



IMAGE RESTORATION & RECONSTRUCTION

Group 3

Edgrant H. S. (2206025016)

Giovan C. S. (2206816084)

Nicholas S. (2206059396)



01 Introduction

Types of degradation

Degradasi adalah perubahan yang terjadi pada gambar yang menyebabkan hilangnya informasi atau kualitas gambar. Beberapa jenis degradasi yang umum terjadi adalah:

- **Noise:** Variasi acak dalam luminansi atau warna piksel
- **Blur:** Kehilangan detail dalam gambar yang biasanya disebabkan oleh pergerakan atau ketidakakuratan fokus
- **Artifact:** Distorsi yang merepresentasi gambar yang tidak ada dalam scene asli



Sources of Degradation

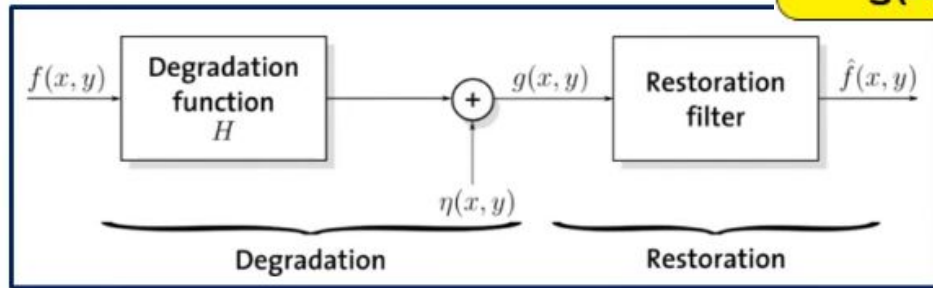
Sumber degradasi dapat dibagi menjadi tiga kategori utama:

- **Acquisition:** Degradasi yang terjadi saat mengambil gambar (misalnya, pergerakan kamera menghasilkan blur atau terjadi aliasing)
- **Transmission:** Degradasi yang terjadi saat mentransmisikan gambar (misalnya, bit flip atau kehilangan paket)
- **Storage:** Degradasi yang terjadi saat menyimpan gambar (misalnya, kompresi JPEG)



Model image Degradation and Restoration

Linear blurring and additive noise
 $g(x, y) = f(x, y) * h(x, y) + \eta(x, y)$



Pada sistem di atas dapat dilihat gambar awal $f(x, y)$ yang mengalami degradasi H dan noise $n(x, y)$ sehingga menghasilkan gambar yang terlihat $g(x, y)$.

Tujuan dari proses restorasi adalah mengembalikan gambar yang terlihat $g(x, y)$ menjadi gambar asli $f(x, y)$. Namun, karena tidak selalu mungkin untuk mengembalikan gambar asli, tujuan dari proses restorasi adalah menghasilkan gambar yang terlihat $\hat{f}(x, y)$ yang mendekati gambar asli $f(x, y)$.

Sifat Degradation Function (H)

- **Linear:** Output dari 2 input yang dijumlahkan sama dengan output dari 2 input yang diolah secara terpisah dan dijumlahkan.
- **Position-invariant:** Output dari input yang digeser sama dengan output dari input yang tidak digeser.
- **Homogeneous:** Output dari input yang dikalikan dengan konstanta sama dengan output dari input yang tidak dikalikan dengan konstanta.

$$H(a_1 f_1(x, y) + a_2 f_2(x, y)) = a_1 H(f_1(x, y)) + a_2 H(f_2(x, y))$$

$$H(f(x - x_0, y - y_0)) = H(f(x, y))$$

$$H(a f(x, y)) = a H(f(x, y))$$

Fourier Transform of Degradation

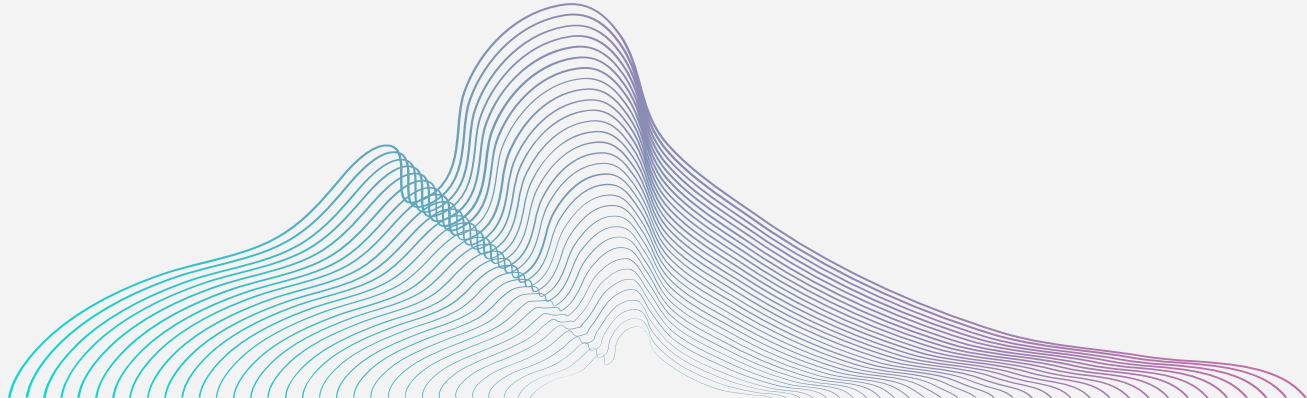
Dalam domain waktu persamaan degradasi adalah

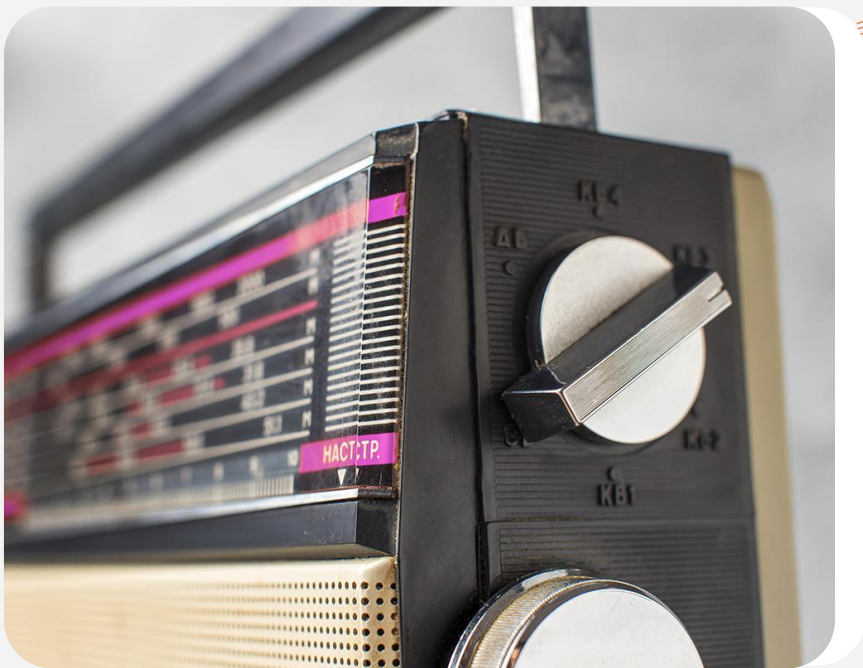
$$g(x, y) = h(x, y) * f(x, y) + n(x, y)$$

