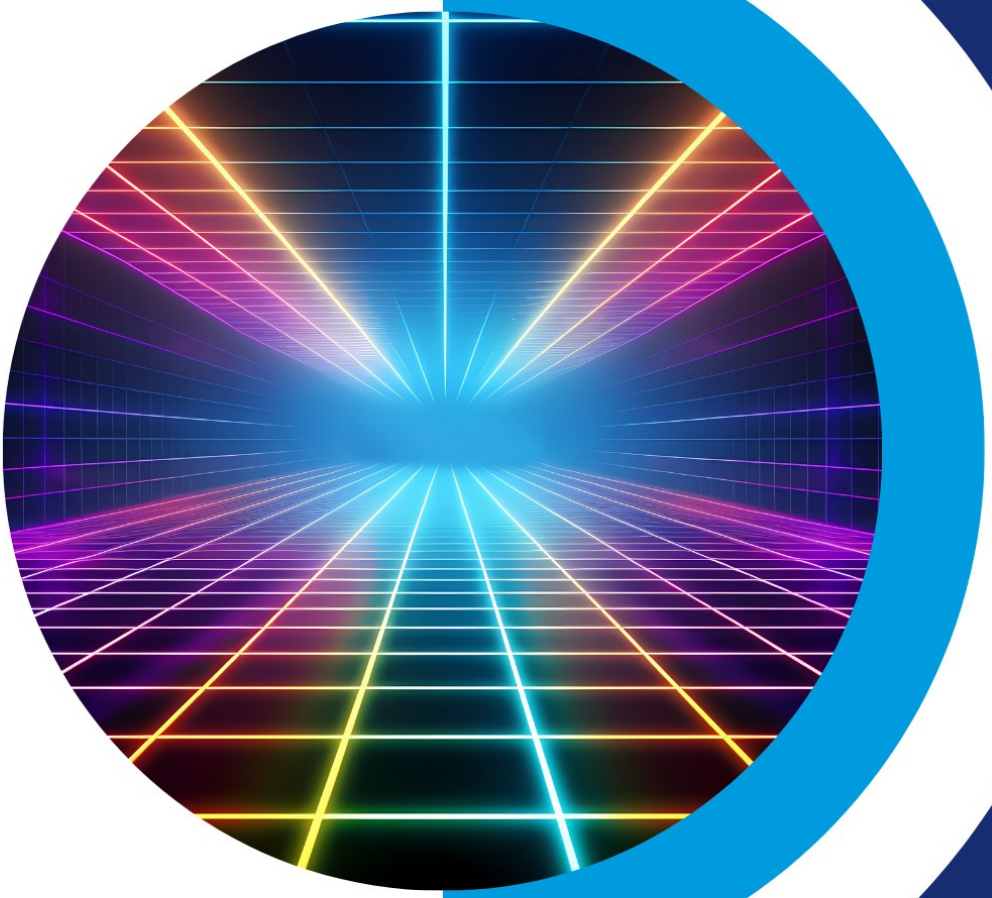
UMSIDA PRESS

PENGOLAHAN CITRA DIGITAL MENGGUNAKAN MATLAB



Penulis :
Hindarto
Ade Eviyanti
Suhendro Busono
Sumarno
Jamaaluddin

BUKU AJAR
PENGELOLAHAN CITRA DIGITAL MENGGUNAKAN MATLAB

Penulis:
Hindarto

Ade Eviyanti

Suhendro Busono

Sumarno

Jamaaluddin

Diterbitkan oleh
UMSIDA PRESS
Jl. Mojopahit 666 B Sidoarjo
ISBN: 978-623-464-121-9

BUKU AJAR PENGELOLAHAN CITRA DIGITAL MENGGUNAKAN MATLAB
Penulis: Hindarto; Ade Eviyanti; Suhendro Busono; Sumarno; Jamaaluddin
ISBN: 978-623-464-121-9
Editor: M. Tanzil Multazam & Mahardika Darmawan K.W.
Copy Editor: Wiwit Wahyu Wijayanti
Design Sampul dan Tata Letak: Wiwit Wahyu Wijayanti
Penerbit: UMSIDA Press
Redaksi: Universitas Muhammadiyah Sidoarjo Jl. Mojopahit No
666B Sidoarjo, Jawa Timur

Cetakan Pertama, Februari 2024

Daftar Isi

		Hal
Bab 1	**Pengenalan tentang Pemrograman Matlab**	1
1.1	Memulai menjalankan Matlab	1
1.2	Menggunakan MATLAB sebagai kalkulator	1
1.3	Variabel	2
1.4	Pesan Kesalahan	2
1.5	Fungsi Matematika	2
1.6	Membuat plot sederhana	3
1.7	Menambahkan judul, label sumbu, dan anotasi	4
1.8	Beberapa kumpulan data dalam satu plot	4
1.9	Menentukan gaya dan warna garis	5
Bab 2	**Pengenalan Pengolahan Citra**	7
2.1	Images and pictures	7
2.2	Akuisisi Gambar dan pengambilan sampel	7
2.3	Greyscale images	8
2.4	Fungsi imshow	9
Bab 3	**Pemrosesan Titik**	12
3.1	Operasi aritmatika	12
3.2	Operasi Aritmatik	13
Bab 4	**Kuantitas dan Kualitas Citra**	15
4.1	Kuantitas Citra	15
4.2	Kualitas Citra	16
Bab 5	**Jenis-Jenis Citra Dan Konversi Citra**	21
5.1	Jenis – Jenis Citra	21
5.2	Konversi Citra	29
Bab 6	**Operasi Ketetanggaan Piksel dan Histogram Citra**	36
6.1	Operasi Ketetanggaan Piksel pada Citra	36
6.2	**H**istogram Citra	39
Bab 7	**Operasi Filter Median, Mean dan Rata-Rata**	42
7.1	Filter Rata-rata (Mean Filter)	42
7.2	Filter Median	43
7.3	Filter Rata-rata (Average Filter)	44
Bab 8	**Operasi Geometri Citra**	47
8.1	Pergeseran (Translasi)	47
8.2	Rotasi	48
8.3	Skalasi (Penyesuaian Ukuran)	49
8.4	Shearing (Penyusutan)	52
8.5	Flipping (Pembalikan)	53
8.6	Perspektif (Keprojekan)	56
8.7	Cropping (Pemotongan)	58
8.8	Resampling	59
Bab 9	**Operasi Morfologi pada Citra**	62
9.1	Dilasi (Dilation)	62
9.2	Erosion	63
9.3	Opening	64
9.4	Closing	65
9.5	Gradien	67

9.6 Hit-Or-Miss Transform. 69
Bab 10 **Segmentasi Citra** 71
10.1 Segmentasi Berdasarkan Ambang Batas 71
10.2 Segmentasi Berdasarkan Pemetaan Warna 72
10.3 Segmentasi Berdasarkan Region 74
10.4 Segmentasi Watershed 77
Bab 11 **Ekstrasi Ciri** 80
11.1 Ekstrasi Ciri Berdasarkan Intensitas 80
11.2 Ekstrasi Ciri Berdasarkan Bentuk 82
11.3 Ekstrasi Ciri Berdasarkan Tepi 84
11.4 Ekstrasi Ciri Berdasarkan Warna 86
11.5 Ekstrasi Ciri Berdasarkan Tekstur 89
11.6 Ekstrasi Ciri Berdasarkan Transformasi 90
Pustaka 97

Daftar Gambar

		Hal
Gambar 1.1	Antarmuka grafis ke ruang kerja MATLAB	1
Gambar 1.2	Plot vektor x dan y	4
Gambar 1.3	Plot fungsi Sinus	4
Gambar 1.4	Contoh tipikal beberapa plot	5
Gambar 2.1	Pengambilan sampel suatu fungsi—pengambilan sampel yang terlalu rendah	7
Gambar 2.2	Mengambil sampel suatu fungsi dengan lebih banyak titik	8
Gambar 2.3	Pengaruh pengambilan sampel	8
Gambar 2.4	Gambar wombat dengan RGB to Grayscale	9
Gambar 2.5	Upaya konversi tipe data	10
Gambar 2.6	Penskalaan dengan membagi matriks gambar dengan skalar	10
Gambar 3.1	Skema pemrosesan transformasi	12
Gambar 3.2	Penjumlahan dan pengurangan suatu konstanta	13
Gambar 3.3	Operasi aritmatika pada suatu gambar: menambah atau mengurangi suatu konstanta	13
Gambar 5.1	Contoh Citra Grayscale	21
Gambar 5.2	Contoh Citra RGB	22
Gambar 5.3	Modifikasi Citra RGB	23
Gambar 5.4	Modifikasi Intensitas Citra RGB	24
Gambar 5.5	Contoh Citra Warna, Grayscale dan Biner	25
Gambar 5.6	Contoh Citra HIS	27
Gambar 5.7	Contoh Citra CMYK (Cyan, Magenta, Yellow, Black)	28
Gambar 5.8	Contoh Konversi Citra darin warna ke grayscale	29
Gambar 5.9	Contoh Konversi Citra darin warna ke HSI	31
Gambar 5.10	Contoh Konversi Citra darin warna ke Biner	32
Gambar 5.11	Contoh Konversi Citra dari Biner ke Grayscale	34
Gambar 6.1	Citra Warna dan Citra hasil Blur menggunakan filter rata-rata	36
Gambar 6.2	Citra Warna dan Citra hasil Blur menggunakan filter Lapacian	37
Gambar 6.3	Citra Warna dan Citra hasil Blur menggunakan filter Sobel	38
Gambar 6.4	Citra Warna dan Citra hasil Blur menggunakan filter Morfologi	39
Gambar 6.5	histogram citra grayscale	39
Gambar 6.6	histogram citra Warna	40
Gambar 6.7	citra asli dan citra hasil equalization	40
Gambar 6.8	citra Biner dengan threshold otsu	41
Gambar 7.1	Citra Asli dan Citra setelah Filter Rata-rata	43
Gambar 7.2	Citra Asli dan Citra setelah Filter Median	44
Gambar 7.3	Citra Asli dan Citra setelah Filter Rata-rata	45
Gambar 7.4	Citra Asli dan Citra setelah Filter Rata-rata	46
Gambar 8.1	Pergeseran Citra Asli dan Citra setelah Translasi	48
Gambar 8.2	Citra Asli dan Citra setelah Rotasi	49
Gambar 8.3	Citra Asli dan Citra setelah diperbesar	51
Gambar 8.4	Citra Asli dan Citra setelah diperkecil	51
Gambar 8.5	Citra Asli dan Citra setelah Shearing	53
Gambar 8.6	Citra Asli dan membalikkan citra secara horizontal (kiri ke kanan).	53
Gambar 8.7	Citra Asli dan membalikkan citra secara vertikal	55
Gambar 8.8	Citra Asli dan membalikkan citra secara vertikal dan horisontal	55
Gambar 8.9	Citra Asli dan citra setelah transformasi perspektif	57
Gambar 8.10	Citra Asli dan citra setelah transformasi perspektif	58
Gambar 8.11	Citra Asli dan citra setelah hasil pemotongan	59
Gambar 9.1	Citra Biner Asli dan citra setelah hasil Dilasi	63

Gambar 9.2	Citra Biner Asli dan citra setelah hasil Erosi	64
Gambar 9.3	Citra Biner Asli dan citra setelah hasil Operasi Opening	65
Gambar 9.4	Citra sebelum closing dan citra setelah closing	66
Gambar 9.5	Citra Asli dan citra setelah hasil Gradient	68
Gambar 9.6	Citra Asli dan citra setelah hasil Hit-or-Miss Transform (HMT)	70
Gambar 10.1	Citra Grayscale dan Citra hasil Segemnetasi	72
Gambar 10.2	Citra Warna dan Citra hasil Segemnetasi	74
Gambar 10.3	Citra Grayscale dan Citra hasil Segemnetasi	76
Gambar 10.4	Citra Grayscale dan Citra hasil Segemnetasi	78
Gambar 11.1	Ekstrasi ciri menggunakan intensitas	81
Gambar 11.2	Ekstrasi ciri menggunakan Convex Hull Objek	84
Gambar 11.3	Ekstrasi ciri menggunakan Sobel	85
Gambar 11.4	Ekstrasi ciri menggunakan Canny	86
Gambar 11.5	Ekstrasi ciri berdasarkan warna	89
Gambar 11.6	Ekstrasi ciri berdasarkan GLCM	90
Gambar 11.7	Ekstrasi ciri menggunakan FFT	92
Gambar 11.8	Ekstrasi ciri menggunakan Wavelet	93
Gambar 11.9	Ekstrasi ciri menggunakan PCA	93
Gambar 11.10	Ekstrasi ciri menggunakan Transformasi Hough	94

Daftar Tabel

		Hal
Tabel 1.1	Operator aritmatika dasar	2
Tabel 1.2	Fungsi Dasar	2
Tabel 1.3	Nilai konstanta yang telah ditentukan sebelumnya	3
Tabel 1.4	Atribut plot	5

PRAKATA

Buku ajar ini ditulis sebagai buku panduan dan referensi mahasiswa Informatika untuk mengambil mata kuliah Pengolahan Citra Digital. Kelebihan buku ajar ini yaitu terdapat soal-soal yang ada pada bagian bab di buku ajar ini dan terdapat contoh-contoh aplikasi yang berhubungan dengan mata kuliah Pengolahan Citra Digital. Sasaran utama buku ajar ini adalah mahasiswa Informatika dan sasaran umum buku ajar ini adalah para dosen dan praktisi yang ingin belajar tentang Pengolahan Citra Digital dengan menggunakan Pemrograman Matlab. Prasyarat agar dapat menggunakan buku ajar ini tidak ada. Penulisan buku ajar Pengolahan Citra Digital dengan menggunakan Pemrograman Matlab ini ditulis dalam 11 BAB yang berisi:

Bab 1 Pengenalan tentang Pemrograman Matlab

Dalam bab 1 ini dijelaskan tentang Konsep Dasar Pemrograman Matlab

Bab 2 Dasar Matematika untuk Pengolahan Citra

Dalam bab 2 dijelaskan tentang Program menggunakan Matlab untuk Operasi Dasar Pengolahan Citra Digital,

Bab 3 Pemrosesan Titik

Dalam bab 3 ini dijelaskan tentang Teori dan Program menggunakan Matlab untuk Pemrosesan titik

Bab 4 Kuantitas dan Kualitas Citra

Dalam bab 4 ini dijelaskan tentang teori dan Program menggunakan Matlab untuk Kuantitas dan Kualitas Citra

Bab 5 Jenis-Jenis Citra Dan Konversi Citra

Dalam bab 5 ini dijelaskan tentang Teori dan Program menggunakan Matlab untuk Jenis-Jenis Citra Dan Konversi Citra

Bab 6 Operasi Ketetanggaan Piksel dan Histogram Citra

Dalam bab 6 ini dijelaskan teori tentang Operasi Ketetanggaan Piksel dan Histogram Citra serta Program menggunakan Matlab untuk Operasi Ketetanggaan Piksel dan Histogram Citra

Bab 7 Operasi Filter Median, Mean dan Rata-Rata

Dalam bab 7 ini dijelaskan teori Operasi Filter Median, Mean dan Rata-Rata serta Program menggunakan Matlab untuk Operasi Filter Median, Mean dan Rata-Rata

Bab 8 Operasi Geometri Citra

Dalam bab 8 ini dijelaskan tentang Program dan menggunakan Matlab untuk Operasi Geometri Citra

Bab 9 Operasi Morfologi pada Citra

Dalam bab 9 ini dijelaskan tentang teori dan Program menggunakan Matlab untuk Operasi Morfologi pada Citra

Bab 10 Segmentasi Citra

Dalam bab 10 ini dijelaskan tentang teori dan Program menggunakan Matlab untuk Segmentasi Citra

Bab 11 Ekstraksi Ciri

Dalam bab 11 ini dijelaskan tentang teori dan Program menggunakan Matlab untuk Ekstraksi Ciri

Buku ajar ini dibaca dari awal bab sampai akhir, sehingga antar bab ada yang saling berkaitan. Buku ajar ada buku pendamping yang saling berhubungan, diantaranya buku Grafika Komputer.

Dengan selesainya penulisan buku ajar ini penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan bahan-bahan tulisan baik langsung maupun tidak langsung. Penulis juga mengucapkan terima kasih khususnya kepada:

1. Dr. Hidayatullah, M.Si pemangku pimpinan tertinggi yaitu Rektor Universitas Muhammadiyah Sidoarjo yang telah memberikan dan memfasilitasi dalam penulisan buku ajar ini.
2. Perpustakaan Universitas Muhammadiyah Sidoarjo yang telah memfasilitasi dan mengkoordinasi dalam penulisan buku ajar ini.

Akhir kata, kritik dan saran sangat diharapkan untuk penyempurnaan buku ajar ini. Harapan kami semoga buku ajar ini dapat digunakan sebagai tambahan informasi dan bermanfaat bagi aktivitas pembelajaran mata kuliah Pengolahan Citra Digital di Program Studi Informatika, Fakultas Sains dan Teknologi, Universitas Muhammadiyah Sidoarjo.

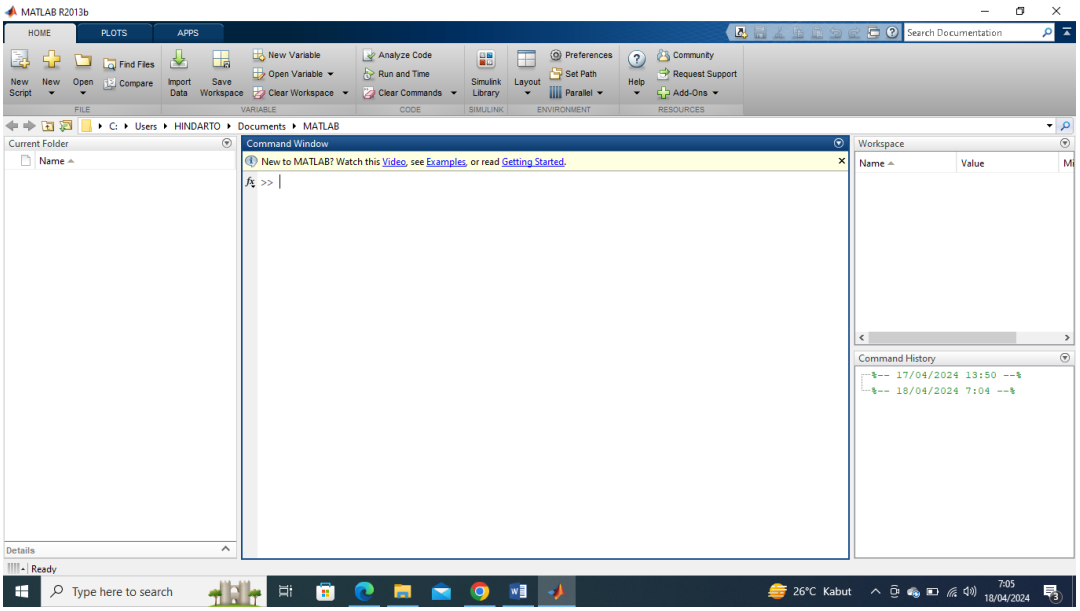
Penulis

Bab 1
Pengenalan tentang Pemrograman Matlab

1.1 Memulai menjalankan Matlab

Ketika Anda memulai MATLAB, jendela khusus yang disebut desktop MATLAB muncul. Desktop adalah jendela yang berisi jendela lainnya. Alat utama di dalam atau dapat diakses dari desktop adalah:

a. The Command Window
b. The Command History
c. The Workspace
d. The Current Directory
e. The Help Browser
f. The Start button



Gambar 1.1: Antarmuka grafis ke ruang kerja MATLAB

Saat MATLAB pertama kali dijalankan, tampilan layarnya seperti pada Gambar 1.1. Ilustrasi ini juga menunjukkan konfigurasi default desktop MATLAB. Anda dapat menyesuaikan susunan alat dan dokumen sesuai kebutuhan Anda. Sekarang, kami tertarik untuk melakukan beberapa perhitungan sederhana. Kami berasumsi bahwa Anda memiliki pemahaman yang cukup tentang komputer tempat MATLAB dijalankan. Anda sekarang dihadapkan dengan desktop MATLAB di komputer Anda, yang berisi prompt (>>) di Command Window. Biasanya ada 2 jenis prompt:

1. untuk versi lengkap
2. untuk versi pendidikan

1.2 Menggunakan MATLAB sebagai kalkulator

Sebagai contoh penghitungan interaktif sederhana, cukup ketikkan ekspresi yang ingin Anda evaluasi. Mari kita mulai dari awal. Misalnya, Anda ingin menghitung ekspresi, 1 + 2 × 3. Anda mengetikkannya pada perintah prompt (>>) sebagai berikut,

```
>> 1+2*3
jawab =
7
```

Anda akan memperhatikan bahwa jika Anda tidak menentukan variabel keluaran, MATLAB menggunakan variabel default ans, kependekan dari jawaban, untuk menyimpan hasil perhitungan saat ini. Perhatikan bahwa variabel ans dibuat (atau ditimpa, jika sudah ada). Untuk menghindari hal ini, Anda dapat memberikan nilai

pada variabel atau nama argumen keluaran. Misalnya,
>> x = 1+2*3
x =
7
akan mengakibatkan x diberi nilai 1 + 2 × 3 = 7. Nama variabel ini selalu dapat digunakan untuk merujuk pada hasil perhitungan sebelumnya. Oleh karena itu, komputasi 4x akan menghasilkan
>> 4*x
jawab =
28.0000
Sebelum kita menyimpulkan sesi minimum ini, Tabel 1.1 memberikan sebagian daftar operator aritmatika.

Tabel 1.1: Operator aritmatika dasar

SIMBOL	OPERASI	CONTOH
+	Jumlah	4 + 3
-	Kurang	4 – 3
*	Kali	4 * 3
/	Bagi	4 / 3

1.3 Variabel

Setelah variabel dibuat, variabel tersebut dapat ditugaskan kembali. Selain itu, jika Anda tidak ingin melihat hasil antara, Anda dapat menyembunyikan keluaran numerik dengan memberi tanda titik koma (;) di akhir baris. Maka urutan perintahnya terlihat seperti ini:
>> t = 5;
>> t = t+1
t =
6

1.4 Pesan kesalahan

Jika kita salah memasukkan ekspresi, MATLAB akan mengembalikan pesan kesalahan. Misalnya, berikut ini, kita tidak menyertakan tanda perkalian, *, pada ekspresi berikut
>> x = 10;
>> 5x
??? 5x
|
Error: Unexpected MATLAB expression.

1.5 Fungsi Matematika

MATLAB menawarkan banyak fungsi matematika yang telah ditentukan sebelumnya untuk komputasi teknis yang berisi sekumpulan besar fungsi matematika. Mengetik help elfun dan help specfun akan memanggil daftar lengkap fungsi dasar dan fungsi khusus. Ada daftar panjang fungsi matematika yang dibangun di MATLAB. Fungsi-fungsi ini disebut bawaan. Banyak fungsi matematika standar, seperti sin(x), cos(x), tan(x), ex, ln(x), dievaluasi dengan fungsi sin, cos, tan, exp, dan log masing-masing di MATLAB. Tabel 1.2 mencantumkan beberapa fungsi yang umum digunakan, dimana variabel x dan y dapat berupa bilangan, vektor, atau matriks.

Tabel 1.2: Fungsi Dasar

cos(x) Cosines	abs(x) Nilai Mutlak
sin(x) Sines	sign(x) Fungsi tanda tangan
tan(x) Tangent	max(x) Nilai Maximum
acos(x) Busur cosines	min(x) Nilai Minimum
asin(x) Busur sines	ceil(x) Putaran ke arah +∞
atan(x) Busur tangent	floor(x) Putaran ke arah −∞
exp(x) Exponensial	round(x) Bulatkan ke bilangan bulat terdekat
sqrt(x) Akar Pangkat dua	rem(x) Sisa setelah pembagian
log(x) Logaritma natural	angle(x) Sudut fase
log10(x) Logaritma umum	conj(x) Konjugat kompleks

Selain fungsi dasar, MATLAB menyertakan sejumlah nilai konstanta yang telah ditentukan sebelumnya. Daftar nilai yang paling umum diberikan pada Tabel 1.3

Tabel 1.3: Nilai konstanta yang telah ditentukan sebelumnya

pi	Angka π, π = 3,14159 . . .
i,j	Satuan imajiner i, √ −1
Inf	Tak terhingga, ∞
NaN	Bukan sebuah angka

Contoh:
Di sini dilustrasikan beberapa contoh umum yang berkaitan dengan fungsi dasar sebelumnya didefinisikan. Sebagai contoh pertama, nilai ekspresi y = e −a sin(x) + 10√y, for a = 5, x = 2, dan y = 8 dihitung dengan

```
>> a = 5; x = 2; y = 8;
>> y = exp(-a)*sin(x)+10*sqrt(y)
y =
28.2904
```

Contoh selanjutnya adalah

```
>> log(142)
jawab =
4.9558
>> log10(142)
jawab =
2.1523
```

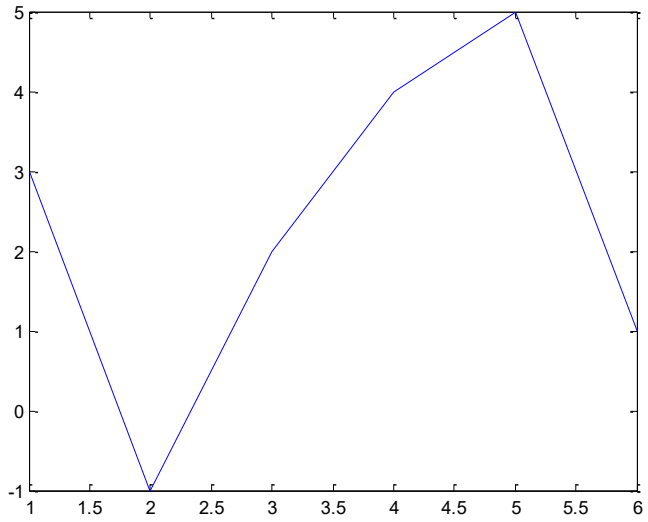
Perhatikan perbedaan antara logaritma natural log(x) dan logaritma desimal (basis 10) catatan10(x). Untuk menghitung sin(π/4) dan e 10, kita masukkan perintah berikut di MATLAB,

```
>> sin(pi/4)
jawab =
0,7071
>> exp(10)
jawab =
2.2026e+004
```

1.6 Membuat plot sederhana

Prosedur dasar pembuatan grafik MATLAB, misalnya dalam 2D, adalah mengambil vektor dengan koordinat x, x = (x1, . . , x_N), dan vektor dengan koordinat y, y = (y1, . . , y_N), Lokasi titik (xi, yi), dengan i = 1, 2, . . . , n lalu gabungkan keduanya dengan garis lurus. Perlu disiapkan x dan y dalam bentuk array yang identik; yaitu, x dan y keduanya merupakan larik baris atau larik kolom dengan panjang yang sama. Perintah MATLAB untuk memplot grafik adalah plot(x,y). Vektor x = (1, 2, 3, 4, 5, 6) dan y = (3, −1, 2, 4, 5, 1) menghasilkan gambar seperti pada Gambar 1.2.

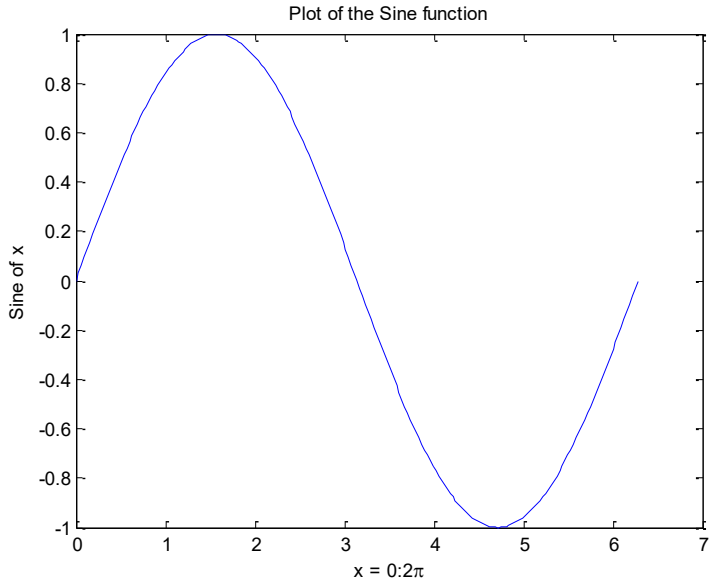
```
>> x = [1 2 3 4 5 6];
>> y = [3 -1 2 4 5 1];
>> plot(x,y)
```



Gambar 1.2: Plot vektor x dan y

1.7 Menambahkan judul, label sumbu, dan anotasi

MATLAB memungkinkan menambahkan label dan judul sumbu. Misalnya, dengan menggunakan grafik dari contoh sebelumnya, tambahkan label sumbu x dan y. Sekarang beri label pada sumbu dan tambahkan judul. Karakter pi menciptakan simbol π. Contoh plot 2D ditunjukkan pada Gambar 1.3.



Gambar 1.3: Plot fungsi Sinus

```
>> xlabel('x = 0:2\pi')
>> ylabel('Sine of x')
 >> title('Plot of the Sine function')
```

Warna satu kurva, secara default, adalah biru, namun warna lain juga dimungkinkan. Warna yang diinginkan ditunjukkan oleh argumen ketiga. Misalnya, warna merah dipilih berdasarkan plot(x,y,'r'). Perhatikan tanda kutip tunggal, ' ', di sekitar r.

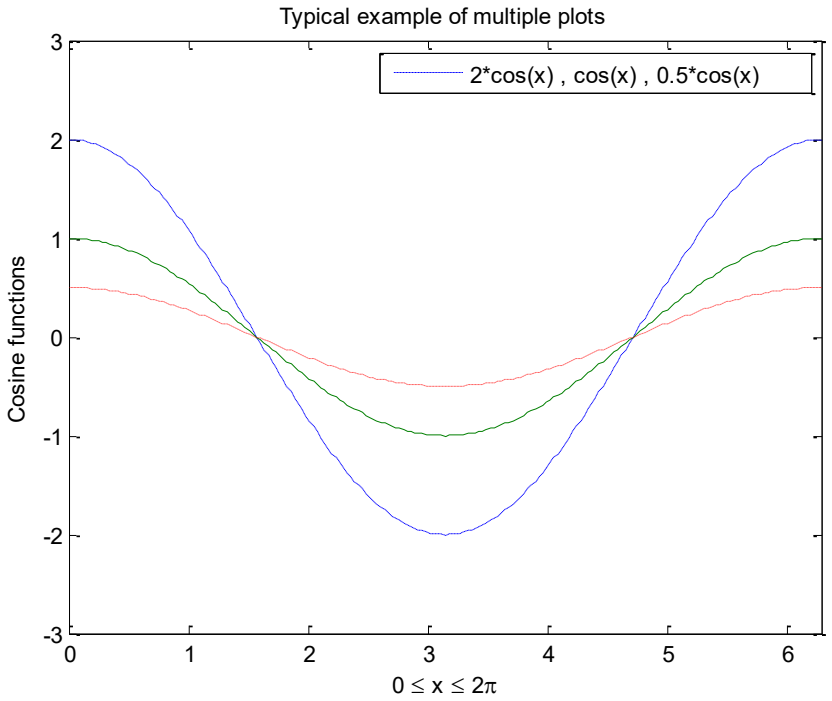
1.8 Beberapa kumpulan data dalam satu plot

Argumen pasangan ganda (x, y) membuat banyak grafik dengan satu panggilan untuk membuat plot. Misalnya, pernyataan-pernyataan ini memplot tiga fungsi terkait dari x: y1 = 2 cos(x), y2 = cos(x), dan y3 = 0,5 ∗ cos(x), dalam interval $0 \leq x \leq 2\pi$.

```
>> x = 0:pi/100:2*pi;
```

```
>> y1 = 2*cos(x);
>> y2 = cos(x);
>> y3 = 0.5*cos(x);
>> plot(x,y1,'--',x,y2,'-',x,y3,':')
>> xlabel('0 \leq x \leq 2\pi')
>> ylabel('Cosine functions')
>> legend('2*cos(x)','cos(x)','0.5*cos(x)')
>> title('Typical example of multiple plots')
>> axis([0 2*pi -3 3])
```

Hasil beberapa kumpulan data dalam satu plot grafik ditunjukkan pada Gambar 1.4.



Gambar 1.4: Contoh tipikal beberapa plot

Secara default, MATLAB menggunakan gaya garis dan warna untuk membedakan kumpulan data yang diplot dalam grafik. Namun, dapat diubah tampilan komponen grafis ini atau menambahkan anotasi ke grafik untuk membantu menjelaskan data untuk presentasi.

1.9 Menentukan gaya dan warna garis

Dimungkinkan untuk menentukan gaya garis, warna, dan penanda (misalnya, lingkaran, tanda plus, . . .) menggunakan perintah plot:

```
plot(x,y,'style_color_marker')
```

Dimana style_color_marker adalah triplet nilai dari Tabel 1.4. Untuk menemukan informasi tambahan, ketik plot bantuan atau plot dokumen.

Tabel 1.4: Atribut plot

Simbol Warna	Gaya Garis Simbol	Penanda Simbol
k Black	− Solid	+ Plus sign
r Red	−− Dashed	o Circle
b Blue	: Dotted	∗ Asterisk
g Green	−. Dash-dot	. Point
c Cyan	none No line	× Cross
m Magenta		s Square
y Yellow		d Diamond

Soal – soal Latihan:

1. Jelaskan apa itu MATLAB dan sebutkan beberapa aplikasi atau bidang yang sering menggunakan MATLAB dalam pemrograman dan analisis data.
2. Apa yang dimaksud dengan variabel dalam MATLAB? Berikan contoh cara mendeklarasikan variabel dalam MATLAB dan jelaskan aturan penamaan variabel yang benar di MATLAB.
3. MATLAB memiliki berbagai fungsi built-in untuk operasi matematika. Jelaskan bagaimana cara menggunakan fungsi dasar seperti sqrt(), sin(), dan log() di MATLAB untuk menghitung akar kuadrat, sinus, dan logaritma dari suatu nilai. Berikan contoh penggunaannya.
4. Dalam MATLAB, terdapat beberapa struktur kontrol seperti if, for, dan while. Jelaskan perbedaan antara struktur kontrol for dan while, dan berikan contoh situasi di mana masing-masing lebih tepat digunakan.
5. Jelaskan konsep array dan matrix dalam MATLAB. Apa perbedaan antara keduanya, dan bagaimana cara mendeklarasikan serta mengakses elemen-elemen dalam array dan matrix di MATLAB? Berikan contoh kode untuk masing-masing.

Bab 2
Pengenalan Pengolahan Citra

2.1 Images and pictures

Sebagian besar manusia adalah makhluk visual: kita sangat bergantung pada penglihatan kita untuk memahami dunia di sekitar kita. Kami tidak hanya melihat sesuatu untuk mengidentifikasi dan mengklasifikasikannya, namun kami dapat memindai perbedaannya, dan memperoleh "perasaan" secara keseluruhan untuk suatu adegan dengan pandangan sekilas. Manusia telah mengembangkan keterampilan visual yang sangat tepat: kita dapat mengidentifikasi wajah dalam sekejap; kita dapat membedakan warna; kita dapat memproses informasi visual dalam jumlah besar dengan sangat cepat. Namun, dunia terus bergerak: menatap sesuatu cukup lama dan hal itu akan berubah. Bahkan suatu bangunan kokoh yang besar, seperti bangunan atau gunung, akan berubah tampilannya tergantung waktu (siang atau malam); jumlah sinar matahari (cerah atau berawan), atau berbagai bayangan yang menimpanya. Kami prihatin dengan gambar tunggal: cuplikan, jika Anda suka, dari suatu pemandangan visual. Meskipun pemrosesan gambar dapat menangani perubahan pemandangan. Untuk tujuan ini, gambar adalah satu gambar yang mewakili sesuatu. Ini bisa berupa gambar seseorang, orang atau hewan, atau pemandangan luar ruangan, atau foto mikro dari komponen elektronik, atau hasil pencitraan medis. Sekalipun gambarnya tidak langsung dikenali, itu bukan sekadar kekaburan acak (Anwar & Priscylo, 2019).
Pemrosesan gambar melibatkan perubahan sifat suatu gambar menjadi keduanya
1. meningkatkan informasi gambarnya untuk interpretasi manusia,
2. menjadikannya lebih cocok untuk persepsi mesin otonom.

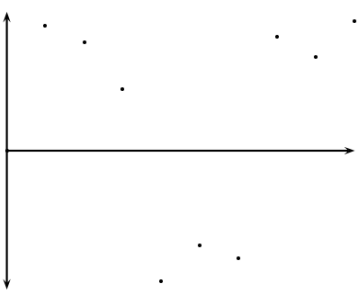
Kita akan membahas pemrosesan gambar digital, yang melibatkan penggunaan komputer untuk mengubah sifat gambar digital (lihat di bawah). Perlu disadari bahwa kedua aspek ini mewakili dua hal aspek yang terpisah tetapi sama pentingnya dari pemrosesan gambar. Prosedur yang memenuhi kondisi (1)—prosedur yang membuat gambar "terlihat lebih baik"—mungkin merupakan prosedur terburuk untuk memenuhi kondisi (2). Manusia menyukai gambarnya yang tajam, jelas, dan detail; mesin lebih memilih mereka gambar menjadi sederhana dan rapi.

2.2 Akuisisi Gambar dan pengambilan sampel

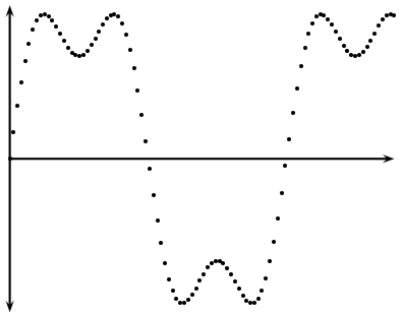
Sampling mengacu pada proses digitalisasi fungsi berkelanjutan. Misalnya kita mengambil fungsi:

$$y = \sin(x) + \frac{1}{3}\sin(3x)$$

dan mengambil sampelnya pada sepuluh nilai x yang berjarak sama. Titik sampel yang dihasilkan ditunjukkan pada gambar 2.1. Ini menunjukkan contoh undersampling, dimana jumlah titik tidak cukup untuk merekonstruksi fungsi. Misalkan kita mengambil sampel fungsi pada 100 titik, seperti yang ditunjukkan pada gambar 2.2. Sekarang dapat dengan jelas merekonstruksi fungsinya; semua propertinya dapat ditentukan dari pengambilan sampel ini. Untuk memastikan bahwa memiliki titik sampel yang cukup, mengharuskan periode pengambilan sampel tidak lebih dari setengah detail terbaik dalam fungsi diatas. Hal ini dikenal sebagai kriteria Nyquist, dan dapat dirumuskan lebih tepat dalam istilah "frekuensi", yang dibahas dalam bab berikutnya. Kriteria Nyquist dapat dinyatakan sebagai teorema sampling, yang pada dasarnya menyatakan bahwa suatu fungsi kontinu dapat dapat direkonstruksi dari sampelnya dengan ketentuan bahwa frekuensi pengambilan sampel setidaknya dua kali frekuensi maksimum dalam fungsi tersebut. Penjelasan formal tentang teorema ini disediakan oleh Castleman (Asiva Noor Rachmayani, 2015a).

