

**Tugas 2 IF4073 Interpretasi dan Pengolahan Citra**  
**Semester I Tahun 2023/2024**



**Disusun oleh:**

Mahesa Lizardy 13520116

Bryan Amirul Husna 13520146

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung**

**Jl. Ganesha 10, Bandung 40132**

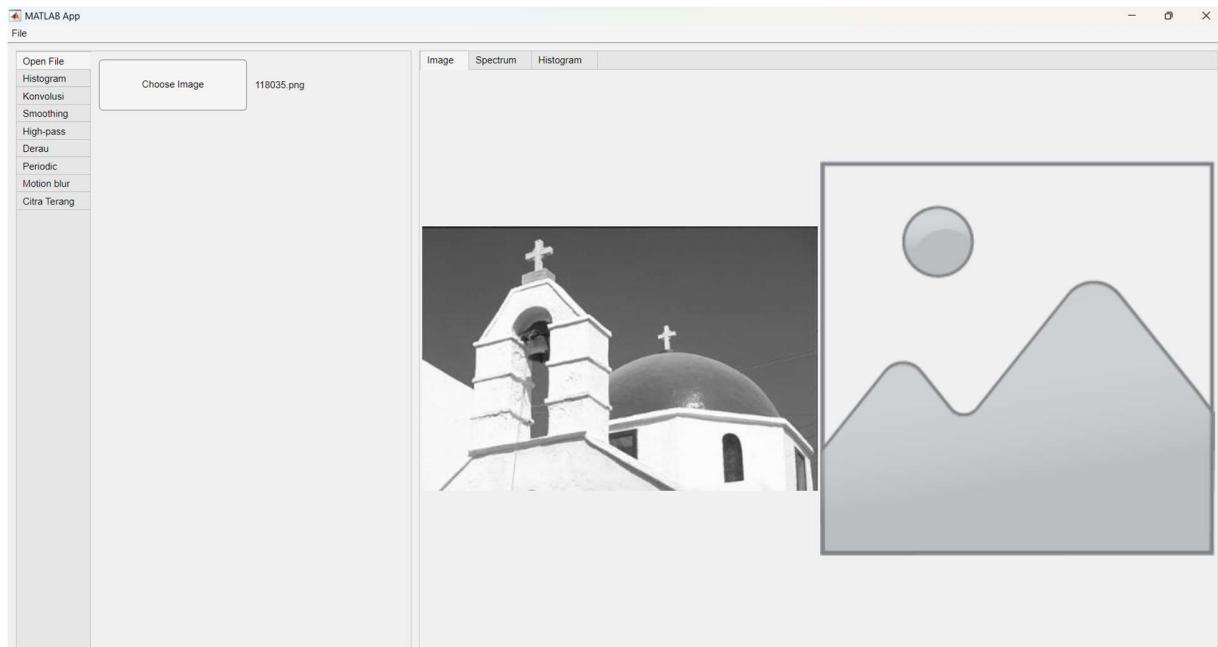
**2023**

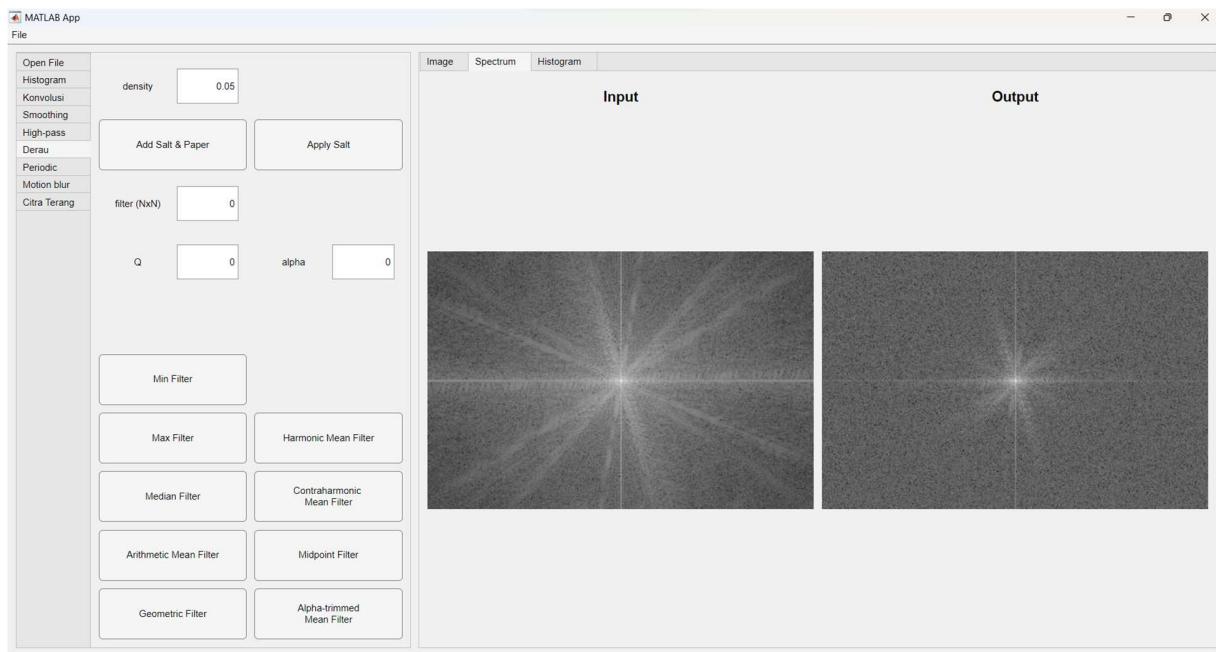
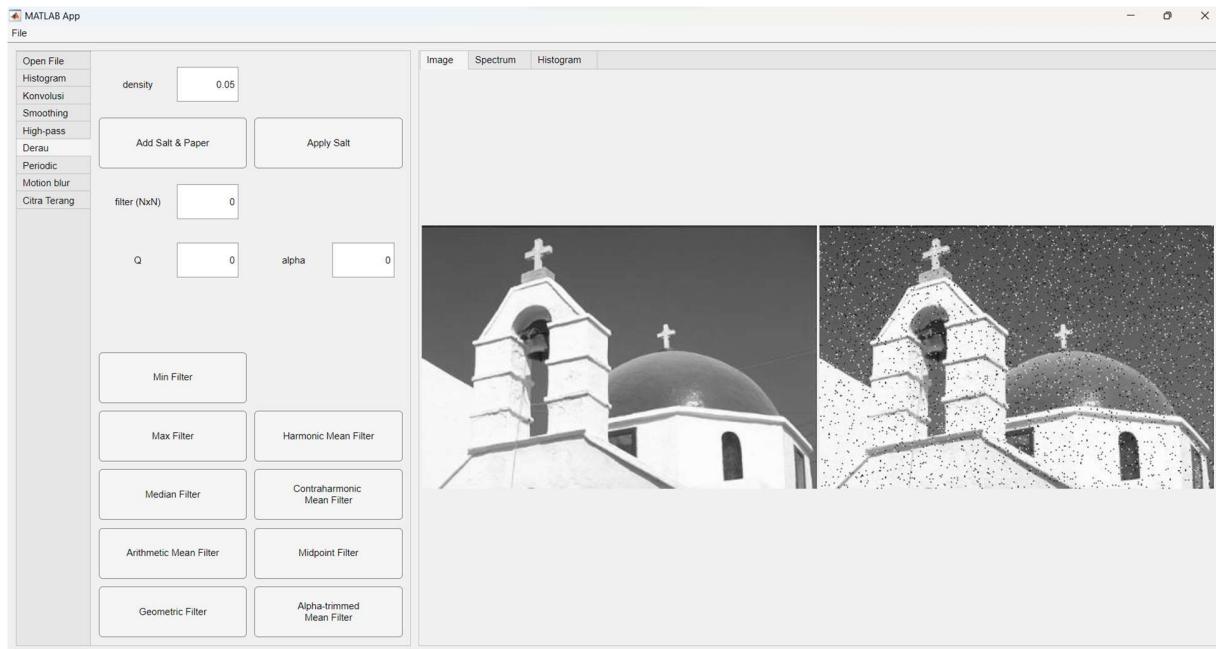
## Daftar Isi

