INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE SÃO PAULO, CÂMPUS BIRIGUI - SP
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO

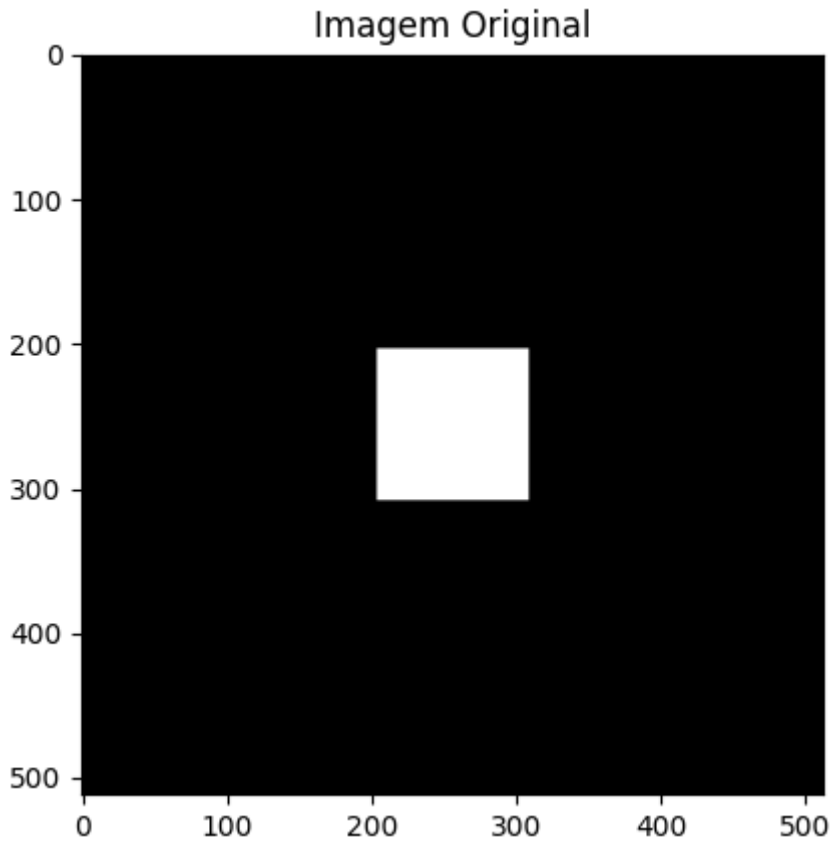
ISADORA DISPOSTI BUENO DOS SANTOS

EXERCÍCIOS – FILTRAGEM FREQUÊNCIA

BIRIGUI - SP
2023

1. Calcule e visualize o espectro de uma imagem 512x512 pixels:

a) crie e visualize uma imagem simples – quadrado branco sobre fundo preto;

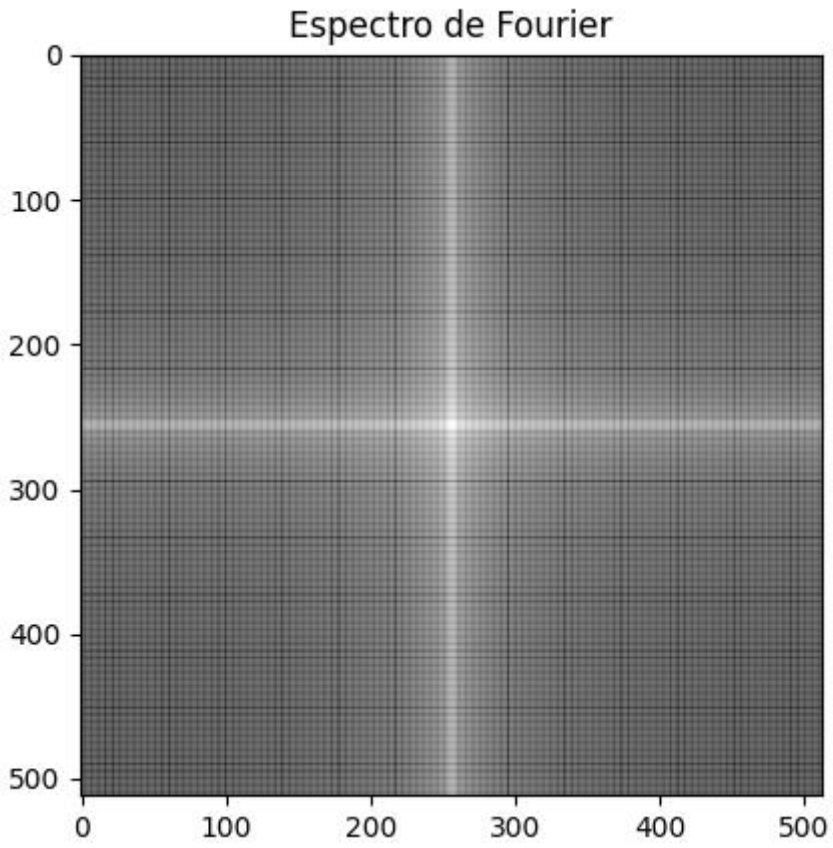


Código:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Criando uma imagem preta de 512x512
imagem = np.zeros((512, 512), dtype=np.uint8)

# Adicionando um quadrado branco no meio da imagem
cv2.rectangle(imagem, (204, 204), (308, 308), 255,
-1)

# Visualizando a imagem
plt.imshow(imagem, cmap='gray')
```

```
plt.title("Imagem Original")
plt.show()
```

b) calcular e visualizar seu espectro de Fourier (amplitudes);

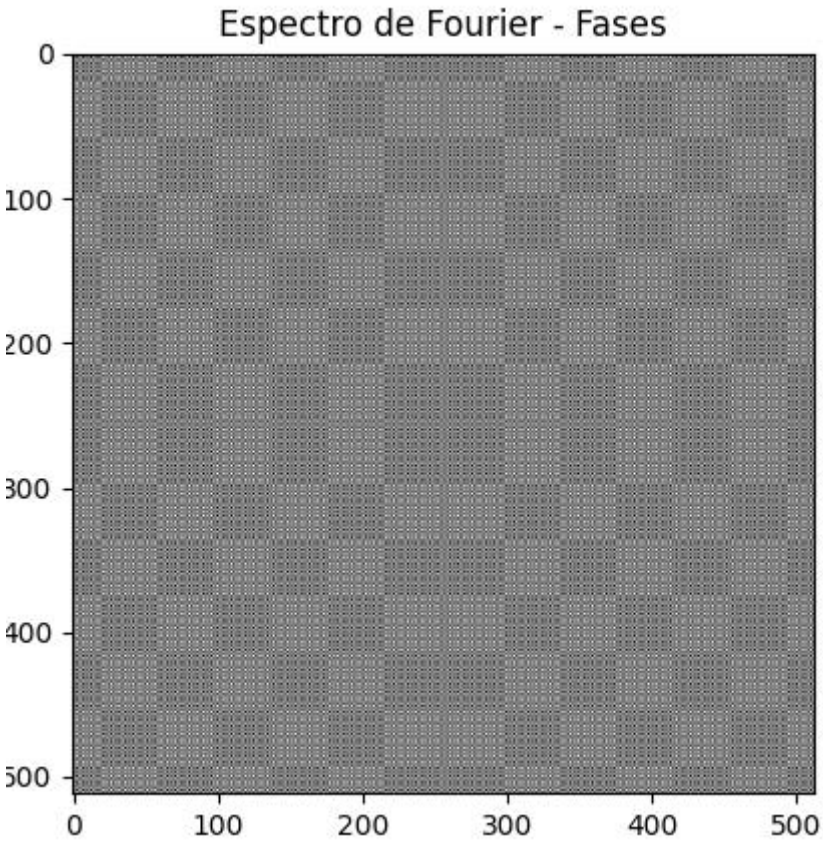


Código:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Imagem 512x512
imagem = np.zeros((512, 512), dtype=np.uint8)

# quadrado branco
cv2.rectangle(imagem, (204, 204), (308, 308), 255,
-1)
```

```
# Calculando a Transformada de Fourier
f = np.fft.fft2(imagem)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))


# exibição
plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Espectro de Fourier')
plt.show()
```

c) calcular e visualizar seu espectro de Fourier (fases);

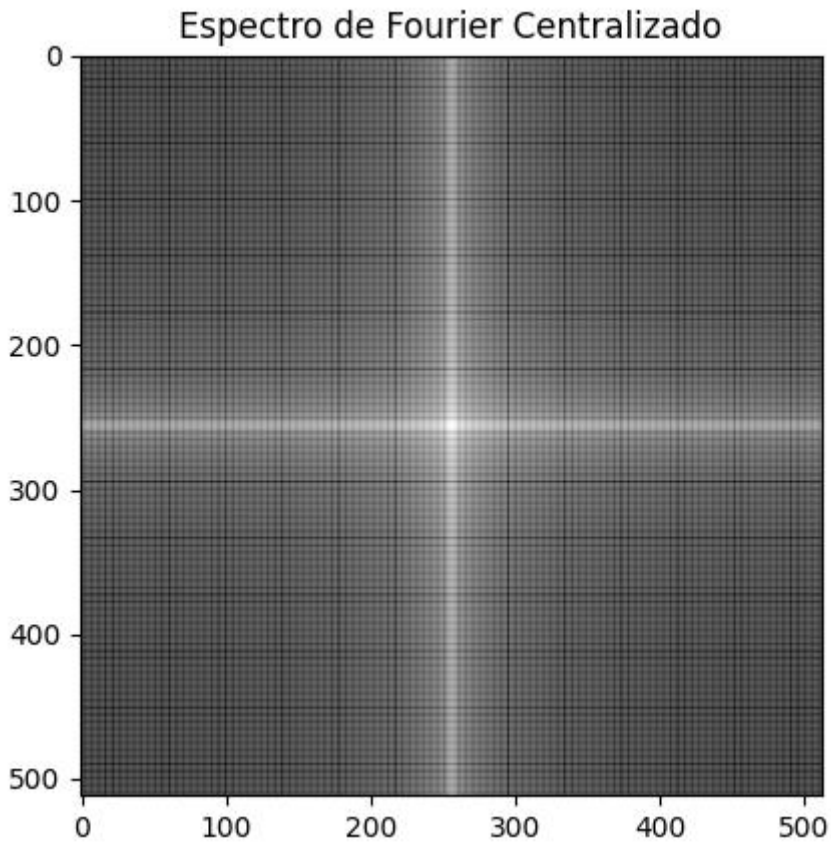


Código:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

```
# Imagem 512x512
imagem = np.zeros((512, 512), dtype=np.uint8)

# quadrado branco
cv2.rectangle(imagem, (204, 204), (308, 308), 255,
-1)

# Calculando a Transformada de Fourier
f = np.fft.fft2(imagem)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20*np.log(np.abs(fshift))

# Calculando as fases
fase_spectrum = np.angle(fshift)

# Exibição
plt.imshow(fase_spectrum, cmap = 'gray')
plt.title('Espectro de Fourier - Fases')
plt.show()
```

d) obter e visualizar seu espectro de Fourier centralizado;

