*).

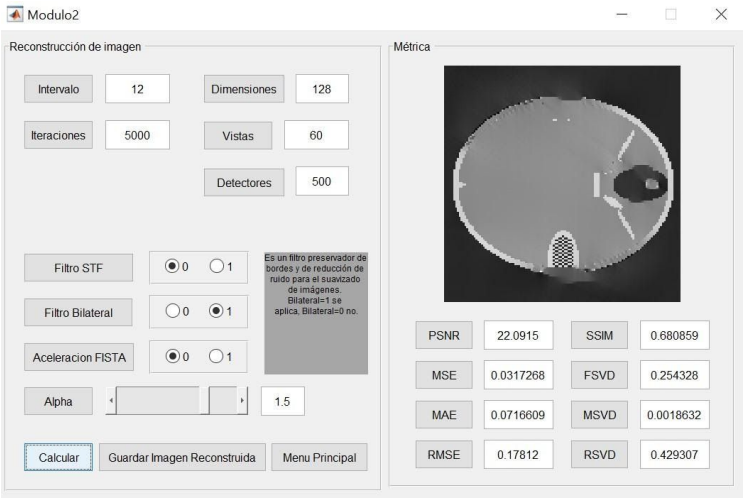


Figura 14: Ejemplo de visualización de imagen.

El módulo de “Generador de sinogramas, phantom y matriz del sistema” también cuenta con la opción de añadir contraste a la imagen para una mejor visualización de la misma (*Figura 15*). En el módulo de “Reconstrucción de imagen” dicha opción ya va incluida en el código.

Fantoma (sin contraste) *Fantoma (con contraste)*

Figura 15: Aplicación de contraste sobre un fantoma.

2.5. Etiquetado y almacenamiento de los datos

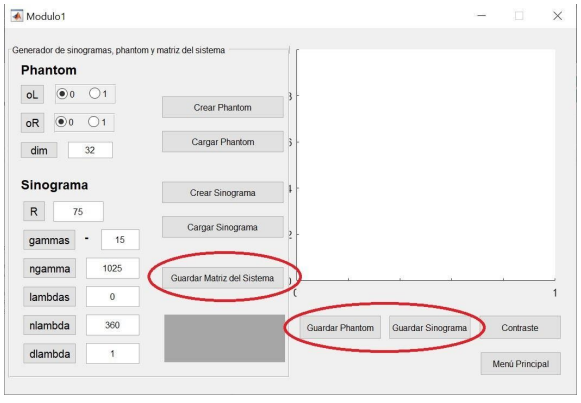
Para que el usuario sepa en cada módulo cuales son las variables y lo que hace cada una de ellas, se ha decidido implementar un recuadro donde aparecerá la información de cada variable al pulsar sobre la misma, ya que el nombre de cada variable está sobre un botón, que al presionarlo hace funcionar el siguiente código:

```
function ngamma_Callback(hObject, eventdata, handles)
     set(handles.text10,'String','Número de detectores.');
```

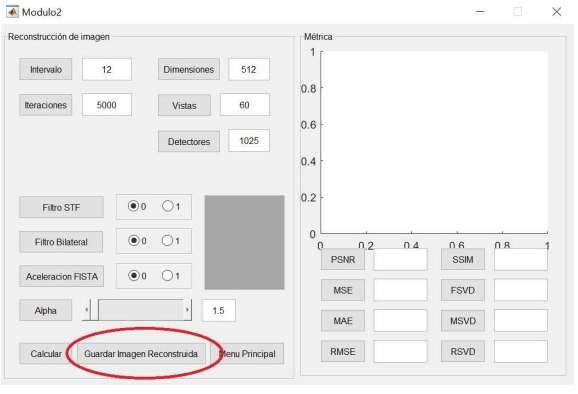
Por ejemplo, si el usuario quiere saber que significa “ngamma” sólo tendrá que presionar sobre el nombre y se enviará el texto “Número de detectores” al recuadro de texto estático, como se ve en la *Figura 16*.

Figura 16: Etiquetado de datos.

El apartado de almacenamiento ayudará en un futuro a crear la base de datos. Este apartado se ha implementado para guardar los fantomas, sinogramas, matrices del sistema y las imágenes reconstruidas. Cada uno de los módulos cuenta con un botón de guardado (*Figura 17*), el cual abre automáticamente una ventana para que el usuario elija donde quiere guardar la imagen creada.



Menú “Generador de sinogramas, phantom y matriz del sistema”



Menú “Reconstrucción de imagen”

Figura 17: Botones de guardado de datos.

A la hora de guardar los datos, cada uno se guarda con una nomenclatura diferente:

- Fantoma: IMref16head.mat (donde 16 es el número de dimensiones).
- Sinograma: Sinograma180_500detectores128_75R_b.mat (donde 180 es el número de vistas, 500 número de detectores, 128 número de dimensiones, y donde 75 es el radio del escaner).
- Matriz del sistema: Matrices180_500detectores128A.mat (donde 180 es el número de vistas, 500 número de detectores y 128 número de dimensiones)
- Imagen reconstruida: ref128x180x12-010.mat (donde 128 es el número de dimensiones, 180 el número de vistas, 12 el número de iteraciones, el primer 0 indica que el filtro stf esta desactivado, el 1 indica que el filtro bilateral esta activado y el último 0 indica que la aceleración FISTA esta desactivado).

2.6. Herramienta: añadir ruido

En lo que respecta a la herramienta de “Añadir Ruido”, la idea principal es poder dar al usuario la opción de añadir distintos tipos de ruido a los fantomas y sinogramas creados. Para ello se ha añadido en el módulo de “Generador de sinogramas, phantom y matriz del sistema” un apartado (*Figura 18*) con 4 tipos de ruidos distintos: gaussiano, poisson, salt and pepper y speckle.

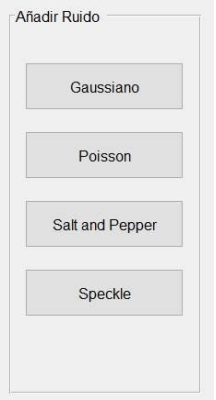


Figura 18: Apartado de la herramienta “Añadir Ruido”.

- **Gauss:** añade ruido blanco gaussiano con varianza de 0.01 a la imagen en escala de grises [15].
- **Poisson:** genera ruido de Poisson a partir de los datos en lugar de añadir ruido artificial..
- **Salt & Pepper:** añade sal y pimienta, con la densidad de ruido predeterminada 0,05. Esto afecta aproximadamente al 5% de los píxeles.
- **Speckle:** añade ruido multiplicativo utilizando la ecuación, donde se distribuye uniformemente el ruido aleatorio con la media 0 y la varianza 0,05 [16].

3. Código

En este apartado se mostrará el código utilizado para la programación de los distintos botones con los que cuenta la aplicación. Para ello, en algunos casos, en vez de mostrar la programación de todos los botones, en aquellos que sean iguales, tan solo se pondrá un ejemplo que los englobará a todos.

