

1. Objetivos

- Calcular la transformada discreta directa e inversa de Fourier de una imagen manipulando sus componentes.
- Realizar operaciones de suavizado y de reducción de ruido en imágenes utilizando filtros en frecuencia.
- Realizar operaciones de detección de bordes en imágenes, tanto limpias como ruidosas, utilizando filtros en frecuencia.

2. Introducción

La contribución Joseph Fourier (1822) establece que cualquier función que se repite de manera periódica puede ser expresada como la suma de senos y/o cosenos de diferentes frecuencias cada una multiplicada por un coeficiente diferente. A lo anterior se le conoce como "series de Fourier".

- Transformada discreta de Fourier en 2D.

Las funciones pares que no son periódicas (pero cuya área bajo la curva es finita) pueden ser expresadas como la integral de senos y/o cosenos multiplicados por una función ponderada (pesos). A esta formulación se la conoce como "transformada de Fourier"

La transformada de Fourier discreta de una función (imagen) $f(x, y)$ de tamaño $M \times N$ está dada por la ecuación:

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M+vy/N)}$$

para $u = 0, 1, 2, \dots, M - 1$ y $v = 0, 1, 2, \dots, N - 1$.

De manera similar, dada $F(u, v)$, obtenemos $f(x, y)$ via la transformada inversa de Fourier:

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M+vy/N)}$$

para $x = 0, 1, 2, \dots, M - 1$ y $y = 0, 1, 2, \dots, N - 1$.

Las ecuaciones anteriores comprenden al par de transformadas discretas de Fourier bi-dimensionales (DFT). Las variables u y v son las variables de la transformada o variables de frecuencia, mientras que x y y son las variables espaciales o variables de la imagen.

Se definen al espectro de Fourier, al ángulo de fase y al espectro de potencia de la siguiente manera, respectivamente:

$$|F(u, v)| = [R^2(u, v) + I^2(u, v)]^{1/2}$$

$$\phi(u, v) = \tan^{-1} \left[\frac{I(u, v)}{R(u, v)} \right]$$

$$P(u, v) = |F(u, v)|^2 = R^2(u, v) + I^2(u, v)$$

donde $R(u, v)$ e $I(u, v)$ son las partes real e imaginaria de $F(u, v)$ respectivamente.

Es de práctica común multiplicar la función de entrada ($f(x, y)$) por $(-1)^{x+y}$ antes de calcular la transformada de Fourier. De las propiedades de los exponentes tenemos:

$$\Im[f(x, y)(-1)^{x+y}] = F(u - M/2, v - N/2)$$

donde $\Im[\bullet]$ denota la transformada de Fourier del argumento. Esta ecuación establece que el origen de la transformada de Fourier de $f(x, y)(-1)^{x+y}$ [que es $F(0, 0)$] está localizada en $u = M - 1$ y $v = N - 1$. En otras palabras multiplicando $f(x, y)$ por $(-1)^{x+y}$ hace que $F(u, v)$ se recorra a las coordenadas de frecuencia $(M/2, N/2)$ que es el centro del área 2D ocupada por la DFT.

- Filtrado en frecuencia

Generalmente es imposible hacer relaciones directas entre los componentes de los dominios del espacio de la imagen y de la frecuencia. Sin embargo, se pueden encontrar algunas relaciones entre los componentes de la frecuencia y algunas características de la imagen.

Por ejemplo, se pueden asociar las frecuencias de la transformada de Fourier con patrones de variación de las intensidades de la imagen. La frecuencia más baja ($u = v = 0$) corresponde al promedio de los valores de gris de la imagen. Mientras nos alejamos del origen, las frecuencias corresponden a variaciones suaves en los tonos de gris. Conforme nos alejamos más las frecuencias altas empiezan a corresponder a cambios rápidos o abruptos en los tonos de gris como son por ejemplo los bordes de los objetos y/o el ruido.

Filtrar en el dominio de la frecuencia consiste en los siguientes pasos:

1. Multiplicar la imagen de entrada por $(-1)^{x+y}$ para centrar la transformación.
2. Calcular, $F(u, v)$, la TDF de la imagen resultado de 1 .
3. Multiplicar $F(u, v)$ por la función filtro $H(u, v)$.
4. Calcular la TDF inversa del resultado de 3 .
5. Obtener la parte real del resultado de 4 .
6. Multiplicar el resultado de 5 por $(-1)^{x+y}$.

La razón por la cual $H(u, v)$ se llama filtro (también se llama filtro de función de transferencia) es porque suprime ciertas frecuencias en la transformada y deja otras sin cambio.

Sea $f(x, y)$ la imagen de entrada y $F(u, v)$ su transformada discreta de Fourier. La transformada de Fourier de la imagen de salida después de aplicar el filtro está dada por:

$$G(u, v) = H(u, v)F(u, v)$$

donde la multiplicación de H por F involucra funciones bidimensionales y está definida elemento a elemento.

3. Desarrollo

Resuelve los problemas de la lista siguiente y describe tu solución en cada inciso. Los incisos en donde únicamente tengas que desplegar imágenes no requieren de ninguna descripción.

1. Calcular la transformada discreta de Fourier a la imagen del Sr. Fourier y generar el ejemplo visto del libro Castleman, 1996, figura 10.10 (ver la figura al final de esta práctica).
- a) La versión sin corrimiento de la amplitud y la fase de la imagen de entrada I.
- b) La versión con corrimiento de la amplitud y la fase de la imagen de entrada I.
- c) Regresar con la transformada inversa completa a la imagen original.
- d) Regresar con la transformada inversa sólo con amplitud.
- e) Regresar con la transformada inversa sólo con la fase.

Nota: Si trabajas en MATLAB puedes utilizar las funciones: fft2, ifft2, fftshift, real, angle, complex.

2. Aplicar a una imagen sin ruido y la misma imagen con ruido "sal y pimienta" filtros de suavizamiento y realce. La imagen con ruido se puede generar a partir de la imagen sin ruido usando el siguiente comando de MATLAB: J=imnoise(I, TIPO,...), donde TIPO es una cadena que puede tomar valores 'gaussian', 'salt & pepper', etc. Ver comando help imnoise.
- a) Aplicar el filtro paso bajas Butterworth, para eliminación de ruido. Prueba para varios valores de frecuencia de corte D_0 y para varios valores de orden n y establece cuales valores son mejores para la imagen en cuestión. Realiza el ejercicio para la imagen con y sin ruido.
- b) Aplicar el filtro paso altas Butterworth para el realce de bordes. Prueba para varios valores de frecuencia de corte D_0 y para varios valores de orden n y establece cuales valores son mejores para la imagen en cuestión.

Nota: Recuerda que debes realizar el proceso de padding según sea necesario.

