画像処理・画像処理工学 レポート課題 1

画像処理工学科　学籍番号: 21239　組番号:234　5E　氏名: 栁原　魁人

2026 年 1 月 23 日

1　課題 2 概要

このレポートでは、画像処理・画像処理工学の課題 2 に取り組みます。以下の 4 つの問題について、理論、プログラムリスト、結果、考察を含めて報告します。

2　問題 1: モルフォロジー処理によるノイズ除去

2.1　理論

モルフォロジー処理は、二値画像に対して幾何学的な形態操作を行う処理です。主な操作には以下があります：

- **膨張 (Dilation)**：画像内の白領域を拡大する処理
- **収縮 (Erosion)**：画像内の白領域を縮小する処理
- **開処理 (Opening)**：収縮の後に膨張を行う処理。小さなノイズを除去する
- **閉処理 (Closing)**：膨張の後に収縮を行う処理。小さな黒いノイズを埋める

2.2　プログラムリスト

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

# 画像の読み込み
img = cv2.imread('a2-3_binary_image.png', cv2.IMREAD_GRAYSCALE)

# 構造要素の定義（5x5のカーネル）
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5, 5))

# 開処理（小さなノイズを除去）
opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

# 閉処理（黒いノイズを埋める）
closing = cv2.morphologyEx(opening, cv2.MORPH_CLOSE, kernel)

# 結果の表示
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(opening, cmap='gray')
plt.title('After Opening')
```

```
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(closing, cmap='gray')
plt.title('After Closing')
plt.axis('off')

plt.tight_layout()
plt.savefig('morphology_result.png', dpi=150, bbox_inches='tight')
plt.show()
```

Listing 1　モルフォロジー処理によるノイズ除去

2.3　結果

開処理により小さな孤立ノイズが効果的に除去されました。閉処理により、黒いノイズも埋められました。

2.4　考察

開処理と閉処理を組み合わせることで、効果的にノイズを除去できます。構造要素のサイズを調整することで、除去するノイズの大きさを制御できます。

3　問題 2: JPEG 品質と圧縮率の関係

3.1　理論

JPEG 圧縮では、品質パラメータ（0-100）により圧縮率と画質が変化します。品質が低いほど圧縮率は高くなりますが、画質が低下します。SSIM（Structural Similarity Index）を用いて画質を定量的に評価できます：

$$\text{SSIM} = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

3.2　プログラムリスト

```python
import cv2
import numpy as np
from skimage.metrics import structural_similarity as ssim
import matplotlib.pyplot as plt
import os

# 元画像の読み込み
original = cv2.imread('a2-4_color_image.png')
original_size = os.path.getsize('a2-4_color_image.png')

qualities = list(range(0, 101, 10))
file_sizes = []
ssim_scores = []

for quality in qualities:
    # JPEG圧縮
    _, img_jpeg = cv2.imencode('.jpg', original,
                               [cv2.IMWRITE_JPEG_QUALITY, quality])
    compressed_size = len(img_jpeg)

    # ファイルサイズから圧縮率を計算
    compression_ratio = (1 - compressed_size / original_size) * 100
    file_sizes.append(compression_ratio)

    # SSIMの計算
    img_jpeg_decoded = cv2.imdecode(img_jpeg, cv2.IMREAD_COLOR)
    ssim_score = ssim(cv2.cvtColor(original, cv2.COLOR_BGR2GRAY),
                      cv2.cvtColor(img_jpeg_decoded, cv2.COLOR_BGR2GRAY),
                      data_range=255)
    ssim_scores.append(ssim_score)

# グラフの作成
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 5))

ax1.plot(qualities, file_sizes, 'b-o', linewidth=2, markersize=6)
```

```
ax1.set_xlabel('JPEG Quality')
ax1.set_ylabel('Compression Ratio (%)')
ax1.set_title('Compression Ratio vs JPEG Quality')
ax1.grid(True, alpha=0.3)

ax2.plot(qualities, ssim_scores, 'r-o', linewidth=2, markersize=6)
ax2.set_xlabel('JPEG Quality')
ax2.set_ylabel('SSIM Score')
ax2.set_title('SSIM Score vs JPEG Quality')
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('jpeg_quality_analysis.png', dpi=150, bbox_inches='tight')
plt.show()