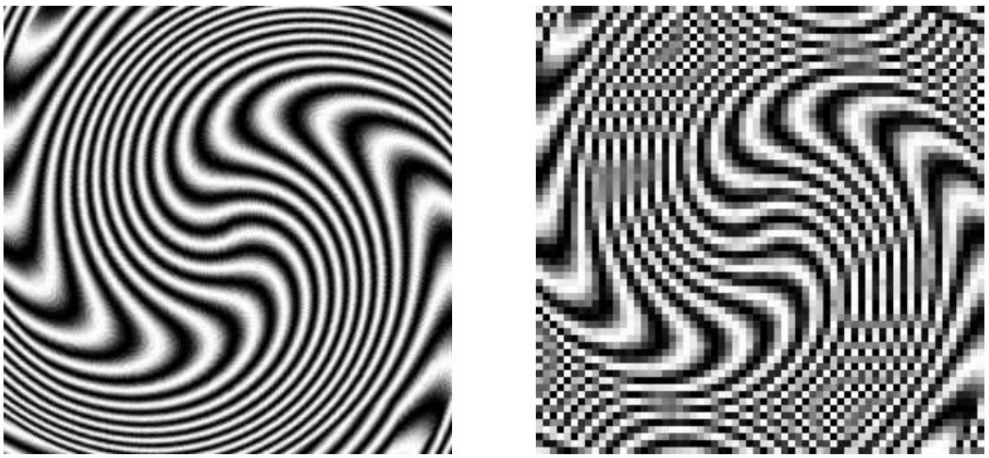


Gambar 2.1: Pengambilan sampel suatu fungsi—pengambilan sampel yang terlalu rendah



Gambar 2.2: Mengambil sampel suatu fungsi dengan lebih banyak titik

Pengambilan sampel suatu gambar sekali lagi mengharuskan pertimbangkan kriteria Nyquist, ketika kita menganggap suatu gambar sebagai fungsi kontinu dari dua variabel, dan ingin mengambil sampelnya untuk menghasilkan gambar digital. Contohnya ditunjukkan pada gambar 2.3 di mana gambar ditampilkan, dan kemudian dengan versi undersampled. Tepi bergerigi pada gambar yang diambil sampelnya adalah contoh aliasing. Kecepatan pengambilan sampel tentu saja akan mempengaruhi resolusi akhir gambar; kita membahasnya di bawah ini. Untuk mendapatkan gambar sampel (digital), kita dapat memulai dengan representasi berkelanjutan dari suatu pemandangan. Untuk melihat pemandangan tersebut, perlu merekam energi yang dipantulkan darinya; atau mungkin menggunakan cahaya tampak, atau sumber energi lainnya (Lévesque, 2014).



Pengambilan sampel yang benar; tidak ada aliasing Versi undersampled dengan aliasing

Gambar 2.3: Pengaruh pengambilan sampel

2.3 Greyscale images

Misalkan Anda sedang duduk di depan komputer dan telah memulai Matlab. Anda akan membuka jendela perintah Matlab, dan di dalamnya ada prompt Matlab. siap menerima perintah. Ketik perintahnya.

```
>> w=imread('madura.jpeg');
```

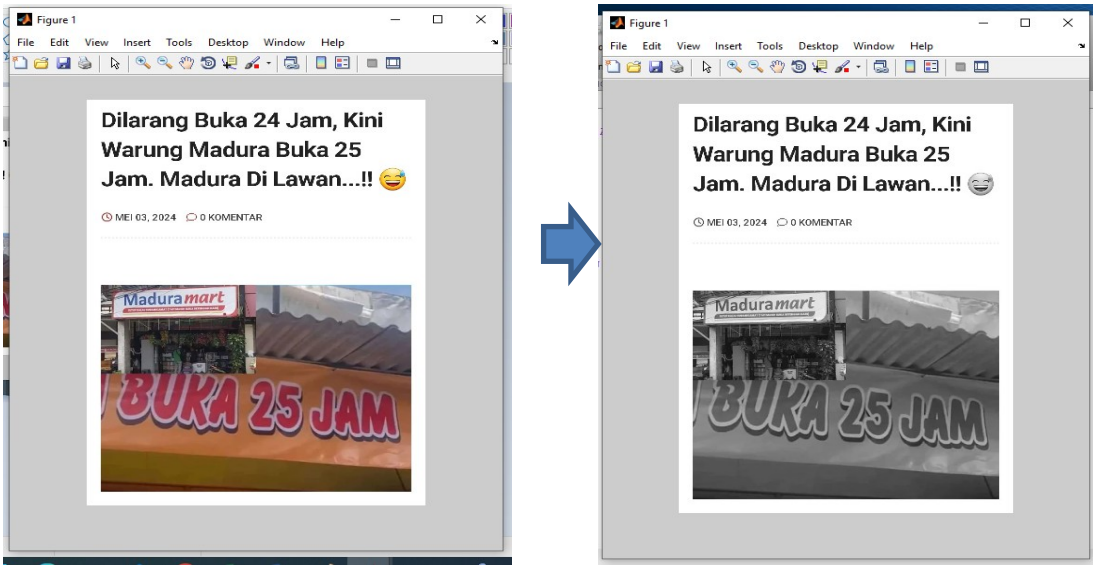
Ini mengambil nilai abu-abu dari semua piksel pada gambar skala abu-abu madura2.jpeg dan memasukkan semuanya ke dalam matriks w. Matriks w ini sekarang menjadi variabel Matlab, sehingga kita dapat melakukan berbagai operasi matriks padanya. Secara umum fungsi imread membaca nilai piksel dari file gambar, dan mengembalikan matriks dari semua nilai piksel (Saravanan, 2010). Dua hal yang perlu diperhatikan tentang perintah ini:

1. Berakhir dengan titik koma; hal ini berdampak pada tidak ditampilkannya hasil perintah ke layar. Hasil dari perintah khusus ini adalah matriks berukuran 256 x 256, atau dengan 65536 elemen, kami tidak benar-benar ingin semua nilainya ditampilkan.
2. Nama madura.jpeg diberikan dalam tanda kutip tunggal. Tanpa itu, Matlab akan berasumsi bahwa madura.jpeg adalah nama variabel, bukan nama file.

Sekarang kita tidak dapat menampilkan matriks ini sebagai gambar skala abu-abu:

```
>> figure,imshow(w),pixval on
```

Ini sebenarnya tiga perintah dalam satu baris. Matlab memungkinkan banyak perintah dimasukkan pada baris yang sama; menggunakan koma untuk memisahkan perintah yang berbeda. Tiga perintah yang kami gunakan di sini adalah:
figure, yang menciptakan gambar di layar. Gambar adalah jendela tempat objek grafis dapat ditempatkan. Objek dapat berupa gambar, atau berbagai jenis grafik.
imshow(g), yang menampilkan matriks g sebagai gambar
pixval aktif, yang mengaktifkan nilai piksel pada gambar. Ini adalah tampilan nilai abu-abu piksel pada gambar. Mereka muncul di bagian bawah gambar dalam form.
c x r = p
di mana c adalah nilai kolom dari piksel yang diberikan; r nilai barisnya, dan p nilai abu-abunya. Karena wombats.tif adalah gambar skala abu-abu 8-bit, nilai piksel muncul sebagai bilangan bulat dalam rentang tersebut.



Gambar 2.4: Gambar wombat dengan RGB to Grayscale

2.4 Fungsi imshow

Telah melihat bahwa jika x adalah matriks bertipe uint8, maka perintahnya imshow(x). akan menampilkan x sebagai gambar. Hal ini masuk akal, karena tipe data uint8 membatasi nilai menjadi bilangan bulat antara 0 dan 255. Namun, tidak semua matriks gambar digabungkan dengan baik ke dalam tipe data ini, dan banyak perintah pemrosesan gambar Matlab menghasilkan matriks keluaran yang bertipe ganda (Lesmana & Ramadhani, 2019). Kami memiliki dua pilihan dengan matriks jenis ini:

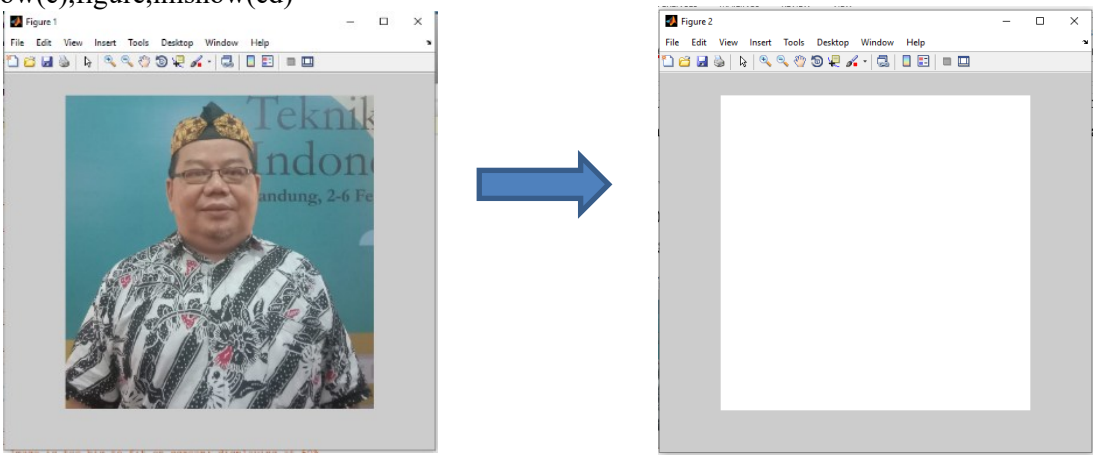
1. ubah menjadi ketik uint8 lalu tampilkan,
2. menampilkan matriks secara langsung.

Opsi kedua dimungkinkan karena imshow akan menampilkan matriks bertipe double sebagai gambar skala abu-abu selama elemen matriksnya antara 0 dan 1. Misalkan kita mengambil gambar dan mengubahnya menjadi tipe double:

```
>> c=imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.png');
>> cd=double(c);
```

>> imshow(c),figure,imshow(cd)

‘’



(a) Gambar aslinya (b) Setelah konversi ke type double

Gambar 2.5: Upaya konversi tipe data

Namun, seperti yang Anda lihat, gambar 2.5 (b) sama sekali tidak mirip dengan gambar aslinya! Hal ini karena untuk matriks bertipe double, fungsi imshow mengharapkan nilai antara 0 dan 1, dimana 0 ditampilkan sebagai hitam, dan 1 ditampilkan sebagai putih. Nilai dengan v dengan $0 < v < 1$ ditampilkan dalam skala abu-abu [255v]. Sebaliknya, nilai yang lebih besar dari 1 akan ditampilkan sebagai 1 (putih) dan nilai yang kurang dari 0 akan ditampilkan sebagai nol (hitam). Pada gambar karibu, setiap piksel mempunyai nilai lebih besar atau sama dengan 1 (sebenarnya nilai minimumnya adalah 21), sehingga setiap piksel akan ditampilkan berwarna putih. Untuk menampilkan matriks cd, kita perlu menskalakannya ke rentang 0—1. Hal ini mudah dilakukan hanya dengan membagi semua nilai dengan 255:

>> imshow(cd/255)

dan hasilnya adalah gambar karibu seperti pada gambar 2.5 (a).

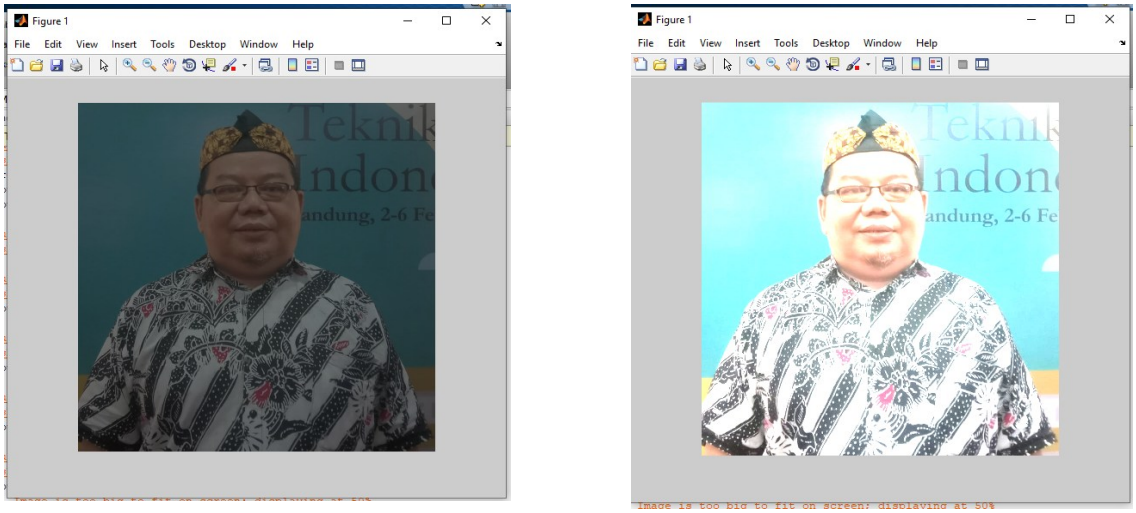
Kita dapat memvariasikan tampilan dengan mengubah skala matriks. Hasil dari perintah:

>> imshow(cd/512)

>> imshow(cd/128)

ditunjukkan pada gambar 2.6.

Membagi dengan 512 akan membuat gambar menjadi lebih gelap, karena semua nilai matriks kini berada di antara 0 dan 0,5, sehingga piksel paling terang pada gambar adalah abu-abu pertengahan. Membagi dengan 128 berarti rentangnya adalah 0—2, dan semua piksel dalam rentang 1—2 akan ditampilkan berwarna putih. Oleh karena itu, gambar memiliki tampilan yang terlalu terang dan pudar.



(a) Matriks cd dibagi 512 (b) Matriks cd dibagi 128

Gambar 2.6: Penskalaan dengan membagi matriks gambar dengan skalar

Tampilan hasil perintah yang keluarannya berupa matriks bertipe double dapat sangat dipengaruhi oleh pilihan faktor penskalaan yang bijaksana. Kita dapat mengubah gambar asli menjadi double dengan lebih baik menggunakan fungsi im2double. Ini menerapkan penskalaan yang benar sehingga nilai keluarannya antara 0 dan 1. Demikian perintahnya

```
>> cd=im2double(c);
>> imshow(cd)
```

akan menghasilkan gambar yang benar. Penting untuk membedakan kedua fungsi tersebut menjadi double dan im2double: double mengubah tipe data tetapi tidak mengubah nilai numerik; im2double mengubah tipe data numerik dan nilainya. Pengecualian tentu saja adalah jika gambar asli bertipe double, dalam hal ini im2double tidak melakukan apa pun. Walaupun perintah double tidak banyak berguna untuk menampilkan gambar secara langsung, perintah ini bisa sangat berguna untuk aritmatika gambar. Kita telah melihat contohnya di atas dengan penskalaan.

Sesuai dengan fungsi double dan im2double adalah fungsi uint8 dan im2uint8. Jika kita mengambil cd gambar bertipe double, dengan skala yang tepat sehingga semua elemen berada di antara 0 dan 1, kita dapat mengonversinya kembali menjadi gambar bertipe uint8 dengan dua cara:

```
>> c2=uint8(255*cd);
>> c3=im2uint8(cd);
```

Penggunaan im2uint8 lebih disukai; dibutuhkan tipe data lain sebagai masukan, dan selalu mengembalikan hasil yang benar.

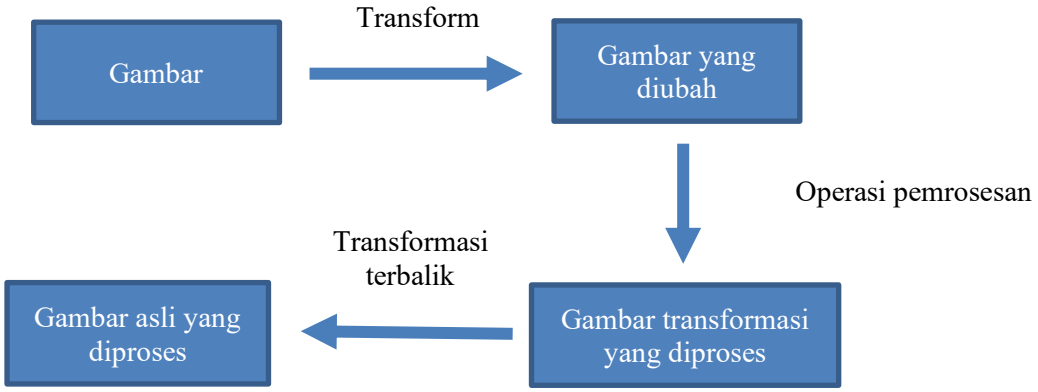
Soal-soal Latihan:

1. Jelaskan apa yang dimaksud dengan citra digital (image) dan sebutkan dua jenis format file citra yang umum digunakan dalam pengolahan citra.
2. Apa yang dimaksud dengan citra grayscale? Jelaskan perbedaan antara citra grayscale dengan citra berwarna, dan bagaimana proses konversi dari citra berwarna menjadi citra grayscale dilakukan.
3. Apa fungsi dari perintah imshow dalam MATLAB? Jelaskan bagaimana cara menggunakan imshow untuk menampilkan citra grayscale dan citra berwarna dalam MATLAB. Sertakan contoh kode untuk masing-masing.
4. Dalam MATLAB, bagaimana cara mengubah citra berwarna menjadi citra grayscale menggunakan fungsi yang ada? Berikan contoh kode untuk konversi citra berwarna menjadi grayscale dan tampilkan hasilnya menggunakan imshow.
5. Bagaimana cara melakukan penyesuaian kontras atau brightness pada citra grayscale di MATLAB? Jelaskan langkah-langkah yang perlu dilakukan dan cara menampilkan citra yang sudah dimodifikasi dengan menggunakan fungsi imshow.

Bab 3
Pemrosesan Titik

Setiap operasi pemrosesan gambar mengubah nilai abu-abu piksel. Namun, operasi pemrosesan gambar dapat dibagi menjadi tiga kelas berdasarkan informasi yang diperlukan untuk melakukan hal tersebut transformasi (Ikbal, 2017). Dari yang paling rumit hingga yang paling sederhana adalah:

1. Transformasi.
   Sebuah "transformasi" mewakili nilai piksel dalam beberapa bentuk lain, namun setara. Transformasi memungkinkan beberapa algoritma yang sangat efisien dan kuat, seperti yang akan kita lihat nanti. Kita dapat menganggap bahwa dalam menggunakan transformasi, seluruh gambar diproses sebagai satu blok besar. Hal ini dapat diilustrasikan dengan diagram yang ditunjukkan pada gambar 3.1.



Gambar 3.1: Skema pemrosesan transformasi

2. Pemrosesan Neighbourhood.
   Untuk mengubah tingkat keabuan suatu piksel, kita hanya perlu mengetahui nilai tingkat keabuan di lingkungan kecil piksel di sekitar piksel tersebut.
3. Operasi titik.
   Nilai abu-abu suatu piksel diubah tanpa sepengetahuan lingkungan sekitarnya.

Meskipun operasi titik adalah yang paling sederhana, operasi ini berisi beberapa operasi pemrosesan gambar yang paling kuat dan banyak digunakan. Mereka sangat berguna dalam pra-pemrosesan gambar, di mana gambar perlu dimodifikasi sebelum pekerjaan utama dilakukan.

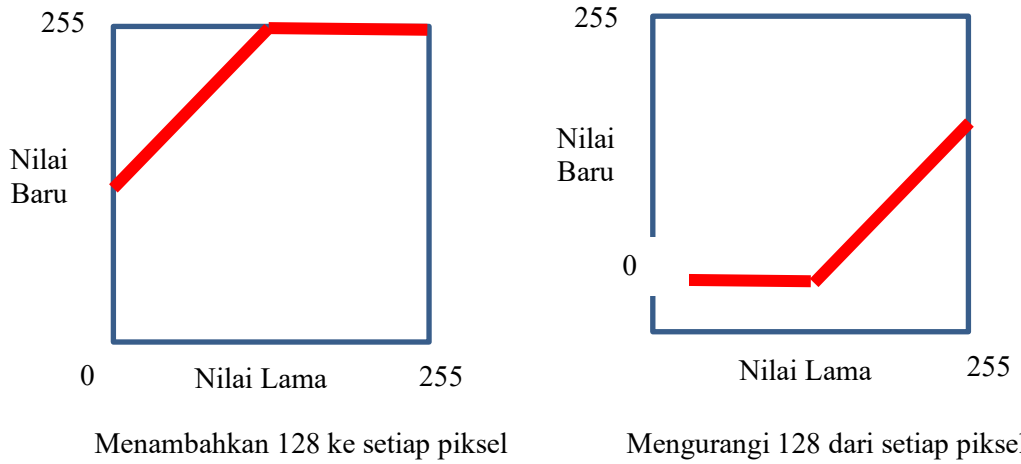
3.1 Operasi aritmatika

Operasi ini bertindak dengan menerapkan fungsi sederhana
y = f(x)
untuk setiap nilai abu-abu pada gambar. Jadi f(x) adalah fungsi yang memetakan rentang 0....255 ke dirinya sendiri. Fungsi sederhananya mencakup penambahan atau pengurangan nilai konstan pada setiap piksel:
$y = x \pm \mathrm{C}$
atau mengalikan setiap piksel dengan konstanta:
$y = C_x$
Dalam setiap kasus, kita mungkin harus mengubah keluarannya sedikit untuk memastikan bahwa hasilnya adalah bilangan bulat dalam rentang 0...255. Kita dapat melakukan ini dengan terlebih dahulu membulatkan hasilnya (jika perlu) untuk mendapatkan bilangan bulat, dan kemudian "memotong" nilainya dengan mengatur:

$$y = \begin{cases} 255. jika\ y > 2555 \\ 0, jika\ y < 0 \end{cases}$$

Untuk dapat memperoleh pemahaman tentang bagaimana operasi ini mempengaruhi gambar dengan memplot y = f(x). Gambar 2.2 menunjukkan hasil penjumlahan atau pengurangan 128 dari setiap piksel pada gambar. Perhatikan bahwa ketika kita menambahkan 128, semua nilai abu-abu dari 127 atau lebih besar akan dipetakan

ke 255. Dan ketika kita mengurangi 128, semua nilai abu-abu dari 128 atau kurang akan dipetakan ke 0. Dengan melihat grafik ini, untuk mengamati bahwa secara umum menambahkan konstanta akan mencerahkan gambar, dan mengurangi konstanta akan menggelapkannya.



Menambahkan 128 ke setiap piksel Mengurangi 128 dari setiap piksel

Gambar 3.2: Penjumlahan dan pengurangan suatu konstanta

Untuk dapat mengujinya pada gambar "foto hin. jpg, yang telah dapat dilihat pada gambar 1.4. Memulai dengan membaca gambar di:

```
b=imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.png');
>> whos b
  Name        Size              Bytes  Class    Attributes

  b         845x829x3         2101515  uint8
```

3.2 OPERASI ARITMATIK

Inti dari perintah kedua adalah mencari tipe data numerik b; itu uint8. Tipe data unit8 digunakan untuk penyimpanan data saja; kita tidak dapat melakukan operasi aritmatika.

```
>> b1=b+128
```

Atau

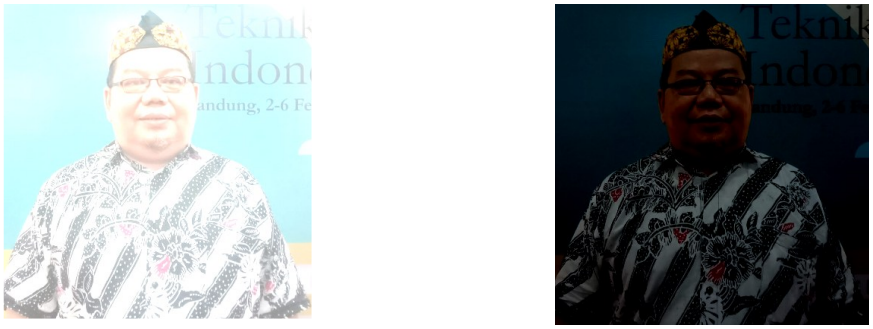
```
 >> b1=imadd(b,128);
```

Pengurangan serupa; kita dapat mengubah matriks menjadi dua kali lipat, atau menggunakan fungsi imsubtract:

```
>> b2=imsubtract(b,128);
```

Dan sekarang dapat melihatnya:

```
>> imshow(b1),figure,imshow(b2)
```



b1: Ditambahkan 128 b2: Dikurangi 128

Gambar 3.3: Operasi aritmatika pada suatu gambar: menambah atau mengurangi suatu konstanta

Soal-soal Latihan:

1. Jelaskan apa yang dimaksud dengan operasi matematika titik (point operations) dalam pemrosesan citra. Berikan dua contoh operasi matematika titik yang umum digunakan dalam pengolahan citra.
2. Pada pengolahan citra, salah satu operasi titik yang sering digunakan adalah penambahan kontras. Jelaskan bagaimana penambahan kontras pada citra grayscale dilakukan secara matematis dan berikan contoh kode untuk meningkatkan kontras citra di MATLAB.
3. Operasi pengurangan intensitas pada citra grayscale dapat digunakan untuk menggelapkan citra. Jelaskan bagaimana operasi pengurangan intensitas dilakukan pada citra grayscale dan berikan contoh penggunaan operasi ini di MATLAB.
4. Fungsi logaritma sering digunakan dalam pengolahan citra untuk meningkatkan detail pada area gelap citra. Jelaskan bagaimana operasi logaritma diterapkan pada citra dan sebutkan bagaimana efeknya terhadap citra tersebut. Berikan contoh kode untuk menerapkan operasi logaritma pada citra grayscale di MATLAB.
5. Salah satu operasi matematika titik adalah normalisasi citra, yang bertujuan untuk menyesuaikan nilai intensitas piksel citra agar berada dalam rentang tertentu. Jelaskan bagaimana cara melakukan normalisasi citra grayscale dengan rentang intensitas antara 0 dan 1, dan berikan contoh implementasinya di MATLAB.

Bab 4
Kuantitas dan Kualitas Citra

Kuantitas dan kualitas citra adalah dua aspek penting dalam pengolahan citra (image processing) yang mempengaruhi bagaimana citra tersebut dianalisis, disimpan, dan diproses.

4.1 Kuantitas Citra

Kuantitas citra merujuk pada jumlah data yang terdapat dalam citra tersebut (Ummah, 2019). Dalam konteks citra digital, kuantitas seringkali berhubungan dengan dua hal berikut:

- Resolusi Citra: Ini merujuk pada jumlah piksel yang membentuk citra. Citra dengan resolusi tinggi memiliki lebih banyak piksel, yang berarti lebih banyak data yang terkandung dalam citra tersebut. Resolusi ini sering diukur dalam satuan piksel per inci (PPI) atau dots per inch (DPI).
- Ukuran File Citra: Ini mengacu pada ukuran penyimpanan citra dalam perangkat keras (misalnya dalam MB atau GB). Semakin banyak piksel dan semakin kompleks warna yang digunakan dalam citra, semakin besar ukuran file citra tersebut.

Kuantitas citra berhubungan langsung dengan kemampuan untuk menangkap detail-detail halus dalam citra tersebut. Misalnya, citra dengan resolusi tinggi (kuantitas tinggi) dapat menyimpan lebih banyak informasi, seperti objek yang lebih kecil atau detail tekstur, dibandingkan dengan citra dengan resolusi rendah.

Untuk mengetahui kuantitas citra menggunakan MATLAB, kita dapat mengukur beberapa parameter yang terkait dengan kuantitas, seperti jumlah piksel, ukuran file, dan resolusi citra. Berikut adalah beberapa langkah untuk mengetahui kuantitas citra di MATLAB:

1. Mengetahui Jumlah Piksel dalam Citra

Jumlah piksel dalam citra dapat dihitung dengan mengalikan dimensi citra (tinggi dan lebar citra). Misalnya, citra berukuran 256x256 piksel berarti ada 65.536 piksel.

```
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg');
% Menghitung jumlah piksel
[tinggi, lebar, saluran] = size(citra);
% Jumlah total piksel
total_piksel = tinggi * lebar;
disp(['Jumlah piksel: ', num2str(total_piksel)]);
```

Hasil:

```
>> membaca_citra
Jumlah piksel: 700505
```

Catatan:

- imread digunakan untuk membaca citra dari file.
- size(citra) memberikan dimensi citra dalam bentuk tinggi, lebar, dan jumlah saluran (misalnya RGB memiliki 3 saluran).

2. Mengetahui Ukuran File Citra

Ukuran file citra bisa dihitung dengan menggunakan fungsi dir, yang memberi informasi tentang file yang ada di sistem.

```
% Mendapatkan informasi file
info = dir('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg');
% Ukuran file dalam byte
ukuran_file = info.bytes;
disp(['Ukuran file citra: ', num2str(ukuran_file), ' bytes']);
```

Hasil:

```
informasi_citra
Ukuran file citra: 190077 bytes
```

Catatan:

- Fungsi dir memberikan informasi tentang file, dan info.bytes akan menunjukkan ukuran file dalam byte.

3. Mengetahui Resolusi Citra
Resolusi citra dapat diukur menggunakan metadata citra jika informasi DPI (dots per inch) tersedia. Di MATLAB, Anda bisa menggunakan fungsi imfinfo untuk mengambil informasi ini.

```
% Mendapatkan informasi metadata citra
info = imfinfo('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\madura2.1.tif');
% Menampilkan DPI (dots per inch)
disp(['Resolusi citra: ', num2str(info.XResolution), ' x ', num2str(info.YResolution), ' DPI']);
```

Hasil:

```
>> resolusi_citra
Resolusi citra: 96 x 96 DPI
```

Catatan:

- Fungsi imfinfo memberikan informasi terkait metadata citra, seperti resolusi, ukuran, tipe file, dan lainnya. XResolution dan YResolution menunjukkan resolusi citra dalam satuan DPI.

Contoh Kode Lengkap

```
% Membaca citra
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\madura2.1.tif');
% Menghitung jumlah piksel
[tinggi, lebar, saluran] = size(citra);
total_piksel = tinggi * lebar;
disp(['Jumlah piksel: ', num2str(total_piksel)]);
% Mendapatkan informasi file menggunakan dir
file_info = dir('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\madura2.1.tif');  % Ganti dengan path file citra Anda
% Mengakses ukuran file dalam byte
ukuran_file = file_info.bytes;
% Menampilkan ukuran file
disp(['Ukuran file citra: ', num2str(ukuran_file), ' bytes']);
% Membaca informasi metadata citra
info = imfinfo('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\madura2.1.tif');  % Ganti dengan path file citra Anda
% Mengakses dimensi citra
disp(['Lebar citra: ', num2str(info.Width), ' piksel']);
disp(['Tinggi citra: ', num2str(info.Height), ' piksel']);
```

Hasil:

```
>> resolusi_citra_kode_lengkap
Jumlah piksel: 1630800
Ukuran file citra: 1296794 bytes
Lebar citra: 1080 piksel
Tinggi citra: 1510 piksel
```

catatan:

- Jumlah Piksel: Menunjukkan berapa banyak data yang ada dalam citra.
- Ukuran File: Memberikan gambaran seberapa besar citra dalam penyimpanan.
- Resolusi: Mengukur ketajaman citra dalam satuan DPI, yang sering kali terkait dengan kualitas cetak citra.

4.2 Kualitas Citra

Kualitas citra merujuk pada seberapa baik citra tersebut menggambarkan objek atau pemandangan yang ditangkap, dalam hal ketajaman, kontras, warna, dan detail (Sugiarti, 2018). Beberapa faktor yang mempengaruhi kualitas citra antara lain:

- Kejernihan dan Ketajaman (Sharpness): Mengacu pada tingkat kejelasan citra. Citra yang tajam memiliki garis tepi yang jelas dan objek-objek yang mudah dikenali. Kualitas rendah biasanya menunjukkan citra buram atau kabur.
- Kontras: Merujuk pada perbedaan antara area terang dan gelap dalam citra. Kontras yang baik akan menghasilkan citra dengan variasi yang lebih jelas antara objek dan latar belakang.
- Warna: Warna juga merupakan elemen penting dalam kualitas citra. Citra yang memiliki kualitas warna yang baik akan menghasilkan reproduksi warna yang akurat dan realistis, sementara citra

dengan kualitas rendah bisa tampak pudar atau terdistorsi.
- Noise: Noise adalah gangguan atau distorsi yang dapat mengurangi kualitas citra, sering kali terlihat dalam bentuk butiran atau titik acak yang tidak diinginkan pada citra.

Hubungan antara Kuantitas dan Kualitas Citra

- Kuantitas tidak selalu menjamin kualitas: Citra dengan resolusi tinggi tidak selalu berarti citra tersebut memiliki kualitas yang baik. Kualitas juga dipengaruhi oleh faktor lain seperti pencahayaan, lensa, teknik pengambilan gambar, dan proses pengolahan citra.
- Optimalisasi antara keduanya: Dalam pengolahan citra, penting untuk mencapai keseimbangan antara kuantitas dan kualitas. Misalnya, meningkatkan resolusi citra (kuantitas) tanpa mengorbankan kualitas visual bisa memerlukan teknik kompresi atau pengolahan citra tertentu.

Dengan kata lain, meskipun kuantitas citra berfokus pada jumlah data yang ada, kualitas citra lebih berfokus pada bagaimana data tersebut disusun dan ditampilkan sehingga menghasilkan citra yang jelas, tajam, dan akurat.

Untuk mengetahui kualitas citra menggunakan MATLAB, kita dapat memeriksa beberapa parameter yang umum digunakan dalam evaluasi kualitas citra. Kualitas citra biasanya mencakup aspek seperti ketajaman (sharpness), kontras, dan noise. Berikut adalah beberapa langkah untuk mengevaluasi kualitas citra di MATLAB:

a. Ketajaman (Sharpness) Citra

Ketajaman citra mengacu pada seberapa jelas objek-objek dalam citra terlihat. Citra yang tajam memiliki detail yang jelas, sedangkan citra yang buram memiliki detail yang kabur. Salah satu cara untuk mengukur ketajaman adalah dengan menghitung **gradient** citra, yang mengukur perubahan intensitas piksel.

Cara Menghitung Ketajaman (Sharpness):

```
% Membaca citra
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg');
% Jika gambar berwarna (RGB), ubah menjadi grayscale
if size(citra, 3) == 3
   citra = rgb2gray(citra);
end
% Menghitung gradien citra menggunakan operator Sobel
% Sobel operator untuk arah X
sobel_x = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
% Sobel operator untuk arah Y
sobel_y = [-1, -2, -1; 0, 0, 0; 1, 2, 1];
% Konversi citra ke tipe double untuk keperluan perhitungan
citra = double(citra);
% Hitung gradien citra pada arah X dan Y menggunakan filter Sobel
gradient_x = conv2(citra, sobel_x, 'same');
gradient_y = conv2(citra, sobel_y, 'same');
% Pastikan gradien juga bertipe double
gradient_x = double(gradient_x);
gradient_y = double(gradient_y);
% Hitung ketajaman citra berdasarkan gradien
% Menggunakan sum tanpa 'all'
magnitude = sqrt(gradient_x.^2 + gradient_y.^2);  % Hitung magnitudo gradien
sharpness = sum(magnitude(:));  % Jumlahkan semua elemen magnitudo
% Tampilkan hasil ketajaman
disp(['Ketajaman citra: ', num2str(sharpness)]);
```

Hasil:

```
>> Ketajaman_citra
Ketajaman citra: 40624265.8122
```

Catatan:

- **Gradient Sobel** digunakan untuk menghitung perubahan intensitas pada setiap piksel. Citra yang tajam memiliki perubahan intensitas yang lebih besar, sedangkan citra buram memiliki perubahan yang lebih kecil.

- **sharpness** adalah jumlah magnitude gradient, semakin tinggi nilai ini, semakin tajam citra tersebut.

b. Kontras Citra

Kontras citra mengukur perbedaan antara area terang dan gelap dalam citra. Citra dengan kontras tinggi memiliki perbedaan yang jelas antara objek dan latar belakang, sementara citra dengan kontras rendah terlihat datar dan kurang jelas.

Cara Menghitung Kontras:

```
% Membaca citra
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg');
% Jika gambar berwarna (RGB), ubah menjadi grayscale
if size(citra, 3) == 3
   citra = rgb2gray(citra);
end
% Menghitung gradien citra menggunakan operator Sobel
% Sobel operator untuk arah X
sobel_x = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
% Sobel operator untuk arah Y
sobel_y = [-1, -2, -1; 0, 0, 0; 1, 2, 1];
% Konversi citra ke tipe double untuk keperluan perhitungan
citra = double(citra);
% Menghitung kontras citra
citra_double = double(citra);
kontras = std(citra_double(:));
disp(['Nilai kontras citra: ', num2str(kontras)]);
```

Hasil:

```
>> kontras_citra
Nilai kontras citra: 37.6536
```

Catatan:

- **std(citra_double(:))** menghitung standar deviasi dari nilai intensitas piksel. Standar deviasi yang tinggi menunjukkan kontras yang tinggi, sedangkan standar deviasi yang rendah menunjukkan kontras rendah

c. Mengukur Noise dalam Citra

Noise adalah gangguan yang muncul dalam citra dan mengurangi kualitas visual. Salah satu cara untuk mengukur noise adalah dengan menggunakan **rasio signal-to-noise (SNR)**. Jika citra mengandung banyak noise, rasio SNR akan rendah.