4. Código

Para esta primera parte de la práctica, no se crearon funciones extras, se reutilizo el código de prácticas pasadas para palotear las imágenes. Se hizo uso de las funciones proporcionadas por numpy de la librería fft: fft2 para hacer la transformación, fftshift para centrar la imagen, abs para obtener la magnitud, angle para la fase, log2 para la amplitud y ifft2 para la transformada inversa.

Practica3.ipynb - Practica3 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

Practica3.ipynb x Practica3.ipynb

Practica3.ipynb > M> Practica3 > M> NC: 420004035 > saveImg(amplitud_tf,amplitud_sin_rec)

+ Code + Markdown | ▶ Run All ⏪ Restart ⏴ Clear All Outputs | Variables ⏴ Outline ... Python 3.10.12

```
rm=io.imread('imagenes/fourier_bw512.bmp')/255
imagenes['fourier_bw'] = rm
rm=io.imread('imagenes/ImagenEspectroF.png')/255
imagenes['Espectro_Fourier']= rm
rm=io.imread('imagenes/saturn_bw.tif')/255
imagenes['Saturn_bw']= rm
[4] ✓ 0.9s Python

for img in imagenes:
    tf_imagenes[img] = np.fft.fft2(imagenes[img])
[5] ✓ 0.1s Python

for img in imagenes:
    ctf_imagenes[img] = np.fft.fftshift(tf_imagenes[img])
[6] ✓ 0.0s Python

for img in imagenes:
    magnitud_cif[img] = abs(ctf_imagenes[img])
for img in imagenes:
    magnitud_tf[img] = abs(tf_imagenes[img])
[7] ✓ 0.2s Python
```

Figure 1: Código para el primer ejercicio

Practica3.ipynb - Practica3 - Visual Studio Code

File Edit Selection View Go Run Terminal Help

Practica3.ipynb x Practica3B.ipynb

Practica3.ipynb > Practica3 > M&NC: 420004035 > savelmg(amplitud_tf,amplitud_sin_rec)

+ Code + Markdown | ▶ Run All ⌘ Restart ⌘ Clear All Outputs | Variables Outline ...

Python 3.10.12

```
for img in imagenes:  
    fase_ctf[img] = np.angle(ctf_imagenes[img])  
for img in imagenes:  
    fase_tf[img] = np.angle(tf_imagenes[img])
```

[8] ✓ 0.1s Python

```
for img in imagenes:  
    amplitud_ctf[img] = np.log1p(magnitud_ctf[img])  
for img in imagenes:  
    amplitud_tf[img] = np.log1p(magnitud_tf[img])
```

[9] ✓ 0.1s Python

```
for img in imagenes:  
    inverse_tf_img[img] = np.fft.ifft2(tf_imagenes[img])  
    inverse_tf_img[img] = abs(inverse_tf_img[img])  
    inverse_tf_img[img] = np.fft.ifftshift(inverse_tf_img[img])  
  
for img in imagenes:  
    inverse_ctf_img[img] = np.fft.ifft2(ctf_imagenes[img])  
    inverse_ctf_img[img] = abs(inverse_ctf_img[img])
```

[10] ✓ 0.3s Python

Figure 2: Código para el primer ejercicio

```

Practica3.ipynb - Practica3 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Practica3.ipynb x Practica3B.ipynb
Practica3.ipynb m Practica3 > NC: 420004035 > for img in imagenes:
+ Code + Markdown | ▶ Run All | ⚡ Restart | Clear All Outputs | Variables | Outline ...
Python 3.10.12

for img in imagenes:
    inverse_amplitud_ctf[img] = np.fft.ifft2(magnitud_ctf[img])
    inverse_amplitud_ctf[img] = abs(inverse_amplitud_ctf[img])
for img in imagenes:
    inverse_amplitud_tf[img] = np.fft.ifft2(amplitud_ctf[img])
    inverse_amplitud_tf[img] = abs(inverse_amplitud_tf[img])

[11] ✓ 0.2s Python

for img in imagenes:
    inverse_fase_ctf[img] = np.fft.ifft2(np.exp(1j*fase_ctf[img]))
    inverse_fase_ctf[img] = abs(inverse_fase_ctf[img]-90)
for img in imagenes:
    inverse_fase_tf[img] = np.fft.ifft2(np.exp(1j*fase_tf[img]))
    inverse_fase_tf[img] = abs(inverse_fase_tf[img]-90)

[12] ✓ 0.4s Python

for img in imagenes:
    !plot_img((img,imagenes[img]),'Imagen original')

[14] ✓ 0.7s Python
... Imagen original fourier_bw
0 100
... 0 100
Ln 2, Col 59 Spaces: 4 LF Cell 14 of 25

```

Figure 3: Código para el primer ejercicio

Para la segunda parte si se realizo la función para el filtro Butterworth y para el ruido sal y pimienta.

```

Practica3B.ipynb - Practica3 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Practica3.ipynb x Practica3B.ipynb
Practica3B.ipynb m Practica3 > NC: 420002035 > !savemg(filtros_BPA/Filtro_PasAlt)
+ Code + Markdown | ▶ Run All | ⚡ Restart | Clear All Outputs | Variables | Outline ...
Python 3.10.12

def salt_pepper_noise(img):
    img2 = img.copy()
    y_space, x_space = img.shape
    pepper = 0.015
    salt = 1 - pepper
    for y in range(y_space):
        for x in range(x_space):
            rnd = np.random.random()
            if rnd < pepper:
                img2[y][x] = 0
            elif rnd > salt:
                img2[y][x] = 1
    return img2

[24] ✓ 0.1s Python

def filter(f,d,n=5):
    M,N = f.shape
    H = np.zeros((M,N), dtype=np.float32)
    for u in range(M):
        for v in range(N):
            d = np.sqrt((u-M/2)**2 + (v-N/2)**2)
            H[u][v] = 1/(1.0 + (d/d_0)**(2*n))

    return H

[25] ✓ 0.1s Python
... 0 100
... 0 100
Cell 22 of 31

```

Figure 4: Código para el segundo ejercicio

The screenshot shows a Visual Studio Code interface with two tabs open: 'Practica3.ipynb' and 'Practica3B.ipynb'. The 'Practica3B.ipynb' tab is active and displays Python code for signal processing. The code includes sections for low-pass filtering ('filtrado_paso_bajas') and high-pass filtering ('filtrado_paso_altas'). It also includes a section for displaying the spectrum ('magnitud_ctf_pb_ruido'). The code is written in Python, utilizing variables like 'ctf_imagenes', 'filtros_BPB', and 'filtros_BPA'. The interface includes a sidebar with various icons, a status bar at the bottom, and a header bar with file navigation and help options.

```
## Filtrado Paso Bajas
filtrado_paso_bajas = {}
filtrado_paso_bajas_ruido = {}

for img in imagenes:
    filtrado_paso_bajas[img] = ctf_imagenes[img] * filtros_BPB[img]
    filtrado_paso_bajas_ruido[img] = ctf_imagenes_ruido[img] * filtros_BPB[img]

## Filtrado Paso altas
filtrado_paso_altas = {}
filtrado_paso_altas_ruido = {}

for img in imagenes:
    filtrado_paso_altas[img] = ctf_imagenes[img] * filtros_BPA[img]
    filtrado_paso_altas_ruido[img] = ctf_imagenes_ruido[img] * filtros_BPA[img]

## Mostramos como se ve el espectro
magnitud_ctf_pb_ruido = {}

for img in imagenes:
```