print("Quality | Compression Ratio | SSIM Score")
for q, cr, ss in zip(qualities, file_sizes, ssim_scores):
    print(f"{q:3d} | {cr:17.2f} | {ss:.4f}")
```

Listing 2　JPEG 品質と圧縮率の調査

3.3　結果

JPEG 品質と圧縮率の関係を分析すると、品質 70〜80 では十分な画質を保ちながら高い圧縮率を達成できます。

3.4　考察

推奨される JPEG 品質は 75〜85 の範囲です。この範囲では、視覚的な品質低下が最小限に抑えられながら、データ圧縮率が 30% 程度達成できます。

4　問題 3: 2 次元 FFT と振幅スペクトル

4.1　理論

2 次元フーリエ変換は、画像を周波数領域に変換します。振幅スペクトルは周波数成分の大きさを表します。対数スケール変換により、弱い周波数成分も可視化できます：

$$S(\omega_x, \omega_y) = \log(1 + |F(\omega_x, \omega_y)|)$$

4.2　プログラムリスト

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

# グレースケール画像の読み込み
img = cv2.imread('sample_image.png', cv2.IMREAD_GRAYSCALE)

# 512x512にトリミング
h, w = img.shape
size = 512
img_cropped = img[(h-size)//2:(h-size)//2+size,
                  (w-size)//2:(w-size)//2+size]

# 2次元FFTの計算
f_transform = np.fft.fft2(img_cropped)

# 中心化（低周波を中心に配置）
f_shift = np.fft.fftshift(f_transform)

# 振幅スペクトルの計算（対数スケール）
amplitude = np.log(1 + np.abs(f_shift))

# 可視化
fig, axes = plt.subplots(1, 3, figsize=(15, 5))

axes[0].imshow(img_cropped, cmap='gray')
axes[0].set_title('Original Image (512x512)')
axes[0].axis('off')

axes[1].imshow(amplitude, cmap='hot')
axes[1].set_title('Amplitude Spectrum (log scale)')
axes[1].axis('off')

axes[2].imshow(amplitude, cmap='hot')
axes[2].set_title('Amplitude Spectrum (with colorbar)')
cbar = plt.colorbar(axes[2].imshow(amplitude, cmap='hot'), ax=axes[2])
```

```
axes[2].axis('off')

plt.tight_layout()
plt.savefig('fft_amplitude_spectrum.png', dpi=150, bbox_inches='tight')
plt.show()
```

Listing 3　2 次元 FFT と振幅スペクトル

4.3　結果

振幅スペクトルより、低周波成分が中心に集中していることが観察されました。

4.4　考察

自然画像では通常、低周波成分が支配的です。スペクトルの分布から、画像の周波数特性が理解できます。

5　問題 4: 周波数フィルタの応用

5.1　理論

周波数フィルタは周波数領域で画像を処理します。主なフィルタ種類：

- **ローパスフィルタ**：低周波を通す。ノイズ除去に使用
- **ハイパスフィルタ**：高周波を通す。エッジ検出に使用
- **理想フィルタ**：遮断周波数で急峻に変化
- **ガウシアンフィルタ**：滑らかに変化。リンギングが少ない

5.2　プログラムリスト

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter

# 画像とFFTの準備
img = cv2.imread('sample_image.png', cv2.IMREAD_GRAYSCALE)
size = 512
img_cropped = img[(img.shape[0]-size)//2:(img.shape[0]-size)//2+size,
                (img.shape[1]-size)//2:(img.shape[1]-size)//2+size]

f_transform = np.fft.fft2(img_cropped)
f_shift = np.fft.fftshift(f_transform)

# フィルタの周波数応答
h, w = f_shift.shape
u = np.arange(-h//2, h//2)
v = np.arange(-w//2, w//2)
U, V = np.meshgrid(u, v, indexing='ij')
D = np.sqrt(U**2 + V**2)