3.1. Menú principal

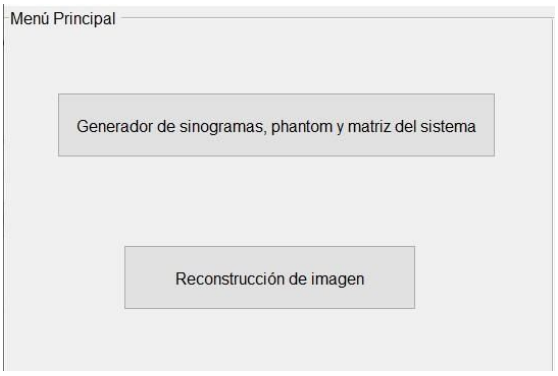


Figura 19: Botones del "Menú principal"

El "Menú principal" (*Figura 19*) cuenta con dos botones con una programación muy similar, cada uno se encarga de abrir su correspondiente ventana y cuentan con el siguiente código:

```
function pushbutton3_Callback(hObject, eventdata, handles)

    % Close: se encarga de cerrar la ventana 'proyecto' (Menú
    % Principal).
    close(proyecto);
    % Abre el archivo Modulo2.m (Reconstrucción de Imagen).
    Modulo2
```

En caso de querer abrir el módulo del "Generador" solo se tendrá que buscar en la programación de dicho botón y escribir el mismo código, cambiando solo el *Modulo2* por *Modulo1*.

3.2. Generador de sinogramas, phantom y matriz del sistema

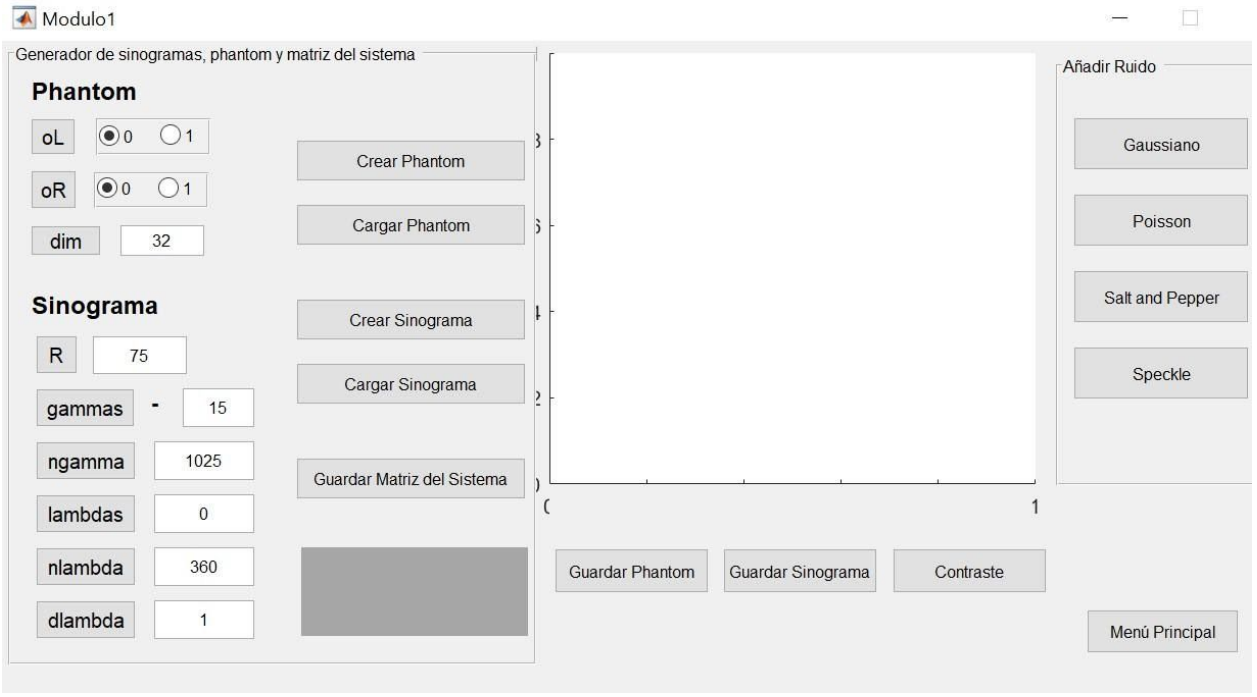


Figura 20: Botones del "Generador de sinogramas, phantom y matriz del sistema".

Aunque haya una gran cantidad de botones en este apartado (*Figura 20*), hay muchos que han sido programados de una manera muy similar, como es el caso del nombre de las variables, que como se ha explicado anteriormente, en la parte de "Etiquetado y almacenamiento de datos", sirven para que el usuario sepa en cada momento que significa cada variable.
Por ejemplo al presionar el botón R, en el cuadro de color gris más oscuro saldra el siginificado de este.

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

    % Envía a text10 el string que se le ha asignado (Distancia
    % desde...).
    set(handles.text10,'String','Distancia desde la fuente de
    Rayos-X hasta el centro ISO (Radio del escáner).');
```

Al lado de los nombres de las variables se puede apreciar una caja de texto en el cual el usuario puede escribir el valor que quiera, y para asignar el valor elegido por el usuario a la variable utilizamos el siguiente trozo de código:

```
    % En este caso, se recoge el valor que hay en la caja de
    % texto "rnum", dicho String se pasa a Double y es asignado
    % a la variable R.
    R = str2double(get(handles.rnum, 'String'));
```

Sin embargo al lado de las variables oL y oR no se ha puesto una caja de texto, si no una que contiene dos opciones, ya que, tanto oL, como oR, solo pueden valer o 0, que indicaría que dicha opción está desactivada, o 1, que indicaría que está activada. El código que utilizan es el siguiente:

```
    % De las dos opciones dadas el usuario escoge una, y luego,
    % como en el caso anterior, dicho String es pasado a Double
    % y asignado a la variable con el nombre correspondiente.
    getoL = get(handles.uibuttongroupoL, 'SelectedObject');
    oL = str2double(get(getoL, 'String'));
    getoR = get(handles.uibuttongroupoR, 'SelectedObject');
    oR = str2double(get(getoR, 'String'));
```

Luego, los botones con los que cuenta este módulo son los que llevan en su código las funciones creadas por el grupo de Investigación de Computación de Altas Prestaciones en Aplicaciones de Ingeniería Médica.

- **Crear Phantom:**

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

    getoL = get(handles.uibuttongroupoL, 'SelectedObject');
    oL = str2double(get(getoL, 'String'));
    getoR = get(handles.uibuttongroupoR, 'SelectedObject');
    oR = str2double(get(getoR, 'String'));

    dim = str2double(get(handles.dimnum, 'String'));

    xs = -12.8;          % Coordenadas de inicio de x.
    nx = dim;            % Número de cuadrícula en x.
    dx = abs(xs)*2/nx;   % Incremento de cuadrícula en x.
    ys = -12.8;          % Coordenadas de inicio de y.
    ny = dim;            % Número de cuadrícula en y.
    dy = abs(ys)*2/ny;   % Incremento de cuadrícula en y.
```

```
    % Creamos el fantoma.
    phantom = phantom_FORBILD_head_2D(oL,oR);
    im = generate_discrete_phantom(xs, dx, nx, ys, dy, ny,
    phantom);
    % Muestra la imagen (fantoma) en una escala de grises.
    imshow(im)

    % Envía la imagen al 'UserData' de la función de sinograma
    % para, más tarde poder utilizarla en la creación del
    % sinograma, a axes1 la enviamos para poder añadirle más
    % adelante el contraste.
    set(handles.axes1, 'UserData', im);
    set(handles.sinophantom, 'UserData', im);
```