Figure 5: Código para el segundo ejercicio

5. Conclusiones

A continuación, se muestran algunas imágenes que son los resultados obtenidos de esta practica:

- Amplitud y Fase sin recorrimiento



Figure 6: Amplitud de la imagen de Saturno sin recorrimiento

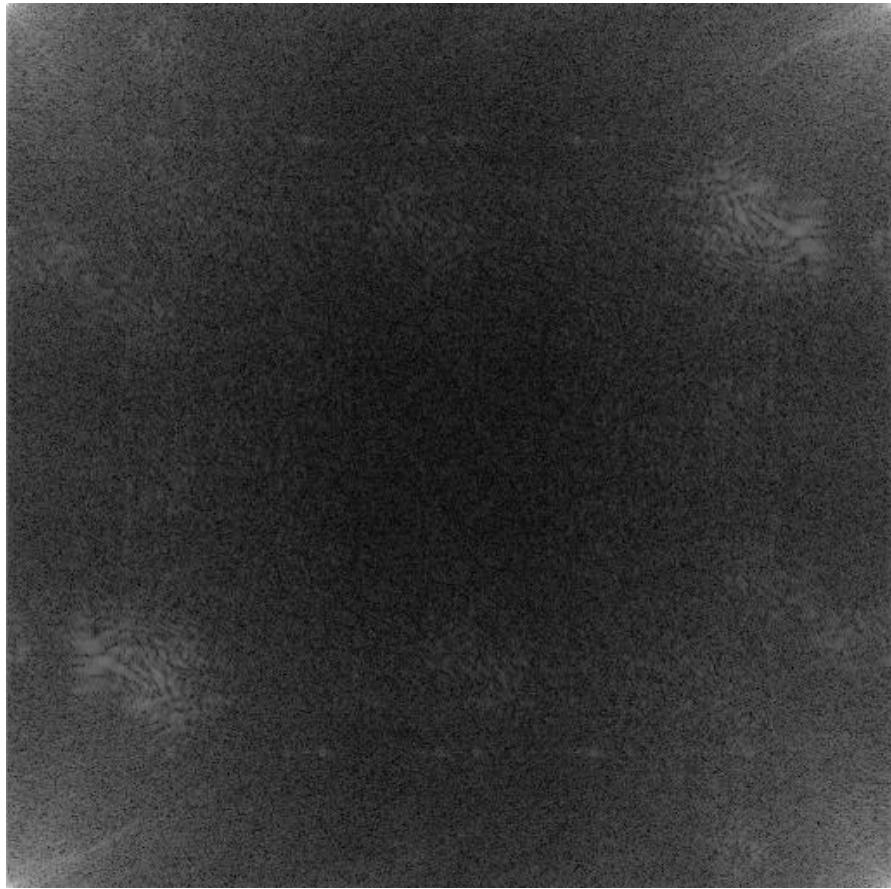


Figure 7: Amplitud de la imagen de Fourier sin recorrimiento

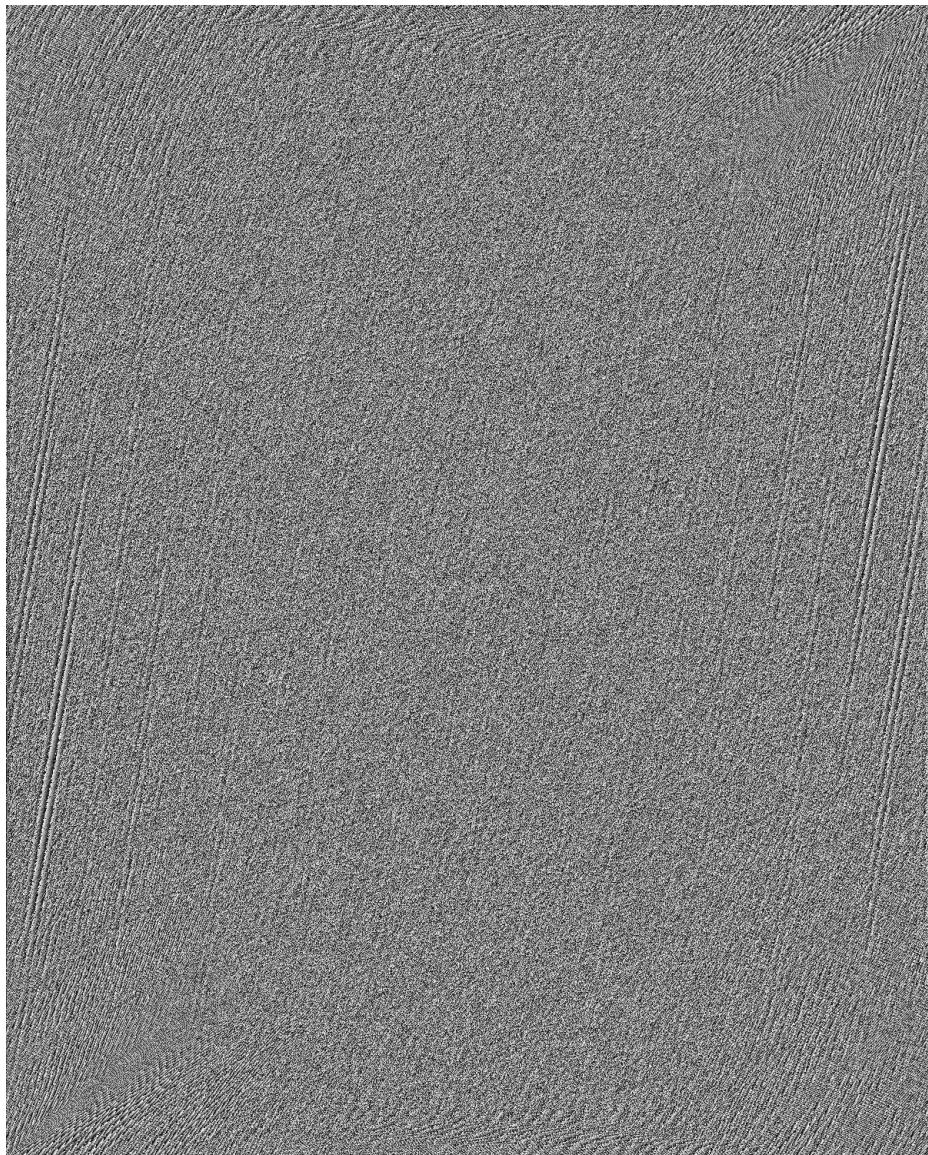


Figure 8: Fase de la imagen de Saturno sin recorrimiento

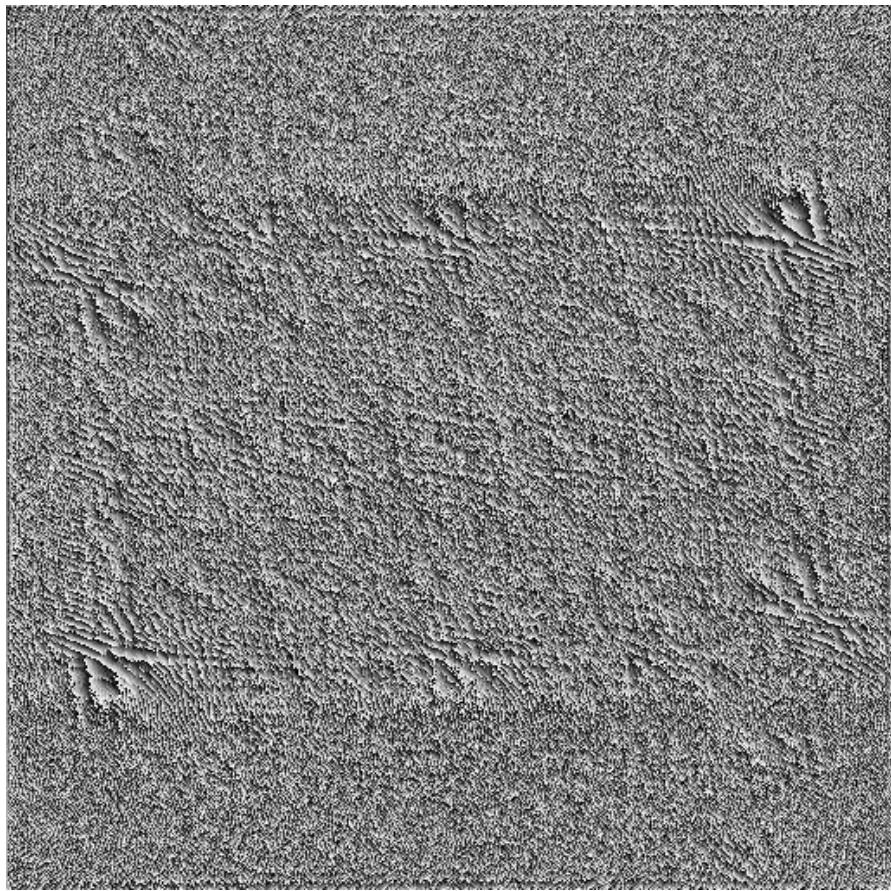


Figure 9: Fase de la imagen de Fourier sin recorrimiento

Podemos observar que sin el recorrimiento la frecuencia cero se encuentra en los extremos.