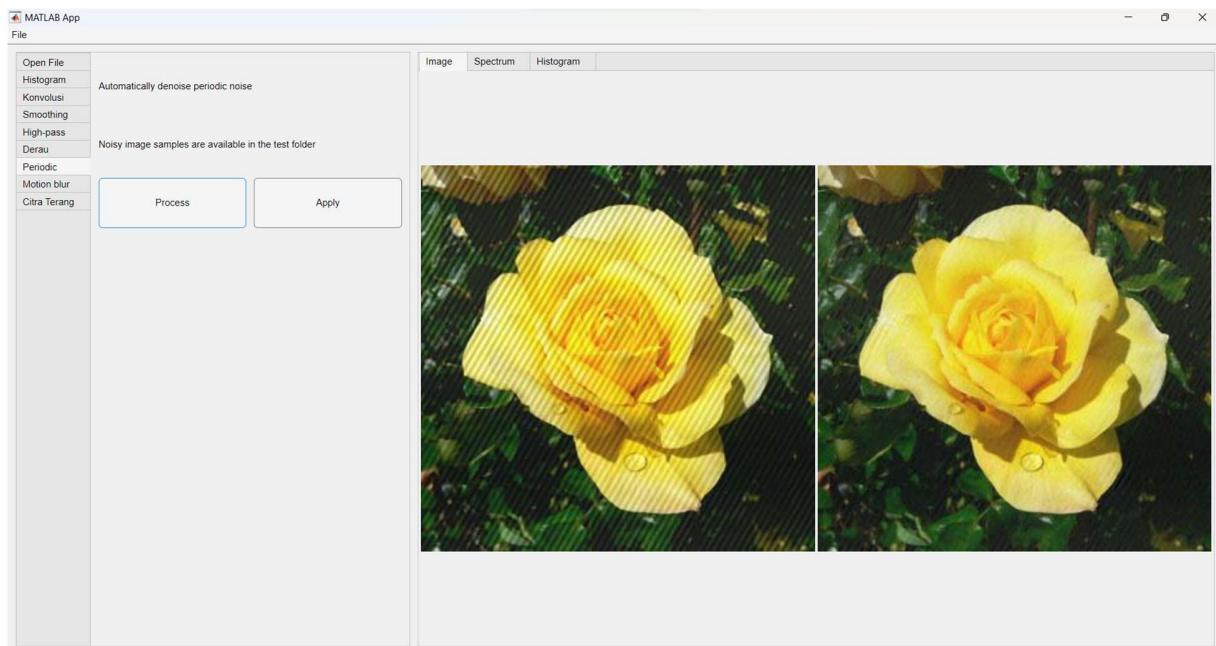
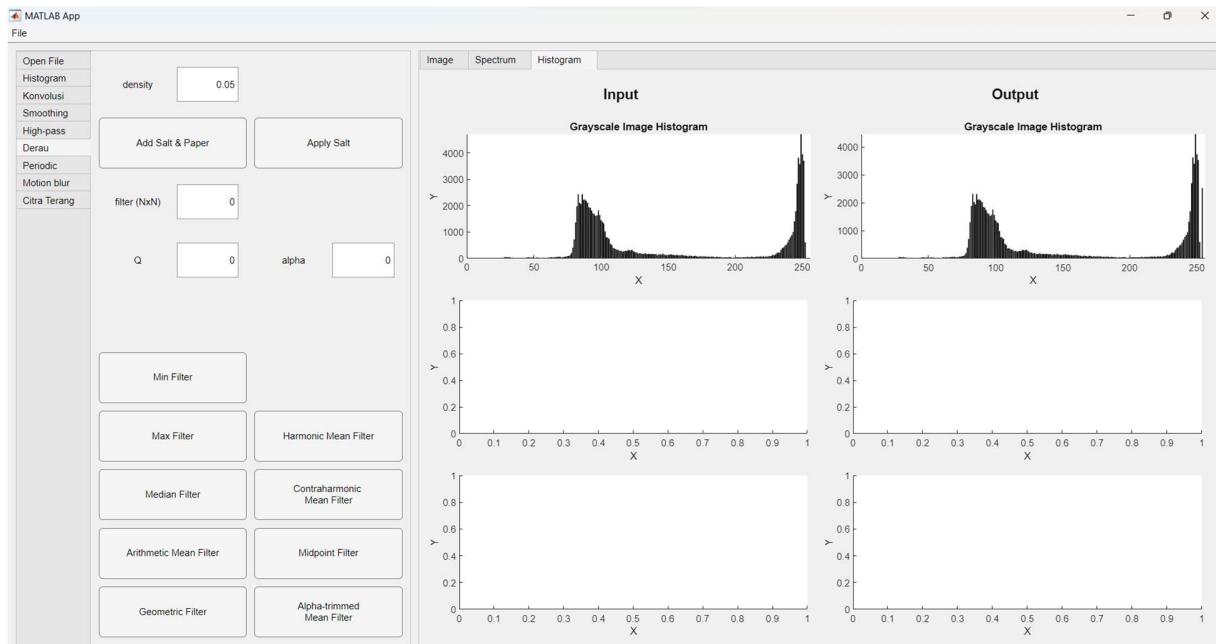
<b>I GUI Program</b>	<b>3</b>
<b>II Pembahasan</b>	<b>7</b>
2.1. Konvolusi citra	7
2.1.1. Kode Program	7
2.1.2. Testing	8
2.1.3. Analisis	10
2.2. Image Smoothing	10
2.2.1. Kode Program	10
2.2.2. Testing	14
2.2.3. Analisis	19
2.3. High-Pass Filtering	19
2.3.1. Kode Program	20
2.3.2. Testing	22
2.3.3. Analisis	25
2.4. Penapisan Citra dalam Ranah Frekuensi untuk 2 Citra Khusus	26
2.4.1. Kode Program	26
2.4.2. Testing	26
2.4.3. Analisis	27
2.5. Penghilangan Derau Salt & Pepper	27
2.5.1. Kode Program	28
2.5.2. Testing	32
2.5.3. Analisis	36
2.6. Penapisan Derau Periodik	36
2.6.1. Kode Program	36
2.6.2. Testing	38
2.6.3. Analisis	44
2.7. Penapis Wiener	45
2.7.1. Kode Program	45
2.7.2. Testing	45
2.7.3. Analisis	47
<b>Alamat GitHub Program</b>	<b>48</b>
<b>Referensi</b>	<b>48</b>

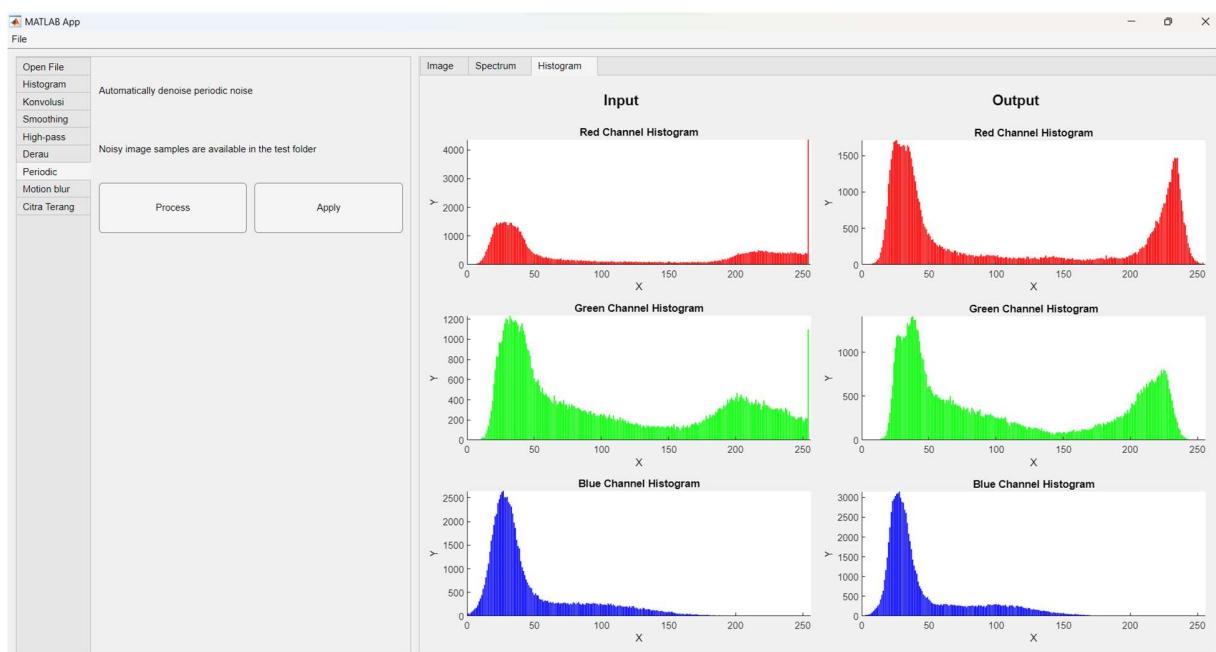
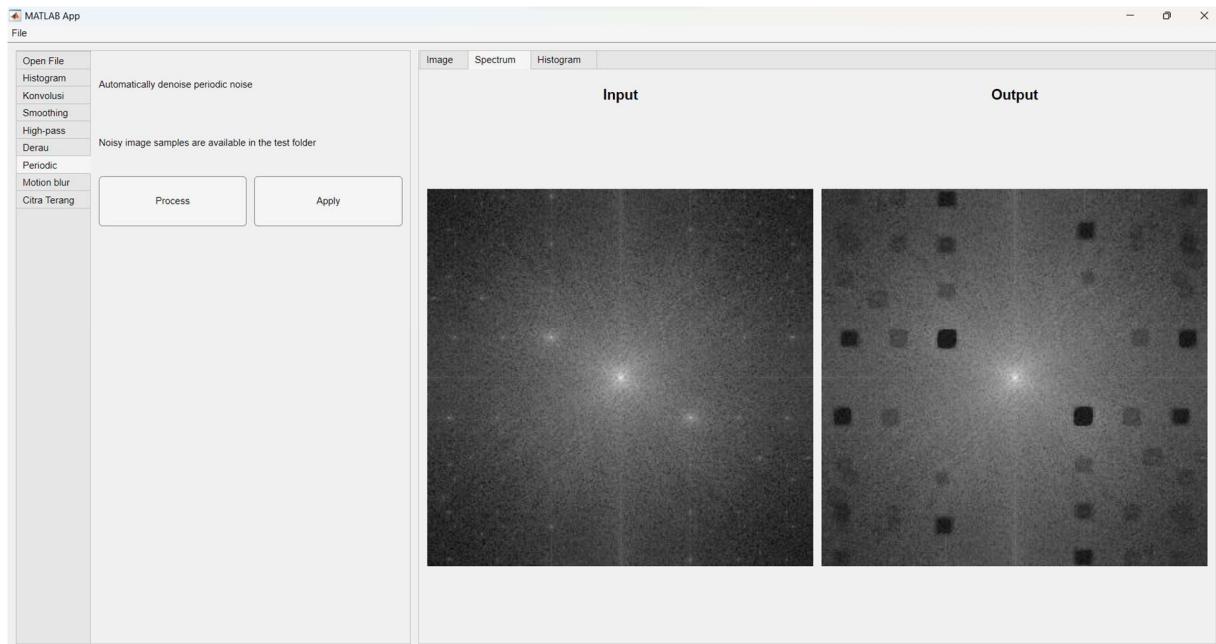
## I GUI Program

Berikut adalah beberapa tangkapan layar GUI program. Sebelum memproses citra, pengguna perlu memasukkan gambar pada menu Open File atau dropdown toolbar “File > Open”. Pada tiap menu transformasi, terdapat tombol “Process” dan “Apply”. Tombol “Process” digunakan untuk melakukan transformasi pada citra masukan, menghasilkan citra, Spectrum, dan histogram keluaran yang dapat dilihat pada tab “Image”, “Spectrum” dan “Histogram”. Tombol “Apply” digunakan untuk mengaplikasikan transformasi sehingga citra luaran menjadi citra masukan (misal ketika ingin menerapkan derau salt & papper / motion blurring ke citra input). Process harus dilakukan sebelum Apply.









## II Pembahasan

### 2.1. Konvolusi citra

Program untuk melakukan konvolusi citra  $f$  berukuran sembarang ( $M \times N$ ) dengan mask berukuran  $n \times n$ , baik pada citra grayscale maupun citra berwarna.

#### 2.1.1. Kode Program

Berikut merupakan potongan kode program untuk konvolusi citra rgb

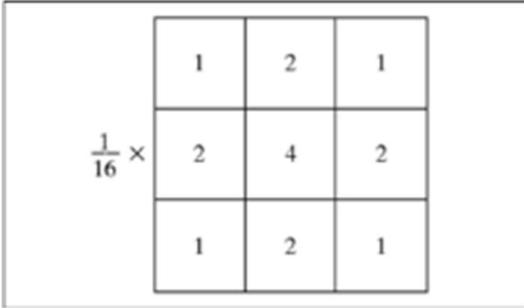
```
% % Melakukan konvolusi citra berukuran sembarang (M x N) dengan mask
% berukuran N x N yang telah ditentukan. Contoh mask 3x3, dan 7x7
gaussian mask
function hasilKonvolusi = konvolusiCitra(citra, mask, padding, stride)
    % mendapatkan ukuran citra dan jumlah saluran warna
    [m, n, numChannels] = size(citra);
    % mendapatkan ukuran mask
    [maskM, maskN] = size(mask);
    % menambahkan Padding citra
    if padding > 0
        citra = padarray(citra, [padding, padding], 0, 'both');
        [m, n, ~] = size(citra);
    end
    % Inisialisasi citra hasil konvolusi
    hasilM = floor((m - maskM) / stride) + 1;
    hasilN = floor((n - maskN) / stride) + 1;
    hasilKonvolusi = zeros(hasilM, hasilN, numChannels);
    % Loop melalui citra citra untuk melakukan konvolusi
    for i = 1:hasilM
        for j = 1:hasilN
            for k = 1:numChannels
                % Tentukan batas atas dan bawah indeks untuk konvolusi
                atas = (i - 1) * stride + 1;
                bawah = atas + maskM - 1;
                kiri = (j - 1) * stride + 1;
                kanan = kiri + maskN - 1;
                % Hitung nilai konvolusi untuk piksel saat ini
                subCitra = citra(atas:bawah, kiri:kanan, k);
                hasilKonvolusi(i, j, k) = sum(sum(subCitra .* mask));
            end
        end
    end
end
```

Program akan mendapatkan ukuran citra dan ukuran filter terlebih dahulu. Program akan menambahkan padding sesuai jumlah yang diinginkan. Program yang dibuat akan menambahkan padding sebesar satu dan stride sebesar satu. Setelah itu, akan diambil pixel-pixel dalam citra dengan ukuran yang sama dengan kernel. Pixel-pixel tersebut nantinya akan dilakukan perkalian dengan kernel.

