MODUL PRAKTIKUM COMPUTER VISION (IMAGE PROCESSING)

Disusun Oleh

Ir. H. Sumijan, M.Sc

LABORATORIUM MULTIMEDIA & PENGOLAHAN CITRA

Universitas Putra Indonesia "YPTK" Padang

BAB I
GRAPHICAL USER INTERFACE (GUI)

1.1 Pendahuluan

GUIDE atau GUI builder merupakan sebuah *graphical user interface* (GUI) yang dibangun dengan obyek grafik seperti tombol (button), kotak teks, slider, menu dan lain-lain. Aplikasi yang menggunakan GUI umumnya lebih mudah dipelajari dan digunakan karena orang yang menjalankannya tidak perlu mengetahui perintah yang ada dan bagaimana kerjanya.

Sampai saat ini, jika kita membicarakan pemrograman berorientasi visual, yangada di benak kita adalah sederetan bahasa pemrograman, seperti visual basic, Delphi, visual C++, visual Fox Pro, dan lainnya yang memang didesai secara khusus untuk itu. Matlab merintis ke arah pemrograman yang menggunakan GUI dimulai dari versi 5, versi 7 yang terus disempurnkan sampai sekarang Matlab 2011a.

GUI Matlab mempunyai kelebihan tersendiri dibandingkan dengan bahasa pemrogram lainnya, diantaranya:

1) GUI Matlab banyak digunakan dan cocok untuk aplikasi-aplikasi berorientasi sains, sehingga banyak peneliti dan mahasiswa menggunakan GUI Matlab untuk menyelesaikan riset atau tugas akhirnya.
2) GUI Matlab mempunyai fungsi built-in yang siap digunakan dan pemakai tidak perlu repot membuatnya sendiri.
3) Ukuran file, baik FIG-file maupun M-file, yang dihasilkan relatif kecil.
4) Kemampuan grafisnya cukup andal dan tidak kalah dibandingkan dengan bahasa pemrograman lainnya.

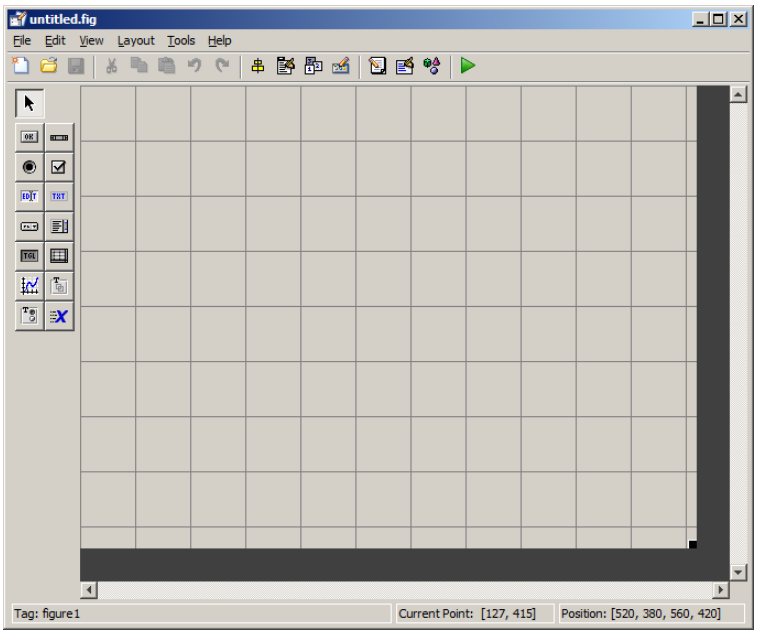
1.2 Memulai GUI Matlab

Memulai GUI Matlab dapat dilakukan dengan dua cara, yaitu:

1) Melalui command window matlab dengan mengetikkan : guide.
2) Pada menu bar > pilih file > new > GUI.

1.3 Tampilan GUI

Tampilan GUI ditunjukan pada gambar dibawah ini :



Gambar 1.1 Tampilan GUI

Tampilan GUI terdapat berbagai komponen yang terdiri dari beberapa uicontrol (kontrol user interface), seperti pada bahasa pemrograman visual lainnya, yaitu: pushbutton, togglebutton, radiobutton, chexkboxes, edit text, static text, slider, frames, listboxes, popup menu, dan axes. Kita dapat meletakkan semua kontrol pada layout editor dan selanjutnya hanya tinggal mengaturnya melalui property inspector.

Semua kontrol pada GUI dapat dimunculkan pada layout/figure dengan cara mendrag and drop kontrol yang diinginkan ke figure. Adapun penjelasan fungsi masing-masing kontrol pada tabel berikut:

Tabel 1.1 UIcontrol pada GUI Matlab

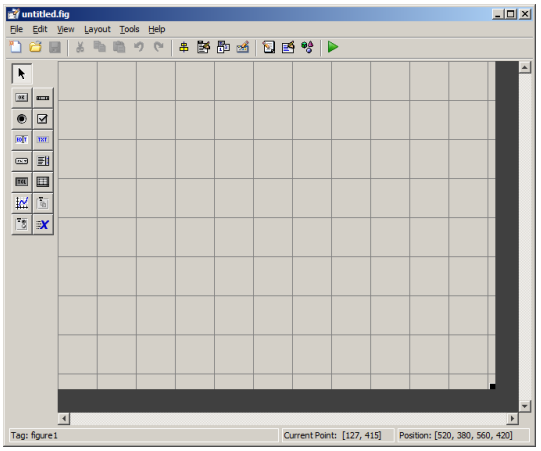
UIcontrol	Nama Uicontrol	Keterangan
	Select Tool	Untuk menyeleksi tools
	Push Button	Untuk membuat Button pada figure
	Radio Button	Untuk membuat button pilihan

	Edit Text	Memasukkan atau memodifikasi suatu text
	Pop-up Menu	Menampilkan daftar pilihan pada string Property
	Toogle Button	Membuat button pada figure
	Axes	Menampilkan sebuah grafik untuk gambar
	Button Group	Membuat button pada figure
	Slider	Membuat slider secara vertical dan horizontal
	Check Box	Membuat button pilihan mandiri
	Statistic Text	Untuk membuat atau menampilkan text
	List Box	Membuat list pada figure
	Tables	Untuk membuat table pada figure
	Panel	Membuat panel pada figure
	ActiveX Control	Mengaktifkan control lain

1.4 Membuat GUI

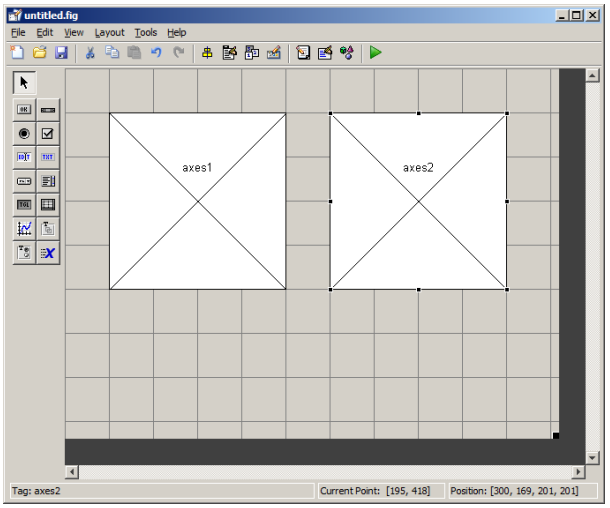
MATLAB mengimplementasikan GUI sebagai sebuah *figure* yang berisi barbagai *style* obyek UIControl. Selanjutnya kita harus memprogram masing-masing obyek agar dapat bekerja ketika diaktifkan oleh pemakai GUI. Sebuah contoh untuk membuat suatu pengolahan citra pada GUI Matlab :

1. Buka atau buatlah GUI baru pada matlab dengan melalui *commond windows* atau dengan lewat *menubar* pada matlab. Seperti pada gambar 1.2



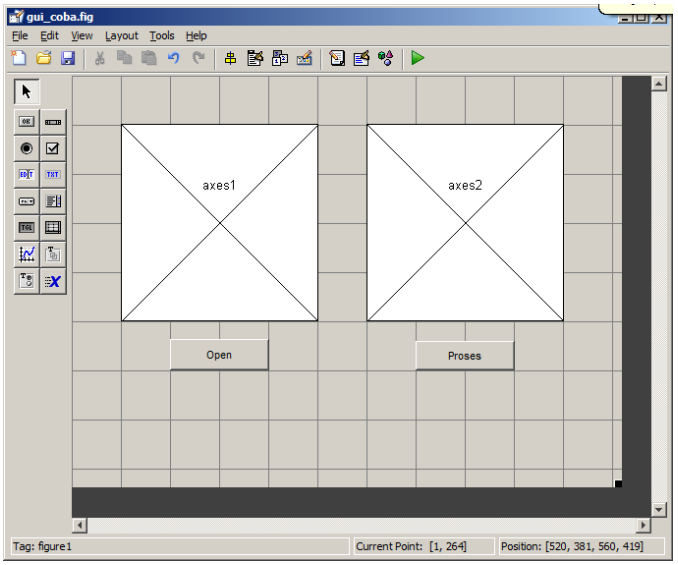
Gambar 1.2 Tampilan GUI baru.

2. Kemudian klik Uicontrol *axes*, buat dua Uicontrol *axes* pada *figure* GUI yang sudah tersedia. Seperti pada gambar 1.3



Gambar 1.3 tampilan dua *axes* pada *figure*

3. Buat 2 tombol *push button* pada *figure*, kemudian letakkan setiap *push button* di bawah *axes1* dan *axes2*. Beri nama 2 *push button* dengan nama **Open** untuk *button* di bawah *axes1* dan **Proses** untuk *push button* di bawah *axes2* dengan klik dua kali pada *push button* tersebut, kemudian pilih *String* untuk merubah nama pada *push button*. Seperti gambar 1.4 di bawah ini.

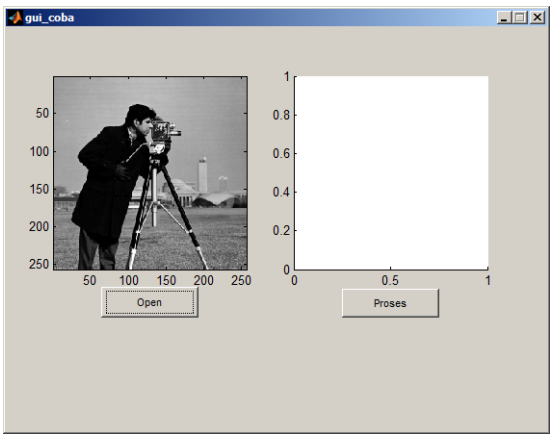


Gambar 1.4 Tampilan dengan *push button*

4. Kemudian klik kanan pada button Citra awal, kemudian pilih view callback > callback. Kemudian akan masuk pada *script* GUI dan masukkan *script* di bawah ini pada function pushbutton1 :

```
open=guidata(gcbo);
[namafile,direktori]=uigetfile({'*.jpg;*.bmp;*.tif'}, 'OpenImage');
I=imread(namafile);
set(open.figure1,'CurrentAxes',open.axes1);
set(imagesc(I));colormap('gray');
set(open.axes1,'Userdata',I);
```

maka akan terlihat seperti Gambar 1.5 dibawah ini

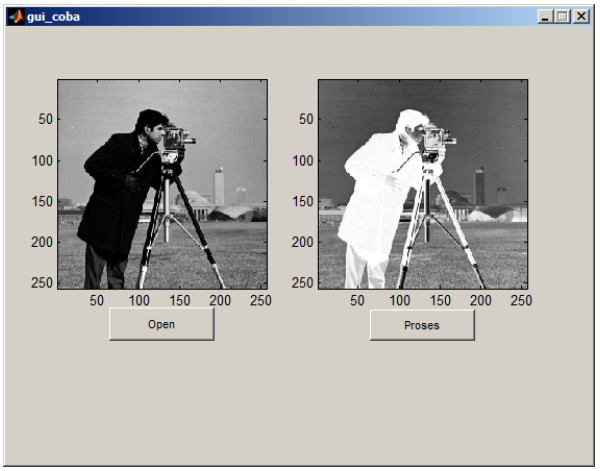


Gambar 1.5 Fungsi *button open*

5. Kemudian klik kanan pada *push button* **Proses** dan berikan *script* pada *button* tersebut dengan *script* dibawah ini:

```
open=guidata(gcbo);
I=get(open.axes1,'Userdata');
[r c] = size(I);
for x = 1 : r
 for y = 1 : c
 J(x,y) = 255 - I(x,y);
 end end
set(open.figure1,'CurrentAxes',open.axes2);
set(imagesc(J));colormap('gray');
set(open.axes2,'Userdata',J);
```

maka akan tampil seperti pada Gambar 1.6 dibawah ini



Gambar 1.6 Tampilan *button* proses

BAB II
OPERASI DASAR PENGOLAHAN CITRA

Citra digital direpresentasikan dengan matriks. Operasi pada citra digital pada dasarnya adalah memanipulasi elemen-elemen matriks.

