Imágenes Médicas - Práctica 3: Reconstrucción de Imágenes Tomográficas: Método Directo

Juan Pablo Morales[1a]

[a] *Instituto Balseiro - Centro Atómico Bariloche. Av Bustillo 9500, 8400 Bariloche, Río Negro, Argentina*

12 de marzo de 2024

1. Introducción

La tomografía computarizada (TC) es una técnica de imagen médica que utiliza rayos X para obtener imágenes transversales del cuerpo humano, con una alta precisión en la visualización de estructuras anatómicas. Esta técnica juega un papel fundamental en el diagnóstico y seguimiento de diversas patologías.

La simulación de imágenes de TC es una herramienta valiosa para el desarrollo y evaluación de algoritmos de reconstrucción y análisis de imágenes. CTSim es un programa de código abierto que permite realizar simulaciones realistas de imágenes de TC, incluyendo la generación de fantomas virtuales con diferentes características.

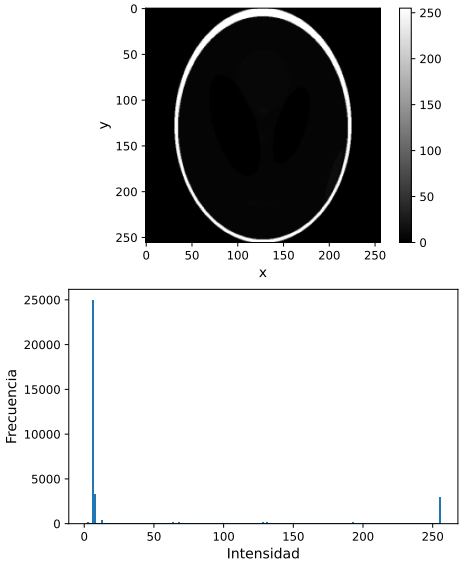

Figura 1: Imagen del fantoma Shepp-Logan simulado y rasterizado mediante CTSim.

2. Fantoma Shepp-Logan

El software CTSim permite simular un fantoma conocido como Shepp-Logan, que representa un cerebro. A partir de esta representación vectorial del fantoma, CTSim también puede generar una imagen rasterizada, es decir, un conjunto de píxeles que conforman la imagen del fantoma. La imagen rasterizada inicial (ver Figura 1) presenta un nivel de grises determinado por el histograma asociado. Sin embargo, este contraste no es óptimo para la visualización de las diferentes estructuras del fantoma.

Para mejorar el contraste, se utiliza el software ImageJ para realizar una ecualización del histograma. La imagen resultante (ver Figura 2) presenta un contraste significativamente mejor, permitiendo distinguir con mayor claridad los distintos objetos que componen el fantoma.

[1] juan.morales@ib.edu.ar

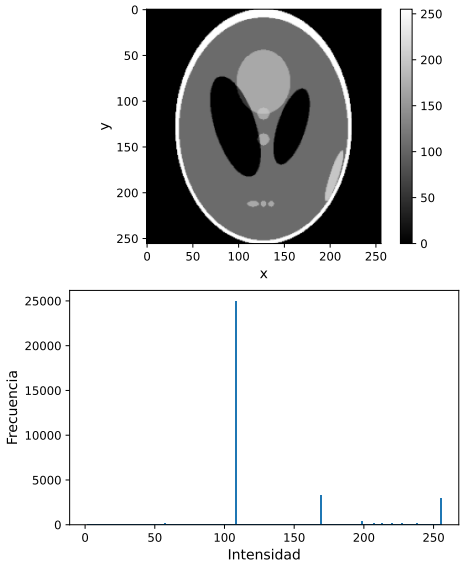

Figura 2: Imagen del fantoma ecualizado Shepp-Logan a partir de la imagen simulada y rasterizada mediante CTSim.

3. Proyección con CTSim

Se calcularon las proyecciones del fantoma mediante la transformada de Radón de el fantoma utilizando los parámetros por defecto de CTSim, esta se realizó con una geometría paralela utilizando 367 detectores y 320 ángulos. El resultado obtenido se muestra en la Figura 3. Estas proyecciones representan lo que se llama el Sinograma del fantoma.

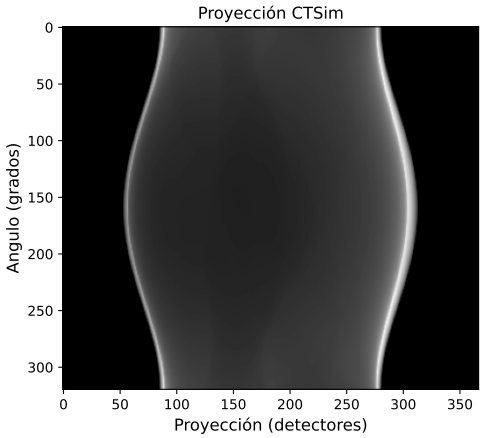

Figura 3: Sinograma del fantoma obtenido mediante el software CTSim.

4. Reconstrucción mediante retroproyección filtrada

A partir del sinograma presentado en la Figura 3, se empleó la técnica de retroproyección filtrada (FBP) para reconstruir la imagen original del fantoma. Esta técnica consiste en proyectar los datos del sinograma sobre cada punto del espacio de la imagen, utilizando un filtro que realza las frecuencias espaciales relevantes.

En este caso, se utilizó un filtro h cuya transformada de Fourier es $\mathcal{F}(h)(\rho) = |\rho|$. Sin embargo, como la función $|\cdot|$ no es de cuadrado integrable, se truncó a una frecuencia mayor que la de Nyquist. Este tipo de filtro se conoce como $|k|$Bandlimit, de este se obtuvo la Figura 4.

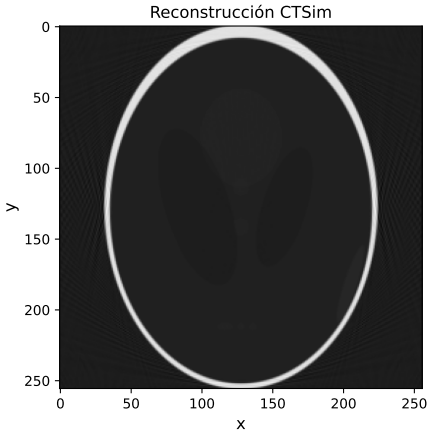

Figura 4: Reconstrucción a partir del sinograma del fantoma utilizando de la técnica de retroproyección filtrada.