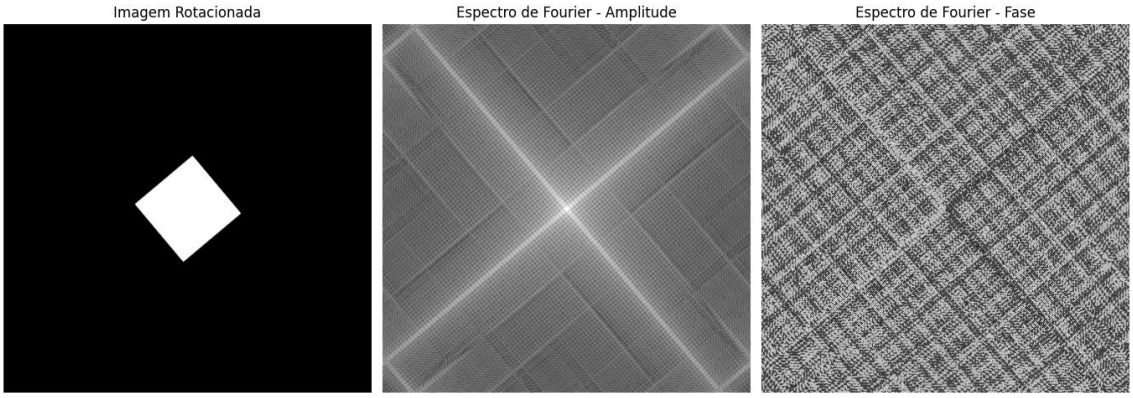


Código:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Imagem 512x512
imagem = np.zeros((512, 512), dtype=np.uint8)

# quadrado branco
cv2.rectangle(imagem, (204, 204), (308, 308), 255,
-1)

# Calculando a Transformada de Fourier
f = np.fft.fft2(imagem)
fshift = np.fft.fftshift(f)  # Centralizando o
espectro
```

```
magnitude_spectrum_centered = 20 *
np.log(np.abs(fshift) + 1)  # Adicionamos 1 para
evitar log(0)

# Visualização do espectro centralizado
plt.imshow(magnitude_spectrum_centered,
cmap='gray')
plt.title('Espectro de Fourier Centralizado')
plt.show()
```

e) Aplique uma rotação de 40º no quadrado e repita os passo b-d;



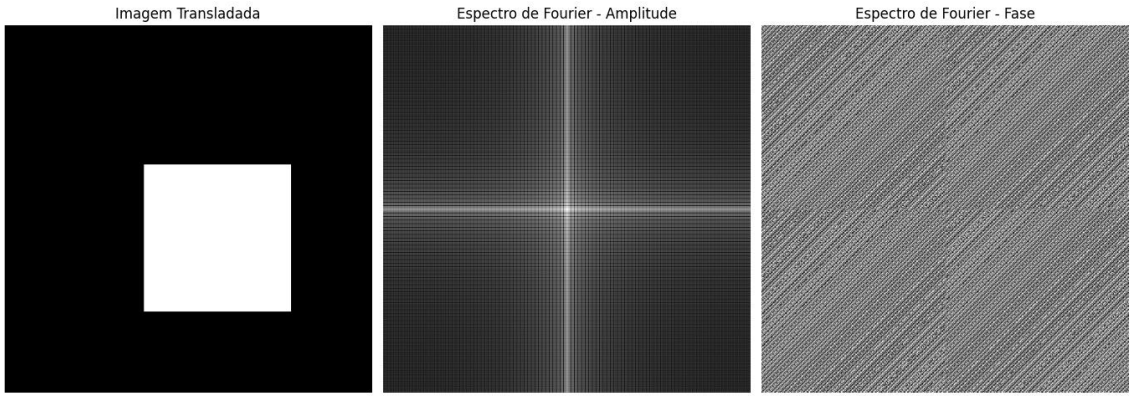
Código:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Imagem 512x512
imagem = np.zeros((512, 512), dtype=np.uint8)

# quadrado branco
cv2.rectangle(imagem, (204, 204), (308, 308), 255,
-1)

# Aplicando rotação de 40º na imagem
rows, cols = imagem.shape
```

```
M = cv2.getRotationMatrix2D((cols/2, rows/2), 40,
1)
imagem_rotacionada = cv2.warpAffine(imagem, M,
(cols, rows))

# b) Calcular e visualizar o espectro de Fourier
(amplitudes) da imagem rotacionada
f_rot = np.fft.fft2(imagem_rotacionada)
fshift_rot = np.fft.fftshift(f_rot)
magnitude_spectrum_rot =
20*np.log(np.abs(fshift_rot))

# c) Calcular e visualizar o espectro de Fourier
(fases) da imagem rotacionada
fase_spectrum_rot = np.angle(fshift_rot)

# Utilizando subplots para exibir as imagens lado a
lado
fig, axs = plt.subplots(1, 3, figsize=(15,5))

# Imagem rotacionada
axs[0].imshow(imagem_rotacionada, cmap='gray')
axs[0].set_title('Imagem Rotacionada')
axs[0].axis('off')

# Espectro de Amplitude
axs[1].imshow(magnitude_spectrum_rot, cmap='gray')
axs[1].set_title('Espectro de Fourier - Amplitude')
axs[1].axis('off')

# Espectro de Fase
axs[2].imshow(fase_spectrum_rot, cmap='gray')
axs[2].set_title('Espectro de Fourier - Fase')
axs[2].axis('off')
```

```
plt.tight_layout()
plt.show()
```

f) Aplique uma translação nos eixos x e y no quadrado e repita os passo b-d;

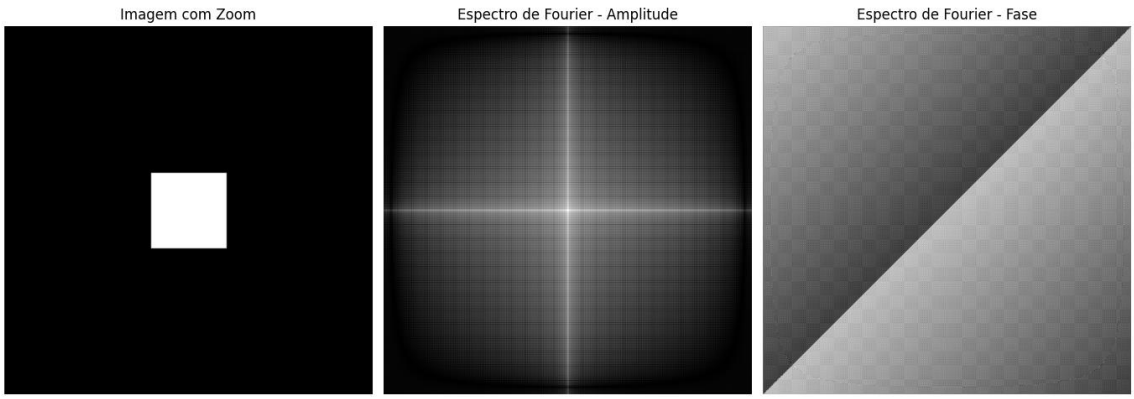


Código:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Criação da imagem original: um quadrado branco em
um fundo preto
imagem = np.zeros((512, 512), dtype=np.uint8)
cv2.rectangle(imagem, (154, 154), (358, 358), 255,
-1)

# Aplicando translação de 40 pixels nos eixos x e y
na imagem original
translacao = np.float32([[1, 0, 40], [0, 1, 40]])
imagem_transladada = cv2.warpAffine(imagem,
translacao, (512, 512))

# b) Calculando o espectro de Fourier (amplitudes)
da imagem transladada
f_trans = np.fft.fft2(imagem_transladada)
fshift_trans = np.fft.fftshift(f_trans)
```

```
magnitude_spectrum_trans =
20*np.log(np.abs(fshift_trans) + 1)  # +1 para
evitar log(0)

# c) Calculando o espectro de Fourier (fases) da
imagem transladada
fase_spectrum_trans = np.angle(fshift_trans)

# Usando subplots para exibir as imagens lado a
lado
fig, axs = plt.subplots(1, 3, figsize=(15,5))

# Imagem transladada
axs[0].imshow(imagem_transladada, cmap='gray')
axs[0].set_title('Imagem Transladada')
axs[0].axis('off')

# Espectro de Amplitude
axs[1].imshow(magnitude_spectrum_trans,
cmap='gray')
axs[1].set_title('Espectro de Fourier - Amplitude')
axs[1].axis('off')

# Espectro de Fase
axs[2].imshow(fase_spectrum_trans, cmap='gray')
axs[2].set_title('Espectro de Fourier - Fase')
axs[2].axis('off')

plt.tight_layout()
plt.show()
```

g) Aplique um zoom na imagem e repita os passo b-d;



Código:

```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Imagem 512x512
imagem = np.zeros((512, 512), dtype=np.uint8)

# quadrado branco
cv2.rectangle(imagem, (204, 204), (308, 308), 255,
-1)

# Aplicando zoom: reduzindo a imagem em 50% e
depois aumentando para o tamanho
imagem_zoom = cv2.resize(imagem, (512, 512))
imagem_zoom = cv2.resize(imagem_zoom, (1024, 1024))

# Calculando o espectro de Fourier (amplitudes) da
imagem com zoom
f_zoom = np.fft.fft2(imagem_zoom)
fshift_zoom = np.fft.fftshift(f_zoom)
magnitude_spectrum_zoom =
20*np.log(np.abs(fshift_zoom) + 1)

# Calculando o espectro de Fourier (fases) da
imagem com zoom
```

```
fase_spectrum_zoom = np.angle(fshift_zoom)

# Usando subplots para exibir as imagens lado a
lado
fig, axs = plt.subplots(1, 3, figsize=(15,5))

# Imagem com zoom
axs[0].imshow(imagem_zoom, cmap='gray')
axs[0].set_title('Imagem com Zoom')
axs[0].axis('off')

# Espectro de Amplitude
axs[1].imshow(magnitude_spectrum_zoom, cmap='gray')
axs[1].set_title('Espectro de Fourier - Amplitude')
axs[1].axis('off')