sedangkan dalam domain frekuensi persamaan degradasi adalah

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

Dalam domain frekuensi, degradasi dapat dihitung dengan mengalikan spektrum gambar asli dengan fungsi transfer degradasi. Jika ingin menghitung gambar asli, kita dapat menggunakan persamaan invers Fourier transform.



A decorative graphic consisting of several concentric, wavy lines in a light pink color, positioned to the right of the radio image.

02 Noise Models

Categories of Noise

- **Distribution-based noise:** Noise yang dihasilkan oleh distribusi acak (misalnya, Gaussian, Poisson, Salt-and-pepper)
- **Correlation-based noise:** Noise yang dihasilkan oleh korelasi antar piksel (misalnya, speckle noise, block noise)
- **Nature-based noise:** Noise yang dihasilkan oleh sifat alam (misalnya, film grain, film scratches)
- **Source-based noise:** Noise yang dihasilkan oleh sumber (misalnya, sensor noise, quantization noise)



Distribution-based Noise

Distribution-based noise adalah noise yang dihasilkan oleh distribusi acak. Persamaannya menghasilkan probabilitas noise yang dihasilkan. Beberapa jenis distribution-based noise adalah:

- **Gaussian noise:** Noise yang dihasilkan oleh distribusi Gaussian
- **Poisson noise:** Noise yang dihasilkan oleh distribusi Poisson, sering ditemukan pada citra X-ray atau citra mikroskop
- **Exponential noise:** Noise yang dihasilkan oleh distribusi exponential. Biasa digunakan untuk menggambarkan noise pada citra yang dihasilkan oleh sensor CCD.
- **Salt-and-pepper noise:** Noise yang dihasilkan oleh distribusi salt-and-pepper, disebut juga impulse noise atau binary noise. Biasa disebabkan oleh kerusakan pada sensor.

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

$$p(z) = \frac{(np)^z}{z!} e^{-np}$$

$$p(z) = \lambda e^{-\lambda z}$$

$$p(z) = \begin{cases} P_s & \text{if } z = 0 \\ P_p & \text{if } z = 1 \\ 1 - P_s - P_p & \text{if } z = 0 \end{cases}$$

Distribution-based Noise

Daftar simbol:

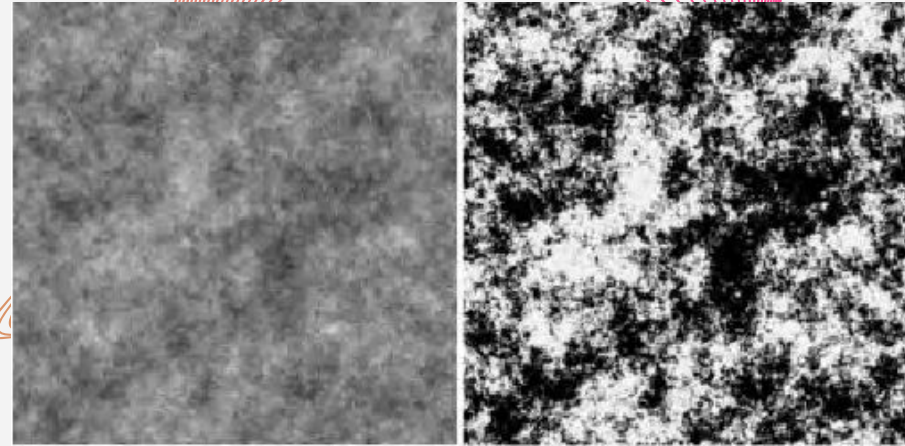
- $p(z)$ adalah probabilitas (pdf) noise yang dihasilkan
- z adalah nilai noise
- μ adalah rata-rata noise
- σ adalah standar deviasi noise
- n adalah jumlah pixel
- p ratio pixel noise dibandingkan dengan total pixel

Correlation-based Noise

Noise yang nilainya **bergantung pada nilai lainnya**. Uncorrelated noise adalah noise yang pixelnya tidak bergantung pada pixel lainnya seperti white noise.

Power spectral density (PSD) adalah ukuran kekuatan noise pada frekuensi tertentu. PSD noise yang tinggi pada frekuensi rendah biasanya disebut sebagai low-frequency noise, sedangkan PSD noise yang tinggi pada frekuensi tinggi biasanya disebut sebagai high-frequency noise.

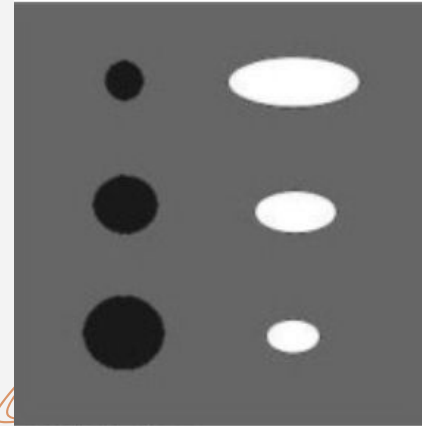
White noise adalah noise yang memiliki PSD konstan pada semua frekuensi. Sedangkan, **pink noise** adalah noise yang memiliki PSD yang menurun seiring dengan peningkatan frekuensi $1/f$. Ini disebut juga flicker noise.