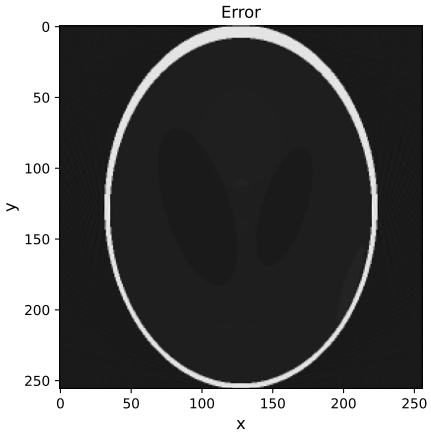

Figura 5: Diferencia entre la imagen original y la imagen reconstruida a partir del sinograma de la misma.

Para evaluar la calidad de la reconstrucción, calculamos la diferencia entre la imagen original (Figura 1) y la imagen reconstruida (Figura 4). Esta operación nos proporciona la imagen mostrada en la figura 5. Es importante destacar que esta imagen de error se calculó después de realizar una normalización por la media de cada imagen. Al calcular el error cuadrático medio, encontramos que el error de reconstrucción es de 2,50.

5. Error de reconstrucción

Se investigó el error de reconstrucción de las imágenes utilizando diferentes parámetros en la técnica de retroproyección filtrada, incluyendo el número de detectores y el número de ángulos utilizados. Para este análisis, se empleó la imagen ecualizada del fantoma Shepp-Logan y su sinograma, como se muestra en la Figura 6.

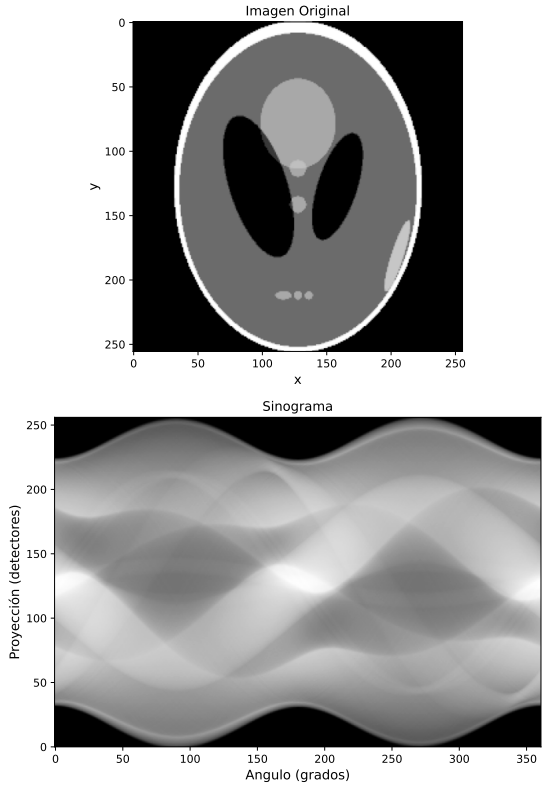

Figura 6: Imagen del fantoma ecualizada mediante ImageJ y su sinograma correspondiente.

En primer lugar, se procedió a variar el número de detectores utilizados. Se seleccionaron 32, 128, 256, 512, 720 y 1024 detectores, manteniendo constante el número de ángulos en 360. Se calculó el sinograma para cada configuración y se realizó la reconstrucción correspondiente. Luego, se evaluó el error cuadrático medio de la resta entre la imagen original ecualizada del fantoma y su reconstrucción. Cabe destacar que antes de calcular el error, se normalizó cada imagen por su media. Los resultados obtenidos se presentan en la Figura 7.

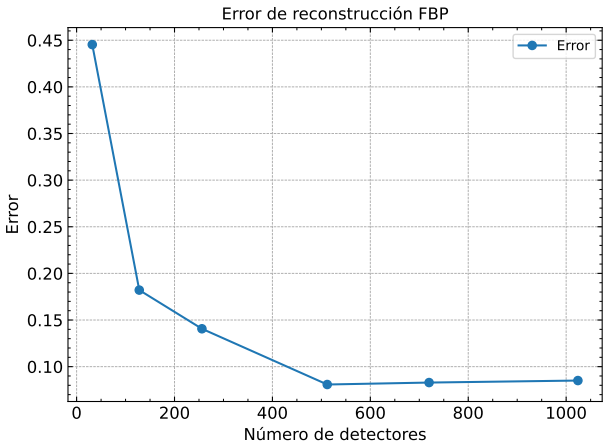

Figura 7: Error cuadrático medio entre la imagen original del fantoma ecualizada y su reconstrucción, en función del número de detectores utilizados.

En esta figura se puede observar que el error cuadrático medio disminuye a medida que aumenta el número de detectores utilizados en la reconstrucción. Sin embargo, este efecto tiene un límite; después de alcanzar los 512 detectores, el error apenas varía.

Posteriormente, se llevó a cabo un análisis similar manteniendo constante el número de detectores en 256 y variando el número de ángulos entre 1 y 1000. Los resultados, representados en la Figura 8, indican que el error disminuye rápidamente con el aumento del número de ángulos utilizados en la reconstrucción, aunque tiende a estabilizarse.

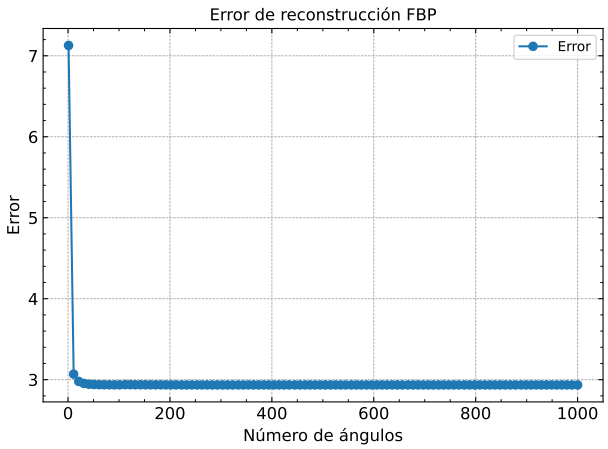

Figura 8: Error cuadrático medio entre la imagen original del fantoma ecualizada y su reconstrucción, en función del número ángulos que se utilizan.