# Espectro de Fase
axs[2].imshow(fase_spectrum_zoom, cmap='gray')
axs[2].set_title('Espectro de Fourier - Fase')
axs[2].axis('off')

plt.tight_layout()
plt.show()
```

h) Explique o que acontece com a transformada de Fourier quando é aplicado a rotação, translação e zoom.

A Transformada de Fourier é uma ferramenta essencial para analisar sinais e imagens, permitindo desmembrar uma função ou imagem em suas partes de frequência. Quando aplicada a uma imagem com operações de rotação, tradução e zoom, isso modifica a representação da imagem no domínio de frequência da Transformada de Fourier.

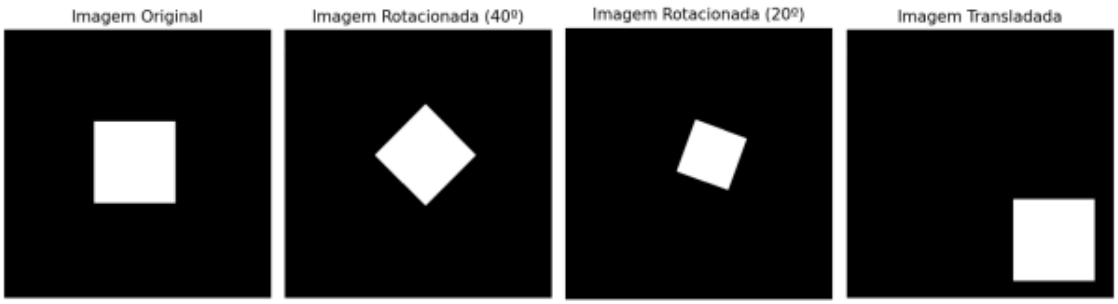
Na rotação, a Transformada de Fourier gira na mesma proporção que a imagem, mudando apenas a orientação das frequências. A tradução desloca a Transformada de Fourier na direção correspondente, mantendo a magnitude das frequências, mas alterando a fase. No caso do zoom, tanto a magnitude

quanto a localização das frequências são afetadas: um zoom in espalha as frequências no domínio de frequência, enquanto um zoom out se agrupa mais.

É importante ressaltar que essas operações alteram a representação no domínio de frequência, mas a informação fundamental da imagem permanece. No entanto, ao analisar imagens com essas transformações, é necessário considerá-las e adaptar o processamento subsequente de acordo com as características da imagem transformada.

2. Crie filtros passa-baixa do tipo ideal, butterworth e gaussiano e aplique-o às imagens disponibilizadas. Visualize o seguinte:

a) a imagem inicial;



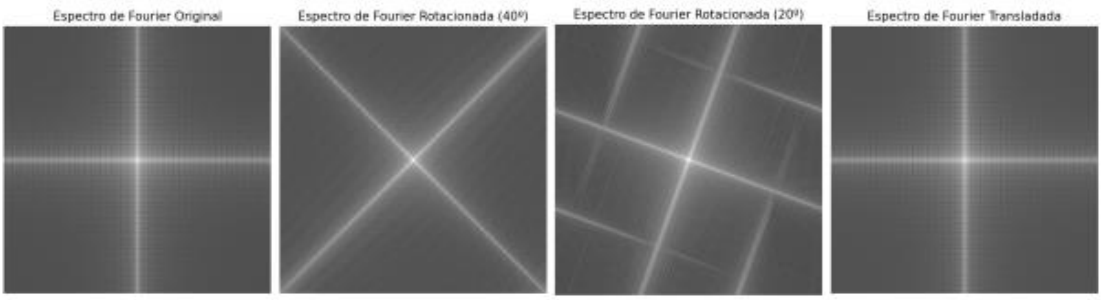
Código:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

sinc_original_path = './imagem/sinc_original.png'
sinc_original_menor_path =
'./imagem/sinc_menor.png'
sinc_rot_path = './imagem/sinc_rot.png'
sinc_rot2_path = './imagem/sinc_rot2.png'
sinc_trans_path = './imagem/sinc_trans.png'

# Ler as imagens
sinc_original = cv2.imread(sinc_original_path,
cv2.IMREAD_GRAYSCALE)
sinc_original_menor =
cv2.imread(sinc_original_menor_path,
cv2.IMREAD_GRAYSCALE)
```

```
sinc_rot = cv2.imread(sinc_rot_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot2 = cv2.imread(sinc_rot2_path,
cv2.IMREAD_GRAYSCALE)
sinc_trans = cv2.imread(sinc_trans_path,
cv2.IMREAD_GRAYSCALE)

# Crie uma figura para organizar as imagens e
legendas
plt.figure(figsize=(15, 5))

# Imagem Original
plt.subplot(151)
plt.imshow(sinc_original, cmap='gray')
plt.title('Imagem Original')
plt.axis('off')

# Rotacionada (40º)
plt.subplot(152)
plt.imshow(sinc_rot, cmap='gray')
plt.title('Imagem Rotacionada (40º)')
plt.axis('off')

# Rotacionada (20º)
plt.subplot(153)
plt.imshow(sinc_rot2, cmap='gray')
plt.title('Imagem Rotacionada (20º)')
plt.axis('off')

# Transladada
plt.subplot(154)
plt.imshow(sinc_trans, cmap='gray')
plt.title('Imagem Transladada')
plt.axis('off')
```

```
plt.tight_layout()
plt.show()
```

b) a imagem do spectro de fourier;



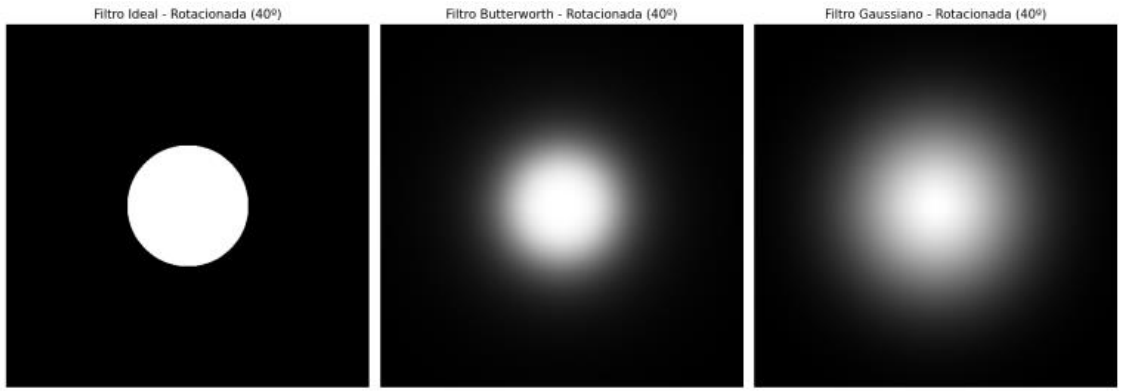
Código:

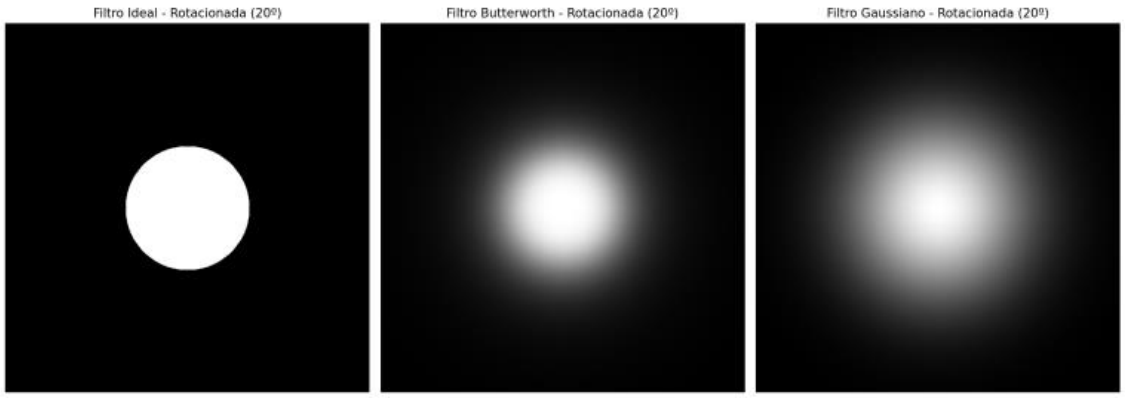
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

sinc_original_path = './imagem/sinc_original.png'
sinc_original_menor_path =
'./imagem/sinc_original_menor.png'
sinc_rot_path = './imagem/sinc_rot.png'
sinc_rot2_path = './imagem/sinc_rot2.png'
sinc_trans_path = './imagem/sinc_trans.png'

# Ler as imagens
sinc_original = cv2.imread(sinc_original_path,
cv2.IMREAD_GRAYSCALE)
sinc_original_menor =
cv2.imread(sinc_original_menor_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot = cv2.imread(sinc_rot_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot2 = cv2.imread(sinc_rot2_path,
cv2.IMREAD_GRAYSCALE)
```

```
sinc_trans = cv2.imread(sinc_trans_path,
cv2.IMREAD_GRAYSCALE)

def fourier_spectrum(image):
# Computa a transformada de Fourier 2D
    f = np.fft.fft2(image)

# Centraliza as frequências baixas
    fshift = np.fft.fftshift(f)

# Calcula a magnitude e aplica o logaritmo para
melhor visualização
    magnitude_spectrum = np.log(np.abs(fshift) + 1)
    return magnitude_spectrum

# Computa o espectro de Fourier para todas as
imagens
spectrum_original = fourier_spectrum(sinc_original)
15
spectrum_rot = fourier_spectrum(sinc_rot)
spectrum_rot2 = fourier_spectrum(sinc_rot2)
spectrum_trans = fourier_spectrum(sinc_trans)

# Organização das subplots em uma única linha com 4
colunas
plt.figure(figsize=(20, 5))

# Espectro de Fourier da Imagem Original
plt.subplot(141)
plt.imshow(spectrum_original, cmap='gray')
plt.title('Espectro de Fourier Original')
plt.axis('off')

# Espectro de Fourier da Imagem Rotacionada 40º
```

```
plt.subplot(142)
plt.imshow(spectrum_rot, cmap='gray')
plt.title('Espectro de Fourier Rotacionada (40º)')
plt.axis('off')

# Espectro de Fourier da Imagem Rotacionada 20º
plt.subplot(143)
plt.imshow(spectrum_rot2, cmap='gray')
plt.title('Espectro de Fourier Rotacionada (20º)')
plt.axis('off')

# Espectro de Fourier da Imagem Transladada
plt.subplot(144)
plt.imshow(spectrum_trans, cmap='gray')
plt.title('Espectro de Fourier Transladada')
plt.axis('off')

# Ajusta o layout e mostra a figura
plt.tight_layout()
plt.show()
```

c) a imagem de cada filtro;





Código:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

sinc_original_path = './imagem/sinc_original.png'
sinc_original_menor_path =
'./imagem/sinc_original_menor.png'
sinc_rot_path = './imagem/sinc_rot.png'
sinc_rot2_path = './imagem/sinc_rot2.png'
sinc_trans_path = './imagem/sinc_trans.png'

# Ler as imagens
sinc_original = cv2.imread(sinc_original_path,
cv2.IMREAD_GRAYSCALE)
sinc_original_menor =
cv2.imread(sinc_original_menor_path,
cv2.IMREAD_GRAYSCALE)
```