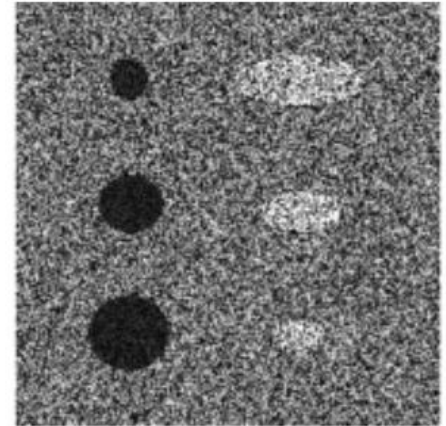
Nature-based Noise

Additive noise: Noise yang ditambahkan ke gambar asli

Multiplicative noise: Noise yang dikalikan dengan gambar asli. Contohnya speckle noise yaitu Average amplitude meningkat seiring dengan peningkatan intensitas gambar.



a) Noiseless synthetic image

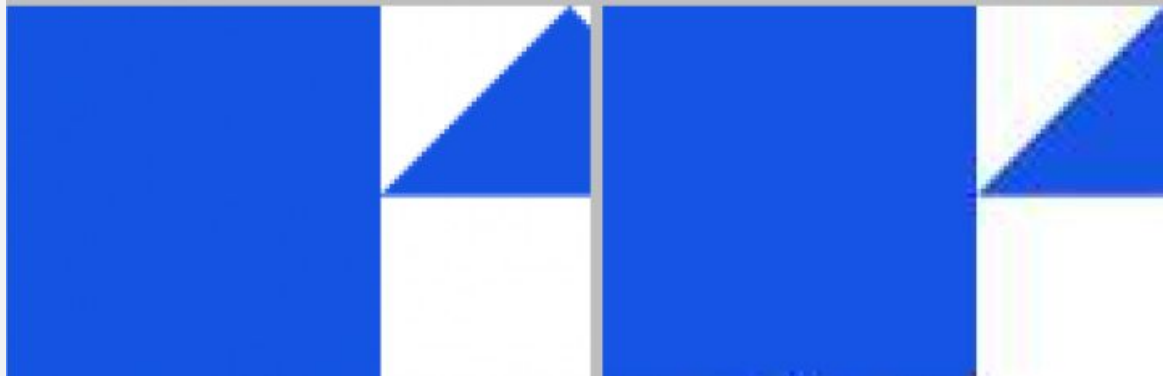



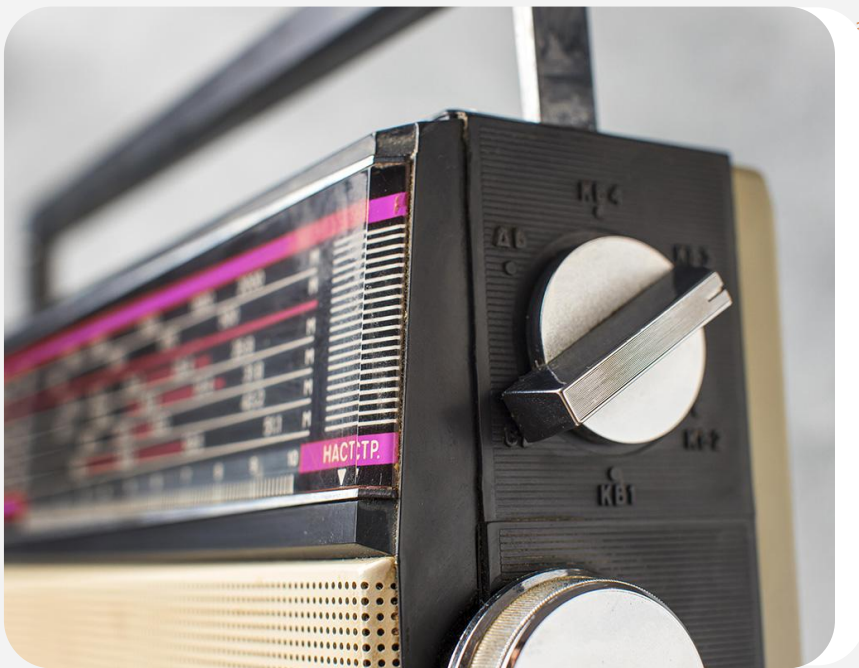
b) Noisy image contaminated with variance 1 speckle noise

Source-based Noise

Quantization noise: Noise yang dihasilkan oleh proses quantization, biasanya terjadi pada proses kompresi gambar

Photon noise: Noise yang dihasilkan oleh proses pengambilan gambar akibat jumlah foton yang masuk ke sensor

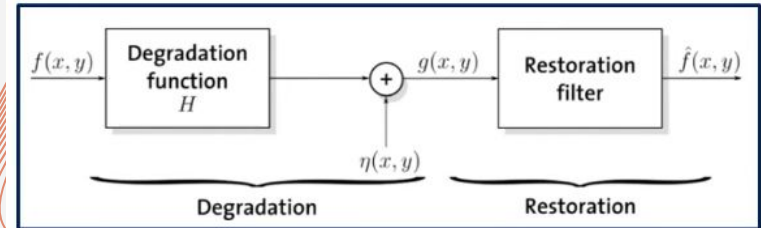


A decorative graphic consisting of several concentric, wavy lines in shades of pink and orange, positioned to the left of the section header.

03 Estimation of Degradation Function

Metode Estimasi H

- **Estimation by Observation:** Mengamati gambar yang terdegradasi dan mengestimasi fungsi degradasi
- **Estimation by Experiment:** Mengambil gambar dengan kondisi tertentu dan mengestimasi fungsi degradasi
- **Estimation by Modeling:** Membuat model matematis dari proses degradasi dan mengestimasi fungsi degradasi



Estimation by Observation

Mencari informasi dari gambar yang terdegradasi untuk mengestimasi fungsi degradasi. Biasa dilakukan pada sub-bagian gambar yang memiliki noise yang tinggi.

$$H(u, v) = \frac{G(u, v)}{\hat{F}(u, v)}$$

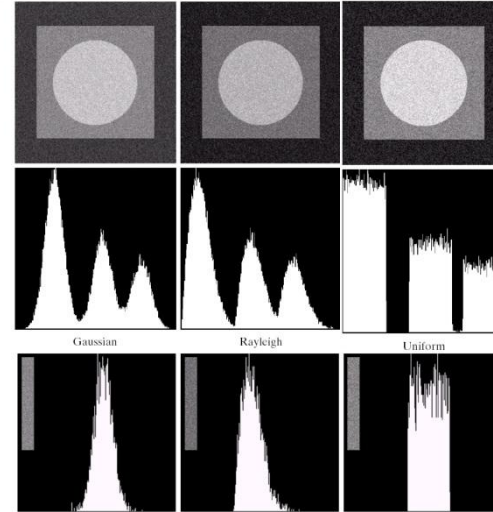
- $H(u, v)$ adalah fungsi transfer degradasi
- $G(u, v)$ adalah spektrum gambar terdegradasi
- $\hat{F}(u, v)$ adalah spektrum gambar asli

Estimation by Experiment

Menggunakan device yang menghasilkan gambar dengan kondisi tertentu untuk mengestimasi fungsi degradasi. Biasa dilakukan pada gambar yang dihasilkan oleh sensor. Atau menggunakan kamera dengan kondisi tertentu untuk menghasilkan gambar dengan noise yang tinggi.