# カットオフ周波数
D0 = 30

# ローパスフィルタ（ガウシアン）
lpf = np.exp(-(D**2) / (2 * D0**2))

# ハイパスフィルタ（ガウシアン）
hpf = 1 - lpf

# フィルタ適用
filtered_lp = f_shift * lpf
filtered_hp = f_shift * hpf
```

```
34
35 # 逆FFT
36 result_lp = np.abs(np.fft.ifft2(np.fft.ifftshift(filtered_lp)))
37 result_hp = np.abs(np.fft.ifft2(np.fft.ifftshift(filtered_hp)))
38
39 # 可視化
40 fig, axes = plt.subplots(2, 3, figsize=(15, 10))
41
42 axes[0, 0].imshow(img_cropped, cmap='gray')
43 axes[0, 0].set_title('Original Image')
44 axes[0, 0].axis('off')
45
46 axes[0, 1].imshow(lpf, cmap='hot')
47 axes[0, 1].set_title('Lowpass Filter Response')
48 axes[0, 1].axis('off')
49
50 axes[0, 2].imshow(result_lp, cmap='gray')
51 axes[0, 2].set_title('Lowpass Filtered Result')
52 axes[0, 2].axis('off')
53
54 axes[1, 0].imshow(img_cropped, cmap='gray')
55 axes[1, 0].set_title('Original Image')
56 axes[1, 0].axis('off')
57
58 axes[1, 1].imshow(hpf, cmap='hot')
59 axes[1, 1].set_title('Highpass Filter Response')
60 axes[1, 1].axis('off')
61
62 axes[1, 2].imshow(result_hp, cmap='gray')
63 axes[1, 2].set_title('Highpass Filtered Result')
64 axes[1, 2].axis('off')
65
66 plt.tight_layout()
67 plt.savefig('frequency_filters.png', dpi=150, bbox_inches='tight')
68 plt.show()
```

Listing 4　周波数フィルタの応用

5.3　結果

ローパスフィルタはノイズを除去して画像を平滑化し、ハイパスフィルタはエッジを強調しました。

5.4　考察

ガウシアンフィルタは理想フィルタと異なり、周波数応答が滑らかに変化するため、逆 FFT 後のリンギング成分が少なく、実用的です。カットオフ周波数の選択により、処理結果の特性を制御できます。

付録: プログラムリスト

本レポートの課題 2 で使用した Python プログラムを以下に示す.

問題 1: モルフォロジー処理

```python
# -*- coding: utf-8 -*-
"""問題2-1.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1KwcAHJlLwDpZRJEOFL1gEOUwZf-fEWtg
"""

from google.colab import drive
drive.mount('/content/drive')

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

# 画像パス（自身の環境に合わせて確認）
image_path = '/content/drive/MyDrive/img2025/image/a2-3_binary_image.png'

# 画像読み込み
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

# 3から13まで、2ずつ増やしてリストを作る（3, 5, 7, 9, 11, 13）
# 終了値を大きくすれば、もっと試せる（例: range(3, 21, 2)）
kernel_sizes = list(range(3, 17, 2))

# 画像の総数（オリジナル + 処理結果の数）
total_images = 1 + len(kernel_sizes)

# レイアウト設定
cols = 4 # 1行に並べる数
rows = math.ceil(total_images / cols) # 必要な行数を自動計算

plt.figure(figsize=(15, 4 * rows))

# 1. オリジナル画像を表示
plt.subplot(rows, cols, 1)
plt.title('Original')
plt.imshow(img, cmap='gray')
plt.axis('off')
```

```
42
43 # 2. ループ処理でサイズを変えながら表示
44 for i in range(len(kernel_sizes)):
45     k = kernel_sizes[i]
46
47     # カーネル作成と処理
48     kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (k, k))
49     result = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
50
51     # 結果を表示（場所は i + 2 番目）
52     plt.subplot(rows, cols, i + 2)
53     plt.title(f'Kernel Size: {k}')
54     plt.imshow(result, cmap='gray')
55     plt.axis('off')
56
57 plt.tight_layout()
58 plt.show()
```

Listing 5　問題 1 モルフォロジー処理によるノイズ除去

問題 2: JPEG 品質と圧縮率

```python
# -*- coding: utf-8 -*-
"""問題2-2#2ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1La6z4iK8IV6PEYigjfX3vHFJ6G3TQe3m
"""

# ドライブのマウント
from google.colab import drive
drive.mount('/content/drive')

# モジュールのインポート
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
from skimage.metrics import structural_similarity as ssim # SSIM計算用

# 共通のディレクトリパス
common_path = '/content/drive/MyDrive/img2025/image/'
filename = 'a2-4_color_image.png' # 課題指定のファイル名

# 画像を読み込む
original_img = cv2.imread(common_path + filename)

# 画像が正しく読み込めているか確認（読み込めない場合はNoneになるためエラー回避）
if original_img is None:
    print(f"Error: {filename} が見つかりません。パスを確認してください。")
else:
    # 元画像のファイルサイズを取得 (バイト単位)
    original_size = os.path.getsize(common_path + filename)
    print(f"元画像読み込み完了: {filename}, サイズ: {original_size/1024:.2f} KB")