### 2.1.2. Testing

Berikut testing untuk melakukan konvolusi pada citra dengan menggunakan kernel berikut.

Tabel 2.2.1.1. Kernel yang digunakan untuk pengujian

Kernel 1	Kernel 2
$\frac{1}{16} \times$  $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\frac{1}{140} \times$ $7 \times 7 \text{ Gaussian mask}$ $\begin{bmatrix} 1 & 1 & 2 & 2 & 2 & 1 & 1 \\ 1 & 2 & 2 & 4 & 2 & 2 & 1 \\ 2 & 2 & 4 & 8 & 4 & 2 & 2 \\ 2 & 4 & 8 & 16 & 8 & 4 & 2 \\ 2 & 2 & 4 & 8 & 4 & 2 & 2 \\ 1 & 2 & 2 & 4 & 2 & 2 & 1 \\ 1 & 1 & 2 & 2 & 2 & 1 & 1 \end{bmatrix}$

Tabel 2.2.1.2. Hasil pengujian dengan gambar 1

Input	Output (Kernel 1)	Output (Kernel 2)
		

Tabel 2.2.1.3. Hasil pengujian dengan gambar 2

Input	Output (Kernel 1)	Output (Kernel 2)



Tabel 2.2.1.4. Hasil pengujian dengan gambar 3

Input	Output (Kernel 1)	Output (Kernel 2)

Tabel 2.2.1.5. Hasil pengujian dengan gambar 4

Input	Output (Kernel 1)	Output (Kernel 2)

Tabel 2.2.1.6. Hasil pengujian dengan gambar 5

Input	Output (Kernel 1)	Output (Kernel 2)



Tabel 2.2.1.7. Hasil pengujian dengan gambar 6

Input	Output (Kernel 1)	Output (Kernel 2)
		

### 2.1.3. Analisis

Berdasarkan konvolusi yang telah dilakukan dengan menggunakan kernel 1 dan 2 gambar berhasil dilakukan konvolusi dengan hasil cukup baik. Dapat digunakan kernel lain untuk tujuan *image processing* lainnya sesuai yang diinginkan.

## 2.2. *Image Smoothing*

Program penghalusan citra memiliki beberapa metode dalam domain spasial maupun frekuensi. *Mean filter* dan *gaussian filter* bekerja pada domain spasial, sedangkan ILPF (*ideal low-pass filter*), GLPF (*gaussian low-pass filter*), dan BLPF (*butterworth low-pass filter*) bekerja pada domain frekuensi.

### 2.2.1. Kode Program

Berikut merupakan potongan kode program untuk melakukan *image smoothing*.

#### a. *Mean filter*

```
function result = spatialMeanFilter(f, n)
```

```

mask = ones(n, n) / (n * n);
result = uint8(konvolusiCitra(double(f), mask, floor(n/2), 1));
end

```

Fungsi spatialMeanFilter menerima masukan sebuah data citra  $f$  dan ukuran  $mask$   $n$ . Fungsi kemudian membuat  $mask$  berukuran  $n \times n$  dengan nilai tiap elemennya adalah satu dibagi banyaknya seluruh elemen. Kemudian dilakukan konvolusi antara  $f$  dan  $mask$  sehingga dihasilkan data citra luaran.

#### b. Gaussian filter

```

function result = spatialGaussFilter(f, n)
half_n = floor(n/2);
sigma = (n - 1) / 4; % nilai sigma yang bagus untuk n tersebut

% membuat grid dan mask
[x,y] = meshgrid(-half_n:half_n,-half_n:half_n);
pangkat = -(x.^2 + y.^2)/(2*sigma*sigma);
mask = exp(pangkat) / (2*pi*sigma*sigma);
mask = mask / sum(sum(mask));

% melakukan konvolusi dan mengembalikan hasil
result = uint8(konvolusiCitra(double(f), mask, floor(n/2), 1));
end

```

Fungsi spatialGaussFilter menerima masukan sebuah data citra  $f$  dan ukuran  $mask$   $n$ . Fungsi kemudian membuat sebuah *gaussian mask* berukuran  $n \times n$  dengan nilai tiap elemennya dihitung berdasarkan rumus untuk *gaussian mask* yang memperhitungkan jarak dengan titik pusat  $mask$ . Kemudian dilakukan konvolusi antara  $f$  dan  $mask$  sehingga dihasilkan data citra luaran.

#### c. ILPF

```

function result = ILPFilter(f, D0)
F = fft2(double(f));

% Create ILP Mask
[M,N,z] = size(f);
u = 0:M-1;
v = 0:N-1;
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;
[V, U] = meshgrid(v, u);
D = sqrt(U.^2 + V.^2);

% ILP formula
H = double(D <= D0);

```

```

if z == 3
    H = cat(3, H, H, H);
End

% Do convolution
F2 = H .* F;

% Convert back to spatial domain (inverse fourier transform)
f2 = ifft2(F2);
result = uint8(f2);
end

```

Fungsi ILPFilter menerima masukan sebuah data citra f dan ukuran radius *mask* D0. Data citra f ditransformasikan ke domain frekuensi menggunakan fungsi fft2 menghasilkan F. Fungsi kemudian membuat sebuah ILP *mask* H dengan ukuran sesuai citra masukan dan nilai elemennya bernilai 1 untuk  $D \leq D_0$ . Kemudian dilakukan perkalian elemen antara H dan F sehingga dihasilkan data dalam domain frekuensi. Data ini ditransformasikan ke domain spasial sehingga didapatkan keluaran fungsi.

#### d. GLPF

```

function result = GLPFilter(f, D0)
F = fft2(double(f));

% Create GLP Mask
[M, N, z] = size(f);
u = 0:M-1;
v = 0:N-1;
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;
[V, U] = meshgrid(v, u);
D = sqrt(U.^2 + V.^2);

% GLP formula
H = exp(- D .^ 2 / (2 * (D0 ^ 2)));
if z == 3
    H = cat(3, H, H, H);
End

% Do convolution
F2 = H .* F;

% Convert back to spatial domain (inverse fourier transform)
f2 = ifft2(F2);
result = uint8(f2);
end

```

Fungsi GLPFilter menerima masukan sebuah data citra f dan ukuran radius *mask* D0. Data citra f ditransformasikan ke domain frekuensi menggunakan fungsi fft2

menghasilkan F. Fungsi kemudian membuat sebuah GLP *mask* H dengan ukuran sesuai citra masukan dan nilai elemennya dihitung berdasarkan rumus *gaussian low-pass filter*. Kemudian dilakukan perkalian elemen antara H dan F sehingga dihasilkan data dalam domain frekuensi. Data ini ditransformasikan ke domain spasial sehingga didapatkan keluaran fungsi.

#### e. BLPF

```

function result = BLPFilter(f, D0)
F = fft2(double(f));

% Create BLP Mask
[M,N,z] = size(f);
u = 0:M-1;
v = 0:N-1;
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;
[V, U] = meshgrid(v, u);
D = sqrt(U.^2 + V.^2);

% BLP formula
n = 1; % use n = 1
H = 1 ./ (1 + (D ./ D0) .^ (2 * n));
if z == 3
    H = cat(3, H, H, H);
end

% Do convolution
F2 = H .* F;

% Convert back to spatial domain (inverse fourier transform)
f2 = ifft2(F2);
result = uint8(f2);
end

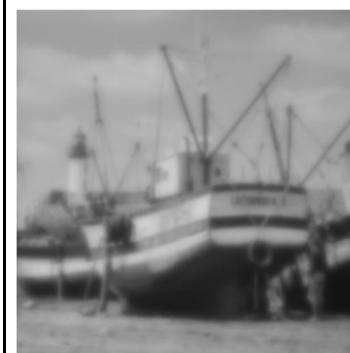
```

Fungsi BLPFilter menerima masukan sebuah data citra f dan ukuran radius *mask* D0. Data citra f ditransformasikan ke domain frekuensi menggunakan fungsi fft2 menghasilkan F. Fungsi kemudian membuat sebuah BLP *mask* H dengan ukuran sesuai citra masukan dan nilai elemennya dihitung berdasarkan rumus *butterworth low-pass filter*. Kemudian dilakukan perkalian elemen antara H dan F sehingga dihasilkan data dalam domain frekuensi. Data ini ditransformasikan ke domain spasial sehingga didapatkan keluaran fungsi.

## 2.2.2. Testing

Berikut testing untuk menguji beberapa penapis yang telah diimplementasikan.

Tabel 2.2.2.1. Hasil pengujian dengan gambar 1

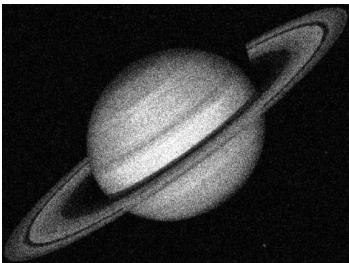
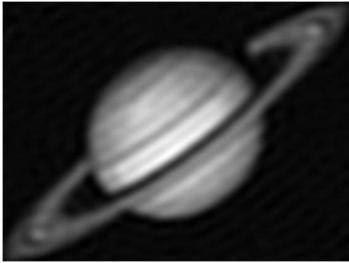
Input	Mean Filter	Gaussian Filter
	 $n = 15$	 $n = 15$
ILPF	GLPF	BLPF
 $D_0 = 20$	 $D_0 = 20$	 $D_0 = 20$

Tabel 2.2.2.2. Hasil pengujian dengan gambar 2

Input	Mean Filter	Gaussian Filter
		

	n = 15	n = 15
<b>ILPF</b>	<b>GLPF</b>	<b>BLPF</b>
		
D0 = 20	D0 = 20	D0 = 20

Tabel 2.2.2.3. Hasil pengujian dengan gambar 3

Input	Mean Filter	Gaussian Filter
		
n = 15	n = 15	n = 15
ILPF	GLPF	BLPF
		
D0 = 20	D0 = 20	D0 = 20

Tabel 2.2.2.4. Hasil pengujian dengan gambar 4

Input	Mean Filter	Gaussian Filter
-------	-------------	-----------------

		
<b>ILPF</b>	<b>GLPF</b>	<b>BLPF</b>
		
D0 = 20	D0 = 20	D0 = 20

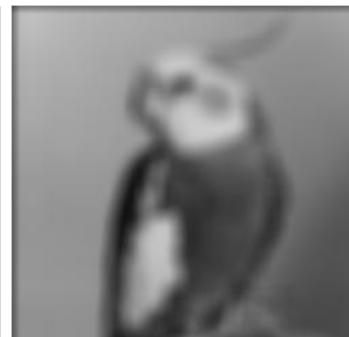
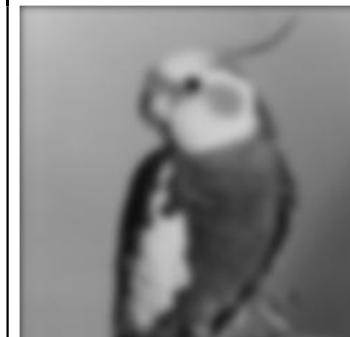
Tabel 2.2.2.5. Hasil pengujian dengan gambar 5