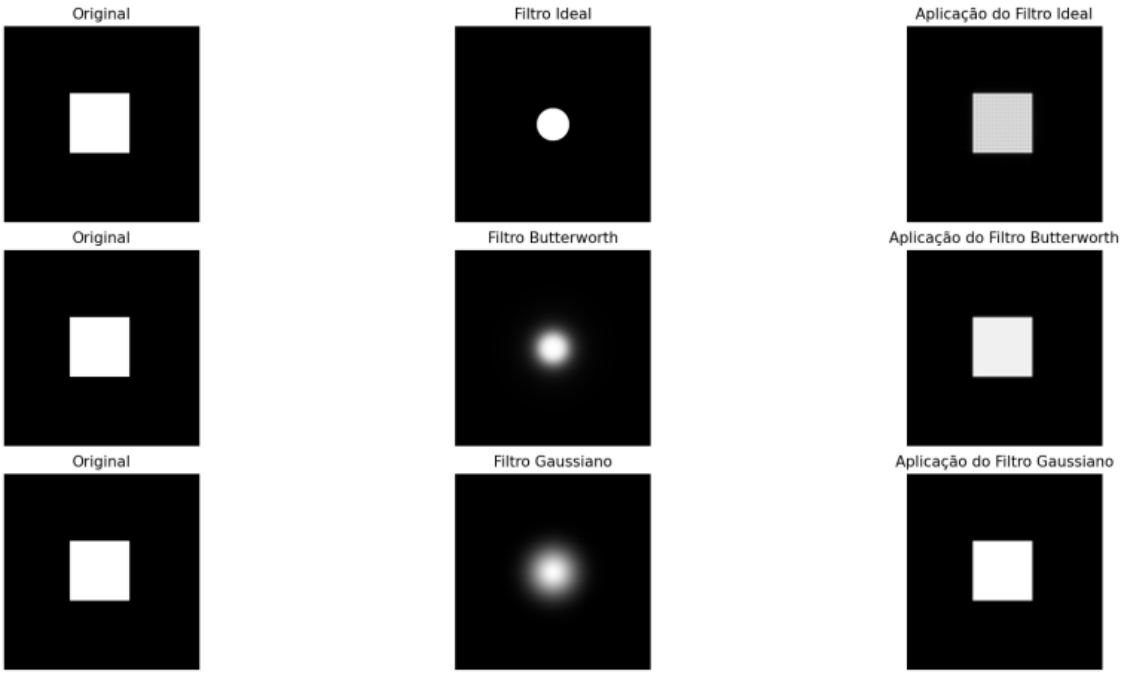
```
sinc_rot = cv2.imread(sinc_rot_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot2 = cv2.imread(sinc_rot2_path,
cv2.IMREAD_GRAYSCALE)
sinc_trans = cv2.imread(sinc_trans_path,
cv2.IMREAD_GRAYSCALE)

def create_filters(image, cutoff):

# Filtro passa-baixa Ideal
    ideal_filter = ideal_lowpass_filter(image,
cutoff)

# Filtro passa-baixa Butterworth
    butter_filter =
butterworth_lowpass_filter(image, cutoff)

# Filtro passa-baixa Gaussiano
    gaussian_filter =
gaussian_lowpass_filter(image, cutoff)
    return ideal_filter, butter_filter,
gaussian_filter

def distance(point1, point2):
    return np.sqrt((point1[0] - point2[0]) ** 2 +
(point1[1] - point2[1]) ** 2)

def ideal_lowpass_filter(image, cutoff):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))

    for x in range(cols):
        for y in range(rows):
```

```
            if distance((y, x), center) < cutoff:
filter[y, x] = 1

    return filter

def butterworth_lowpass_filter(image, cutoff,
order=2):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))

    for x in range(cols):
        for y in range(rows):
            filter[y, x] = 1 / (1 + (distance((y,
x), center) / cutoff) ** (2 * order))
    return filter

def gaussian_lowpass_filter(image, cutoff):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))

    for x in range(cols):
        for y in range(rows):
            filter[y, x] = np.exp(-(distance((y,
x), center) ** 2) / (2 * (cutoff ** 2)))
    return filter
cutoff = 30

def normalize_image(image):
    min_val = np.min(image)
    max_val = np.max(image)
    return (image - min_val) / (max_val - min_val)
```

```
cutoff = 100 # Ajuste conforme necessário

images = [sinc_original, sinc_rot, sinc_rot2,
sinc_trans]
titles = ['Original', 'Rotacionada (40º)',
'Rotacionada (20º)',
'Transladada']

for idx, image in enumerate(images):
    # Cria os filtros
    ideal_filter, butter_filter, gaussian_filter =
create_filters(image, cutoff)

# Normaliza para melhor visualização
ideal_filter = normalize_image(ideal_filter)
butter_filter = normalize_image(butter_filter)
gaussian_filter = normalize_image(gaussian_filter)

# Exibe os filtros
plt.figure(figsize=(15, 5))

# Filtro Ideal
plt.subplot(131)
plt.imshow(ideal_filter, cmap='gray')
plt.title(f'Filtro Ideal - {titles[idx]}')
plt.axis('off')

# Filtro Butterworth
plt.subplot(132)
plt.imshow(butter_filter, cmap='gray')
plt.title(f'Filtro Butterworth - {titles[idx]}')
plt.axis('off')

# Filtro Gaussiano
```

```
plt.subplot(133)
plt.imshow(gaussian_filter, cmap='gray')
plt.title(f'Filtro Gaussiano - {titles[idx]}')
plt.axis('off')

# Ajusta o layout e mostra a figura
plt.tight_layout()
plt.show()
```

d) a imagem resultante após aplicação de cada filtro.



Código:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

sinc_original_path = './imagem/sinc_original.png'
sinc_original_menor_path =
'./imagem/sinc_original_menor.png'
```

```
sinc_rot_path = './imagem/sinc_rot.png'
sinc_rot2_path = './imagem/sinc_rot2.png'
sinc_trans_path = './imagem/sinc_trans.png'

# Ler as imagens
sinc_original = cv2.imread(sinc_original_path,
cv2.IMREAD_GRAYSCALE)
sinc_original_menor =
cv2.imread(sinc_original_menor_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot = cv2.imread(sinc_rot_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot2 = cv2.imread(sinc_rot2_path,
cv2.IMREAD_GRAYSCALE)
sinc_trans = cv2.imread(sinc_trans_path,
cv2.IMREAD_GRAYSCALE)

def create_filters(image, cutoff):
    # Filtro passa-baixa Ideal
    ideal_filter = ideal_lowpass_filter(image,
cutoff)

    # Filtro passa-baixa Butterworth
    butter_filter =
butterworth_lowpass_filter(image, cutoff)

    # Filtro passa-baixa Gaussiano
    gaussian_filter =
gaussian_lowpass_filter(image, cutoff)

    return ideal_filter, butter_filter,
gaussian_filter

def distance(point1, point2):
```

```
    return np.sqrt((point1[0] - point2[0]) ** 2 +
(point1[1] - point2[1]) ** 2)

def ideal_lowpass_filter(image, cutoff):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            if distance((y, x), center) < cutoff:
                filter[y, x] = 1
    return filter

def butterworth_lowpass_filter(image, cutoff,
order=2):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            filter[y, x] = 1 / (1 + (distance((y,
x), center) / cutoff) ** (2 * order))
    return filter

def gaussian_lowpass_filter(image, cutoff):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            filter[y, x] = np.exp(-(distance((y,
x), center) ** 2) / (2* (cutoff ** 2)))
    return filter
cutoff = 30
```

```
def normalize_image(image):
    min_val = np.min(image)
    max_val = np.max(image)
    return (image - min_val) / (max_val - min_val)

def apply_filter(image, filter):
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    fshift = fshift * filter
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)
    return img_back

# Define um cutoff para os filtros
cutoff = 50

# Cria os filtros
ideal_filter = ideal_lowpass_filter(sinc_original,
cutoff)
butter_filter =
butterworth_lowpass_filter(sinc_original, cutoff)
gaussian_filter =
gaussian_lowpass_filter(sinc_original, cutoff)

# Aplica os filtros
img_ideal = apply_filter(sinc_original,
ideal_filter)
img_butter = apply_filter(sinc_original,
butter_filter)
img_gaussian = apply_filter(sinc_original,
gaussian_filter)
```

```
# Visualização
fig, axs = plt.subplots(3, 3, figsize=(15,15))

# Imagens originais
axs[0, 0].imshow(sinc_original, cmap='gray')
axs[0, 0].set_title('Original')
axs[0, 1].imshow(ideal_filter, cmap='gray')
axs[0, 1].set_title('Filtro Ideal')
axs[0, 2].imshow(img_ideal, cmap='gray')
axs[0, 2].set_title('Aplicação do Filtro Ideal')

axs[1, 0].imshow(sinc_original, cmap='gray')
axs[1, 0].set_title('Original')
axs[1, 1].imshow(butter_filter, cmap='gray')
axs[1, 1].set_title('Filtro Butterworth')
axs[1, 2].imshow(img_butter, cmap='gray')
axs[1, 2].set_title('Aplicação do Filtro
Butterworth')

axs[2, 0].imshow(sinc_original, cmap='gray')
axs[2, 0].set_title('Original')
axs[2, 1].imshow(gaussian_filter, cmap='gray')
axs[2, 1].set_title('Filtro Gaussiano')
axs[2, 2].imshow(img_gaussian, cmap='gray')
axs[2, 2].set_title('Aplicação do Filtro
Gaussiano')

for ax in axs.ravel():
    ax.axis('off')