- Amplitud y Fase con recorrimiento

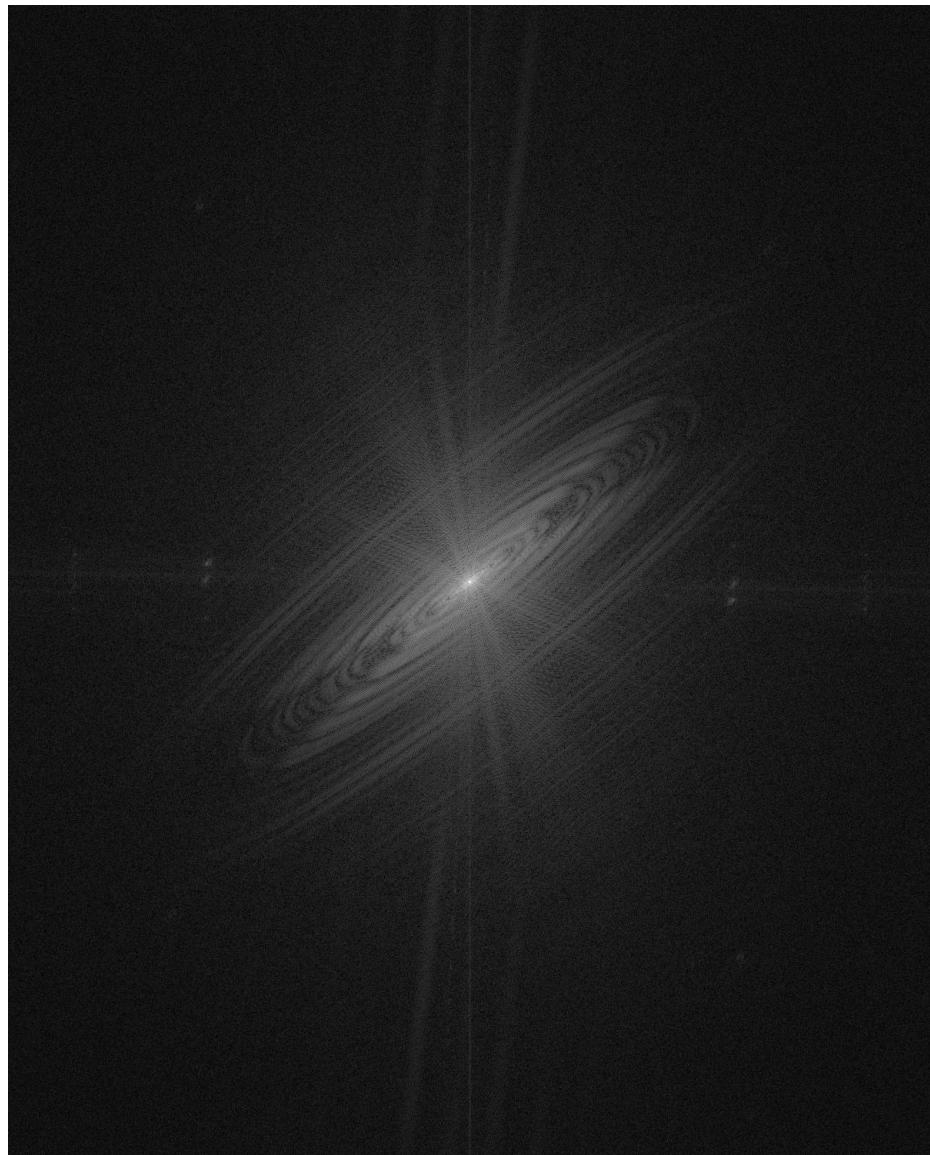


Figure 10: Amplitud de la imagen de Saturno con recorrimiento

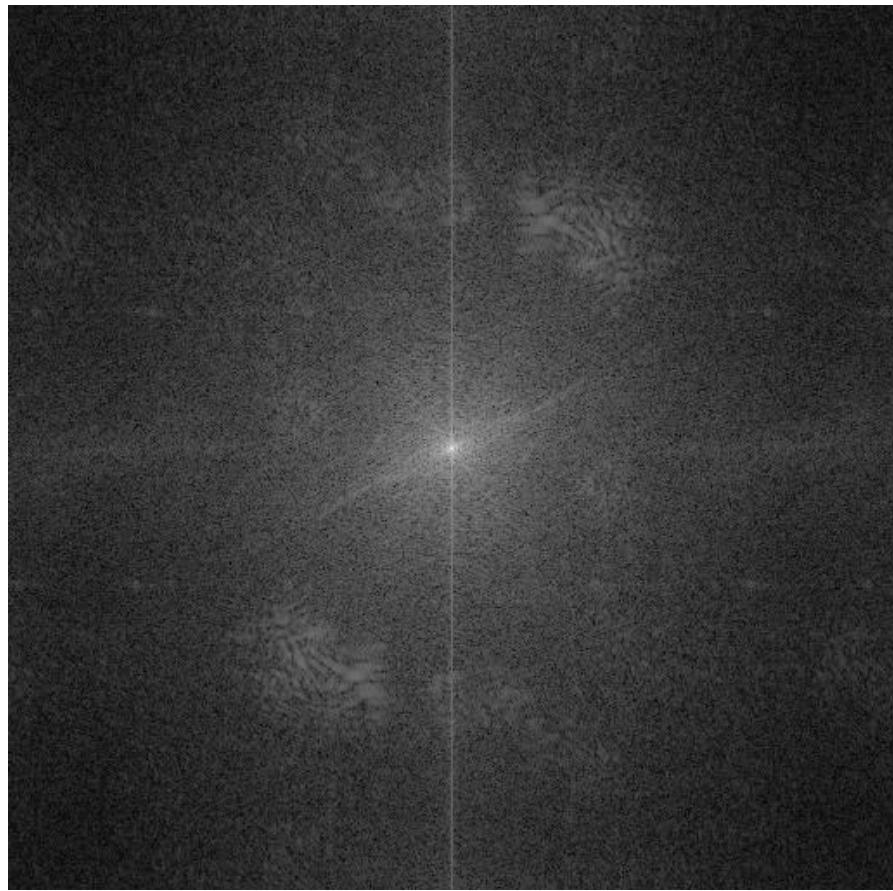


Figure 11: Amplitud de la imagen de Fourier con recorrimiento

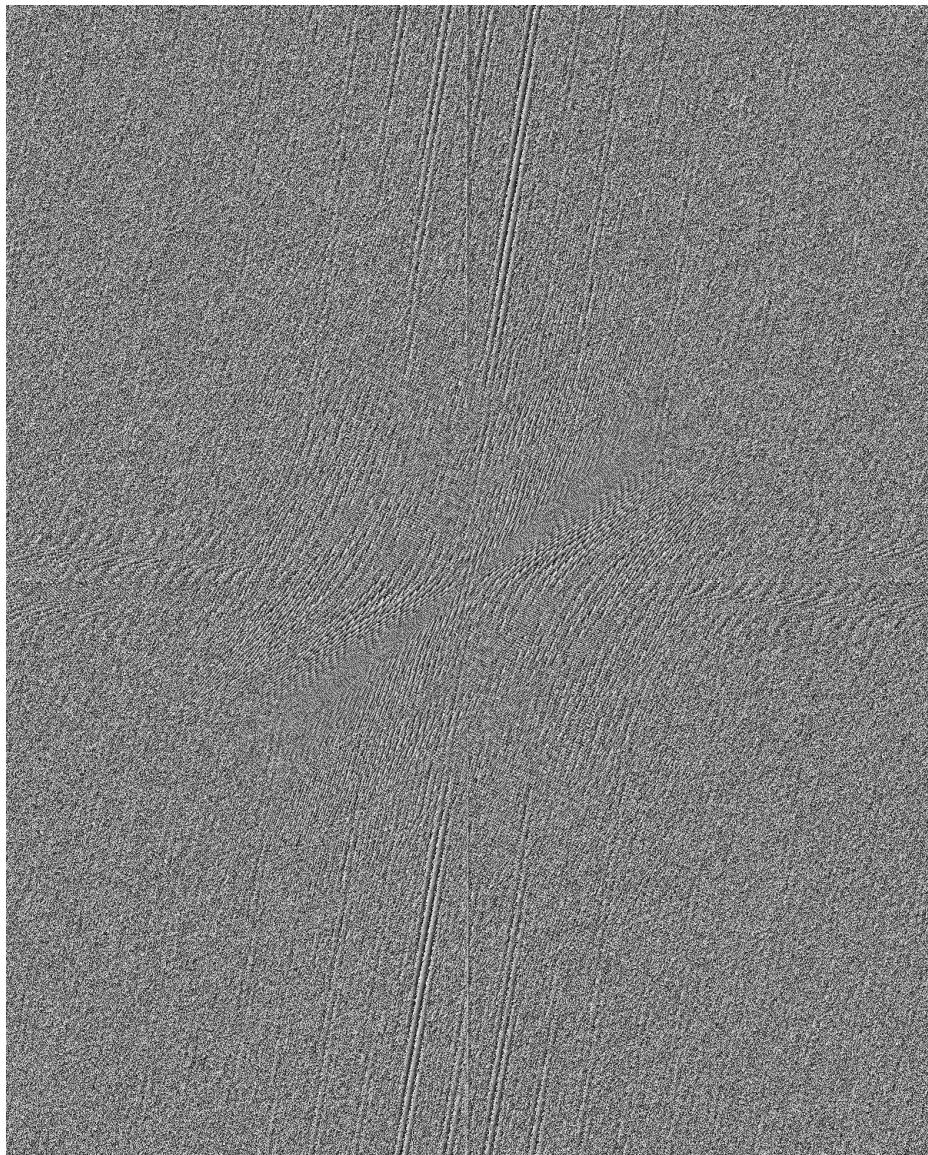


Figure 12: Fase de la imagen de Saturno con recorrimiento

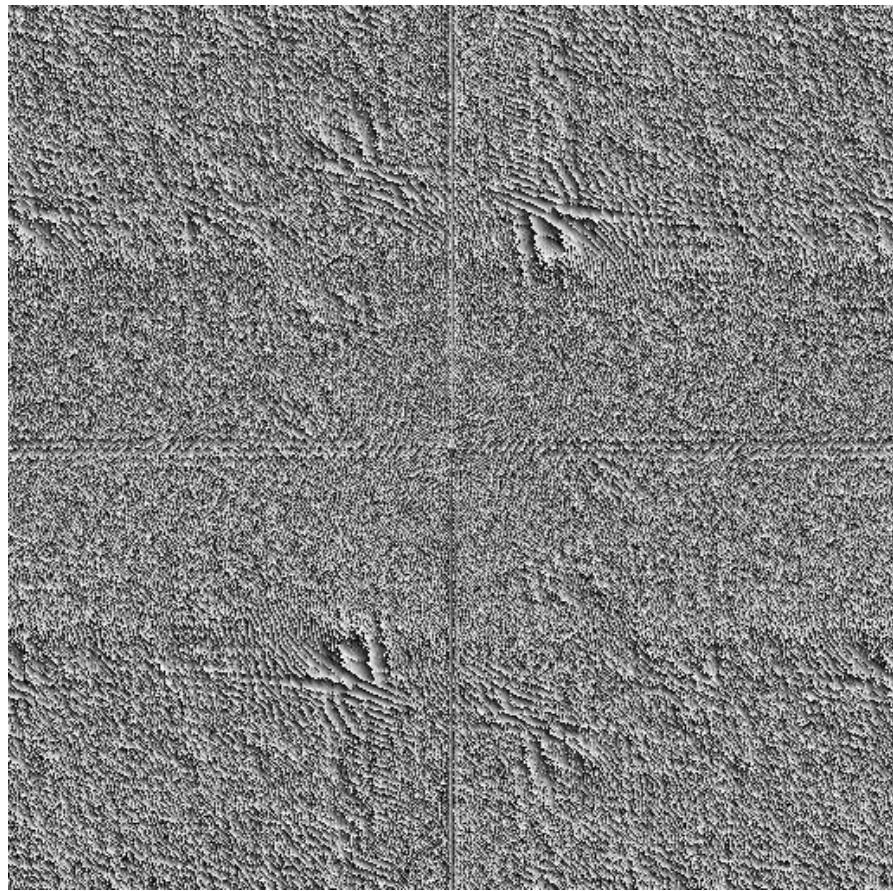


Figure 13: Fase de la imagen de Fourier con recorrimiento

- Regreso de la imagen con la transformada inversa completa



Figure 14: Regreso de la transformada completa pero sin recorrimiento



Figure 15: Regreso de la transformada completa con recorrimiento

Podemos observar que al regresar con la transformada inversa sin previamente volver a realizar el recorrimiento, el centro de la imagen se encuentra distribuido en los extremos.

- Transformada inversa solo con amplitud.