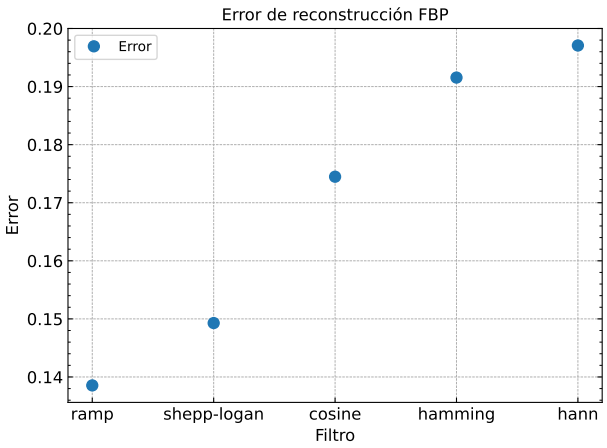

Figura 9: Error cuadrático medio entre la imagen original del fantoma ecualizada y su reconstrucción, en función del filtro utilizado.

Esto indica que tanto el número de detectores como el número de ángulos son factores críticos que influyen en la precisión de la reconstrucción en la técnica de retroproyección filtrada. Sin embargo, existe un punto de rendimiento óptimo, más allá del cual aumentar estos parámetros no proporciona mejoras significativas en la calidad de la reconstrucción.

Se procedió a analizar otro parámetro relevante en la reconstrucción de imágenes: los filtros aplicados durante el proceso. Se evaluaron los filtros ramp, shepp-logan, cosine, hamming y hann. Manteniendo un conjunto de 256 detectores y 360 ángulos, se llevó a cabo el mismo procedimiento que en los análisis anteriores para determinar el error de reconstrucción.

Los resultados se muestran en la Figura 9. Se observa un ligero aumento en el error de reconstrucción al utilizar filtros diferentes a los empleados en la transformada de Radón inversa, que sería el ramp, que se encarga de restaurar la imagen original.

6. Ruido sobre el sinograma

Se realizó un análisis del efecto del ruido en la reconstrucción del sinograma. Para ello, se agregó ruido blanco gaussiano con una varianza de $\sigma = 6$ al sinograma original mostrado en la Figura 6, lo que generó el sinograma con ruido representado en la Figura 10. Es evidente que al introducir ruido en el sinograma, la reconstrucción también se verá afectada. Se aplicaron los mismos filtros utilizados anteriormente (ramp, shepp-logan, cosine, hamming y hann) para reconstruir la imagen, obteniendo los resultados mostrados en la Figura 11.

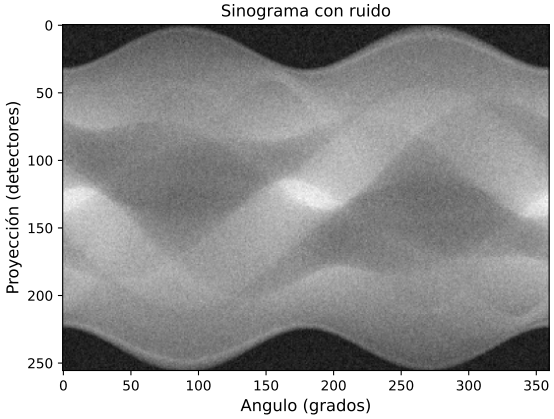

Figura 10: Sinograma de la imagen del fantoma Shepp-Logan equalizado.

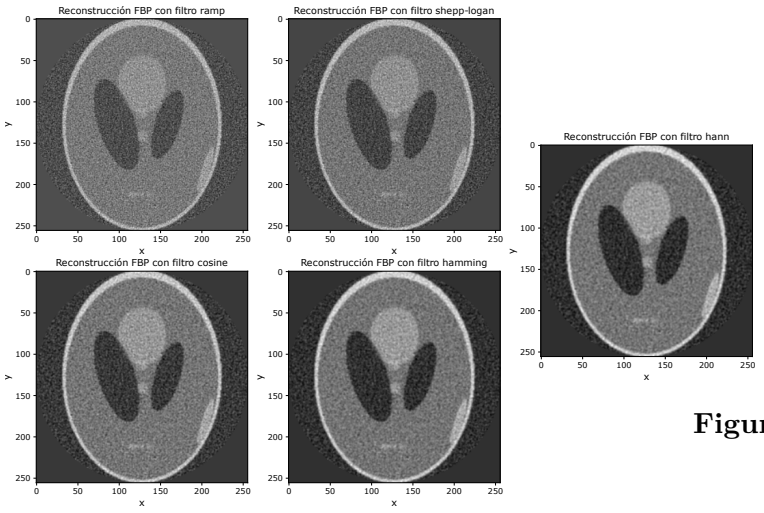

Figura 11: Reconstrucción de la imagen del fantoma Shepp-Logan mediante FBP utilizando distintos filtros.

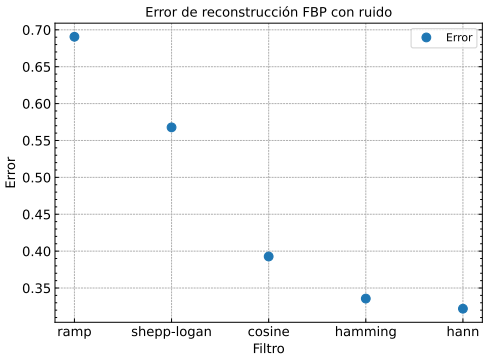

Figura 12: Error de reconstrucción entre la imagen original y la imagen reconstruida mediante FBP en función del filtro utilizado.

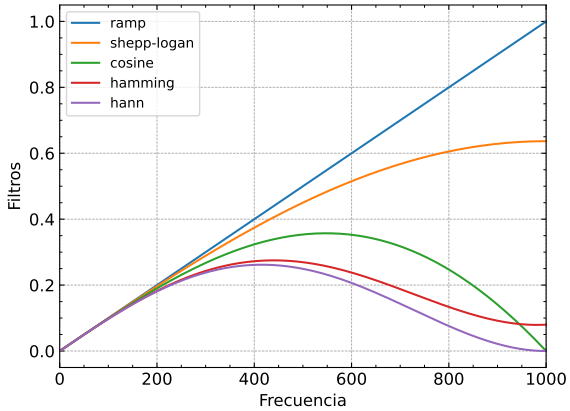

Figura 13: Filtros utilizados para la reconstrucción en función de la frecuencia

Para evaluar el efecto de los filtros se calculó el error cuadrático medio después de normalizar las imágenes por su media. Los resultados se presentan en la Figura 12. Se observa un aumento en el error en el siguiente orden: ramp, shepp-logan, cosine, hamming y hann. Este comportamiento se debe a la forma de los filtros en el espacio de Fourier, como se ilustra en la Figura 13. En esta figura se aprecia que el orden en el que se presentan los filtros está relacionado con su comportamiento en frecuencias altas.