- **Cargar Phantom:**

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

    % Abre una ventana para que el usuario escoja que fantoma
    % quiere abrir.
    [file, path, filterindex] = uigetfile('*.mat', 'Cargar
    Phantom');
    % Dicho archivo lo guarda en la variable "im".
    im=load(strcat(file));
    % Lo envía al apartado de axes donde se podrá visualizar.
    im=im.im;
    imshow(im)

    set(handles.sinophantom, 'UserData', im);
    set(handles.axes1, 'UserData', im);
    % Cogemos el nombre del archivo y lo enviamos a un recuadro
    % de texto estático para saber que fantoma se ha cargado.
    set(handles.nombre,'String',file);

    % Este código a sido creado para, a partir del nombre del
    % archivo, sacar el número de dimensiones de la imagen, y
    % enviarla a la caja de texto de "dim", para evitar errores
    % a la hora de crear el sinograma, aunque si el usuario
    % quiere también podría cambiarlo.
    [f, c] = size(file);
    b = [];

    for i = 1:c
         a = str2double(file(i));
         if isnumeric(a) == 1 && ~isnan(a)
              e = strcat(file(i));
```

```
            b = [b, e];
        end
    end

    str2double(strcat(b));
    set(handles.dimnum, 'String', b)
```

Figura 21: Nombre del fantoma cargado.

- **Crear Sinograma:**

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

    % Obtenemos el fantoma que previamente habíamos enviado.
    im=get(handles.sinophantom, 'UserData');

    % Obtenemos todas las variables que el usuario ha escogido.
    R = str2double(get(handles.rnum, 'String'));
    gammas =-str2double(get(handles.gammasnum,'String'))*pi/180;
    ngamma = str2double(get(handles.ngammanum, 'String'));
    dgamma = (2*abs(gammas))/ngamma;
    lambdas = str2double(get(handles.lambdasnum, 'String'));
    nlambda = str2double(get(handles.nlambdanum, 'String'));
    dlambda=str2double(get(handles.dlambdanum,'String'))*pi/180;
    xs = -12.8;
    dim = str2double(get(handles.dimnum, 'String'));
    nx = dim;
    dx = abs(xs)*2/nx;
    ys = -12.8;
    ny = dim;
    dy = abs(ys)*2/ny;

    % Este código nos dice, en caso de que el usuario no haya
    % escogido un valor de R y lo deje en 0, cuál es el valor
    % mínimo que debe valer dicha variable.
    if R == 0
        [R2, dgamma2, ngamma2] = CTparameters (xs, ys, R,
        dgamma, ngamma);
        R = R2;
```

```
          set(handles.rnum, 'String', R);
     end

     fano_joseph = fpj_joseph_FB_curve(im, R, xs, dx, nx, ys, dy,
     ny, lambdas, dlambda, nlambda, gammas, dgamma, ngamma);

     imshow(fano_joseph)
     set(handles.axes1, 'UserData', fano_joseph);
```

- **Cargar Sinograma:**

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

     [file, path, filterindex] = uigetfile('*.mat', 'Cargar
     Sinograma');
     fano_joseph=load(strcat(file));
     fano_joseph=fano_joseph.fano_joseph;
     imshow(fano_joseph)
     set(handles.nombre,'String',file);
```

- **Guardar Matriz del Sistema:**

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

     im=get(handles.sinophantom, 'UserData');

     R = str2double(get(handles.rnum, 'String'));
     gammas =-str2double(get(handles.gammasnum,'String'))*pi/180;
     ngamma = str2double(get(handles.ngammanum, 'String'));
     dgamma = (2*abs(gammas))/ngamma;
     lambdas = str2double(get(handles.lambdasnum, 'String'));
     nlambda = str2double(get(handles.nlambdanum, 'String'));
     dlambda=str2double(get(handles.dlambdanum,'String'))*pi/180;
     xs = -12.8;
     dim = str2double(get(handles.dimnum, 'String'));
     nx = dim;
     dx = abs(xs)*2/nx;
     ys = -12.8;
     ny = dim;
     dy = abs(ys)*2/ny;

     % Volvemos a obtener todos los datos necesarios, como hemos
     % hecho a la hora de crear el sinograma.
     A_real = [];
```

```
    for nview=1:nlambda
         lambda = lambdas+(nview-1)*dlambda;
         A = sparse(generate_fpjmatrix_singleview_joseph( R, xs,
         dx, nx, ys, dy, ny, lambda, gammas, dgamma, ngamma ));
         A_real = [A_real; A];
    end
    A=A_real;
    % Creamos el nombre del fichero.
    file = strcat('../Matrices', int2str(nlambda), '_',
    int2str(ngamma), 'detectores', int2str(dim), 'A.mat');
    % Guardamos A, la matriz del sistema.
    uisave('A', file);
```

- **Guardar Phantom:**

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

    im=get(handles.sinophantom, 'UserData');
    dim = str2double(get(handles.dimnum, 'String'));

    file = strcat('../IMref',int2str(dim),'head.mat');
    uisave('im', file);
```

- **Guardar Sinograma:**

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

    fano_joseph=get(handles.axes1, 'UserData');
    dim = str2double(get(handles.dimnum, 'String'));
    ngamma = str2double(get(handles.ngammanum, 'String'));
    nlambda = str2double(get(handles.nlambdanum, 'String'));
    R = str2double(get(handles.rnum, 'String'));

    file = strcat('../Sinograma', int2str(nlambda), '_',
    int2str(ngamma), 'detectores', int2str(dim), '_',
    int2str(R), 'R', '_', 'b.mat');
    uisave('fano_joseph', file);
```

- **Contraste:**

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

    im=get(handles.axes1, 'UserData');
    % Obtenemos la imagen que hemos guardado previamente, ya
    % sea el fantoma o el sinograma, y le añadimos el
    % contraste.
    imcontrast(imshow(im));
```

- **Añadir Ruido:**

En este caso se va a poner de ejemplo el código del botón de Gauss solamente, ya que el resto de botones cuentan con la misma programación. Lo único que cambia es que, a la hora de llamar a la función *imnoise,* en vez de escribir 'gaussian', se escribiría: 'poisson', 'salt & pepper' o 'speckle'.

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

    I=get(handles.axes1, 'UserData');
    % Se añade contraste a la imagen antes de añadir el ruido.
    im = 255*mat2gray(I);
    % Se cambia el formato de la imagen.
    imn=uint8(im);
    % Se añade ruido gaussiano.
    J = imnoise(imn,'gaussian');

    imshow(J)
```