Cara Menghitung SNR (Signal-to-Noise Ratio):

```
% Membaca citra
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg');
% Jika gambar berwarna (RGB), ubah menjadi grayscale
if size(citra, 3) == 3
   citra = rgb2gray(citra);
end
% Membaca citra asli dan citra dengan noise
citra_noisy = imnoise(citra, 'gaussian', 0, 0.01); % Menambahkan noise Gaussian
% Menghitung SNR
signal_power = sum(double(citra(:)).^2) / numel(citra);
noise_power = sum((double(citra(:)) - double(citra_noisy(:))).^2) / numel(citra);
SNR = 10 * log10(signal_power / noise_power);
disp(['Nilai SNR citra: ', num2str(SNR), ' dB']);
```

Hasil:

```
>> noise_citra
Nilai SNR citra: 14.451 dB
```

Catatan:

- **Signal Power** dihitung dengan mengkuadratkan intensitas piksel dan menghitung rata-ratanya.
- **Noise Power** dihitung dengan menghitung selisih antara citra asli dan citra dengan noise, kemudian

mengkuadratkan selisih tersebut dan menghitung rata-ratanya.

- **SNR** yang lebih tinggi menunjukkan kualitas citra yang lebih baik, karena noise yang lebih rendah.

d. Struktur Similarity Index (SSIM)

SSIM adalah metrik yang lebih canggih untuk mengevaluasi kualitas citra dengan membandingkan citra asli dan citra terkompresi atau citra yang telah terdistorsi. Nilai SSIM berkisar antara 0 (sangat buruk) hingga 1 (sangat baik).

Cara Menghitung SSIM:

```
% Membaca citra asli dan citra yang terdistorsi
citra_asli = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg');  % Ganti dengan nama file citra asli
citra_terdistorsi = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin_d.jpg');  % Ganti dengan nama file citra terdistorsi
 % Mengonversi citra dari uint8 ke double (rentang [0, 1])
citra_asli = im2double(citra_asli);
citra_terdistorsi = im2double(citra_terdistorsi);
 % Memastikan kedua citra memiliki ukuran yang sama
if size(citra_asli) ~= size(citra_terdistorsi)
    error('Ukuran citra asli dan citra terdistorsi harus sama!');
end
 % Parameter SSIM
C1 = (0.01 * 255) ^ 2;
C2 = (0.03 * 255) ^ 2;
 % Membuat kernel Gaussian (filter untuk rata-rata dan varians)
h = fspecial('gaussian', [11, 11], 1.5);  % Filter Gaussian 11x11 dengan sigma = 1.5
 % Menghitung rata-rata dan varians dari kedua citra
mu_x = imfilter(citra_asli, h, 'replicate');  % Rata-rata citra asli
mu_y = imfilter(citra_terdistorsi, h, 'replicate');  % Rata-rata citra terdistorsi
 sigma_x = imfilter(citra_asli .^ 2, h, 'replicate') - mu_x .^ 2;  % Varians citra asli
sigma_y = imfilter(citra_terdistorsi .^ 2, h, 'replicate') - mu_y .^ 2;  % Varians citra terdistorsi
sigma_xy = imfilter(citra_asli .* citra_terdistorsi, h, 'replicate') - mu_x .* mu_y;  % Kovarians antara citra asli dan citra terdistorsi
 % Menghitung SSIM
ssim_map = ((2 * mu_x .* mu_y + C1) .* (2 * sigma_xy + C2)) ./ ((mu_x .^ 2 + mu_y .^ 2 + C1) .* (sigma_x + sigma_y + C2));
ssim_value = mean(ssim_map(:));  % Nilai rata-rata SSIM
 % Menampilkan hasil
disp(['Nilai SSIM: ', num2str(ssim_value)]);
 % Menampilkan peta SSIM untuk visualisasi lebih lanjut
```

Hasil:

```
>> ssim_citra
Nilai SSIM: 0.99959
Warning: Image is too big to fit on screen; displaying at 50%
> In imuitools\private\initSize at 71
  In imshow at 282
  In ssim_citra at 38
```

Catatan:

- **ssim** menghitung indeks kesamaan struktur antara dua citra. Semakin tinggi nilai SSIM, semakin mirip citra yang diuji dengan citra referensi.

Soal-soal Latihan:

1. Jelaskan apa yang dimaksud dengan kuantitas citra dalam pengolahan citra digital. Apa saja faktor-faktor yang memengaruhi kuantitas citra dalam proses digitalisasi gambar?
2. Apa yang dimaksud dengan resolusi citra? Jelaskan perbedaan antara resolusi spasial dan resolusi kedalaman warna (bit depth) pada citra digital.
3. Sebutkan dan jelaskan dua teknik yang dapat digunakan untuk meningkatkan kualitas citra, serta

bagaimana teknik tersebut mempengaruhi citra yang dihasilkan.

4. Apa yang dimaksud dengan kompresi citra dan bagaimana kompresi dapat memengaruhi kualitas citra? Jelaskan perbedaan antara kompresi lossy dan lossless dalam konteks citra digital.
5. Jelaskan hubungan antara ukuran citra (jumlah piksel) dan kualitas citra. Bagaimana pengaruh ukuran citra terhadap kualitas visual dan ukuran file citra tersebut? Berikan contoh di mana peningkatan ukuran citra dapat meningkatkan kualitas, dan kapan pengurangan ukuran citra dapat berguna.

Bab 5
Jenis-Jenis Citra Dan Konversi Citra

5.1 Jenis-jenis Citra

Citra dapat diklasifikasikan berdasarkan berbagai aspek, seperti dimensi, format, warna, dan konten (Irianto, 2016). Berikut adalah beberapa jenis citra yang sering digunakan dalam pengolahan citra:

a. Citra Grayscale (Hitam-Putih)

Citra grayscale adalah citra yang hanya memiliki dua komponen warna utama, yaitu hitam dan putih, dengan berbagai tingkat keabuan di antaranya. Citra ini biasanya menggambarkan intensitas cahaya atau tingkat kecerahan dari objek dalam citra. Setiap piksel dalam citra grayscale memiliki nilai intensitas yang menunjukkan seberapa terang atau gelap piksel tersebut. Nilai ini biasanya diukur dalam skala dari 0 hingga 255, di mana:

0 melambangkan warna hitam (intensitas terendah),

255 melambangkan warna putih (intensitas tertinggi),

Nilai di antaranya melambangkan tingkat keabuan yang bervariasi.

Citra grayscale sering digunakan dalam berbagai aplikasi pengolahan citra dan analisis visual, karena lebih sederhana dibandingkan citra berwarna dan lebih mudah diproses.

Ciri-ciri Citra Grayscale

Ukuran Data Lebih Kecil: Karena hanya menggunakan satu kanal warna (terutama intensitas), ukuran data citra grayscale lebih kecil dibandingkan citra berwarna (RGB).

Lebih Cepat Diproses: Dalam pengolahan citra, citra grayscale lebih cepat diproses karena hanya membutuhkan satu nilai intensitas untuk setiap piksel.

Berikut adalah contoh kode MATLAB yang membaca sebuah citra berwarna, mengonversinya menjadi citra grayscale, dan kemudian menampilkannya.

```
% Membaca citra berwarna
citra_warna = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra berwarna ke citra grayscale
citra_grayscale = rgb2gray(citra_warna);
% Menampilkan citra grayscale
imshow(citra_grayscale);
title('Citra Grayscale');
% Menyimpan citra grayscale
imwrite(citra_grayscale, 'gambar_grayscale.jpg');
```

Hasil:

Gambar 5.1: Contoh Citra Grayscale

Penjelasan Kode:
imread: Fungsi ini digunakan untuk membaca citra dari file yang diberikan, dalam hal ini 'gambar.jpg'.
rgb2gray: Fungsi ini digunakan untuk mengonversi citra berwarna (RGB) menjadi citra grayscale.
imshow: Fungsi ini digunakan untuk menampilkan citra dalam jendela gambar MATLAB.
imwrite: Fungsi ini digunakan untuk menyimpan citra grayscale ke dalam file baru.

b. Citra Warna (RGB)
Citra warna (RGB) adalah jenis citra yang memiliki tiga kanal warna, yaitu Merah (Red), Hijau (Green), dan Biru (Blue), yang digabungkan untuk menghasilkan berbagai macam warna. Citra RGB menggunakan model warna aditif, di mana warna-warna dihasilkan dengan menambahkan intensitas ketiga warna primer tersebut. Setiap piksel dalam citra RGB memiliki tiga nilai intensitas, masing-masing untuk kanal Merah, Hijau, dan Biru.
Nilai intensitas untuk setiap kanal (R, G, B) berada dalam rentang 0 hingga 255, di mana:
0 berarti tidak ada intensitas (warna tersebut tidak ada).
255 berarti intensitas maksimum untuk warna tersebut.
Dengan mengombinasikan berbagai nilai intensitas untuk ketiga warna dasar ini, kita dapat menghasilkan warna yang sangat beragam.
Sebagai contoh:
(255, 0, 0) akan menghasilkan warna merah murni.
(0, 255, 0) akan menghasilkan warna hijau murni.
(0, 0, 255) akan menghasilkan warna biru murni.
(255, 255, 255) akan menghasilkan warna putih (semua warna hadir dengan intensitas maksimum).
(0, 0, 0) akan menghasilkan warna hitam (tidak ada warna sama sekali).
Ciri-ciri Citra RGB
Kompleksitas Lebih Tinggi: Karena setiap piksel memerlukan tiga komponen (R, G, B), citra RGB lebih besar dan lebih kompleks dibandingkan citra grayscale.
Lebih Realistis: Citra RGB mampu merepresentasikan warna alami dalam dunia nyata, seperti gambar foto atau video.
Berikut adalah contoh kode MATLAB yang membaca citra warna (RGB), menampilkannya, serta memodifikasi salah satu kanal warna (misalnya, hanya menampilkan kanal merah).
1. Membaca dan Menampilkan Citra RGB

```
% Membaca citra RGB
citrargb = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menampilkan citra RGB
imshow(citrargb);
title('Citra RGB');
```

Hasil:

Gambar 5.2: Contoh Citra RGB

2. Memodifikasi dan Menampilkan Salah Satu Kanal (Merah, Hijau, Biru)
Untuk menampilkan hanya salah satu kanal warna (misalnya, Merah), kita dapat mengekstrak satu komponen dan menghapus komponen lainnya.

```
% Membaca citra RGB
citrargb = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menampilkan hanya kanal Merah
kanal_merah = citrargb; % Salin citra RGB
kanal_merah(:,:,2) = 0; % Set komponen hijau menjadi 0
kanal_merah(:,:,3) = 0; % Set komponen biru menjadi 0
imshow(kanal_merah);
title('Citra Hanya Kanal Merah');
```

Hasil:

Gambar 5.3: Modifikasi Citra RGB

3. Mengubah Intensity Citra RGB
Untuk mengubah intensitas salah satu kanal warna (misalnya, meningkatkan intensitas hijau), kita dapat menyesuaikan nilai setiap piksel pada kanal tersebut.

```
% Membaca citra RGB
citra_rgb = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Meningkatkan intensitas kanal Hijau
citra_rgb_ubah = citra_rgb;
citra_rgb_ubah(:,:,2) = citra_rgb_ubah(:,:,2) + 50; % Menambahkan 50 ke kanal hijau
% Pastikan nilai tidak melebihi 255
citra_rgb_ubah(:,:,2) = min(citra_rgb_ubah(:,:,2), 255);
imshow(citra_rgb_ubah);
title('Citra dengan Intensitas Hijau Ditingkatkan');
```

Hasil:

Gambar 5.4: Modifikasi Intensitas Citra RGB

4. Menyimpan Citra RGB yang Dimodifikasi
Setelah memodifikasi citra, maka bisa menyimpannya dengan fungsi imwrite.

```
% Membaca citra RGB
citra_rgb = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Meningkatkan intensitas kanal Hijau
citra_rgb_ubah = citra_rgb;
citra_rgb_ubah(:,:,2) = citra_rgb_ubah(:,:,2) + 50; % Menambahkan 50 ke kanal hijau
% Pastikan nilai tidak melebihi 255
citra_rgb_ubah(:,:,2) = min(citra_rgb_ubah(:,:,2), 255);
imshow(citra_rgb_ubah);
title('Citra dengan Intensitas Hijau Ditingkatkan');
% Menyimpan citra yang telah dimodifikasi
imwrite(citra_rgb_ubah, 'gambar_modifikasi.jpg');
```

c. Citra Biner (Binary)
Citra biner adalah citra yang hanya memiliki dua nilai piksel yang berbeda, yaitu **0** dan **1**, yang mewakili dua keadaan atau kondisi yang berbeda. Biasanya, nilai 0 digunakan untuk mewakili **warna hitam** (atau objek latar belakang), dan nilai 1 digunakan untuk mewakili **warna putih** (atau objek depan).
Citra biner digunakan dalam berbagai aplikasi pengolahan citra, seperti **deteksi tepi**, **segmentasi citra**, dan **analisis objek**. Citra ini lebih sederhana dibandingkan dengan citra grayscale atau RGB karena hanya membutuhkan satu kanal intensitas, yaitu 0 atau 1.
Ciri-ciri Citra Biner
Sederhana dan Ringan: Karena hanya membutuhkan dua nilai untuk setiap piksel, citra biner memiliki ukuran file yang jauh lebih kecil dibandingkan citra grayscale atau RGB.
Digunakan untuk Segmentasi: Citra biner sering digunakan dalam pengolahan citra untuk memisahkan objek dari latar belakang dalam proses **thresholding**.
Representasi Logis: Citra biner sering digunakan untuk memodelkan status atau kondisi logis, seperti objek yang ada (1) atau tidak ada (0).
Konversi Citra ke Citra Biner
Proses untuk mengonversi citra grayscale atau citra berwarna menjadi citra biner dilakukan dengan cara menentukan sebuah **ambang batas (threshold)**. Piksel yang memiliki nilai intensitas lebih besar dari threshold akan dipetakan menjadi **1** (putih), sementara yang lebih kecil dari threshold dipetakan menjadi **0** (hitam).
Berikut adalah contoh kode MATLAB yang menunjukkan bagaimana cara mengonversi citra berwarna atau citra grayscale menjadi citra biner dan bagaimana cara menampilkan hasilnya.
1. Membaca Citra, Mengonversi ke Grayscale, dan Mengonversi ke Citra Biner

```
% Membaca citra berwarna
citra_warna = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama
```

```
file gambar Anda
% Mengonversi citra berwarna ke citra grayscale
citra_grayscale = rgb2gray(citra_warna);
% Mengonversi citra grayscale menjadi citra biner
threshold = 128; % Ambang batas untuk thresholding (antara 0 hingga 255)
citrabiner = citra_grayscale > threshold;
% Menampilkan citra Warna
subplot(1, 3, 1);
imshow(citra_warna);
title('Citra Warna');
% Menampilkan citra grayscale
subplot(1, 3, 2);
imshow(citra_grayscale);
title('Citra Grayscale');
% Menampilkan citra biner
subplot(1, 3, 3);
imshow(citrabiner);
title('Citra Biner');
```

Hasil:

Gambar 5.5: Contoh Citra Warna, Grayscale dan Biner

Penjelasan Kode:
imread: Fungsi ini digunakan untuk membaca citra dari file (misalnya 'gambar.jpg').
rgb2gray: Fungsi ini mengonversi citra berwarna menjadi citra grayscale.
Thresholding: Pada baris citra_biner = citra_grayscale > threshold;, citra grayscale diubah menjadi citra biner berdasarkan ambang batas (threshold). Piksel yang lebih besar dari 128 akan bernilai 1 (putih), sementara yang lebih kecil atau sama dengan 128 akan bernilai 0 (hitam).
imshow: Fungsi ini digunakan untuk menampilkan citra. Di sini, dua gambar ditampilkan berdampingan: citra grayscale dan citra biner.

3. Menyimpan Citra Biner

Setelah konversi ke citra biner, kita dapat menyimpan citra tersebut menggunakan imwrite seperti berikut:

```
% Membaca citra berwarna
citra_warna = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra berwarna ke citra grayscale
citra_grayscale = rgb2gray(citra_warna);
% Mengonversi citra grayscale menjadi citra biner
threshold = 128; % Ambang batas untuk thresholding (antara 0 hingga 255)
citrabiner = citra_grayscale > threshold;
% Menampilkan citra Warna
subplot(1, 3, 1);
imshow(citra_warna);
title('Citra Warna');
% Menampilkan citra grayscale
subplot(1, 3, 2);
imshow(citra_grayscale);
```

```
title('Citra Grayscale');
% Menampilkan citra biner
subplot(1, 3, 3);
imshow(citrabiner);
title('Citra Biner');
% Menyimpan citra biner
imwrite(citrabiner, 'gambar_biner.jpg');
```

Aplikasi Citra Biner
Citra biner digunakan dalam berbagai aplikasi pengolahan citra, antara lain:
Deteksi Objek: Untuk memisahkan objek dari latar belakang dan menganalisis bentuk atau ukuran objek.
Pengenalan Pola: Citra biner sering digunakan dalam pengenalan pola untuk klasifikasi objek.
Analisis Bentuk dan Struktur: Menyederhanakan analisis bentuk dan struktur objek dalam citra.

d. Citra HSI (Hue, Saturation, Intensity)
Citra HSI (Hue, Saturation, Intensity) adalah representasi citra warna yang berbeda dengan representasi RGB (Red, Green, Blue). Dalam model warna HSI, warna suatu citra diwakili oleh tiga komponen utama:
Hue (H): Menunjukkan jenis warna, seperti merah, hijau, biru, dll. Nilainya diukur dalam derajat, biasanya dalam rentang 0° hingga 360°.
Saturation (S): Menunjukkan kejenuhan atau kemurnian warna. Semakin tinggi saturasi, semakin murni warna tersebut. Nilai saturasi berada dalam rentang 0 hingga 1, dengan 0 berarti abu-abu (tidak ada warna), dan 1 berarti warna murni.
Intensity (I): Menunjukkan kecerahan atau intensitas warna. Nilainya juga berada dalam rentang 0 hingga 1, dengan 0 berarti hitam (tidak ada cahaya), dan 1 berarti warna paling cerah.
Model HSI banyak digunakan dalam pengolahan citra karena lebih mendekati persepsi warna manusia, yang lebih sensitif terhadap komponen **Hue** dan **Saturation**, dan kurang sensitif terhadap **Intensity**.
Mengkonversi Citra RGB ke HSI
Untuk mengonversi citra RGB ke HSI, Anda bisa menggunakan rumus konversi berikut:
Hue (H) dihitung dengan mempertimbangkan perbedaan antara komponen RGB.
Saturation (S) dihitung berdasarkan perbedaan antara nilai maksimum dan minimum komponen RGB.
Intensity (I) adalah rata-rata dari nilai R, G, dan B.
Contoh Kode Matlab untuk Mengonversi Citra RGB ke HSI

```
% Membaca citra RGB
I_rgb = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra RGB ke HSI
I_hsi = rgb2hsi(I_rgb);
% Menampilkan hasil konversi HSI
hue = I_hsi(:,:,1); % Hue
saturation = I_hsi(:,:,2); % Saturation
intensity = I_hsi(:,:,3); % Intensity
% Menampilkan citra Hue, Saturation, dan Intensity
figure;
subplot(1,3,1); imshow(hue, []); title('Hue');
subplot(1,3,2); imshow(saturation, []); title('Saturation');
subplot(1,3,3); imshow(intensity, []); title('Intensity');
```

Hasil:

Gambar 5.6: Contoh Citra HSI

Pada kode di atas:
Citra RGB dibaca menggunakan imread.
Fungsi rgb2hsi digunakan untuk mengonversi citra RGB ke HSI.
Citra HSI dipisahkan menjadi tiga komponen (Hue, Saturation, dan Intensity) dan ditampilkan satu per satu.
Namun, perlu dicatat bahwa fungsi rgb2hsi tidak tersedia secara langsung di MATLAB, sehingga Anda harus mengimplementasikan konversi RGB ke HSI secara manual menggunakan rumus-rumus yang telah disebutkan di atas atau mencari implementasi lain yang tersedia dalam File Exchange MATLAB.

e. Citra CMYK (Cyan, Magenta, Yellow, Black)

Citra CMYK (Cyan, Magenta, Yellow, Black) adalah model warna yang digunakan terutama dalam pencetakan. Model ini terdiri dari empat komponen warna, yaitu Cyan (C), Magenta (M), Yellow (Y), dan Black (K). Setiap komponen warna mewakili proporsi dari tinta yang digunakan dalam proses pencetakan.

Penjelasan tentang Citra CMYK:

1. **Cyan (C)**: Warna biru kehijauan, digunakan untuk mencetak warna biru dan hijau.
2. **Magenta (M)**: Warna merah keunguan, digunakan untuk mencetak warna merah dan ungu.
3. **Yellow (Y)**: Warna kuning, digunakan untuk mencetak warna kuning.
4. **Black (K)**: Warna hitam, yang digunakan untuk memberikan kedalaman dan ketajaman pada gambar. Seringkali digunakan untuk memperdalam bayangan atau membuat teks lebih jelas.

Di dalam proses pencetakan, tinta-tinta tersebut dipakai secara bersama-sama untuk menghasilkan berbagai warna yang diinginkan. Menggunakan empat warna ini memungkinkan pencetakan yang lebih akurat daripada menggunakan model warna RGB (Red, Green, Blue), yang lebih banyak digunakan dalam tampilan layar.

Mengonversi dari RGB ke CMYK

Model warna RGB digunakan untuk menampilkan warna di layar, sementara CMYK digunakan untuk pencetakan. Oleh karena itu, untuk mencetak gambar dari data RGB, konversi perlu dilakukan.
Formula konversi dari RGB ke CMYK adalah sebagai berikut:

1. Normalisasi nilai RGB ke rentang [0, 1].
   - R' = R / 255
   - G' = G / 255
   - B' = B / 255
2. Temukan nilai **K** (key/black):
   - K = 1 - max(R', G', B')
3. Tentukan nilai **C**, **M**, dan **Y**:
   - C = (1 - R' - K) / (1 - K)
   - M = (1 - G' - K) / (1 - K)
   - Y = (1 - B' - K) / (1 - K)

Jika K = 1, maka C, M, dan Y semuanya adalah 0 (warna hitam murni).
Berikut adalah contoh program MATLAB yang mengonversi citra dari RGB ke CMYK:

```
% Membaca citra RGB
rgb_image = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Normalisasi citra RGB ke rentang [0, 1]
rgb_image = double(rgb_image) / 255;
% Inisialisasi variabel CMYK
[rows, cols, ~] = size(rgb_image);
cmyk_image = zeros(rows, cols, 4);
```

```
% Proses konversi RGB ke CMYK
for i = 1:rows
   for j = 1:cols
      R = rgb_image(i, j, 1);
      G = rgb_image(i, j, 2);
      B = rgb_image(i, j, 3);
      % Hitung K (Key/Black)
      K = 1 - max(R, max(G, B));
      if K < 1
         % Hitung C, M, Y jika K < 1
         C = (1 - R - K) / (1 - K);
         M = (1 - G - K) / (1 - K);
         Y = (1 - B - K) / (1 - K);
      else
         % Jika K = 1, set C, M, Y ke 0
         C = 0;
         M = 0;
         Y = 0;
      end
    % Simpan nilai CMYK
      cmyk_image(i, j, 1) = C;
      cmyk_image(i, j, 2) = M;
      cmyk_image(i, j, 3) = Y;
      cmyk_image(i, j, 4) = K;
   end
end
% Menampilkan hasil konversi
figure;
subplot(1,2,1);
imshow(rgb_image);
title('Citra RGB');
subplot(1,2,2);
imshow(cmyk_image(:,:,1:3)); % Menampilkan C, M, Y saja
title('Citra CMY (tanpa K)');
```

Hasil:

Gambar 5.7: Contoh Citra CMYK (Cyan, Magenta, Yellow, Black)

Penjelasan Program:

1. **Membaca Citra**: Citra RGB dibaca dengan menggunakan imread.
2. **Normalisasi**: Nilai piksel citra RGB dinormalisasi ke rentang [0, 1] untuk mempermudah perhitungan.
3. **Konversi RGB ke CMYK**: Program melakukan iterasi untuk setiap piksel citra dan mengonversinya ke dalam nilai CMYK dengan rumus yang telah disebutkan.
4. **Menampilkan Hasil**: Gambar RGB ditampilkan di sebelah kiri, dan gambar CMYK (hanya C, M, dan Y) ditampilkan di sebelah kanan.

Dengan cara ini, kita dapat melihat perbedaan antara citra RGB dan citra dalam format CMYK, yang berguna untuk tujuan pencetakan.

5.2 Konversi Citra

Konversi citra adalah proses mengubah citra dari satu format atau jenis ke format atau jenis lainnya (Noor Santi, 2011). Konversi ini sering kali dilakukan untuk memudahkan pemrosesan, analisis, atau untuk tujuan tertentu seperti kompresi, pengenalan pola, atau visualisasi.

a. Konversi Citra Berwarna ke Grayscale

Konversi citra berwarna (RGB) ke citra grayscale adalah proses yang mengubah citra yang memiliki tiga saluran warna (merah, hijau, biru) menjadi citra dengan satu saluran intensitas. Citra grayscale hanya memiliki satu nilai untuk setiap piksel yang merepresentasikan tingkat kecerahan atau intensitas cahaya, tanpa informasi warna.

Secara matematis, proses konversi dapat dilakukan dengan menghitung rata-rata tertimbang dari tiga saluran warna (RGB). Salah satu rumus yang sering digunakan adalah:

$Y=0.2989\times R + 0.5870\times G + 0.1140\times B$

Di mana:

- Y adalah nilai grayscale (intensitas cahaya).
- R, G, dan B adalah nilai untuk saluran merah, hijau, dan biru, masing-masing.
- Koefisien (0.2989, 0.5870, dan 0.1140) digunakan untuk menyesuaikan kontribusi relatif masing-masing warna terhadap intensitas terang.

Konversi citra berwarna ke grayscale sering dilakukan dalam pengolahan citra untuk menyederhanakan analisis atau memfokuskan pada struktur atau bentuk objek dalam citra tanpa gangguan warna. Beberapa aplikasi yang memanfaatkan citra grayscale adalah:

- Pengenalan pola
- Deteksi tepi
- Pemrosesan citra medis
- Pengolahan citra untuk pengenalan wajah

Berikut adalah contoh program MATLAB untuk mengonversi citra berwarna menjadi citra grayscale:

```
% Membaca citra berwarna
citra_rgb = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Konversi citra RGB ke grayscale
citra_grayscale = rgb2gray(citra_rgb);
% Menampilkan citra berwarna dan citra grayscale
subplot(1, 2, 1);
imshow(citra_rgb);
title('Citra Berwarna');
subplot(1, 2, 2);
imshow(citra_grayscale);
title('Citra Grayscale');
% Menyimpan citra grayscale
imwrite(citra_grayscale, 'gambar_grayscale.jpg');
```

Hasil:

Gambar 5.8: Contoh Konversi Citra darin warna ke grayscale

Penjelasan Program:

1. **Membaca Citra Berwarna**: Fungsi imread() digunakan untuk membaca citra berwarna dari file (misalnya 'gambar.jpg').
2. **Konversi ke Grayscale**: Fungsi rgb2gray() secara otomatis mengonversi citra RGB menjadi citra grayscale menggunakan rumus bobot yang telah dijelaskan.
3. **Menampilkan Citra**: Fungsi imshow() digunakan untuk menampilkan citra berwarna dan citra grayscale di layar. Fungsi subplot() digunakan untuk menampilkan kedua citra dalam satu jendela.
4. **Menyimpan Citra Grayscale**: Fungsi imwrite() digunakan untuk menyimpan citra grayscale ke dalam file baru (misalnya 'gambar_grayscale.jpg').

b. Konversi Citra RGB ke HSI

Konversi citra dari ruang warna RGB (Red, Green, Blue) ke ruang warna HSI (Hue, Saturation, Intensity) adalah proses yang sering digunakan dalam pengolahan citra untuk analisis warna dan citra. Sistem warna HSI lebih mendekati cara manusia memandang dan mengklasifikasikan warna, yang terdiri dari tiga komponen utama:

Hue (H): Merepresentasikan warna atau rona dari citra. Nilai hue menunjukkan jenis warna yang ada, mulai dari 0 hingga 360 derajat. Misalnya, 0° adalah merah, 120° adalah hijau, 240° adalah biru, dan nilai-nilai lainnya merepresentasikan warna-warna yang ada di antara warna-warna dasar ini.

Saturation (S): Mengindikasikan kejenuhan warna atau kemurnian warna. Saturasi tinggi berarti warna lebih murni (lebih kuat), sementara saturasi rendah berarti warna lebih pudar atau mendekati abu-abu.

Intensity (I): Menunjukkan tingkat kecerahan atau intensitas cahaya dari warna. Nilai intensitas berkisar antara 0 hingga 1 (atau 0 hingga 255, tergantung pada representasi skala warna).

Konversi dari RGB ke HSI melibatkan perhitungan matematis yang relatif lebih kompleks daripada konversi ke grayscale. Proses konversi ini terdiri dari beberapa tahap, yang melibatkan perhitungan komponen Hue, Saturation, dan Intensity berdasarkan nilai RGB.

Berikut adalah rumus dasar untuk konversi dari RGB ke HSI:

Intensity (I):

I=(R+G+B)/3

Saturation (S):

S=1− (3/(R+G+B) × min(R,G,B)

Jika R + G + B = 0, maka S = 0.

Hue (H): Hue dihitung dengan menggunakan rumus berikut, yang berbeda-beda tergantung pada hubungan antara nilai R, G, dan B:

Jika R ≥ G dan R ≥ B

$$H = Cos^{-1}\left(\frac{{}^{1}/_{2}\ x(R-G+R-B)}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right)$$

Jika G ≥ R dan G ≥ B

$$H = {}^{2\pi}/_{3}\, Cos^{-1}\left(\frac{{}^{1}/_{2}\ x\,(G-R+G-B)}{\sqrt{(G-R)^2+(G-B)(R-B)}}\right)$$

Jika B ≥ R dan B ≥ G

$$H = {}^{4\pi}/_{3}\, Cos^{-1}\left(\frac{{}^{1}/_{2}\ x\,(B-R+B-G)}{\sqrt{(B-R)^2+(B-G)(R-G)}}\right)$$

Berikut adalah program MATLAB untuk mengonversi citra RGB menjadi citra HSI:

```
function [H, S, I] = rgb2hsi(RGB)
  % Membaca citra RGB
  RGB = double(RGB) / 255;  % Normalisasi ke rentang 0-1
  R = RGB(:,:,1);
  G = RGB(:,:,2);
  B = RGB(:,:,3);
  % Menghitung Intensity (I)
  I = (R + G + B) / 3;
    % Menghitung Saturation (S)
```

```
    minRGB = min(RGB, [], 3);  % Nilai minimum dari R, G, B
    S = 1 - (3 ./ (R + G + B)) .* minRGB;
    S(R + G + B == 0) = 0;  % Jika R + G + B == 0, set S = 0
    % Menghitung Hue (H)
    num = 0.5 * ((R - G) + (R - B));
    den = sqrt((R - G).^2 + (R - B).*(G - B));
    theta = acos(num ./ den);
    % Menentukan H berdasarkan segmen warna
    H = theta;
    H(B > G) = 2*pi - H(B > G);  % Sesuaikan untuk warna biru dominan
    H = H / (2*pi);  % Normalisasi H ke rentang 0-1
    % Menampilkan hasil
    figure;
    subplot(1,3,1); imshow(RGB); title('Citra RGB');
    subplot(1,3,2); imshow(H, []); title('Hue (H)');
    subplot(1,3,3); imshow(S, []); title('Saturation (S)');
end
```

Penjelasan Program:

Membaca Citra: Citra RGB dibaca dan dinormalisasi ke rentang [0, 1].

Menghitung Intensity (I): Menggunakan rumus rata-rata komponen RGB.

Menghitung Saturation (S): Menggunakan rumus yang melibatkan nilai minimum dari RGB untuk menghitung tingkat kejenuhan.

Menghitung Hue (H): Menggunakan rumus yang melibatkan perhitungan sudut dengan menggunakan fungsi arccos.

Menampilkan Hasil: Program ini akan menampilkan citra asli (RGB), komponen Hue (H), dan komponen Saturation (S).

Penggunaan Program:

Untuk mengonversi citra RGB ke HSI, Anda bisa memanggil fungsi rgb2hsi() di MATLAB dengan citra RGB yang ingin diubah. Contoh penggunaan:

```
% Membaca citra RGB
I_rgb = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra RGB ke HSI
I_hsi = rgb2hsi(I_rgb);
% Menampilkan hasil konversi HSI
hue = I_hsi(:,:,1); % Hue
saturation = I_hsi(:,:,2); % Saturation
intensity = I_hsi(:,:,3); % Intensity
% Menampilkan citra Hue, Saturation, dan Intensity
figure;
subplot(1,4,1); imshow(I_rgb, []); title('Citra RGB');
subplot(1,4,2); imshow(hue, []); title('Hue');
subplot(1,4,3); imshow(saturation, []); title('Saturation');
subplot(1,4,4); imshow(intensity, []); title('Intensity');
```

Hasil:

Gambar 5.9: Contoh Konversi Citra darin warna ke HSI

c. Konversi Citra Grayscale ke Biner

Konversi citra grayscale ke citra biner adalah proses mengubah citra berwarna abu-abu (grayscale) menjadi citra dengan dua nilai intensitas piksel: 0 (hitam) dan 1 (putih), berdasarkan ambang batas (threshold) tertentu. Proses ini digunakan untuk memudahkan analisis citra dengan memfokuskan pada objek atau fitur tertentu dalam citra.

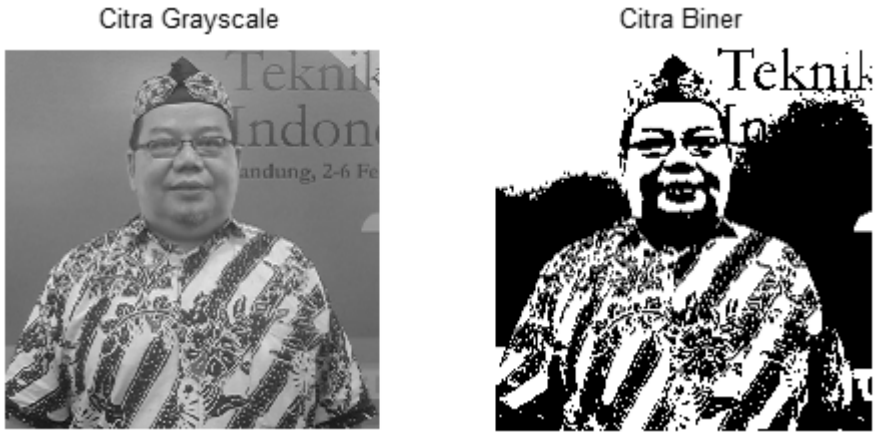
Langkah-langkah Konversi Citra Grayscale ke Biner

1. **Membaca citra grayscale**: Citra grayscale memiliki nilai intensitas piksel yang berkisar antara 0 hingga 255, di mana 0 mewakili warna hitam dan 255 mewakili warna putih.
2. **Menentukan ambang batas (threshold)**: Ambang batas ini digunakan untuk memisahkan piksel yang dianggap sebagai objek (nilai piksel lebih besar dari ambang batas) dan latar belakang (nilai piksel lebih kecil dari ambang batas).
3. **Mengubah nilai piksel menjadi biner**: Jika nilai piksel lebih besar dari ambang batas, maka nilai piksel tersebut diubah menjadi 1 (putih), jika lebih kecil atau sama dengan ambang batas, nilai piksel tersebut diubah menjadi 0 (hitam).
4. **Menyimpan hasil konversi**: Citra biner yang dihasilkan disimpan dalam format yang sesuai.

Berikut adalah contoh program MATLAB untuk mengonversi citra grayscale ke citra biner:

```
% Membaca citra grayscale
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra menjadi citra grayscale jika citra berwarna
if size(img, 3) == 3
   img = rgb2gray(img);  % Mengonversi citra berwarna menjadi grayscale
end
% Menentukan ambang batas untuk konversi
threshold = 128;  % Bisa disesuaikan, nilai antara 0 hingga 255
% Mengonversi citra grayscale ke citra biner
binary_img = img > threshold;  % Piksel yang lebih besar dari threshold menjadi 1, lainnya menjadi 0
% Menampilkan citra asli dan citra biner
subplot(1,2,1);
imshow(img);
title('Citra Grayscale');
subplot(1,2,2);
imshow(binary_img);
title('Citra Biner');
% Menyimpan citra biner
imwrite(binary_img, 'citra_biner.png');  % Menyimpan citra biner
```

Hasil:



Gambar 5.10: Contoh Konversi Citra darin warna ke Biner

Penjelasan Program:

1. **Membaca citra grayscale**: Fungsi imread() digunakan untuk membaca citra. Jika citra yang dibaca berwarna, fungsi rgb2gray() digunakan untuk mengonversinya menjadi grayscale.
2. **Menentukan ambang batas (threshold)**: Nilai ambang batas ditentukan sebagai 128 dalam contoh

ini, yang bisa disesuaikan.

3. **Konversi ke citra biner**: Operasi img > threshold menghasilkan citra biner di mana piksel yang lebih besar dari ambang batas diubah menjadi 1 (putih) dan lainnya menjadi 0 (hitam).
4. **Menampilkan hasil**: Fungsi imshow() digunakan untuk menampilkan citra grayscale dan citra biner.
5. **Menyimpan hasil**: Fungsi imwrite() digunakan untuk menyimpan citra biner ke dalam file.

Penyesuaian Ambang Batas

- Anda bisa menyesuaikan nilai ambang batas untuk melihat bagaimana citra biner berubah berdasarkan ambang batas yang berbeda. Ambang batas otomatis juga bisa diterapkan menggunakan metode seperti **Otsu's method** yang mengoptimalkan pemilihan ambang batas.

d. Konversi Citra Biner ke Grayscale

Konversi citra biner ke grayscale adalah proses di mana citra biner (yang hanya memiliki dua nilai intensitas piksel: hitam dan putih) diubah menjadi citra grayscale, yang memiliki nilai intensitas piksel dalam rentang yang lebih luas (misalnya, dari 0 hingga 255, di mana 0 adalah hitam, 255 adalah putih, dan nilai di antara keduanya mewakili abu-abu).

Pada dasarnya, citra biner hanya terdiri dari dua nilai piksel:

0 (hitam)

1 (putih)

Namun, dalam citra grayscale, intensitas piksel memiliki nilai yang lebih beragam antara 0 dan 255.