Frente a este tipo de ruido, debemos tomar decisiones sobre la similitud deseada entre el filtro utilizado y el de la transformada de Radón inversa para evitar artefactos no deseados y considerar cuánto se amplificará este ruido de alta frecuencia durante el proceso de reconstrucción debido a esto.

7. Proyecciones de objeto circular

Se creó una imagen de tamaño 100x100 que contiene únicamente un círculo con un diámetro de 5 píxeles, centrado en el origen, como se muestra en la Figura 14. Luego, se llevaron a cabo proyecciones utilizando 8, 16, 32 y 64 ángulos.

Posteriormente, se realizaron reconstrucciones sin aplicar ningún filtro en el algoritmo de retroproyección, y luego aplicando el filtro de rampa. Los resultados se presentan en la Figura 15.

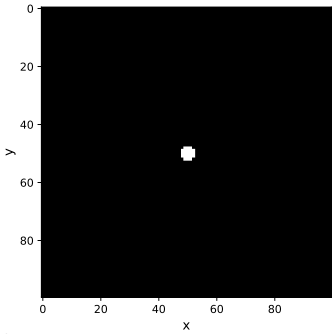

Figura 14: Imagen de tamaño 100x100 píxeles que contiene un único círculo de diámetro $D = 5$ píxeles.

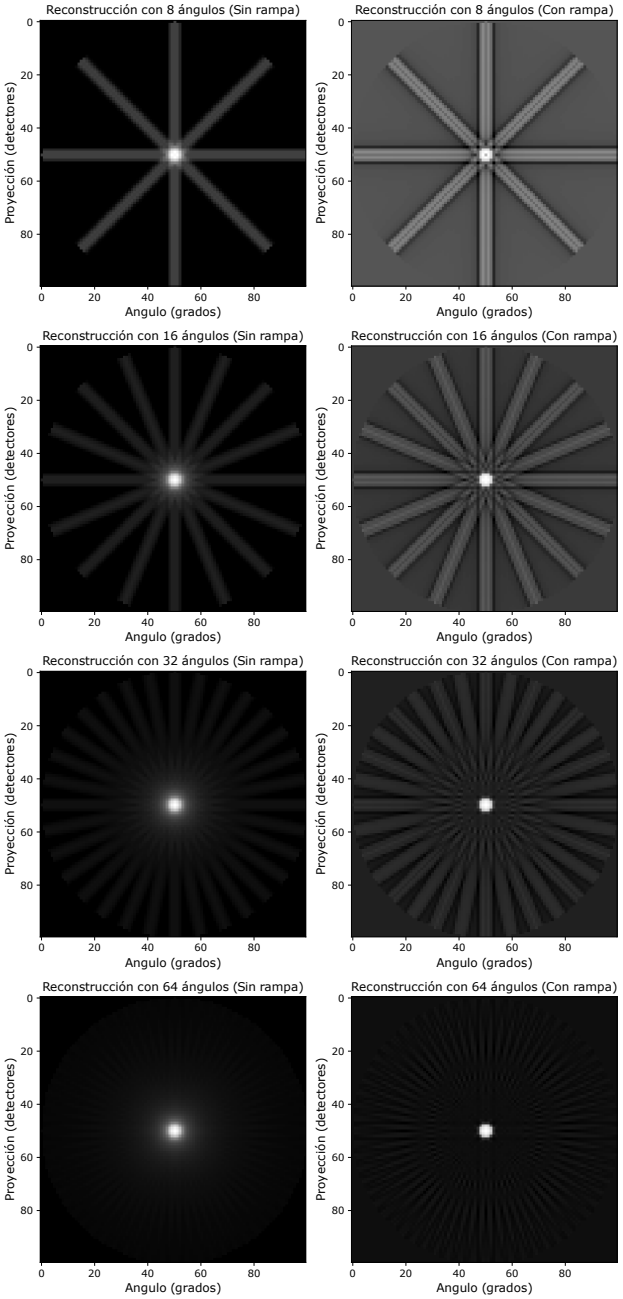

Figura 15: Comparación de las distintas reconstrucciones con y sin filtro rampa y utilizando una distinta cantidad de ángulos.

De estos resultados, podemos extraer dos conclusiones. Primero, al utilizar un número reducido de proyecciones, surgen artefactos en forma de líneas que atraviesan la imagen, y su cantidad coincide con la cantidad de proyecciones utilizadas. Además, estas líneas tienden a volverse más delgadas a medida que se aumenta el número de proyecciones.

Otra conclusión importante es que se observa un efecto notable de difuminado cuando no se

aplica ningún filtro durante el proceso de retroproyección para la reconstrucción de la imagen. Esto se evidencia en la presencia de un halo alrededor del círculo en las imágenes reconstruidas sin el uso del filtro.

8. Defectos en los detectores

Se analiza el caso de reconstrucciones de sinogramas para los cuales haya fallado algún detector. En primer lugar se debe simular cómo se vería el sinograma si uno de los detectores estuviera defectuoso. La presencia de un detector defectuoso se reflejaría como una línea negra en el sinograma para todos los ángulos, como se ilustra en la Figura 16, que muestra el sinograma del fantoma ecualizado.

Se llevaron a cabo reconstrucciones utilizando diferentes detectores defectuosos, identificados por las columnas del sinograma. Los resultados se muestran en la Figura 17. Se observa que las reconstrucciones presentan artefactos en forma de anillos, cuyo radio depende de la proximidad del detector defectuoso al centro. Específicamente, cuando el detector central está defectuoso (en este caso, el detector 128), la mayor parte de la imagen aparece casi completamente blanca, lo que imposibilita la distinción de las estructuras presentes.