- **Menú Principal:**

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

    close(Modulo1);
    proyecto
```

3.3. Reconstrucción de imagen

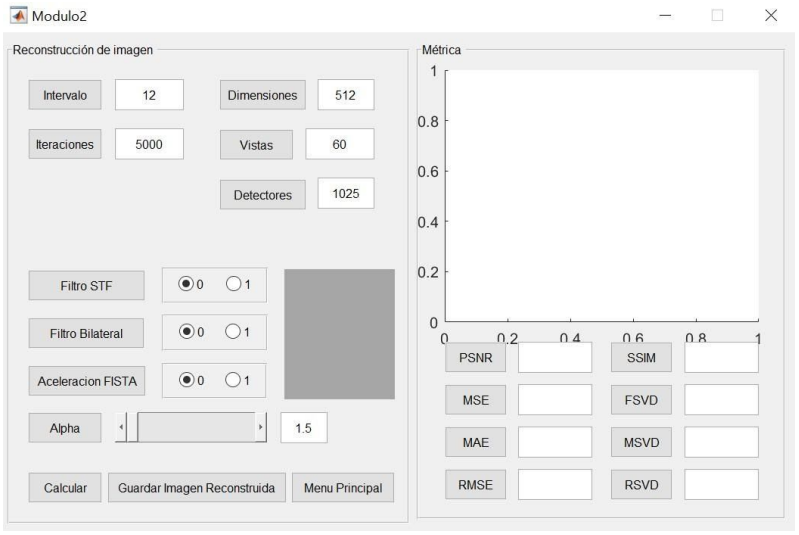


Figura 22: Botones de la "Reconstrucción de Imagen"

El código de las variables y de los filtros es el mismo que en el apartado anterior, con lo cual, para no repetir información, vamos a pasar directamente a ver aquellos que tienen una programación diferente.

- **Alpha:**

Puesto que el valor de Alpha puede variar entre 0 y 2 se ha decidido a poner un *slider* que varía entre dichos valores. Para que el usuario pueda saber el número del *slider* se ha añadido un recuadro de texto, en el cual, si se quisiera, también se podría cambiar el valor de alpha desde ahí.

```
function slider1_Callback(hObject, eventdata, handles)

    % Obtenemos el valor de el slider y se lo enviamos al
    % recuadro de alpha, llamado alphanum.
    valor = get(hObject, 'Value');
    set(handles.alphanum, 'String', valor)
    guidata(hObject, handles);

function alphanum_Callback(hObject, eventdata, handles)

    % En caso de querer cambiar el valor sin el slider este
    % método también hará el cambio.
    edit = get(hObject, 'String');
```

```
    set(handles.slider1, 'value',str2num(edit));
    guidata(hObject, handles);
```

- **Guardar Imagen Reconstruida:**

```
function pushbutton28_Callback(hObject, eventdata, handles)

    IM2=get(handles.axes1, 'UserData');
    % Se obtienen las variables para incluirlas en el nombre.
    interval = str2double(get(handles.intervalonum, 'String'));
    dim = str2double(get(handles.dimnum, 'String'));
    vistas = str2double(get(handles.vistasnum, 'String'));

    getfil = get(handles.grupostf, 'SelectedObject');
    fil = str2double(get(getfil, 'String'));
    getfilbi = get(handles.grupobilateral, 'SelectedObject');
    filbi = str2double(get(getfilbi, 'String'));
    getacc = get(handles.grupoacc, 'SelectedObject');
    acc = str2double(get(getacc, 'String'));
    file = strcat('./', 'ref', int2str(dim), 'x',
    int2str(vistas), 'x', int2str(interval), '-',
    int2str(fil), int2str(filbi), int2str(acc),'.mat');

    uisave('IM2', file);
```

- **Menú Principal:**

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

    close(Modulo2);
    proyecto
```

3.4. Métricas de calidad

- **Calcular:**

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

    interval = str2double(get(handles.intervalonum, 'String'));
    totaliter = str2double(get(handles.iteracionnum,'String'));
    dim = str2double(get(handles.dimnum, 'String'));
    vistas = str2double(get(handles.vistasnum, 'String'));
    detectores = str2double(get(handles.detectnum, 'String'));

    % Cargamos la matriz del sistema (A).
    A = load(strcat('Matrices', int2str(vistas), '_',
    int2str(detectores), 'detectores', int2str(dim), 'A.mat'));
    A = A.A;

    % El usuario escoge un sinograma compatible con la matriz.
    [file, path, filterindex] = uigetfile('*.mat',
    strcat('Cargar: Sinograma', int2str(vistas), '_',
    int2str(detectores), 'detectores', int2str(dim), '...'));
    fano_joseph=load(strcat(file));
    b1 = fano_joseph.fano_joseph;
    b = reshape(b1, detectores*vistas, 1);

    [n,m]=size(A);
    getfil = get(handles.grupostf, 'SelectedObject');
    fil = str2double(get(getfil, 'String'));
    getfilbi = get(handles.grupobilateral, 'SelectedObject');
    filbi = str2double(get(getfilbi, 'String'));
    getacc = get(handles.grupoacc, 'SelectedObject');
    acc = str2double(get(getacc, 'String'));

    alpha = str2double(get(handles.alphanum, 'String'));

    % Cargamos la imagen de referencia con la que compararemos
    % la imagen reconstruida.
    im=load(strcat('IMref', int2str(dim), 'head.mat'));
    IMref=im.im;

    xL = rand(m,1);
    t=1;
    xtemp=zeros(m,1);
    rel=inf;

    for iter=1:int16(totaliter/interval)
```

```
% LSQR

      [xL, flag, relres]=lsqr(A, b,1.e-6, interval,[],[], xL);

      if flag==0
      % Esto significa convergencia, en ese caso almacena la
      % solución y, guarda y visualiza la imagen reconstruida.