Bisa dilakukan dengan mengambil gambar putih dan hitam, kemudian menganalisis noise yang dihasilkan untuk mengestimasi fungsi degradasi.

Estimation of Noise



We cannot use the image histogram to estimate noise PDF.

It is better to use the histogram of one area of an image that has constant intensity to estimate noise PDF.

Estimation by Modeling

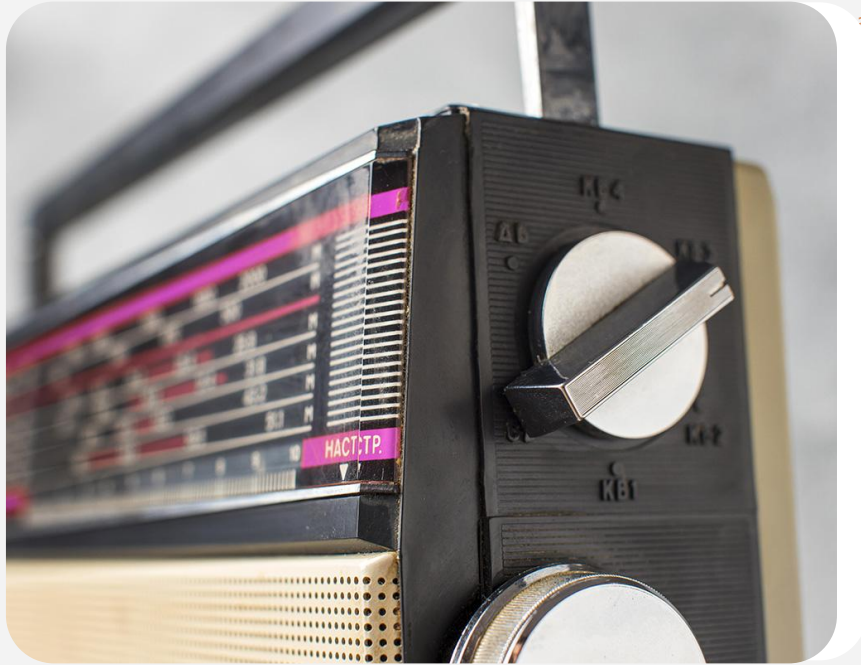
Membuat model matematis dari proses degradasi untuk mengestimasi fungsi degradasi. Biasa dilakukan pada proses degradasi yang kompleks seperti proses kompresi gambar.

$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

$$p(z) = \frac{(np)^z}{z!} e^{-np}$$

$$p(z) = \lambda e^{-\lambda z}$$

$$p(z) = \begin{cases} P_s & \text{if } z = 0 \\ P_p & \text{if } z = 1 \\ 1 - P_s - P_p & \text{if } z = 0 \end{cases}$$



04

Metode Restorasi Sederhana

Mean Filter

Mean filter adalah filter yang mengambil rata-rata dari piksel tetangga.

Mean filter biasa digunakan untuk menghilangkan Gaussian noise dan Uniform noise.

Mean filter dapat dihitung dengan persamaan berikut:

$$\hat{f}(x, y) = \frac{1}{mn} \sum_{s=-a}^a \sum_{t=-b}^b g(x + s, y + t)$$

Order-Statistic Filter

Order-statistic filter adalah filter yang mengambil nilai piksel tertentu dari piksel tetangga.

Contoh dari order-statistic filter adalah median filter.

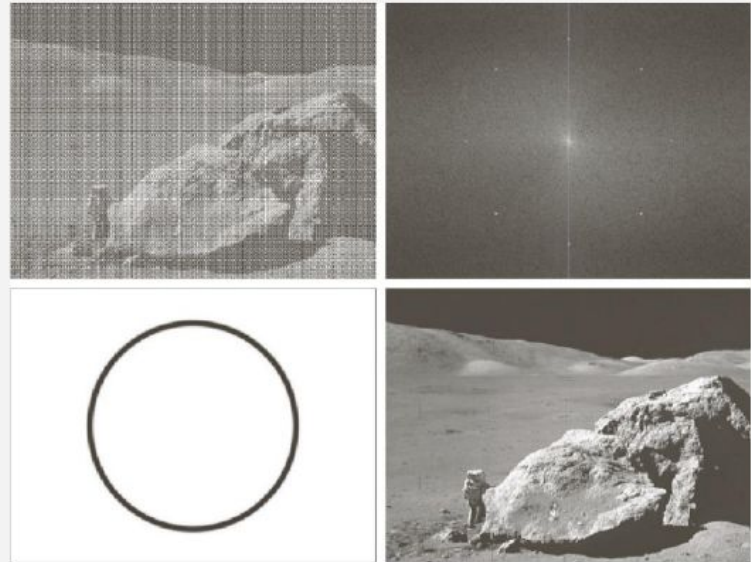
$$\hat{f}(x, y) = \text{median}\{g(x + s, y + t)\}$$