Proses Konversi:

Nilai 0 (hitam) dalam citra biner akan dikonversi menjadi nilai 0 pada citra grayscale.

Nilai 1 (putih) dalam citra biner akan dikonversi menjadi nilai 255 pada citra grayscale.

Jika ada nilai antara 0 dan 1 (misalnya, nilai 0.5), itu akan dipetakan ke nilai antara 0 dan 255.

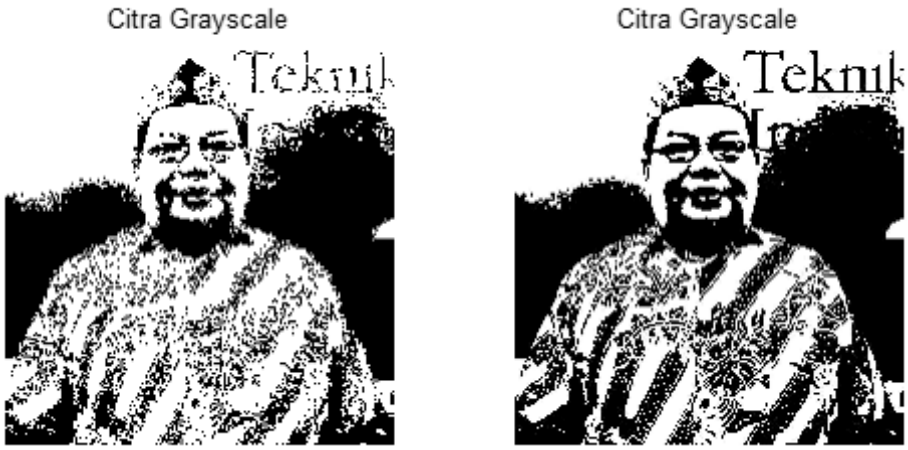
Langkah-langkah dalam Program Matlab:

1. Membaca citra biner (dengan nilai 0 dan 1).
2. Mengubah nilai-nilai citra biner ke rentang grayscale (0-255).
3. Menampilkan citra grayscale.

Program Matlab untuk Konversi Citra Biner ke Grayscale:

```
% Membaca citra biner
binaryImage = imread('E:\HINDARTO\2024\Buku Ajar 2024\program\gambar_biner.jpg');  % Ganti dengan path file citra biner
% Memastikan citra biner memiliki nilai 0 dan 1
binaryImage = double(binaryImage);  % Mengubah ke tipe data double jika perlu
% Konversi biner ke grayscale (0 -> 0, 1 -> 255)
grayscaleImage = binaryImage * 255;
% Menampilkan citra grayscale
subplot(1,2,1);
imshow(binaryImage);
title('Citra Biner');
subplot(1,2,2);
imshow(grayscaleImage, []);
title('Citra Grayscale');
```

Hasil:



Gambar 5.11: Contoh Konversi Citra dari Biner ke Grayscale

Penjelasan Program:
imread('citra_biner.png'): Membaca citra biner yang ada. Pastikan citra tersebut memiliki format biner, yaitu hanya terdiri dari piksel dengan nilai 0 atau 1.
double(binaryImage): Mengubah tipe data citra menjadi double agar perhitungan lebih mudah dilakukan.
binaryImage * 255: Mengonversi nilai 1 menjadi 255 (putih pada citra grayscale) dan nilai 0 tetap 0 (hitam pada citra grayscale).
imshow(grayscaleImage, []): Menampilkan citra grayscale yang telah diubah.
Catatan:
Pastikan citra biner yang digunakan hanya memiliki dua nilai, yaitu 0 dan 1. Jika citra memiliki nilai lain, Anda bisa menggunakan thresholding untuk mengubahnya menjadi citra biner terlebih dahulu. Citra grayscale memiliki rentang nilai 0 hingga 255, yang memungkinkan representasi tingkat keabuan di antara hitam dan putih. Program ini memberikan cara yang sederhana dan efektif untuk mengubah citra biner menjadi citra grayscale dalam Matlab.

e. Konversi Citra ke Format Lain (misalnya, PNG ke JPEG)
Konversi citra ke format lain merujuk pada proses mengubah file citra dari satu format file gambar ke format lainnya, seperti mengubah file dari .jpg ke .png, .bmp, .tiff, atau format gambar lainnya. Hal ini sering dilakukan untuk memudahkan pengolahan citra dalam aplikasi atau perangkat yang berbeda, karena beberapa aplikasi atau perangkat mungkin lebih mendukung format gambar tertentu.
Alasan Konversi Citra
Kompresi: Beberapa format gambar, seperti .jpg, dapat memberikan kompresi yang lebih baik untuk mengurangi ukuran file.
Transparansi: Format .png mendukung transparansi, yang tidak didukung oleh format .jpg.
Kualitas: Format seperti .tiff dapat digunakan untuk mempertahankan kualitas gambar tinggi tanpa kompresi lossy.
Kompatibilitas: Beberapa aplikasi mungkin hanya mendukung format gambar tertentu.
MATLAB menyediakan berbagai fungsi untuk membaca dan menulis file gambar dalam berbagai format. Berikut adalah langkah-langkah umum untuk konversi citra dengan MATLAB:
Membaca Citra: Menggunakan fungsi imread untuk membaca citra dari file.
Menyimpan Citra: Menggunakan fungsi imwrite untuk menyimpan citra dalam format yang diinginkan.
Contoh Program MATLAB untuk Konversi Citra

```
% Membaca citra dengan format .jpg
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menyimpan citra dalam format .png
imwrite(img, 'gambar_konversi.png');
% Menyimpan citra dalam format .bmp
imwrite(img, 'gambar_konversi.bmp');
% Menyimpan citra dalam format .tiff
imwrite(img, 'gambar_konversi.tiff');
```

Penjelasan Kode:
imread('gambar_asli.jpg'): Fungsi imread digunakan untuk membaca gambar dari file gambar_asli.jpg dan menyimpannya dalam variabel img.
imwrite(img, 'gambar_konversi.png'): Fungsi imwrite digunakan untuk menyimpan citra yang disimpan dalam variabel img ke dalam file dengan nama gambar_konversi.png. Anda dapat mengganti format file yang diinginkan, seperti .png, .bmp, atau .tiff.
Format yang Didukung oleh MATLAB
MATLAB mendukung banyak format gambar, antara lain:
.jpg, .jpeg: Format gambar kompresi lossy yang sering digunakan di web dan fotografi.
.png: Format gambar lossless yang mendukung transparansi.
.bmp: Format gambar tanpa kompresi.
.tiff: Format gambar yang mendukung berbagai mode warna dan tanpa kompresi.
.gif: Format untuk gambar bergerak atau gambar dengan palet warna terbatas.
Tips:
Pastikan file gambar yang ingin Anda baca berada dalam folder yang sama dengan skrip MATLAB, atau tentukan path lengkap file gambar.
Beberapa format, seperti .jpg atau .png, menggunakan kompresi yang dapat mempengaruhi kualitas citra. Untuk kualitas terbaik, gunakan format lossless seperti .png atau .tiff.
Dengan demikian, konversi citra dalam MATLAB sangat mudah dilakukan dengan menggunakan imread dan imwrite. Anda hanya perlu menyesuaikan format sesuai kebutuhan aplikasi atau perangkat yang akan digunakan.

Soal-soal Latihan:

1. Jelaskan cara memuat citra berwarna dalam format RGB menggunakan MATLAB. Kemudian, tampilkan citra tersebut menggunakan fungsi imshow. Berikan contoh kode MATLAB untuk memuat dan menampilkan citra berwarna.
2. Konversi citra RGB menjadi citra grayscale dalam MATLAB menggunakan rumus yang umum. Setelah konversi, tampilkan citra grayscale yang dihasilkan menggunakan fungsi imshow. Berikan contoh kode MATLAB untuk konversi dan penampilannya.
3. Dalam pengolahan citra, Anda diminta untuk mengonversi citra RGB menjadi citra biner dengan thresholding. Jelaskan cara mengonversi citra RGB ke citra biner menggunakan threshold tertentu di MATLAB dan tampilkan hasilnya. Sertakan contoh kode MATLAB yang melakukan hal tersebut.
4. Sebuah citra grayscale perlu dikonversi menjadi citra biner menggunakan teknik thresholding. Tulis kode MATLAB untuk melakukan konversi tersebut dan tampilkan hasil citra biner yang dihasilkan. Jelaskan bagaimana cara memilih nilai threshold yang tepat.
5. Tulis kode MATLAB untuk mengonversi citra RGB ke format citra HSI (Hue, Saturation, Intensity). Setelah konversi, tampilkan komponen Hue, Saturation, dan Intensity menggunakan fungsi imshow. Jelaskan proses konversi dan pengaruhnya terhadap citra.

Bab 6
Operasi Ketetanggaan Piksel dan Histogram Citra

6.1 Operasi Ketetanggaan Piksel pada Citra

Operasi ketetanggaan piksel pada citra (atau *pixel neighborhood operations*) merujuk pada teknik yang digunakan untuk memproses nilai-nilai piksel dalam citra digital berdasarkan hubungan atau ketetanggaan antar piksel-piksel di sekitarnya (Pangaribuan, 2019). Dalam banyak kasus, operasi ini digunakan untuk memperbaiki citra atau untuk mendeteksi fitur-fitur tertentu dalam citra, seperti batas, tepi, atau tekstur.
Berikut adalah contoh implementasi operasi ketetanggaan piksel menggunakan filter konvolusi di MATLAB. Kita akan menggunakan berbagai kernel untuk melakukan operasi penyaringan (filtering) pada citra, seperti operasi *blur*, *sharpen*, dan *edge detection*. Setiap kernel akan diterapkan pada citra dengan melakukan konvolusi.
1. Operasi Blur dengan Filter Rata-Rata (Mean Filter)
Filter rata-rata digunakan untuk melakukan blur pada citra dengan cara menghitung rata-rata dari piksel-piksel tetangga di sekitar setiap piksel.
Program MATLAB:

```
% Membaca citra input
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra ke grayscale (jika citra berwarna)
gray_img = rgb2gray(img);
% Membuat filter rata-rata 3x3
mean_filter = ones(3, 3) / 9;  % Matriks 3x3 yang berisi nilai 1/9
% Melakukan konvolusi antara citra dan filter rata-rata
blurred_img = conv2(double(gray_img), mean_filter, 'same');
% Menampilkan citra hasil blur
figure;
subplot(1,2,1);
imshow(img);
title('Citra RGB');
subplot(1,2,2);
imshow(uint8(blurred_img));
title('Citra Hasil Blur Menggunakan Filter Rata-Rata');
```

Hasil:

Gambar 6.1: Citra Warna dan Citra hasil Blur menggunakan filter rata-rata

Penjelasan:

- Filter rata-rata 3x3: Matriks filter berukuran 3x3 yang semua elemennya diisi dengan nilai 1/9 (jumlah total 1).

- conv2: Fungsi ini melakukan konvolusi antara citra dan kernel. Opsi 'same' memastikan ukuran citra output tetap sama dengan citra input.
- Citra hasil blur akan memiliki efek kabur karena nilai setiap piksel dihitung dari rata-rata piksel-piksel di sekitarnya.

2. Operasi Sharpening dengan Filter Laplacian
Filter Laplacian digunakan untuk mempertegas (sharpen) citra dengan cara meng-highlight perubahan intensitas pada citra, sehingga tepi-tepi objek lebih tajam.

```
% Membaca citra input
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra ke grayscale (jika citra berwarna)
gray_img = rgb2gray(img);
% Membuat filter laplacian 3x3 untuk sharpening
laplacian_filter = [0 -1 0; -1 4 -1; 0 -1 0];  % Filter Laplacian
% Melakukan konvolusi antara citra dan filter Laplacian
sharpened_img = conv2(double(gray_img), laplacian_filter, 'same');
% Menambahkan hasil konvolusi ke citra asli untuk meningkatkan ketajaman
sharpened_img = uint8(double(gray_img) - sharpened_img);
% Menampilkan citra hasil sharpening
figure;
subplot(1,2,1);
imshow(img);
title('Citra RGB');
subplot(1,2,2);
imshow(sharpened_img);
title('Citra Hasil Sharpening Menggunakan Filter Laplacian');
```

Hasil:

Gambar 6.2: Citra Warna dan Citra hasil Blur menggunakan filter Lapacian

Penjelasan:
- Filter Laplacian: Filter ini berfungsi untuk mendeteksi tepi dan meningkatkan kontras antara area terang dan gelap di citra. Biasanya digunakan untuk sharpening.
- conv2: Digunakan untuk melakukan konvolusi antara citra dengan filter Laplacian.
- Hasil konvolusi dari Laplacian kemudian dikurangkan dari citra asli untuk meningkatkan ketajaman.

3. Operasi Deteksi Tepi dengan Filter Sobel
Filter Sobel digunakan untuk mendeteksi tepi citra dengan menghitung gradien perubahan intensitas dalam arah horizontal dan vertikal.

```
% Membaca citra input
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra ke grayscale (jika citra berwarna)
```

```
gray_img = rgb2gray(img);
% Membuat kernel Sobel untuk deteksi tepi horizontal dan vertikal
sobel_horizontal = [-1 0 1; -2 0 2; -1 0 1];  % Deteksi tepi horizontal
sobel_vertical = [-1 -2 -1; 0 0 0; 1 2 1];    % Deteksi tepi vertikal
% Melakukan konvolusi dengan kernel Sobel horizontal dan vertikal
edge_horizontal = conv2(double(gray_img), sobel_horizontal, 'same');
edge_vertical = conv2(double(gray_img), sobel_vertical, 'same');
% Menghitung magnitudo dari kedua hasil konvolusi
edge_magnitude = sqrt(edge_horizontal.^2 + edge_vertical.^2);
% Menampilkan citra hasil deteksi tepi
figure;
subplot(1,2,1);
imshow(img);
title('Citra RGB');
subplot(1,2,2);
imshow(uint8(edge_magnitude));
title('Citra Hasil Deteksi Tepi Menggunakan Filter Sobel');
```

Hasil:

Gambar 6.3: Citra Warna dan Citra hasil Blur menggunakan filter Sobel

Penjelasan:

- Kernel Sobel: Filter Sobel digunakan untuk mendeteksi perubahan intensitas citra dalam arah horizontal dan vertikal. Filter horizontal dan vertikal akan menangkap perubahan di kedua arah tersebut.
- Magnitudo Tepi: Hasil dari deteksi tepi dihitung dengan menggabungkan informasi dari kedua arah menggunakan rumus Pythagoras (akar dari jumlah kuadrat).

4. Operasi Erosi dan Dilasi dengan Filter Morfologi

Erosi dan dilasi adalah operasi morfologi yang digunakan pada citra biner untuk mengurangi atau memperbesar objek dalam citra.

```
% Membaca citra biner
binary_img = imread('E:\HINDARTO\2024\Buku Ajar 2024\program\citra_biner.png');  % Pastikan citra sudah biner
% Melakukan erosi
se = strel('square', 3);  % Struktur elemen berbentuk persegi 3x3
eroded_img = imerode(binary_img, se);
% Melakukan dilasi
dilated_img = imdilate(binary_img, se);
% Menampilkan hasil erosi dan dilasi
subplot(1, 2, 1);
imshow(eroded_img);
title('Citra Hasil Erosi');
subplot(1, 2, 2);
```

imshow(dilated_img);
title('Citra Hasil Dilasi');
Hasil:

Gambar 6.4: Citra Warna dan Citra hasil Blur menggunakan filter Morfologi

Penjelasan:

- Erosi dan Dilasi: Dalam operasi morfologi, erosi mengurangi objek dalam citra, sementara dilasi memperbesarnya. Filter ini bekerja pada citra biner (hitam-putih).
- strel: Fungsi ini digunakan untuk membuat struktur elemen untuk operasi morfologi (misalnya, persegi 3x3).
- imerode dan imdilate: Fungsi-fungsi ini digunakan untuk menerapkan operasi erosi dan dilasi pada citra biner.

6.2 Histogram Citra

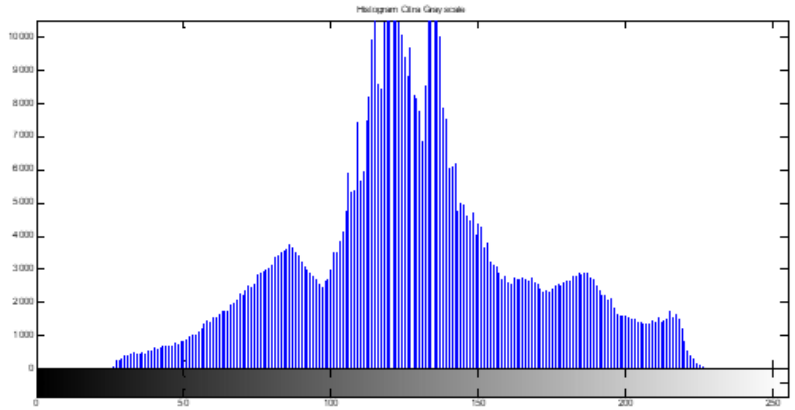
Histogram citra adalah representasi grafis dari distribusi intensitas piksel dalam citra (Kurniastuti & Andini, 2018). Histogram menggambarkan frekuensi kemunculan setiap nilai intensitas dalam citra, yang dapat memberikan wawasan tentang karakteristik citra, seperti kontras, kecerahan, dan distribusi warna.

1. Histogram Grayscale

Untuk citra grayscale, histogram akan menunjukkan berapa kali setiap tingkat intensitas (dari 0 hingga 255) muncul dalam citra. Citra dengan kontras tinggi akan memiliki histogram yang tersebar di seluruh rentang intensitas, sementara citra dengan kontras rendah akan memiliki histogram yang lebih terkonsentrasi pada nilai intensitas tertentu.

```
% Membaca citra grayscale
citra_gray = imread('E:\HINDARTO\2024\Buku Ajar 2024\program\gambar_grayscale.jpg');
% Menampilkan histogram citra grayscale
figure;
imhist(citra_gray);
title('Histogram Citra Grayscale');
```

Hasil:

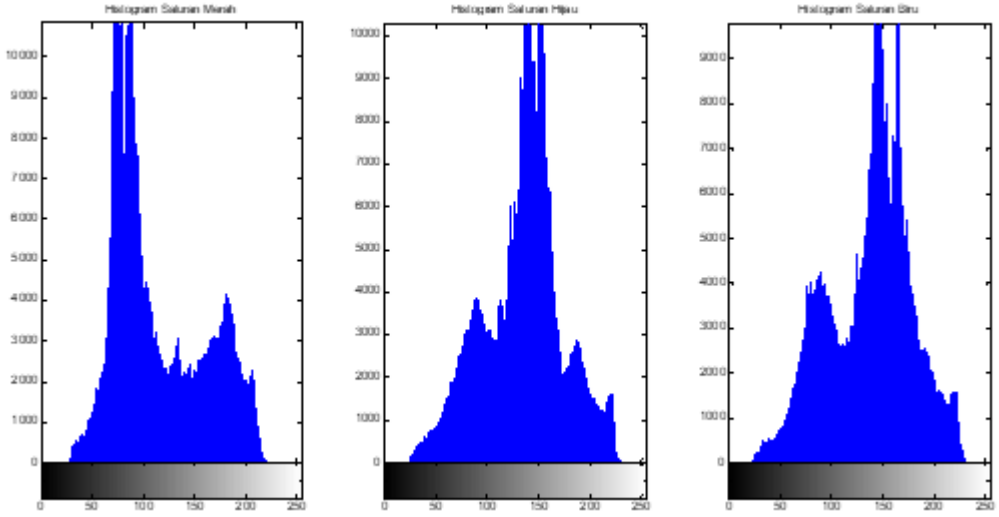


Gambar 6.5: histogram citra grayscale

2. Histogram Citra Warna (RGB)
Untuk citra berwarna, histogram dapat dihitung untuk masing-masing saluran warna (R, G, B). Setiap saluran warna memiliki histogram terpisah yang menunjukkan distribusi intensitas untuk warna merah, hijau, dan biru.

```
% Membaca citra RGB
citra_rgb = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menampilkan histogram untuk masing-masing saluran warna
figure;
subplot(1,3,1), imhist(citra_rgb(:,:,1)), title('Histogram Saluran Merah');
subplot(1,3,2), imhist(citra_rgb(:,:,2)), title('Histogram Saluran Hijau');
subplot(1,3,3), imhist(citra_rgb(:,:,3)), title('Histogram Saluran Biru');
```

Hasil:



Gambar 6.6: histogram citra Warna

3. Penggunaan Histogram dalam Pemrosesan Citra
Histogram memiliki banyak kegunaan dalam pengolahan citra, antara lain:
Peningkatan Citra (Histogram Equalization): Histogram equalization adalah teknik untuk meningkatkan kontras citra dengan meratakan distribusi intensitas piksel. Tujuannya adalah untuk membuat citra lebih mudah dilihat, terutama ketika citra memiliki rentang intensitas yang sempit.

```
% Melakukan histogram equalization pada citra grayscale
citra_eq = histeq(citra_gray);
% Menampilkan citra asli dan citra hasil equalization
figure;
subplot(1, 2, 1), imshow(citra_gray), title('Citra Asli');
subplot(1, 2, 2), imshow(citra_eq), title('Citra setelah Equalization');
```

Hasil:

Gambar 6.7: citra asli dan citra hasil equalization

Thresholding berdasarkan Histogram: Teknik thresholding dapat dilakukan dengan menggunakan histogram untuk menentukan nilai ambang batas yang memisahkan objek dari latar belakang. Citra biner dapat dibuat dengan memilih piksel yang memiliki nilai intensitas lebih besar atau lebih kecil dari nilai ambang.

```
% Membaca citra
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra ke grayscale (jika citra berwarna)
citra_gray = rgb2gray(citra);
% Menghitung threshold otomatis menggunakan metode Otsu
level = graythresh(citra_gray);
% Mengubah citra grayscale menjadi citra biner berdasarkan threshold yang dihitung
citra_biner = citra_gray > level * 255;  % Mengalikan dengan 255 untuk konversi ke skala 8-bit
% Menampilkan citra biner
imshow(citra_biner);
title('Citra Biner dengan Threshold Otsu');
```

Hasil:

Gambar 6.8: citra Biner dengan threshold otsu

Soal-soal Latihan:

1. Jelaskan apa yang dimaksud dengan operasi ketetanggaan piksel (neighborhood operations) dalam pengolahan citra. Sebutkan dua contoh operasi ketetanggaan yang sering digunakan dan jelaskan bagaimana operasi tersebut diterapkan pada citra.
2. Apa itu filter konvolusi dalam pengolahan citra? Jelaskan bagaimana operasi konvolusi dapat digunakan untuk memodifikasi citra dengan menggunakan mask atau kernel. Berikan contoh penerapan filter konvolusi pada citra untuk operasi seperti pemblur atau deteksi tepi.
3. Apa yang dimaksud dengan histogram citra? Jelaskan bagaimana histogram citra dapat menggambarkan distribusi intensitas piksel dalam citra. Sebutkan dua cara yang dapat dilakukan untuk menganalisis histogram citra.
4. Dalam MATLAB, bagaimana cara membuat dan menampilkan histogram dari sebuah citra grayscale? Berikan contoh kode MATLAB untuk menghasilkan histogram citra, serta jelaskan apa yang ditunjukkan oleh histogram tersebut.
5. Salah satu teknik yang sering digunakan dalam pengolahan citra adalah *equalization* histogram untuk meningkatkan kontras citra. Jelaskan proses histogram equalization dan berikan contoh implementasi dalam MATLAB untuk meningkatkan kontras citra grayscale.

Bab 7
Operasi Filter Median, Mean dan Rata-Rata

Filter adalah salah satu teknik dasar dalam pengolahan citra untuk memodifikasi citra dengan cara menerapkan suatu operasi matematis yang melibatkan tetangga piksel. Filter digunakan untuk berbagai tujuan, seperti mengurangi noise, meningkatkan kualitas citra, atau menonjolkan fitur-fitur tertentu seperti tepi atau tekstur. Beberapa jenis filter yang umum digunakan adalah **filter median**, **filter rata-rata (mean)**, dan **filter rata-rata (average)**.

7.1 Filter Rata-rata (Mean Filter)

Filter Rata-rata (Mean Filter) adalah teknik pemrosesan citra yang digunakan untuk menghaluskan citra atau mengurangi noise (Yuwono, 2015). Filter ini bekerja dengan cara menggantikan setiap piksel dalam citra dengan rata-rata nilai piksel-piksel yang ada di sekitarnya, berdasarkan ukuran kernel yang digunakan. Proses ini membantu mengurangi fluktuasi intensitas yang tajam di citra, yang sering kali disebabkan oleh noise.
Prinsip Kerja Mean Filter:
Kernel: Filter rata-rata menggunakan matriks kecil yang disebut kernel atau masukan. Biasanya, kernel ini berbentuk matriks 3x3, 5x5, atau ukuran lainnya yang digunakan untuk menghitung rata-rata intensitas piksel sekitarnya.
Proses: Untuk setiap piksel di citra, filter akan mengganti nilai piksel tersebut dengan rata-rata dari nilai piksel-piksel yang ada di dalam kernel. Jika kernel berukuran 3x3, misalnya, maka nilai baru piksel pusat adalah rata-rata dari 9 piksel yang ada di sekitarnya.
Contoh Proses:
Misalkan ada citra dengan ukuran 5x5 seperti berikut:
100 100 100 100 100
100 150 150 150 100
100 150 200 150 100
100 150 150 150 100
100 100 100 100 100
Jika kita menggunakan kernel 3x3 untuk filter rata-rata, maka nilai piksel di tengah (misalnya 200) akan diganti dengan rata-rata dari 9 piksel yang ada di sekitarnya. Dalam hal ini, nilai rata-ratanya adalah (100+100+100+100+150+150+150+150+200) / 9 = 141.11. Proses ini akan diterapkan pada seluruh citra.
Keunggulan dan Kekurangan:
Keunggulan: Filter rata-rata sangat efektif dalam mengurangi noise, terutama noise jenis Gaussian atau salt-and-pepper.
Kekurangan: Filter ini dapat menyebabkan citra menjadi kabur atau blur, terutama jika ukuran kernel terlalu besar, karena tepi objek akan menjadi lebih halus.
Berikut adalah contoh implementasi filter rata-rata 3x3 di MATLAB:

```
% Membaca citra
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
I_gray = rgb2gray(I);     % Mengonversi citra menjadi grayscale jika citra berwarna
% Menentukan ukuran kernel (3x3 untuk contoh ini)
kernel = ones(3, 3) / 9;  % Matriks kernel 3x3 dengan nilai rata-rata 1/9
% Menerapkan filter rata-rata pada citra
I_filtered = imfilter(I_gray, kernel);
% Menampilkan citra asli dan citra yang sudah difilter
subplot(1, 2, 1);
imshow(I_gray);
title('Citra Asli');
subplot(1, 2, 2);
imshow(I_filtered);
title('Citra Setelah Filter Rata-rata');
```

Hasil:

Gambar 7.1: Citra Asli dan Citra setelah Filter Rata-rata

Penjelasan Program:
Membaca Citra: imread('image.jpg') digunakan untuk membaca citra dari file.
Konversi ke Grayscale: rgb2gray(I) mengonversi citra warna menjadi citra grayscale, yang lebih mudah diproses dengan filter rata-rata.
Kernel 3x3: kernel = ones(3, 3) / 9 membuat kernel 3x3 dengan elemen-elemen yang bernilai 1/9 (karena rata-rata 3x3 adalah 1/9).
Filter dengan imfilter: Fungsi imfilter digunakan untuk menerapkan filter rata-rata ke citra.
Menampilkan Hasil: imshow digunakan untuk menampilkan citra asli dan citra yang sudah difilter.
Dengan kode ini, Anda dapat melihat bagaimana citra yang telah difilter rata-rata akan terlihat lebih halus dan noise akan berkurang.

7.2 Filter Median

Filter median adalah filter non-linier yang menggantikan setiap piksel dengan nilai median dari intensitas piksel di sekitar piksel tersebut (Tarigan, 2018). Dalam hal ini, median adalah nilai yang terletak di tengah urutan piksel yang sudah diurutkan berdasarkan intensitasnya. Filter median sangat efektif dalam menghilangkan **salt and pepper noise** tanpa mengaburkan detail citra sebanyak filter rata-rata.

Proses Filter Median:

Misalnya, pada sebuah jendela 3x3, kita mengurutkan intensitas piksel dalam jendela tersebut dan memilih nilai yang berada di posisi tengah (median).

Langkah-langkah:

1. Ambil jendela 3x3 (atau ukuran lainnya) dari citra.
2. Urutkan nilai intensitas dalam jendela tersebut.
3. Gantikan nilai piksel pusat dengan nilai median dari urutan tersebut.

```
% Membaca citra
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra menjadi grayscale jika citra berwarna
if size(citra, 3) == 3
   citra_gray = rgb2gray(citra);
else
   citra_gray = citra;
end
% Menerapkan filter median 3x3
citra_median = medfilt2(citra_gray, [3 3]);
% Menampilkan citra asli dan citra yang sudah difilter
subplot(1, 2, 1);
imshow(citra_gray);
title('Citra Asli');
subplot(1, 2, 2);
imshow(citra_median);
```

title('Citra Setelah Filter Median');
Hasil:

Gambar 7.2: Citra Asli dan Citra setelah Filter Median

Keunggulan:

- Filter median sangat baik dalam menghilangkan **salt and pepper noise** tanpa mengaburkan tepi atau detail penting pada citra.

Kelemahan:

- Bisa lebih lambat dalam pemrosesan dibandingkan dengan filter rata-rata, terutama pada citra yang besar.

7.3 Filter Rata-rata (Average Filter)

Filter Rata-rata (Average Filter) adalah teknik pengolahan citra yang digunakan untuk mengurangi noise (gangguan) pada gambar dengan menggantikan setiap piksel citra dengan rata-rata nilai piksel di sekitarnya (Wulandari, 2019). Teknik ini tergolong dalam filter linier dan sering digunakan untuk menghaluskan gambar.
Prinsip Kerja Filter Rata-rata
Filter rata-rata bekerja dengan cara menghitung nilai rata-rata dari sekelompok piksel yang ada di dalam jendela filter yang bergerak (kernel) di sekitar setiap piksel citra. Sebagai contoh, jika kita menggunakan kernel 3x3, maka untuk setiap piksel dalam citra, kernel akan menghitung rata-rata nilai intensitas dari piksel-piksel yang ada di dalam jendela 3x3 tersebut, lalu mengganti nilai piksel tengah dengan nilai rata-rata tersebut.
Proses Filter Rata-rata

1. Pilih ukuran kernel: Ukuran kernel adalah area piksel yang akan digunakan untuk menghitung rata-rata. Misalnya, kernel 3x3 atau 5x5.
2. Geser kernel ke seluruh citra: Kernel digeser pixel per pixel melalui seluruh citra. Untuk setiap posisi kernel, hitung rata-rata intensitas piksel di dalam jendela.
3. Terapkan rata-rata: Setiap piksel yang diproses diganti dengan rata-rata nilai dari kernel yang bersangkutan.

Keuntungan dan Kekurangan Filter Rata-rata

- Keuntungan:
  - Efektif untuk menghilangkan noise yang tersebar secara acak (random noise).
  - Mudah diterapkan dan diimplementasikan.
- Kekurangan:
  - Dapat mengaburkan tepi-tepi gambar (blurring) karena penggantian nilai piksel dengan rata-rata.
  - Kurang efektif untuk mengatasi noise yang terstruktur atau noise yang memiliki pola tertentu.

Berikut adalah contoh implementasi Filter Rata-rata menggunakan MATLAB untuk citra berwarna dan citra grayscale.
1. Filter Rata-rata untuk Citra Grayscale

```
% Membaca citra grayscale
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menentukan ukuran kernel (misalnya 3x3)
```

```
kernel = ones(3, 3) / 9;
% Menerapkan filter rata-rata menggunakan fungsi imfilter
filtered_img = imfilter(img, kernel, 'same');
% Menampilkan hasilnya
subplot(1,2,1);
imshow(img);
title('Citra Asli');
subplot(1,2,2);
imshow(filtered_img);
title('Citra Setelah Diterapkan Filter Rata-rata');
```

Hasil:

Gambar 7.3: Citra Asli dan Citra setelah Filter Rata-rata

2. Filter Rata-rata untuk Citra Warna

Jika citra yang digunakan berwarna, kita perlu memproses setiap saluran warna (R, G, B) secara terpisah.

```
% Membaca citra berwarna
img_color = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Membagi citra menjadi 3 saluran warna (Red, Green, Blue)
red_channel = img_color(:, :, 1);
green_channel = img_color(:, :, 2);
blue_channel = img_color(:, :, 3);
% Kernel rata-rata 3x3
kernel = ones(3, 3) / 9;
% Terapkan filter rata-rata pada masing-masing saluran
red_filtered = imfilter(red_channel, kernel, 'same');
green_filtered = imfilter(green_channel, kernel, 'same');
blue_filtered = imfilter(blue_channel, kernel, 'same');
% Gabungkan saluran-saluran yang telah difilter
filtered_img_color = cat(3, red_filtered, green_filtered, blue_filtered);
% Menampilkan hasilnya
subplot(1,2,1);
imshow(img);
title('Citra Asli');
subplot(1,2,2);
imshow(filtered_img_color);
title('Citra Berwarna Setelah Diterapkan Filter Rata-rata');
```

Hasil:

Gambar 7.4: Citra Asli dan Citra setelah Filter Rata-rata

Penjelasan Kode

- imread('image.jpg'): Membaca gambar dari file.
- ones(3,3) / 9: Membuat kernel 3x3 yang berisi nilai 1/9, yang digunakan untuk menghitung rata-rata.
- imfilter(img, kernel, 'same'): Menggunakan fungsi imfilter untuk menerapkan filter rata-rata pada citra. Parameter 'same' memastikan ukuran citra yang dihasilkan tetap sama dengan citra input.
- cat(3, red_filtered, green_filtered, blue_filtered): Menggabungkan saluran-saluran warna setelah difilter untuk membentuk citra warna kembali.

Dengan menggunakan filter rata-rata ini, citra yang memiliki noise akan terlihat lebih halus, meskipun bisa juga terjadi efek blur pada bagian tepi atau detail halus citra.

Soal-soal Latihan:

1. Jelaskan apa yang dimaksud dengan filter median dalam pengolahan citra. Bagaimana cara kerja filter median untuk mengurangi noise pada citra, dan dalam situasi apa filter median lebih efektif dibandingkan dengan filter lain?
2. Apa yang dimaksud dengan filter rata-rata (mean filter) dalam pengolahan citra? Jelaskan bagaimana filter rata-rata bekerja untuk melakukan pemblur (blurring) pada citra dan apa dampak dari penggunaan filter ini pada kualitas citra.
3. Bagaimana cara menerapkan filter median pada citra grayscale menggunakan MATLAB? Berikan contoh kode untuk menerapkan filter median pada citra yang terpengaruh noise (misalnya salt-and-pepper noise).
4. Jelaskan perbedaan antara filter median dan filter rata-rata dalam hal pengaruh terhadap citra. Apa keuntungan dan kerugian masing-masing filter? Berikan contoh aplikasi di mana masing-masing filter lebih disukai.
5. Dalam MATLAB, bagaimana cara mengaplikasikan filter rata-rata (mean filter) pada citra menggunakan fungsi fspecial dan imfilter? Jelaskan langkah-langkah yang diperlukan dan tampilkan citra hasil filter rata-rata. Berikan contoh kode MATLAB untuk ini.

Bab 8
Operasi Geometri Citra

Operasi geometri citra adalah serangkaian teknik yang digunakan untuk mengubah posisi, ukuran, atau orientasi citra. Tujuan utama dari operasi geometri adalah untuk mengubah atau memanipulasi citra agar lebih sesuai dengan kebutuhan aplikasi tertentu, seperti pemetaan citra, pengenalan objek, dan analisis citra. Operasi ini sangat penting dalam banyak bidang, seperti pemrosesan citra medis, pengenalan pola, dan aplikasi pengolahan citra satelit.

Jenis-jenis Operasi Geometri Citra

Beberapa operasi geometri yang sering digunakan dalam pengolahan citra antara lain:

1. **Translasi (Pergeseran)**
2. **Rotasi**
3. **Skalasi (Penyesuaian Ukuran)**
4. **Shearing (Penyusutan)**
5. **Flipping (Pembalikan)**
6. **Perspektif (Keprojekan)**
7. **Cropping (Pemotongan)**
8. **Resampling**

8.1 Translasi (Pergeseran)

Translasi adalah suatu transformasi dalam pengolahan citra yang menggeser posisi piksel dalam gambar pada arah tertentu, tanpa mengubah bentuk atau ukuran citra tersebut (Sembiring, 2012). Dalam translasi, setiap piksel pada citra dipindahkan sejauh nilai tertentu dalam arah horizontal (sumbu x) dan vertikal (sumbu y). Matematis, translasi pada koordinat gambar dapat digambarkan dengan persamaan berikut:

$$(x', y') = (x + t_y, y + t_y)$$

Di mana:

- (x, y) adalah koordinat piksel asli.
- (x', y') adalah koordinat piksel setelah pergeseran.
- tx adalah jarak translasi pada sumbu x (horizontal).
- ty adalah jarak translasi pada sumbu y (vertikal).

Untuk melakukan translasi pada citra menggunakan MATLAB, kita dapat menggunakan fungsi imtranslate dari Image Processing Toolbox, atau melakukan pergeseran manual dengan menggeser indeks piksel pada citra.

Berikut adalah contoh implementasi translasi citra dalam MATLAB:

```
% Membaca citra
image = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menampilkan citra asli
subplot(1, 2, 1);
imshow(image);
title('Citra Asli');
% Menentukan nilai translasi
tx = 1000;  % Pergeseran pada sumbu x
ty = 500;  % Pergeseran pada sumbu y
% Membuat matriks transformasi translasi
tform = affine2d([1 0 0; 0 1 0; tx ty 1]);

% Mentranslasi citra menggunakan transformasi affine
translated_image = imwarp(image, tform);
```

```
% Menampilkan citra yang telah ditranslasi
subplot(1, 2, 2);
imshow(translated_image);
title('Citra Setelah Translasi');
```
Hasil:

Gambar 8.1: Pergeseran Citra Asli dan Citra setelah Translasi

Penjelasan Program:

1. **Membaca Citra**: Fungsi imread digunakan untuk membaca citra dari file.
2. **Menampilkan Citra Asli**: Citra asli ditampilkan menggunakan imshow.
3. **Menentukan Jarak Translasi**: Nilai translasi untuk sumbu x (tx) dan sumbu y (ty) ditentukan.
4. **Transformasi Afine**: Matriks transformasi affine2d digunakan untuk membuat transformasi translasi, dengan nilai tx dan ty.
5. **Penerapan Translasi**: Fungsi imwarp digunakan untuk mentransformasi citra sesuai dengan matriks transformasi yang telah ditentukan.
6. **Menampilkan Citra yang Telah Ditranslasi**: Citra yang telah diterjemahkan ditampilkan untuk perbandingan.