           % Pasa la solución que está almacenada en formato
           % vector (dim^2x1) unidimensional a una estructura
           % dimxdim.
           IM2=reshape(xL, dim, dim);

           % Cálculo de las métricas.
           psnr1=PSNRImages(IM2, IMref);
           mse=MSEImages(IM2, IMref);
           mae=MAEImages(IM2, IMref);
           rmse=RMSEImages(IM2, IMref);
           ssimv=ssim(IM2, IMref);
           % Métricas nuevas basadas en la descomposición en
           % valores singulares.
           fsvd=FSVDImages(IMref, IM2);
           msvd=MSVDImages(IMref, IM2);
           rsvd=RSVDImages(IMref, IM2);

           % Elementos a imprimir.
           p=[dim vistas interval iter fil filbi acc];
           p1=[ssimv psnr1 alpha];
           p2 = [mse mae relres];
           p3= [fsvd, msvd, rsvd];
            % En que formato imprimirlo.
            formatSpec = '%d %d %d %d %d %d %d';
            formatSpec1= ' %3.5f %3.5f %3.3f';
            formatSpec2= ' %3.5e %3.5e %3.5e\n';
            formatSpec3= ' %3.5e %3.5e %3.5e\n';

            imcontrast(imshow(IM2));
            break

      end

      xLold=xL;

      %% FILTRO STF

      % Optional:

      if fil==1
           w = max(abs(A'*(b-A*xL)));
```

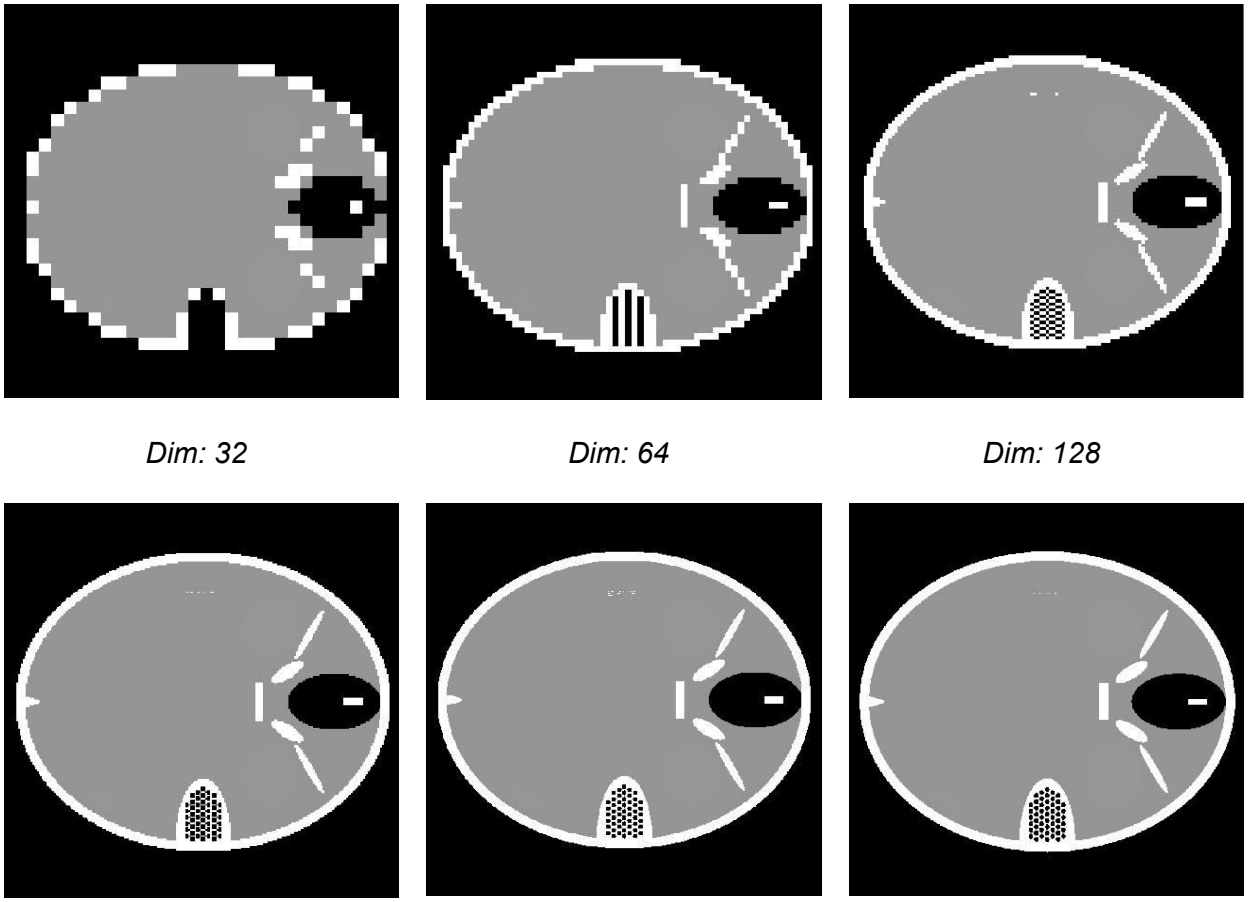
```
            xL = filtro(xL,w,alpha, dim);
        end

        %% FILTRO BILATERAL

        % Optional:

        if filbi==1
            xL=reshape(xL,dim,dim);
            xL=imbilatfilt(xL , 'NeighborhoodSize' ,5) ;
            xL = reshape(xL,dim*dim,1);
        end

        %% FISTA

        % Optional:

        if acc==1
            [ xL, t, xtemp ] = aceleracion( xL, t, xtemp );
        end

    end

    if flag~=0

        IM2=reshape(xLold, dim, dim);

        psnr1=PSNRImages(IM2, IMref);
        mse=MSEImages(IM2, IMref);
        mae=MAEImages(IM2, IMref);
        rmse=RMSEImages(IM2, IMref);
        ssimv=ssim(IM2, IMref);

        fsvd=FSVDImages(IMref, IM2);
        msvd=MSVDImages(IMref, IM2);
        rsvd=RSVDImages(IMref, IM2);

        p=[dim vistas interval iter fil filbi acc];
        p1=[ssimv psnr1 alpha];
        p2 = [mse mae relres];
        p3= [fsvd, msvd, rsvd];

        formatSpec = '%d %d %d %d %d %d %d';
        formatSpec1= ' %3.5f %3.5f %3.3f';
        formatSpec2= ' %3.5e %3.5e %3.5e\n';
        formatSpec3= ' %3.5e %3.5e %3.5e\n';

        imcontrast(imshow(IM2));
```

```
    end

    % Enviamos los resultados a los cuadros de texto
    % correspondientes para que el usuario pueda visualizarlos.
    set(handles.psnrnum,'String',psnr1);
    set(handles.msenum,'String',mse);
    set(handles.maenum,'String',mae);
    set(handles.rmsenum,'String',rmse);
    set(handles.ssimnum,'String',ssimv);
    set(handles.fsvdnum,'String',fsvd);
    set(handles.msvdnum,'String',msvd);
    set(handles.rsvdnum,'String',rsvd);