<b>Input</b>	<b>Mean Filter</b>	<b>Gaussian Filter</b>
		
n = 15	n = 15	n = 15
<b>ILPF</b>	<b>GLPF</b>	<b>BLPF</b>
		
D0 = 20	D0 = 20	D0 = 20

Tabel 2.2.2.6. Hasil pengujian dengan gambar 6

<b>Input</b>	<b>Mean Filter</b>	<b>Gaussian Filter</b>
	 n = 15	 n = 15
<b>ILPF</b>	<b>GLPF</b>	<b>BLPF</b>
 D0 = 20	 D0 = 20	 D0 = 20

Tabel 2.2.2.7. Hasil pengujian dengan gambar 7

<b>Input</b>	<b>Mean Filter</b>	<b>Gaussian Filter</b>
	 n = 15	 n = 15
<b>ILPF</b>	<b>GLPF</b>	<b>BLPF</b>



Tabel 2.2.2.8. Hasil pengujian dengan gambar 8

Input	Mean Filter	Gaussian Filter
		
n = 15	n = 15	n = 15
ILPF	GLPF	BLPF
		
D0 = 20	D0 = 20	D0 = 20

### 2.2.3. Analisis

Berdasarkan beberapa percobaan yang telah dilakukan, metode-metode yang dibuat untuk melakukan *low-pass filtering* dapat bekerja dengan baik. Citra-citra hasil penapisan menjadi halus/*blur*. Hasil tiap metode penapisan lolos-rendah memiliki karakteristik tersendiri yang membedakannya dengan yang lain.

## 2.3. High-Pass Filtering

Program untuk melakukan *high-pass filtering* memiliki beberapa metode yang keseluruhannya bekerja pada domain frekuensi. Metode-metode tersebut adalah IHPF (*ideal high-pass filter*), GHPF (*gaussian high-pass filter*), dan BHPF (*butterworth high-pass filter*). Program dapat bekerja pada citra abu maupun berwarna.

### 2.3.1. Kode Program

#### a. IHPF

```
function result = IHPF(f, D0)
F = fft2(double(f));

% Create IHP Mask
[M,N,z] = size(f);
u = 0:M-1;
v = 0:N-1;
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;
[V, U] = meshgrid(v, u);
D = sqrt(U.^2 + V.^2);

% IHP formula
H = 1 - double(D <= D0);
if z == 3
    H = cat(3, H, H, H);
end

% Do convolution
F2 = H .* F;

% Convert back to spatial domain (inverse fourier transform)
f2 = ifft2(F2);
result = uint8(f2);
end
```

Fungsi IHPF menerima masukan sebuah data citra f dan ukuran radius *mask* D0. Secara umum, cara kerjanya mirip dengan ILPFilter yang dapat dilihat pada subbab

2.2.1.c. Perbedaannya terletak pada rumus *mask* yang digunakan. *Mask* yang digunakan merupakan kebalikan dari ILPF, yaitu  $H_{\text{IHPF}} = 1 - H_{\text{ILPF}}$ .

### b. GHPF

```
function result = GHPFilter(f, D0)
F = fft2(double(f));

% Create GHP Mask
[M,N,z] = size(f);
u = 0:M-1;
v = 0:N-1;
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;
[V, U] = meshgrid(v, u);
D = sqrt(U.^2 + V.^2);

% GHP formula
H = 1 - exp(- D .^ 2 / (2 * (D0 ^ 2)));
if z == 3
    H = cat(3, H, H, H);
end

% Do convolution
F2 = H .* F;

% Convert back to spatial domain (inverse fourier transform)
f2 = ifft2(F2);
result = uint8(f2);
end
```

Fungsi GHPFilter menerima masukan sebuah data citra f dan ukuran radius *mask* D0. Secara umum, cara kerjanya mirip dengan GLPFilter yang dapat dilihat pada subbab 2.2.1.d. Perbedaannya terletak pada rumus *mask* yang digunakan. *Mask* yang digunakan merupakan kebalikan dari GLPF, yaitu  $H_{\text{GHPF}} = 1 - H_{\text{GLPF}}$ .

### c. BHPF

```
function result = BHPFilter(f, D0)
F = fft2(double(f));

% Create BHP Mask
[M,N,z] = size(f);
u = 0:M-1;
v = 0:N-1;
idx = find(u > M/2);
u(idx) = u(idx) - M;
idy = find(v > N/2);
v(idy) = v(idy) - N;
```

```

[V, U] = meshgrid(v, u);
D = sqrt(U.^2 + V.^2);

% BHP formula
n = 1; % use n = 1
H = 1 - 1 ./ (1 + (D ./ D0) .^ (2 * n));
if z == 3
    H = cat(3, H, H, H);
end

% Do convolution
F2 = H .* F;

% Convert back to spatial domain (inverse fourier transform)
f2 = ifft2(F2);
result = uint8(f2);
end

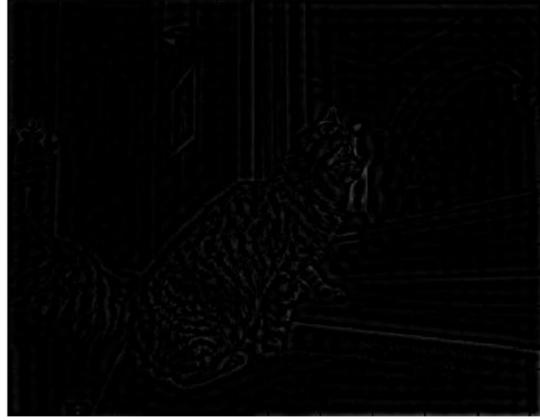
```

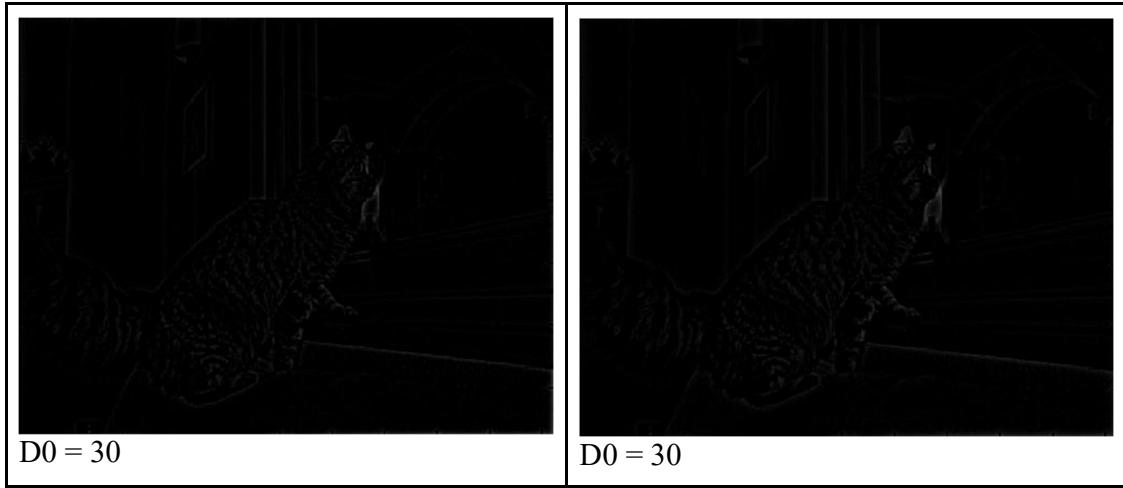
Fungsi BHPFilter menerima masukan sebuah data citra f dan ukuran radius *mask* D0. Secara umum, cara kerjanya mirip dengan GHPFilter yang dapat dilihat pada subbab 2.2.1.e. Perbedaannya terletak pada rumus *mask* yang digunakan. *Mask* yang digunakan merupakan kebalikan dari BLPF, yatu  $H_{BHPF} = 1 - H_{BLPF}$ .

### 2.3.2. Testing

Berikut testing untuk menguji penapis wiener yang telah diimplementasikan

Tabel 2.2.3.1. Hasil pengujian dengan gambar 1

Input	IHPF
	 D0 = 30
GHPF	BHPF



Tabel 2.2.3.2. Hasil pengujian dengan gambar 2

Input	IHPF
	
D0 = 30	D0 = 30
GHPF	BHPF
	
D0 = 30	D0 = 30

Tabel 2.2.3.3. Hasil pengujian dengan gambar 3

Input	IHPF
	 D0 = 30
GHPF	BHPF
 D0 = 30	 D0 = 30

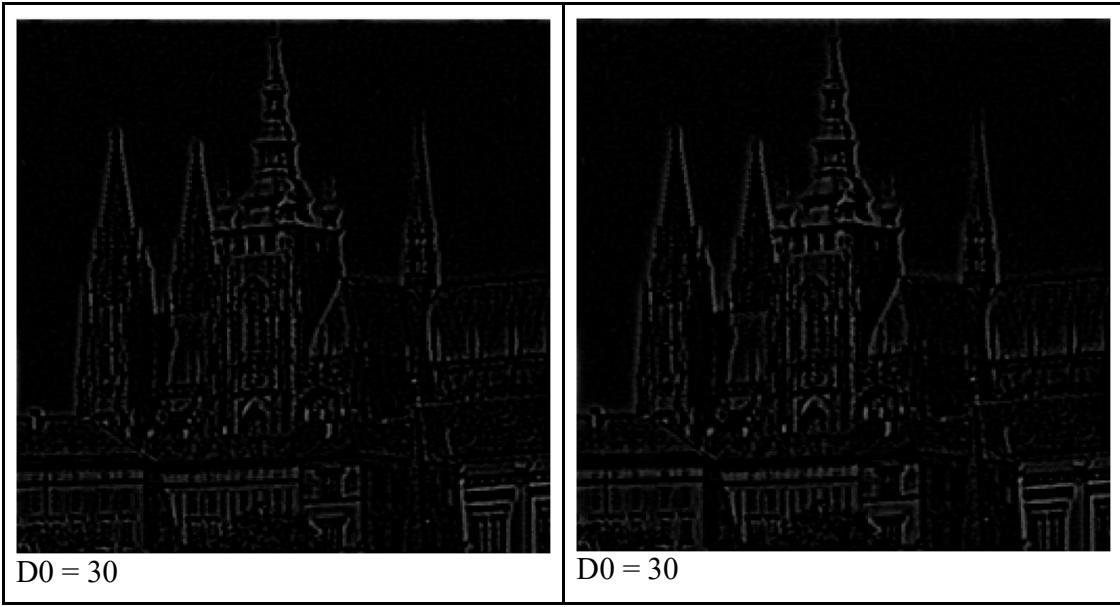
Tabel 2.2.3.4. Hasil pengujian dengan gambar 4

Input	IHPF
	 D0 = 30

GHPF	BHPF
 D0 = 30	 D0 = 30

Tabel 2.2.3.5. Hasil pengujian dengan gambar 5

Input	IHPF
	 D0 = 30
GHPF	BHPF



### 2.3.3. Analisis

Berdasarkan beberapa percobaan yang telah dilakukan, metode-metode yang dibuat untuk melakukan *high-pass filtering* dapat bekerja dengan baik. Dapat dihasilkan citra keluaran berupa komponen frekuensi tinggi dari citra masukan. Hasil tiap metode penapisan lolos-tinggi memiliki karakteristik tersendiri yang membedakannya dengan yang lain.