plt.tight_layout()
plt.show()
```

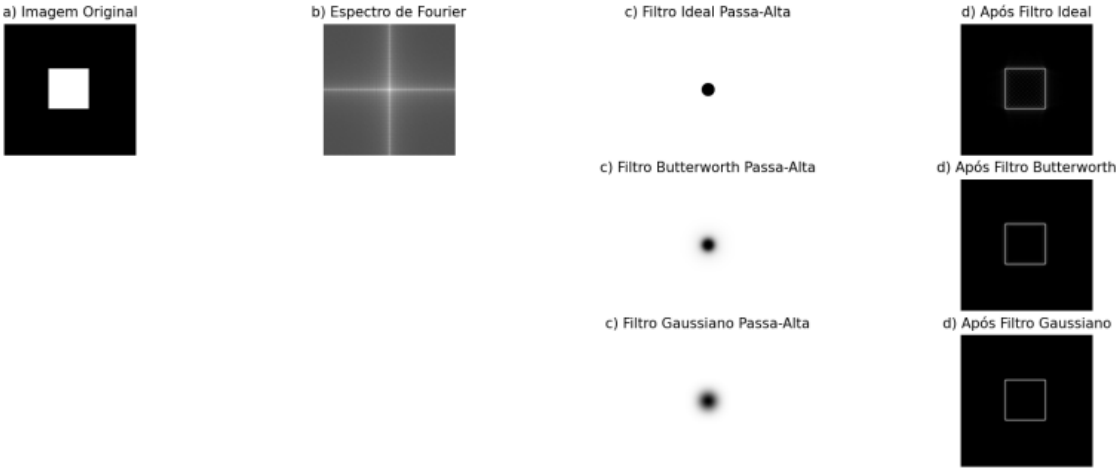
3. Crie um filtro passa-alta do tipo ideal, butterworth e gaussiano e aplique-o às imagens disponibilizadas. Visualize os mesmos dados da tarefa anterior:

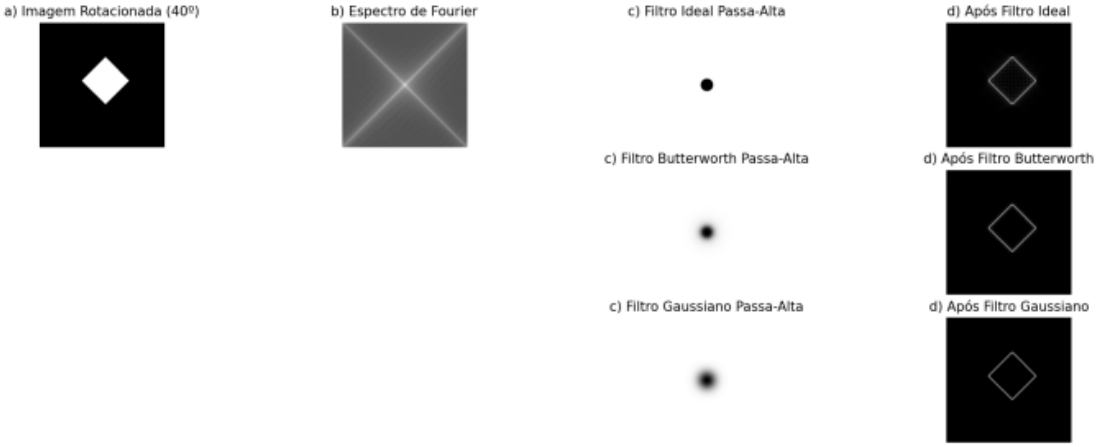
a) a imagem inicial;

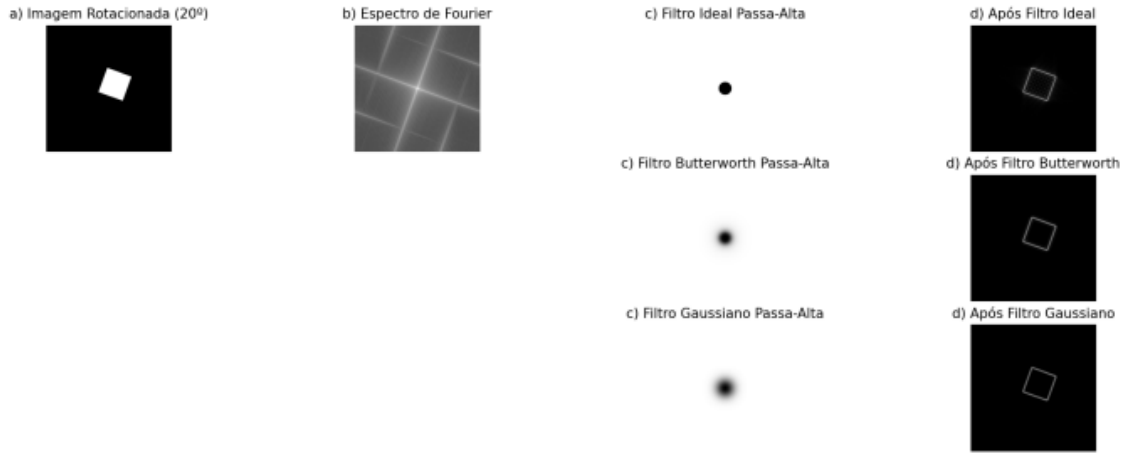
b) a imagem do spectro de fourier;

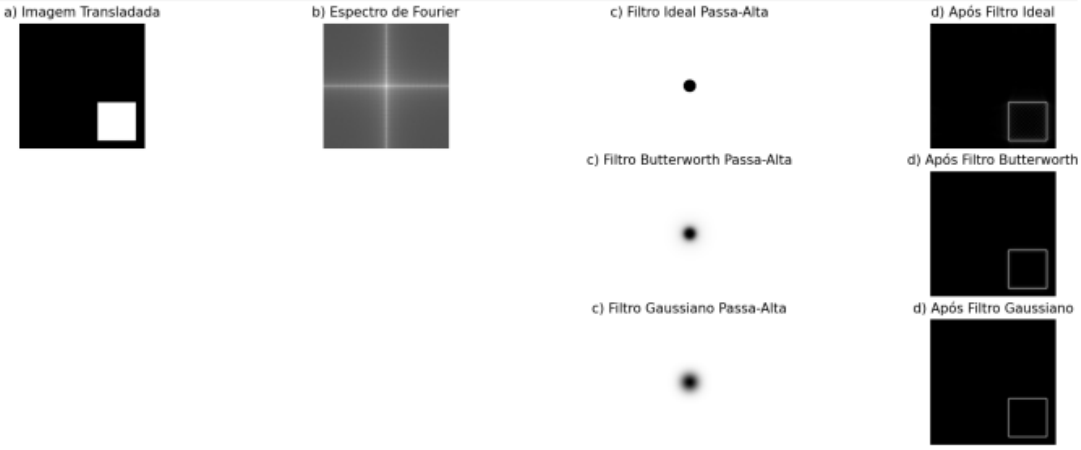
c) a imagem de cada filtro;

d) a imagem resultante após aplicação de cada filtro.









Código:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

sinc_original_path = './imagem/sinc_original.png'
sinc_original_menor_path =
'./imagem/sinc_original_menor.png'
sinc_rot_path = './imagem/sinc_rot.png'
sinc_rot2_path = './imagem/sinc_rot2.png'
sinc_trans_path = './imagem/sinc_trans.png'

# Ler as imagens
```

```
sinc_original = cv2.imread(sinc_original_path,
cv2.IMREAD_GRAYSCALE)
sinc_original_menor =
cv2.imread(sinc_original_menor_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot = cv2.imread(sinc_rot_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot2 = cv2.imread(sinc_rot2_path,
cv2.IMREAD_GRAYSCALE)
sinc_trans = cv2.imread(sinc_trans_path,
cv2.IMREAD_GRAYSCALE)

def fourier_spectrum(image):
# Computa a transformada de Fourier 2D
    f = np.fft.fft2(image)

    # Centraliza as frequências baixas
    fshift = np.fft.fftshift(f)

    # Calcula a magnitude e aplica o logaritmo para
melhor visualização
    magnitude_spectrum = np.log(np.abs(fshift) + 1)
    return magnitude_spectrum

def distance(point1, point2):
    return np.sqrt((point1[0] - point2[0]) ** 2 +
(point1[1] - point2[1]) ** 2)

def ideal_lowpass_filter(image, cutoff):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
```

```
            if distance((y, x), center) < cutoff:
                filter[y, x] = 1
    return filter

def butterworth_lowpass_filter(image, cutoff,
order=2):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            filter[y, x] = 1 / (1 + (distance((y,
x), center) / cutoff) ** (2 * order))
    return filter

def gaussian_lowpass_filter(image, cutoff):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            filter[y, x] = np.exp(-(distance((y,
x), center) ** 2) / (2* (cutoff ** 2)))
    return filter
cutoff = 30

def normalize_image(image):
    min_val = np.min(image)
    max_val = np.max(image)
    return (image - min_val) / (max_val - min_val)

def apply_filter(image, filter):
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
```

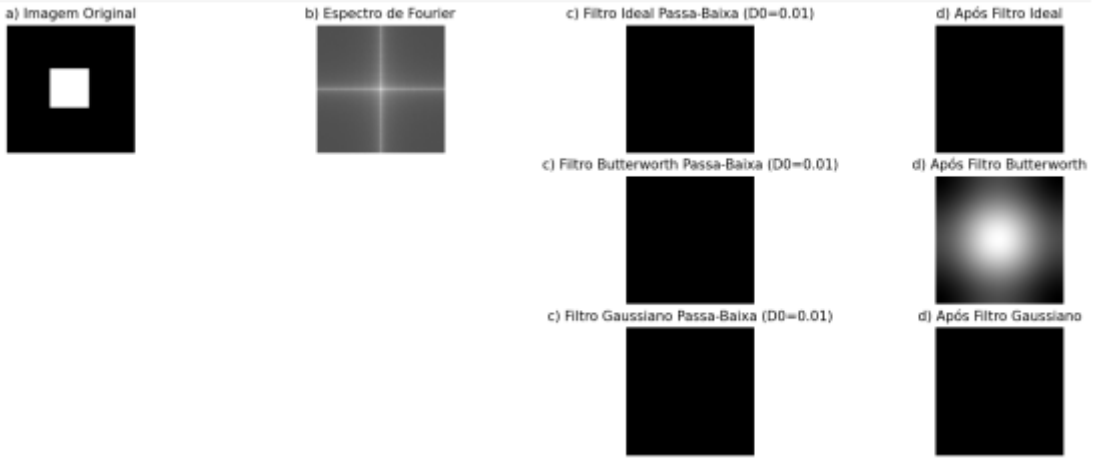
```
    fshift = fshift * filter
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)
    return img_back

def ideal_highpass_filter(image, cutoff):
    return 1 - ideal_lowpass_filter(image, cutoff)

def butterworth_highpass_filter(image, cutoff,
order=2):
    return 1 - butterworth_lowpass_filter(image,
cutoff, order)

def gaussian_highpass_filter(image, cutoff):
    return 1 - gaussian_lowpass_filter(image,
cutoff)

# Função para aplicar o filtro usando transformada
de Fourier
def apply_filter(image, filter):
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    fshift = fshift * filter
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)
    return img_back
    images = [sinc_original, sinc_rot, sinc_rot2,
sinc_trans]
    titles = ['Original', 'Rotacionada (40º)',
'Rotacionada (20º)','Transladada']

for idx, image in enumerate(images):
```