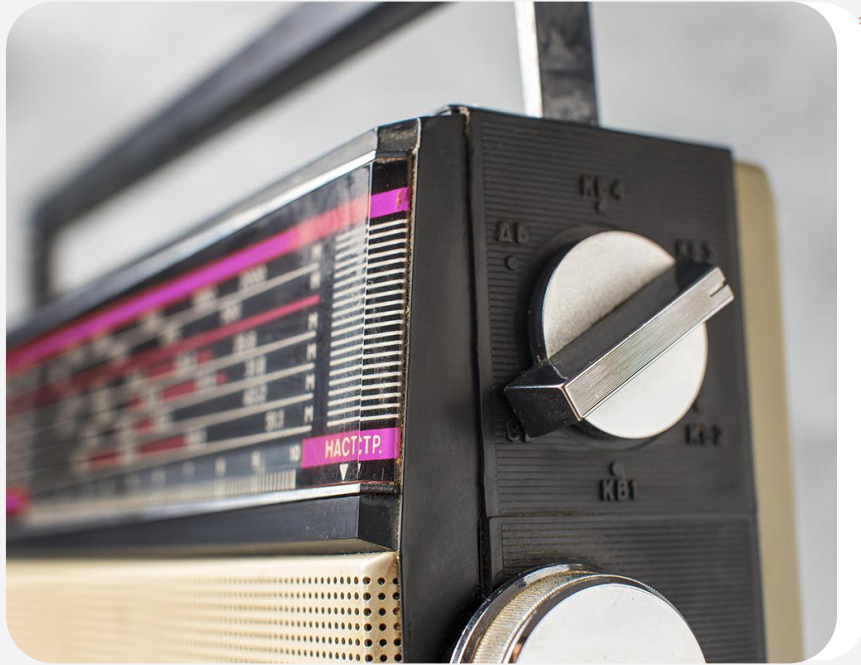
Periodic Noise Reduction

Periodic noise adalah noise yang memiliki frekuensi tertentu. Periodic noise biasa dihilangkan dengan menggunakan filter notch atau filter bandpass.

Dapat dihilangkan dengan bandpass filter yang menghilangkan frekuensi noise dan mempertahankan frekuensi gambar.

Bandpass and Band Reject Filtering



A decorative graphic consisting of several concentric, wavy lines in shades of orange and pink, positioned to the right of the radio image.

05 Metode Restorasi Lanjutan

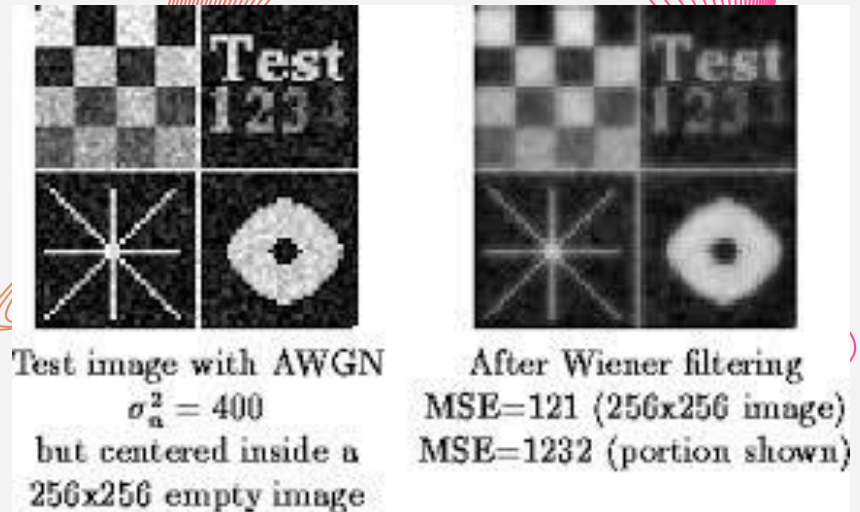
Jenis Metode Restorasi

Deconvolution/Inverse Filtering:

Menggunakan invers dari fungsi degradasi untuk mengembalikan gambar asli. Seluruh informasi gambar asli dapat dikembalikan.

Wiener Filtering: Menggunakan filter Wiener untuk mengembalikan gambar asli. Filter Wiener mempertimbangkan noise yang ada pada gambar.

Blind Deconvolution: Menggunakan metode deconvolution tanpa mengetahui fungsi degradasi. Biasa digunakan pada gambar yang tidak diketahui fungsi degradasinya.



Unconstrained Restoration

Inverse filter unconstrained restoration:

Menggunakan metode restorasi tanpa mempertimbangkan batasan. Proses akan meminimalisir noise, biasa digunakan pada gambar yang memiliki noise yang tinggi.

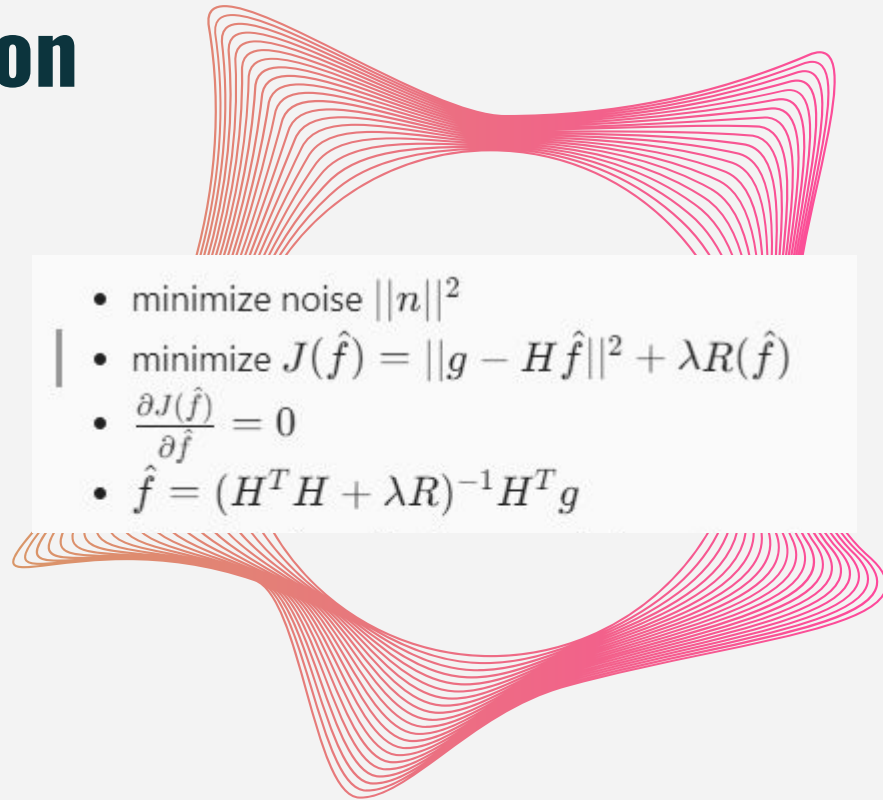
- minimize noise $\|n\|^2$
- $n = g - H\hat{f}$
- minimize $J(\hat{f}) = \|g - H\hat{f}\|^2$
- $\frac{\partial J(\hat{f})}{\partial \hat{f}} = 0$
- $\hat{f} = H^\dagger g = (H^T H)^{-1} H^T g$

Constrained Restoration

Inverse filter constrained restoration:

Menggunakan metode restorasi dengan mempertimbangkan batasan. Proses akan mempertimbangkan noise dan batasan lainnya, biasa digunakan pada gambar yang memiliki noise yang rendah.