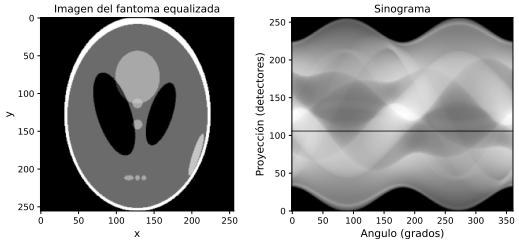

Figura 16: Sinograma obtenido a partir del fantoma ecualizado simulando un detector defectuoso.

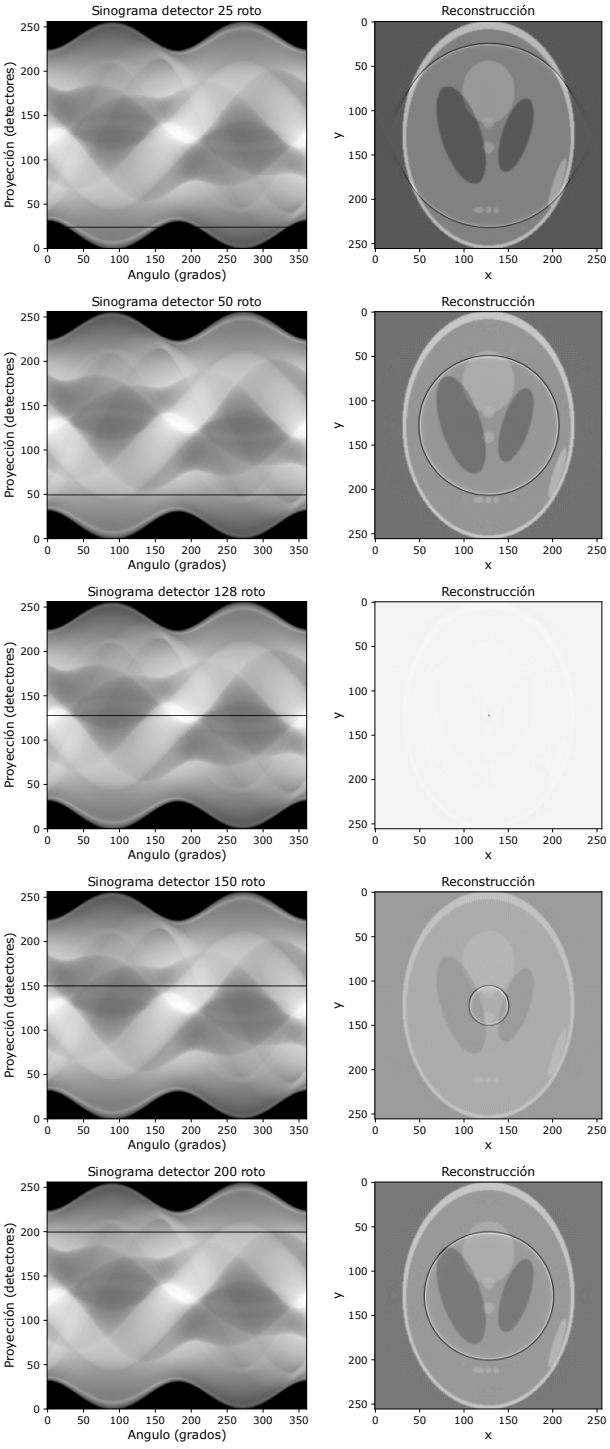

Figura 17: Sinogramas con detectores rotos y sus respectivas reconstrucciones.