    set(handles.axes1, 'UserData', IM2);
```

4. Verificación de la herramienta

A continuación se van a mostrar algunos resultados, a modo de ejemplo, para poder observar el correcto funcionamiento de la aplicación creada.
Lo que se va a mostrar serán comparaciones de las imágenes de referencia (fantomas), con las imágenes reconstruidas a partir de las distintas simulaciones, así se podrá ver cómo afectan, tanto los cambios de variables a los fantomas y sinogramas creados, como la adición de ruido.
En el caso de las imágenes reconstruidas se llevará a cabo el mismo procedimiento, y también se comparará el uso de los filtros, además de comparar las tablas de las métricas de calidad de cada imagen.

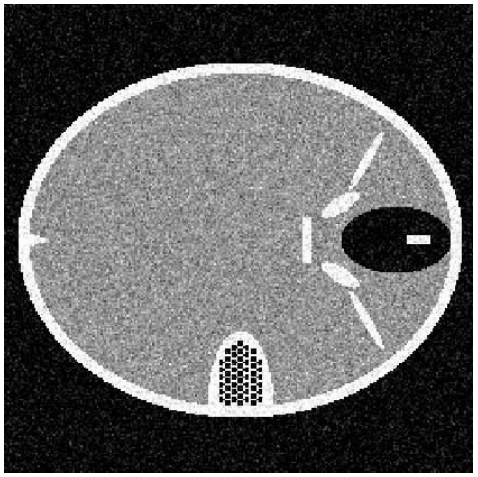
4.1. Fantomas



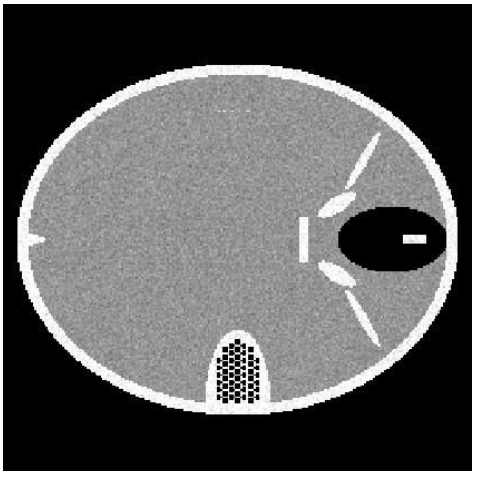
Dim: 32 *Dim: 64* *Dim: 128*

Dim: 256 *Dim: 512* *Dim: 720*

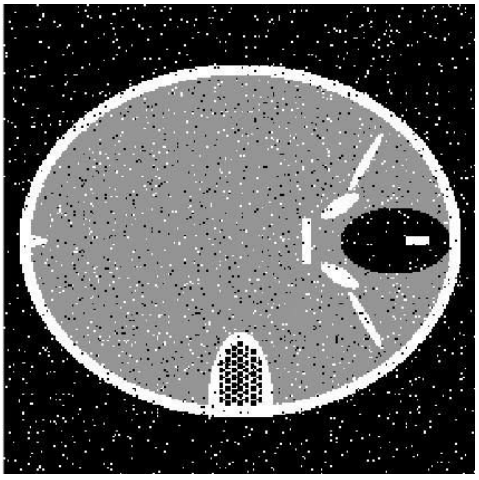
Figura 23: Resultados Fantoma (sin ruido).



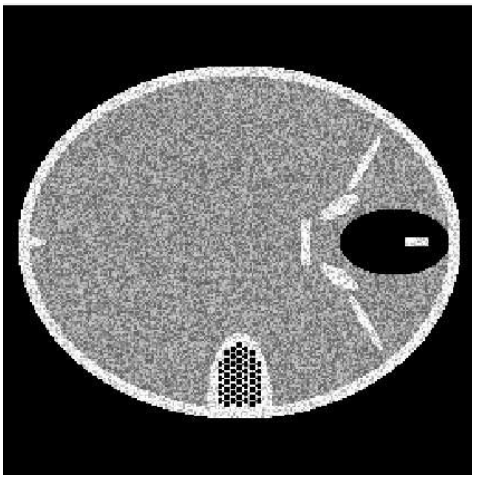
Gaussiano



Poisson



Salt & Pepper



Speckle

Figura 24: Resultados Fantoma (dim.: 256, con ruido).

Como se puede observar en las imágenes de la *Figura 23*, cuanto mayor sea el número de la dimensión escogida por el usuario, mayor definición se tendrá del fantoma.

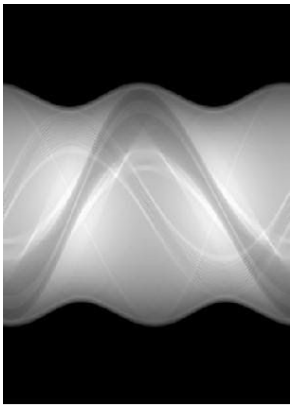
Por otra parte, en la *Figura 24*, podemos ver cómo afectan los distintos tipos de ruido a una misma imagen, de dimensión 256.

4.2. Sinogramas

En el apartado de sinogramas hay varios datos que podemos modificar, en los siguientes ejemplos se podrá ver cómo afecta el cambio del valor del radio del escáner, de las vistas y de los detectores. En la *Figura 25*, como imagen de referencia (fantoma) se utilizará una cuya dimensión sea de 256.

R=45;
Vistas= 360;
Detectores=1025.

R=75;
Vistas= 180;
Detectores=1025.



R=75;
Vistas= 360;
Detectores=600.

R=75;
Vistas= 360;
Detectores=1025.

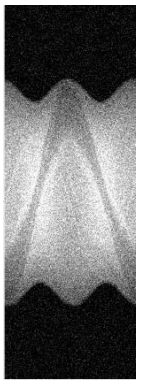
Figura 25: Resultados del Sinograma (sin ruido).

Dependiendo del valor que se decida cambiar:

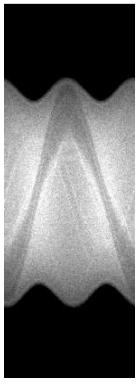
- Radio del escáner: al cambiar la R, comparando el primer sinograma con el último, se puede observar que, que su amplitud varia.

- Vistas: en el caso de la variable encargada de las visas, nlambda, cuanto mayor sea más información se podrá obtener del sinograma, ya que se obtendrá la señal de una vuelta completa del escáner.

- Detectores: el número de detectores permite que se capte más o menos información, así que cuantos más detectores se tengan, mejor será la reconstrucción.

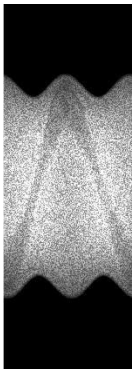


Gaussiano



Poisson

Salt & Pepper

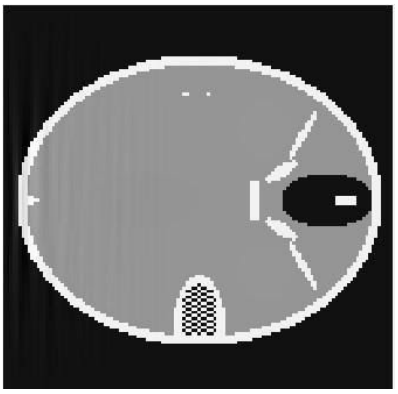


Speckle

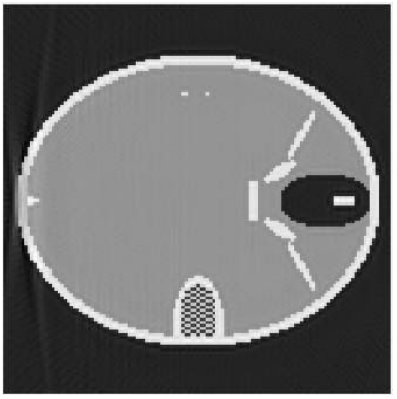
Figura 26: Resultados Sinogramas (con ruido).

Como en el apartado de los fantomas, en los resultados de la *Figura 26* también se puede observar cómo afecta el ruido a los sinogramas creados.
En este caso los valores son: R=75, Vistas =360 y Detectores=1025.

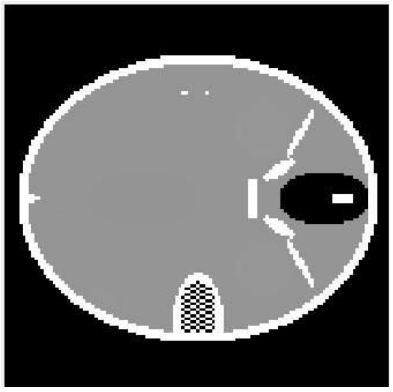
4.3. Imágenes reconstruidas



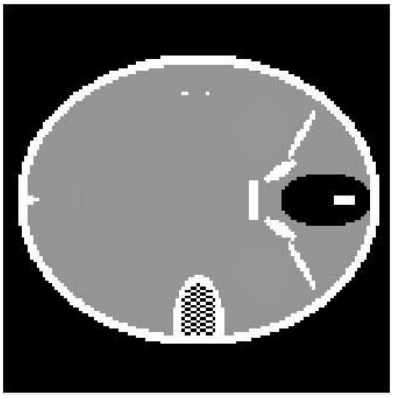
Dim=128; Vistas=180; Detectores=600



Filtro STF



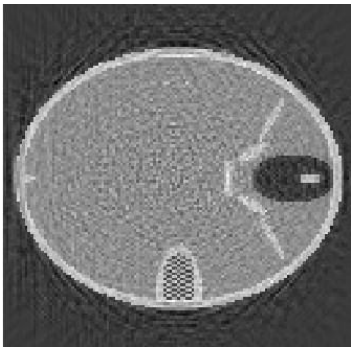