## 2.4. Penapisan Citra dalam Ranah Frekuensi untuk 2 Citra Khusus

Program untuk menghasilkan citra yang lebih terang dengan menggunakan penapisan citra dalam ranah frekuensi.

### 2.4.1. Kode Program

Berikut merupakan fungsi untuk melakukan penapisan citra dalam rana frekuensi untuk menghasilkan citra yang lebih terang

```
function result = citraTerang(f, X)
F =(fft2(f));
F2 = X.*F;
% Convert back to spatial domain (inverse fourier transform)
F2 = ifft2(F2);
% result = uint8(f2);
result = uint8(F2);
end
```

Function citraTerang akan mengubah citra  $f$  menjadi domain frekuensi. Lalu domain frekuensi citra akan dikalikan dengan  $X$ . untuk menghasilkan citra yang lebih terang maka nilai  $X > 1$ .

#### 2.4.2. Testing

Berikut testing untuk menguji fungsi untuk menghasilkan citra yang lebih terang

Tabel 2.2.4.1. Hasil pengujian dengan gambar 1

Input	Output
	 $X=2$

Tabel 2.2.4.2. Hasil pengujian dengan gambar 2

Input	Output



#### 2.4.3. Analisis

Berdasarkan Percobaan yang telah dilakukan fungsi yang dibuat sudah cukup baik untuk membuat citra 1 dan citra 2. Didapatkan citra yang lebih terang dengan melakukan penapisan citra dalam ranah frekuensi.

### 2.5. Penghilangan Derau *Salt & Papper*

Pada bagian ini terdiri dari program untuk menambahkan derau *salt & papper* dan melakukan penghilangan derau dengan min filter, max filter, median filter, arithmetic mean filter, geometric filter, harmonic mean filter, contraharmonic mean filter, midpoint filter, alpha-trimmed mean filter.

#### 2.5.1. Kode Program

##### a. Min Filter

```
%% Function min filter  
% finding the darkest point in an image
```

```
% reducing salt noise
function hasilFilter = minFilter(citra, ukuranFilter)
    % mendapatkan ukuran citra
    [m, n] = size(citra);
    % inisialisasi output image hasil min filter
    hasilFilter = citra;
    % menghitung radius
    radius = floor(ukuranFilter / 2);
    for i = (1+radius):(m-radius)
        for j = (1+radius):(n-radius)
            % Hitung nilai minFilter untuk sub image
            subCitra = citra(i-radius:i+radius, j-radius:j+radius);
            hasilFilter(i, j) = min(subCitra(:));
        end
    end
end
```

Function minFilter akan mengambil sub citra yang merupakan pixel-pixel pada citra yang berukuran sama dengan kernel. Berdasarkan sub citra tersebut akan dicari nilai yang minimum. Nilai terkecil tersebut akan mengganti pixel tengah yang terdapat pada sub citra.

### b. Max Filter

```
%% Function max filter
% finding the brightest point in an image
% reducing papper noise
function hasilFilter = maxFilter(citra, ukuranFilter)
    % mendapatkan ukuran citra
    [m, n] = size(citra);
    % inisialisasi output image hasil min filter
    hasilFilter = citra;
    % menghitung radius
    radius = floor(ukuranFilter / 2);
    for i = (1+radius):(m-radius)
        for j = (1+radius):(n-radius)
            % Hitung nilai maxFilter untuk sub image
            subCitra = citra(i-radius:i+radius, j-radius:j+radius);
            hasilFilter(i, j) = max(subCitra(:));
        end
    end
end
```

Function maxFilter akan mengambil sub citra yang merupakan pixel-pixel pada citra yang berukuran sama dengan kernel. Berdasarkan sub citra tersebut akan dicari nilai yang maksimum. Nilai terkecil tersebut akan mengganti pixel tengah yang terdapat pada sub citra.

### c. Median Filter

```
%% Function median filter
```

```
% handling both bipolar or unipolar impulse noise
function hasilFilter = medianFilter(citra, ukuranFilter)
    % mendapatkan ukuran citra
    [m, n] = size(citra);
    % inisialisasi output image hasil min filter
    hasilFilter = citra;
    % menghitung radius
    radius = floor(ukuranFilter / 2);
    for i = (1+radius):(m-radius)
        for j = (1+radius):(n-radius)
            % Hitung nilai medianFilter untuk sub image
            subCitra = citra(i-radius:i+radius, j-radius:j+radius);
            hasilFilter(i, j) = median(subCitra(:));
        end
    end
end
```

Function medianFilter akan mengambil sub citra yang merupakan pixel-pixel pada citra yang berukuran sama dengan kernel. Berdasarkan sub citra tersebut akan dicari nilai median-nya. Nilai terkecil tersebut akan mengganti pixel tengah yang terdapat pada sub citra.

#### d. Arithmetic Mean Filter

```
%% Function untuk Arithmetic Mean Filter
% Melakukan filter rata-rata aritmatika pada citra
function hasilFilter = arithmeticMeanFilter(citra, ukuranFilter)
    % mendapatkan ukuran citra
    [m, n] = size(citra);
    % inisialisasi output image hasil min filter
    hasilFilter = citra;
    % menghitung radius
    radius = floor(ukuranFilter / 2);
    for i = (1+radius):(m-radius)
        for j = (1+radius):(n-radius)
            % Hitung nilai arithmetic mean Filter untuk sub citra
            subCitra = citra(i-radius:i+radius, j-radius:j+radius);
            hasilFilter(i, j) = uint8(mean(subCitra(:)));
        end
    end
end
```

Function meanFilter akan mengambil sub citra yang merupakan pixel-pixel pada citra yang berukuran sama dengan kernel. Berdasarkan sub citra tersebut akan dicari nilai rata-ratanya. Nilai rata-rata tersebut akan mengganti pixel tengah yang terdapat pada sub citra.

#### e. Geometric Mean Filter

```
%% Function untuk Geometric Mean Filter
% Melakukan filter rata-rata geometrik pada citra
```

```

function hasilFilter = geometricMeanFilter(citra, ukuranFilter)
    % mendapatkan ukuran citra
    [m, n] = size(citra);
    % inisialisasi output image hasil min filter
    hasilFilter = citra;
    % menghitung radius
    radius = floor(ukuranFilter / 2);
    for i = (1+radius):(m-radius)
        for j = (1+radius):(n-radius)
            % Hitung nilai geometric mean Filter untuk sub citra
            subCitra = double(citra(i-radius:i+radius, j-radius:j+radius));
            hasilFilter(i, j) = uint8(prod(subCitra(:)) .^
                (1/numel(subCitra))));
        end
    end
end

```

Function geometricMeanFilter akan mengambil sub citra yang merupakan pixel-pixel pada citra yang berukuran sama dengan kernel. Berdasarkan sub citra tersebut akan dihitung nilai Geometric Mean Filter. Nilai tersebut akan mengganti pixel tengah yang terdapat pada sub citra.

#### f. Harmonic Mean Filter

```

%% Fungsi untuk Harmonic Mean Filter
function hasilFilter = harmonicMeanFilter(citra, ukuranFilter)
    % mendapatkan ukuran citra
    [m, n] = size(citra);
    % inisialisasi output image hasil min filter
    hasilFilter = citra;
    % menghitung radius
    radius = floor(ukuranFilter / 2);
    for i = (1+radius):(m-radius)
        for j = (1+radius):(n-radius)
            % Hitung nilai harmonic mean Filter untuk sub citra
            subCitra = citra(i-radius:i+radius, j-radius:j+radius);
            rataRataHarmonik = numel(subCitra) / sum(1 ./
                double(subCitra(:)));
            hasilFilter(i, j) = uint8(rataRataHarmonik);
        end
    end
end

```

Function harmonicMeanFilter akan mengambil sub citra yang merupakan pixel-pixel pada citra yang berukuran sama dengan kernel. Berdasarkan sub citra tersebut akan dihitung nilai Harmonic Mean Filter. Nilai tersebut akan mengganti pixel tengah yang terdapat pada sub citra.

#### g. Contraharmonic Mean Filter

```

%% Fungsi untuk Contraharmonic Mean Filter

```

```

function hasilFilter = contraharmonicMeanFilter(citra, ukuranFilter,
Q)
    % mendapatkan ukuran citra
    [m, n] = size(citra);
    % inisialisasi output image hasil min filter
    hasilFilter = citra;
    % menghitung radius
    radius = floor(ukuranFilter / 2);
    for i = (1+radius):(m-radius)
        for j = (1+radius):(n-radius)
            % Hitung nilai Contraharmonic mean Filter untuk sub citra
            subCitra = double(citra(i-radius:i+radius, j-radius:j+radius));
            rataRataContraharmonik = sum(subCitra(:) .^ (Q+1.0)) /
            sum(subCitra(:) .^ Q);
            hasilFilter(i, j) = uint8(rataRataContraharmonik);
        end
    end
end

```

Function contraharmonicMeanFilter akan mengambil sub citra yang merupakan pixel-pixel pada citra yang berukuran sama dengan kernel. Pada Contraharmonic Mean Filter terdapat input Q yang merupakan parameter kontras yang menentukan jenis filter contraharmonic yang akan diterapkan. Nilai Q dapat berupa nilai positif, negatif, maupun nol. Berdasarkan sub citra tersebut akan dihitung nilai Contraharmonic Mean Filter. Nilai tersebut akan mengganti pixel tengah yang terdapat pada sub citra.

#### h. Midpoint Filter

```

function hasilFilter = midpointFilter(citra, ukuranFilter)
    % hitung min filter
    minFiltered = minFilter(citra, ukuranFilter);
    % hitung max filter
    maxFiltered = maxFilter(citra, ukuranFilter);
    % hitung midpoint
    hasilFilter = (minFiltered + maxFiltered) / 2;
end

```

Function midpointFilter akan menghitung nilai minFilter dan maxFilter dari citra. Hasil dari keduanya akan ditambahkan dan dibagi dengan dua.

#### i. Alpha-Trimmed Mean Filter

```

% Fungsi untuk menghilangkan derau dengan alpha-trimmed mean filter
function hasilFilter = alphaTrimmedMeanFilter(citra, ukuranFilter,
alpha)
    % Mendapatkan ukuran citra input
    [m, n] = size(citra);
    % Menginisialisasi citra hasil filter dengan citra input
    hasilFilter = citra;
    % Menghitung radius filter

```

```

radius = floor(ukuranFilter / 2);
for i = (1+radius):(m-radius)
    for j = (1+radius):(n-radius)
        % Mengambil sub-citra dengan ukuran filter
        subCitra = citra(i-radius:i+radius, j-radius:j+radius);
        % Mengubah sub-citra menjadi array satu dimensi
        subCitra = double(subCitra(:));
        % sorting pixel
        subCitraSorted = sort(subCitra);
        % Menghapus alpha piksel terendah dan tertinggi
        subCitraTrimmed = subCitraSorted(alpha+1:end-alpha);
        % Menghitung nilai rata-rata
        hasilFilter(i, j) = uint8(mean(subCitraTrimmed(:)));
    end
end

```

Function alpha-trimmed seperti pada fungsi arithmetic mean filter pada biasanya namun subcitra yang akan dihitung rata-ratanya akan dipotong sepanjang alpha awal dan alpha akhir sebelum dihitung.