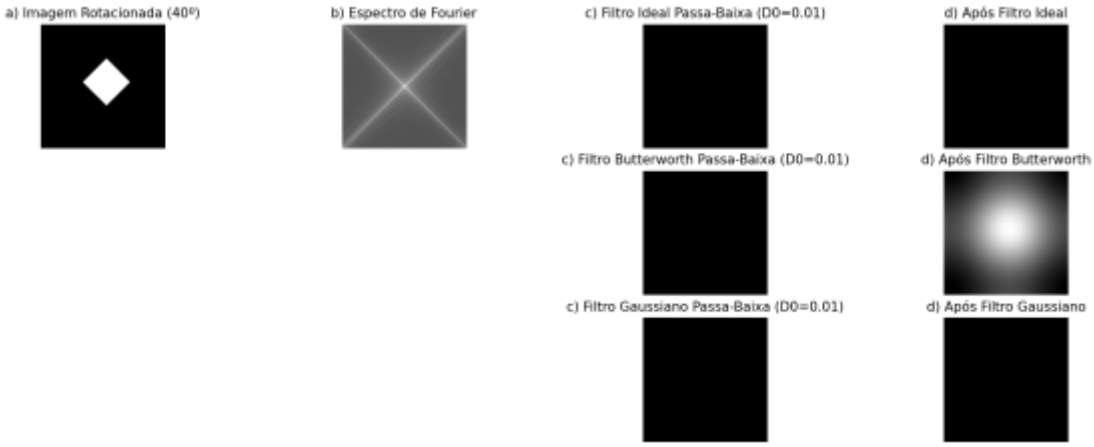
```
    # Fourier
    spectrum = fourier_spectrum(image)

# Criação dos filtros
ideal_hp = ideal_highpass_filter(image, cutoff)
butter_hp = butterworth_highpass_filter(image,
cutoff)
gaussian_hp = gaussian_highpass_filter(image,
cutoff)

# Aplicação dos filtros
result_ideal = apply_filter(image, ideal_hp)
result_butter = apply_filter(image, butter_hp)
result_gaussian = apply_filter(image, gaussian_hp)

# Exibição
plt.figure(figsize=(20, 10))

# Imagem original
plt.subplot(4, 4, 1)
plt.imshow(image, cmap='gray')
plt.title(f'a) Imagem {titles[idx]}')
plt.axis('off')

# Espectro de Fourier
plt.subplot(4, 4, 2)
plt.imshow(spectrum, cmap='gray')
plt.title('b) Espectro de Fourier')
plt.axis('off')

# Filtro passa-alta Ideal
plt.subplot(4, 4, 3)
plt.imshow(ideal_hp, cmap='gray')
plt.title('c) Filtro Ideal Passa-Alta')
```

```
plt.axis('off')

# Resultado filtro Ideal
plt.subplot(4, 4, 4)
plt.imshow(result_ideal, cmap='gray')
plt.title('d) Após Filtro Ideal')
plt.axis('off')

# Filtro passa-alta Butterworth
plt.subplot(4, 4, 7)
plt.imshow(butter_hp, cmap='gray')
plt.title('c) Filtro Butterworth Passa-Alta')
plt.axis('off')

# Resultado filtro Butterworth
plt.subplot(4, 4, 8)
plt.imshow(result_butter, cmap='gray')
plt.title('d) Após Filtro Butterworth')
plt.axis('off')

# Filtro passa-alta Gaussiano
plt.subplot(4, 4, 11)
plt.imshow(gaussian_hp, cmap='gray')
plt.title('c) Filtro Gaussiano Passa-Alta')
plt.axis('off')

# Resultado filtro Gaussiano
plt.subplot(4, 4, 12)
plt.imshow(result_gaussian, cmap='gray')
plt.title('d) Após Filtro Gaussiano')
plt.axis('off')

# Ajusta o layout e mostra a figura
plt.tight_layout()
```

```
plt.show()
```

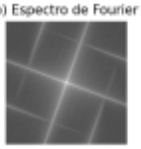
4. Varie o parâmetro de frequência de corte no filtro passa-baixa criado na tarefa 2. Por exemplo, tome valores de D_0 iguais a 0,01, 0,05, 0,5. A imagem inicial é igual à anterior. Visualize as imagens dos filtros e as imagens resultantes. Explique os resultados.





a) Imagem Rotacionada (20º)

b) Espectro de Fourier



c) Filtro Ideal Passa-Baixa (D0=0.01)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Baixa (D0=0.01)

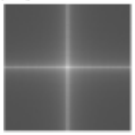
d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Baixa (D0=0.01)

d) Após Filtro Gaussiano

a) Imagem Transladada

b) Espectro de Fourier



c) Filtro Ideal Passa-Baixa (D0=0.01)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Baixa (D0=0.01)

d) Após Filtro Butterworth

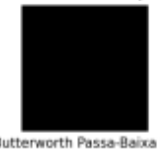
c) Filtro Gaussiano Passa-Baixa (D0=0.01)

d) Após Filtro Gaussiano

a) Imagem Original

b) Espectro de Fourier

c) Filtro Ideal Passa-Baixa (D0=0.05)



d) Após Filtro Ideal

c) Filtro Butterworth Passa-Baixa (D0=0.05)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Baixa (D0=0.05)

d) Após Filtro Gaussiano

a) Imagem Rotacionada (40º)

b) Espectro de Fourier

c) Filtro Ideal Passa-Baixa (D0=0.05)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Baixa (D0=0.05)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Baixa (D0=0.05)

d) Após Filtro Gaussiano

a) Imagem Rotacionada (20º)

b) Espectro de Fourier

c) Filtro Ideal Passa-Baixa (D0=0.05)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Baixa (D0=0.05)

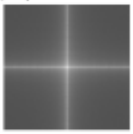
d) Após Filtro Butterwort

c) Filtro Gaussiano Passa-Baixa (D0=0.05)

d) Após Filtro Gaussianc

a) Imagem Transladada

b) Espectro de Fourier

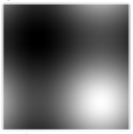


c) Filtro Ideal Passa-Baixa (D0=0.05)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Baixa (D0=0.05)

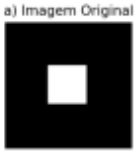
d) Após Filtro Butterworth



c) Filtro Gaussiano Passa-Baixa (D0=0.05)

d) Após Filtro Gaussiano

a) Imagem Original



b) Espectro de Fourier

c) Filtro Ideal Passa-Baixa (D0=0.5)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Baixa (D0=0.5)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Baixa (D0=0.5)

d) Após Filtro Gaussiano

a) Imagem Rotacionada (40º)

b) Espectro de Fourier

c) Filtro Ideal Passa-Baixa (D0=0.5)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Baixa (D0=0.5)

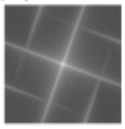
d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Baixa (D0=0.5)

d) Após Filtro Gaussiano

a) Imagem Rotacionada (20º)

b) Espectro de Fourier



c) Filtro Ideal Passa-Baixa (D0=0.5)

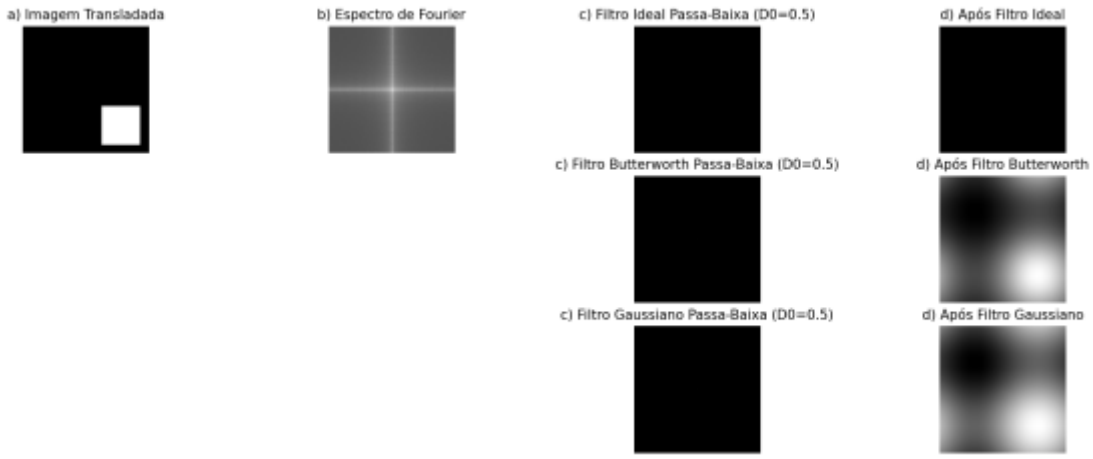
d) Após Filtro Ideal

c) Filtro Butterworth Passa-Baixa (D0=0.5)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Baixa (D0=0.5)

d) Após Filtro Gaussiano



Código:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

sinc_original_path = './imagem/sinc_original.png'
sinc_original_menor_path =
'./imagem/sinc_original_menor.png'
sinc_rot_path = './imagem/sinc_rot.png'
sinc_rot2_path = './imagem/sinc_rot2.png'
sinc_trans_path = './imagem/sinc_trans.png'

# Ler as imagens
sinc_original = cv2.imread(sinc_original_path,
cv2.IMREAD_GRAYSCALE)
sinc_original_menor =
cv2.imread(sinc_original_menor_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot = cv2.imread(sinc_rot_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot2 = cv2.imread(sinc_rot2_path,
cv2.IMREAD_GRAYSCALE)
sinc_trans = cv2.imread(sinc_trans_path,
cv2.IMREAD_GRAYSCALE)
```

```
def fourier_spectrum(image):
    # Computa a transformada de Fourier 2D
    f = np.fft.fft2(image)
    # Centraliza as frequências baixas
    fshift = np.fft.fftshift(f)
    # Calcula a magnitude e aplica o logaritmo para
melhor visualização
    magnitude_spectrum = np.log(np.abs(fshift) + 1)
    return magnitude_spectrum

def distance(point1, point2):
    return np.sqrt((point1[0] - point2[0]) ** 2 +
(point1[1] - point2[1]) ** 2)

def ideal_lowpass_filter(image, cutoff):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            if distance((y, x), center) < cutoff:
                filter[y, x] = 1
    return filter

def butterworth_lowpass_filter(image, cutoff,
order=2):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            filter[y, x] = 1 / (1 + (distance((y,
x), center) / cutoff) ** (2 * order))
```

```
    return filter

def gaussian_lowpass_filter(image, cutoff):
    rows, cols = image.shape
    center = (rows / 2, cols / 2)
    filter = np.zeros((rows, cols))
    for x in range(cols):
        for y in range(rows):
            filter[y, x] = np.exp(-(distance((y,
x), center) ** 2) / (2 * (cutoff ** 2)))
    return filter
cutoff = 30

def normalize_image(image):
    min_val = np.min(image)
    max_val = np.max(image)
    return (image - min_val) / (max_val - min_val)

def apply_filter(image, filter):
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    fshift = fshift * filter
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)
    return img_back

def ideal_highpass_filter(image, cutoff):
    return 1 - ideal_lowpass_filter(image, cutoff)

def butterworth_highpass_filter(image, cutoff,
order=2):
    return 1 - butterworth_lowpass_filter(image,
cutoff, order)
```

```
def gaussian_highpass_filter(image, cutoff):
    return 1 - gaussian_lowpass_filter(image,
cutoff)

# Função para aplicar o filtro usando transformada
de Fourier
def apply_filter(image, filter):

    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    fshift = fshift * filter
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)
    return img_back
images = [sinc_original, sinc_rot, sinc_rot2,
sinc_trans]
titles = ['Original', 'Rotacionada (40º)',
'Rotacionada (20º)',
'Transladada']

# Definindo os valores de D0 para variação
cutoffs = [0.01, 0.05, 0.5]
for cutoff in cutoffs:
    for idx, image in enumerate(images):

# Fourier
        spectrum = fourier_spectrum(image)

# Criação dos filtros
ideal_lp = ideal_lowpass_filter(image, cutoff)
butter_lp = butterworth_lowpass_filter(image,
cutoff)
```

```
gaussian_lp = gaussian_lowpass_filter(image,
cutoff)

# Aplicação dos filtros
result_ideal = apply_filter(image, ideal_lp)
result_butter = apply_filter(image, butter_lp)
result_gaussian = apply_filter(image, gaussian_lp)

# Exibição
plt.figure(figsize=(20, 10))

# Imagem original
plt.subplot(4, 4, 1)
plt.imshow(image, cmap='gray')
plt.title(f'a) Imagem {titles[idx]}')
plt.axis('off')

# Espectro de Fourier
plt.subplot(4, 4, 2)
plt.imshow(spectrum, cmap='gray')
plt.title('b) Espectro de Fourier')
plt.axis('off')

# Filtro passa-baixa Ideal
plt.subplot(4, 4, 3)
plt.imshow(ideal_lp, cmap='gray')
plt.title(f'c) Filtro Ideal Passa-Baixa
(D0={cutoff})')
plt.axis('off')