2.1 Operasi Titik

2.1.1 Operasi Negatif

Operasi negatif bertujuan untuk mendapatkan citra negatif dengan cara mengurangi nilai intensitas piksel dari nilai keabuan maksimum. Secara umum persamaannya adalah sebagai berikut :

$$f(x,y)' = 255 - f(x,y)$$

Source code :

```
F = imread('cameraman.tif');
[r c]  = size(F);
for x = 1 : r
   for y = 1 : c
      G(x,y) = 255 - F(x,y);
   end
end
figure, imshow(F);
figure, imshow(G);
```

Gambar 2.1 Operasi negatif pada citra

2.1.2 Operasi Clipping

Yang dimaksud operasi clipping adalah operasi pemotongan jika nilai intensitas piksel hasil suatu operasi pengolahan citra terletak di bawah nilai intensitas minimum atau di atas nilai intensitas maksimum.

$$f(x,y)' = \begin{cases} 255 & f(x,y) > 255 \\ f(x,y), 0 \le & f(x,y) \le 255 \\ 0, & f(x,y) < 0 \end{cases}$$

Source code :

```
function J = clipping(I)
for x = 1 : size(I,1)
   for y = 1 : size(I,2)
      if I(x,y) > 255
         J(x,y) = 255;
      elseif I(x,y) < 0
         J(x,y) = 0;
      else
         J(x,y) = I(x,y);
      end
   end
end
```

2.2 Operasi Aritmatika

Karena Citra digital adalah matriks, maka operasi-operasi aritmatika matriks juga berlaku pada citra. Operasi matriks yang dilakukan adalah :

2.2.1 Penjumlahan Citra

Penjumlahan citra adalah operasi menjumlahkan dua matriks yang berukuran sama. Secara umum, persamaannya adalah sebagai berikut :

$$C(x,y) = A(x,y) + B(x,y)$$

C adalah citra baru yang intensitas tiap pikselnya adalah jumlah dari intensitas tiap piksel pada matriks A dan matriks B.

Source code :

```
A = double(imread('cameraman.tif'));
B = double(imread('rice.png'));
[r1 c1]  = size(A);
[r2 c2] = size(B);
if (r1 == r2) && (c1 == c2)
   for x = 1 : r1
      for y = 1 : c1
         C(x,y) = A(x,y) + B(x,y);
      end
   end
end
C = clipping(C);
figure, imshow(uint8(C));
```

Gambar 2.2 Operasi penjumlahan dua buah citra

Perlu diingat bahwa syarat penjumlahan dua buah matriks adalah ukuran kedua matriks harus sama. Jika hasil penjumlahan intensitas lebih besar dari 255, maka intensitasnya dibulatkan ke 255.

2.2.2 Pengurangan Citra

Pengurangan citra adalah operasi saling mengurangkan dua matriks yang berukuran sama. Secara umum, persamaannya adalah sebagai berikut :

$$C(x,y) = A(x,y) - B(x,y)$$

C adalah citra baru yang intensitas tiap pikselnya adalah selisih dari intensitas tiap piksel pada matriks A dan matriks B.

Source code :

```
A = double(imread('cameraman.tif'));
B = double(imread('rice.png'));
[r1 c1]  = size(A);
[r2 c2] = size(B);
if (r1 == r2) && (c1 == c2)
   for x = 1 : r1
      for y = 1 : c1
         C(x,y) = A(x,y) - B(x,y);
      end
   end
end
C = clipping(C);
figure, imshow(uint8(C));
```

Gambar 2.3 Operasi pengurangan dua buah citra

2.2.3 Perkalian Citra

Perkalian citra A dengan scalar c akan menghasilkan citra baru B yang intensitasnya lebih terang dari semula. Kenaikan intensitas setiap piksel sebanding dengan c. Operasi perkalian citra dengan scalar digunakan untuk kalibrasi kecerahan.

$$B(x,y) = A(x,y) * c$$

Source code :

```
A = double(imread('cameraman.tif'));
[r c]   = size(A);
   for x = 1 : r
      for y = 1 : c
         B(x,y) = A(x,y) .* 2;
      end
   end
B = clipping(B);
figure, imshow(uint8(B));
```

Gambar 2.4 Operasi perkalian citra dengan scalar

2.2.4 Pembagian citra

Pembagian citra A dengan scalar c akan menghasilkan citra baru B yang intensitasnya lebih gelap dari semula. Penurunan intensitas setiap piksel berbanding terbalik dengan c. Operasi pembagian citra dengan scalar digunakan untuk normalisasi kecerahan.

$$B(x,y) = A(x,y) / c$$

Source code :

```
A = double(imread('cameraman.tif'));
[r c]   = size(A);
   for x = 1 : r
      for y = 1 : c
         B(x,y) = A(x,y) ./ 2;
      end
   end
B = clipping(B);
figure, imshow(uint8(B));
```

Gambar 2.5 Operasi pembagian citra dengan scalar

2.3 Operasi Boolean

Selain operasi aritmatika, pemrosesan citra digital juga melibatkan operasi boolean (AND, OR, NOT, XOR).

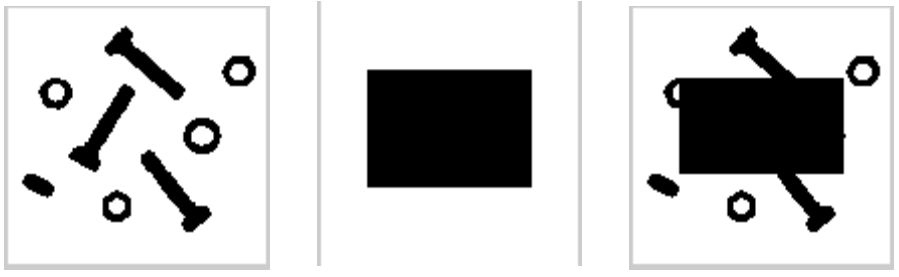
2.3.1 Operasi AND

Tabel 2.1 Tabel kebenaran logika AND

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

Source code :

```
A = not(imread('logika1.bmp'));
B = not(imread('logika2.bmp'));
[r1 c1]  = size(A);
[r2 c2] = size(B);
for x = 1 : r1
   for y = 1 : c1
      C(x,y) = and(A(x,y),B(x,y));
   end
end
figure, imshow(C);
```



Gambar 2.7 Operasi logika AND pada citra

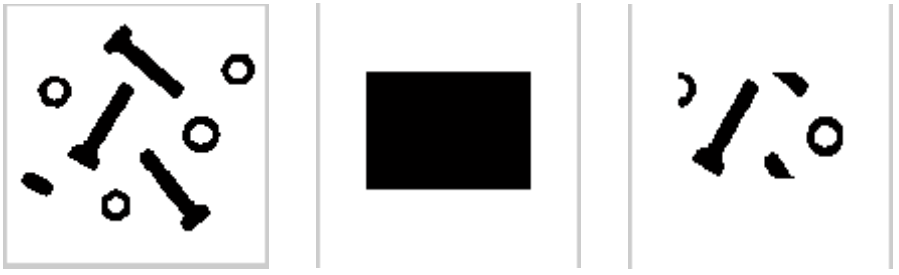
2.3.2 Operasi OR

Tabel 22 Tabel kebenaran logika OR

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

Source code :

```
A = not(imread('logika1.bmp'));
B = not(imread('logika2.bmp'));
[r1 c1] = size(A);
[r2 c2] = size(B);
for x = 1 : r1
   for y = 1 : c1
      C(x,y) = or(A(x,y),B(x,y));
   end
end
figure, imshow(C);
```



Gambar 2.8 Operasi logika OR pada citra

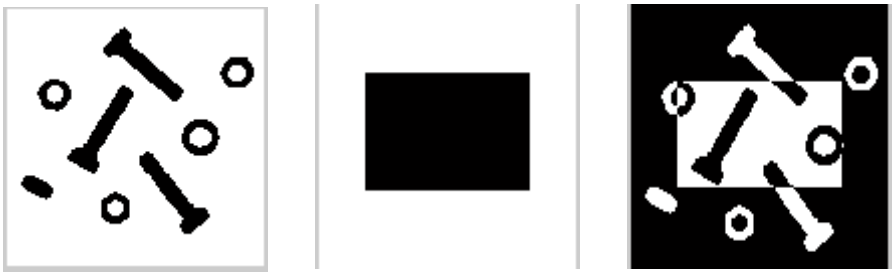
2.3.3 Operasi XOR

Tabel 2.3 Tabel kebenaran logika XOR

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Source code :

```
A = not(imread('logika1.bmp'));
B = not(imread('logika2.bmp'));
[r1 c1] = size(A);
[r2 c2] = size(B);
for x = 1 : r1
   for y = 1 : c1
      C(x,y) = xor(A(x,y),B(x,y));
   end
end
figure, imshow(C);
```



Gambar 2.9 Operasi logika XOR pada citra

2.4 Operasi Geometri

2.4.1 Operasi Translasi

Rumus translasi citra :

$$x' = x + Tx$$

$$y' = y + Ty$$

Yang dalam hal ini, Tx adalah besar pergeseran dalam arah x, sedangkan Ty adalah besar pergeseran dalam arah y.

Source code :

```
Tx = 20;
Ty = 50;
A = imread('cameraman.tif');
[brs kol] = size(A);
for x = 1 : brs
   for y = 1 : kol
      B(x+Tx, y+Ty) = A(x,y);
   end
end
B = B(1:brs,1:kol);
figure, imshow(uint8(A));
figure, imshow(uint8(B));
```

Gambar 2.10 Operasi translasi pada citra

2.4.2 Operasi Cropping

Rumus operasi cropping pada citra :

$$w' = xR - xL$$

$$h' = yB - yT$$

Yang dalam hal ini, w' adalah lebar citra baru yang diperoleh setelah proses cropping. Sedangkan h' adalah tinggi citra baru. xR dan xL adalah dua titik disebelah kiri dan kanan pada arah sumbu x. yB dan yT adalah dua titik disebelah atas dan bawah pada arah sumbu y. Keempat titik xR, xL, yB, yT akan digunakan sebagai koordinat-koordinat dimana citra akan dipotong.

Source code :

```
I = imread('cameraman.tif');
[brs kol] = size(I);
x1  = 50;
x2 = 200;
y1 = 50;
y2 = 200;
I(1:x1,:)     = 0;
I(x2:brs,:) = 0;
I(:,1:y1)     = 0;
I(:,y2:kol) = 0;
imshow(I);
```

Gambar 2.11 Operasi cropping pada citra

2.4 3 Operasi Flipping

Flipping adalah operasi geometri yang sama dengan pencerminan. Ada dua macam flipping : horizontal dan vertical.