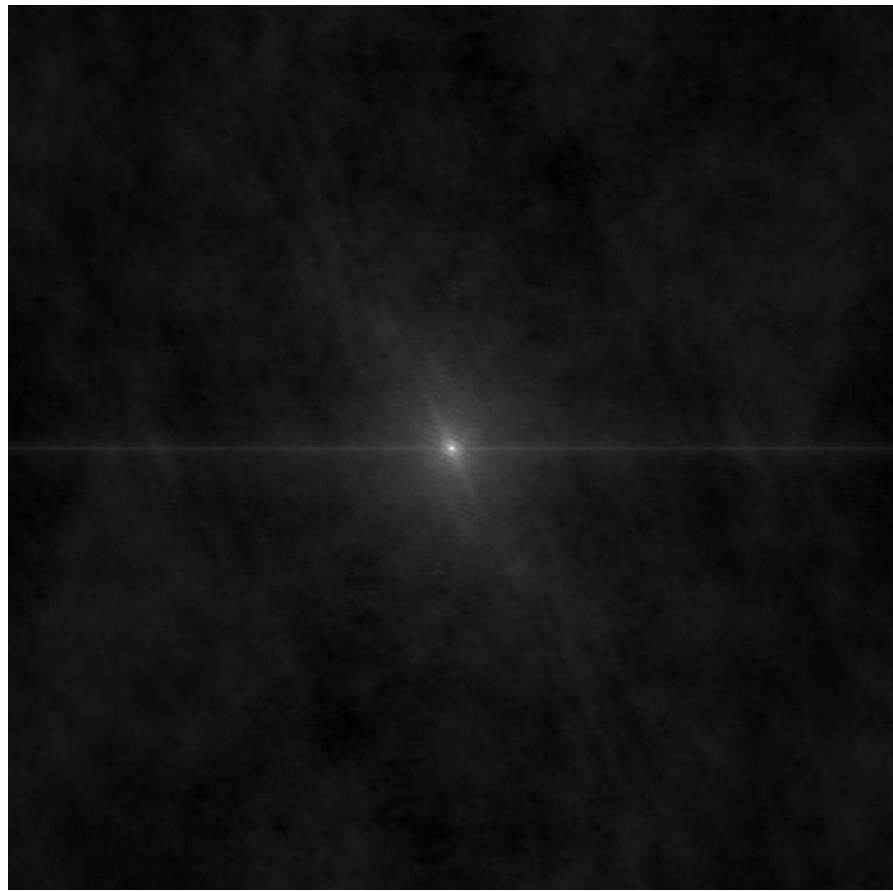


Figure 16: Regreso de la transformada solo amplitud de la imagen de Fourier

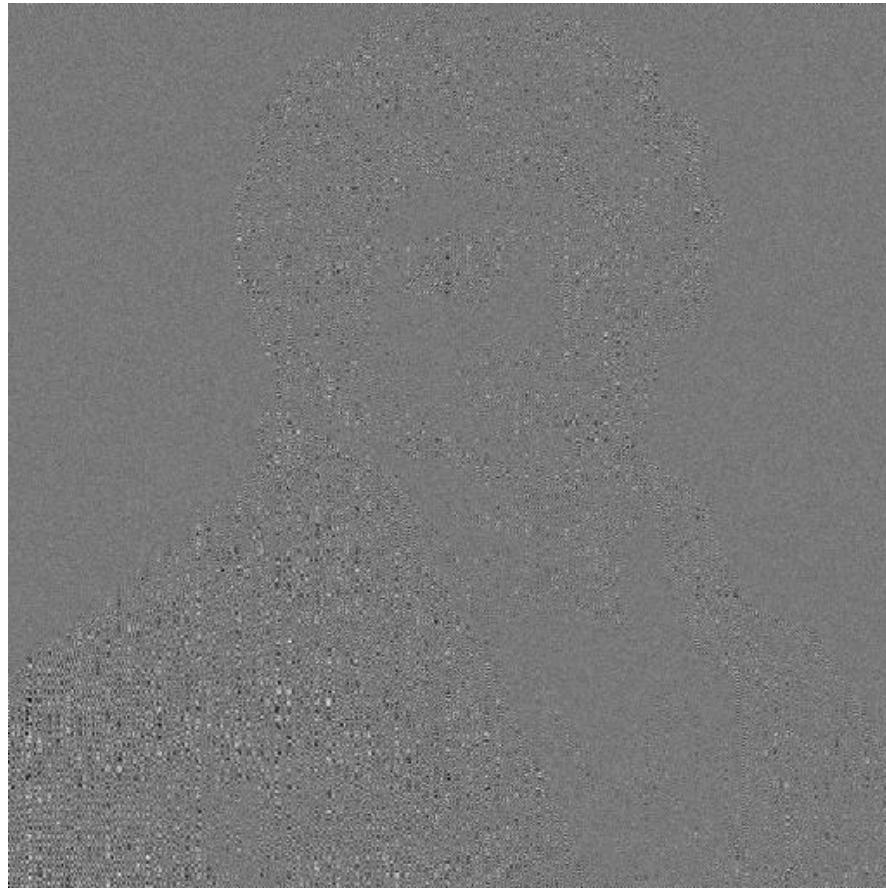


Figure 17: Regreso de la transformada solo fase de la imagen de Fourier

Vemos que entre la amplitud y al fase, la que nos llega a proporcionar mas información sobre la imagen original, es la fase, dándonos la "forma".

Segunda parte de la practica

- Filtros Butterworth de cada imagen.