## 2.5.2. Testing

Berikut testing untuk menguji filter yang telah diimplementasikan

Tabel 2.2.5.1. Hasil pengujian dengan gambar 1

Input	Add Salt AndPaper	Output Min Filter
	 Density = 0.05	 Filter = 3 x 3;
Output Max Filter	Output Median Filter	Output Arithmetic Mean Filter
 Filter = 3 x 3;	 Filter = 3 x 3;	 Filter = 3 x 3;
Output Geometric Mean	Output Harmonic Mean	Output Contraharmonic

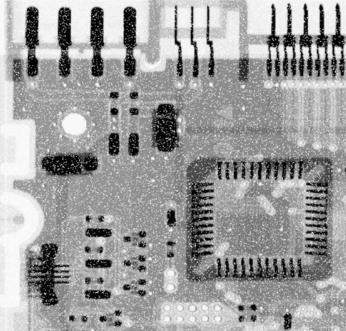
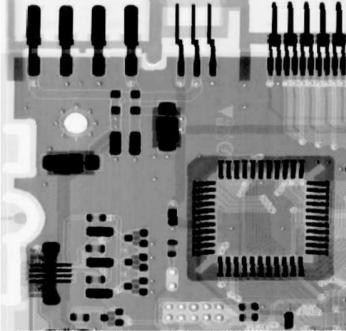
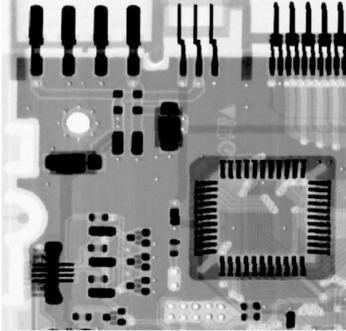
Filter	Filter	Mean Filter
		
Filter = 3 x 3;	Filter = 3 x 3;	Filter = 3 x 3; Q = 0.01
Output Midpoint Filter	Output Alpha-Trimmed Mean Filter	
		
Filter = 3 x 3;	Filter = 3 x 3; alpha = 2	

Tabel 2.2.5.2. Hasil pengujian dengan gambar 2

Input	Add Salt AndPaper	Output Min Filter
		
Density = 0.01		Filter = 3 x 3
Output Max Filter	Output Median Filter	Output Arithmetic Mean Filter
		
Filter = 3 x 3;	Filter = 3 x 3;	Filter = 3 x 3;

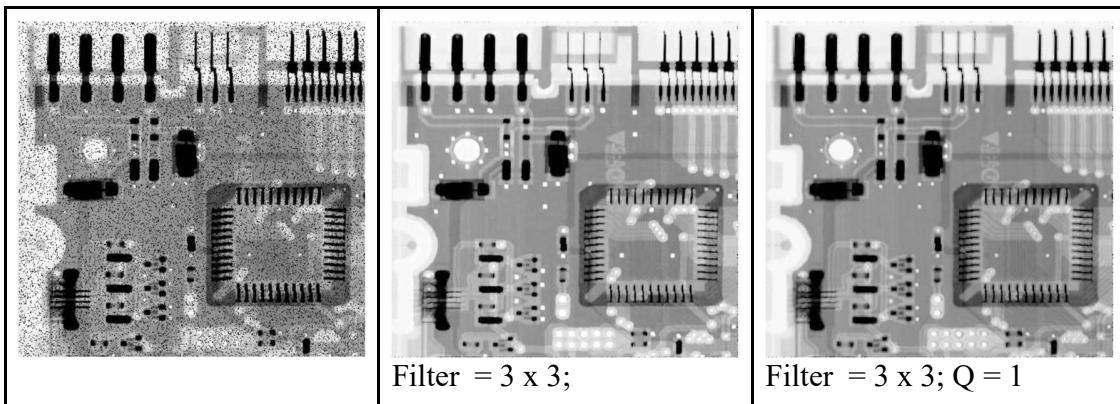
<b>Output Geometric Mean Filter</b>	<b>Output Harmonic Mean Filter</b>	<b>Output Contraharmonic Mean Filter</b>
		
Filter = 3 x 3;	Filter = 3 x 3;	Filter = 3 x 3; Q = 0.1
<b>Output Midpoint Filter</b>	<b>Output Alpha-Trimmed Mean Filter</b>	
		
Filter = 3 x 3;	Filter = 3 x 3; alpha = 2	

Tabel 2.2.5.3. Hasil pengujian dengan gambar 3

<b>Input (Salt Noise)</b>	<b>Min Filter</b>	<b>Contraharmonic Mean Filter</b>
		
	Filter = 3 x 3;	Filter = 3 x 3; Q = -5

Tabel 2.2.5.4. Hasil pengujian dengan gambar 4

<b>Input (Papper Noise)</b>	<b>Max Filter</b>	<b>Contraharmonic Mean Filter</b>
-----------------------------	-------------------	-----------------------------------



Tabel 2.2.5.5. Hasil pengujian dengan gambar 5

Input (Noisy Gaussian)	Output Arithmetic Mean Filter	Output Geometric Mean Filter

### 2.5.3. Analisis

Berdasarkan Percobaan yang dilakukan filter sudah dapat menghilangkan derau dengan baik. Namun, tidak setiap gambar dapat menggunakan filter yang sama untuk menghilangkan derau. Seperti pada max filter dan Contraharmonic Min Filter dengan parameter  $Q < 0$  dapat mengurangi derau *salt*, sedangkan pada min filter, harmonic filter, dan Contraharmonic Mean Filter dengan parameter  $Q > 0$  dapat mengurangi derau *papper*

## 2.6. Penapisan Derau Periodik

Program menghilangkan atau mengurangi derau periodik dengan *gaussian notch filter* secara otomatis. Dengan kata lain, pengguna tidak perlu menspesifikasikan titik-titik terang pada spektrum Fourier citra yang menyebabkan derau periodik. Program mengacu

pada artikel "Adaptive Gaussian notch filter for removing periodic noise from digital images" yang dapat diakses di <https://doi.org/10.1049/iet-ipr.2018.5707>.

### 2.6.1. Kode Program

Berikut merupakan potongan kode program untuk menghilangkan atau mengurangi derau periodik.

```
% Cara diambil dari https://doi.org/10.1049/iet-ipr.2018.5707
% "Adaptive Gaussian notch filter for removing periodic noise from
digital images"
function result = denoisePeriodic(f)
    % Parameters
    center_radius = 6;
    th = 0.35;
    wmax = 5;

    F = fftshift(fft2(f)); % do dft and shift to the input image
    Fout = F;
    [M, N, numChannels] = size(F);
    for i = 1:floor(M/2)
        for j = 1:N
            % skip the edge
            if i-wmax < 1 || i+wmax > M || j-wmax < 1 || j+wmax > N
                continue;
            end
            % begin the process
            f1 = false;
            if numChannels == 3
                f1 = [false, false, false];
            end
            d = sqrt((i - M/2)^2 + (j - N/2)^2);
            if d > center_radius % avoid masking the center bright spot
                w1 = 3;
                w2 = w1 + 2;
                stop = false;
                while stop == false
                    w2 = w1+2;
                    % calculate average value of F in a window
                    s1 = F(i-w1:i+w1, j-w1:j+w1, 1:numChannels);
                    s2 = F(i-w2:i+w2, j-w2:j+w2, 1:numChannels) - padarray(s1,
                    [w2-w1, w2-w1], 0);
                    s1 = mean(mean(abs(s1))); % calculate average of F covered
                    % in w1xw1 window
                    s2 = mean(mean(abs(s2))); % calculate average of F covered
                    % in w2xw2 window outside w1xw1
                    f1 = (s2 ./ s1) <= th;
                    if numChannels == 1
                        if f1 % if inner window significantly brighter than
                        % outside window,
                        if w2 >= wmax % limit w2 so it is not too big
                            stop = true;
                        else

```

```

        w1 = w1 + 2; % increase window size, in case bright
        spot is larger
    end
else
    stop = true;
end
elseif numChannels == 3
if f1(1) == true || f1(2) == true || f1(3) == true % if
    inner window significantly brighter than outside window,
    if w2 >= wmax % limit w2 (maks is wmax) so it is not too
        big
        stop = true;
    else
        w1 = w1 + 2; % increase window size, in case bright
        spot is larger
    end
else
    stop = true;
end
end
% apply notch filter to (i,j) and its mirror (M-i,N-j)
for k = 1:numChannels
if f1(k) == true
    % calculate gaussian notch filter
    u = i-w2:i+w2;
    v = j-w2:j+w2;
    [V, U] = meshgrid(v, u);
    D = sqrt((U-i).^2 + (V-j).^2);
    H = 1 - exp(-0.5 * D.^2 / w2.^2);
    % apply notch filter to (i,j) and its mirror (M-i,N-j)
    Fout(i-w2:i+w2, j-w2:j+w2, k) = H .* Fout(i-w2:i+w2,
    j-w2:j+w2,k);
    Fout(M-(i+w2):M-(i-w2), N-(j+w2):N-(j-w2), k) = H .**
    Fout(M-(i+w2):M-(i-w2), N-(j+w2):N-(j-w2), k);
end
end
end
end
result = uint8(real(ifft2(ifftshift(Fout)))); % convert back to
proper image
end

```

Fungsi denoisePeriodic menerima sebuah citra  $f$  yang akan dihilangkan derau periodiknya. Citra tersebut kemudian ditransformasikan ke domain frekuensi dengan *Fourier transform* dan dilakukan fftshift. Kemudian, dilakukan pencarian titik-titik yang menjadi titik terang/*bright spot* (penyebab derau). Pencarian menggunakan dua jendela, yaitu  $w_1$  dan  $w_2$  dengan ukuran  $w_1$  lebih kecil daripada  $w_2$ . Kedua jendela akan bergerak memindai piksel-piksel pada citra domain frekuensi. Untuk optimisasi, pemindaian hanya

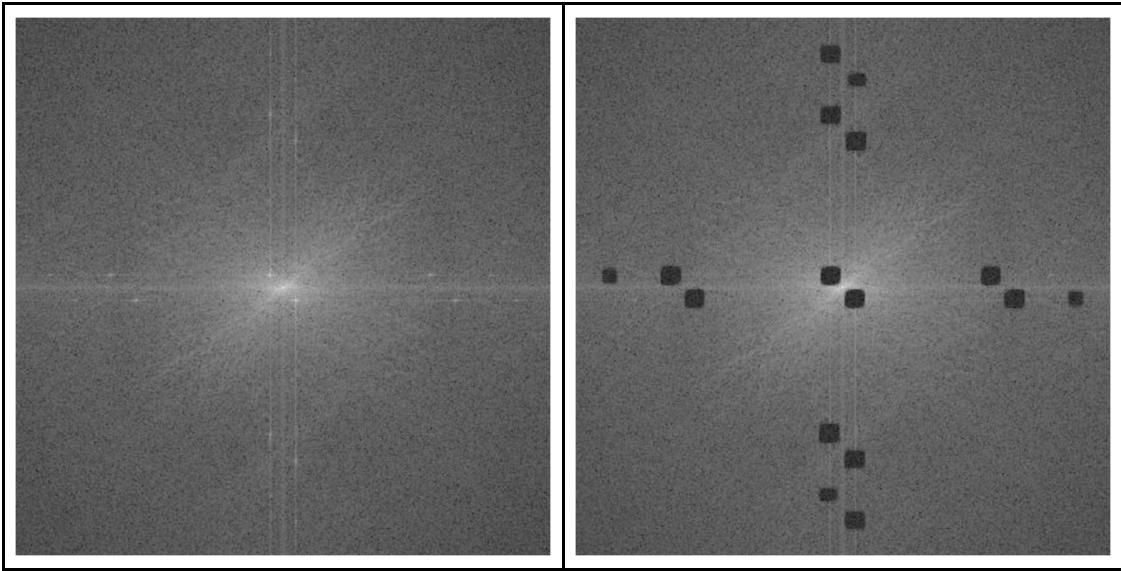
sampai setengah citra, setengah sisanya dapat disimpulkan dengan pencerminan. Suatu titik adalah titik terang jika rata-rata nilai  $di$  dalam jendela  $w2$  dan  $di$  luar  $w1$  yang memuat titik tersebut jauh lebih kecil daripada  $di$  dalam jendela  $w1$ . Selanjutnya, titik-titik terang tersebut dikalikan dengan *gaussian notch filter* seluas  $w2$  sehingga menghilangkan atau mengurangi derau periodik. Di fungsi di atas, juga terdapat mekanisme adaptif untuk mencari luas jendela  $w1$  dan  $w2$  yang optimal.