Flipping vertikal adalah pencerminan pada sumbu-X dari citra A menjadi citra B, yang diberikan oleh :

$$x' = 2xc - x$$

$$y' = y$$

Source code :

```
I = imread('cameraman.tif');
[brs kol] = size(I);
J = repmat(0,brs,kol);
mirror = floor(kol/2);
for x = 1 : brs-1
   for y = 1 : kol-1
      J(x,y) = I((2*mirror)-x, y);
   end
end
figure, imshow(uint8(J));
```

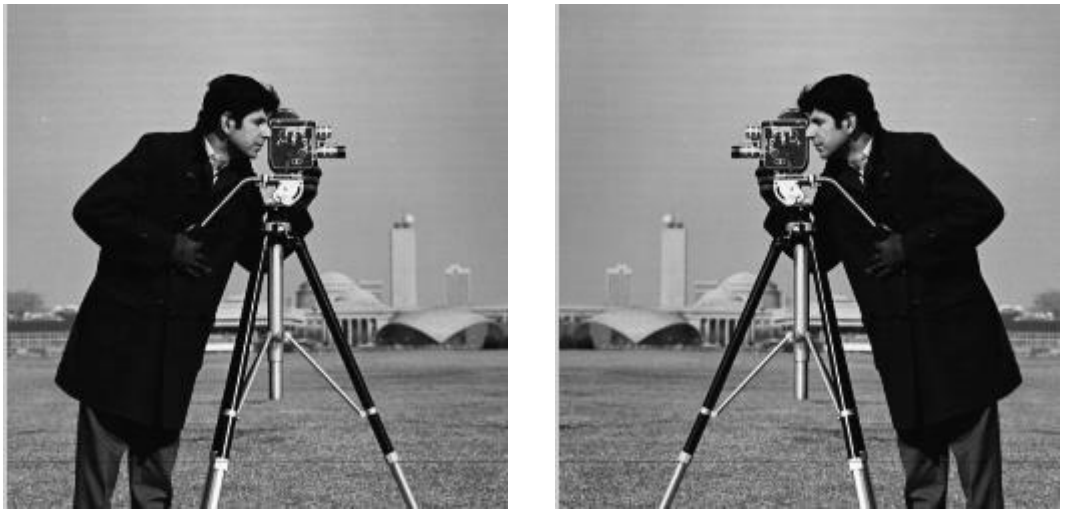
Gambar 2.12 Operasi flipping vertikal pada citra

Flipping horizontal adalah pencerminan pada sumbu-Y dari citra A menjadi citra B, yang diberikan oleh :

$$x' = x$$
$$y' = 2yc - y$$

Source code :

```
I = imread('cameraman.tif');
[brs kol] = size(I);
J = repmat(0,brs,kol);
mirror = floor(brs/2);
for x = 1 : brs-1
   for y = 1 : kol-1
      J(x,y) = I(x, (2*mirror)-y);
   end
end
figure, imshow(uint8(J));
```



Gambar 2.13 Operasi flipping horizontal pada citra

2.4.4 Operasi Rotasi

Rumus rotasi pada citra :

$$x' = Xp + (x - Xp) * \cos\theta - (y - Yp) * \sin\theta$$
$$y' = Yp + (x - Xp) * \sin\theta + (y - Yp) * \cos\theta$$

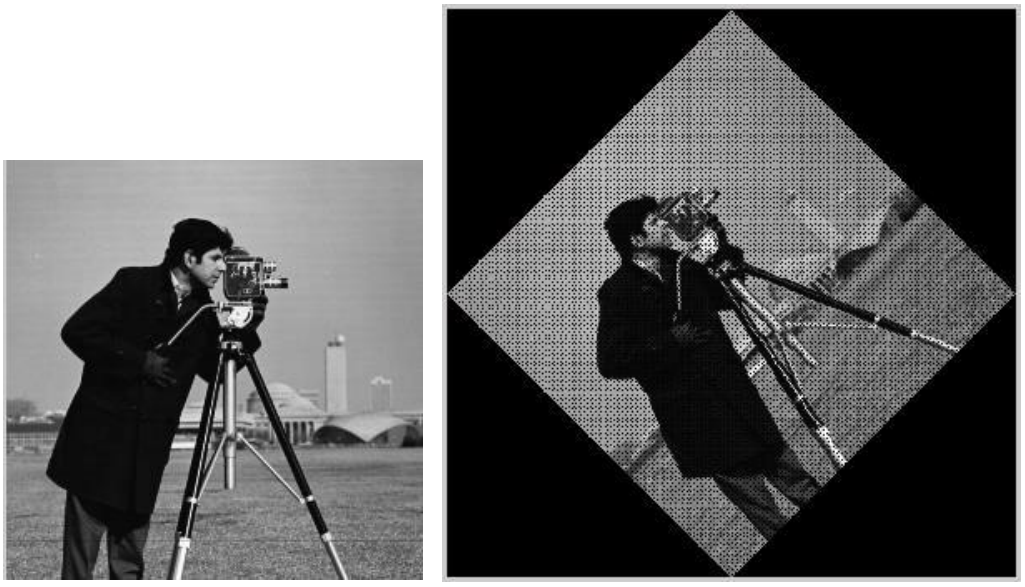
Yang dalam hal ini, Xp dan Yp merupakan titik pivot rotasi. Pivot yang dimaksud adalah koordinat titik tengah dari citra.

Source code :

```
function J = rotasi(I,T)
m = size(I,1);
n  = size(I,2);
if rem(m,2) == 0,   Xp = floor((m+1)/2)+1;
else                Xp = floor((m+1)/2);
end
if rem(n,2) == 0,   Yp = floor((n+1)/2)+1;
else                Yp = floor((n+1)/2);
end
X = zeros(m,n);
Y = zeros(m,n);
for x = 1 : m,    X(x,1:n) = x;    end
for y = 1 : n,    Y(1:m,y) = y;    end
Xa = round(Xp + (X - Xp)*cosd(T) - (Y - Yp)*sind(T));
Ya = round(Yp + (X - Xp)*sind(T)  + (Y - Yp)*cosd(T));
r = size(min(Xa(:)) : max(Xa(:)),2);
c = size(min(Ya(:)) : max(Ya(:)),2);
xs = round(abs(r-m)/2);
ys = round(abs(c-n)/2);
J = zeros(r,c);
for x = 1 : m
   for y = 1 : n
      J(Xa(x,y)+xs,Ya(x,y)+ys) = I(x,y);
   end
end
```

Source code di atas disimpan dengan nama m-file '*rotasi*.m'. Selanjutnya untuk menguji keberhasilan source code di atas, buatlah suatu m-file lagi dan tuliskan source code di bawah ini

```
clear all; clc;
T = 45;
I = imread('cameraman.tif'); J = rotasi(I,T);
imshow(uint8(J),'initialmagnification','fit');
```



Gambar 2.14 Operasi rotasi pada citra

2.4.5 Operasi Scalling

Penskalaan citra, disebut juga scalling, yaitu pengubahan ukuran citra. Rumus penskalaan citra :

$$x' = ShX * x$$

$$y' = ShY * y$$

Dalam hal ini, ShX dan ShY adalah factor skala masing-masing dalam arah x dan arah y pada koordinat citra.

Source code :

```
function J = perbesar(I,ShX,ShY)
m  =  size(I,1);
n  =  size(I,2);
r = m*ShX;
c = n*ShY;
J = zeros(r,c);
for x = 1 : m
   for y = 1 : n
      J((x-1)*ShX+1 : x*ShX, (y-1)*ShY+1 : y*ShY) = I(x,y);
   end
end
```

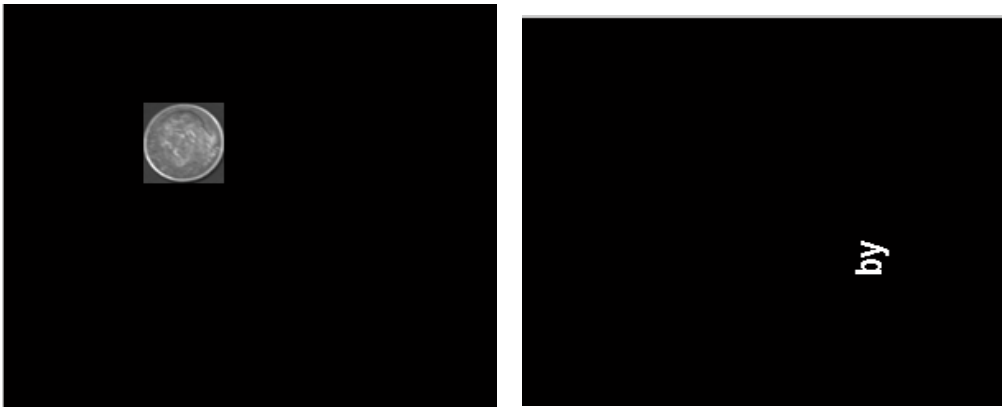
Source code di atas disimpan dengan nama m-file '*perbesar*.m'. Selanjutnya untuk menguji keberhasilan source code di atas, buatlah suatu m-file lagi dan tuliskan source code di bawah ini

```
clear all; clc;
I = imread('cameraman.tif');
ShX =  2;
ShY = 1;
J = perbesar(I,ShX,ShY);
imshow(uint8(J));
```

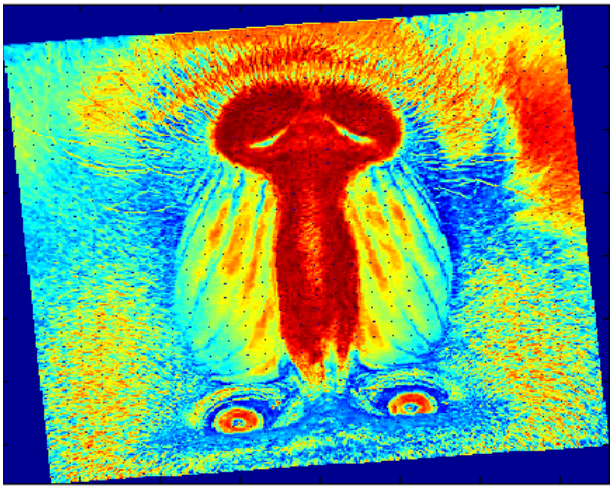
Gambar 2.15 Operasi scalling pada citra

Tugas..!!!

a. Cropping



b. Rotasi



2.5 Konvolusi

Konvolusi merupakan operasi yang mendasar dalam pengolahan citra. Konvolusi dua buah fungsi *f(x)* dan *g(x)* didefinisikan sebagai berikut :

$$h(x) = f(x) * g(x) = \int_{-\infty}^{\infty} f(a)g(x-a)da$$

Yang dalam hal ini, tanda * menyatakan operator konvolusi, dan peubah (variabel) *a* adalah peubah bantu. Untuk fungsi diskrit, konvolusi didefinisikan sebagai :

$$h(x) = f(x) * g(x) = \sum_{a=-\infty}^{\infty} f(a)g(x-a)$$

Pada operasi konvolusi di atas, *g(x)* disebut kernel konvolusi atau kernel penapis (filter). Kernel *g(x)* merupakan suatu jendela yang dioperasikan secara bergeser pada sinyal masukan *f(x),* yang dalam hal ini, jumlah perkalian kedua fungsi pada setiap titik merupakan hasil konvolusi yang dinyatakan dengan keluaran *h(x).*

Untuk fungsi dua peubah (fungsi dua dimensi atau dwimatra), operasi konvolusi didefinisikan sebagai berikut :

a) Untuk fungsi malar

$$h(x,y) = f(x,y) * g(x,y) = \int_{-\infty-\infty}^{\infty}\int^{\infty} f(a,b)g(x-a,y-b)dadb$$

b) Untuk fungsi diskrit

$$h(x,y) = f(x,y) * g(x,y) = \sum_{a=-\infty}^{\infty}\sum_{b=-\infty}^{\infty} f(a,b)g(x-a,y-b)$$

Fungsi penapis *g(x,y)* disebut juga *convolution filter*, *convolution mask*, *convolution kernel*, atau *template*. Di dalam MATLAB, filter linear diimplementasikan melalui konvolusi dua dimensi. Dalam konvolusi, nilai dari sebuah piksel keluaran dihitung dengan mengalikan elemen-elemen pada dua buah matriks dan menjumlahkan hasilnya. Satu dari matriks tersebut merepresentasikan citra, sementara matriks yang lainnya adalah filter-nya.