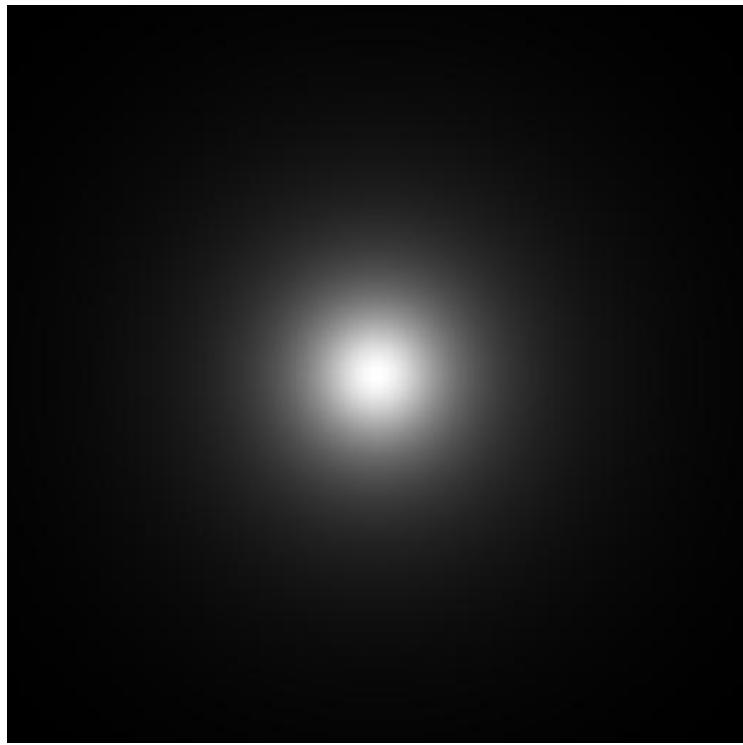


Figure 18: Filtro butterworth paso bajas de la imagen de fourier



Figure 19: Filtro butterworth paso altas de la imagen de saturno

- Filtro aplicado a la transformada de la imagen

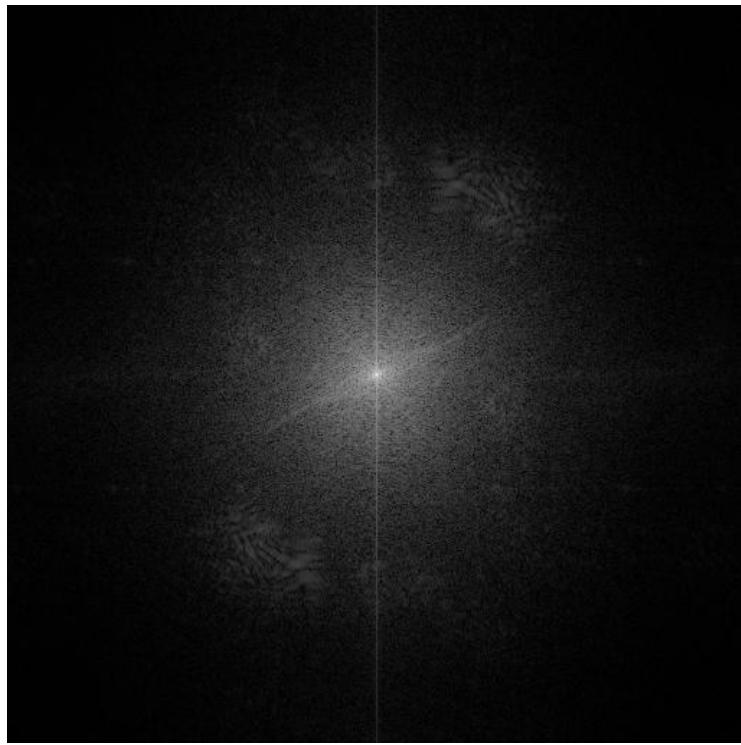


Figure 20: Filtro butterworth paso bajas aplicado a la imagen de fourier

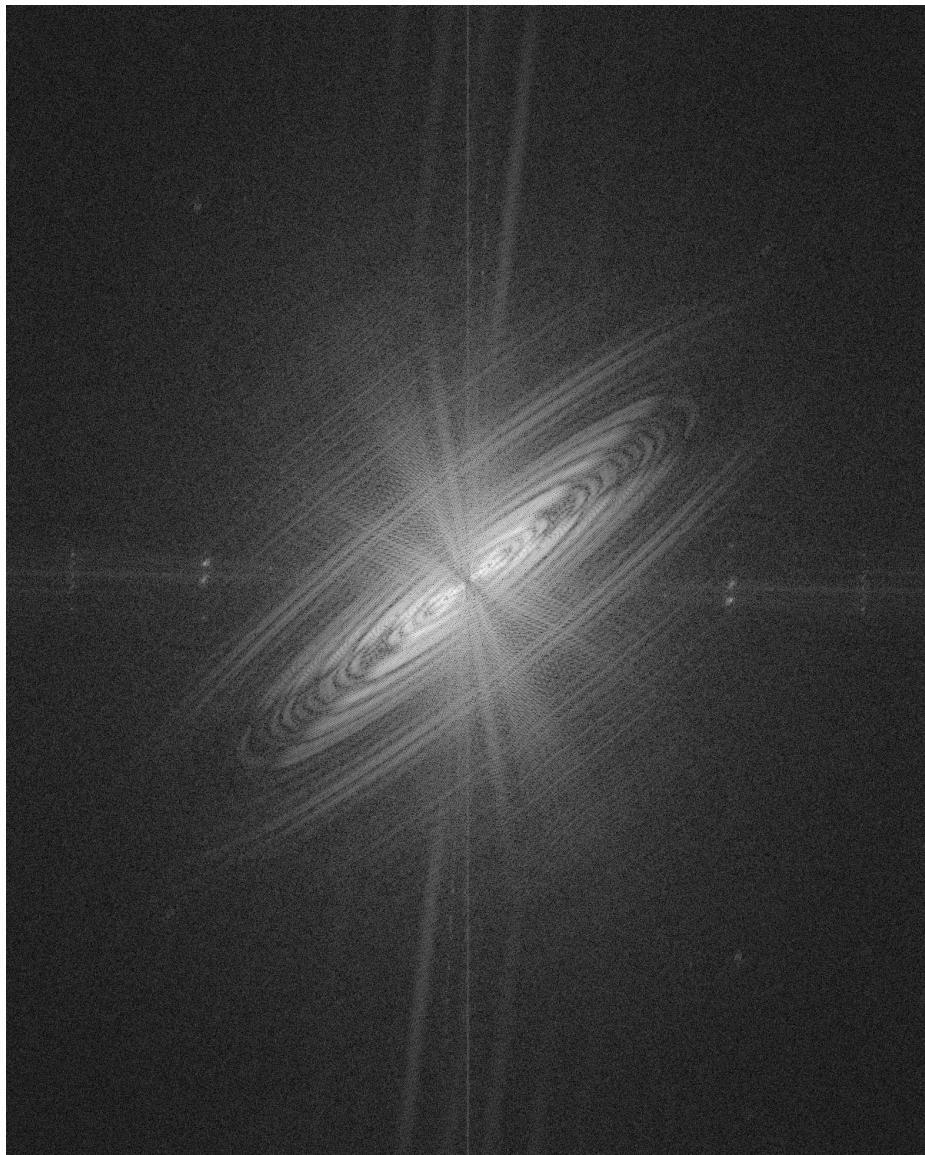


Figure 21: Filtro butterworth paso altas aplicado a la imagen de saturno

- Imagenes con el filtro aplicado



Figure 22: Imagen de Fourier aplicado el filtro

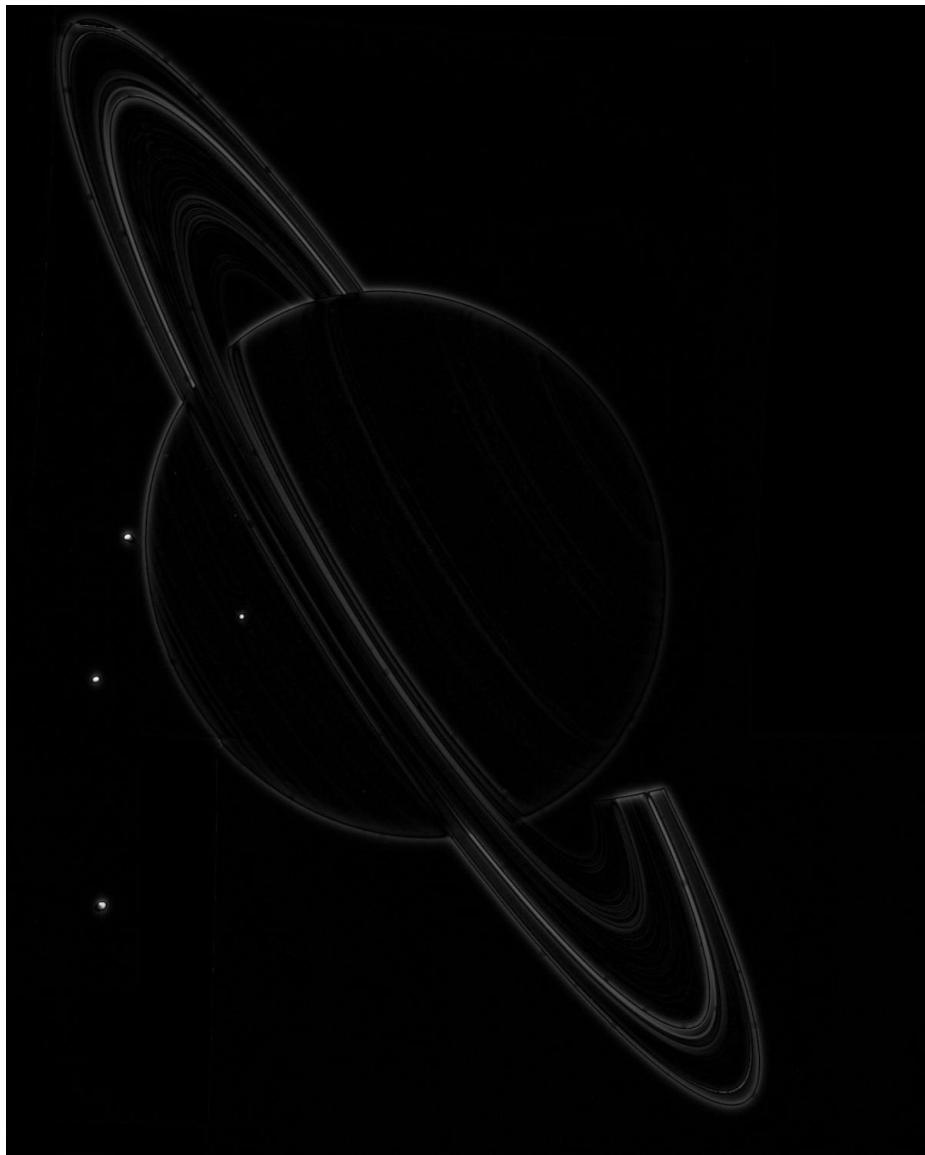


Figure 23: Imagen de Saturno aplicado el filtro

- Resultado de las mismas imágenes pero con ruido



Figure 24: Imagen de Fourier con ruido aplicado el filtro



Figure 25: Imagen de Saturno con ruido aplicado el filtro

De las ultimas imágenes podemos observar que el filtro butterworth en paso bajas, es muy bueno ya que no se pierde información en el suavizado y no hay mucha diferencia entre las imágenes con y sin ruido. En cambio, en el caso de paso altas, es bueno para resaltar los bordes, pero, debido a que el ruido sal y pimienta son pixeles con intensidades altas y bajas distribuidas en la imagen, las de intensidades alta, al ser paso altas, las mantiene, por lo que este ruido se preserva.

6. Referencias

1. <https://matplotlib.org/stable/tutorials/pyplot.htm>
2. <https://numpy.org/>
3. <https://scikit-image.org/>
4. <https://pillow.readthedocs.io/en/stable/reference/Image.html>
5. <https://numpy.org/doc/stable/reference/routines.fft.html>