# Resultado filtro Ideal
plt.subplot(4, 4, 4)
plt.imshow(result_ideal, cmap='gray')
plt.title('d) Após Filtro Ideal')
```

```
plt.axis('off')

# Filtro passa-baixa Butterworth
plt.subplot(4, 4, 7)
plt.imshow(butter_lp, cmap='gray')
plt.title(f'c) Filtro Butterworth Passa-Baixa
(D0={cutoff})')
plt.axis('off')

# Resultado filtro Butterworth
plt.subplot(4, 4, 8)
plt.imshow(result_butter, cmap='gray')
plt.title('d) Após Filtro Butterworth')
plt.axis('off')

# Filtro passa-baixa Gaussiano
plt.subplot(4, 4, 11)
plt.imshow(gaussian_lp, cmap='gray')
plt.title(f'c) Filtro Gaussiano Passa-Baixa
(D0={cutoff})')
plt.axis('off')

# Resultado filtro Gaussiano
plt.subplot(4, 4, 12)
plt.imshow(result_gaussian, cmap='gray')
plt.title('d) Após Filtro Gaussiano')
plt.axis('off')

# Ajusta o layout e mostra a figura
plt.tight_layout()
plt.show()
```

Ao escolhermos D0 = 0,01 como a frequência de corte (muito baixa), estamos optando por reter apenas as partes da imagem com frequências extremamente baixas, incluindo o componente de baixa frequência DC. Isso resulta em uma

imagem altamente suavizada, quase como uma versão "borrada" da imagem original.

Com D0 = 0,05 (frequência de corte moderadamente baixa), estamos permitindo a preservação de mais componentes de frequência da imagem em comparação com D0 = 0,01. A imagem ainda terá alguma suavização, mas começarão a surgir detalhes mais sutis em comparação com a configuração anterior.

D0 = 0,5 (frequência de corte moderada) mantém a maioria dos componentes de frequência baixa da imagem. Isso resulta em uma imagem mais próxima do original, exibindo menos suavização em comparação com as configurações anteriores. À medida que D0 aumenta, mais detalhes finos da imagem tornam-se visíveis, pois frequências mais altas são retidas.

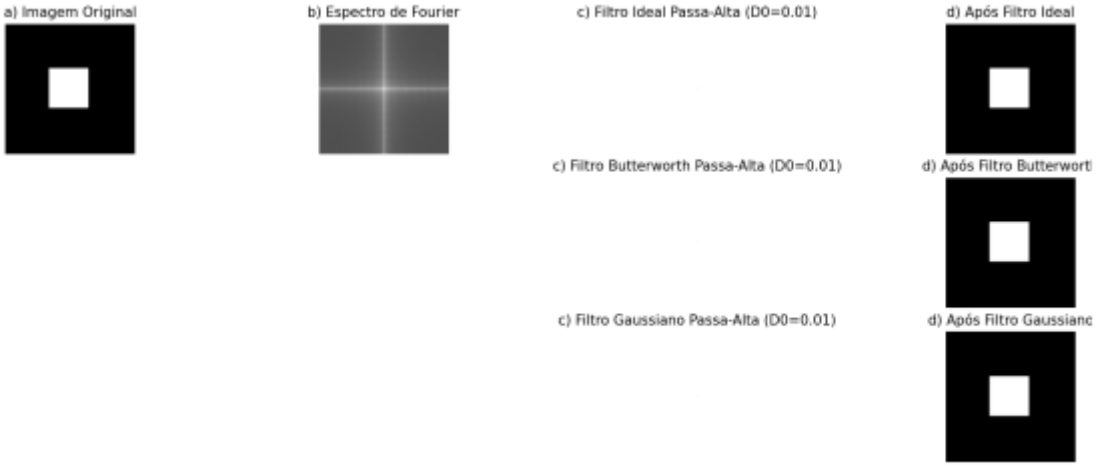
Conforme D0 aumenta, a diferença entre os três tipos de filtros (ideal, Butterworth e Gaussiano) fica evidente na forma como eles atenuam as frequências próximas ao limite de corte:

O filtro ideal corta abruptamente as frequências além do limite de corte, sem suavização.

O filtro Butterworth possui uma transição suave controlada pela ordem do filtro, afetando a nitidez da transição.

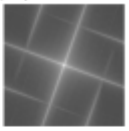
O filtro Gaussiano tem uma transição gradual seguindo uma distribuição gaussiana, tornando a atenuação das frequências próximas ao limite de corte mais suave e gradual.

5. Efetue o mesmo que se pede no item 4, mas use o filtro passa-alta em vez do filtro passa-baixa.



a) Imagem Rotacionada (20º)

b) Espectro de Fourier



c) Filtro Ideal Passa-Alta (D0=0.01)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Alta (D0=0.01)

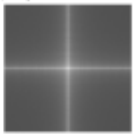
d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Alta (D0=0.01)

d) Após Filtro Gaussiano

a) Imagem Transladada

b) Espectro de Fourier



c) Filtro Ideal Passa-Alta (D0=0.01)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Alta (D0=0.01)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Alta (D0=0.01)

d) Após Filtro Gaussiano

a) Imagem Original

b) Espectro de Fourier

c) Filtro Ideal Passa-Alta (D0=0.05)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Alta (D0=0.05)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Alta (D0=0.05)

d) Após Filtro Gaussiano

a) Imagem Rotacionada (40º)

b) Espectro de Fourier

c) Filtro Ideal Passa-Alta (D0=0.05)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Alta (D0=0.05)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Alta (D0=0.05)

d) Após Filtro Gaussiano

a) Imagem Rotacionada (20º)

b) Espectro de Fourier

c) Filtro Ideal Passa-Alta (D0=0.05)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Alta (D0=0.05)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Alta (D0=0.05)

d) Após Filtro Gaussiano

a) Imagem Rotacionada (40º)

b) Espectro de Fourier

c) Filtro Ideal Passa-Alta (D0=0.01)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Alta (D0=0.01)

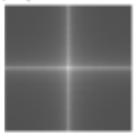
d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Alta (D0=0.01)

d) Após Filtro Gaussiano

a) Imagem Transladada

b) Espectro de Fourier



c) Filtro Ideal Passa-Alta (D0=0.05)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Alta (D0=0.05)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Alta (D0=0.05)

d) Após Filtro Gaussiano

a) Imagem Original

b) Espectro de Fourier

c) Filtro Ideal Passa-Alta (D0=0.5)

d) Após Filtro Ideal

c) Filtro Butterworth Passa-Alta (D0=0.5)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Alta (D0=0.5)

d) Após Filtro Gaussiano

a) Imagem Rotacionada (40º)

b) Espectro de Fourier

c) Filtro Ideal Passa-Alta (D0=0.5)

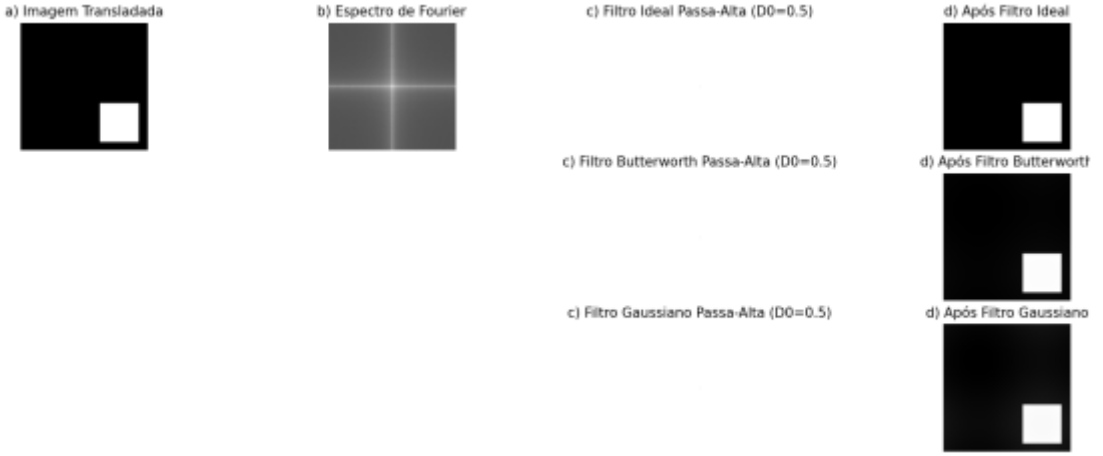
d) Após Filtro Ideal

c) Filtro Butterworth Passa-Alta (D0=0.5)

d) Após Filtro Butterworth

c) Filtro Gaussiano Passa-Alta (D0=0.5)

d) Após Filtro Gaussiano



Código:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

sinc_original_path = './imagem/sinc_original.png'
sinc_original_menor_path =
'./imagem/sinc_original_menor.png'
sinc_rot_path = './imagem/sinc_rot.png'
sinc_rot2_path = './imagem/sinc_rot2.png'
sinc_trans_path = './imagem/sinc_trans.png'

# Ler as imagens
sinc_original = cv2.imread(sinc_original_path,
cv2.IMREAD_GRAYSCALE)
```

```
sinc_original_menor =
cv2.imread(sinc_original_menor_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot = cv2.imread(sinc_rot_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot2 = cv2.imread(sinc_rot2_path,
cv2.IMREAD_GRAYSCALE)
sinc_trans = cv2.imread(sinc_trans_path,
cv2.IMREAD_GRAYSCALE)
images = [sinc_original, sinc_rot, sinc_rot2,
sinc_trans]
titles = ['Original', 'Rotacionada (40º)',
'Rotacionada (20º)','Transladada']

def fourier_spectrum(image):
    # Computa a transformada de Fourier 2D
    f = np.fft.fft2(image)
    # Centraliza as frequências baixas
    43
    fshift = np.fft.fftshift(f)
    # Calcula a magnitude e aplica o logaritmo para
melhor visualização
    magnitude_spectrum = np.log(np.abs(fshift) + 1)
    return magnitude_spectrum

def ideal_highpass_filter(image, cutoff):
    rows, cols = image.shape
    center_x, center_y = rows // 2, cols // 2
    filter = np.ones((rows, cols))
    for i in range(rows):
        for j in range(cols):
            if np.sqrt((i - center_x) ** 2 + (j -
center_y) ** 2) <= cutoff:
                filter[i, j] = 0
```

```
    return filter

def butterworth_highpass_filter(image, cutoff,
order=2):
    rows, cols = image.shape
    center_x, center_y = rows // 2, cols // 2
    filter = np.ones((rows, cols))
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - center_x) ** 2
+ (j - center_y) **2)
            filter[i, j] = 1 / (1 + (cutoff /
distance) ** (2 * order))
    return filter

def gaussian_highpass_filter(image, cutoff):
    rows, cols = image.shape
    center_x, center_y = rows // 2, cols // 2
    filter = np.ones((rows, cols))
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - center_x) ** 2
+ (j - center_y) **2)
            filter[i, j] -= np.exp(-(distance ** 2)
/ (2 * (cutoff **2)))
    return filter

def apply_filter(image, filter):
    # Aqui assumo que você está usando a
Transformada de Fourier para aplicar o filtro.
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    fshift = fshift * filter
    f_ishift = np.fft.ifftshift(fshift)
```

```
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)

    # Normalizando a imagem resultante para o
intervalo [0, 255]
    img_normalized = np.divide(img_back -
np.min(img_back),
    np.max(img_back) - np.min(img_back)) * 255
    return img_normalized

cutoffs = [0.01, 0.05, 0.5]
for cutoff in cutoffs:
    for idx, image in enumerate(images):

# Fourier
        spectrum = fourier_spectrum(image)

# Criação dos filtros
ideal_hp = ideal_highpass_filter(image, cutoff)
butter_hp = butterworth_highpass_filter(image,
cutoff)
gaussian_hp = gaussian_highpass_filter(image,
cutoff)

# Aplicação dos filtros
result_ideal = apply_filter(image, ideal_hp)
result_butter = apply_filter(image, butter_hp)
result_gaussian = apply_filter(image, gaussian_hp)

# Exibição
plt.figure(figsize=(20, 10))

# Imagem original
plt.subplot(4, 4, 1)
```

```
plt.imshow(image, cmap='gray')
plt.title(f'a) Imagem {titles[idx]}')
plt.axis('off')

# Espectro de Fourier
plt.subplot(4, 4, 2)
plt.imshow(spectrum, cmap='gray')
plt.title('b) Espectro de Fourier')
plt.axis('off')

# Filtro passa-alta Ideal
plt.subplot(4, 4, 3)
plt.imshow(ideal_hp, cmap='gray')
plt.title(f'c) Filtro Ideal Passa-Alta
(D0={cutoff})')
plt.axis('off')

# Resultado filtro Ideal
plt.subplot(4, 4, 4)
plt.imshow(result_ideal, cmap='gray')
plt.title('d) Após Filtro Ideal')
plt.axis('off')

# Filtro passa-alta Butterworth
plt.subplot(4, 4, 7)
plt.imshow(butter_hp, cmap='gray')
plt.title(f'c) Filtro Butterworth Passa-Alta
(D0={cutoff})')
plt.axis('off')

# Resultado filtro Butterworth
plt.subplot(4, 4, 8)
plt.imshow(result_butter, cmap='gray')
plt.title('d) Após Filtro Butterworth')
```

```
plt.axis('off')

# Filtro passa-alta Gaussiano
plt.subplot(4, 4, 11)
plt.imshow(gaussian_hp, cmap='gray')
plt.title(f'c) Filtro Gaussiano Passa-Alta
(D0={cutoff})')
plt.axis('off')

# Resultado filtro Gaussiano
plt.subplot(4, 4, 12)
plt.imshow(result_gaussian, cmap='gray')
plt.title('d) Após Filtro Gaussiano')
plt.axis('off')

# Ajusta o layout e mostra a figura
plt.tight_layout()
plt.show()
```