2.5.1 Algoritma

Ukuran matriks kernel biasanya lebih kecil dari ukuran matriks citra. Sebagai contoh misalkan terdapat sebuah matriks citra *I* dan sebuah kernel *k* sebagai berikut :

$$\begin{bmatrix} -4 & 0 & 2 & 3 & 5 & -2 & -3 \\ -4 & 2 & 0 & -1 & 3 & 0 & 0 \\ 1 & 1 & 3 & 2 & 3 & 3 & 4 \\ 2 & 5 & 5 & 3 & 2 & 0 & 5 \\ -4 & 4 & 3 & 5 & -4 & -2 & -4 \end{bmatrix} \qquad \begin{bmatrix} 4 & -3 & 1 \\ 4 & 6 & 2 \end{bmatrix}$$

Matriks *I* Kernel *k*

Algoritma atau langkah-langkah untuk melakukan konvolusi terhadap matriks *I* adalah sebagai berikut :

- Memutar *convolution kernel* 180 derajat untuk menghasilkan sebuah *computational molecule*

$$\begin{bmatrix} 4 & -3 & 1 \\ 4 & 6 & 2 \end{bmatrix} \xrightarrow{\text{Rotasi } 180^{O}} \begin{bmatrix} 2 & 6 & 4 \\ 1 & -3 & 4 \end{bmatrix}$$

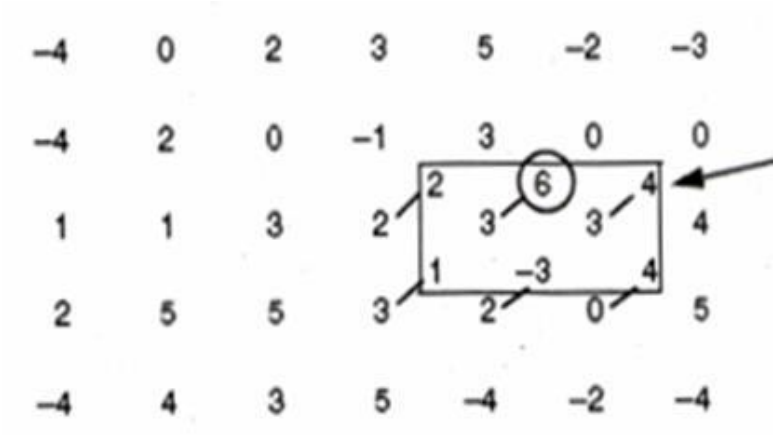
- Menentukan piksel tengah dari *computational molecule.*

  Untuk menggunakan *computational molecule*, kita pertama-tama harus menentukan piksel tengah. Piksel tengah didefinisikan sebagai *floor((size(h)+1)/2)*. Sebagai contoh dalam sebuah molekul berukuran 2×3, maka piksel tengah adalah (1, 2).

$$\begin{bmatrix} 2 & 6 & 4 \\ 1 & -3 & 4 \end{bmatrix}$$

- Menerapkan computational molecule pada setiap piksel citra masukan.

  Nilai dari piksel yang ditentukan dalam matriks keluaran B dihitung dengan menerapkan *computational molecule h* dengan piksel yang berkorespondensi pada matriks *I*. Kita dapat menggambarkan hal ini dengan menumpuk *h* pada *I*, dengan piksel tengah *h* menimpa piksel yang akan dihitung pada *I*. Kemudian kita mengalikan setiap elemen dari *h* dengan piksel pada *I* dan kemudian menjumlahkan hasilnya.



Gambar 2.14 Ilustrasi proses konvolusi pada elemen piksel (3,5)

Sebagai contoh kita akan menentukan nilai dari elemen piksel (3,5) pada matriks *B*, dengan cara menimpa *h* pada matriks *I* dengan piksel tengah dari *h* menutupi elemen piksel (3,5) dari matriks *I*. Piksel tengah diberi tanda lingkaran pada gambar.

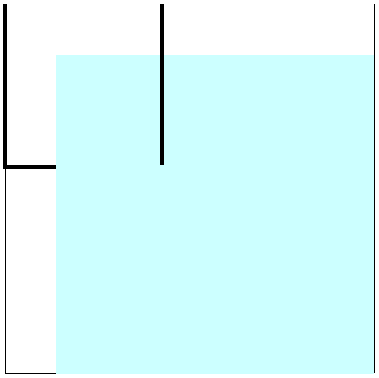
Terlihat ada enam piksel yang tertutupi oleh *h*, untuk setiap piksel-piksel ini, kalikan nilai dari piksel tersebut dengan nilai pada *h*. Jumlahkan hasilnya, lalu letakkan penjumlahan ini pada B(3,5).

$$B(3,5)=2*2 + 3*6 + 3*4 + 3*1 + 2*-3 + 0*4 =31$$

Lakukan hal tersebut diatas untuk setiap piksel pada matriks *I* untuk menghasilkan setiap piksel yang berkorespondensi pada matriks *B*.

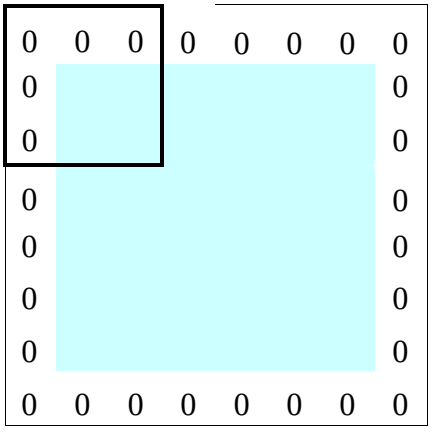
2.5.2 Proses Padding pada Batas Citra

Ketika kita menerapkan filter pada piksel-piksel yang berada pada pinggir sebuah citra, beberapa elemen dari *computational molecule* tidak dapat menimpa piksel citra. Sebagai contoh jika molekul berukuran 3×3 dan kita sedang menghitung untuk piksel pada ujung kiri atas dari citra, beberapa elemen pada molekul berada diluar batas citra, seperti terlihat pada gambar 1.2 berikut ini.



Gambar 2.15 Permasalahan *padding* pada batas citra

Untuk menghitung kondisi seperti di atas, proses konvolusi menerapkan suatu metode yang disebut dengan *zero padding*. *Zero padding* mengasumsikan bahwa piksel-piksel yang tidak dapat tertimpa oleh *computational molecule* bernilai nol.



Gambar 2.16 Ilustrasi penerapan *zero padding* pada proses konvolusi

Contoh perintah untuk melakukan konvolusi terhadap matriks *I* yang berukuran 5×5 dan kernel yang berukuran 3×3 adalah sebagai berikut :

```
function B = convolve(A, k);
[r c] = size(A);
[m n] = size(k);
h = rot90(k, 2);
```

```
center = floor((size(h)+1)/2);
left = center(2) - 1;
right = n - center(2);
top = center(1) - 1;
bottom = m - center(1);
Rep = zeros(r + top + bottom, c + left + right);
for x = 1 + top : r + top
   for y = 1 + left : c + left
      Rep(x,y) = A(x - top, y - left);
   end
end
B = zeros(r , c);
for x = 1 : r
   for y = 1  : c
      for i = 1 : m
         for j = 1 : n
            q = x - 1;
            w = y -1;
            B(x, y) = B(x, y) + (Rep(i + q, j + w) * h(i, j));
         end
      end
   end
end
```

Source code di atas disimpan dengan nama m-file 'convolve.m'. Selanjutnya untuk menguji keberhasilan source code di atas, buatlah suatu m-file lagi dan tuliskan source code di bawah ini :

```
clear; clc;
```

```
I = [4 4 3 5 4;
     6 6 5 5 2;
     5 6 6 6 2;
     6 7 5 5 3;
     3 5 2 4 4];
k = [0 -1 0;
     -1 4 -1;
     0 -1 0];
Hsl = convolve(I, k)
```

Operasi konvolusi yang dilakukan adalah dengan melibatkan *zero padding* di dalamnya. Output yang dihasilkan oleh source code di atas adalah sebagai berikut :

```
Hsl =
    6    3   -2    8    9
    9    3    0    2   -3
    2    0    2    6   -3
    9    6    0    2    1
    1    8   -6    5    9
```

Contoh perintah untuk melakukan konvolusi pada citra ‘cameraman.tif’ adalah sebagai berikut :

```
clear; clc;
I = imread('cameraman.tif');
k = ones(3)/9;
Hsl = convolve(I, k);
imshow(I);
figure, imshow(uint8(Hsl));
```

(a) (b)

Gambar 2.17 *(*a) Citra sebelum dilakukan konvolusi, (b) Hasil konvolusi dengan kernel 3×3

2.6 Segmentasi Citra

2.6.1 Operasi Pengambangan

Operasi pengambangan (thresholding) adalah operasi memetakan nilai intensitas piksel ke salah satu dari dua nilai, a1 atau a2, berdasarkan nilai ambang (threshold) T

$$f(x,y)' = \begin{cases} a1, & f(x,y) < T \\ a2, & f(x,y) \geq T \end{cases}$$

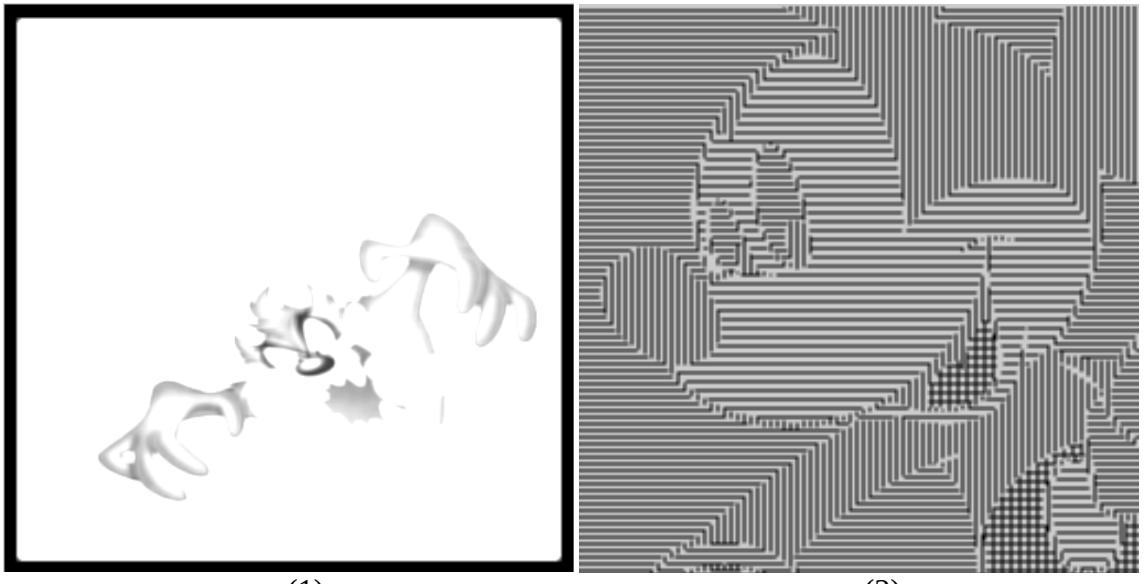
Souce code :

```
F = imread('cameraman.tif');
[r c]  = size(F);
T = 128;
for x = 1 : r
   for y = 1 : c
      if F(x,y) >= T
         G(x,y) = 255;
      else
         G(x,y) = 0;
      end
   end
```

```
end
figure, imshow(F);
figure, imshow(G);
```

Gambar 2.18 Operasi pengambangan pada citra

Tugas...!!!!