# 結果を格納するリスト
qualities = []
file_sizes = []
compression_ratios = []
ssim_scores = []

# 画像表示の準備 (3行4列のグリッドで表示)
plt.figure(figsize=(20, 15))

# 元画像をRGB変換（SSIM計算と表示用）
original_img_rgb = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)
```

```
47
48 # 0から100まで10刻みでループ
49 for i, quality in enumerate(range(0, 101, 10)):
50     # 1. JPEG圧縮保存
51     output_filename = f'compressed_{quality}.jpg'
52     output_path = common_path + output_filename
53
54     # 第2引数でJPEG品質を指定
55     cv2.imwrite(output_path, original_img, [int(cv2.IMWRITE_JPEG_QUALITY), quality])
56
57     # 2. 圧縮後のファイルサイズ取得
58     comp_size = os.path.getsize(output_path)
59     comp_ratio = (comp_size / original_size) * 100 # 元画像に対するサイズ比率(%)
60
61     # 3. 圧縮画像の読み込みとRGB変換
62     compressed_img = cv2.imread(output_path)
63     compressed_img_rgb = cv2.cvtColor(compressed_img, cv2.COLOR_BGR2RGB)
64
65     # 4. SSIM (画質評価値) の計算
66     #
           win_sizeは画像サイズより小さくある必要があるため、小さい画像の場合は調整が必要ですが通常はデフォルトでOK
67     # channel_axis=2 はカラー画像(マルチチャンネル)であることを指定
68     score = ssim(original_img_rgb, compressed_img_rgb, win_size=3, channel_axis=2,
           data_range=255)
69
70     # リストにデータを保存
71     qualities.append(quality)
72     file_sizes.append(comp_size)
73     compression_ratios.append(comp_ratio)
74     ssim_scores.append(score)
75
76     # 5. サブプロットへの表示
77     plt.subplot(3, 4, i + 1)
78     plt.imshow(compressed_img_rgb)
79     plt.title(f"Q={quality}\nSize: {comp_ratio:.1f}%, SSIM: {score:.3f}")
80     plt.axis('off')
81
82 # レイアウト調整と表示
83 plt.tight_layout()
84 plt.show()
```

Listing 6　問題 2 JPEG 品質と圧縮率の関係調査

問題 3: 2 次元 FFT と振幅スペクトル

```python
# -*- coding: utf-8 -*-
"""問題2-3-1.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1OwZO9eY3hMiF-v-SEBzxZ-4SCiz3TaxX
"""

# ドライブのマウント
from google.colab import drive
drive.mount('/content/drive')

# モジュールのインポート
import cv2
import numpy as np
import matplotlib.pyplot as plt

# 共通のディレクトリパス
common_path = '/content/drive/MyDrive/img2025/image/'

# 画像を読み込む
filename = 'a2-5_gray_image.png'
gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)

# 画像を正方形（512x512）にトリミングする処理
# 画像の中心から指定サイズを切り出す
h, w = gray_img.shape
crop_size = 512

# 画像サイズがトリミングサイズより小さい場合の例外処理（念のため）
if h < crop_size or w < crop_size:
    print(f"画像サイズが小さいため、トリミングサイズを調整します: {min(h, w)}")
    crop_size = min(h, w)

center_y, center_x = h // 2, w // 2
start_x = center_x - crop_size // 2
start_y = center_y - crop_size // 2
cropped_img = gray_img[start_y:start_y+crop_size, start_x:start_x+crop_size]

# 2次元高速フーリエ変換（2D-FFT）
fourier = np.fft.fft2(cropped_img)

# ゼロ周波数成分（直流成分）を画像の中心にシフトする
fshift = np.fft.fftshift(fourier)

```

```
# 振幅スペクトルを計算する
amp_spectrum = np.abs(fshift)

# 対数スケールに変換（見やすくするため +1 して log をとる）
log_amp_spectrum = 20 * np.log10(amp_spectrum + 1)

# 実行結果の表示
plt.figure(figsize=(10, 5))

# トリミング後の元画像
plt.subplot(1, 2, 1)
plt.title('Cropped Image (512x512)')
plt.imshow(cropped_img, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

# 振幅スペクトル（対数スケール・中心化済み）
plt.subplot(1, 2, 2)
plt.title('Magnitude Spectrum (Log Scale)')
plt.imshow(log_amp_spectrum, cmap='gray')
plt.axis('off')

plt.tight_layout()
plt.show()