Filtro Bilateral



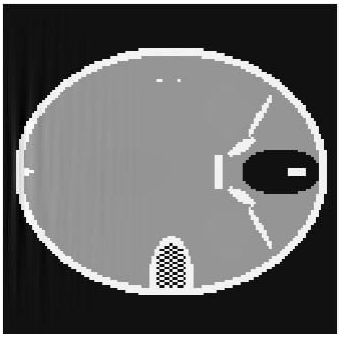
Aceleración FISTA

Figura 27: Resultados Imágenes Reconstruidas.

Con la ayuda de los filtros en la *Figura 27* se puede observar como la imagen reconstruida se ve con menos ruido y sin deformaciones en la imagen. Cabe destacar que cuantos más detectores se tengan, mejor será la resolución de la imagen reconstruida, como se puede apreciar en la Figura 28.



Dim=128; Vistas=180; Detectores=100



Dim=128; Vistas=180; Detectores=600

Figura 28: Diferencia entre detectores en imagen reconstruida.

No se aceptará como correcto el resultado obtenido en imagen de la izquierda de la *Figura 28*, ya que el número mínimo de detectores que una imagen debería tener para que se pueda aceptar dicho resultado es de 600, como en la imagen de la derecha.

Los resultados de las métricas de calidad se encuentran en el anexo adjunto.

5. Conclusiones

En este trabajo, se ha llevado a cabo el diseño e implementación de una aplicación, con la finalidad de poderse utilizar en un futuro para generar una base de datos asociados a la imagen médica de Tomografía Computarizada (TC). En particular, se podrá generar una base de datos de sinogramas, que pueden ayudar a entrenar una red neuronal, para obtener una solución inicial de los procesos algebraicos iterativos utilizando técnicas de inteligencia artificial.

La aplicación implementada cuenta con varios módulos que pretenden facilitar la tarea de crear las imágenes de referencia, sinogramas y matrices del sistema, ya que el usuario solo tendrá que elegir los valores que quiera simular, relacionados con las dimensiones y con las características del escáner, y guardar los resultados obtenidos. A partir de los datos generados, el usuario podrá reconstruir y filtrar la imagen médica. También podrá compararla con la imagen de referencia utilizando las métricas de calidad disponibles, de esa forma se dispondrá de una evaluación objetiva de la calidad. Con dicha información y la visualización de la imagen reconstruida podrá verificarse su calidad y podrá ayudar a diagnosticar al paciente.

La aplicación se ha diseñado para que el usuario sepa en cada momento que variables está cambiando y a que afectan, ya que dichas variables se han creado en botones que proporcionan dicha información al usuario con solo pulsarlos.

Por último, para dicha implementación se ha utilizado el entorno MATLAB ya que es un entorno de programación muy amigable que nos permite modificar fácilmente el programa generado, además las funciones base estaban implementadas en dicho lenguaje de programación.

6. Referencias

[1] Lauritsch, G., Bruder, H. *Head Phantom.* (http://imp.uni-erlangen.de)

[2] Chillarón, M., (2016, Septiembre). *Estudio, Análisis y Parametrización de la Combinación del Método LSQR con Filtro de Ruido Gaussiano para Reconstrucción de Imágenes de TAC.*

[3] Chillarón, M., Vidal, V., Quilis, J. D. S, Blanquer, I., & Verdú, G. (2017, June). *Combining Grid Computing and Docker Containers for the Study and Parametrization of CT Image Reconstruction Methods. In ICCS (pp. 1195-1204)*

[4] Paige, C., Saunders, M. (1982). *LSQR: An algorithm for sparse linear equations and sparse least squares. ACM Transactions on Mathematical Software (TOMS), 8(1), 43-71.*