(1) (2)

(3)

BAB III FILTER SPASIAL

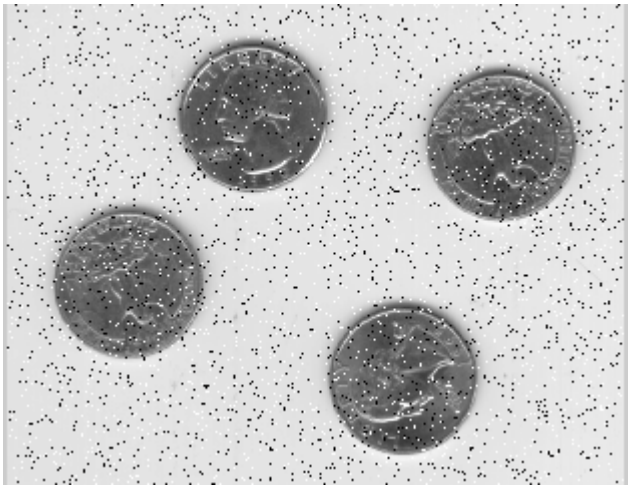
Pentapisan pada pengolahan citra biasa disebut dengan pentapisan spasial (*spasial filtering*). Pada proses pentapisan, nilai piksel baru umumnya dihitung berdasarkan piksel tetangga (*neighborhood*).

Proses-proses yang termasuk ke dalam filter spasial citra adalah pelembutan citra (*image smoothing*) dan penajaman citra (*image sharphening*).

3.1 Pelembutan Citra *(Image Smoothing)*

Pelembutan citra bertujuan untuk menekan gangguan (*noise*) pada citra. Gangguan tersebut biasanya muncul sebagai akibat dari hasil penerokan yang tidak bagus. Gangguan pada citra umumnya berupa variasi intensitas suatu piksel yang tidak berkorelasi dengan piksel-piksel tetangganya. Secara visual, gangguan mudah dilihat oleh mata karena tampak berbeda dengan piksel tetangganya.

Piksel yang mengalami gangguan umumnya memiliki frekuensi tinggi. Komponen citra yang berfrekuensi rendah umumnya mempunyai nilai piksel konstan atau berubah sangat lambat. Operasi pelembutan citra dilakukan untuk menekan komponen berfrekuensi tinggi dan meloloskan komponen berfrekuensi rendah.



Gambar 3.1 Citra yang mengalami gangguan (noise)

3.2 Filter Linier *(Low Pass Filter)*

Pada prinsipnya, filter yang digunakan dalam filter linear adalah *neighborhood averaging* merupakan salah satu jenis *low-pass filter*, yang bekerja dengan cara mengganti nilai suatu piksel pada citra asal dengan nilai rata-rata dari piksel tersebut dan lingkungan tetangganya.

Penapis rata-rata adalah salah satu penapis lolos rendah yang paling sederhana. Aturan untuk penapis lolos rendah adalah :

- Semua koefisien penapis harus positif
- Jumlah semua koefisian harus sama dengan satu

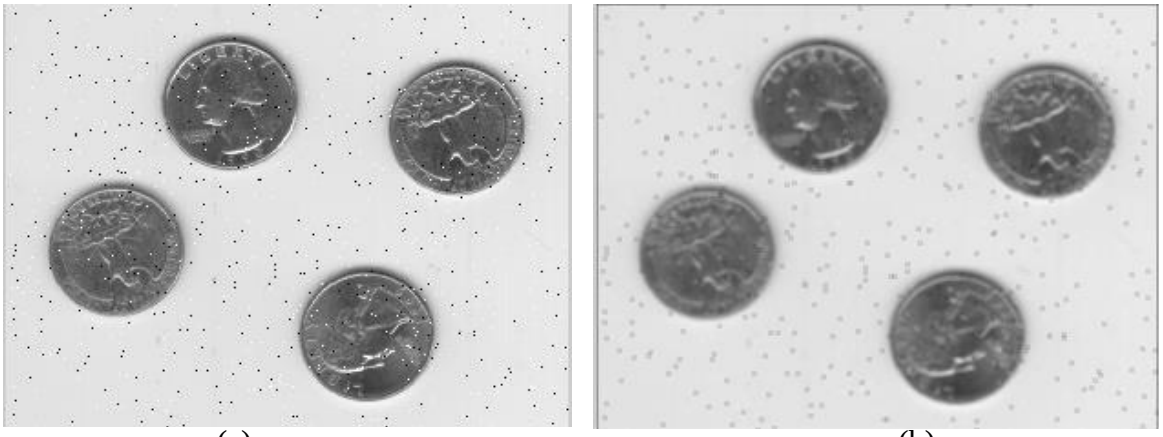
Berikut beberapa penapis rata-rata, yaitu penapis lolos rendah yang sering digunakan pada operasi pelembutan.

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \quad \begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix} \quad \begin{bmatrix} 1/10 & 1/10 & 1/10 \\ 1/10 & 1/5 & 1/10 \\ 1/10 & 1/10 & 1/10 \end{bmatrix}$$

Contoh perintah untuk melakukan *neighborhood averaging* dengan kernel berukuran 3×3:

```
clear; clc;
A = imread('eight.tif');
A = imnoise(A, 'salt & pepper', 0.01);
k = ones(3) / 9;
[r c] = size(A);
[m n] = size(k);
h = rot90(k, 2);
center = floor((size(h)+1)/2);
left = center(2) - 1;
right = n - center(2);
top = center(1) - 1;
bottom = m - center(1);
Rep = zeros(r + top + bottom, c + left + right);
```

```
for x = 1 + top : r + top
   for y = 1 + left : c + left
      Rep(x,y) = A(x - top, y - left);
   end
end
B = zeros(r , c);
for x = 1 : r
   for y = 1  : c
      for i = 1 : m
         for j = 1 : n
            q = x - 1;
            w = y -1;
            B(x, y) = B(x, y) + (Rep(i + q, j + w) * h(i, j));
         end
      end
   end
end
figure, imshow(A);
figure, imshow(uint8(B));
```



(a) (b)

Gambar 3.2 (a) Citra yang mengandung noise, (b) Hasil pelembutan dengan kernel 3×3

3.3 Filter Linier *(High Pass Filter)*

3.31 Penajaman Citra *(Image Sharpening)*

Inti dari penajaman (*sharpening*) citra adalah memperjelas tepi pada objek di dalam citra. Penajaman citra merupakan kebalikan dari operasi pelembutan citra karena operasi ini menghilangkan bagian citra yang lembut. Metode atau *filtering* yang digunakan adalah *high-pass filtering*.

Operasi penajaman dilakukan dengan melewatkan citra pada penapis lolos tinggi (*high-pass filter*). Penapis lolos tinggi akan meloloskan (atau memperkuat) komponen yang berfrekuensi tinggi (misalnya tepi atau pinggiran objek) dan akan menurunkan komponen berfrekuensi rendah. Akibatnya, pinggiran objek terlihat lebih tajam dibandingkan sekitarnya.

Karena penajaman citra lebih berpengaruh pada tepi *(edge*) objek, maka penajaman citra sering disebut juga penajaman tepi (*edge sharpening*) atau peningkatan kualitas tepi (*edge enhancement*).

(a) (b)

Gambar 3.3 (a) Citra sebelum dikenai operasi penajaman, (b) Citra setelah dikenai operasi penajaman

Penapis pada operasi penajaman citra disebut penapis lolos tinggi. Aturan dari penapis lolos tinggi adalah sebagai berikut:

- Koefisien boleh positif, negatif, atau nol
- Jumlah semua koefisien adalah satu

Berikut contoh-contoh penapis lolos tinggi yang sering digunakan dalam penajaman citra :

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Contoh perintah untuk melakukan operasi penajaman terhadap suatu citra adalah sebagai berikut :

```
clear; clc;
I = imread('cameraman.tif');
hpf =[-1 -1 -1; -1 9 -1; -1 -1 -1];
[r c] = size(I);
[m n] = size(hpf);
h = rot90(hpf, 2);
center = floor((size(h)+1)/2);
left = center(2) - 1;
right = n - center(2);
top = center(1) - 1;
bottom = m - center(1);
Rep = zeros(r + top + bottom, c + left + right);
for x = 1 + top : r + top
   for y = 1 + left : c + left
      Rep(x,y) = I(x - top, y - left);
   end
```

```
end
B = zeros(r , c);
for x = 1 : r
   for y = 1  : c
      for i = 1 : m
         for j = 1 : n
            q = x - 1;
            w = y -1;
            B(x, y) = B(x, y) + (Rep(i + q, j + w) * h(i, j));
         end
      end
   end
end
figure, imshow(I);
figure, imshow(uint8(B));
```

(a) (b)

Gambar 3.4 (a) Citra ‘cameraman.tif’ sebelum dikenai operasi penajaman, (b) Citra setelah dikenai operasi penajaman

3.4 Filter Non-Linier

3.4.1 Filter Median

Filter median sangat bermanfaat untuk menghilangkan *outliers*, yaitu nilai-nilai piksel yang ekstrim. Filter median menggunakan *sliding neighborhood* untuk memproses suatu citra, yaitu suatu operasi dimana filter ini akan menentukan nilai masing-masing piksel keluaran dengan memeriksa tetangga m×n di sekitar piksel masukan yang bersangkutan. Filter median mengatur nilai-nilai piksel dalam satu tetangga dan memilih nilai tengah atau median sebagai hasil.

Median filter merupakan salah satu jenis *low-pass filter*, yang bekerja dengan mengganti nilai suatu piksel pada citra asal dengan nilai median dari piksel tersebut dan lingkungan tetangganya. Dibandingkan dengan *neighborhood averaging*, filter ini lebih tidak sensitif terhadap perbedaan intensitas yang ekstrim.

Pada penapis median, suatu 'jendela' (*window*) memuat sejumlah piksel. Jendela digeser titik demi titik pada seluruh daerah citra. Pada setiap pergeseran dibuat jendela baru. Titik tengah dari jendela ini diubah dengan nilai median dari jendela tersebut.

13	**10**	**15**	**14**	**18**
12	**10**	**10**	**10**	**15**
11	**11**	**35**	**10**	**10**
13	**9**	**12**	**10**	**12**
13	**12**	**9**	**8**	**10**

Gambar 3.5 Piksel bernilai 35 terkena derau

Sebagai contoh, tinjau jendela berupa kelompok piksel (daerah berwarna biru cerah) pada sebuah citra pada gambar di atas. Piksel yang sedang diproses adalah yang mempunyai intensitas 35.

10 10 10 11 35 10 9 12 10

Urutkan piksel-piksel tersebut:

9 10 10 10 **10** 10 11 12 35

Median dari kelompok tersebut adalah 10 (dicetak tebal). Titik tengah dari jendela (35) sekarang diganti dengan nilai median (10). Hasil dari penapis median diperlihatkan pada gambar di bawah ini. Jadi, penapis median menghilangkan nilai piksel yang sangat berbeda dengan piksel tetangganya.