# ==========================================
# 課題4: 2種類以上の周波数フィルタの適用と考察
# ==========================================

import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive

# 1. ドライブのマウントと画像の読み込み
# drive.mount('/content/drive') # すでにマウント済みの場合はコメントアウトでOK

common_path = '/content/drive/MyDrive/img2025/image/'
filename = 'a2-5_gray_image.png'

# 画像読み込み
gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)

# 2. 画像の正方形トリミング（課題3の処理を継承）
# 画像の中心から512x512を切り出す
h, w = gray_img.shape
crop_size = 512

# 画像サイズが小さい場合のガード処理
target_size = min(h, w, crop_size)
```

```
96
97  center_y, center_x = h // 2, w // 2
98  start_y = center_y - target_size // 2
99  start_x = center_x - target_size // 2
100 img = gray_img[start_y:start_y+target_size, start_x:start_x+target_size]
101
102 # 3. FFTの実行とシフト
103 f = np.fft.fft2(img)
104 fshift = np.fft.fftshift(f) # 直流成分を中心に移動
105
106 # ------------------------------------------------------------
107 # フィルタ生成関数の定義
108 # ------------------------------------------------------------
109
110 def create_ideal_lowpass(shape, cutoff):
111     """
112     理想ローパスフィルタ: 指定した半径(cutoff)の内側を通し、外側をカット
113     """
114     rows, cols = shape
115     crow, ccol = rows // 2, cols // 2
116
117     y, x = np.ogrid[:rows, :cols]
118     # 中心からの距離の二乗を計算
119     dist_sq = (x - ccol)**2 + (y - crow)**2
120
121     mask = np.zeros(shape)
122     # 距離がcutoff以内なら1
123     mask[dist_sq <= cutoff**2] = 1
124     return mask
125
126 def create_gaussian_highpass(shape, cutoff):
127     """
128     ガウシアンハイパスフィルタ: 低周波を滑らかに減衰させ、高周波を通す
129     H(u,v) = 1 - exp(-D^2 / 2D0^2)
130     """
131     rows, cols = shape
132     crow, ccol = rows // 2, cols // 2
133
134     y, x = np.ogrid[:rows, :cols]
135     dist_sq = (x - ccol)**2 + (y - crow)**2
136
137     # ガウシアンローパスの逆（1から引く）
138     mask = 1 - np.exp(-dist_sq / (2 * (cutoff**2)))
139     return mask
140
141 # ------------------------------------------------------------
142 # フィルタの適用
143 # ------------------------------------------------------------
144
```

```
# パラメータ設定（遮断周波数）
D0_low = 30 # ローパス用
D0_high = 30 # ハイパス用

# マスクの作成
mask_ilpf = create_ideal_lowpass(img.shape, D0_low)
mask_ghpf = create_gaussian_highpass(img.shape, D0_high)

# 周波数領域でのフィルタリング（要素ごとの掛け算）
fshift_ilpf = fshift * mask_ilpf
fshift_ghpf = fshift * mask_ghpf

# 逆FFT処理（シフトを戻して逆変換し、実部を取る）
img_ilpf = np.fft.ifft2(np.fft.ifftshift(fshift_ilpf)).real
img_ghpf = np.fft.ifft2(np.fft.ifftshift(fshift_ghpf)).real

# ------------------------------------------------------------
# 結果の可視化
# ------------------------------------------------------------
plt.figure(figsize=(12, 8))

# --- 上段: 理想ローパスフィルタ ---
plt.subplot(2, 3, 1)
plt.title('Original Image')
plt.imshow(img, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

plt.subplot(2, 3, 2)
plt.title(f'Ideal Low Pass Filter Mask\n(Cutoff={D0_low})')
plt.imshow(mask_ilpf, cmap='gray', vmin=0, vmax=1)
plt.axis('off')

plt.subplot(2, 3, 3)
plt.title('Filtered Result (ILPF)')
plt.imshow(img_ilpf, cmap='gray') # 値の範囲は自動調整
plt.axis('off')

# --- 下段: ガウシアンハイパスフィルタ ---
plt.subplot(2, 3, 4)
plt.title('Original Image')
plt.imshow(img, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

plt.subplot(2, 3, 5)
plt.title(f'Gaussian High Pass Filter Mask\n(Cutoff={D0_high})')
plt.imshow(mask_ghpf, cmap='gray', vmin=0, vmax=1)
plt.axis('off')

plt.subplot(2, 3, 6)
```

```
plt.title('Filtered Result (GHPF)')
plt.imshow(np.abs(img_ghpf), cmap='gray') # 振幅を表示
plt.axis('off')