Dengan langkah-langkah di atas, citra akan digeser sesuai dengan nilai translasi yang telah ditentukan.

Catatan:

- **imwarp** memungkinkan aplikasi transformasi geometrik yang lebih fleksibel, sementara **imtranslate** bisa lebih sederhana jika hanya melakukan translasi dasar tanpa perubahan lainnya.
- Pastikan citra yang ditranslasi tetap berada dalam ukuran yang diinginkan setelah translasi (misalnya, pastikan tidak ada piksel yang terpotong).

Dengan teknik ini, kita bisa memindahkan citra ke posisi baru tanpa merubah isi atau struktur citra lainnya.

8.2 Rotasi

Rotasi pada citra adalah transformasi geometris yang memutar citra pada sudut tertentu (biasanya dalam derajat) terhadap titik pusat citra atau titik lain yang ditentukan (Sunardi et al., 2021). Rotasi ini dapat dilakukan dalam dua dimensi (2D) atau tiga dimensi (3D), tetapi di sini kita akan fokus pada rotasi citra 2D.

Proses rotasi pada citra 2D melibatkan pemutaran setiap piksel pada citra tersebut sesuai dengan sudut tertentu. Sebagai contoh, rotasi citra sebanyak 90 derajat akan memutar citra pada titik pusatnya, menghasilkan citra yang berubah orientasinya.

Matematika rotasi pada citra 2D dapat dijelaskan dengan menggunakan matriks rotasi. Untuk rotasi sebesar θ derajat, matriks rotasi R(θ) adalah sebagai berikut:

$$R\ (\theta) = \begin{bmatrix} Cos\ (\theta) & -Sin\ (\theta) \\ Sin\ (\theta) & Cos\ (\theta) \end{bmatrix}$$

Jika kita memiliki titik (x, y) pada citra dan ingin merotasi titik tersebut, kita akan mengalikan titik tersebut dengan matriks rotasi.

Berikut adalah langkah-langkah dan contoh program untuk melakukan rotasi citra di MATLAB:

1. **Membaca citra** yang akan dirotasi.
2. **Menentukan sudut rotasi** (dalam derajat).

3. **Menghitung matriks rotasi** dan melakukan transformasi pada citra.
4. **Menampilkan citra asli dan citra hasil rotasi**.

Berikut adalah contoh kode MATLAB untuk merotasi citra:

```
% Membaca citra
image = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menentukan sudut rotasi (dalam derajat)
theta = 45;  % Rotasi 45 derajat
% Melakukan rotasi menggunakan fungsi imrotate
rotated_image = imrotate(image, theta, 'bilinear', 'crop');
% Menampilkan citra asli dan citra yang sudah dirotasi
subplot(1, 2, 1);
imshow(image);
title('Citra Asli');
subplot(1, 2, 2);
imshow(rotated_image);
title(['Citra Setelah Rotasi ', num2str(theta), ' Derajat']);
```

Hasil:

Gambar 8.2: Citra Asli dan Citra setelah Rotasi

Penjelasan Kode:

1. **imread('citra.jpg')**: Membaca citra dari file.
2. **theta = 45**: Menentukan sudut rotasi dalam derajat.
3. **imrotate(image, theta, 'bilinear', 'crop')**: Fungsi imrotate digunakan untuk merotasi citra. Opsi 'bilinear' menentukan metode interpolasi yang digunakan untuk menghitung nilai piksel setelah rotasi. Opsi 'crop' memastikan ukuran citra output tetap sama dengan ukuran citra asli (tetapi beberapa informasi bisa hilang di tepi citra jika diperlukan).
4. **subplot(1, 2, 1)** dan **subplot(1, 2, 2)**: Menampilkan citra asli dan citra yang sudah dirotasi dalam satu jendela gambar.

Metode Interpolasi:

- **Nearest Neighbor**: Piksel yang paling dekat dengan posisi baru akan dipilih. Ini cepat tetapi bisa menghasilkan citra yang kasar.
- **Bilinear**: Menggunakan nilai piksel terdekat di sekitar posisi baru dengan interpolasi linear. Ini memberikan hasil yang lebih halus dibandingkan metode nearest neighbor.
- **Bicubic**: Lebih rumit daripada bilinear, menggunakan nilai piksel lebih banyak untuk interpolasi. Hasilnya lebih halus tetapi lebih lambat.

Catatan:

- Fungsi imrotate di MATLAB secara otomatis menangani perhitungan matriks rotasi dan penyesuaian ukuran citra untuk memastikan citra yang dihasilkan tetap lengkap.
- Anda bisa menyesuaikan nilai sudut theta untuk rotasi sesuai kebutuhan.

8.3 Skalasi (Penyesuaian Ukuran)

Skalasi (penyesuaian ukuran) pada citra adalah proses mengubah ukuran gambar, baik memperbesar

(upscaling) maupun memperkecil (downscaling) gambar tersebut (JASMINE, 2014). Dalam pemrosesan citra digital, skalasi digunakan untuk berbagai tujuan, seperti pengurangan ukuran file, perubahan resolusi gambar, atau persiapan gambar untuk analisis lebih lanjut.

Jenis-jenis Skalasi pada Citra

1. **Upscaling**: Memperbesar gambar, yang biasanya melibatkan penambahan piksel dengan interpolasi untuk mengisi ruang yang lebih besar.
2. **Downscaling**: Memperkecil gambar, yang melibatkan pengurangan jumlah piksel, dan biasanya membutuhkan pemilihan atau agregasi data untuk mencocokkan ukuran yang lebih kecil.

Metode Interpolasi pada Skalasi Citra

Saat melakukan skalasi, terutama saat memperbesar gambar, interpolasi digunakan untuk menghasilkan nilai warna piksel baru yang sesuai dengan ukuran yang diinginkan. Beberapa metode interpolasi yang umum digunakan adalah:

1. **Nearest Neighbor**: Metode yang paling sederhana, di mana setiap piksel baru diambil dari piksel terdekat di gambar asli.
2. **Bilinear Interpolation**: Menggunakan dua nilai terdekat dalam kedua arah (horizontal dan vertikal) untuk menghitung nilai piksel baru.
3. **Bicubic Interpolation**: Menggunakan 16 piksel terdekat untuk menghitung nilai piksel baru, menghasilkan hasil yang lebih halus dibandingkan bilinear.

Di MATLAB, skalasi citra dapat dilakukan dengan menggunakan fungsi-fungsi built-in yang sederhana dan efisien. Berikut adalah contoh bagaimana melakukan skalasi citra menggunakan MATLAB:

1. **Membaca Gambar**: Menggunakan imread untuk memuat gambar dari file.

```
img = imread('image.jpg');
imshow(img);
```

2. **Mengubah Ukuran Gambar**: Untuk melakukan skalasi, gunakan fungsi imresize. Fungsi ini memungkinkan kita untuk memperbesar atau memperkecil citra sesuai dengan faktor skala atau ukuran target.

Upscaling (memperbesar citra)

```
% Langkah 1: Membaca gambar
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Langkah 2: Memperbesar gambar dengan faktor 2
scaled_img_up = imresize(img, 2);  % Memperbesar gambar menjadi dua kali ukuran asli
% Langkah 3: Menampilkan gambar asli dan gambar yang diperbesar dalam satu figur
figure;  % Membuat figur baru
% Menampilkan gambar asli di posisi pertama (1 baris, 2 kolom, posisi 1)
subplot(1, 2, 1);  % Membagi area menjadi 1 baris, 2 kolom, gambar pertama
imshow(img);
title('Gambar Asli');
axis image;  % Menjaga rasio aspek gambar tetap (tidak terdistorsi)
% Menampilkan gambar yang diperbesar di posisi kedua (1 baris, 2 kolom, posisi 2)
subplot(1, 2, 2);  % Gambar kedua di kolom kedua
imshow(scaled_img_up);
title('Gambar yang Diperbesar');
axis image;  % Menjaga rasio aspek gambar tetap (tidak terdistorsi)
% Mengatur agar subplot kedua menampilkan gambar yang lebih besar
set(gca, 'Position', [0.55, 0.1, 0.4, 0.8]);  % Mengubah posisi dan ukuran subplot kedua
```

Hasil:

Gambar 8.3: Citra Asli dan Citra setelah diperbesar

Downscaling (memperkecil citra)

```
% Langkah 1: Membaca gambar asli
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Langkah 2: Memperkecil gambar dengan faktor 0.5
scaled_img_down = imresize(img, 0.5);  % Memperkecil gambar menjadi setengah ukuran asli
% Langkah 3: Menampilkan ukuran gambar asli dan yang diperkecil
disp('Ukuran Gambar Asli:');
disp(size(img));  % Menampilkan ukuran gambar asli
disp('Ukuran Gambar yang Diperkecil:');
disp(size(scaled_img_down));  % Menampilkan ukuran gambar yang diperkecil
% Langkah 4: Membuat figur baru dengan ukuran yang lebih besar
figure;
% Menampilkan gambar asli di posisi pertama (1 baris, 2 kolom, posisi 1)
subplot(1, 2, 1);  % Membagi area menjadi 1 baris, 2 kolom, gambar pertama
imshow(img);
title('Gambar Asli');
axis image;  % Menjaga rasio aspek gambar tetap (tidak terdistorsi)
% Menampilkan gambar yang diperkecil di posisi kedua (1 baris, 2 kolom, posisi 2)
subplot(1, 2, 2);  % Gambar kedua di kolom kedua
imshow(scaled_img_down, 'InitialMagnification', 50);  % Mengatur skala awal untuk gambar yang diperkecil
title('Gambar yang Diperkecil');
axis image;  % Menjaga rasio aspek gambar tetap (tidak terdistorsi)
% Langkah 5: Menyesuaikan ukuran figur untuk memastikan gambar yang diperkecil lebih kecil terlihat
set(gcf, 'Position', [100, 100, 1200, 600]);  % Mengatur ukuran figur yang lebih besar
```

Hasil:

Gambar 8.4: Citra Asli dan Citra setelah diperkecil

8.4 Shearing (Penyusutan)

Shearing (Penyusutan) pada Citra adalah salah satu transformasi geometri yang digunakan untuk mengubah bentuk citra dengan cara memanipulasi koordinat pixel citra (Yelliy N, 2019). Pada dasarnya, proses shearing ini akan merubah bentuk citra dengan "menarik" satu bagian citra dalam arah horizontal atau vertikal, tanpa mengubah area atau ukuran total citra tersebut.

Shearing dapat diterapkan dengan cara:

1. **Shearing Horizontal**: Memindahkan pixel dalam arah horizontal (sumbu x), sedangkan posisi vertikal (sumbu y) tetap.
2. **Shearing Vertikal**: Memindahkan pixel dalam arah vertikal (sumbu y), sedangkan posisi horizontal (sumbu x) tetap.

Rumus untuk Shearing:

- Untuk shearing horizontal:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x + sh_x \,.\, y \\ y \end{bmatrix}$$

Di mana shxsh_x adalah faktor shearing horizontal.

- Untuk shearing vertikal:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y + sh_x \,.\, y \end{bmatrix}$$

Di mana sh_y adalah faktor shearing vertikal.

Implementasi Shearing dalam Pemrograman MATLAB

Untuk melakukan shearing pada citra di MATLAB, kita perlu menggunakan matriks transformasi dan fungsi imtransform atau affine2d untuk menerapkan transformasi tersebut.

Langkah-langkah untuk Shearing pada Citra:

1. **Baca Citra**: Pertama, muat citra yang ingin Anda transformasi.
2. **Definisikan Matriks Transformasi**: Tentukan matriks shearing untuk horizontal atau vertikal.
3. **Gunakan Fungsi Transformasi**: Terapkan transformasi pada citra menggunakan fungsi imwarp atau affine2d.

Contoh Kode MATLAB untuk Shearing:

```
% Langkah 1: Membaca gambar asli
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Langkah 2: Tentukan faktor shearing
sh_x = 0.5;  % Shearing Horizontal
sh_y = 0.3;  % Shearing Vertikal
% Matriks transformasi untuk shearing horizontal (3x3)
T = [1, sh_x, 0; 0, 1, 0; 0, 0, 1];  % Matriks untuk shearing horizontal
% Atau matriks untuk shearing vertikal (3x3)
% T = [1, 0, 0; sh_y, 1, 0; 0, 0, 1];  % Matriks untuk shearing vertikal
% Langkah 3: Lakukan transformasi menggunakan imwarp
tform = affine2d(T);  % Matriks transformasi affine
output_img = imwarp(img, tform);  % Terapkan transformasi pada citra
% Langkah 4: Tampilkan hasilnya
subplot(1, 2, 1);
imshow(image);
title('Citra Asli');
subplot(1, 2, 2);
imshow(output_img);
title('Citra setelah Shearing');
```

Hasil:

Gambar 8.5: Citra Asli dan Citra setelah Shearing

Penjelasan Kode:

1. **Baca Citra**: Menggunakan imread untuk memuat citra yang akan dishearing.
2. **Definisi Matriks Shearing**: Matriks T mendefinisikan bagaimana shearing diterapkan. Matriks untuk shearing horizontal memindahkan piksel berdasarkan nilai sh_x (faktor shearing horizontal), sedangkan matriks untuk shearing vertikal memindahkan piksel berdasarkan nilai sh_y.
3. **Transformasi Citra**: affine2d digunakan untuk membuat objek transformasi affine, dan imwarp digunakan untuk menerapkan transformasi ini ke citra.
4. **Tampilkan Hasil**: imshow digunakan untuk menampilkan citra yang sudah mengalami transformasi shearing.

Dengan pendekatan ini, citra akan terlihat terdistorsi, dimana bentuknya akan tergerak dalam arah horizontal atau vertikal tergantung dari matriks transformasi yang dipilih.

8.5 Flipping (Pembalikan)

Flipping atau pembalikan pada citra adalah teknik transformasi yang digunakan untuk membalikkan citra secara horizontal atau vertikal (Zufar, 1998). Proses ini mengubah posisi piksel dalam citra berdasarkan arah yang ditentukan. Flipping sering digunakan dalam pengolahan citra untuk berbagai tujuan, seperti augmentasi data pada pemodelan pembelajaran mesin atau untuk memperbaiki orientasi citra.

Jenis Flipping

1. **Flipping Horizontal**:
   Pembalikan citra secara horizontal berarti membalikkan citra dari kiri ke kanan. Proses ini akan mengubah posisi piksel sehingga kolom pertama menjadi kolom terakhir, kolom kedua menjadi kolom kedua terakhir, dan seterusnya.
2. **Flipping Vertikal**:
   Pembalikan citra secara vertikal berarti membalikkan citra dari atas ke bawah. Proses ini akan mengubah posisi piksel sehingga baris pertama menjadi baris terakhir, baris kedua menjadi baris kedua terakhir, dan seterusnya.

Di MATLAB, flipping citra bisa dilakukan menggunakan fungsi built-in seperti fliplr() untuk flipping horizontal dan flipud() untuk flipping vertikal.

8.5.1 Flipping Horizontal (fliplr)

Fungsi fliplr() digunakan untuk membalikkan citra secara horizontal (kiri ke kanan).

```
% Membaca citra
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Memastikan bahwa citra memiliki 3 dimensi (RGB)
```

```
if ndims(img) == 3
    % Melakukan flipping horizontal pada seluruh citra (RGB)
    img_flipped_h = img(:, end:-1:1, :);
else
    % Jika citra hanya 2D (grayscale), langsung gunakan fliplr
    img_flipped_h = fliplr(img);
end
% Menampilkan citra asli dan citra setelah flipping horizontal
subplot(1, 2, 1);
imshow(img);
title('Citra Asli');
subplot(1, 2, 2);
imshow(img_flipped_h);
title('Citra Flipping Horizontal');
```

Hasil:

Gambar 8.6: Citra Asli dan membalikkan citra secara horizontal (kiri ke kanan)

8.5.2 Flipping Vertikal (flipud)

Fungsi flipud() digunakan untuk membalikkan citra secara vertikal (atas ke bawah).

```
% Membaca citra
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Memastikan bahwa citra memiliki 3 dimensi (RGB)
if ndims(img) == 3
    % Melakukan flipping vertikal pada seluruh citra (RGB)
    img_flipped_v = img(end:-1:1, :, :);
else
    % Jika citra hanya 2D (grayscale), langsung gunakan flipud
    img_flipped_v = flipud(img);
end
% Menampilkan citra asli dan citra setelah flipping vertikal
subplot(1, 2, 1);
imshow(img);
title('Citra Asli');
subplot(1, 2, 2);
imshow(img_flipped_v);
title('Citra Flipping Vertikal');
```

Hasil:

Gambar 8.7: Citra Asli dan membalikkan citra secara vertikal

8.5.3 Flipping Kedua Arah (Horizontal dan Vertikal)

Melakukan flipping kedua arah, yaitu horizontal dan vertikal, dengan menggunakan kombinasi kedua fungsi tersebut.

```
% Membaca citra
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Melakukan flipping horizontal dan vertikal pada citra RGB
img_flipped_both = img(end:-1:1, end:-1:1, :);  % Membalikkan citra secara horizontal dan vertikal
% Menampilkan citra asli dan citra setelah pembalikan
subplot(1, 2, 1);
imshow(img);
title('Citra Asli');
subplot(1, 2, 2);
imshow(img_flipped_both);
title('Citra Flipping Horizontal dan Vertikal');
```

Hasil:

Gambar 8.8: Citra Asli dan membalikkan citra secara vertikal dan horisontal

Penjelasan Kode:
imread('image.jpg'): Membaca citra dari file yang diberikan (misalnya 'image.jpg').
fliplr(img): Melakukan pembalikan citra secara horizontal.
flipud(img): Melakukan pembalikan citra secara vertikal.
subplot(1, 2, x): Menampilkan dua citra dalam satu jendela, dengan x menentukan posisi citra pertama dan kedua.
imshow(): Menampilkan citra pada jendela MATLAB.

8.6 Perspektif (Keprojekan)

Perspektif atau **keprojekan** dalam pengolahan citra merujuk pada transformasi geometris yang mengubah pandangan atau sudut tampilan citra seolah-olah dilihat dari posisi atau sudut pandang yang berbeda (Asiva Noor Rachmayani, 2015b). Hal ini sering disebut juga sebagai **proyeksi perspektif**.

Proyeksi perspektif mengubah bentuk objek dalam citra, di mana objek yang lebih dekat dengan kamera akan tampak lebih besar dan objek yang lebih jauh akan tampak lebih kecil. Transformasi ini digunakan untuk mensimulasikan bagaimana objek tampak dari sudut pandang tertentu, yang sangat penting dalam aplikasi seperti grafik komputer, fotogrametri, dan pemrosesan citra.

Jenis-jenis Transformasi Perspektif:

1. **Transformasi Perspektif 2D (Homografi)**:
    - Ini adalah jenis proyeksi perspektif yang mengubah tampilan citra 2D menjadi perspektif yang berbeda dengan cara memanipulasi posisi piksel.
    - Transformasi ini dapat dilakukan dengan menggunakan **matriks homografi** yang mengubah posisi titik-titik dalam citra sesuai dengan transformasi perspektif yang diinginkan.
2. **Proyeksi Perspektif dalam Pengolahan Citra**:
    - Transformasi ini digunakan untuk mengubah citra ke perspektif baru dengan memetakan piksel citra ke posisi yang baru berdasarkan matriks transformasi perspektif.
    - Transformasi perspektif ini dapat digunakan untuk memperbaiki distorsi pada gambar yang diambil dengan sudut pandang yang tidak ideal atau untuk membuat efek visual tertentu.

Di MATLAB, transformasi perspektif bisa dilakukan menggunakan fungsi **imtransform** atau **projective2d** bersama dengan **matriks transformasi perspektif** yang diterapkan ke citra.

Langkah-langkah untuk Menerapkan Perspektif di MATLAB:

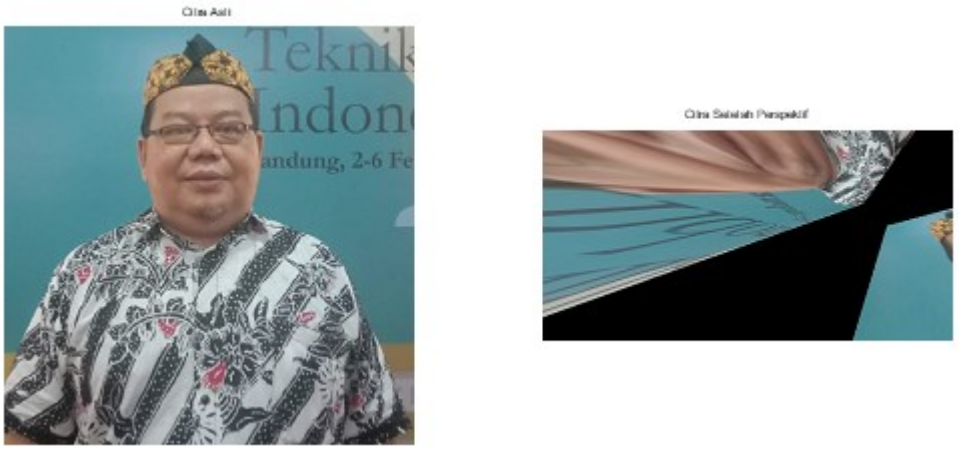
1. **Menentukan Matriks Transformasi Perspektif:**
    - Matriks perspektif 3x3 digunakan untuk memetakan koordinat citra ke posisi yang baru. Matriks ini dapat dihitung atau diberikan jika Anda sudah mengetahui pasangan titik kontrol antara citra asal dan citra target.
2. **Melakukan Transformasi Perspektif pada Citra:**
    - Setelah matriks transformasi dibuat, Anda bisa menggunakan fungsi **imwarp** untuk menerapkan transformasi pada citra.

Contoh Kode Pemrograman Perspektif Menggunakan MATLAB:

1. Menggunakan Fungsi projective2d untuk Transformasi Perspektif

```
% Membaca citra
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menentukan titik asal (titik dalam citra asli)
input_points = [50 50; 150 50; 150 150; 50 150]; % Contoh titik dalam citra asli
% Menentukan titik tujuan (titik di citra yang sudah diproyeksikan)
output_points = [60 60; 160 40; 170 160; 40 160]; % Titik setelah perspektif
% Menghitung matriks transformasi perspektif dari titik asal ke titik tujuan
tform = fitgeotrans(input_points, output_points, 'projective');
% Menerapkan transformasi perspektif pada citra
img_transformed = imwarp(img, tform);
% Menampilkan citra asli dan citra setelah transformasi perspektif
subplot(1, 2, 1);
imshow(img);
title('Citra Asli');
subplot(1, 2, 2);
imshow(img_transformed);
title('Citra Setelah Perspektif');
```

Hasil:



Gambar 8.9: Citra Asli dan citra setelah transformasi perspektif

Penjelasan Kode:

1. **input_points dan output_points**: Ini adalah titik koordinat yang digunakan untuk mendefinisikan transformasi perspektif. input_points adalah titik pada citra asli, sementara output_points adalah titik-titik yang menunjukkan posisi yang diinginkan dalam citra yang telah diproyeksikan.
2. **fitgeotrans**: Fungsi ini digunakan untuk menghitung matriks transformasi perspektif (homografi) yang menghubungkan titik-titik asal dan titik tujuan.
3. **imwarp**: Fungsi ini digunakan untuk menerapkan transformasi perspektif pada citra menggunakan matriks transformasi yang telah dihitung sebelumnya.
4. **subplot dan imshow**: Digunakan untuk menampilkan citra asli dan citra hasil transformasi perspektif.

2. Menggunakan Matriks Transformasi Perspektif Secara Langsung

Jika Anda memiliki matriks transformasi perspektif 3x3, Anda dapat menggunakannya untuk melakukan transformasi manual pada citra.

```
% Membaca citra
inputImage = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menentukan empat titik sumber (misalnya, koordinat pada gambar input)
% Format: [x1, y1; x2, y2; x3, y3; x4, y4]
% Misalnya, pilih empat sudut dari gambar
sourcePoints = [100, 100; 400, 100; 100, 400; 400, 400];
% Menentukan empat titik tujuan pada perspektif baru
% Format: [x1, y1; x2, y2; x3, y3; x4, y4]
% Gantilah dengan koordinat yang diinginkan untuk perspektif baru
destinationPoints = [50, 50; 450, 80; 80, 450; 450, 450];
% Menghitung matriks transformasi perspektif
tform = fitgeotrans(sourcePoints, destinationPoints, 'projective');
% Menampilkan matriks transformasi perspektif
disp('Matriks Transformasi Perspektif:');
disp(tform.T);
% Menerapkan transformasi pada gambar
outputImage = imwarp(inputImage, tform);
% Menampilkan gambar hasil transformasi
subplot(1, 2, 1);
imshow(inputImage);
title('Gambar Asli');
subplot(1, 2, 2);
imshow(outputImage);
title('Gambar Setelah Transformasi Perspektif');
```

Hasil:

Gambar 8.10: Citra Asli dan citra setelah transformasi perspektif

Penjelasan Kode:

- **H**: Matriks transformasi perspektif 3x3 yang mendefinisikan bagaimana citra harus diproyeksikan. Matriks ini mengubah piksel citra sesuai dengan proyeksi perspektif yang diinginkan.
- **affine2d**: Mengonversi matriks transformasi ke objek yang dapat digunakan oleh fungsi **imwarp**.

8.7 Cropping (Pemotongan)

Pemotongan citra (cropping) adalah proses mengambil sebagian kecil dari gambar (citra) asli dengan memotongnya menggunakan koordinat tertentu (Putra et al., 2022). Tujuan dari pemotongan citra biasanya adalah untuk fokus pada bagian tertentu dari gambar, seperti objek atau area yang menarik, atau untuk mengurangi ukuran citra agar lebih efisien dalam penyimpanan atau pemrosesan.

Pada dasarnya, cropping pada citra melibatkan pemilihan sebuah area persegi panjang atau area berbentuk lainnya yang didefinisikan oleh dua pasangan koordinat (titik kiri atas dan kanan bawah) dari gambar asli.

Langkah-langkah Cropping Citra:

1. Tentukan koordinat area yang ingin dipotong. Biasanya dalam bentuk (x1, y1) untuk titik kiri atas dan (x2, y2) untuk titik kanan bawah.
2. Ambil bagian gambar yang berada di dalam batas yang ditentukan oleh koordinat tersebut.
3. Gambar baru (hasil cropping) berisi bagian citra yang diinginkan.

MATLAB adalah salah satu perangkat lunak yang sering digunakan untuk pemrosesan citra. Di bawah ini adalah contoh program MATLAB untuk melakukan cropping pada citra:

1. Membaca citra

Langkah pertama adalah membaca citra yang akan dipotong menggunakan fungsi imread().

```
% Membaca citra
img = imread('gambar.jpg');
imshow(img);  % Menampilkan citra asli
```

2. Menentukan koordinat cropping

Koordinat untuk cropping biasanya didefinisikan berdasarkan posisi titik kiri atas dan kanan bawah. Misalnya, kita ingin memotong bagian citra yang dimulai dari koordinat (100, 50) sampai (300, 200).

```
% Menentukan area yang ingin dipotong (x1, y1, x2, y2)
x1 = 100;
y1 = 50;
x2 = 300;
y2 = 200;

% Melakukan cropping
cropped_img = img(y1:y2, x1:x2, :);
imshow(cropped_img);  % Menampilkan citra yang telah dipotong
```

Pada kode di atas:

- img(y1:y2, x1:x2, :) melakukan pemotongan gambar berdasarkan koordinat yang telah ditentukan. Tanda : berarti mengambil semua nilai warna (RGB).
- Variabel cropped_img menyimpan citra yang sudah dipotong.

3. Menyimpan citra hasil cropping

Setelah citra dipotong, kita dapat menyimpan hasilnya ke file baru menggunakan fungsi imwrite().

```
% Menyimpan citra hasil cropping
imwrite(cropped_img, 'cropped_image.jpg');
```

kode program dijadikan jadi satu:

```
% Membaca citra
inputImage = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Menampilkan gambar asli
figure;
subplot(1, 2, 1);  % Membagi figure menjadi 1 baris dan 2 kolom, menampilkan gambar pertama di kolom pertama
imshow(img);
title('Gambar Asli');
% Menentukan area cropping
x1 = 100;
y1 = 50;
x2 = 300;
y2 = 200;
cropped_img = img(y1:y2, x1:x2, :);
% Menampilkan gambar hasil pemotongan
subplot(1, 2, 2);  % Menampilkan gambar kedua di kolom kedua
imshow(cropped_img);
title('Gambar Hasil Pemotongan');

% Menyimpan gambar hasil pemotongan ke file baru
imwrite(cropped_img, 'cropped_image.jpg');
```

Hasil:

Gambar 8.11: Citra Asli dan citra setelah hasil pemotongan

Penjelasan Kode:

1. imshow(img): Menampilkan citra asli di jendela figure.
2. subplot(1, 2, 1): Fungsi subplot digunakan untuk membagi figure menjadi grid 1x2. Angka pertama (1) menunjukkan jumlah baris, angka kedua (2) menunjukkan jumlah kolom, dan angka ketiga (1) menunjukkan posisi gambar yang ditampilkan di grid (gambar pertama di kolom pertama).
3. Menentukan area cropping: Koordinat (x1, y1) dan (x2, y2) digunakan untuk memotong gambar.
4. subplot(1, 2, 2): Menampilkan gambar hasil pemotongan pada kolom kedua.
5. title(): Menambahkan judul pada setiap gambar untuk membedakan gambar asli dan hasil pemotongan.

8.8 Resampling

Resampling pada Citra adalah proses perubahan ukuran atau transformasi data gambar dengan cara memperkirakan nilai-nilai baru untuk piksel dalam gambar tersebut (Danoedoro & Adiwijoyo, 2014). Tujuan resampling bisa untuk memperbesar atau memperkecil ukuran citra, atau untuk melakukan rotasi, transformasi geometris, atau distorsi lainnya.

Resampling dilakukan dengan cara menghitung nilai piksel pada posisi tertentu dengan metode interpolasi

tertentu. Ada beberapa metode yang digunakan dalam resampling citra, di antaranya:

1. **Nearest Neighbor**: Metode ini memilih nilai piksel terdekat dengan posisi yang baru. Ini adalah metode yang sangat cepat tetapi menghasilkan kualitas gambar yang kurang halus dan bisa memperlihatkan artefak (seperti blok-blok kasar).
2. **Bilinear Interpolation**: Dalam metode ini, nilai piksel pada posisi baru dihitung berdasarkan rata-rata tertimbang dari 4 piksel terdekat. Metode ini memberikan hasil yang lebih halus dibandingkan nearest neighbor, tetapi bisa menyebabkan sedikit kekaburan (blurring) pada citra.
3. **Bicubic Interpolation**: Metode ini menggunakan 16 piksel terdekat untuk menghitung nilai piksel baru dan lebih kompleks dibandingkan bilinear. Hasilnya lebih halus dan lebih baik dalam menangani transisi warna, tetapi memerlukan lebih banyak perhitungan.
4. **Spline Interpolation**: Merupakan metode interpolasi yang lebih canggih dan dapat menghasilkan gambar yang sangat halus. Biasanya digunakan untuk aplikasi yang membutuhkan ketelitian tinggi dalam proses resampling.

Pemrograman Resampling Citra dengan MATLAB: Di MATLAB, resampling citra dapat dilakukan dengan menggunakan beberapa fungsi bawaan yang mempermudah proses tersebut. Beberapa fungsi yang sering digunakan adalah:

1. **imresize**: Fungsi ini digunakan untuk memperbesar atau memperkecil ukuran citra. Anda dapat memilih metode interpolasi yang diinginkan, seperti 'nearest', 'bilinear', atau 'bicubic'.

Contoh kode:

```
% Membaca gambar
img = imread('image.jpg');
% Mengubah ukuran citra menjadi setengah ukuran asli menggunakan interpolasi bilinear
resized_img = imresize(img, 0.5, 'bilinear');
% Menampilkan gambar yang telah diubah ukuran
imshow(resized_img);
```

2. **imwarp**: Fungsi ini digunakan untuk transformasi geometris, termasuk rotasi, translasi, atau distorsi lain pada citra. Biasanya digunakan bersama dengan matriks transformasi yang telah didefinisikan.

Contoh kode untuk rotasi citra:

```
% Membaca gambar
img = imread('image.jpg');
% Membuat objek transformasi untuk rotasi
tform = affine2d([cosd(45) -sind(45) 0; sind(45) cosd(45) 0; 0 0 1]);
% Melakukan transformasi citra
rotated_img = imwarp(img, tform);
% Menampilkan gambar yang telah dirotasi
imshow(rotated_img);
```

3. **interp2**: Fungsi ini digunakan untuk melakukan interpolasi bilinear pada citra dua dimensi. Ini lebih fleksibel jika Anda ingin mengontrol metode interpolasi atau melakukan perhitungan secara manual.

Contoh kode interpolasi bilinear:

```
% Membaca gambar
img = imread('image.jpg');
img_gray = rgb2gray(img);  % Mengubah gambar menjadi grayscale
% Membuat grid untuk interpolasi
[X, Y] = meshgrid(1:size(img_gray,2), 1:size(img_gray,1));
% Menentukan ukuran output citra
[Xq, Yq] = meshgrid(linspace(1, size(img_gray,2), 500), linspace(1, size(img_gray,1), 500));
% Melakukan interpolasi bilinear
resized_img = interp2(X, Y, double(img_gray), Xq, Yq, 'linear');
% Menampilkan gambar hasil interpolasi
imshow(uint8(resized_img));
```

Soal-soal Latihan:

1. Jelaskan apa yang dimaksud dengan operasi geometri citra dalam pengolahan citra. Sebutkan beberapa contoh operasi geometri yang sering digunakan dan jelaskan bagaimana operasi tersebut memengaruhi citra.
2. Apa yang dimaksud dengan transformasi afine pada citra? Jelaskan bagaimana operasi transformasi afine

seperti rotasi, translasi, dan skala diterapkan pada citra. Berikan contoh kasus di mana transformasi afine digunakan dalam pengolahan citra.

3. Jelaskan perbedaan antara transformasi geometri bersifat linier dan non-linier pada citra. Berikan contoh penggunaan transformasi linier dan non-linier dalam pemrosesan citra.
4. Dalam MATLAB, bagaimana cara melakukan rotasi citra sebesar 45 derajat? Jelaskan langkah-langkah yang diperlukan dan tampilkan citra hasil rotasi. Sertakan contoh kode MATLAB untuk melakukan rotasi citra.
5. Pada pengolahan citra, scaling digunakan untuk mengubah ukuran citra. Jelaskan bagaimana cara melakukan operasi scaling pada citra di MATLAB dan bagaimana pengaruhnya terhadap kualitas citra. Berikan contoh kode untuk melakukan scaling citra dengan faktor tertentu.

Bab 9
Operasi Morfologi pada Citra

Operasi morfologi adalah teknik pengolahan citra yang sering digunakan untuk memanipulasi dan menganalisis bentuk atau struktur objek dalam citra biner (citra yang hanya memiliki dua nilai piksel: hitam dan putih). Operasi ini lebih fokus pada pengolahan citra berdasarkan bentuk dan struktur piksel, dan digunakan secara luas dalam analisis citra untuk mendeteksi objek, menghilangkan noise, serta melakukan transformasi bentuk objek (Dwdv et al., n.d.).
Operasi morfologi umumnya digunakan dalam citra biner, meskipun ada juga versi yang diterapkan pada citra grayscale. Beberapa operasi morfologi yang paling umum digunakan meliputi **dilasi**, **erosion**, **opening**, **closing**, **gradient**, dan **hit-or-miss transform**.

9.1 Dilasi (Dilation)

Dilasi adalah salah satu operasi dasar dalam pemrosesan citra yang digunakan dalam *morphological image processing*. Operasi ini berfungsi untuk memperbesar objek pada citra biner dengan cara menambahkan piksel ke tepi objek yang ada. Dalam dilasi, setiap piksel dari citra akan digantikan oleh nilai maksimum dari piksel sekitarnya, sesuai dengan ukuran kernel yang digunakan. Secara sederhana, dilasi memperluas wilayah objek yang ada di dalam citra (Yuwono, 2015).
Cara Kerja Dilasi:

1. Dilasi menggunakan sebuah struktur elemen (*structuring element*) yang berbentuk matriks kecil (biasanya berbentuk persegi atau lingkaran).
2. Untuk setiap piksel dalam citra biner, struktur elemen tersebut ditempatkan di sekitar piksel tersebut, dan nilai piksel di citra dilasi dengan nilai maksimum dari piksel-piksel yang ada dalam struktur elemen tersebut.
3. Hasil dari dilasi adalah citra baru di mana objek yang ada menjadi lebih besar (atau memperluas area objek).

Efek Dilasi pada Citra:

- Memperbesar objek dalam citra.
- Mengisi lubang atau celah kecil di dalam objek.
- Membuat objek lebih solid dengan menambahkan piksel pada tepi objek.

Di MATLAB, operasi dilasi dapat dilakukan dengan menggunakan fungsi imdilate. Fungsi ini membutuhkan dua input utama:

1. Citra biner yang akan didilasi.
2. Struktur elemen (biasanya berupa matriks atau objek khusus seperti strel).

Berikut adalah contoh kode MATLAB untuk dilasi citra biner:

```
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Mengonversi citra ke grayscale (jika citra berwarna)
I_gray = rgb2gray(I);
% Menentukan threshold (misalnya 128)
threshold = 128;
% Mengonversi citra menjadi biner berdasarkan threshold
BW = I_gray > threshold;
% Membuat struktur elemen (misalnya struktur elemen berbentuk persegi 3x3)
se = strel('square', 3);  % Membuat struktur elemen persegi dengan ukuran 3x3
% Melakukan operasi dilasi
BW_dilated = imdilate(BW, se);
% Menampilkan hasil
subplot(1,2,1);
imshow(BW);
title('Citra Biner Asli');
subplot(1,2,2);
```

```
imshow(BW_dilated);
title('Hasil Dilasi');
```
Hasil:

Gambar 9.1: Citra Biner Asli dan citra setelah hasil Dilasi

Penjelasan Kode:

1. **Baca citra**: imread('image.png') digunakan untuk membaca citra.
2. **Grayscale**: Jika citra berwarna, rgb2gray(I) akan mengonversinya menjadi citra grayscale.
3. **Thresholding**: Citra didefinisikan sebagai biner dengan membandingkan nilai piksel dengan ambang tertentu, dalam hal ini 128. Jika piksel lebih besar dari 128, itu akan menjadi 1 (putih), jika lebih kecil, itu akan menjadi 0 (hitam).
4. **Dilasi**: Setelah citra biner diperoleh, operasi dilasi dilakukan dengan imdilate.