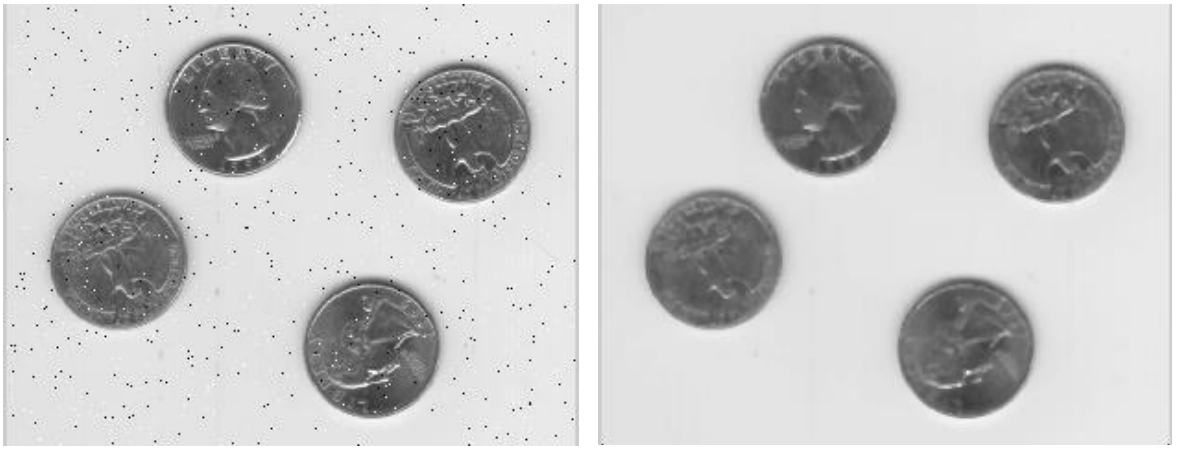
13	**10**	**15**	**14**	**18**
12	**10**	**10**	**10**	**15**
11	**11**	**10**	**10**	**10**
13	**9**	**12**	**10**	**12**
13	**12**	**9**	**8**	**10**

Gambar 3.6 Piksel bernilai 35 diganti dengan nilai 10

Contoh perintah untuk melakukan *median filtering* terhadap citra yang terkena gangguan dengan kernel berukuran 3×3 :

```
clear; clc;
I = imread('eight.tif');
I = imnoise(I, 'salt & pepper', 0.01);
[r c] = size(I);
Rep = zeros(r + 2, c + 2);
for x = 2 : r + 1
   for y = 2 : c + 1
      Rep(x,y) = I(x - 1, y - 1);
   end
end
Rep;
B = zeros(r, c);
for x = 1 : r
   for y = 1  : c
```

```
        for i = 1 : 3
            for j = 1 : 3
                q = x - 1;
                w = y -1;
                array((i - 1) * 3 + j) = Rep(i + q, j + w);
            end
        end
        B(x, y) = median(array(:));
    end
end
figure, imshow(I);
figure, imshow(uint8(B));
```



(a) (b)

Gambar 3.7 (a) Citra yang mengandung noise, (b) Hasil pelembutan dengan median filtering

BAB IV

PENDETEKSIAN TEPI (EDGE DETECTION)

Tepi adalah perubahan intensitas derajat keabuan yang mendadak (besar) dalam jarak yang singkat. Perbedaan intensitas inilah yang menampakkan rincian pada gambar. Tepi biasanya terdapat pada batas antara dua daerah berbeda pada suatu citra.

Tujuan operasi pendeteksian tepi adalah untuk meningkatkan penampakan garis batas suatu daerah atau objek di dalam citra. Karena tepi termasuk komponen berfrekuensi tinggi, maka pendeteksian tepi dapat dilakukan dengan penapis lolos tinggi.

Sebenarnya ada beberapa teknik untuk medeteksi tepi. Teknik untuk mendeteksi tepi yaitu :

1. Operator gradien pertama
2. Operator turunan kedua
3. Operator kompas

4.1 Operator Gradien Pertama

1. Operator Robert

Operator Robert adalah nama lain dari teknik differensial yang sedang dikembangkan, yaitu differensial pada arah horisontal dan differensial pada arah vertikal, dengan ditambahkan proses konversi biner setelah dilakukan differensial. Teknik konversi biner yang disarankan adalah konversi biner dengan meratakan distribusi warna hitam dan putih.

Operator Robert ini juga disamakan dengan teknik DPCM (*Differential Pulse Code Modulation*). Operator Robert Cross merupakan salah satu operator yang menggunakan jendela matrik 2x2, operator ini melakukan perhitungan dengan mengambil arah diagonal untuk melakukan perhitungan nilai gradiennya.

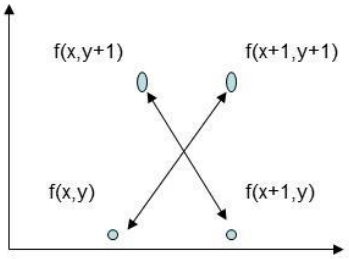
$$Gx = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad Gy = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

- Operator Robert

  disebut operator silang, gradien robert dalam arah x dan y dapat dihitung :

  $R+(x,y) = f(x+1, y+1) - f(x,y)$

  $R-(x,y) = f(x,y+1) - f(x+1,y)$



operator R+ adalah turunan berarah dalam arah 45 derajat, dan R- turunan berarah dalam arah 135 derajat

Dalam bentuk mask operator adalah :

$$R+ = \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix} \qquad R- = \begin{vmatrix} 0 & 1 \\ -1 & 0 \end{vmatrix}$$

Nilai kekuatan tepi :

$G[f(x,y)] = |R+| + |R-|$

- Contoh deteksi tepi dengan robert :

$$\begin{vmatrix} 4 & 5 & 7 & 5 & 1 \\ 2 & 1 & 3 & 4 & 5 \\ 4 & 3 & 2 & 6 & 9 \\ 4 & 2 & 5 & 7 & 1 \\ 2 & 4 & 8 & 6 & 3 \end{vmatrix} \qquad \begin{vmatrix} 6 & 8 & 5 & 3 & 1 \\ 4 & 1 & 5 & 6 & 5 \\ 3 & 2 & 6 & 7 & 9 \\ 0 & 7 & 2 & 5 & 1 \\ 2 & 4 & 8 & 6 & 3 \end{vmatrix}$$

citra awal

citra hasil pendeteksian tepi

$f'[0,0] = |4-1| + |5-2| = 6$

2. Operator Prewitt

Metode Prewitt merupakan pengembangan metode robert dengan menggunakan filter HPF yang diberi satu angka nol penyangga. Metode ini mengambil prinsip dari fungsi laplacian yang dikenal sebagai fungsi untuk membangkitkan HPF.

$$px = \begin{vmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{vmatrix} \qquad py = \begin{vmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{vmatrix}$$

3. Operator Sobel

Metode Sobel merupakan pengembangan metode robert dengan menggunakan filter HPF yang diberi satu angka nol penyangga. Metode ini mengambil prinsip dari fungsi laplacian dan gaussian yang dikenal sebagai fungsi untuk membangkitkan HPF. Kelebihan dari metode sobel ini adalah kemampuan untuk mengurangi noise sebelum melakukan perhitungan deteksi tepi.

- Digunakan operator sobel

Tinjau pengaturan pixel disekitar pixel (x,y)

$$\begin{bmatrix} A0 & a1 & a2 \\ a7 & (x,y) & a3 \\ a6 & a5 & a4 \end{bmatrix}$$

Operator sobel adalah magnitude dari gradien yang dihitung dengan

$$M = \sqrt{sx^2 + sy^2} \qquad \text{Atau } M = |sx| + |sy|$$

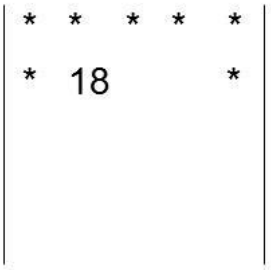
Turunan parsial dihitung dengan :

Sx = (a2+ca3+a4) – (a0+ca7+a6) ; sy = (a0+ca1+a2) – (a6+ca5+a4)

Dengan konstanta c=2 dalam bentuk mask, sx dan sy dinyatakan sebagai

$$Sx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad Sy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- Contoh citra yang akan dilakukan pendeteksian tepi dengan operator sobel,



citra awal citra hasil konvolusi

sx = 3x(-1)+2x(-1)+ 3x(-1)+2x(1)+6x(2)+7x(1) = 11
sy = 3x(1)+4x(2)+2x(1)+3x(-1)+5x(-2)+7x(-1) = -7
maka M = 18

Source code pendeteksian tepi dengan operator gradien pertama ditunjukkan oleh barisan perintah di bawah ini.

```
function J = edge_detection(I, Thres)
Gx = [-1 1];
Gy = [-1 1];
Gradien_x = convolve(I, Gx)
Gradien_y = convolve(I, Gy)
Magnitudo = sqrt((Gradien_x.^2) + (Gradien_y.^2))
%Arah_Gradien = atan(Gradien_y./Gradien_x);
J = thresholding(Magnitudo, Thres);


function B = convolve(A, k)
   [r c] = size(A);
   [m n] = size(k);
   h = rot90(k, 2);
   center = floor((size(h)+1)/2);
   left = center(2) - 1;
```

```
    right = n - center(2);
    top = center(1) - 1;
    bottom = m - center(1);
    Rep = zeros(r + top + bottom, c + left + right);
    for x = 1 + top : r + top
       for y = 1 + left : c + left
          Rep(x,y) = A(x - top, y - left);
       end
    end
    B = zeros(r , c);
    for x = 1 : r
       for y = 1  : c
          for i = 1 : m
             for j = 1 : n
                q = x - 1;
                w = y -1;
                B(x, y) = B(x, y) + (Rep(i + q, j + w) * h(i, j));
             end
          end
       end
    end


function Hasil = thresholding(Array, T)
    row = size(Array, 1);
    col = size(Array, 2);
    Hasil = zeros(row, col);
    for x = 1 : row
       for y = 1 : col
```

```
            if Array(x, y) >= T
               Hasil(x, y) = 1;
            else
               Hasil(x, y) = 0;
            end
         end
      end
```

Source code di atas disimpan dengan nama m-file 'edge_detection.m'. Contoh perintah untuk melakukan *edge detection* pada citra 'cameraman.tif' adalah sebagai berikut :

```
clear; clc;
I = imread('cameraman.tif');
Hsl = edge_detection(im2double(I), 0.25);
imshow(I);
figure, imshow(im2uint8(Hsl));
```

(a) (b)

Gambar 4.1 (a) Citra cameraman.tif, (b) Hasil pendeteksian sisi dengan nilai threshold 0.25

4.2 Operator Turunan Kedua

Operator turunan kedua disebut juga operator Laplace. Operator Laplace mendeteksi lokasi tepi yang lebih akurat khususnya pada tepi yang curam. Pada tepi yang curam, turunan keduanya memiliki persilangan nol (zero-crossing), yaitu titik di mana terdapat pergantian tanda nilai turunan kedua, sedangkan pada tepi yang landai tidak terdapat persilangan nol. Persilangan nol merupakan lokasi tepi yang akurat.

1. Operator Laplacian

a. Titik-titik tepi dilacak dengan cara menemukan titik perpotongan dengan sumbu x oleh turunan kedua → sehingga sering di sebut sebagai zero crossing operator

b. Sangat sensitif terhadap noise yang terletak pada titik-titik tepi. → dapat diatasi dengan Laplacian of Gaussian yang merupakan kombinasi dari operator laplacian dengan operator gaussian

Persamaan Laplacian

- Persamaan Laplacian

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- dimana

$$\frac{\partial^2 f}{\partial x^2} = + \frac{\partial G^2}{\partial x} = f(x+\mathbf{2},y) - \mathbf{2} f(x+\mathbf{1},y) + f(x,y)$$

$$\frac{\partial^2 f}{\partial y^2} = + \frac{\partial G^2}{\partial y} = f(x,y+\mathbf{2}) - \mathbf{2} f(x,y+\mathbf{1}) + f(x,y)$$

Persamaan laplacian

- Dengan demikian diperoleh

$$-\nabla^2 f(x,y) = -\left(\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}\right)$$

$$-\nabla^2 f(x,y) = 4f(x,y) - [f(x-1,y) + f(x+1,y) + f(x,y-1) + f(x,y+1)]$$

- Terlepas dari tandanya yang negatif atau positif, bila diimplementasikan dalam bentuk kernel:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Kernel Laplacian lain

- Dengan memberikan bobot yang lebih besar pada titik pusat, didapatkan beberapa kernel lainnya

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \qquad \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix} \qquad \begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}$$

2. Operator Laplacian of Gaussian (LOG)

Dapat dilakukan dengan cara:

a. Sebuah citra di konvolusi dengan operator gaussian, kemudian hasilnya di konvolusi dengan operator laplacian

b. Di konvolusi langsung dengan menggunakan operator Laplacian of Gaussian

Operator Laplacian of Gaussian

- Operator Laplacian of Gaussian diperoleh dari konvolusi sbb:

$$h(x,y) = \nabla^2[g(x,y) * f(x,y)]$$

$$h(x,y) = \nabla^2[g(x,y)] * f(x,y)$$

- Dimana:

$$\nabla^2[g(x,y)] = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right) e^{\frac{x^2+y^2}{2\sigma^2}}$$