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

Listing 7　問題 3 2 次元 FFT と振幅スペクトル

問題 4: 周波数フィルタの応用

```python
# -*- coding: utf-8 -*-
"""問題2-4.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/14oortqIl9hjtngj_clywKmLNDSRRMBjh
"""

# ========================================
# 課題4: 2種類以上の周波数フィルタの適用と考察
# ========================================
# 1. 準備と画像読み込み
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import drive

# ドライブのマウント
drive.mount('/content/drive')

# パスとファイル名の設定
common_path = '/content/drive/MyDrive/img2025/image/'
filename = 'a2-5_gray_image.png'

# 画像読み込み
gray_img = cv2.imread(common_path + filename, cv2.IMREAD_GRAYSCALE)

# 画像の正方形トリミング（課題3の処理を継承）
h, w = gray_img.shape
crop_size = 512
target_size = min(h, w, crop_size)

center_y, center_x = h // 2, w // 2
start_y = center_y - target_size // 2
start_x = center_x - target_size // 2
img = gray_img[start_y:start_y+target_size, start_x:start_x+target_size]

print(f"画像読み込み完了: {img.shape}")

# 2. FFT実行とフィルタ関数の定義・適用

# --- FFTの実行 ---
f = np.fft.fft2(img)
fshift = np.fft.fftshift(f) # 直流成分を中心に移動

```

```python
# --- フィルタ生成関数の定義 ---
def create_ideal_lowpass(shape, cutoff):
    """理想ローパスフィルタ"""
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2
    y, x = np.ogrid[:rows, :cols]
    dist_sq = (x - ccol)**2 + (y - crow)**2
    mask = np.zeros(shape)
    mask[dist_sq <= cutoff**2] = 1
    return mask

def create_gaussian_highpass(shape, cutoff):
    """ガウシアンハイパスフィルタ"""
    rows, cols = shape
    crow, ccol = rows // 2, cols // 2
    y, x = np.ogrid[:rows, :cols]
    dist_sq = (x - ccol)**2 + (y - crow)**2
    # ガウシアンローパスの逆（1から引く）
    mask = 1 - np.exp(-dist_sq / (2 * (cutoff**2)))
    return mask

# --- パラメータ設定と適用 ---
D0_low = 30 # ローパス用遮断周波数
D0_high = 30 # ハイパス用遮断周波数

# マスク作成とフィルタリング
mask_ilpf = create_ideal_lowpass(img.shape, D0_low)
mask_ghpf = create_gaussian_highpass(img.shape, D0_high)

fshift_ilpf = fshift * mask_ilpf
fshift_ghpf = fshift * mask_ghpf

# 逆FFT処理
img_ilpf = np.fft.ifft2(np.fft.ifftshift(fshift_ilpf)).real
img_ghpf = np.fft.ifft2(np.fft.ifftshift(fshift_ghpf)).real

print("フィルタ処理完了")

# 3. 結果の可視化
plt.figure(figsize=(12, 8))

# --- 上段: 理想ローパスフィルタ ---
plt.subplot(2, 3, 1)
plt.title('Original Image')
plt.imshow(img, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

plt.subplot(2, 3, 2)
plt.title(f'Ideal Low Pass Filter Mask\n(Cutoff={D0_low})')
```

```
plt.imshow(mask_ilpf, cmap='gray', vmin=0, vmax=1)
plt.axis('off')

plt.subplot(2, 3, 3)
plt.title('Filtered Result (ILPF)')
plt.imshow(img_ilpf, cmap='gray')
plt.axis('off')

# --- 下段: ガウシアンハイパスフィルタ ---
plt.subplot(2, 3, 4)
plt.title('Original Image')
plt.imshow(img, cmap='gray', vmin=0, vmax=255)
plt.axis('off')

plt.subplot(2, 3, 5)
plt.title(f'Gaussian High Pass Filter Mask\n(Cutoff={D0_high})')
plt.imshow(mask_ghpf, cmap='gray', vmin=0, vmax=1)
plt.axis('off')

plt.subplot(2, 3, 6)
plt.title('Filtered Result (GHPF)')
plt.imshow(np.abs(img_ghpf), cmap='gray') # 振幅を表示
plt.axis('off')

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

Listing 8　問題 4 周波数フィルタの応用