9. Código

```
# %%
import cv2
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import AutoMinorLocator
from matplotlib.ticker import MultipleLocator
import os.path
from skimage.data import shepp_logan_phantom
from skimage.transform import radon, rescale
from skimage.transform import iradon

# %% [markdown]
# # Ejercicio 1
# * Usando el programa ctsim generar un fantoma Shepp-Logan. Rasterizarlo para observar los niveles de gris. Alternativamente, usar radon-skimage.py. Almacenarlo como PGM-Ascii. Usando imagej o una version modificada de read-write.c, ecualizar el histograma y ver la imagen generada

# %%
def read_pgm_file(file_name, path = r'C:\Users\Propietario\Desktop\ib\5-Maestría\Imágenes Médicas\Practicas\Practica3\Ejercicio1\\'):
    """ Función que lee un archivo .pgm

    Args:
        file_name (str): Nombre del archivo .pgm
        path (str): Ruta donde se encuentra el archivo .pgm

    Returns:
        img (np.array): Imagen en formato de array de numpy
    """
    # Para abrir la imagen hay que cambiar el directorio a la carpeta donde está la imagen
    data_dir = os.path.dirname(path)

    # Vemos si la imagen existe
    file_path = os.path.join(data_dir, file_name)
    assert os.path.isfile(file_path), 'file \'{0}\' does not exist'.format(file_path)

    # Leemos la imágen utilizando OpenCV, con la bandera cv2.IMREAD_GRAYSCALE que indica que se leera en escala de grises
    img = cv2.imread(file_name,flags=cv2.IMREAD_GRAYSCALE)

    # Si la imagen se leyo correctamente, mostramos su tamano sino mostramos un mensaje de error
    if img is not None:
        print('img.size: ', img.size)
    else:
        print('imread({0}) -> None'.format(file_path))
```

```
43
44      return img
45
46  # %%
47  def show_img_hist(im, fileout = 'Out_ImagenAHist.pdf'):
48      """
49      Función que calcula y muestra el histograma de la imagen cargada
50
51      Args:
52          im (np.array): Imagen a la que se le calculará el histograma
53
54      Returns:
55          None: Simplemente muestra el histograma junto con la imágen
56      """
57
58      # Calculamos el valor mínimo y máximo de la intensidad de la imagen
59      vmin = np.amin(im)
60      vmax = np.max(im)
61
62      print("Intensity Min: {}   Max:{}".format(vmin,vmax))
63
64      # Calculamos el número de niveles de intensidad
65      L = vmax - vmin
66      print("Number of Levels: {}".format(L))
67      fig = plt.figure(figsize=(6,8))
68      ax1 = plt.subplot(2, 1, 1)
69      ax2 = plt.subplot(2, 1, 2)
70      imgplot = ax1.imshow(im,cmap='gray', vmin=vmin, vmax=vmax)
71      ax1.set_xlabel('x', fontsize = 12)
72      ax1.set_ylabel('y', fontsize = 12)
73      fig.colorbar(imgplot, ax=ax1)
74      # cv2.imshow(infile,img)
75      # cv2.waitKey(0)
76
77      # ravel() hace que la matriz de la imagen se convierta en un vector
            de una sola dimensión y luego calculamos el histograma
78      hist, bin_edges = np.histogram(im.ravel(),bins=L)
79      print("Histogram: ", len(hist))
80      print("Bin Edges: ", len(bin_edges))
81      ax2.bar(bin_edges[2:], hist[1:])
82      ax2.set_xlabel('Intensidad', fontsize = 12)
83      ax2.set_ylabel('Frecuencia', fontsize = 12)
84      plt.savefig(fileout)
85      plt.show()
86
87  # %%
88  file_fantoma = 'FantomaSheppLogan.pgm'
89  img = read_pgm_file(file_fantoma)
90
91  # %%
92  show_img_hist(img, 'FantomaSheppLoganHist.pdf')
93
```

```
# %%
file_eqFantoma = 'FantomaEqualizado.pgm' # Fantoma equalizado con
    ImageJ dejando el 0.35% de saturados
img_eq = read_pgm_file(file_eqFantoma)

# %%
show_img_hist(img_eq, 'FantomaEqSheppLoganHist.pdf')

# %% [markdown]
# # Ejercicio 2
# * Calcular las proyecciones usando los valores default de ctsim

# %%
file_projection = 'ProjectionCTSimSheppLogan.pgm'
img_projection = read_pgm_file(file_projection)
plt.imshow(img_projection, cmap='gray')
plt.title('Proyección CTSim', fontsize = 12)
plt.ylabel('Angulo (grados)', fontsize = 12)
plt.xlabel('Proyección (detectores)', fontsize = 12)
plt.savefig("ProyeccionCTSim.pdf")
plt.show()

# %% [markdown]
# # Ejercicio 3
# * Usar retroproyeccion filtrada para reconstruir la imagen. Calcular
    la imagen diferencia respecto a la imagen original. Exportarla y
    calcular el error de reconstruccion (normalizar la media de las
    imagenes reconstruidas antes de calcular el error)

# %%
file_reconstuct = 'ReconstuctProjectionCTSimSheppLogan.pgm'
img_reconstruct = read_pgm_file(file_reconstuct)
plt.imshow(img_reconstruct, cmap='gray')
plt.title('Reconstrucción CTSim', fontsize = 12)
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
plt.savefig("ReconstruccionCTSim.pdf")
plt.show()

# %%
# Para calcular el error noramlizamos la media de las imágenes
    reconstruidas
def normaliza_medias(img):
    img_norm = img/np.mean(img)
    return img_norm

# %%
# Para calcular el error noramlizamos la media de las imágenes
    reconstruidas
error = normaliza_medias(img) - normaliza_medias(img_reconstruct)
plt.imshow(error, cmap='gray')
plt.title('Error', fontsize = 12)
```

```
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
plt.savefig("Error.pdf")
plt.show()

# %%
# Calculo del error de reconstruccion
print(f"El error de reconstrucción es: {np.sqrt(np.mean(error**2)):.2f}
    ")

# %% [markdown]
# # Ejercicio 4
# * Estudiar el error de reconstruccion como funcion de los parametros
    de adquisicion (numero de detectores, etc) y de los parametros de
    reconstruccion (filtro, etc).

# %%
sizes = [(32,32), (128,128), (256,256), (512, 512), (720, 720), (1024,
    1024)]
theta = np.linspace(0., 360., 320, endpoint=False)
fig, (ax1,ax2) = plt.subplots(2,1, figsize=(8, 12))
ax1.imshow(img_eq, cmap='gray')
ax1.set_title('Imagen Original', fontsize = 12)
ax1.set_xlabel('x', fontsize = 12)
ax1.set_ylabel('y', fontsize = 12)
sinogram = radon(img_eq, theta=theta)
dx, dy = 0.5 * 360.0 / max(img_eq.shape), 0.5 / sinogram.shape[0]