Source code pendeteksian tepi dengan operator laplace ditunjukkan oleh barisan perintah di bawah ini.

```
function J = laplacian(I, Thres)
Mask = [0 1 0;
        1 -4 1;
        0 1 0];
Tepi = convolve(I, Mask)
J = thresholding(Tepi, Thres);

function B = convolve(A, k)
   [r c] = size(A);
   [m n] = size(k);
   h = rot90(k, 2);
   center = floor((size(h)+1)/2);
```

```
left = center(2) - 1;
right = n - center(2);
top = center(1) - 1;
bottom = m - center(1);
Rep = zeros(r + top + bottom, c + left + right);
for x = 1 + top : r + top
   for y = 1 + left : c + left
      Rep(x,y) = A(x - top, y - left);
   end
end
B = zeros(r , c);
for x = 1 : r
   for y = 1  : c
      for i = 1 : m
         for j = 1 : n
            q = x - 1;
            w = y -1;
            B(x, y) = B(x, y) + (Rep(i + q, j + w) * h(i, j));
         end
      end
   end
end


function Hasil = thresholding(Array, T)
   row = size(Array, 1);
   col = size(Array, 2);
   Hasil = zeros(row, col);
   for x = 1 : row
```

```
    for y = 1 : col
        if Array(x, y) >= T
            Hasil(x, y) = 1;
        else
            Hasil(x, y) = 0;
        end
    end
end
```

Source code di atas disimpan dengan nama m-file 'laplacian.m'. Selanjutnya untuk menguji keberhasilan source code di atas, buatlah suatu m-file lagi dan tuliskan source code di bawah ini :

```
clear; clc;
I = [4 4 4 8 8 8 8;
     4 4 4 8 8 8 8;
     4 4 4 8 8 8 8;
     4 4 4 8 8 8 8;
     4 4 4 8 8 8 8];
Hsl = laplacian(I, 1.0)
```

Output yang dihasilkan oleh source code di atas berturut-turut hasil konvolusi matriks citra dengan mask dan hasil akhir setelah dilakukan thresholding adalah sebagai berikut :

```
Tepi =
   -8   -4    0   -12   -8   -8   -16
   -4    0    4    -4    0    0    -8
   -4    0    4    -4    0    0    -8
   -4    0    4    -4    0    0    -8
   -8   -4    0   -12   -8   -8   -16
```

Hsl =

```
     0     0     0     0     0     0     0
     0     0     1     0     0     0     0
     0     0     1     0     0     0     0
     0     0     1     0     0     0     0
     0     0     0     0     0     0     0
```

Contoh perintah untuk melakukan *edge detection* pada citra 'cameraman.tif' adalah sebagai berikut :

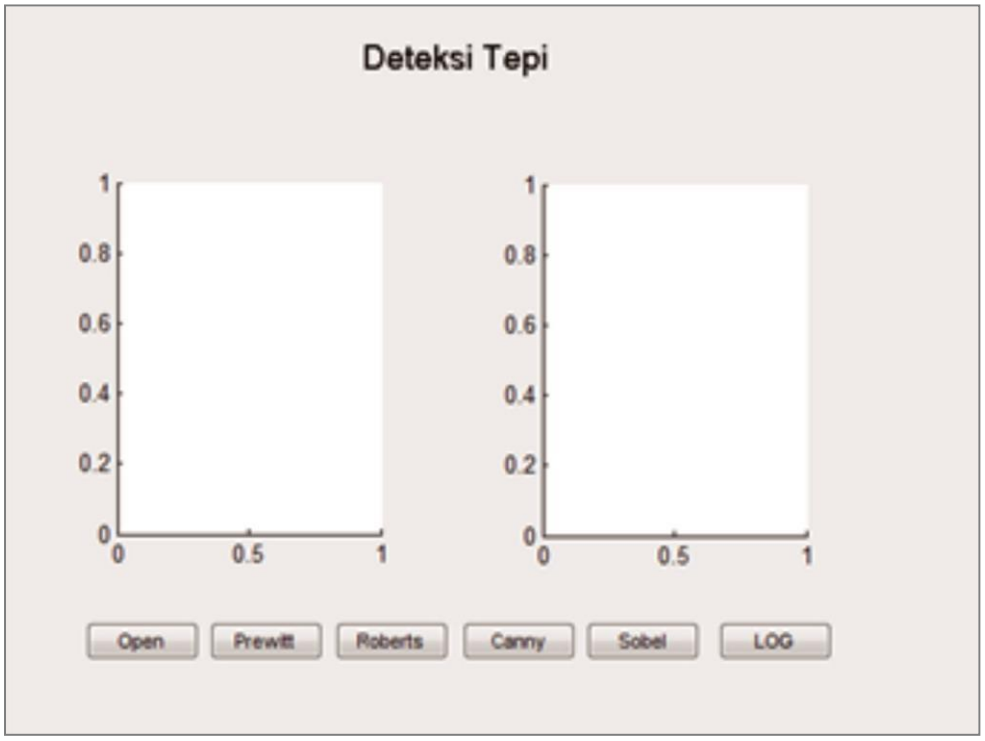
```
clear; clc;
I = imread('cameraman.tif');
Hsl = laplacian(im2double(I), 0.25);
imshow(I);
figure, imshow(im2uint8(Hsl));
```

(a) (b)

Gambar 4.2 (a) Citra cameraman.tif, (b) Hasil pendeteksian sisi dengan nilai threshold 0.25

Perhatikan bahwa pendeteksian sisi yang dilakukan oleh operator laplacian menghasilkan tepi yang lebih akurat dibandingkan dengan operator gradien meskipun dengan nilai *threshold luminance* yang sama.

4.3 Aplikasi Deteksi Tepi



Gambar 4.3 Aplikasi Deteksi Tepi

Koding Operator Prewitt

```
proyek=guidata(gcbo);
I=get(proyek.axes1,'Userdata');
gray=rgb2gray(I);
BW=edge(gray,'prewitt');
set(proyek.figure1,'CurrentAxes',proyek.axes2);
set(imshow(BW));
set(proyek.axes2,'Userdata',A);
redo_Callback(hObject,eventdata, handles);
```

Koding Operator Sobel

```
proyek=guidata(gcbo);
I=get(proyek.axes1,'Userdata');
```

```
gray=rgb2gray(I); BW=edge(gray,'sobel');
set(proyek.figure1,'CurrentAxes',proyek.axes2);
set(imshow(BW));
set(proyek.axes2,'Userdata',A);
redo_Callback(hObject,eventdata, handles);
```

Koding Operator Roberts

```
proyek=guidata(gcbo);
I=get(proyek.axes1,'Userdata');
gray=rgb2gray(I);
BW=edge(gray,'roberts');
set(proyek.figure1,'CurrentAxes',proyek.axes2);
set(imshow(BW));
set(proyek.axes2,'Userdata',A);
redo_Callback(hObject,eventdata, handles);
```

Koding Operator Canny

```
proyek=guidata(gcbo);
I=get(proyek.axes1,'Userdata');
gray=rgb2gray(I);
BW=edge(gray,'canny');
set(proyek.figure1,'CurrentAxes',proyek.axes2);
set(imshow(BW));
set(proyek.axes2,'Userdata',A);
redo_Callback(hObject,eventdata, handles);
```

Koding Operator LOG

```
proyek=guidata(gcbo);
I=get(proyek.axes1,'Userdata');
gray=rgb2gray(I);
BW=edge(gray,'log');
set(proyek.figure1,'CurrentAxes',proyek.axes2);
set(imshow(BW));
set(proyek.axes2,'Userdata',A);
redo_Callback(hObject,eventdata, handles);
```

BAB V
PEMAMPATAN CITRA

Pemampatan atau kompresi citra merupakan suatu teknik untuk mengkodekan piksel-piksel di dalam citra sehingga diperoleh representasi memori yang minimal. Kebanyakan citra mengandung duplikasi data. Duplikasi data pada citra berarti dua hal. Pertama, besar kemungkinan suatu piksel dengan piksel tetangganya memiliki intensitas yang sama, sehingga penyimpanan setiap piksel memboroskan tempat. Kedua, citra banyak mengandung bagian (region) yang sama, sehingga bagian yang sama ini tidak perlu dikodekan berulang kali. Pemampatan citra bertujuan untuk meminimalkan kebutuhan memori untuk merepresentasikan citra digital. Prinsip umum yang digunakan pada proses pemampatan citra adalah mengurangi duplikasi data di dalam citra sehingga memori yang dibutuhkan untuk merepresentasikan citra menjadi lebih sedikit daripada representasi citra semula. Ada beberapa metode dasar kompresi, diantaranya:

1. Metode Huffman
2. Metode RLE *(Run Length Encoding)*
3. Metode Aritmatik
4. Metode KUantisasi
5. Metode LZW, dan
6. Pendekatan Fraktal

5.1 Metode Run-Length Encoding

Metode RLE cocok digunakan untuk memampatkan citra yang memiliki kelompok-kelompok piksel yang berderajat keabuan yang sama. Pemampatan citra dengan metode RLE dilakukan dengan membuat rangkaian pasangan nilai (p,q) untuk setiap baris piksel, nilai pertama (p) menyatakan derajat keabuan, sedangkan nilai kedua (q) menyatakan jumlah piksel berurutan yang memiliki derajat keabuan tersebut (dinamakan run length).

Contoh berikut akan menunjukkan citra yang akan ditinjau 10x10 piksel dengan 8 derajat keabuan yang dinyatakan sebagai matriks derajat keabuan sebagai berikut :

0	0	0	0	0	2	2	2	2	2
0	0	0	1	1	1	1	2	2	2

1	1	1	1	1	1	1	1	1	1
4	4	4	4	3	3	3	3	2	2
3	3	3	5	5	7	7	7	7	6
2	2	6	0	0	0	0	1	1	0
3	3	4	4	3	2	2	2	1	1
0	0	0	0	0	0	0	0	1	1
1	1	1	1	0	0	0	2	2	2
3	3	3	2	2	2	1	1	1	1

Semuanya ada 100 buah nilai. Pasangan nilai untuk setiap baris run yang dihasilkan dengan metode pemampatan RLE:

(0,5),(2,5)
(0,3),(1,4),(2,3)
(1,10)
(4,4),(3,4),(2,2)
(3,3),(5,2),(7,4),(6,1)
(2,2),(6,1),(0,4),(1,2),(0,1)
(3,2),(4,2),(3,1),(2,2),(1,2)
(0,8),(1,2)
(1,4),(0,3),(2,3)
(3,3),(2,3),(1,4)

Hasil Pengkodean:

0 5 2 5 0 3 1 4 2 3 1 10 4 4 3 4 2 2 3 3 5 2 7 4 6 1 2 2 6 1 0 4 1 2 0 1 3 2 4 2 3 1 2 2 1 2 0 8 1 2 1 4 0 3 2 3 3 3 2 3 1 4

Semuanya = 64 piksel

Ukuran citra sebelum dikompres = 10 x 10 x 3 bit = 300 bit

Ukuran citra setelah dikompres = 64 x 3 bit = 192 bit

(—) , artinya 36% dari citra semula telah dimampatkan

Source code untuk kompresi :

```
function rle = RLE_Encode(I)
[row, col] = size(I);
rle(1) = I(1, 1);
rle(2) = 0;
idk = 1;
for x = 1 : row
   for y = 1 : col
      currpixel = I(x,y);
      if currpixel == rle(idk)
         rle(idk + 1) = rle(idk + 1) + 1;
      else
         idk = idk + 2;
         rle(idk) = currpixel;
         rle(idk + 1) = 1;
      end
   end
end
```