[5] Yu, W., & Zeng, L. (2014). *A novel weighted total difference based image reconstruction algorithm for few-view computed tomography. PLoS One, 9(10), e109345.*

[6] Beck, A., & Teboulle, M. (2009). *A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. SIAM journal on imaging sciences, 2(1), 183-202.*

[7] Tomasi, C., & Manduchi, R. (1998, January). *Bilateral filtering for gray and color images. In Sixth international conference on computer vision (IEEE Cat. No. 98CH36271) (pp. 839-846). IEEE.*

[8] Chillarón, M., Vidal, V., & Verdú, G. (2020). *Evaluation of image filters for their integration with LSQR computerized tomography reconstruction method. PLoS One, 15(3), e0229113.*

[9] MathWorks. *Métricas de calidad de imagen.* (https://es.mathworks.com)

[10] Hore, A., & Ziou, D. (2010, August). *Image quality metrics: PSNR vs. SSIM. In 2010 20th international conference on pattern recognition (pp. 2366-2369). IEEE.*

[11] Chai, T., & Draxier, R. R. (2014). *Root mean square error (RMSE) or mean absolute error (MAE)? - Arguments against avoiding RMSE in the literature. Geoscientific model development, 7(3), 1247-1250.*

[12] Esmaeilpour, M., Mansouri, A., & Mahmoudi-Aznaveh, A. (2013, September). *A new SVD-based image quality assessment. In 2013 8th Iranian Conference on Machine Vision and Image Processing (MVIP) (pp. 370-374). IEEE.*

[13] Shnayderman, A., Gusev, A., & Eskicioglu, A. M. (2006). *An SVD-based grayscale image quality measure for local and global assessment. IEEE transactions on Image Processing, 15(2), 422-429.*

[14] Mansouri, A., Aznavh, A. M., Torkamani-Azar, F., & Jahanshahi, J. A. (2009). *Image quality assessment using the singular value decomposition theorem. Optical Review, 16(2), 49-53.*

[15] Jain, A. K. *Fundamentals of digital image processing. Englewood Cliffs, NJ: Prentice Hall,. (1989).*

[16] MathWorks. *Añadir ruido a la imagen.* (https://es.mathworks.com)

7. Anexos

En este apartado se va a verificar que el código de las métricas funciona adecuadamente. Para ello se van a utilizar diversas imágenes reconstruidas sin utilizar y utilizando los filtros.

Se puede observar el incremento del valor de la métrica PSNR cuando se utilizan los diversos filtros, obteniéndose un incremento de prácticamente el doble cuando se utiliza el filtro bilateral. Otra de las métricas que más se utiliza en calidad de imagen, el SSIM, toma el valor de prácticamente 1 cuando se utilizan filtros.

Los valores que se obtienen son los esperados, por tanto la implementación es la adecuada.

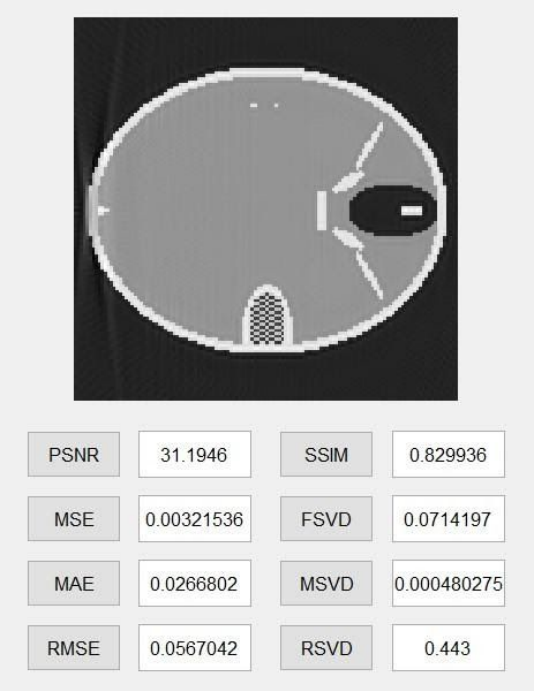
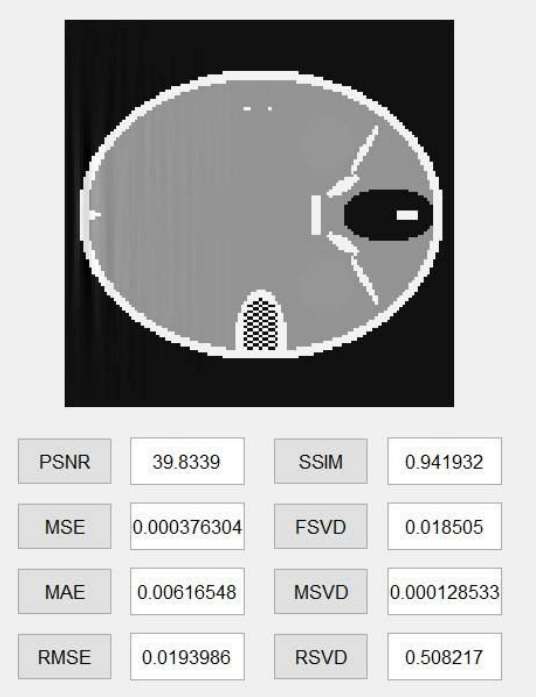
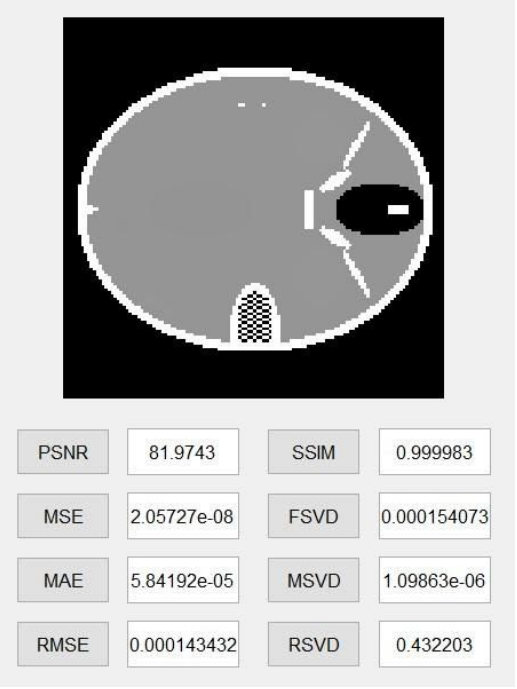


Imagen reconstruida sin filtros.



PSNR	39.8339	SSIM	0.941932
MSE	0.000376304	FSVD	0.018505
MAE	0.00616548	MSSVD	0.000128533
RMSE	0.0193986	RSVD	0.508217

Imagen reconstruida con filtro STF.



PSNR	81.9743	SSIM	0.999983
MSE	2.05727e-08	FSVD	0.000154073
MAE	5.84192e-05	MSSVD	1.09863e-06
RMSE	0.000143432	RSVD	0.432203

Imagen reconstruida con filtro bilateral.

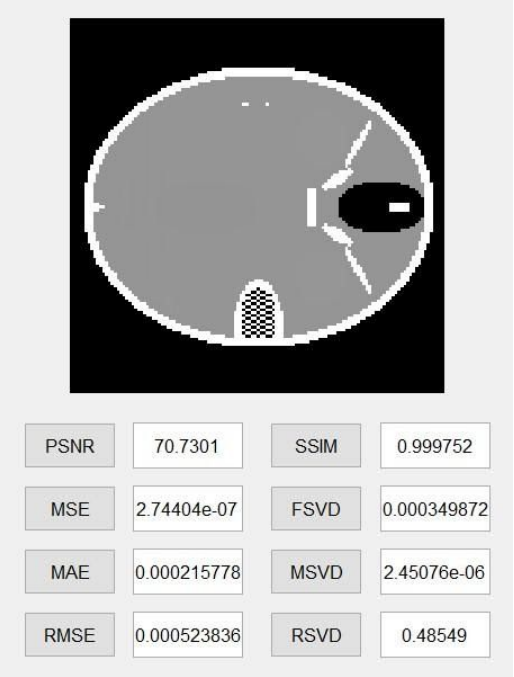


Imagen reconstruida con aceleración FISTA.