6. Além dos filtros passa-baixa e passa-alta também existe o filtro passa-banda? Explique seu funcionamento e aplique um filtro passa-banda na imagem.

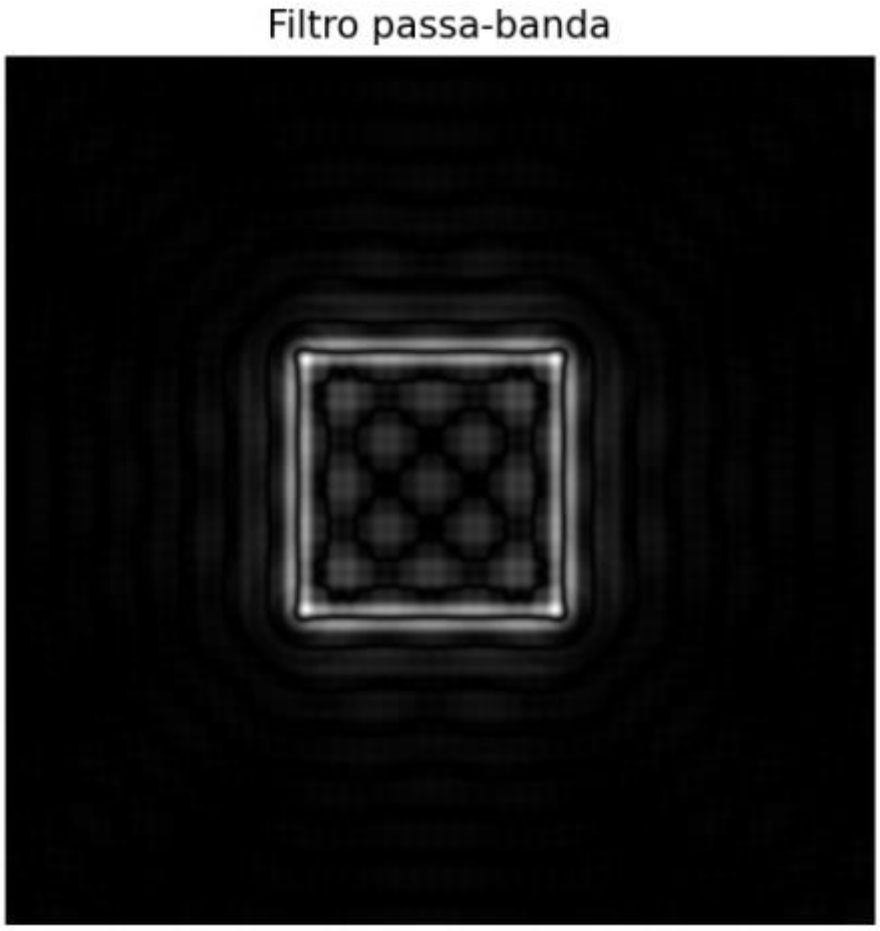
O filtro passa-banda permite que um intervalo específico de frequências passe, enquanto bloqueia todas as outras frequências acima e abaixo desse intervalo. Isso é útil para realçar ou isolar partes específicas de frequência em um sinal ou imagem.

A operação de um filtro passa-banda é explicada da seguinte forma:

Definição das Frequências de Corte: Um filtro passa-banda requer a previsão de duas frequências de corte: uma frequência de corte inferior (geralmente chamada de f1) e uma frequência de corte superior (geralmente chamada de f2). Essas frequências determinam a faixa de frequência que o filtro permite.

Transferência de Frequência: O filtro passa-banda funciona permitindo que todas as frequências dentro da faixa definida (f1 a f2) passem, enquanto

atenua (rejeita) todas as outras frequências fora dessa faixa. A largura e a forma da faixa de frequência dependem da configuração específica do filtro.



Código:

```
import numpy as np
import matplotlib.pyplot as plt

sinc_original_path = './imagem/sinc_original.png'
sinc_original_menor_path =
'./imagem/sinc_original_menor.png'
sinc_rot_path = './imagem/sinc_rot.png'
sinc_rot2_path = './imagem/sinc_rot2.png'
sinc_trans_path = './imagem/sinc_trans.png'

# Ler as imagens
sinc_original = cv2.imread(sinc_original_path,
cv2.IMREAD_GRAYSCALE)
```

```
sinc_original_menor =
cv2.imread(sinc_original_menor_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot = cv2.imread(sinc_rot_path,
cv2.IMREAD_GRAYSCALE)
sinc_rot2 = cv2.imread(sinc_rot2_path,
cv2.IMREAD_GRAYSCALE)
sinc_trans = cv2.imread(sinc_trans_path,
cv2.IMREAD_GRAYSCALE)

images = [sinc_original, sinc_rot, sinc_rot2,
sinc_trans]
titles = ['Original', 'Rotacionada (40º)',
'Rotacionada (20º)',
'Transladada']

def fourier_spectrum(image):
    # Computa a transformada de Fourier 2D
    f = np.fft.fft2(image)
    # Centraliza as frequências baixas
    fshift = np.fft.fftshift(f)
    # Calcula a magnitude e aplica o logaritmo para
melhor visualização
    magnitude_spectrum = np.log(np.abs(fshift) + 1)
    return magnitude_spectrum

def ideal_highpass_filter(image, cutoff):
    rows, cols = image.shape
    center_x, center_y = rows // 2, cols // 2
    filter = np.ones((rows, cols))
    for i in range(rows):
        for j in range(cols):
            if np.sqrt((i - center_x) ** 2 + (j -
center_y) ** 2) <= cutoff:
```

```
                filter[i, j] = 0
    return filter

def butterworth_highpass_filter(image, cutoff,
order=2):
    rows, cols = image.shape
    center_x, center_y = rows // 2, cols // 2
    filter = np.ones((rows, cols))
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - center_x) ** 2
+ (j - center_y) **2)
            filter[i, j] = 1 / (1 + (cutoff /
distance) ** (2 * order))
    return filter

def gaussian_highpass_filter(image, cutoff):
    rows, cols = image.shape
    center_x, center_y = rows // 2, cols // 2
    filter = np.ones((rows, cols))
    for i in range(rows):
        for j in range(cols):
            distance = np.sqrt((i - center_x) ** 2
+ (j - center_y) **2)
            filter[i, j] -= np.exp(-(distance ** 2)
/ (2 * (cutoff **2)))
    return filter

def apply_filter(image, filter):
# Aqui assumo que você está usando a Transformada
de Fourier paraaplicar o filtro.
    f = np.fft.fft2(image)
    fshift = np.fft.fftshift(f)
    fshift = fshift * filter
```

```
    f_ishift = np.fft.ifftshift(fshift)
    img_back = np.fft.ifft2(f_ishift)
    img_back = np.abs(img_back)
    # Normalizando a imagem resultante para o
intervalo [0, 255]
    img_normalized = np.divide(img_back -
np.min(img_back),
    np.max(img_back) - np.min(img_back)) * 255
    return img_normalized

cutoffs = [0.01, 0.05, 0.5]

def ideal_bandpass_filter(image, Dl, Dh):
    rows, cols = image.shape
    center_x, center_y = rows // 2, cols // 2
    filter = np.zeros((rows, cols), dtype=np.uint8)
    for x in range(rows):
        for y in range(cols):
            distance = np.sqrt((x - center_x)**2 +
(y - center_y)**2)
            if Dl <= distance <= Dh:
                filter[x, y] = 1
    return filter

def apply_bandpass_filter(image, Dl, Dh):
    bandpass_filter = ideal_bandpass_filter(image,
Dl, Dh)
    filtered_image = apply_filter(image,
bandpass_filter)
    return filtered_image

# Aplicando o filtro
Dl = 10
Dh = 50
```

```
filtered_image =
apply_bandpass_filter(sinc_original, Dl, Dh)

# Exibindo a imagem resultante
plt.figure(figsize=(10, 5))
plt.imshow(filtered_image, cmap='gray')
plt.title("Filtro passa-banda")
plt.axis('off')
plt.show()
```

Neste código, criamos um filtro passa-banda ideal, onde definimos uma frequência de corte inferior (Dl) e uma frequência de corte superior (Dh). Em seguida, aplicamos esse filtro à imagem chamada 'sinc_original'. Você pode ajustar os valores de Dl e Dh de acordo com suas necessidades específicas para personalizar o filtro.