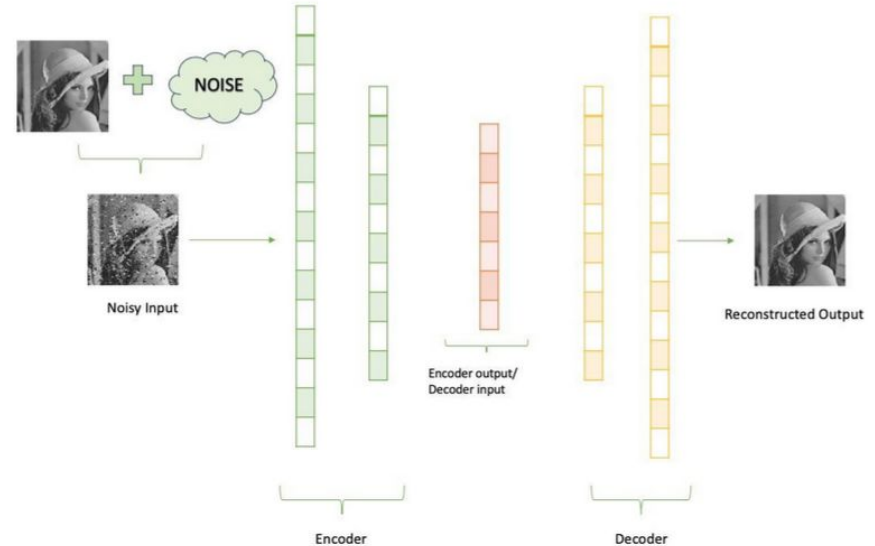
- R adalah fungsi batasan, contohnya fungsi smoothness agar gambar yang dihasilkan lebih halus
- λ adalah hyperparameter yang mengontrol trade-off antara minimasi noise dan mempertimbangkan batasan

- 
- minimize noise $\|n\|^2$
 - minimize $J(\hat{f}) = \|g - H\hat{f}\|^2 + \lambda R(\hat{f})$
 - $\frac{\partial J(\hat{f})}{\partial \hat{f}} = 0$
 - $\hat{f} = (H^T H + \lambda R)^{-1} H^T g$

Denoising Autoencoder

Denoising autoencoder adalah metode restorasi yang menggunakan **autoencoder** untuk **menghilangkan noise** pada gambar.

Autoencoder adalah neural network yang terdiri dari encoder dan decoder. Encoder digunakan untuk menghasilkan **representasi gambar** yang direpresentasikan dengan **sedikit neuron**, sedangkan decoder digunakan untuk **mengembalikan gambar asli** dari representasi gambar.



DAE architecture

Denoising Autoencoder

Autoencoder akan training **meminimalisir error** antara gambar asli dan **gambar yang direkonstruksi** oleh autoencoder.

- minimize $\|E\|^2 = \|f - D(E(f))\|^2$
- Dengan E adalah encoder, D adalah decoder, dan f adalah gambar asli

Denoising autoencoder akan training **meminimalisir error** antara **gambar asli** dan **gambar yang direkonstruksi** oleh autoencoder **dengan noise** yang ditambahkan. Hal ini akan membuat autoencoder belajar untuk menghilangkan noise pada gambar.

- minimize $\|E\|^2 = \|f - D(E(f + n))\|^2$
- Dengan n adalah noise yang ditambahkan pada gambar

Denoising Autoencoder Implementasi

Import Library

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
from keras.preprocessing.image import img_to_array
from keras.layers import Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.models import Sequential
```

✓ 0.0s

Python

Denoising Autoencoder Implementasi

Import gambar dan mengubah ukurannya

```
# Set random seed for reproducibility
np.random.seed(42)

# Load the image
img = cv2.imread('Mona_Lisa.jpg', 1) # Load in color
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # Convert BGR to RGB
rgb_img = cv2.resize(rgb_img, (256, 256)) # Resize to 256x256

# Convert to array and normalize
img_array = img_to_array(rgb_img) / 255.0
img_final = np.expand_dims(img_array, axis=0) # Reshape for model input
```

✓ 0.0s

Python

Denoising Autoencoder Implementasi

Membuat gambar yang memiliki noise untuk autoencoder belajar rentan terhadap noise

- minimize $\|E\|^2 = \|f - D(E(f + n))\|^2$
- Dengan n adalah noise yang ditambahkan pada gambar

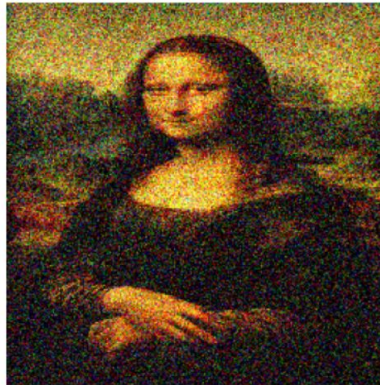
```
# Add Gaussian noise
noise_factor = 0.2
noisy_img = img_final + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=img_final.shape)
noisy_img = np.clip(noisy_img, 0., 1.) # Ensure values remain in [0,1]

# Convert back to uint8 and save
noisy_img_uint8 = (noisy_img[0] * 255).astype(np.uint8)
cv2.imwrite('Noisy_Mona_Lisa.jpg', cv2.cvtColor(noisy_img_uint8, cv2.COLOR_RGB2BGR))
cv2.imwrite('Original_Mona_Lisa.jpg', cv2.cvtColor(rgb_img, cv2.COLOR_RGB2BGR))

# Display the noisy image
plt.imshow(noisy_img[0])
plt.axis('off')
plt.show()
```

✓ 0.0s

Python



Denoising Autoencoder Implementasi

Inisialisasi
model
Autoencoder
Convolutional
Neural
Network (CNN)

```
model = Sequential()

# Encoder
model.add(Conv2D(64, (3,3), activation='relu', padding='same', input_shape=(256,256,3)))
model.add(MaxPooling2D((2,2), padding='same'))
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), padding='same'))
model.add(Conv2D(16, (3,3), activation='relu', padding='same'))
model.add(MaxPooling2D((2,2), padding='same'))