9.2 Erosi (Erosion)

Erosi (Erosion) adalah salah satu operasi morfologi dalam pengolahan citra yang digunakan untuk mengurangi ukuran objek yang ada dalam citra biner. Erosi mengurangi objek di citra biner dengan cara menurunkan piksel objek sehingga objek yang ada akan "tererosi" atau mengecil (Said & Jambek, 2021).

Proses erosi dilakukan dengan menggeser struktur elemen (sebuah matriks kecil yang digunakan untuk memodifikasi citra) di atas citra biner. Jika seluruh bagian struktur elemen dapat ditempatkan pada piksel objek, maka piksel pusat dari struktur elemen tersebut akan tetap menjadi objek. Jika ada bagian dari struktur elemen yang berada di luar objek (misalnya di area latar belakang), maka piksel pusatnya akan menjadi latar belakang (0).

Ciri-ciri Erosi:

- Mengurangi ukuran objek.
- Memotong bagian objek yang lebih kecil atau terpisah.
- Memperkuat batas objek dan menghilangkan noise yang lebih kecil.

Langkah-langkah Erosi pada Citra:

1. Pilih Struktur Elemen: Biasanya berupa matriks 3x3 atau bentuk lain, yang digunakan untuk menilai piksel di sekitar piksel pusatnya.
2. Terapkan Operasi pada Setiap Piksel: Struktur elemen digeser ke seluruh citra, dan untuk setiap piksel pusat yang sedang dianalisis, jika semua piksel dalam struktur elemen berada dalam objek (nilai 1), maka piksel pusatnya tetap 1. Jika ada nilai 0, maka piksel pusat diubah menjadi 0.
3. Hasil: Citra yang mengalami erosi akan memiliki objek yang lebih kecil.

Di MATLAB, fungsi untuk melakukan erosi pada citra biner adalah imerode(). Fungsi ini memerlukan dua argumen, yaitu citra input dan struktur elemen yang digunakan untuk erosi.

Contoh Kode:

```
% Membaca citra biner
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\program\gambar_biner.jpg');
% Menampilkan citra asli
subplot(1, 2, 1);
imshow(citra);
title('Citra Asli');
% Membuat struktur elemen (misalnya 3x3 matriks)
```

```
struktur_elemen = strel('square', 3);  % atau bisa juga 'disk' untuk bentuk bulat
% Melakukan erosi pada citra
citra_erosion = imerode(citra, struktur_elemen);
% Menampilkan citra hasil erosi
subplot(1, 2, 2);
imshow(citra_erosion);
title('Citra setelah Erosi');
```

Hasil:

Gambar 9.2: Citra Biner Asli dan citra setelah hasil Erosi

Penjelasan Kode:

1. Membaca Citra: Menggunakan imread() untuk membaca citra biner yang akan diproses.
2. Struktur Elemen: Struktur elemen dibuat dengan menggunakan fungsi strel(). Dalam contoh ini, strel('square', 3) membuat struktur elemen berbentuk kotak 3x3. Anda bisa mengganti bentuk struktur elemen, seperti menggunakan strel('disk', 3) untuk bentuk bulat.
3. Operasi Erosi: Fungsi imerode() digunakan untuk melakukan operasi erosi dengan struktur elemen yang telah dibuat.
4. Menampilkan Hasil: imshow() digunakan untuk menampilkan citra asli dan citra hasil erosi dalam satu jendela.

Contoh Hasil:

- Citra sebelum erosi mungkin memiliki objek yang lebih besar dan lebih terhubung.
- Setelah erosi, objek-objek tersebut akan lebih kecil dan terpisah, dan bagian yang lebih kecil atau noise akan hilang.

9.3 Opening

Opening adalah operasi morfologi dalam pemrosesan citra yang terdiri dari dua tahap utama: erosion (erosi) diikuti oleh dilasi (dilasi). Tujuan dari operasi ini adalah untuk menghilangkan noise kecil yang ada pada citra dan mempertahankan bentuk objek utama dalam citra. Opening sering digunakan untuk menghilangkan elemen kecil atau gangguan yang tidak diinginkan pada citra, seperti noise atau detail kecil yang tidak relevan (Trianto et al., 2022).

Langkah-langkah Opening:

1. Erosi (Erosion): Mengurangi objek dengan "memotong" bagian tepinya. Hal ini menghilangkan elemen-elemen kecil pada citra dan memperkecil objek.
2. Dilasi (Dilation): Setelah erosi, dilasi digunakan untuk memperbesar kembali objek yang telah diperkecil, namun bentuk objek utama tetap lebih bersih dari noise yang lebih kecil.

Operasi Opening ini cenderung memfilter noise kecil, sehingga objek yang lebih besar tetap dipertahankan.

Kegunaan Opening pada Citra:

- Menghilangkan noise atau titik-titik kecil pada citra biner.
- Memperbaiki bentuk objek dalam citra, seperti menghaluskan batas objek.
- Memisahkan objek yang terhubung oleh noise kecil.

Di MATLAB, operasi morfologi seperti opening dapat dilakukan dengan menggunakan fungsi imopen(), yang sudah termasuk dalam toolbox Image Processing Toolbox. Berikut adalah contoh bagaimana cara melakukan operasi opening pada citra menggunakan MATLAB:

```
% Membaca citra grayscale atau biner
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Jika citra berwarna, konversi ke citra grayscale
if size(I, 3) == 3
   I = rgb2gray(I);
end
% Tentukan nilai threshold (misalnya 128)
threshold = 128;
% Mengonversi citra ke biner dengan thresholding manual
BW = I > threshold;  % Pixel dengan nilai lebih dari threshold menjadi 1 (putih), sisanya 0 (hitam)
% Menentukan ukuran struktur elemen untuk operasi morfologi
se = strel('disk', 5);  % Struktur elemen berbentuk disk dengan radius 5 piksel
% Melakukan operasi Opening
BW_opened = imopen(BW, se);
% Menampilkan hasilnya
subplot(1, 2, 1);
imshow(BW);
title('Citra Biner Asli');
subplot(1, 2, 2);
imshow(BW_opened);
title('Citra Setelah Operasi Opening');
```

Hasil:

Gambar 9.3: Citra Biner Asli dan citra setelah hasil Operasi Opening

Penjelasan Kode:

1. Thresholding Manual: BW = I > threshold; mengonversi citra grayscale menjadi citra biner dengan membandingkan setiap pixel dengan nilai threshold. Pixel yang lebih besar dari threshold menjadi 1 (putih), sementara sisanya menjadi 0 (hitam).
2. strel('disk', 5): Membuat struktur elemen berbentuk disk dengan radius 5 piksel untuk operasi morfologi.
3. imopen(): Melakukan operasi opening pada citra biner yang telah dihasilkan.

9.4 Closing

Closing dalam pengolahan citra adalah operasi morfologi yang menggabungkan dua operasi dasar, yaitu dilasi (dilation) dan erosion. Pada dasarnya, operasi closing bertujuan untuk menghilangkan lubang-lubang kecil atau titik-titik hitam pada objek putih (foreground) dalam gambar, serta memperhalus tepi objek (Trianto et al., 2022).

Konsep Operasi Closing:

1. Dilation (Dilasi): Operasi yang mengembang bentuk objek dalam citra. Dengan dilasi, piksel objek akan 'meluas' ke area sekitarnya.
2. Erosion (Erosi): Operasi yang mengerutkan atau mengecilkan objek dalam citra. Erosi menghilangkan piksel objek yang terletak di pinggiran.

Closing dilakukan dengan urutan:

- Melakukan dilasi pada citra terlebih dahulu.
- Kemudian, melakukan erosi pada citra yang sudah ter-dilasi tersebut.

Tujuan utama dari closing adalah untuk mengisi lubang-lubang kecil (holes) di dalam objek atau untuk menghilangkan noise kecil yang ada di citra.

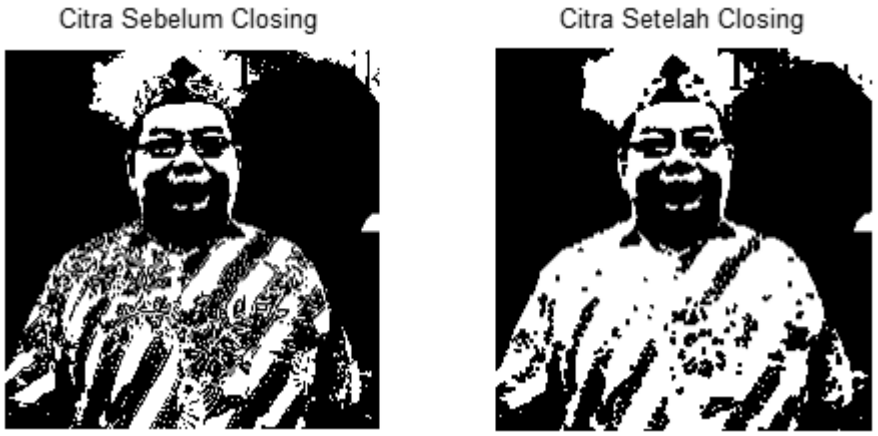
Langkah-langkah Closing pada Citra:

1. Dilasi: Memperluas objek (misalnya objek putih) pada citra biner.
2. Erosi: Mengurangi objek yang telah diperluas, mengembalikan citra ke bentuk semula, tetapi lubang atau noise yang kecil sudah tertutup.

Dalam MATLAB, untuk melakukan operasi morfologi seperti closing, kamu bisa menggunakan fungsi imclose yang sudah disediakan oleh MATLAB. Berikut adalah contoh pemrograman closing pada citra biner menggunakan imclose:

```
% Membaca citra input
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Jika citra berwarna, ubah menjadi grayscale terlebih dahulu
if size(I, 3) == 3
   I = rgb2gray(I);
end
% Menggunakan metode Otsu untuk mendapatkan threshold otomatis
level = graythresh(I);
% Mengonversi citra grayscale menjadi citra biner berdasarkan threshold Otsu
BW = I > level * 255; % Graythresh memberikan level antara 0 dan 1, jadi kalikan dengan 255
% Menentukan elemen struktur untuk operasi morfologi
se = strel('disk', 5); % Elemen struktur berbentuk disk dengan radius 5 piksel
% Melakukan operasi closing
BW_closed = imclose(BW, se);
% Menampilkan hasil sebelum dan sesudah closing
subplot(1, 2, 1);
imshow(BW);
title('Citra Sebelum Closing');
subplot(1, 2, 2);
imshow(BW_closed);
title('Citra Setelah Closing');
```

Hasil:



Gambar 9.4: Citra sebelum closing dan citra setelah closing

Penjelasan Kode:

- rgb2gray(I): Jika citra Anda berwarna, Anda harus mengubahnya menjadi grayscale terlebih dahulu dengan rgb2gray.
- graythresh(I): Menggunakan metode Otsu untuk menghitung threshold otomatis, dan kemudian kita kalikan dengan 255 karena graythresh memberikan nilai antara 0 dan 1, sementara citra uint8 memiliki rentang 0-255.

- I > level * 255: Mengonversi citra grayscale menjadi citra biner berdasarkan threshold yang dihitung.

Aplikasi Closing dalam Pengolahan Citra:

- Menghilangkan Noise: Menghapus noise kecil yang ada dalam citra, terutama yang berbentuk titik-titik kecil di latar belakang.
- Menutupi Lubang dalam Objek: Mengisi lubang atau celah kecil dalam objek pada citra biner.
- Meningkatkan Kualitas Citra: Menjaga objek utama tetap utuh sambil menghapus elemen-elemen kecil yang tidak diinginkan.

Dengan menggunakan closing, citra yang awalnya memiliki lubang kecil atau noise dapat menjadi lebih bersih dan lebih terstruktur.

9.5 Gradient

Operasi morfologi Gradient adalah salah satu teknik dalam pengolahan citra yang digunakan untuk mendeteksi tepi atau perbedaan intensitas piksel antara objek dan latar belakang. Dalam konteks morfologi, gradient adalah perbedaan antara hasil operasi dilasi (dilatation) dan erosi (erosion) pada citra biner atau citra grayscale (Bone et al., n.d.). Operasi ini menyoroti batas objek atau tepi, dan sering digunakan untuk mendeteksi peralihan intensitas secara tajam.

Pada citra biner atau grayscale, operasi gradient didefinisikan sebagai:

Gradient(I) = Dilasi(I) − Erosi(I)\

- Dilasi (Dilatation): Proses ini memperluas objek dalam citra, dengan menggantikan nilai piksel di area objek dengan nilai maksimum di sekitarnya.
- Erosi (Erosion): Proses ini mengurangi ukuran objek dengan menggantikan nilai piksel di area objek dengan nilai minimum di sekitarnya.

Hasil dari operasi gradient akan menonjolkan tepi atau perbedaan intensitas antara area objek dan latar belakang citra.

Aplikasi Operasi Gradient

- Deteksi Tepi: Menyoroti peralihan intensitas yang tajam, seperti perbatasan antara objek dan latar belakang.
- Penyaringan dan Penghalusan Citra: Mengurangi noise dengan cara menyoroti perbedaan antara objek dan latar belakang.

Berikut adalah contoh implementasi operasi morfologi Gradient pada citra menggunakan MATLAB.

1. Membaca citra dan konversi ke grayscale

```
% Membaca citra
I = imread('gambar.jpg');
% Mengkonversi citra ke grayscale jika citra berwarna
Igray = rgb2gray(I);
```

2. Melakukan Operasi Gradient

Pada citra grayscale, kita bisa langsung menggunakan fungsi imdilate dan imerode untuk melakukan dilasi dan erosi, kemudian mengurangkan keduanya untuk mendapatkan hasil gradient.

```
% Membuat structuring element (elemen struktural) yang digunakan untuk dilasi dan erosi
se = strel('disk', 3);  % Disk dengan radius 3 piksel
% Dilasi citra
I_dilated = imdilate(Igray, se);
% Erosi citra
I_eroded = imerode(Igray, se);
% Menghitung Gradient
I_gradient = I_dilated - I_eroded;
% Menampilkan hasil
subplot(1,3,1), imshow(Igray), title('Citra Asli');
subplot(1,3,2), imshow(I_gradient), title('Gradient');
```

3. Menampilkan hasil

Kode di atas akan menampilkan tiga gambar:

- Citra Asli: Citra grayscale yang telah dibaca.
- Gradient: Hasil operasi gradient yang menunjukkan perbedaan antara dilasi dan erosi.

Kode lengkapnya:

```
% Langkah 1: Membaca citra
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan nama file gambar Anda
% Langkah 2: Mengkonversi citra ke grayscale jika citra berwarna
Igray = rgb2gray(I);  % Jika citra sudah grayscale, langkah ini bisa dilewati
% Langkah 3: Membuat structuring element (elemen struktural)
% Elemen struktural berupa disk dengan radius 3 piksel
se = strel('disk', 3);
% Langkah 4: Melakukan dilasi pada citra
I_dilated = imdilate(Igray, se);
% Langkah 5: Melakukan erosi pada citra
I_eroded = imerode(Igray, se);
% Langkah 6: Menghitung hasil operasi gradient (selisih antara dilasi dan erosi)
I_gradient = I_dilated - I_eroded;
% Langkah 7: Menampilkan hasil
figure;
% Menampilkan citra asli
subplot(1, 3, 1);
imshow(Igray);
title('Citra Asli');
% Menampilkan citra hasil gradient
subplot(1, 3, 2);
imshow(I_gradient);
title('Hasil Gradient');
% Menampilkan citra dilasi
subplot(1, 3, 3);
imshow(I_dilated);
title('Citra Dilasi');
```

Hasil:

Gambar 9.5: Citra Asli dan citra setelah hasil Gradient

Penjelasan Langkah-langkah Kode:

1. **Membaca citra**:
   - Citra dibaca menggunakan imread. Pastikan untuk mengganti 'gambar.jpg' dengan nama file citra yang sesuai.
2. **Konversi ke grayscale**:
   - Citra yang berwarna diubah menjadi grayscale menggunakan rgb2gray. Jika citra sudah dalam format grayscale, langkah ini bisa dilewati.
3. **Membuat structuring element**:
   - strel('disk', 3) membuat elemen struktural berbentuk disk dengan radius 3 piksel yang akan digunakan pada operasi dilasi dan erosi.
4. **Operasi Dilasi**:
   - imdilate(Igray, se) digunakan untuk memperbesar objek dalam citra dengan menggantikan nilai piksel dengan nilai maksimum dari sekitarnya.
5. **Operasi Erosi**:

- imerode(Igray, se) digunakan untuk mengecilkan objek dalam citra dengan menggantikan nilai piksel dengan nilai minimum dari sekitarnya.

6. **Operasi Gradient**:
   - Hasil gradient dihitung dengan cara mengurangkan citra yang telah didilasi dan dierosikan, yang akan menonjolkan tepi atau perbedaan intensitas pada citra.
7. **Menampilkan Hasil**:
   - subplot digunakan untuk menampilkan tiga gambar dalam satu window:
     - **Citra Asli**: Menampilkan citra grayscale asli.
     - **Hasil Gradient**: Menampilkan hasil operasi gradient yang menunjukkan tepi.
     - **Citra Dilasi**: Menampilkan citra setelah operasi dilasi.

9.6 Hit-or-Miss Transform

Hit-or-Miss Transform (HMT) adalah salah satu operasi dalam morfologi citra yang digunakan untuk mendeteksi objek atau pola tertentu dalam citra biner. Operasi ini terutama digunakan untuk mendeteksi fitur spesifik seperti titik atau bentuk objek yang ada dalam citra (Hendrawan et al., 2021).

Hit-or-Miss Transform bekerja dengan menggunakan dua bentuk struktur elemen (SE) yang saling melengkapi, satu untuk "hit" dan satu lagi untuk "miss". Dalam konteks ini:

- Hit: Struktur elemen pertama mencocokkan objek pada citra.
- Miss: Struktur elemen kedua mencocokkan latar belakang atau area kosong dari citra.

Tujuan dari HMT adalah untuk mencari lokasi dimana salah satu struktur elemen (hit) mencocokkan objek pada citra, sementara struktur elemen lainnya (miss) mencocokkan bagian yang kosong di sekitar objek tersebut.

Secara matematis, operasi ini dapat diartikan sebagai:

1. Operasi hit (menggunakan struktur elemen pertama): melakukan operasi penutupan atau erosi untuk mencocokkan objek dengan struktur elemen pertama.
2. Operasi miss (menggunakan struktur elemen kedua): melakukan operasi dilasi untuk mencocokkan area kosong dengan struktur elemen kedua.
3. Hasil dari HMT adalah hasil dari operasi AND antara hasil hit dan miss.

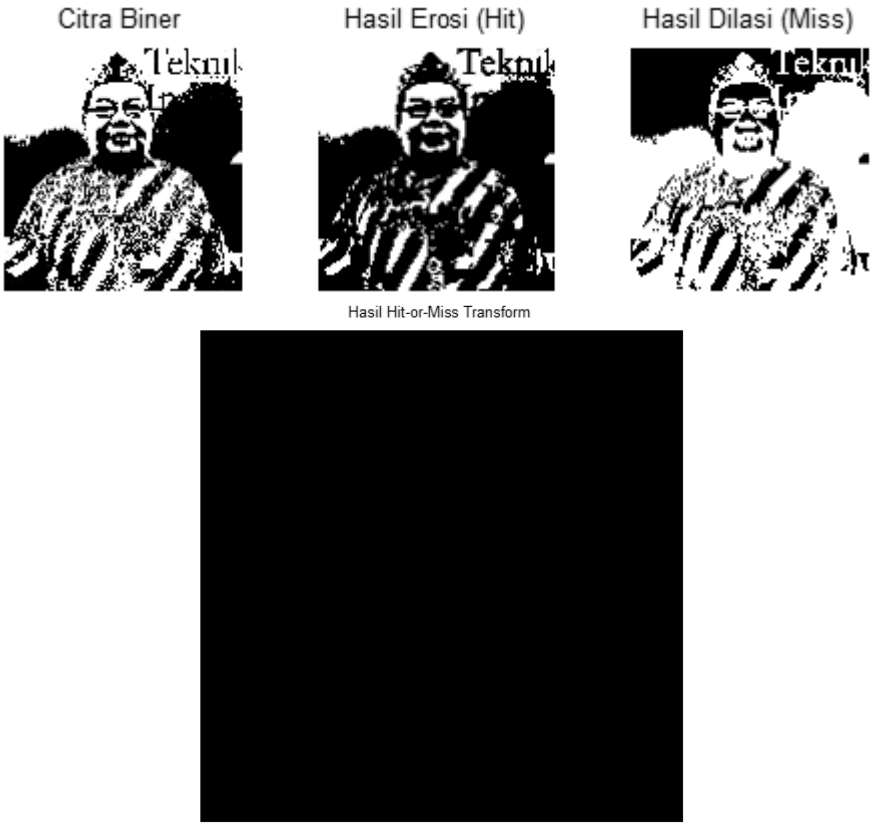
Langkah-langkah Hit-or-Miss Transform:

1. Erosi dengan struktur elemen pertama (SE1), menghasilkan citra yang menyaring bagian-bagian yang tidak cocok.
2. Dilasi dengan struktur elemen kedua (SE2), untuk memeriksa bagian kosong dari citra.
3. Operasi AND antara hasil erosi dan dilasi memberikan hasil akhir yang menandakan apakah ada kecocokan pada lokasi tertentu dalam citra.

Berikut adalah contoh implementasi dari Hit-or-Miss Transform dalam MATLAB:

```
% Membaca citra biner
A = imread('E:\HINDARTO\2024\Buku Ajar 2024\program\gambar_biner.jpg');
% Definisikan Struktur Elemen
SE1 = strel('disk', 3); % Struktur elemen pertama (misalnya disk dengan radius 3)
SE2 = strel('disk', 2); % Struktur elemen kedua (misalnya disk dengan radius 2)
% Lakukan operasi erosi dengan SE1
hit = imerode(A, SE1);
% Lakukan operasi dilasi dengan SE2 pada invers citra
miss = imdilate(~A, SE2);
% Hit-or-Miss Transform (AND antara hit dan miss)
HMT1 = hit & miss;
% Tampilkan hasilnya
figure;
subplot(1,3,1), imshow(A), title('Citra Biner');
subplot(1,3,2), imshow(hit), title('Hasil Erosi (Hit)');
subplot(1,3,3), imshow(miss), title('Hasil Dilasi (Miss)');
figure;
imshow(HMT1), title('Hasil Hit-or-Miss Transform');
```

Hasil:



Gambar 9.6: Citra Asli dan citra setelah hasil Hit-or-Miss Transform (HMT)

Penjelasan Kode:

1. **imread('citra_biner.png')**: Membaca citra biner input.
2. **strel('disk', radius)**: Membuat struktur elemen berupa disk dengan radius yang ditentukan.
3. **imerode(A, SE1)**: Melakukan operasi erosi pada citra A dengan struktur elemen SE1.
4. **imdilate(~A, SE2)**: Melakukan operasi dilasi pada invers dari citra A dengan struktur elemen SE2.
5. **hit & miss**: Menggabungkan hasil erosi dan dilasi menggunakan operasi logika AND.
6. **imshow()**: Menampilkan citra.

Aplikasi Hit-or-Miss Transform:

- Deteksi titik: Misalnya mendeteksi titik atau pola khusus dalam citra.
- Penyaringan objek: Memfilter objek berdasarkan bentuk atau karakteristik tertentu.
- Pengolahan citra biner: Digunakan dalam berbagai teknik pemrosesan citra biner untuk mengidentifikasi dan mendeteksi objek dengan bentuk tertentu.

Soal-soal Latihan:

1. Jelaskan apa yang dimaksud dengan operasi morfologi pada citra. Sebutkan beberapa operasi morfologi dasar yang sering digunakan dalam pengolahan citra biner dan jelaskan fungsi masing-masing operasi tersebut.
2. Apa yang dimaksud dengan operasi dilasi (dilation) dalam morfologi citra? Jelaskan bagaimana operasi dilasi memengaruhi bentuk objek dalam citra biner dan berikan contoh aplikasinya.
3. Jelaskan bagaimana operasi erosi (erosion) bekerja pada citra biner dan apa dampaknya terhadap objek dalam citra. Sebutkan situasi di mana operasi erosi lebih bermanfaat daripada dilasi.
4. Pada citra biner, kombinasi antara operasi dilasi dan erosi dapat digunakan untuk berbagai tujuan. Jelaskan apa yang dimaksud dengan operasi opening dan closing, serta bagaimana masing-masing operasi tersebut diterapkan untuk memproses citra biner.
5. Dalam MATLAB, bagaimana cara menerapkan operasi dilasi dan erosi pada citra biner? Jelaskan cara menggunakan fungsi imdilate dan imerode untuk melakukan kedua operasi tersebut, dan tampilkan hasilnya. Sertakan contoh kode MATLAB untuk menerapkan operasi morfologi pada citra.

Bab 10
Segmentasi Citra

Segmentasi citra adalah proses dalam pengolahan citra untuk membagi citra menjadi beberapa bagian atau region yang lebih kecil yang lebih mudah untuk dianalisis. Tujuan utama dari segmentasi adalah untuk memisahkan objek atau fitur yang relevan dari latar belakang, serta memudahkan analisis lebih lanjut, seperti pengenalan objek, pelacakan, atau klasifikasi (Orisa & Hidayat, 2019).
Segmentasi citra sangat penting dalam banyak aplikasi pengolahan citra, termasuk pengenalan objek, analisis medis (seperti pemindaian CT atau MRI), pengawasan video, dan sistem visi komputer.

Tujuan Segmentasi Citra

- Memisahkan objek atau struktur penting dalam citra dari latar belakang atau noise.
- Mengidentifikasi dan mengisolasi fitur atau area tertentu dalam citra yang relevan dengan aplikasi yang diinginkan.
- Mempermudah analisis citra dengan mengurangi kompleksitas.

Metode Segmentasi Citra

Terdapat berbagai metode untuk melakukan segmentasi citra, yang masing-masing memiliki kekuatan dan kelemahan tergantung pada jenis citra dan tujuan analisis.

10.1 Segmentasi Berdasarkan Ambang Batas

Segmentasi berdasarkan ambang batas (thresholding) adalah salah satu teknik dasar dalam pengolahan citra untuk memisahkan objek dari latar belakang atau mengelompokkan piksel-piksel citra berdasarkan nilai intensitasnya. Dalam teknik ini, citra akan diubah menjadi citra biner (hitam-putih) dengan cara menetapkan ambang batas tertentu. Piksel yang memiliki nilai intensitas lebih besar atau sama dengan ambang batas akan dianggap sebagai objek (biasanya diwakili oleh warna putih), sedangkan piksel dengan nilai intensitas lebih kecil akan dianggap sebagai latar belakang (diwakili oleh warna hitam).

Proses Umum Segmentasi Berdasarkan Ambang Batas

1. **Pemilihan Ambang Batas**: Tentukan nilai ambang batas, yang biasanya berupa angka antara 0 hingga 255 (untuk citra grayscale 8-bit).
    - Jika piksel memiliki nilai intensitas ≥ ambang batas, maka piksel tersebut akan menjadi objek (biasanya berwarna putih).
    - Jika piksel memiliki nilai intensitas < ambang batas, maka piksel tersebut akan menjadi latar belakang (biasanya berwarna hitam).
2. **Aplikasi Ambang Batas**: Setelah ambang batas ditentukan, ambang tersebut diterapkan ke setiap piksel dalam citra untuk menghasilkan citra biner.

Langkah-langkah Segmentasi Berdasarkan Ambang Batas

1. Membaca citra grayscale.
2. Menentukan ambang batas (threshold).
3. Menerapkan ambang batas untuk menghasilkan citra biner.
4. Menampilkan hasil segmentasi.

Berikut adalah contoh kode MATLAB untuk segmentasi berdasarkan ambang batas pada citra grayscale:

```
% Langkah 1: Membaca citra
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
grayCitra = rgb2gray(citra); % Mengubah citra ke grayscale jika citra berwarna
% Langkah 2: Menentukan ambang batas
threshold = 128; % Misalnya, ambang batas adalah 128 (Anda dapat menyesuaikan ini)
% Langkah 3: Menerapkan ambang batas
binerCitra = grayCitra >= threshold; % Piksel > threshold = 1 (putih), piksel < threshold = 0 (hitam)

% Langkah 4: Menampilkan citra asli dan citra hasil segmentasi
subplot(1,2,1); % Gambar pertama
```

```
imshow(grayCitra);
title('Citra Grayscale');
subplot(1,2,2); % Gambar kedua
imshow(binerCitra);
title('Citra Hasil Segmentasi');
```
Hasil:

Gambar 10.1: Citra Grayscale dan Citra hasil Segemnetasi

Penjelasan Kode:

- **imread('citra.jpg')**: Membaca citra dari file yang diberikan.
- **rgb2gray(citra)**: Mengubah citra RGB menjadi citra grayscale (jika citra yang dimuat adalah citra berwarna).
- **threshold = 128**: Menentukan nilai ambang batas (dalam contoh ini, 128). Anda bisa menggantinya dengan nilai lain sesuai kebutuhan.
- **binerCitra = grayCitra >= threshold**: Proses segmentasi dengan membandingkan setiap piksel citra grayscale dengan nilai ambang batas. Piksel yang lebih besar atau sama dengan ambang batas akan menjadi 1 (putih), dan sisanya menjadi 0 (hitam).
- **imshow()**: Menampilkan citra pada jendela gambar.

Pemilihan Ambang Batas

Pemilihan ambang batas sangat penting dalam segmentasi citra. Ambang batas yang tepat akan menghasilkan pemisahan yang jelas antara objek dan latar belakang. Beberapa teknik untuk memilih ambang batas meliputi:

- **Ambang Batas Global**: Menentukan satu nilai ambang untuk seluruh citra, seperti yang ditunjukkan dalam contoh di atas.
- **Ambang Batas Adaptif**: Menentukan ambang batas berdasarkan statistik lokal citra, seperti rata-rata lokal atau median.

Jika diperlukan, Anda juga bisa menggunakan teknik seperti **Otsu's method** untuk menentukan ambang batas secara otomatis.

Contoh Peningkatan:

- Untuk memilih nilai ambang secara otomatis, MATLAB menyediakan fungsi **graythresh()** yang menggunakan metode Otsu untuk mencari ambang batas terbaik.

```
threshold = graythresh(grayCitra); % Otsu thresholding
binerCitra = imbinarize(grayCitra, threshold);
imshow(binerCitra);
```

10.2 Segmentasi Berdasarkan Pemetaan Warna

Segmentasi berdasarkan pemetaan warna adalah teknik segmentasi citra yang memanfaatkan informasi warna (RGB atau ruang warna lain seperti HSV atau Lab) untuk membagi citra menjadi beberapa bagian yang lebih homogen, berdasarkan perbedaan warna. Teknik ini sangat berguna terutama pada citra yang memiliki objek dengan warna yang berbeda jelas dari latar belakangnya. Dengan memanfaatkan pemetaan warna, kita dapat mengidentifikasi objek yang memiliki warna tertentu dan memisahkannya dari bagian citra lainnya.

Pada dasarnya, segmentasi berdasarkan pemetaan warna bekerja dengan cara:

1. **Mengekstrak komponen warna tertentu** dari citra (misalnya, komponen R, G, B dalam ruang warna RGB, atau H, S, V dalam ruang warna HSV).

2. **Menerapkan ambang batas** pada komponen warna yang relevan untuk memisahkan objek berdasarkan perbedaan warna.
3. **Menghasilkan citra biner atau citra dengan label** untuk menunjukkan bagian-bagian yang telah tersegmentasi.

Proses Umum Segmentasi Berdasarkan Pemetaan Warna

1. **Konversi Citra ke Ruang Warna yang Tepat**: Citra RGB sering kali dikonversi ke ruang warna lain seperti HSV (Hue, Saturation, Value) atau Lab karena ruang warna ini lebih memisahkan komponen warna dan lebih mudah untuk melakukan pemisahan berdasarkan warna.
2. **Ekstraksi Kanal Warna**: Dalam ruang warna seperti RGB atau HSV, komponen warna tertentu diekstraksi. Misalnya, pada ruang warna RGB, kita bisa mengambil saluran merah (R), hijau (G), atau biru (B) untuk analisis.
3. **Penerapan Ambang Batas pada Komponen Warna**: Untuk memisahkan objek, kita menetapkan ambang batas tertentu pada salah satu atau lebih komponen warna.
4. **Membuat Citra Biner**: Setelah menerapkan ambang batas, citra akan menjadi biner (hitam-putih), dengan objek yang diinginkan diwakili oleh piksel putih dan latar belakang oleh piksel hitam.
5. **Pemrosesan Lanjutan (Opsional)**: Setelah segmentasi, kita bisa melakukan operasi tambahan seperti penghapusan noise, penghalusan objek, atau label komponen yang terpisah.

Berikut adalah contoh kode MATLAB untuk segmentasi citra berdasarkan pemetaan warna menggunakan ruang warna HSV:

Langkah-langkah:

1. Membaca citra RGB.
2. Mengubah citra dari ruang warna RGB ke HSV.
3. Menentukan rentang warna (nilai ambang batas) untuk komponen Hue (H), Saturation (S), dan Value (V).
4. Membuat citra biner berdasarkan rentang nilai warna yang diinginkan.
5. Menampilkan hasil segmentasi.

```
% Langkah 1: Membaca citra
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
% Langkah 2: Mengubah citra RGB ke ruang warna HSV
hsvCitra = rgb2hsv(citra); % Konversi citra ke HSV
% Langkah 3: Menentukan rentang nilai untuk komponen H, S, dan V
% Misalnya, kita ingin mengekstrak objek dengan warna merah
H_min = 0;    H_max = 0.1;  % Rentang Hue untuk warna merah (0 - 0.1)
S_min = 0.5;  S_max = 1;    % Rentang Saturasi yang cukup tinggi untuk warna yang cerah
V_min = 0.3;  V_max = 1;    % Rentang Value untuk objek yang cukup terang
% Ekstrak komponen H, S, dan V dari citra HSV
H = hsvCitra(:,:,1); % Kanal Hue
S = hsvCitra(:,:,2); % Kanal Saturation
V = hsvCitra(:,:,3); % Kanal Value
% Langkah 4: Menerapkan ambang batas untuk setiap kanal
mask = (H >= H_min & H <= H_max) & ...
    (S >= S_min & S <= S_max) & ...
    (V >= V_min & V <= V_max);
% Langkah 5: Menampilkan hasil segmentasi
subplot(1,2,1);
imshow(citra);
title('Citra Asli');
subplot(1,2,2);
imshow(mask);
title('Citra Hasil Segmentasi Berdasarkan Warna');
```

Hasil:

Gambar 10.2: Citra Warna dan Citra hasil Segemnetasi

Penjelasan Kode:

1. **rgb2hsv(citra)**: Fungsi ini mengubah citra RGB menjadi citra dengan ruang warna HSV (Hue, Saturation, Value).
2. **Ekstraksi Kanal H, S, dan V**: Kita mengekstrak masing-masing komponen warna (H, S, dan V) dari citra HSV untuk memisahkan dan menganalisis warna.
3. **Rentang Ambang Batas**:
    - H_min dan H_max adalah batasan untuk komponen Hue. Misalnya, untuk warna merah, kita bisa memilih rentang sekitar 0 hingga 0.1.
    - S_min dan S_max adalah rentang untuk Saturasi. Biasanya, kita menggunakan nilai tinggi untuk memastikan objek memiliki warna yang jelas.
    - V_min dan V_max adalah rentang untuk Value, yang mengontrol seberapa terang atau gelap objek yang tersegmentasi.
4. **Masking**: Proses ini menghasilkan citra biner (mask) berdasarkan ambang batas yang diterapkan pada setiap kanal warna.
5. **imshow()**: Menampilkan citra asli dan citra hasil segmentasi dalam dua subplot yang terpisah.

Pemilihan Rentang Warna

Pemilihan rentang warna (H, S, dan V) sangat bergantung pada warna objek yang ingin Anda segmentasikan. Misalnya, jika Anda ingin mengekstrak objek merah, Anda harus mengetahui rentang nilai Hue untuk warna merah, dan Saturation serta Value yang sesuai untuk objek yang ingin dipisahkan. Hal ini bisa dilakukan secara eksperimental dengan mengamati histogram atau dengan menggunakan alat seperti impixelinfo di MATLAB untuk melihat nilai pixel pada citra.

Keuntungan dan Tantangan

Keuntungan:

- Sangat efektif untuk segmentasi objek dengan warna yang berbeda jelas dari latar belakang.
- Dapat menangani variasi pencahayaan yang kecil, tergantung pada ruang warna yang digunakan (misalnya, HSV lebih tahan terhadap perubahan pencahayaan daripada RGB).

Tantangan:

- Rentang warna yang digunakan harus dipilih dengan hati-hati, terutama jika ada banyak objek dengan warna serupa atau latar belakang yang memiliki warna mirip dengan objek.
- Segmentasi bisa terpengaruh oleh variasi pencahayaan yang besar, meskipun ruang warna seperti HSV lebih tahan terhadap perubahan pencahayaan.

10.3 Segmentasi Berdasarkan Region

Segmentasi berdasarkan region adalah teknik segmentasi citra yang mengelompokkan piksel-piksel citra menjadi beberapa wilayah atau region berdasarkan kesamaan dalam beberapa atribut, seperti intensitas warna, tekstur, atau fitur lainnya. Teknik ini berfokus pada pengenalan wilayah-wilayah yang memiliki kesamaan dalam citra, yang disebut sebagai *region*, dan memisahkannya dari wilayah lain yang berbeda.

Metode segmentasi berdasarkan region dapat dibagi menjadi dua pendekatan utama:

Region Growing: Dimulai dengan piksel atau region seed (titik awal), dan kemudian piksel-piksel yang terhubung dengan piksel seed yang memiliki karakteristik serupa (misalnya, intensitas atau warna yang serupa) ditambahkan ke region tersebut.