ax2.imshow(sinogram, cmap=plt.cm.Greys_r,extent=(-dx, 360.0 + dx, -dy,
    sinogram.shape[0] + dy),aspect='auto')
ax2.set_title('Sinograma', fontsize = 12)
ax2.set_xlabel('Angulo (grados)', fontsize = 12)
ax2.set_ylabel('Proyección (detectores)', fontsize = 12)
plt.savefig("SinogramaEQ.pdf")


# %%
errors_theta = np.zeros(len(sizes))
for i,size in enumerate(sizes):
        resized_img = cv2.resize(img_eq, size, interpolation = cv2.
            INTER_CUBIC)
        fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 8))
        ax1.imshow(resized_img, cmap='gray')
        ax1.set_title('Imagen Equalizada de tamano ' + str(size),
            fontsize = 12)
        ax1.set_xlabel('x', fontsize = 12)
        ax1.set_ylabel('y', fontsize = 12)
        sinogram = radon(resized_img, theta=theta)
        dx, dy = 0.5 * 360.0 / max(img_eq.shape), 0.5 / sinogram.shape
            [0]
        ax2.imshow(sinogram, cmap=plt.cm.Greys_r,extent=(-dx, 360.0 +
            dx, -dy, sinogram.shape[0] + dy),aspect='auto')
```

```
182         ax2.set_title('Sinograma de tamano ' + str(size), fontsize =
                12)
183         ax2.set_xlabel('Angulo (grados)', fontsize = 12)
184         ax2.set_ylabel('Proyección (detectores)', fontsize = 12)
185         reconstruction_fbp = iradon(sinogram, theta=theta, filter_name=
                'ramp')
186         reconstruction_fbp_resize =  cv2.resize(reconstruction_fbp,
                size, interpolation = cv2.INTER_AREA)
187         ax3.imshow(reconstruction_fbp_resize, cmap='gray')
188         ax3.set_title('Reconstrucción FBP de tamano ' + str(size),
                fontsize = 12)
189         ax3.set_xlabel('x', fontsize = 12)
190         ax3.set_ylabel('y', fontsize = 12)
191         plt.show()
192         # Para el error normalizamos
193         error = normaliza_medias(reconstruction_fbp_resize) -
                normaliza_medias(resized_img)
194         errors_theta[i] = np.sqrt(np.mean(error**2))
195         print(f'FBP rms reconstruction error for {size} = {np.sqrt(np.
                mean(error**2)):.3g}')
196
197 # %% [markdown]
198 # Grafico del error en función del número de detectores
199
200 # %%
201 plt.figure()
202 detectores = [size[0] for size in sizes]
203 plt.plot(detectores, errors_theta, 'o-', label = 'Error')
204 plt.title('Error de reconstrucción FBP', fontsize = 12)
205 plt.xlabel('Número de detectores', fontsize = 12)
206 plt.ylabel('Error', fontsize = 12)
207 plt.gca().xaxis.set_major_locator(plt.AutoLocator())
208 plt.gca().xaxis.set_minor_locator(AutoMinorLocator())
209 plt.gca().yaxis.set_major_locator(plt.AutoLocator())
210 plt.gca().yaxis.set_minor_locator(AutoMinorLocator())
211 plt.gca().yaxis.set_ticks_position('both')
212 plt.gca().xaxis.set_ticks_position('both')
213 plt.tick_params(axis='both', which='both', direction='in', top=True,
      right=True, labelsize=12)
214 plt.legend(fontsize=10, loc = "best")
215 plt.grid(linestyle='--', linewidth=0.5)
216 plt.tight_layout()
217 plt.savefig("ErrorReconstruccionFBP_detectores.pdf")
218 plt.show()
219
220 # %% [markdown]
221 # Ahora podemos hacer lo mismo en función del número de ángulos
222
223 # %%
224 img_eq.shape
225
226 # %%
```

```
nroangles = np.linspace(1, 1000, 100, dtype=int)
errors_angles = np.zeros(len(nroangles))

for i,nroangle in enumerate(nroangles):
    theta = np.linspace(0., 360., nroangle, endpoint=False)
    sinogram = radon(img_eq, theta=theta)
    reconstruction_fbp = iradon(sinogram, theta=theta, filter_name='
        ramp')
    error = normaliza_medias(reconstruction_fbp) - normaliza_medias(img
        )
    errors_angles[i] = np.sqrt(np.mean(error**2))

# %%
plt.figure()
plt.plot(nroangles, errors_angles, 'o-', label = 'Error')
plt.title('Error de reconstrucción FBP', fontsize = 12)
plt.xlabel('Número de ángulos', fontsize = 12)
plt.ylabel('Error', fontsize = 12)
plt.gca().xaxis.set_major_locator(plt.AutoLocator())
plt.gca().xaxis.set_minor_locator(AutoMinorLocator())
plt.gca().yaxis.set_major_locator(plt.AutoLocator())
plt.gca().yaxis.set_minor_locator(AutoMinorLocator())
plt.gca().yaxis.set_ticks_position('both')
plt.gca().xaxis.set_ticks_position('both')
plt.tick_params(axis='both', which='both', direction='in', top=True,
    right=True, labelsize=12)
plt.legend(fontsize=10, loc = "best")
plt.grid(linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.savefig("ErrorReconstruccionFBP_theta.pdf")
plt.show()

# %% [markdown]
# Ahora realizamos reconstrucciones pero cambiando el filtro, dejamos
    los detectores y número de ángulos constantes

# %%
filters = ['ramp', 'shepp-logan', 'cosine', 'hamming', 'hann']
errors_filters = np.zeros(len(filters))

# %%
theta = np.linspace(0., 360., 360, endpoint=False)
plt.imshow(img_eq, cmap='gray')
plt.show()
sinogram = radon(img_eq, theta=theta)
for i, filter in enumerate(filters):
    reconstruction_fbp = iradon(sinogram, theta=theta, filter_name=
        filter)
    plt.imshow(reconstruction_fbp, cmap='gray')
    plt.show()
    error = normaliza_medias(reconstruction_fbp) - normaliza_medias(
        img_eq)
```

```
273     errors_filters[i] = np.sqrt(np.mean(error**2))
274
275 # %%
276 plt.figure()
277 plt.plot(errors_filters, 'o', label = 'Error', markersize = 8)
278 plt.title('Error de reconstrucción FBP', fontsize = 12)
279 plt.xlabel('Filtro', fontsize = 12)
280 plt.ylabel('Error', fontsize = 12)
281 plt.xticks([0,1,2,3,4], [r'ramp', r'shepp-logan', r'cosine', r'hamming'
    , r'hann'], fontsize = 12)
282 plt.gca().yaxis.set_major_locator(plt.AutoLocator())
283 plt.gca().yaxis.set_minor_locator(AutoMinorLocator())
284 plt.gca().yaxis.set_ticks_position('both')
285 plt.tick_params(axis='y', which='both', direction='in', top=True, right
   =True, labelsize=12)
286 plt.tick_params(axis='x', direction='in', top=False, labelsize=12)
287 plt.legend(fontsize=10, loc = "best")
288 plt.grid(linestyle='--', linewidth=0.5)
289 plt.tight_layout()