# Decoder
model.add(Conv2D(16, (3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(32, (3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(64, (3,3), activation='relu', padding='same'))
model.add(UpSampling2D((2,2)))
model.add(Conv2D(3, (3,3), activation='sigmoid', padding='same')) # Sigmoid for normalized output

model.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])

model.summary()
```

Denoising Autoencoder Implementasi

Parameter setiap
layer

Model: "sequential_2"

Layer (type)	Output Shape	Param #
conv2d_14 (Conv2D)	(None, 256, 256, 64)	1,792
max_pooling2d_6 (MaxPooling2D)	(None, 128, 128, 64)	0
conv2d_15 (Conv2D)	(None, 128, 128, 32)	18,464
max_pooling2d_7 (MaxPooling2D)	(None, 64, 64, 32)	0
conv2d_16 (Conv2D)	(None, 64, 64, 16)	4,624
max_pooling2d_8 (MaxPooling2D)	(None, 32, 32, 16)	0
conv2d_17 (Conv2D)	(None, 32, 32, 16)	2,320
up_sampling2d_6 (UpSampling2D)	(None, 64, 64, 16)	0
conv2d_18 (Conv2D)	(None, 64, 64, 32)	4,640
up_sampling2d_7 (UpSampling2D)	(None, 128, 128, 32)	0
conv2d_19 (Conv2D)	(None, 128, 128, 64)	18,496
up_sampling2d_8 (UpSampling2D)	(None, 256, 256, 64)	0
conv2d_20 (Conv2D)	(None, 256, 256, 3)	1,731

Total params: 52,067 (203.39 KB)

Trainable params: 52,067 (203.39 KB)

Non-trainable params: 0 (0.00 B)

Denoising Autoencoder Implementasi

Training model
selama 10000 epoch

```
model.fit(noisy_img, img_final, epochs=10000, shuffle=True)
```

✓ 22m 57.3s

Python

```
Epoch 1/10000
1/1 ————— 0s 68ms/step - accuracy: 0.9864 - loss: 0.0025
Epoch 2/10000
1/1 ————— 0s 68ms/step - accuracy: 0.9862 - loss: 0.0026
Epoch 3/10000
1/1 ————— 0s 70ms/step - accuracy: 0.9864 - loss: 0.0025
Epoch 4/10000
1/1 ————— 0s 77ms/step - accuracy: 0.9864 - loss: 0.0025
Epoch 5/10000
1/1 ————— 0s 70ms/step - accuracy: 0.9865 - loss: 0.0025
Epoch 6/10000
1/1 ————— 0s 74ms/step - accuracy: 0.9864 - loss: 0.0024
Epoch 7/10000
1/1 ————— 0s 69ms/step - accuracy: 0.9861 - loss: 0.0025
Epoch 8/10000
1/1 ————— 0s 70ms/step - accuracy: 0.9863 - loss: 0.0025
Epoch 9/10000
1/1 ————— 0s 74ms/step - accuracy: 0.9860 - loss: 0.0025
Epoch 10/10000
1/1 ————— 0s 76ms/step - accuracy: 0.9864 - loss: 0.0025
Epoch 11/10000
1/1 ————— 0s 75ms/step - accuracy: 0.9863 - loss: 0.0025
Epoch 12/10000
1/1 ————— 0s 71ms/step - accuracy: 0.9864 - loss: 0.0025
Epoch 13/10000
...
Epoch 9999/10000
1/1 ————— 0s 156ms/step - accuracy: 0.9876 - loss: 0.0012
Epoch 10000/10000
1/1 ————— 0s 127ms/step - accuracy: 0.9875 - loss: 0.0012
```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)..

Denoising Autoencoder Implementasi

Menampilkan gambar noisy dan rekonstruksi autoencoder. Jika training dengan epoch lebih banyak, loss dapat dikurangi dan image dapat lebih jelas

```
# Add noise again for testing
test_noisy_img = img_final + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=img_final.shape)
test_noisy_img = np.clip(test_noisy_img, 0., 1.)

# Predict the denoised image
denoised_img = model.predict(test_noisy_img)

# Display results
plt.figure(figsize=(10, 5))
suptitle = "Denoising Autoencoder, epochs: 10000, loss: 0.12%"
plt.suptitle(suptitle)