### 2.6.2. Testing

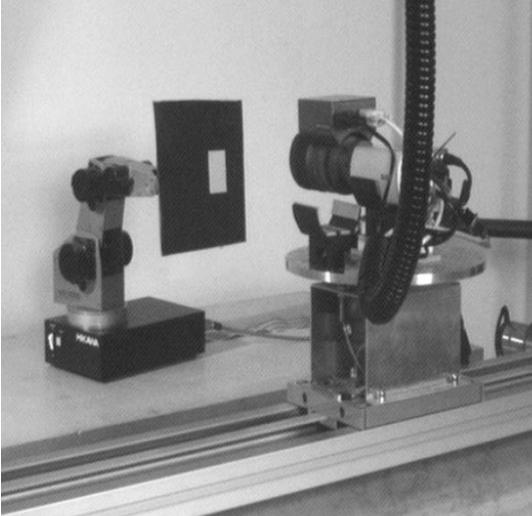
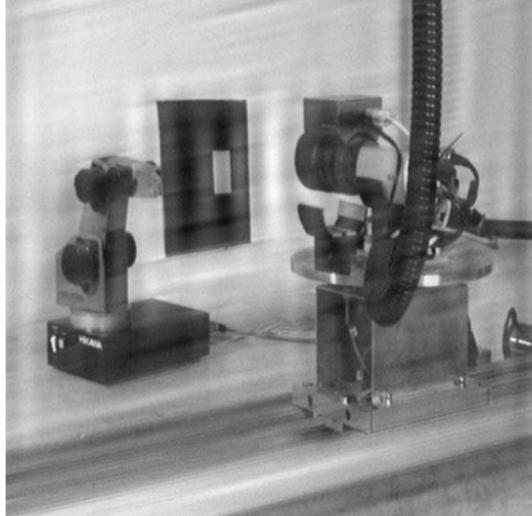
Berikut beberapa pengujian penghilangan derau periodik otomatis menggunakan program yang telah dibuat. Tiap tabel terdiri atas tiga baris dengan baris pertama berisi *header*, baris kedua berisi citra, dan baris ketiga berisi spektrum Fourier citra tersebut. Tiap baris terdiri atas dua kolom dengan kolom pertama adalah masukan dan kolom kedua adalah keluaran.

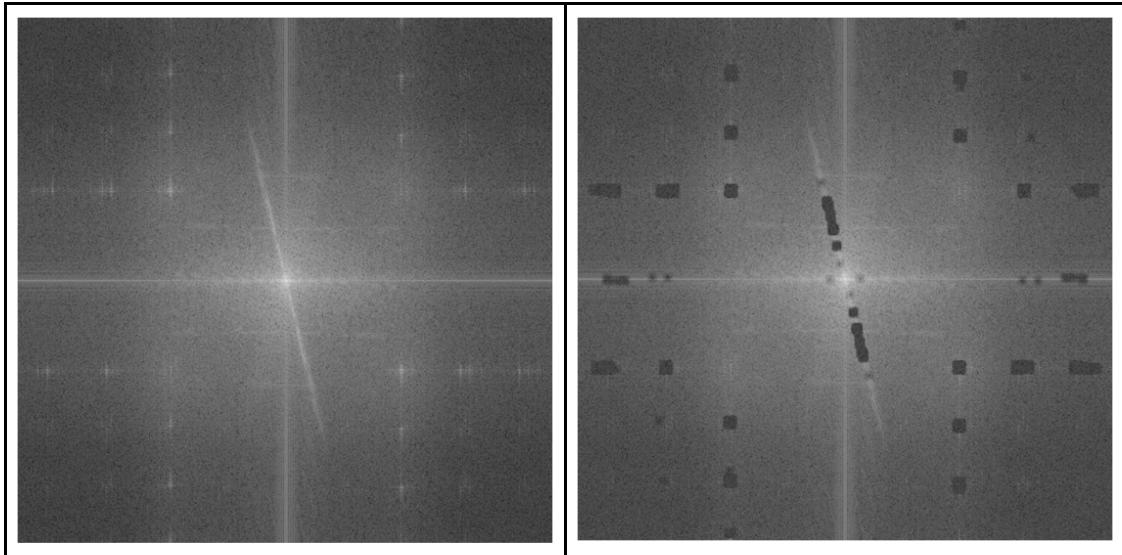
Tabel 2.6.1. Pengujian untuk citra a

Input	Output
	



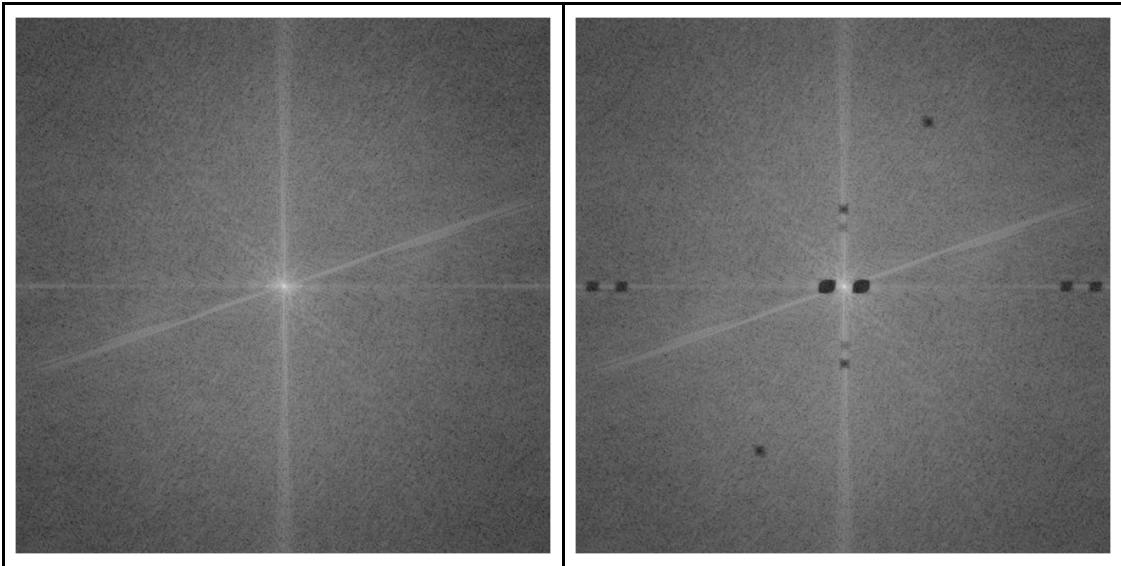
Tabel 2.6.2. Pengujian untuk citra b

Input	Output
	



Tabel 2.6.3. Pengujian untuk citra c

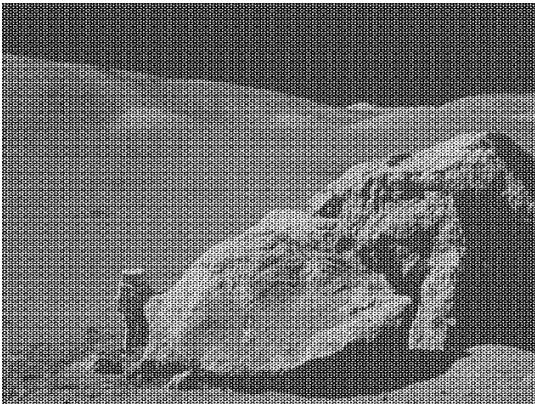
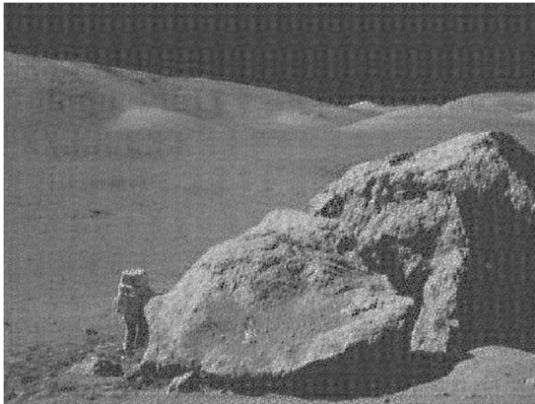
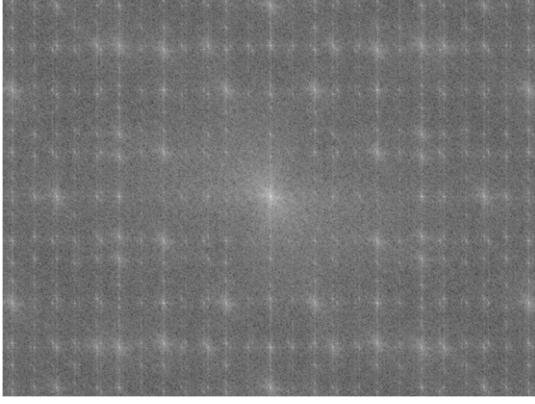
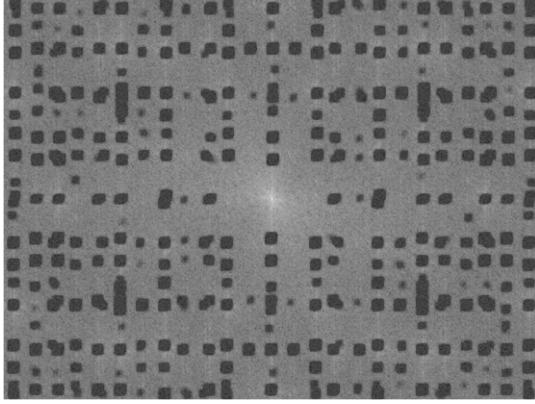
Input	Output
A grayscale photograph of a duck walking away from the camera on a paved sidewalk. The duck is dark-colored with a light-colored belly. The background shows some foliage and a curb.	A grayscale photograph of the same duck, showing improved contrast and clarity compared to the input. The details of the duck's feathers and the sidewalk are more distinct.