Source code di atas disimpan dengan nama m-file 'RLE_Encode.m'. Selanjutnya untuk menguji keberhasilan source code di atas, buatlah suatu m-file lagi dan tuliskan source code di bawah ini

```
clear all; clc;
I = [1 1 1 7 1 3;
     4 4 6 1 2 2;
```

```
    7 7 7 5 5 5;
    6 4 4 2 2 2;
    5 5 2 2 2 1;
    2 3 3 3 0 0];
rle = RLE_Encode(I)
```

Source code untuk dekompresi :

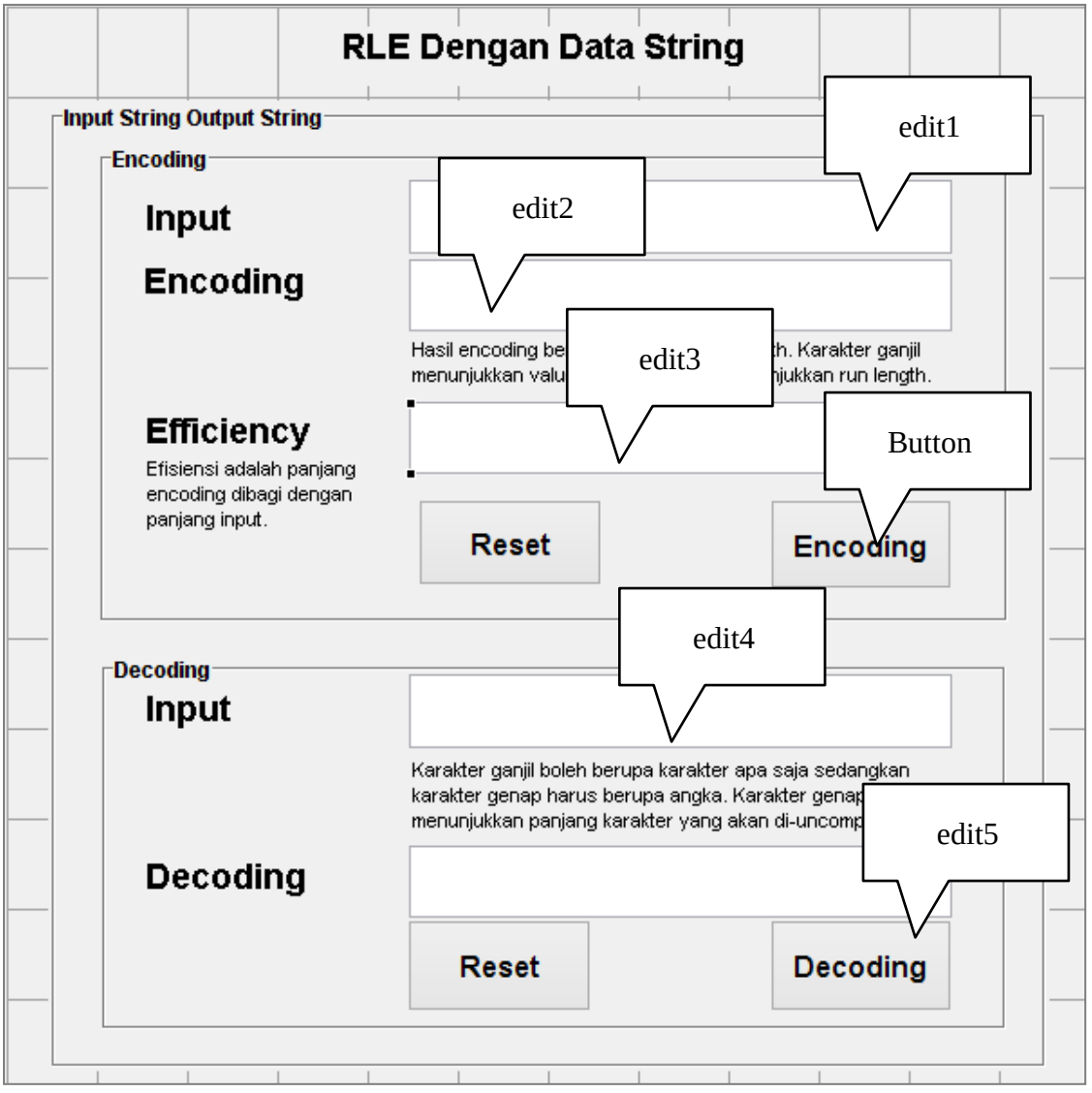
```
function Decompressed = RLE_Decode(rle, r, c)
Decompressed = zeros(r, c);
index = 1;
kali = 0;
for x = 1 : r
   for y = 1 : c
      kali = kali + 1;
      if kali == rle(1,index+1)
         Decompressed(x, y) = rle(1,index);
         index = index + 2;
         kali = 0;
      else
         Decompressed(x, y) = rle(1, index);
      end
   end
end
```

Source code di atas disimpan dengan nama m-file 'RLE_Decode.m'. Selanjutnya untuk menguji keberhasilan source code di atas, buatlah suatu m-file lagi dan tuliskan source code di bawah ini

```
clear all; clc;
rle = [1 3 7 1 1 1 3 1 4 2 6 1 1 1 2 2 7 3 5 3 6 1 4 2 2 3 5 2 ...
       2 3 1 1 2 1 3 3 0 2];
r = 6;
c = 6;
Decompressed = RLE_Decode(rle, r, c)
```

5.2 Aplikasi Kompresi RLE Data String

Buatlah desain GUI pada matlab seperti di bawah ini:



Gambar 5.1. Aplikasi Kompresi RLE pada Data String

Koding Button Reset

```
set(handles.edit1,'String','')
set(handles.edit2,'String','')
set(handles.edit3,'String','')
```

Koding Button Encoding

```
st = get(handles.edit1,'string');
b=st;
output = ' ';
while length(st)
    output = [output st(1)];
    st = st(2:end);
    count = 1;
    while st & (output(end) == st(1))
        st = st(2:end);
        count = count + 1;
    end
    output = [output num2str(count)];
end
a = [output];
   set(handles.edit2,'string',(a));
   c=length(a)-1;
efisiensi=c/length(b);
set(handles.edit3,'string',(efisiensi));
```

Koding Button Decoding

```
output = get(handles.edit4,'string');
%output=char(output(2:end));
```

```
y=output(1);
a=1;
while length(output)
   num=str2num(output(2));
   count=1;
   if a==1
    if num>1
      while count < num
      y=[y output(1)];
      count=count+1;
      a=a+1;
      end;
      else
      while count <= num
      y=[y];
      count=count+1;
      a=a+1;
      end;
   end;
   else
      while count <= num
      y=[y output(1)];
      count=count+1;
      end; end;
   output=output(3:end);
end;
set(handles.edit5,'string',(y));
```

BAB VI WATERMARK CITRA

Salah satu cara untuk melindungi hak cipta multimedia adalah dengan menyisipikan informasi ke dalam data multimedia tersebut dengan teknik watermarking. Informasi yang disisipkan ke dalam data multimedia disebut watermark, dan watermark dapat dianggap sebagai sidik digital (digital signature) dari pemilik yang sah atas produk multimedia tersebut. Dengan kata lain, watermark yang disisipkan menjadi label hak cipta dari pemiliknya. Pemberian signature dengan teknik watermarking ini dilakukan sedemikian sehingga informasi yang disisipkan tidak merusak data digital yang dilindungi. Sehingga, seseorang yang membuka produk multimedia yang sudah disisipi watermark tidak menyadari kalau di dalam data multimedia tersebut terkandung label kepemilikan pembuatnya.

Jika ada orang lain yang mengklaim bahwa produk multimedia yang didapatkan adalah miliknya, maka pemegang hak cipta atas karya multimedia tersebut dapat membantahnya dengan mengektrasi watermark dari dalam data multimedia yang disengketakan. Watermark yang diekstrasi dibandingkan dengan watermark pemegang hak cipta. Jika sama, berarti memang dialah pemegang hak cipta produk multimedia tersebut.

6.1 Teknik Penyembunyian Data

Penyembunyian data dilakukan dengan mengganti bit-bit data di dalam segmen citra dengan bit-bit rahasia. Hingga saat ini, metode yang paling sederhana adalah metode modifikasi LSB (Least Significant Bit Modification). Pada susunan bit di dalam sebuah byte (1 byte = 8 bit), ada :

- Bit yang paling berarti (Most Significant Bit atau MSB) dan
- Bit yang paling kurang berarti (Least Significant Bit atau LSB)

Misalnya pada byte 11010010, bit 1 yang pertama (yang digarisbawahi) adalah bit MSB dan bit 0 yang terakhir (digarisbawahi) adalah bit LSB. Bit yang cocok diganti adalah bit LSB, sebab penggantian hanya mengubah nilai byte tersebut satu lebih tinggi atau satu lebih rendah dari nilai sebelumnya.

Seperti kita ketahui bersama bahwa byte 11111111 berarti bernilai 255. Apabila kita mengubah LSB, yaitu nilai satu pada deret yang paling akhir, hal ini hanya akan menyebabkan nilai naik / turun satu dari sebelumnya.

11111111 = 255 => Nilai piksel awal 255 yaitu pada citra grayscale berarti warna putih
11111110 = 254 => Nilai piksel hanya turun satu, yaitu 254 berarti pada citra warna putih tidak akan berubah terlalu jauh dari sebelumnya.

Bayangkan apabila kita memilih untuk memodifikasi MSB. Maka, yang terjadi adalah nilai intensitas akan turun / naik begitu banyak

11111111 = 255 => Nilai piksel awal 255 yaitu pada citra grayscale berarti warna putih
01111111 = 127 => Nilai piksel menjadi 127. Pada citra grayscale intensitas warna putih akan turun banyak menjadi keabu-abuan.

Misalkan segmen piksel-piksel citra sebelum penambahan bit-bit watermark adalah :

00110011 10100010 11100010 01101111

Misalkan data rahasia (yang telah dikonversi ke system biner) adalah 0111. Setiap bit dari watermark menggantikan posisi LSB dari segmen data citra menjadi :

00110010 10100011 11100011 01101111

Contoh listing di bawah ini akan menunjukkan bagaimana cara kerja penyisipan watermark ke dalam citra :

Source code untuk encoding :

```
function I = w_encode(cover, message, bitpos)
message = double(message);
message = round(message./256);
message = uint8(message);
Mc = size(cover, 1);
Nc = size(cover, 2);
I = cover;
```

```
for i = 1 : Mc
   for j = 1 : Nc
      I(i, j) = bitset(I(i, j), bitpos, message(i, j));
   end
end
```

Source code di atas disimpan dengan nama m-file 'w_encode.m'. Selanjutnya untuk menguji keberhasilan source code di atas, buatlah suatu m-file lagi dan tuliskan source code di bawah ini

```
clear all; clc;
bitpos = 1;
cover = imread('lena.jpg');
message = imread('cameraman.tif');
I = w_encode(cover, message, bitpos);
imwrite(I,'watermarked.bmp');
imshow(cover);
figure, imshow(uint8(I));
```

Gambar 6.1 (a) Citra sebelum disisipi watermark, (b) Citra sesudah disisipi watermark

Decoding dalam proses watermark merupakan teknik mengungkap / membongkar watermark yang telah disisipkan pada citra digital. Contoh listing di bawah ini akan menunjukkan bagaimana cara pengungkapan watermark dari dalam citra digital:

Source code untuk decoding :

```
function W = w_decode(I, bitpos)
Mw = size(I, 1);
Nw = size(I, 2);
for i = 1 : Mw
   for j = 1 : Nw
      W(i, j) = bitget(I(i, j), bitpos);
   end
end
W = 256 * W;
```

Source code di atas disimpan dengan nama m-file 'w_decode.m'. Selanjutnya untuk menguji keberhasilan source code di atas, buatlah suatu m-file lagi dan tuliskan source code di bawah ini

```
clear all; clc;
bitpos = 1;
I = imread('watermarked.bmp');
W = w_decode(I, bitpos);
imshow(I);
figure, imshow(W);
```

Gambar 6.2 (a) Citra cover, (b) Hasil pengungkapan watermark yang disisipkan