# Noisy Image
plt.subplot(1, 2, 1)
plt.imshow(test_noisy_img[0])
plt.title("Noisy Image")
plt.axis('off')

# Denoised Image
plt.subplot(1, 2, 2)
plt.imshow(denoised_img[0])
plt.title("Denoised Image")
plt.axis('off')

plt.show()
```

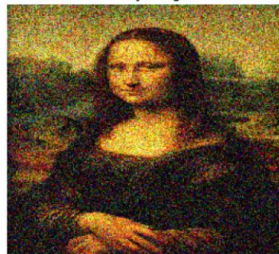
✓ 0.2s

Python

1/1 — 0s 46ms/step

Denoising Autoencoder, epochs: 10000, loss: 0.12%

Noisy Image

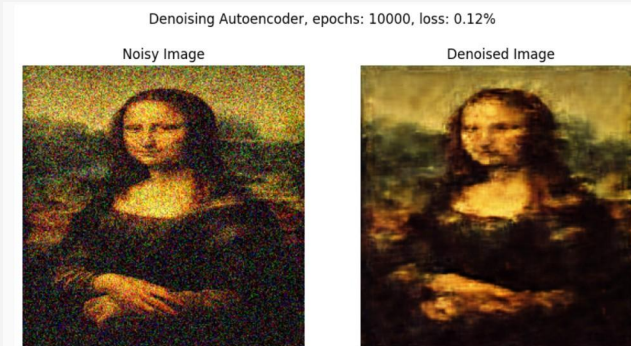
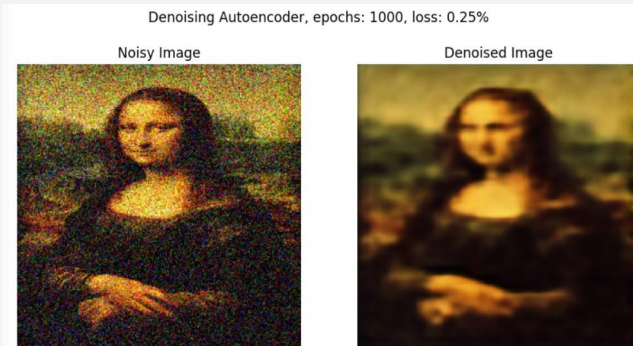
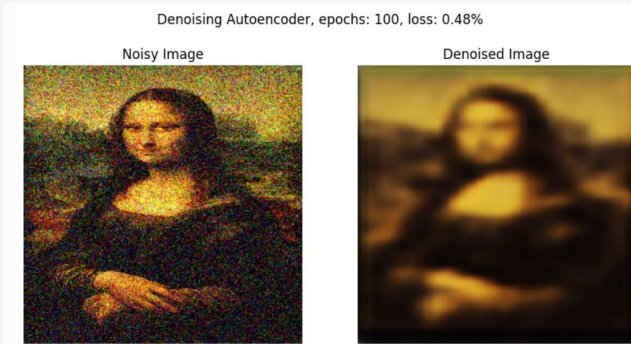
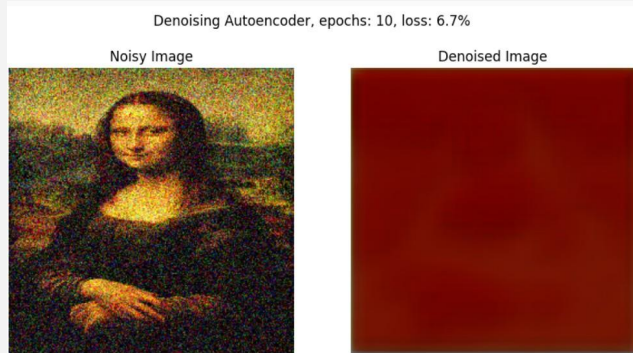


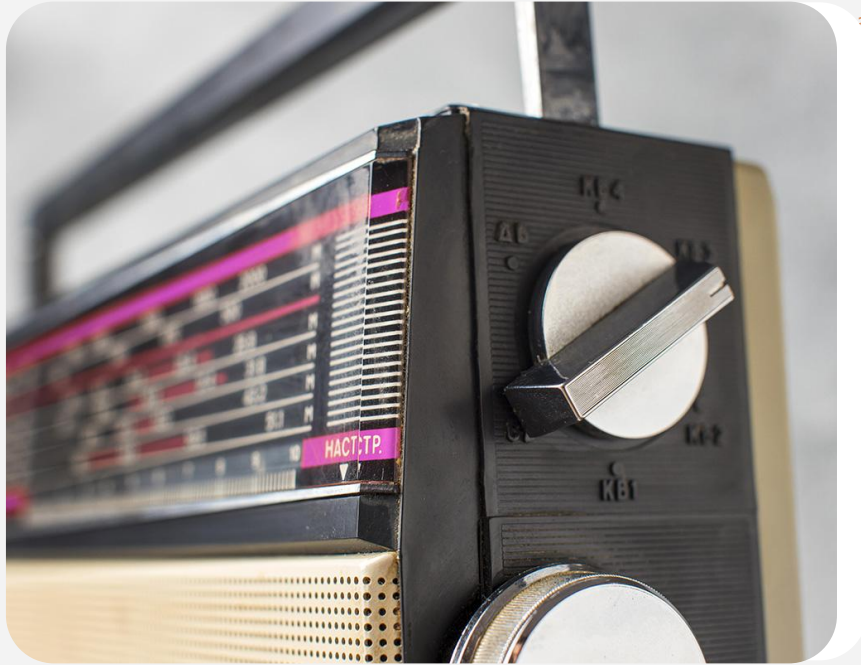
Denoised Image



Denoising Autoencoder Implementasi

Perbandingan jumlah epoch





06 Implementasi Matlab

Kode Periodic Noise

Bandreject (Bandstop) Filtering Script (Gaussian Filter)

```
% This script implements a Gaussian Bandreject (Bandstop) filter in the frequency domain
% and applies it to an input grayscale image.
%
% Before running, ensure you have a grayscale image loaded in your workspace,
% for example, by using:
%
    inputImage = imread('image.png'); % Replace 'your_image.jpg' with your image file
    if size(inputImage, 3) == 3
        inputImage = rgb2gray(inputImage);
    end
%
% You can adjust the filter parameters D0, W below.

% --- Parameters (Adjust these values as needed) ---
D0_reject = 50;    % Center frequency of the bandreject filter
W_reject = 30;     % Bandwidth of the bandreject filter

% --- Input Image (Assuming 'inputImage' is already in your workspace) ---
if ~exist('inputImage', 'var')
    error('Error: Input image ''inputImage'' not found in workspace. Please load a grayscale image into ''inputImage'' first.');
```

```
end

% Convert input image to double if it's uint8 for calculations
if isa(inputImage, 'uint8')
    img_double_reject = im2double(inputImage);
else
    img_double_reject = inputImage;
end
```


Kode Periodic Noise

```
% 1. Fourier Transform of the image
FT_img_reject = fft2(img_double_reject);
FT_img_shifted_reject = fftshift(FT_img_reject); % Shift zero frequency to the center

% 2. Design a Gaussian Bandreject Filter in the frequency domain
[M_reject, N_reject] = size(img_double_reject);
[X_reject, Y_reject] = meshgrid(1:N_reject, 1:M_reject);
centerX_reject = ceil(M_reject/2); centerY_reject = ceil(N_reject/2);
D_reject = sqrt((X_reject - centerX_reject - 1).^2 + (Y_reject - centerY_reject - 1).^2); % Distance from center

% Gaussian Bandreject Filter Transfer Function
H_bandreject = 1 - exp(-(D_reject.^2)/(2*((M_reject/2).^2)) .* (exp(-(D_reject.^2)/(2*D0_reject.^2))));

% 3. Apply the Bandreject Filter in the frequency domain
filtered_FT_shifted_reject = H_bandreject .* FT_img_shifted_reject;
filtered_FT_reject = ifftshift(filtered_FT_shifted_reject); % Shift zero frequency back to corner

% 4. Inverse Fourier Transform to get the filtered image in spatial domain
filtered_img_double_reject = ifft2(filtered_FT_reject);

% 5. Convert back to uint8 for display purposes if original image was uint8
if isa(inputImage, 'uint8')
    filtered_image_reject = im2uint8(real(filtered_img_double_reject)); % Take real part and convert to uint8
else
    filtered_image_reject = real(filtered_img_double_reject); % Take real part
end

% --- Display Results ---
figure; imshow(inputImage); title('Original Image');
```



```
figure; imshow(filtered_image_reject); title('Gaussian Bandreject Filtered Image');
```

Gaussian Bandreject Filtered Image



Perbandingan