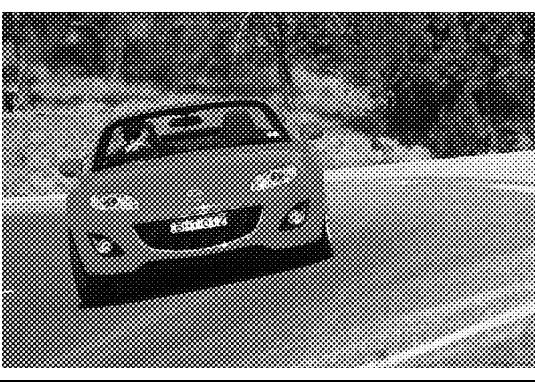
Tabel 2.6.4. Pengujian untuk citra d

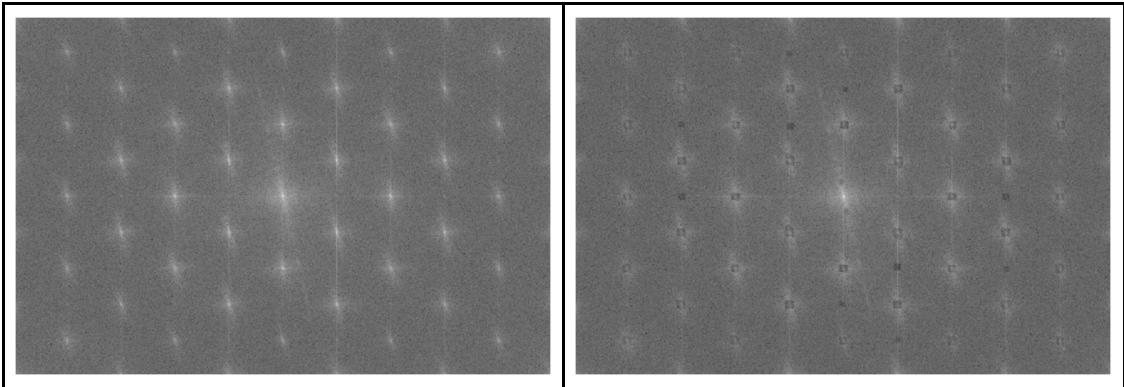
Input	Output
A black and white photograph of a large, ornate building with multiple towers and domes. The image is heavily corrupted by horizontal noise patterns, appearing as a series of thin, dark lines across the entire scene.	The same building as the input, but with significantly reduced horizontal noise. The image appears clearer, though it still contains vertical noise and some artifacts.
A grayscale image featuring a central bright starburst pattern with radiating lines, similar to the one in the top image. This version is set against a dark, textured background.	The starburst pattern from the input, but with a prominent cross-shaped artifact consisting of small black squares centered on the radiating lines. The background is also darker than the input.

Tabel 2.6.5. Pengujian untuk citra e

Input	Output
	
	

Tabel 2.6.6. Pengujian untuk citra f

Input	Output
	



Tabel 2.6.7. Pengujian untuk citra warna g

Input	Output
A photograph of a yellow rose flower with green leaves in the background. The image is heavily corrupted by horizontal noise, appearing as a series of vertical bands of different shades of gray across the entire frame.	The same yellow rose flower as the input, but with significantly reduced horizontal noise. The image appears much clearer and more vibrant, with the yellow petals and green leaves more distinct.
A dark gray background featuring a few small white stars of varying sizes. A single, very bright star is positioned near the center of the image.	The same dark background with stars, but with a regular grid of black squares overlaid. These squares are arranged in a grid pattern that covers most of the image area, obscuring the stars.

### 2.6.3. Analisis

Berdasarkan beberapa pengetesan yang telah dilakukan, program penapisan derau periodik secara adaptif dan otomatis dapat bekerja dengan baik. Hal ini ditunjukkan dengan titik-titik terang pada spektrum Fourier yang ada di citra masukan dapat dideteksi secara otomatis dan ditapis dengan *gaussian notch filter*. Dari hasil pengetesan, dapat disimpulkan terdapat beberapa kondisi citra hasil penapisan, yaitu:

- a. jauh lebih baik daripada citra masukan (citra e, f, g),
- b. cukup baik meskipun ada artifak yang cukup nampak (citra a, c, d),
- c. lebih buruk daripada citra masukan (citra b).

## 2.7. Penapis Wiener

Program untuk melakukan dekonvolusi pada citra yang memiliki *motion blurring* tersebut dengan penapis Wiener.

### 2.7.1. Kode Program

Berikut merupakan potongan kode program untuk penapis wiener

```
function deconvolved_image = penapisWiener(blurred_image, kernel,
noise_variance)
    % Menghitung spektrum Fourier dari kernel dan citra terblur
    H = double(fft2(double(kernel), size(blurred_image, 1),
    size(blurred_image, 2)));
    Y = double(fft2(double(blurred_image)));
    ConjH = conj(H);
    % Menghitung penapis Wiener
    wiener_filter = ConjH ./ ((ConjH .* H) + noise_variance);
    % Melakukan dekonvolusi menggunakan penapis Wiener
    F = wiener_filter .* Y;
    deconvolved_image = ifft2(F);
    % Mengonversi hasil dekonvolusi ke dalam rentang nilai yang benar
    deconvolved_image = uint8(deconvolved_image);
end
```

Function penapisWiener akan menghitung  $H(u,v)$  yaitu merupakan nilai fungsi degradasi. Nilai tersebut akan disimpan pada variabel “H”. Fungsi juga akan menghitung citra degradasi yang akan disimpan pada variabel “Y”. Setelah itu, akan menghitung  $H^*(u,v)$  yang merupakan complex conjugate dari  $H(u,v)$  dan disimpan dalam variabel “ConjH”. Berdasarkan variabel tersebut akan dihitung penapis wiener dengan noise variance yang diinginkan. Hasil penapis wiener tersebut akan dilakukan dekonvolusi

terhadap citra degradasi “Y”. Setelah itu, akan dilakukan Inverse Fast Fourier Transform untuk mengembalikan citra hasil dekonvolusi.

### 2.7.2. Testing

Berikut testing untuk menguji penapis wiener yang telah diimplementasikan

Tabel 2.7.2.1. Hasil pengujian dengan gambar 1

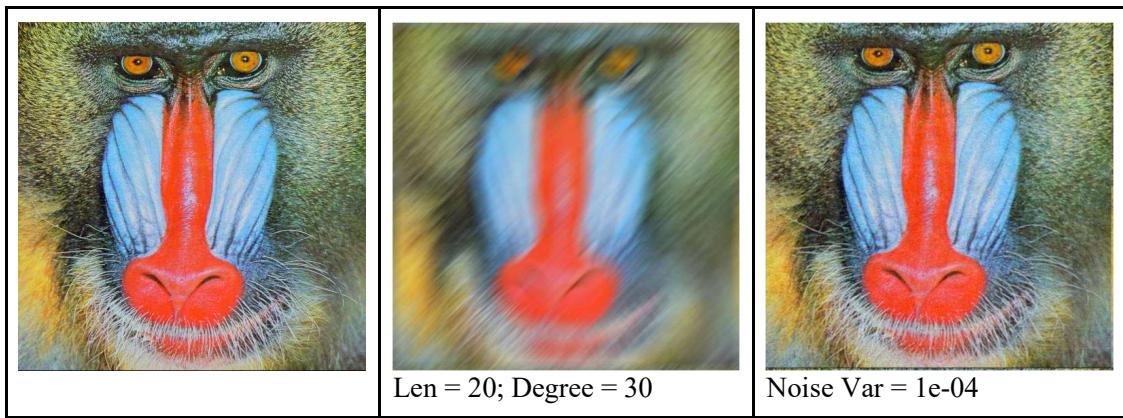
Input	Add Motion Blur	Output
	 Len = 10; Degree = 45	 Noise Var = 1e-05

Tabel 2.7.2.2. Hasil pengujian dengan gambar 2

Input	Add Motion Blur	Output
	 Len = 20; Degree = 30	 Noise Var = 1e-04

Tabel 2.7.2.3. Hasil pengujian dengan gambar 3

Input	Add Motion Blur	Output
-------	-----------------	--------

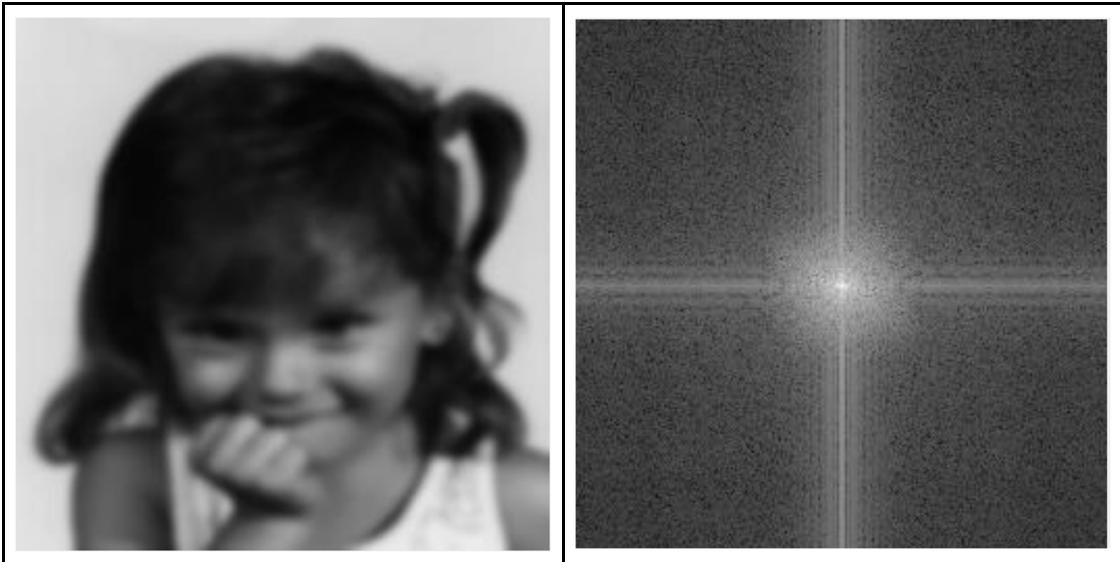


Tabel 2.7.2.4. Hasil pengujian dengan gambar 4

Input	Add Motion Blur	Output

Tabel 2.7.2.5. Hasil pengujian dengan gambar 5

Input	Spectrum
-------	----------



### 2.7.3. Analisis

Berdasarkan Percobaan yang dilakukan Fungsi Penapis Wiener sudah dapat menghilangkan *motion blurring* dengan baik. Pada Tabel 2.7.2.5 Citra memiliki spektrum frekuensi garis putih vertikal dan horizontal yang lebar, sehingga perlu penanganan untuk hal tersebut.

## **Alamat GitHub Program**

<https://github.com/lizardyy/Tugas-2-Citra>

### **Referensi**

PPT      IF4073      Interpretasi      dan      Pengolahan      Citra      2023/2024  
[\(https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2023-2024/citra23-24.htm \)](https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2023-2024/citra23-24.htm).

Justin Varghese, Saudia Subhash, Kamalraj Subramaniam, Kuttaiyur Palaniswamy Sridhar. 2020.  
“*Adaptive Gaussian notch filter for removing periodic noise from digital images*”.  
<https://doi.org/10.1049/iet-ipr.2018.5707>.