Region Splitting and Merging: Dimulai dengan membagi citra menjadi beberapa wilayah yang lebih kecil, dan kemudian wilayah-wilayah yang serupa digabungkan, atau jika sebuah wilayah terlalu homogen, ia dapat dibagi lebih lanjut.
Pendekatan Region Growing lebih sering digunakan dan akan lebih banyak dibahas dalam penjelasan ini.
Proses Umum Segmentasi Berdasarkan Region (Region Growing)
Pemilihan Seed Point: Pilih piksel atau region awal yang akan digunakan untuk memulai proses segmentasi. Biasanya, seed point dipilih secara manual atau berdasarkan kriteria tertentu.
Pertumbuhan Region: Proses ini melibatkan penambahan piksel yang terhubung dengan seed point yang memiliki karakteristik serupa (misalnya, intensitas piksel dalam citra grayscale atau nilai warna dalam citra berwarna).
Pembatasan Threshold: Setiap piksel yang bergabung dengan region harus memenuhi kriteria tertentu, seperti selisih intensitas atau warna yang tidak melebihi nilai ambang batas (threshold).
Pengulangan: Proses ini diulang hingga seluruh region terbentuk, atau tidak ada piksel yang dapat ditambahkan lagi ke region.
Langkah-langkah Region Growing
1. Tentukan titik seed (titik awal).
2. Tentukan kriteria untuk pertumbuhan region (misalnya, perbedaan intensitas piksel atau perbedaan warna).
3. Terapkan kriteria tersebut untuk menambahkan piksel ke region yang ada.
4. Ulangi proses ini sampai tidak ada piksel lagi yang bisa ditambahkan ke region.

Di bawah ini adalah contoh implementasi teknik segmentasi region growing pada citra grayscale menggunakan MATLAB. Citra akan dibagi menjadi beberapa region berdasarkan kesamaan intensitas piksel.
Langkah-langkah Implementasi:
1. Membaca citra grayscale.
2. Menentukan titik seed.
3. Menentukan kriteria (perbedaan intensitas) untuk pertumbuhan region.
4. Menerapkan region growing untuk menemukan region yang terhubung.
5. Menampilkan hasil segmentasi.

Buat fungsi regiongrowing:

```
function region = regionGrowing(citra, seed, threshold)
    [m, n] = size(citra);
    region = false(m, n);  % Citra hasil segmentasi (bernilai true untuk region yang ditemukan)
    visited = false(m, n); % Matriks untuk menandai piksel yang sudah dikunjungi
    % Inisialisasi dengan piksel seed
    region(seed(1), seed(2)) = true;
    visited(seed(1), seed(2)) = true;
    pixelValue = citra(seed(1), seed(2));  % Nilai intensitas piksel seed
    % Antrian untuk pencarian piksel yang terhubung
    queue = seed;
    while ~isempty(queue)
        % Ambil piksel dari antrian
        currentPixel = queue(1, :);
        queue(1, :) = []; % Menghapus piksel dari antrian
        x = currentPixel(1);
        y = currentPixel(2);
        % Mengecek tetangga (8 arah)
        for dx = -1:1
            for dy = -1:1
                nx = x + dx;
                ny = y + dy;
                % Pastikan koordinat berada dalam batas citra
                if nx >= 1 && nx <= m && ny >= 1 && ny <= n
                    if ~visited(nx, ny)
                        % Cek apakah selisih intensitas <= threshold
                        if abs(int32(citra(nx, ny)) - int32(pixelValue)) <= threshold
                            region(nx, ny) = true; % Tandai sebagai bagian dari region
                            visited(nx, ny) = true;
```

```
                    queue = [queue; nx, ny]; % Tambahkan ke antrian untuk dicek
                end
            end
        end
    end
  end
end
```

kemudia tulis program dibawah ini:

```
% Langkah 1: Membaca citra grayscale
citra = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
grayCitra = rgb2gray(citra);  % Mengubah citra menjadi grayscale jika citra berwarna
% Langkah 2: Menentukan titik seed (misalnya, piksel di tengah citra)
seed = [100, 100];  % Koordinat seed [x, y]
threshold = 20;     % Ambang batas intensitas untuk pertumbuhan region
% Membuat citra label untuk menandai wilayah
[m, n] = size(grayCitra);
labelCitra = zeros(m, n);  % Inisialisasi citra label dengan nilai 0 (belum terlabel)
% Langkah 3: Fungsi Region Growing
% Fungsi ini akan menambahkan piksel yang terhubung dengan seed ke region
region = regionGrowing(grayCitra, seed, threshold);
% Langkah 4: Menampilkan citra asli dan hasil segmentasi
subplot(1,2,1);
imshow(grayCitra);
title('Citra Asli');
subplot(1,2,2);
imshow(region);
title('Citra Hasil Segmentasi Region Growing');
```

Hasil:

Gambar 10.3: Citra Grayscale dan Citra hasil Segemnetasi

Penjelasan Kode:

1. **Baca dan Konversi Citra**: Citra dibaca dengan imread() dan jika citra berwarna, diubah menjadi grayscale menggunakan rgb2gray().
2. **Titik Seed**: Titik seed ditentukan secara manual dalam kode. Dalam hal ini, seed berada di posisi (100, 100). Anda bisa memilih titik lain sesuai kebutuhan.
3. **Fungsi regionGrowing**: Fungsi ini melakukan proses region growing dengan menggunakan antrian untuk mencari piksel yang terhubung dengan seed. Fungsi ini mengecek tetangga dari piksel saat ini (dalam 8 arah) dan menambahkannya ke region jika perbedaan intensitasnya tidak lebih besar dari ambang batas yang ditentukan.
4. **Threshold**: Ambang batas (threshold) ditentukan sebagai perbedaan intensitas yang diizinkan antara piksel yang terhubung. Dalam kode ini, nilai threshold diatur ke 20.

5. **Menampilkan Hasil**: Dua citra ditampilkan: citra asli dan citra hasil segmentasi region growing.

Pemahaman Lebih Lanjut:

- **Kriteria Pertumbuhan Region**: Kriteria pertumbuhan region pada implementasi ini berdasarkan selisih intensitas antara piksel yang terhubung dengan seed. Anda bisa menyesuaikan kriteria ini dengan fitur lain seperti warna (untuk citra berwarna) atau tekstur.
- **Pencarian Tetangga**: Untuk setiap piksel dalam region, kita memeriksa tetangga dalam 8 arah (atas, bawah, kiri, kanan, dan 4 diagonal). Piksel yang memenuhi kriteria ditambahkan ke region.

Keuntungan dan Tantangan

Keuntungan:

- Dapat mendeteksi objek yang terhubung berdasarkan kesamaan intensitas atau warna.
- Cocok untuk citra dengan objek yang memiliki kontras yang jelas dengan latar belakang.

Tantangan:

- Memilih titik seed yang tepat sangat penting. Jika titik seed berada di tempat yang salah, hasil segmentasi bisa tidak akurat.
- Region growing bisa menjadi sangat lambat jika citra sangat besar atau jika ada banyak piksel yang perlu diproses.
- Sensitif terhadap ambang batas; ambang batas yang salah bisa menyebabkan region yang tidak sesuai atau terlalu banyak noise.

10.4 Segmentasi Watershed

Segmentasi Watershed adalah teknik yang digunakan dalam pengolahan citra untuk memisahkan objek atau wilayah dalam gambar berdasarkan perbedaan intensitas piksel. Teknik ini sering digunakan untuk memisahkan wilayah yang saling tumpang tindih, seperti objek dalam citra medis atau citra mikroskopik.

Metode Watershed bekerja dengan prinsip analogi geografi. Bayangkan sebuah peta topografi di mana setiap piksel memiliki ketinggian tertentu, dan daerah dengan ketinggian yang lebih tinggi akan "mengalirkan air" ke daerah yang lebih rendah. Teknik ini memetakan citra sebagai peta topografi, dan "air" mengalir dari daerah yang lebih tinggi menuju daerah yang lebih rendah, sehingga membentuk batas yang disebut "segmen" atau "wilayah".

Langkah-langkah Umum Segmentasi Watershed:

1. Praproses Citra:
    - Biasanya, citra yang diolah dengan teknik Watershed adalah citra grayscale.
    - Peningkatan kontras atau pemfilteran sering diterapkan untuk memperjelas batas objek dan mengurangi noise.
2. Membentuk Peta Topografi:
    - Citra diperlakukan sebagai peta topografi, di mana intensitas piksel diubah menjadi ketinggian. Piksel dengan nilai intensitas yang rendah akan berada di "lembah", sedangkan piksel dengan nilai tinggi akan berada di "puncak".
3. Flooding Proses:
    - Proses ini dimulai dengan "memenuhi" lembah dengan air dan kemudian mencari batas-batas alami yang terbentuk antara wilayah dengan perbedaan intensitas yang signifikan.
4. Penentuan Batas (Segmentation):
    - Titik batas (watershed lines) ditemukan di tempat yang memiliki transisi tajam antara dua wilayah, yang kemudian digunakan untuk memisahkan objek-objek yang berbeda dalam citra.

MATLAB menyediakan fungsi yang sangat baik untuk segmentasi Watershed melalui toolbox seperti *Image Processing Toolbox*. Berikut adalah contoh kode MATLAB untuk melakukan segmentasi Watershed:

```
% Membaca citra input
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
Igray = rgb2gray(I);  % Mengubah citra menjadi grayscale jika gambar berwarna
% Meningkatkan kontras citra untuk mempermudah segmentasi
Ienhanced = imadjust(Igray);
% Mengkonversi citra menjadi tipe double sebelum menerapkan filter
Ienhanced_double = double(Ienhanced);
% Membuat filter Gaussian menggunakan fspecial
h = fspecial('gaussian', [5 5], 2);  % [5 5] adalah ukuran filter, 2 adalah standar deviasi (sigma)
% Menerapkan filter Gaussian menggunakan imfilter
Ifiltered = imfilter(Ienhanced_double, h, 'same');  % 'same' agar hasil filter memiliki ukuran yang sama dengan
```

```
input
% Menampilkan kedua gambar dalam satu figure
figure;
% Menampilkan citra asli (gambar 1) di subplot pertama
subplot(1, 2, 1);  % 1 baris, 2 kolom, gambar pertama
imshow(Igray);
title('Citra Grayscale');
% Menampilkan citra hasil filter Gaussian (gambar 2) di subplot kedua
subplot(1, 2, 2);  % 1 baris, 2 kolom, gambar kedua
imshow(uint8(Ifiltered));  % Mengkonversi kembali ke uint8 untuk ditampilkan
title('Citra yang Telah Difilter');
```

Hasil:

Gambar 10.4: Citra Grayscale dan Citra hasil Segemnetasi

Penjelasan Kode:

1. Mengubah citra menjadi grayscale: Jika citra input berwarna, kita konversi menjadi grayscale menggunakan rgb2gray().
2. Meningkatkan kontras: Fungsi imadjust() digunakan untuk meningkatkan kontras citra, yang membantu memperjelas perbedaan intensitas.
3. Gaussian Filter: Citra difilter menggunakan imgaussfilt() untuk mengurangi noise yang bisa mengganggu segmentasi.
4. Membuat peta gradien: Menggunakan gradient() untuk menghitung gradien citra. Gradien ini membantu mengidentifikasi tepi atau batas objek.
5. Watershed transformasi: Fungsi watershed() digunakan untuk melakukan segmentasi berdasarkan gradien yang telah dihitung.
6. Mengubah label menjadi RGB: Fungsi label2rgb() digunakan untuk memberi warna pada setiap label yang dihasilkan oleh proses Watershed.

Soal-soal Latihan:

1. Jelaskan apa yang dimaksud dengan segmentasi citra dalam pengolahan citra. Mengapa segmentasi sangat penting dalam analisis citra, dan sebutkan beberapa aplikasi nyata yang menggunakan segmentasi citra.
2. Salah satu metode segmentasi citra yang umum digunakan adalah thresholding. Jelaskan bagaimana metode thresholding bekerja untuk membagi citra menjadi dua wilayah (objek dan latar belakang). Berikan contoh penerapan thresholding pada citra grayscale.
3. Apa yang dimaksud dengan segmentasi berbasis region growing? Jelaskan bagaimana proses region growing dilakukan untuk mengidentifikasi wilayah yang memiliki karakteristik yang sama dalam citra.
4. Jelaskan perbedaan antara segmentasi citra berbasis klastering (misalnya k-means) dan segmentasi berbasis thresholding. Kapan masing-masing metode lebih efektif digunakan dalam pengolahan citra?

5. Dalam MATLAB, bagaimana cara melakukan segmentasi citra menggunakan metode thresholding? Jelaskan langkah-langkah yang diperlukan, termasuk cara memilih nilai threshold yang tepat. Sertakan contoh kode MATLAB untuk melakukan segmentasi citra grayscale menggunakan thresholding.

Bab 11
Ekstraksi Citra

Ekstraksi ciri (feature extraction) adalah proses untuk mengekstrak informasi yang relevan dari citra agar dapat digunakan dalam analisis lebih lanjut, seperti pengenalan objek, klasifikasi, atau deteksi. Ciri atau fitur adalah representasi dari karakteristik objek dalam citra yang dapat membedakan objek satu dengan yang lainnya. Tujuan utama dari ekstraksi ciri adalah untuk merangkum informasi yang ada dalam citra menjadi bentuk yang lebih ringkas dan lebih mudah dikelola untuk analisis lanjutan (Windana et al., 2014).
Ekstraksi ciri dapat diterapkan pada berbagai jenis citra, baik citra biner, grayscale, atau citra warna, dan digunakan dalam berbagai aplikasi seperti pengenalan wajah, pengenalan teks, analisis medis, dan banyak lagi.

11.1 Ekstrasi Ciri Berdasarkan Intensitas

Ekstraksi ciri berdasarkan intensitas pada citra adalah teknik yang digunakan untuk mengidentifikasi fitur atau pola dalam citra berdasarkan nilai intensitas piksel di dalam citra tersebut. Teknik ini sangat penting dalam berbagai aplikasi pemrosesan citra dan pengenalan pola, seperti pengenalan objek, segmentasi citra, dan analisis tekstur.
Berikut adalah penjelasan mengenai ekstraksi ciri berdasarkan intensitas pada citra dan bagaimana melakukannya menggunakan MATLAB:

1. Pengenalan Ciri Berdasarkan Intensitas

Ekstraksi ciri berdasarkan intensitas melibatkan identifikasi informasi atau fitur citra yang bergantung pada variasi intensitas piksel. Ini bisa melibatkan beberapa metode, antara lain:

- **Histogram:** Menyajikan distribusi intensitas piksel dalam citra.
- **Mean dan Variance:** Rata-rata dan variasi intensitas piksel pada wilayah citra.
- **Edge Detection:** Mengidentifikasi perbedaan intensitas yang tajam di antara piksel yang berdekatan.
- **Thresholding:** Menetapkan ambang batas untuk membedakan objek dengan latar belakang berdasarkan intensitas.

2. Langkah-langkah Ekstraksi Ciri Berdasarkan Intensitas

Beberapa langkah umum dalam ekstraksi ciri berdasarkan intensitas antara lain:

- **Konversi Citra ke Skala Abu-abu (Grayscale):** Jika citra berwarna, biasanya citra akan dikonversi menjadi skala abu-abu terlebih dahulu untuk mempermudah analisis intensitas.
- **Penghitungan Histogram Intensitas:** Menghitung dan menganalisis distribusi intensitas dalam citra.
- **Deteksi Tepi:** Menggunakan algoritma deteksi tepi seperti Sobel, Canny, atau Prewitt untuk menemukan perubahan intensitas yang signifikan.
- **Pengukuran Statistik Intensitas:** Menghitung rata-rata, varians, dan fitur statistik lain yang berhubungan dengan distribusi intensitas piksel.

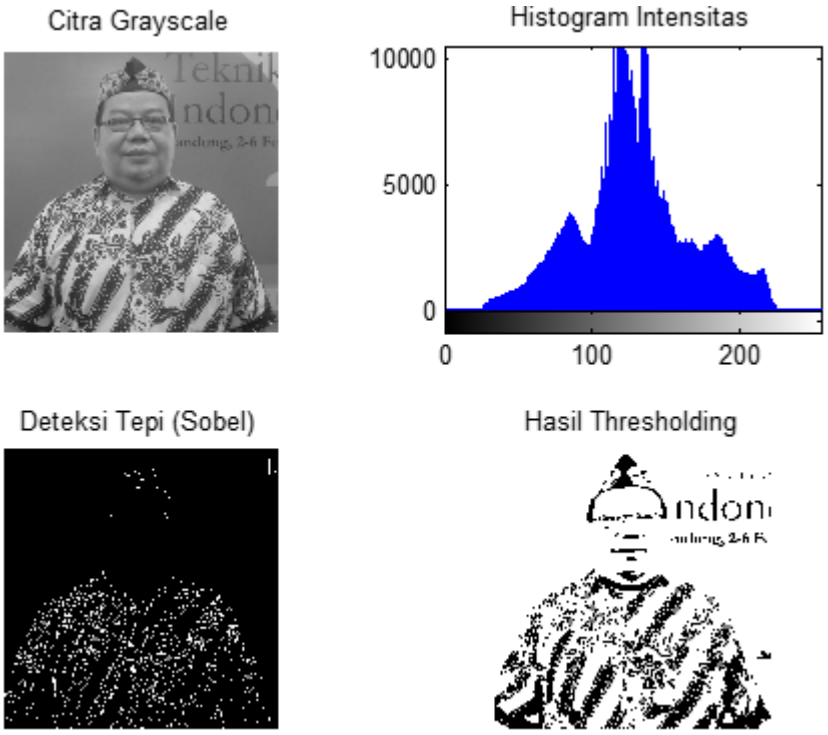
3. Pemrograman Ekstraksi Ciri Berdasarkan Intensitas dengan MATLAB

Di bawah ini adalah contoh pemrograman MATLAB untuk ekstraksi ciri berdasarkan intensitas citra:

```
% Membaca citra
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
% Mengkonversi citra ke grayscale jika citra berwarna
if size(img, 3) == 3
   img_gray = rgb2gray(img);
else
   img_gray = img;
end
% Membuat figure untuk menampilkan semua gambar
figure;
% Menampilkan citra grayscale
subplot(2, 2, 1);
imshow(img_gray);
title('Citra Grayscale');
```

```
% Menampilkan histogram intensitas
subplot(2, 2, 2);
imhist(img_gray);
title('Histogram Intensitas');
% Menghitung statistik intensitas: rata-rata dan varians
mean_intensity = mean(img_gray(:));
var_intensity = var(double(img_gray(:)));
% Menampilkan hasil statistik
disp(['Rata-rata Intensitas: ', num2str(mean_intensity)]);
disp(['Varians Intensitas: ', num2str(var_intensity)]);
% Deteksi tepi menggunakan operator Sobel
edges = edge(img_gray, 'Sobel');
% Menampilkan hasil deteksi tepi
subplot(2, 2, 3);
imshow(edges);
title('Deteksi Tepi (Sobel)');
% Thresholding: Misalnya ambang batas 100
thresholded_img = img_gray > 100;
% Menampilkan hasil thresholding
subplot(2, 2, 4);
imshow(thresholded_img);
title('Hasil Thresholding');
```

Hasil:



Gambar 11.1: Ekstrasi ciri menggunakan intensitas

4. Penjelasan Kode di atas:

- **Citra Grayscale:** Pertama, citra dibaca dan dikonversi ke grayscale jika citra berwarna. Ini dilakukan menggunakan fungsi rgb2gray.
- **Histogram:** Fungsi imhist digunakan untuk menghitung dan menampilkan histogram intensitas citra.
- **Statistik Intensitas:** Menghitung rata-rata dan varians intensitas piksel citra dengan menggunakan fungsi mean dan var.
- **Deteksi Tepi:** Menggunakan operator Sobel (fungsi edge) untuk mendeteksi tepi dalam citra berdasarkan perubahan intensitas yang tajam.
- **Thresholding:** Menggunakan threshold untuk memisahkan objek dan latar belakang berdasarkan nilai intensitas piksel tertentu.

5. Aplikasi Ekstraksi Ciri Berdasarkan Intensitas:

- **Pengenalan Objek:** Ekstraksi ciri intensitas digunakan untuk mendeteksi objek dalam citra dengan memanfaatkan perbedaan intensitas objek dan latar belakang.
- **Segmentasi Citra:** Teknik thresholding dapat digunakan untuk segmentasi citra, misalnya untuk memisahkan objek dari latar belakang.
- **Analisis Tekstur:** Analisis statistik seperti rata-rata dan varians intensitas dapat digunakan untuk mengidentifikasi pola tekstur dalam citra.

11.2 Ekstrasi Ciri Berdasarkan Bentuk

Ekstraksi ciri berdasarkan bentuk pada citra adalah teknik yang digunakan untuk mengekstrak informasi penting yang berkaitan dengan bentuk objek dalam citra digital. Informasi bentuk ini bisa meliputi parameter seperti ukuran, kelengkungan, sudut, dan lainnya. Tujuan utama dari ekstraksi ciri ini adalah untuk menganalisis dan mendeskripsikan objek dalam citra agar dapat digunakan dalam aplikasi seperti pengenalan pola, pengklasifikasian objek, atau penghitungan berbagai karakteristik geometris objek.

Berikut adalah beberapa metode umum dalam ekstraksi ciri berdasarkan bentuk pada citra:

1. Deteksi Kontur

Kontur adalah garis yang menghubungkan titik-titik dengan intensitas yang sama dalam citra. Dengan mendeteksi kontur, kita bisa mendapatkan informasi tentang bentuk objek dalam citra. Misalnya, kita bisa menggunakan algoritma deteksi tepi seperti Canny atau Sobel untuk mendeteksi kontur objek.

2. Area dan Perimeter

Beberapa ciri yang sering diekstraksi dalam analisis bentuk adalah **area** dan **perimeter** objek. Area adalah jumlah piksel yang membentuk objek, sedangkan perimeter adalah panjang garis yang mengelilingi objek.

3. Rasio Aspek

Rasio aspek menggambarkan perbandingan antara panjang dan lebar objek. Ini dapat memberikan informasi tentang apakah objek tersebut lebih memanjang secara horizontal atau vertikal.

4. Momen Invarian

Momen invarian adalah representasi numerik dari objek dalam citra yang tidak terpengaruh oleh rotasi, translasi, atau skala. Dengan momen invarian, kita dapat mengidentifikasi bentuk objek meskipun objek tersebut diputar atau diubah ukurannya.

5. Convex Hull

Convex hull adalah poligon terkecil yang dapat melingkupi seluruh objek. Ekstraksi convex hull digunakan untuk menganalisis bentuk geometris objek dan memberikan informasi lebih lanjut tentang batas objek.

Di bawah ini adalah contoh sederhana tentang bagaimana ekstraksi ciri berdasarkan bentuk dapat dilakukan menggunakan MATLAB.

1. Preprocessing Citra

Langkah pertama dalam ekstraksi ciri adalah mengubah citra menjadi citra biner. Ini dilakukan dengan mengonversi citra grayscale menjadi citra biner menggunakan thresholding.

```
% Membaca citra
img = imread('image.png');
% Mengonversi citra ke grayscale
gray_img = rgb2gray(img);
% Mengubah citra ke citra biner
bw_img = imbinarize(gray_img);
```

2. Deteksi Kontur

Setelah citra biner, kita bisa menggunakan fungsi bwboundaries untuk mendeteksi kontur dari objek yang ada dalam citra.

```
% Mendeteksi kontur objek
boundaries = bwboundaries(bw_img);
```

3. Menghitung Area dan Perimeter

Untuk menghitung area dan perimeter, kita dapat menggunakan fungsi regionprops, yang juga menyediakan berbagai ciri lainnya seperti centroid dan bounding box.

```
% Menghitung properti geometris dari objek
stats = regionprops(bw_img, 'Area', 'Perimeter');
for i = 1:length(stats)
    area = stats(i).Area;
    perimeter = stats(i).Perimeter;
```

```
    fprintf('Area: %f, Perimeter: %f\n', area, perimeter);
end
```

4. Momen Invarian

Untuk ekstraksi momen invarian, kita bisa menggunakan fungsi regionprops yang menyediakan momen-zernike.

```
% Menghitung momen invarian
stats = regionprops(bw_img, 'MomentInvariant');
for i = 1:length(stats)
    moments = stats(i).MomentInvariant;
    fprintf('Momen Invarian: %s\n', num2str(moments));
end
```

5. Convex Hull

Untuk menemukan convex hull, kita bisa menggunakan fungsi bwconvhull.

```
% Menghitung convex hull
convex_hull = bwconvhull(bw_img);
% Menampilkan citra convex hull
imshow(convex_hull);
```

kode lenkapnaya:

```
% Membaca citra
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
figure;
imshow(img);  % Menampilkan citra asli
title('Citra Asli');
% Membaca citra

% Mengonversi citra ke grayscale
gray_img = rgb2gray(img);

% Thresholding manual untuk mengubah citra ke citra biner
threshold_value = 100;  % Tentukan nilai threshold (Anda bisa menyesuaikan)
bw_img = gray_img > threshold_value;

% Menampilkan citra biner
figure;
imshow(bw_img);
title('Citra Biner');

% Memastikan ada objek dalam citra biner
if sum(bw_img(:)) == 0
    disp('Tidak ada objek terdeteksi dalam citra biner.');
else
    disp('Objek terdeteksi dalam citra biner.');
end

% Menampilkan convex hull dengan kontur yang lebih jelas
% Menghitung convex hull (poligon terkecil yang melingkupi objek)
convex_hull = bwconvhull(bw_img);

% Memeriksa apakah convex hull berhasil dihitung
if sum(convex_hull(:)) == 0
    disp('Tidak ada convex hull yang dapat dihitung.');
else
    % Menampilkan citra asli dengan convex hull
    figure;
```

```
    imshow(img);  % Menampilkan citra asli
    hold on;
    % Menggambar convex hull di atas citra asli dengan warna yang lebih jelas
    boundary = bwboundaries(convex_hull);
    for k = 1:length(boundary)
        plot(boundary{k}(:,2), boundary{k}(:,1), 'r', 'LineWidth', 3);  % Menampilkan convex hull dengan warna merah yang lebih tebal
    end
    title('Convex Hull Objek');
end
```

Hasil:

Gambar 11.2: Ekstrasi ciri menggunakan Convex Hull Objek

11.3 Ekstrasi Ciri Berdasarkan Tepi

Ekstraksi ciri berdasarkan tepi (edge feature extraction) adalah teknik dalam pengolahan citra untuk mendeteksi perubahan intensitas yang tajam pada citra. Tepi pada citra biasanya menunjukkan batas-batas objek atau transisi antara area yang berbeda dalam citra, sehingga deteksi tepi sering digunakan untuk pengenalan objek, pemrosesan citra medis, pemetaan, dan aplikasi visi komputer lainnya.

Konsep Dasar Ekstraksi Ciri Berdasarkan Tepi

Tepi dalam citra sering diartikan sebagai perubahan signifikan dalam intensitas pixel. Proses ekstraksi ciri berdasarkan tepi bertujuan untuk menemukan lokasi-lokasi di mana terjadi perubahan besar dalam intensitas, yang bisa menunjukkan adanya batas objek atau perubahan tekstur.

Tepi pada citra biasanya dapat dideteksi menggunakan beberapa teknik, di antaranya:

1. **Deteksi Sobel**: Teknik ini menggunakan operator Sobel untuk menghitung gradien intensitas citra dalam arah horizontal dan vertikal, sehingga dapat menyoroti tepi-tepi utama.
2. **Deteksi Canny**: Algoritma ini menggunakan beberapa tahapan untuk mendeteksi tepi, termasuk pemrosesan filtrasi untuk mengurangi noise, dan penentuan ambang batas untuk mendeteksi tepi yang kuat.
3. **Prewitt**: Teknik ini mirip dengan Sobel, tetapi menggunakan kernel Prewitt untuk menghitung gradien.
4. **Roberts Cross**: Teknik ini menggunakan operator kecil untuk mendeteksi perubahan intensitas.

Langkah-langkah Umum Ekstraksi Ciri Berdasarkan Tepi

1. **Persiapan Citra**: Citra diubah ke format grayscale jika diperlukan, karena deteksi tepi umumnya dilakukan pada citra grayscale.
2. **Penyaringan (Filtering)**: Biasanya dilakukan penyaringan awal untuk mengurangi noise, misalnya dengan menggunakan filter Gaussian.
3. **Penghitungan Gradien**: Menghitung perubahan intensitas citra di sepanjang sumbu x dan y (misalnya menggunakan operator Sobel atau Prewitt).
4. **Penyaringan Tepi**: Setelah gradien dihitung, lakukan pemilihan tepi dengan menggunakan ambang batas (thresholding) untuk menentukan apakah perubahan intensitas cukup signifikan untuk dianggap sebagai tepi.
5. **Deteksi Tepi**: Tepi dapat ditentukan berdasarkan hasil perhitungan gradien dan ambang batas yang ditetapkan.

Berikut adalah contoh pemrograman untuk ekstraksi ciri berdasarkan tepi menggunakan operator Sobel dan algoritma Canny di MATLAB:

1. Deteksi Tepi dengan Operator Sobel

```
% Membaca citra
```

```
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
% Mengubah citra ke grayscale jika diperlukan
grayI = rgb2gray(I);
% Menerapkan filter Sobel untuk mendeteksi tepi
BW_sobel = edge(grayI, 'sobel');
% Subplot 1: Menampilkan citra asli
subplot(1, 2, 1);  % 1 baris, 3 kolom, gambar pertama
imshow(I);
title('Citra Asli');
% Menampilkan hasil deteksi tepi
subplot(1, 2, 2);  % 1 baris, 3 kolom, gambar kedua
imshow(BW_sobel);
title('Deteksi Tepi dengan Sobel');
```

Hasil:

Gambar 11.3: Ekstrasi ciri menggunakan Sobel

2. Deteksi Tepi dengan Algoritma Canny

```
% Membaca citra
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda

% Mengubah citra ke grayscale jika diperlukan
grayI = rgb2gray(I);

% Menerapkan deteksi tepi dengan Canny
BW_canny = edge(grayI, 'canny');

% Subplot 1: Menampilkan citra asli
subplot(1, 2, 1);  % 1 baris, 3 kolom, gambar pertama
imshow(I);
title('Citra Asli');

% Menampilkan hasil deteksi tepi
subplot(1, 2, 2);  % 1 baris, 3 kolom, gambar kedua
imshow(BW_canny);
title('Deteksi Tepi dengan Canny');
```

Hasil:

Gambar 11.4: Ekstrasi ciri menggunakan Canny

Penjelasan Kode:

1. **imread('gambar.jpg')**: Membaca citra dari file.
2. **rgb2gray(I)**: Mengubah citra berwarna menjadi citra grayscale, karena deteksi tepi biasanya dilakukan pada citra grayscale.
3. **edge(grayI, 'sobel')**: Menggunakan operator Sobel untuk mendeteksi tepi dalam citra grayscale.
4. **edge(grayI, 'canny')**: Menggunakan algoritma Canny untuk mendeteksi tepi dalam citra grayscale.
5. **imshow(BW_sobel/canny)**: Menampilkan citra hasil deteksi tepi.

Penyesuaian Parameter

Pada fungsi edge, terdapat berbagai metode deteksi tepi seperti Sobel, Canny, Prewitt, dan Roberts, dan beberapa di antaranya juga memungkinkan kita untuk mengatur ambang batas (threshold) atau parameter lainnya. Misalnya, pada deteksi Canny, Anda dapat menyesuaikan nilai ambang batas atas dan bawah:

BW_canny = edge(grayI, 'canny', [0.1 0.3]);

Di sini, [0.1 0.3] adalah nilai ambang batas yang mengontrol sensitivitas deteksi tepi.

Aplikasi Ekstraksi Ciri Berdasarkan Tepi

- **Pengenalan objek**: Menyaring objek berdasarkan bentuk tepi untuk membedakan objek satu dengan lainnya.
- **Segmentasi citra**: Memisahkan citra menjadi bagian-bagian berdasarkan perbedaan tepi.
- **Pemetaan dan analisis citra medis**: Mengidentifikasi batas-batas organ atau struktur tubuh pada citra medis.

Dengan menggunakan teknik ekstraksi ciri berbasis tepi ini, kita dapat memperoleh informasi penting mengenai struktur objek dalam citra dan mempermudah proses analisis citra lebih lanjut.

11.4 Ekstrasi Ciri Berdasarkan Warna

Ekstraksi ciri berdasarkan warna adalah salah satu teknik penting dalam pengolahan citra, yang digunakan untuk menganalisis informasi warna dalam citra digital. Pada dasarnya, ekstraksi ciri berdasarkan warna bertujuan untuk mendapatkan informasi tentang distribusi warna dalam citra yang dapat digunakan untuk berbagai aplikasi, seperti pengenalan objek, segmentasi citra, atau pelacakan objek.

Proses ekstraksi ciri berdasarkan warna umumnya melibatkan langkah-langkah berikut:

1. **Pemilihan Ruang Warna**: Ruang warna adalah model yang digunakan untuk merepresentasikan warna dalam citra. Salah satu ruang warna yang sering digunakan dalam pengolahan citra adalah RGB (Red, Green, Blue). Namun, kadang-kadang lebih baik menggunakan ruang warna lain, seperti HSV (Hue, Saturation, Value) atau Lab, karena ruang warna ini lebih sesuai dengan persepsi manusia terhadap warna.
2. **Pemisahan Saluran Warna**: Dalam ruang warna RGB, citra memiliki tiga saluran warna (Red, Green, dan Blue), dan pada ruang HSV terdapat tiga komponen (Hue, Saturation, dan Value). Setiap saluran atau komponen ini menyimpan informasi yang berbeda tentang warna dalam citra, dan kita dapat mengekstrak informasi warna dari saluran-saluran tersebut.
3. **Ekstraksi Ciri Warna**: Setelah pemilahan warna, ciri warna dapat diekstraksi dengan berbagai cara. Beberapa teknik ekstraksi ciri warna yang umum digunakan adalah:

- **Histogram Warna**: Histogram warna adalah representasi distribusi frekuensi warna pada setiap saluran warna. Ini memberikan gambaran tentang bagaimana warna tersebar dalam citra.
- **Rata-rata Warna (Mean Color)**: Ciri warna dapat digambarkan dengan menghitung rata-rata intensitas warna untuk setiap saluran warna.
- **Momen Warna**: Teknik ini melibatkan perhitungan momen statistik dari histogram warna untuk menggambarkan karakteristik warna yang lebih mendalam.

4. **Penggunaan Ciri Warna**: Setelah ciri warna diekstraksi, ciri-ciri ini dapat digunakan untuk aplikasi seperti klasifikasi, segmentasi, pelacakan objek, dan pengenalan pola.