290 plt.savefig("ErrorReconstruccionFBP_filter.pdf")
291 plt.show()
292
293 # %% [markdown]
294 # La diferencia es muy poca pero el filtro ramp es el que mejor se
    ajusta a la imagen original ya que no hay presencia de ruidos de
    alta frecuencia en la imagen.
295
296 # %% [markdown]
297 # # Ejercicio 5
298 # * Usando el programa CTSim-noise.c o la función numpy.random intro-
299 # ducir ruido sobre el sinograma del fantoma de Shepp-Logan.
    reconstruir
300 # la imagen y estudiar el efecto del ruido utilizando diversos filtros
301
302 # %%
303 def introduce_ruido(img):
304     # Introducimos ruido gaussiano con media = 0 y desviación estándar
        sigma = 6
305     mean = 0
306     var = 36
307     sigma = var**0.5
308     gauss = np.random.normal(mean,sigma,img.shape)
309     gauss = gauss.reshape(img.shape)
310     noisy_img = img + gauss
311     return noisy_img
312
313 # %%
314 theta = np.linspace(0., 360., 360, endpoint=False)
315 sinogram = radon(img_eq, theta=theta)
316 noisy_sinogram = introduce_ruido(sinogram)
317 plt.imshow(noisy_sinogram, cmap='gray')
318 plt.title('Sinograma con ruido', fontsize = 12)
```

```
plt.xlabel('Angulo (grados)', fontsize = 12)
plt.ylabel('Proyección (detectores)', fontsize = 12)
plt.savefig("Sinograma_ruido.pdf")
plt.show()
for i, filter in enumerate(filters):
    reconstruction_fbp = iradon(noisy_sinogram, theta=theta,
       filter_name=filter)
    plt.imshow(reconstruction_fbp, cmap='gray')
    plt.title('Reconstrucción FBP con filtro ' + filter, fontsize = 12)
    plt.xlabel('x', fontsize = 12)
    plt.ylabel('y', fontsize = 12)
    plt.savefig("ReconstruccionFBP_ruido_" + filter + ".pdf")
    plt.show()
    error = normaliza_medias(reconstruction_fbp) - normaliza_medias(
       img_eq)
    errors_filters[i] = np.sqrt(np.mean(error**2))

plt.figure()
plt.plot(errors_filters, 'o', label = 'Error', markersize = 8)
plt.title('Error de reconstrucción FBP con ruido', fontsize = 12)
plt.xlabel('Filtro', fontsize = 12)
plt.ylabel('Error', fontsize = 12)
plt.xticks([0,1,2,3,4], [r'ramp', r'shepp-logan', r'cosine', r'hamming'
    , r'hann'], fontsize = 12)
plt.gca().yaxis.set_major_locator(plt.AutoLocator())
plt.gca().yaxis.set_minor_locator(AutoMinorLocator())
plt.gca().yaxis.set_ticks_position('both')
plt.tick_params(axis='y', which='both', direction='in', top=True, right
   =True, labelsize=12)
plt.tick_params(axis='x', direction='in', top=False, labelsize=12)
plt.legend(fontsize=10, loc = "best")
plt.grid(linestyle='--', linewidth=0.5)
plt.tight_layout()
plt.savefig("ErrorReconstruccionFBP_ruido_filter.pdf")
plt.show()

# %%
from skimage.transform.radon_transform import _get_fourier_filter

filters = ['ramp', 'shepp-logan', 'cosine', 'hamming', 'hann']

for ix, f in enumerate(filters):
    response = _get_fourier_filter(2000, f)
    plt.plot(response, label=f)

plt.xlim([0, 1000])
plt.xlabel('Frecuencia', fontsize = 12)
plt.ylabel('Filtros', fontsize = 12)
plt.gca().xaxis.set_major_locator(plt.AutoLocator())
plt.gca().xaxis.set_minor_locator(AutoMinorLocator())
plt.gca().yaxis.set_major_locator(plt.AutoLocator())
plt.gca().yaxis.set_minor_locator(AutoMinorLocator())
```

```
plt.gca().yaxis.set_ticks_position('both')
plt.gca().xaxis.set_ticks_position('both')
plt.tick_params(axis='both', which='both', direction='in', top=True,
    right=True, labelsize=12)
plt.legend(fontsize=10, loc = "best")
plt.grid(linestyle='--', linewidth=0.5)
plt.savefig("Filtros.pdf")
plt.show()

# %%
def crear_circulo(img_size, D = 5):
    w,h = img_size
    circle_image = np.zeros((w,h), dtype = np.uint8)
    r = D/2
    for i in range(w):
        for j in range(h):
            if (i - int(w//2))**2 + (j - int(h//2))**2 < r**2:
                circle_image[i,j] = 255
    return circle_image

# %%
img_size = (100,100)

# %%
circle_image = crear_circulo(img_size, D = 5)

# %%
plt.imshow(circle_image, cmap='gray')
plt.xlabel('x', fontsize = 12)
plt.ylabel('y', fontsize = 12)
plt.savefig("Circulo.pdf")
plt.show()

# %%
num_angles = [8, 16, 32, 64]
for n in num_angles:
    theta = np.linspace(0, 360, n, endpoint=False)
    sinogram = radon(circle_image, theta=theta)
    # dx, dy = 0.5 * 360.0 / max(circle_image.shape), 0.5 / sinogram.
       shape[0]
    # plt.imshow(sinogram, cmap=plt.cm.Greys_r,extent=(-dx, 360.0 + dx,
        -dy, sinogram.shape[0] + dy),aspect='auto')
    reconstruction_fbp = iradon(sinogram, theta=theta, filter_name='
       ramp')
    plt.imshow(reconstruction_fbp, cmap='gray')
    plt.title(f'Reconstruccion con {n} angulos (Con rampa)', fontsize =
        12)
    plt.xlabel('Angulo (grados)', fontsize = 12)
    plt.ylabel('Proyección (detectores)', fontsize = 12)
    plt.savefig(f'Reconstrucción con {n} angulos (Con rampa).pdf')
    plt.show()
```

```
# %% [markdown]
# # Ejercicio 7
# * Supongamos que un elemento del detector de Rayos-X tiene una falla
# y no registra ninguna actividad. Como se traduce este efecto en el
# sinograma Reconstruir una imagen a partir de un sinograma con este
# defecto y evaluar los artefactos que produce.
#

# %%
img_eq.shape

# %%
fig, (ax0,ax1) = plt.subplots(num='subplots', figsize=(10,4), nrows=1,
    ncols=2)
theta = np.linspace(0, 360, 360, endpoint=False)
sinogram_imgeq = radon(img_eq, theta=theta)
sinogram_imgeq[-200,:] = 0
dx, dy = 0.5 * 360.0 / max(img_eq.shape), 0.5 / sinogram_imgeq.shape[0]
ax0.imshow(sinogram_imgeq, cmap=plt.cm.Greys_r,extent=(-dx, 360.0 + dx,
     -dy, sinogram_imgeq.shape[0] + dy),aspect='auto')
ax0.set_title("Sinograma detector 200 roto", fontsize = 12)
ax0.set_xlabel('Angulo (grados)', fontsize = 12)
ax0.set_ylabel('Proyeccion (detectores)', fontsize = 12)
reconstruction_fbp = iradon(sinogram_imgeq, theta=theta, filter_name='
    ramp')
ax1.imshow(reconstruction_fbp, cmap='gray')
ax1.set_title("Reconstruccion", fontsize = 12)
ax1.set_xlabel('x', fontsize = 12)
ax1.set_ylabel('y', fontsize = 12)
plt.savefig("Detector200Roto.pdf")
```