Berikut adalah contoh implementasi ekstraksi ciri berdasarkan warna dengan menggunakan MATLAB:

Langkah 1: Membaca Citra

Citra dibaca menggunakan fungsi imread. Misalnya, kita menggunakan citra berformat JPEG.

```
% Membaca citra
img = imread('image.jpg');
imshow(img);
title('Citra Asli');
```

Langkah 2: Mengubah Ruang Warna (jika diperlukan)

Jika kita ingin bekerja dalam ruang warna HSV, kita dapat mengubah citra dari RGB ke HSV.

```
% Mengubah citra RGB ke ruang warna HSV
hsv_img = rgb2hsv(img);
```

Langkah 3: Ekstraksi Saluran Warna

Setelah citra berada dalam ruang warna yang sesuai, kita dapat memisahkan saluran warna. Misalnya, jika kita menggunakan ruang warna HSV, kita dapat mengekstrak komponen Hue, Saturation, dan Value.

```
% Ekstraksi komponen Hue, Saturation, dan Value
H = hsv_img(:,:,1);  % Hue
S = hsv_img(:,:,2);  % Saturation
V = hsv_img(:,:,3);  % Value
```

Langkah 4: Menghitung Histogram Warna

Histogram warna dapat dihitung untuk setiap saluran warna (RGB atau HSV).

```
% Menghitung histogram untuk saluran Hue
hist_H = imhist(H);
figure;
bar(hist_H);
title('Histogram Hue');
```

Langkah 5: Menghitung Rata-rata Warna

Rata-rata warna dapat dihitung untuk setiap saluran dengan menghitung nilai rata-rata piksel pada saluran tersebut.

```
% Menghitung rata-rata warna untuk saluran Hue, Saturation, dan Value
mean_H = mean(H(:));
mean_S = mean(S(:));
mean_V = mean(V(:));
fprintf('Rata-rata Hue: %f\n', mean_H);
fprintf('Rata-rata Saturation: %f\n', mean_S);
fprintf('Rata-rata Value: %f\n', mean_V);
```

Langkah 6: Penggunaan Ciri Warna untuk Klasifikasi atau Segmentasi

Setelah ciri warna diekstraksi, kita dapat menggunakannya untuk aplikasi seperti klasifikasi citra atau segmentasi. Misalnya, kita dapat membandingkan rata-rata warna dengan nilai referensi untuk menentukan kategori citra.

```
% Contoh penggunaan rata-rata warna untuk segmentasi
threshold = 0.5;
binary_mask = (mean_H > threshold);  % Segmentasi berdasarkan nilai Hue
imshow(binary_mask);
title('Segmentasi Berdasarkan Hue');
```

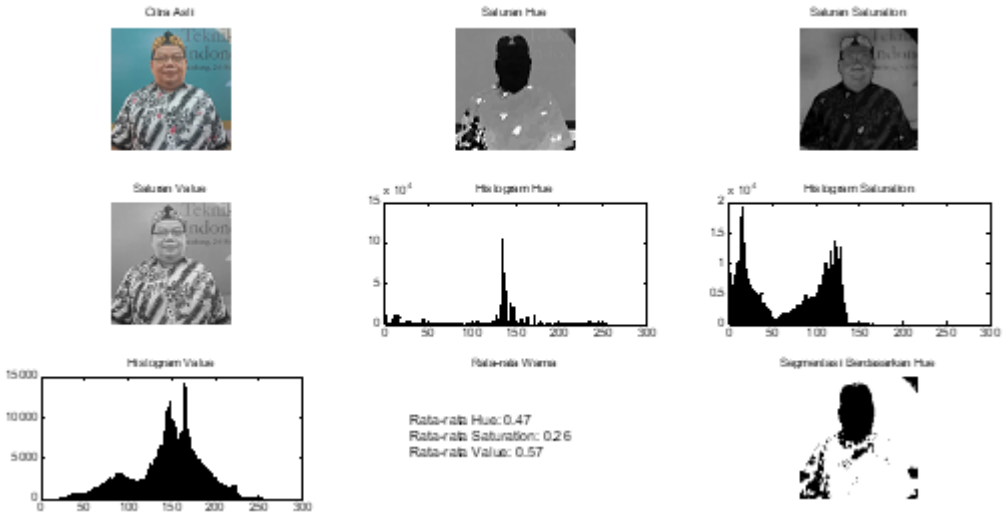
Kode lengkapnya:

```
% Langkah 1: Membaca Citra
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra
```

Anda

```
% Langkah 2: Mengubah Citra RGB ke Ruang Warna HSV
hsv_img = rgb2hsv(img);  % Mengubah citra RGB ke HSV
% Langkah 3: Ekstraksi Saluran Warna (Hue, Saturation, Value)
H = hsv_img(:,:,1);  % Komponen Hue
S = hsv_img(:,:,2);  % Komponen Saturation
V = hsv_img(:,:,3);  % Komponen Value
% Langkah 4: Membuat Figure dengan Subplot
% Menampilkan citra asli di subplot pertama (posisi 1)
figure;
subplot(3,3,1);  % 3x3 grid, gambar pertama
imshow(img);
title('Citra Asli');
% Menampilkan saluran Hue di subplot kedua (posisi 2)
subplot(3,3,2);  % 3x3 grid, gambar kedua
imshow(H);
title('Saluran Hue');
% Menampilkan saluran Saturation di subplot ketiga (posisi 3)
subplot(3,3,3);  % 3x3 grid, gambar ketiga
imshow(S);
title('Saluran Saturation');
% Menampilkan saluran Value di subplot keempat (posisi 4)
subplot(3,3,4);  % 3x3 grid, gambar keempat
imshow(V);
title('Saluran Value');
% Menghitung Histogram Warna untuk Komponen Hue
hist_H = imhist(H);
subplot(3,3,5);  % 3x3 grid, gambar kelima
bar(hist_H);
title('Histogram Hue');
% Menghitung Histogram Warna untuk Komponen Saturation
hist_S = imhist(S);
subplot(3,3,6);  % 3x3 grid, gambar keenam
bar(hist_S);
title('Histogram Saturation');
% Menghitung Histogram Warna untuk Komponen Value
hist_V = imhist(V);
subplot(3,3,7);  % 3x3 grid, gambar ketujuh
bar(hist_V);
title('Histogram Value');
% Langkah 5: Menghitung Rata-rata Warna untuk Setiap Saluran
mean_H = mean(H(:));  % Rata-rata Hue
mean_S = mean(S(:));  % Rata-rata Saturation
mean_V = mean(V(:));  % Rata-rata Value
% Menampilkan hasil rata-rata warna di subplot kedelapan (posisi 8)
subplot(3,3,8);  % 3x3 grid, gambar kedelapan
text(0.1, 0.5, sprintf('Rata-rata Hue: %.2f\nRata-rata Saturation: %.2f\nRata-rata Value: %.2f', mean_H,
mean_S, mean_V), 'FontSize', 12);
axis off;
title('Rata-rata Warna');
% Langkah 6: Segmentasi Berdasarkan Nilai Hue (Sebagai Contoh)
threshold_H = 0.5;  % Nilai threshold untuk komponen Hue
binary_mask_H = H > threshold_H;  % Segmentasi berdasarkan Hue
subplot(3,3,9);  % 3x3 grid, gambar kesembilan
imshow(binary_mask_H);
title('Segmentasi Berdasarkan Hue');
```

Hasil:



Gambar 11.5: Ekstrasi ciri berdasarkan warna

11.5 Ekstrasi Ciri Berdasarkan Tekstur

Ekstraksi ciri berdasarkan tekstur adalah proses untuk mengidentifikasi dan mengekstrak informasi yang terkait dengan pola atau struktur tekstural dalam citra. Tekstur pada citra merujuk pada pengulangan pola atau variasi intensitas yang dapat menggambarkan permukaan objek dalam citra, seperti halus, kasar, teratur, atau acak. Dalam pengolahan citra, tekstur sering digunakan untuk tujuan klasifikasi atau segmentasi objek.

Tekstur citra dapat diperoleh dengan berbagai metode, antara lain:

1. **Statistik Lokal (Statistical Methods):**
   - Menggunakan statistik lokal pada citra untuk mengidentifikasi pola tekstur, seperti rata-rata, standar deviasi, energi, entropi, kontrast, homogeneity, dll.
2. **Matrix Ko-ocurrence (GLCM – Gray-Level Co-occurrence Matrix):**
   - GLCM adalah metode yang sangat umum digunakan untuk menganalisis tekstur dalam citra. Matriks ini menggambarkan seberapa sering pasangan piksel dengan nilai tertentu muncul pada jarak dan arah tertentu dalam citra.
3. **Transformasi Frekuensi (Fourier Transform, Wavelet Transform):**
   - Metode ini mengubah citra ke domain frekuensi untuk mendapatkan informasi tekstural yang lebih terstruktur dan terpisah dari noise.
4. **Filter Gabor:**
   - Gabor filter digunakan untuk mengekstraksi fitur tekstur dengan menggunakan filter berbasis frekuensi lokal yang dapat menangkap variasi pada berbagai orientasi dan skala.

Untuk ekstraksi ciri berbasis tekstur pada citra menggunakan **GLCM (Gray-Level Co-occurrence Matrix)**, berikut adalah contoh program MATLAB.

Langkah-langkah:

1. **Membaca Citra**
2. **Konversi ke Citra Grayscale (jika diperlukan)**
3. **Menerapkan GLCM**
4. **Menghitung Fitur Tekstur seperti Kontras, Entropi, Energi, dll.**
5. **Menampilkan Hasil**

Berikut adalah contoh kode untuk ekstraksi ciri tekstur menggunakan GLCM pada MATLAB:

```
% Langkah 1: Membaca citra
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
% Langkah 2: Konversi citra ke grayscale (jika citra berwarna)
Igray = rgb2gray(I);  % Hanya diperlukan jika citra berwarna
% Langkah 3: Membuat GLCM (misalnya, untuk jarak 1 dan arah 0 derajat)
glcm = graycomatrix(Igray, 'Offset', [0 1]);  % Offset [0 1] berarti arah horizontal

% Langkah 4: Menghitung fitur tekstur dari GLCM
stats = graycoprops(glcm);  % Mengambil statistik dari GLCM
```

```
% Menampilkan statistik GLCM
fprintf('Kontras: %.4f\n', stats.Contrast);  % Menampilkan kontras
fprintf('Energi: %.4f\n', stats.Energy);    % Menampilkan energi
fprintf('Homogenitas: %.4f\n', stats.Homogeneity); % Menampilkan homogenitas

% Langkah 5: Menghitung Entropi secara manual
% Normalisasi GLCM
glcm_norm = glcm / sum(glcm(:));  % Normalisasi untuk menjadikan nilai totalnya 1

% Hitung entropi
entropy_val = -sum(glcm_norm(:) .* log2(glcm_norm(:) + eps));  % Menambahkan eps untuk menghindari log(0)
% Menampilkan entropi
fprintf('Entropi: %.4f\n', entropy_val);
% Langkah 6: Menampilkan GLCM untuk visualisasi
figure;
imshow(glcm, []);
title('GLCM');
```

Hasil:

Gambar 11.6: Ekstrasi ciri berdasarkan GLCM

Penjelasan Kode:

1. **imread('citra.jpg')**: Membaca citra dari file. Jika citra berwarna, kita perlu mengonversinya menjadi citra grayscale menggunakan **rgb2gray()**.
2. **graycomatrix(Igray)**: Fungsi ini digunakan untuk menghitung GLCM pada citra grayscale. Parameter Offset menentukan arah dan jarak piksel yang dihitung. Dalam contoh ini, [0 1] berarti arah horizontal dengan jarak 1 piksel.
3. **graycoprops(glcm)**: Fungsi ini mengembalikan statistik dari GLCM seperti kontras, energi, homogenitas, dan entropi. Statistik ini digunakan sebagai fitur tekstur untuk karakterisasi citra.
4. **imshow(glcm, [])**: Menampilkan GLCM dalam bentuk gambar, untuk visualisasi hasil.

Fitur Tekstur Umum yang Dihitung:

- **Kontras**: Mengukur variasi intensitas antara piksel yang berdekatan.
- **Energi**: Mengukur seberapa banyak informasi yang terkandung dalam tekstur (keteraturan).
- **Homogenitas**: Mengukur seberapa seragam tekstur di seluruh citra.
- **Entropi**: Mengukur kompleksitas atau ketidakteraturan tekstur.

Penggunaan Lanjut

Setelah ekstraksi fitur tekstur, Anda bisa menggunakan hasilnya untuk aplikasi lebih lanjut, seperti:

- **Klasifikasi Citra**: Menggunakan algoritma machine learning untuk mengklasifikasikan objek berdasarkan fitur tekstur.
- **Segmentasi**: Memisahkan area citra berdasarkan tekstur yang berbeda.
- **Pengenalan Pola**: Mengidentifikasi objek berdasarkan ciri-ciri tekstur.

11.6 Ekstrasi Ciri Berdasarkan Transformasi

Ekstraksi ciri adalah proses untuk mengidentifikasi dan mengekstrak fitur-fitur penting dari citra yang digunakan untuk analisis lebih lanjut, seperti pengenalan pola, klasifikasi, dan segmentasi citra. Salah satu metode yang populer dalam ekstraksi ciri adalah menggunakan transformasi pada citra. Transformasi ini digunakan untuk mengubah citra ke dalam bentuk yang lebih mudah dianalisis dan mengekstrak informasi penting yang ada di dalamnya.

Jenis-Jenis Transformasi dalam Ekstraksi Ciri

Beberapa transformasi yang sering digunakan dalam ekstraksi ciri berdasarkan citra antara lain:

1. **Transformasi Fourier (FFT - Fast Fourier Transform)**:

- Transformasi ini digunakan untuk menganalisis frekuensi dalam citra.
- Ciri yang diekstrak berupa informasi tentang frekuensi spasial yang ada dalam citra, yang bisa digunakan untuk mendeteksi pola atau tekstur.

2. **Transformasi Wavelet**:
   - Transformasi wavelet dapat digunakan untuk mendapatkan informasi multi-resolusi atau multi-skala dari citra.
   - Ciri yang dihasilkan berupa komponen-komponen frekuensi dengan resolusi tinggi di beberapa area citra dan komponen frekuensi rendah untuk keseluruhan citra.
3. **Transformasi Principal Component Analysis (PCA)**:
   - PCA digunakan untuk mereduksi dimensi citra dan menemukan komponen utama yang menggambarkan variasi data.
   - Ciri yang diekstrak berupa kombinasi linear dari piksel citra yang memiliki variansi terbesar, sehingga memungkinkan representasi citra dengan informasi yang lebih sedikit.
4. **Transformasi Hough**:
   - Digunakan untuk mendeteksi objek berbentuk geometris seperti garis atau lingkaran dalam citra.
   - Ciri yang diekstrak berupa parameter dari objek yang terdeteksi, misalnya koordinat dan sudut garis.
5. **Transformasi Zernike Moments**:
   - Digunakan untuk mengekstrak ciri berbasis bentuk dan kontur dari citra.
   - Zernike moments memberikan representasi citra dalam bentuk koefisien yang dapat digunakan untuk analisis bentuk objek.

MATLAB adalah salah satu perangkat lunak yang sering digunakan untuk memproses citra dan melakukan ekstraksi ciri dengan berbagai jenis transformasi. Berikut adalah contoh pemrograman dasar untuk beberapa transformasi dalam ekstraksi ciri citra menggunakan MATLAB.

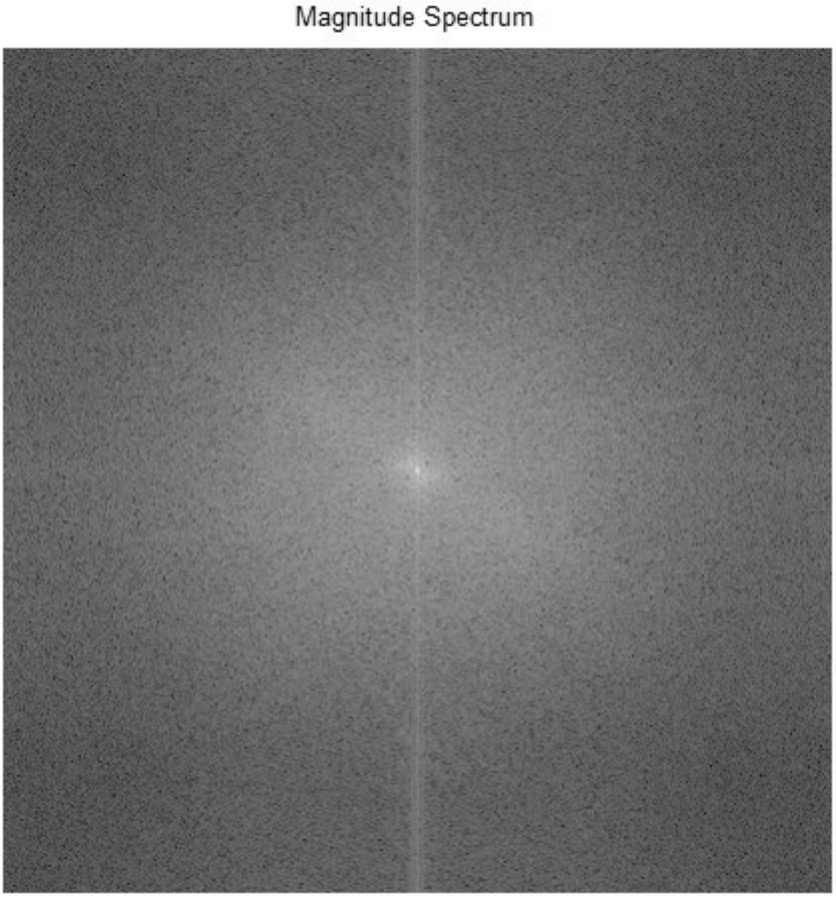
1. Transformasi Fourier (FFT)

```
% Membaca citra
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
I_gray = rgb2gray(I); % Mengonversi citra menjadi grayscale

% Menghitung Transformasi Fourier
F = fft2(double(I_gray));
Fshift = fftshift(F); % Memindahkan komponen frekuensi nol ke pusat

% Menampilkan magnitudo spektrum
magnitude = abs(Fshift);
imshow(log(1+magnitude), []); % Log skala untuk visualisasi yang lebih baik
title('Magnitude Spectrum');
```

Hasil:



Gambar 11.7: Ekstrasi ciri menggunakan FFT

2. Transformasi Wavelet

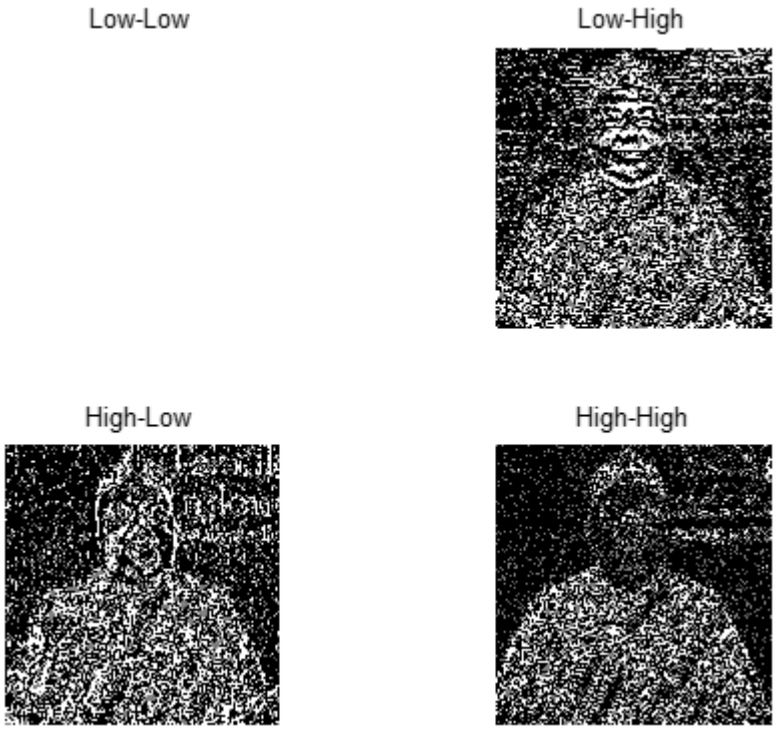
MATLAB memiliki fungsi untuk melakukan transformasi wavelet menggunakan paket Wavelet Toolbox. Berikut adalah contoh penerapannya:

```
% Membaca citra
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
I_gray = rgb2gray(I);

% Menghitung transformasi wavelet dengan dekomposisi 2 level
[LL, LH, HL, HH] = dwt2(I_gray, 'haar');

% Menampilkan hasil dekomposisi wavelet
subplot(2,2,1), imshow(LL), title('Low-Low');
subplot(2,2,2), imshow(LH), title('Low-High');
subplot(2,2,3), imshow(HL), title('High-Low');
subplot(2,2,4), imshow(HH), title('High-High');
```

Hasil:



Gambar 11.8: Ekstrasi ciri menggunakan Wavelet

3. Transformasi PCA

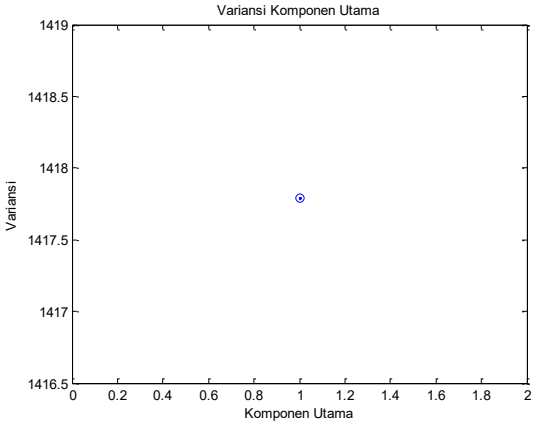
Untuk melakukan PCA pada citra dan mengekstrak ciri utama:

```
% Membaca citra
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
I_gray = rgb2gray(I);
I_vector = double(I_gray(:)); % Mengubah citra menjadi vektor

% Menghitung PCA
[coeff, score, latent] = pca(I_vector);

% Menampilkan komponen utama
figure;
plot(latent, 'o-');
title('Variansi Komponen Utama');
xlabel('Komponen Utama');
ylabel('Variansi');
```

Hasil:



Gambar 11.9: Ekstrasi ciri menggunakan PCA

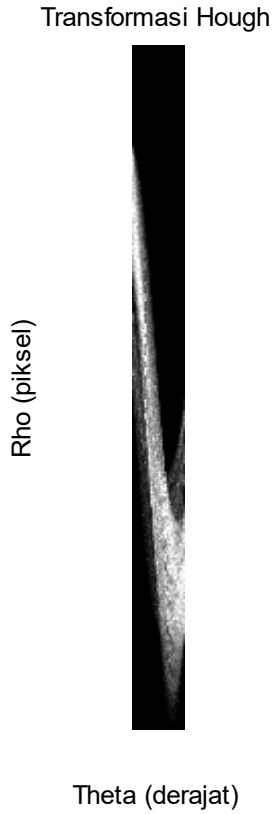
4. Transformasi Hough
Untuk mendeteksi garis menggunakan Transformasi Hough:

```
% Membaca citra
I = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path citra Anda
I_gray = rgb2gray(I);
BW = edge(I_gray, 'Canny'); % Deteksi tepi dengan metode Canny

% Transformasi Hough untuk mendeteksi garis
[H, theta, rho] = hough(BW);

% Menampilkan citra Hough
figure;
imshow(imadjust(mat2gray(H)), [], 'XData', theta, 'YData', rho, 'InitialMagnification', 'fit');
xlabel('Theta (derajat)');
ylabel('Rho (piksel)');
title('Transformasi Hough');
```

Hasil:



Gambar 11.10: Ekstrasi ciri menggunakan Transformasi Hough

5. **Transformasi Zernike Moments**
Membuat fungsi untu transformasi Zernike Moments:

```
function ZM = ZernikeMoments(image, n_max)
   % Mengubah citra ke grayscale (jika citra berwarna)
   if size(image, 3) == 3
      image = rgb2gray(image);
   end

   % Mengonversi citra grayscale menjadi citra biner
   image = double(image);
   image = image / 255;  % Menormalisasi citra ke rentang [0, 1]
```

```
    % Tentukan threshold untuk binarisasi manual
    threshold = 0.5;
    image = image > threshold;  % Citra biner (1 untuk objek, 0 untuk latar belakang)

    % Menentukan ukuran citra
    [rows, cols] = size(image);

    % Membuat koordinat polar
    [X, Y] = meshgrid(1:cols, 1:rows);
    [theta, r] = cart2pol(X - cols/2, Y - rows/2);

    % Normalisasi radius
    r_max = min(rows, cols) / 2;
    r = r / r_max;

    % Menyimpan Zernike Moments
    ZM = zeros(n_max+1, n_max+1);  % Matriks untuk menyimpan Zernike Moments

    % Iterasi untuk menghitung Zernike Moments
    for n = 0:n_max
        for m = -n:n
            if mod(n-m, 2) == 0
                % Polinomial radial Zernike R_{n,m}(r)
                Rnm = ZernikeRadialPolynomial(n, m, r);

                % Menentukan indeks matriks untuk Zernike Moment
                if m < 0
                    row_idx = n + 1;
                    col_idx = m + n + 1;  % Menyesuaikan indeks untuk m negatif
                else
                    row_idx = n + 1;
                    col_idx = m + n + 1;
                end

                % Menghitung Zernike Moment
                ZM(row_idx, col_idx) = sum(sum(image .* Rnm .* exp(1i * m * theta))) / (pi * r_max^2);
            end
        end
    end
end

function Rnm = ZernikeRadialPolynomial(n, m, r)
    % Fungsi untuk menghitung polinomial radial Zernike
    Rnm = zeros(size(r));
    for k = 0:floor((n-abs(m))/2)
        Rnm = Rnm + (-1)^k * factorial(n-k) / (factorial(k) * factorial((n+abs(m))/2-k) * factorial((n-
abs(m))/2-k)) * r.^(n-2*k);
    end
end

Program untuk menjalankan:
img = imread('E:\HINDARTO\2024\Buku Ajar 2024\Gambar\foto hin.jpg'); % Gantilah dengan path
citra Anda

ZM = ZernikeMoments(img, 5)
```

Hasil:

```
ZM =
  Columns 1 through 4
   0.6339 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.0738 + 0.0851i   0.0000 + 0.0000i  -0.0738 - 0.0851i   0.0000 + 0.0000i
  -0.0864 - 0.0567i   0.0000 + 0.0000i   0.2604 + 0.0000i   0.0000 + 0.0000i
   0.0227 + 0.0590i   0.0000 + 0.0000i  -0.0450 + 0.1558i   0.0000 + 0.0000i
  -0.2205 - 0.0176i   0.0000 + 0.0000i  -0.0209 - 0.0383i   0.0000 + 0.0000i
   0.0503 - 0.1082i   0.0000 + 0.0000i   0.0364 + 0.2214i   0.0000 + 0.0000i


  Columns 5 through 8

   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.0864 + 0.0567i   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.0450 - 0.1558i   0.0000 + 0.0000i   0.0227 - 0.0590i   0.0000 + 0.0000i
   0.5735 + 0.0000i   0.0000 + 0.0000i  -0.0209 + 0.0383i   0.0000 + 0.0000i
  -0.0806 + 0.3686i   0.0000 + 0.0000i  -0.0806 - 0.3686i   0.0000 + 0.0000i

  Columns 9 through 11

   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0000 + 0.0000i   0.0000 + 0.0000i   0.0000 + 0.0000i
  -0.2205 + 0.0176i   0.0000 + 0.0000i   0.0000 + 0.0000i
   0.0364 - 0.2214i   0.0000 + 0.0000i   0.0503 + 0.1082i
```

Soal-soal Latihan:

1. Jelaskan apa yang dimaksud dengan ekstraksi fitur citra dalam pengolahan citra. Mengapa ekstraksi fitur penting dalam aplikasi seperti pengenalan pola, deteksi objek, dan klasifikasi citra?
2. Apa yang dimaksud dengan ekstraksi tepi (edge detection) dalam pengolahan citra? Sebutkan dua teknik yang digunakan untuk ekstraksi tepi dan jelaskan bagaimana masing-masing metode bekerja.
3. Jelaskan apa yang dimaksud dengan ekstraksi tekstur pada citra. Sebutkan dua metode ekstraksi tekstur yang sering digunakan dalam pengolahan citra dan berikan contoh aplikasi di mana ekstraksi tekstur digunakan.
4. Dalam konteks ekstraksi fitur, apa yang dimaksud dengan fitur bentuk (shape features)? Sebutkan beberapa jenis fitur bentuk yang dapat diekstraksi dari citra dan bagaimana fitur tersebut dapat digunakan untuk pengenalan objek atau klasifikasi citra.
5. Dalam MATLAB, bagaimana cara menerapkan deteksi tepi menggunakan operator Sobel untuk ekstraksi fitur tepi pada citra grayscale? Jelaskan langkah-langkah yang perlu dilakukan dan berikan contoh kode MATLAB untuk menerapkan deteksi tepi menggunakan operator Sobel.

Pustaka

Anwar, S., & Priscylo, G. (2019). 薛加玉 1， 龚 尹 2， 韩顺平 2 （. *J. Pijar MIP14*, *14*(1), 1–12.

Asiva Noor Rachmayani. (2015a). *No 主観的健康感を中心とした在宅高齢者における 健康関連指標に関する共分散構造分析Title*. 6.

Asiva Noor Rachmayani. (2015b). *No 主観的健康感を中心とした在宅高齢者における 健康関連指標に関する共分散構造分析Title*.

Bone, R., Detection, F., Morphological, U., Based, G., Technique, S., Penstruktur, E., Residue, D., & Operator, E. (n.d.). *DETEKSI KERETAKAN TULANG MENGGUNAKAN GRADIENT*.

Danoedoro, P., & Adiwijoyo. (2014). Perbandingan Teknik Resampling Pada Citra Hasil Pan-Sharpening Untuk Pemetaan Penutup Lahan Dengan Menggunakan Klasifikasi Terselia Maximum Likelihood. *Jurnal Bumi Indonesia*, *3*(4).

Dwdv, L., Gdq, H. N., Hodndqj, D., Lqwhqvlwdv, G., Sdgd, F. D., & Gzlpdwud, E. (n.d.). *, Psohphqwdvl 0Ruskrorj \ & Rqfhsw Dqg 7Hfkqltxh Gdodp 3Hqjrodkdq & Lwud 3Urvhv Shqjdpelodq Gdwd Lpdjh Glnhpedqjndq Xqwxn Phpshurohk Shqjdpdwdq Re \ Hn Sdvwl Sdgd Vxdwx*. 0–4.

Hendrawan, A., Kom, M., Wakhidah, N., Kom, S., Cs, M., Prathivi, R., & Kom, M. (2021). *Pengolahan Citra Digital Dengan Menggunakan Python* (Issue January).

Ikbal, M. C. (2017). Analisis Strategi Pengolahan Baseline Gps Berdasarkan Jumlah Titik Ikat dan Variasi Waktu Pengamatan. *Jurnal Geodesi Undip*, *6*, 237.

Irianto, S. Y. (2016). Analisa Citra Digital dan Content Based Image Retrieval. *Perpustakaan Nasional RI*, *December*.

JASMINE, K. (2014). 済無No Title No Title No Title. *Penambahan Natrium Benzoat Dan Kalium Sorbat (Antiinversi) Dan Kecepatan Pengadukan Sebagai Upaya Penghambatan Reaksi Inversi Pada Nira Tebu*, 1–10.

Kurniastuti, I., & Andini, A. (2018). Perancangan Program Penentuan Histogram Citra Dengan Graphical User Interface (Gui). *Applied Technology and Computing Science Journal*, *1*(1), 11–17. https://doi.org/10.33086/atcsj.v1i1.4

Lesmana, C., & Ramadhani, D. (2019). *Ekstraksi Fitur Boundary Region Untuk Uang Logam*. *3*(1).

Lévesque, L. (2014). Nyquist sampling theorem: Understanding the illusion of a spinning wheel captured with a video camera. *Physics Education*, *49*(6), 697–705. https://doi.org/10.1088/0031-9120/49/6/697

Noor Santi, C. (2011). Mengubah Citra Berwarna Menjadi Gray Scale dan Citra biner Rina. *Teknologi Informasi DINAMIK*, *16*(1), 14–19.

Orisa, M., & Hidayat, T. (2019). Analisis Teknik Segmentasi Pada Pengolahan Citra. *Jurnal Mnemonic*, *2*(2), 9–13. https://doi.org/10.36040/mnemonic.v2i2.84

Pangaribuan, H. (2019). Optimalisasi Kualitas Citra Digital Dengan Metode Ketetanggaan Piksel. *Jurnal Ilmiah Informatika*, *7*(01), 18.

Putra, R. D., Napitupulu, H. S., Nugraha, A. H., Suhana, M. P., Ritonga, A. R., & Sari, T. E. Y. (2022). Pemetaan Luasan Hutan Mangrove Dengan Menggunakan Citra Satelit Di Pulau Mapur, Provinsi Kepulauan Riau. *Jurnal Kelautan Tropis*, *25*(1), 20–30. https://doi.org/10.14710/jkt.v25i1.12294

Said, K. A. M., & Jambek, A. B. (2021). Analysis of Image Processing Using Morphological Erosion and Dilation. *Journal of Physics: Conference Series*, *2071*(1). https://doi.org/10.1088/1742-6596/2071/1/012033

Saravanan, C. (2010). Color image to grayscale image conversion. *2010 2nd International Conference on Computer Engineering and Applications, ICCEA 2010*, *2*(April 2010), 196–199. https://doi.org/10.1109/ICCEA.2010.192

Sembiring, A. S. (2012). Operasi-operasi Dasar Pengolahan Citra Digital. *Operasi-Operasi Dasar Pengolahan Citra Digital*, 41–60. https://asanisembiring.wordpress.com/modul-kuliah/pengolahan-citra-s1/

Sugiarti, S. (2018). Peningkatan Kualitas Citra Dengan Metode Fuzzy Possibility Distribution. *ILKOM Jurnal Ilmiah*, *10*(1), 100–104. https://doi.org/10.33096/ilkom.v10i1.226.100-104

Sunardi, H., Zulkifli, Z., & Antony, F. (2021). Transformasi Geometri Rotasi Citra Digital Untuk Mendapatkan Kompresi Optimal Menggunakan Metode Lossless Dan Lossy. *Jurnal Ilmiah Informatika Global*, *12*(1), 15–22. https://doi.org/10.36982/jiig.v12i1.1540

Tarigan, I. J. (2018). Teknik Filter Mean dan Median untuk Perbaikan Citra. *Jurnal Armada Informatika*, *2*(1), 30–36. https://jurnal.stmikmethodistbinjai.ac.id/

Trianto, G. A., Sinaga, F. J., Marzuki, M. F., & Qorni, Q. Al. (2022). Operasi Opening dan Closing pada

Pengolahan Citra Digital Menggunakan Matlab. *Mdp Student Conference (Msc)*, 104–110.
Ummah, M. S. (2019). No 主観的健康感を中心とした在宅高齢者における 健康関連指標に関する共分散構造分析Title. *Sustainability (Switzerland)*, *11*(1), 1–14. http://scioteca.caf.com/bitstream/handle/123456789/1091/RED2017-Eng-8ene.pdf?sequence=12&isAllowed=y%0Ahttp://dx.doi.org/10.1016/j.regsciurbeco.2008.06.005%0Ahttps://www.researchgate.net/publication/305320484_SISTEM_PEMBETUNGAN_TERPUSAT_STRATEGI_MELESTARI
Windana, F., Sarosa, M., & Santoso, P. (2014). Implementasi Kombinasi Feature Extraction Untuk Content Based Image Retrieval. *Jurnal EECCIS*, *8*(2), 169–174.
Wulandari, M. (2019). Filterisasi Noise Pada Citra Uang Logam Indonesia. *TESLA: Jurnal Teknik Elektro*, *20*(1), 60. https://doi.org/10.24912/tesla.v20i1.2967
Yelliy N, N. (2019). Pengolahan Citra Digital Perbandingan Metode Histogram Equalization Dan Spesification Pada Citra Abu-Abu. *J-Icon*, *7*(1), 87–95. https://ejurnal.undana.ac.id/index.php/jicon/article/view/889
Yuwono, B. (2015). Image Smoothing Menggunakan Mean Filtering, Median Filtering, Modus Filtering Dan Gaussian Filtering. *Telematika*, *7*(1). https://doi.org/10.31315/telematika.v7i1.416
Zufar, M. (1998). Introductory Computer Vision and Image Processing. *Sensor Review*, *18*(3), 2–4. https://doi.org/10.1108/sr.1998.08718cae.001

Biodata Penulis:

Dr. Hindarto, S. Kom, MT. born in Surabaya, 30 July 1973. In 1995, the author received a Diploma from the Surabaya State Polytechnic and continued his Bachelor's degree in the Informatics study program, Faculty of Engineering, Umsida, Sidoarjo Indonesia. The author continues his master's degree in ITS Electrical Engineering with a scholarship program from DIKTI. In 2007, the author officially received the title M.T. The author continued his doctoral studies at ITS Electrical Engineering Multimedia Smart Network Surabaya, Indonesia with a scholarship program from DIKTI. In 2016, the author officially received the title Dr. The author started his career as a lecturer at the Informatics study program at Muhaammdiyah Sidoarjo University in 2004. Some of the research carried out by the author was on Heart Signal Detection and EEG Signal Detection.

Ade Eviyanti, S,Kom., M.Kom. Saya berdomisili di BUMI CANDI ASRI BLOK C6 NO 12 NGAMPELSARI CANDI SIDOARJO. Saya lahir di Jakarta pada 24 Juni 1978, dan memulai pendidikan di SDN 6 Makassar. Setelah itu, ia melanjutkan ke jenjang pendidikan menengah pertama di Hang Tuah Makassar, dan sekolah menengah atas di SMAN 6 Makassar. Ade berhasil meraih gelar sarjana di Program Studi Teknik Informatika, Fakultas Teknik, Universitas Muhammadiyah Sidoarjo. Kemudian, ia melanjutkan studi magister di STTS dan memperoleh gelar Magister Komputer pada tahun 2018. Saat ini, Ade sedang menempuh pendidikan doktoral di Institut Teknologi Sepuluh Nopember (ITS) dengan masuk pada tahun akademik 2021/2022, berkomitmen untuk mengembangkan pengetahuan dan keterampilan di bidang teknologi informasi dan komputer.

Suhendro Busono, ST, MT, Dilahirkan tanggal 30 December 1982 penulis mendapatkan sarjana Diploma III,PENS(Politeknik Elektronika Surabaya) ITS Surabaya, Jurusan Teknik Telekomunikasi, Surabaya - Jawa Timur, Periode 2002 – 2005, Diploma IV, PENS(Politeknik Elektronika Surabaya) ITS Surabaya, Jurusan Teknik Informatika, Surabaya – Jawa Timur, Periode 2008 – 2010, dan Strata 2, Universitas Dian Nuswantoro, Jurusan Ilmu Komputer, Semarang – Jawa Tengah , Periode 2015 – 2017. Penulis menjadi dosen tetap di Universitas Muhammadiyah Sidoarjo tahun 2018.

Ir. Sumarno, MM, Lahir di Surabaya, 27 Mei 1961, Sarjana S 1 Teknik elektro di fakultas Teknik Universitas Muhammadiyah Surabaya tahun 1991, pada tahun 2008 Sarjana S2, magister manajemen sains di Universitas Muhammadiyah Malang, di fakultas Saintek, memberikan mata kuliah teori bahasa automata ,Sistem Informasi dan Manajemen proyek teknologi informasi sampai sekarang.

Jamaaluddin, lahir di Surabaya, 17 Oktober 1970. Ia tercatat sebagai lulusan Universitas Brawijaya Malang (Teknik Elektro-Strata 1), Universitas Muhammadiyah Yogjakarta (Magister Manajemen-Strata 2) dan Institut Teknologi Sepuluh Nopember Surabaya (Electrical POwer Engineering – Strata 3). Pria yang memiliki nama panggilan Jamaal ini adalah Dosen di Prodi Teknik Elektro Universitas Muhammadiyah Sidoarjo sejak tahun 2013. Mata kuliah yang diampu adalah Bahan Bahan Listrik, Teknik Tenaga Listrik dan Kecerdasan Buatan. Dalam Kegiatan sebagai dosen telah melakukan penelitian rata – rata 3 skema tiap tahun, yaitu dari Riset Dikti, BRIN LPDP dan Riset Muhammadiyah. Begitu juga untuk skema Pengabdian pada Masyarakatnya. Disamping itu juga produktif dalam membuat buku baik buku ajar dan buku referensi. Dalam kurun 10 tahun ini telah memproduksi buku lebih dari 10 buku. Sedangkan dalam penulisan artikel yang

terindeks Scopus sebanyak 31 artikel dengan H Index – 6, dan H Index Google Schoolar 13. Jamaaluddin memiliki Paten Sederhana yang telah Granted dengan Judul "Lampu dengan Sensor Gerak", dan yang masih dalam proses pengajuan Paten sederhana sebanyak 5 judul, disamping puluhan Hak cipta. Disamping sebagai dosen Jamaaluddin juga menjadi reviewer Jurnal Nasional dan Internasional. Juga mendapatkan amanah sebagai Pengurus Forum Teknik Elektro Indonesia (Jawa Timur dan Indonesia). Untuk kegiatan kemasyarakatan yang dilakukan adalah sebagai Wakil Ketua Lembaga Pembinaan Haji dan Umrah Pimpinan Wilayah Muhammadiyah Jawa Timur, Sebagai Ketua Lembaga Pembinaan Haji dan Umrah Pimpinan Daerah Muhammadiyah Sidoarjo dan lain sebagainya. Jamaaluddin banyak memberikan pelatihan dan motivasi tentang Renewable energy, tentang Power system, Kewirausahaan dan tentang Hypnoteraphy.

ISBN 978-623-464-121-9 (PDF)

Umsida Press
Universitas Muhammadiyah Sidoarjo
Jl.Mojopahit No. 666 B
Sidoarjo, Jawa Timur