Pengolahan Citra Digital dan Dasar Visi Komputer dengan Python

Aulia Akhrian Syahidi

Pengolahan Citra Digital dan Dasar Visi Komputer dengan Python

Undang-Undang No. 28 Tahun 2014 Tentang Hak Cipta

Fungsi dan sifat hak cipta Pasal 4

Hak Cipta sebagaimana dimaksud dalam Pasal 3 huruf a merupakan hak eksklusif yang terdiri atas hak moral dan hak ekonomi.

Pembatasan Perlindungan Pasal 26

Ketentuan sebagaimana dimaksud dalam Pasal 23, Pasal 24, dan Pasal 25 tidak berlaku terhadap :

i. penggunaan kutipan singkat Ciptaan dan/atau produk Hak Terkait untuk pelaporan peristiwa aktual yang ditujukan hanya untuk keperluan penyediaan informasi aktual;
ii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk kepentingan penelitian ilmu pengetahuan;
iii. Penggandaan Ciptaan dan/atau produk Hak Terkait hanya untuk keperluan pengajaran, kecuali pertunjukan dan Fonogram yang telah dilakukan Pengumuman sebagai bahan ajar; dan
iv. penggunaan untuk kepentingan pendidikan dan pengembangan ilmu pengetahuan yang memungkinkan suatu Ciptaan dan/atau produk Hak Terkait dapat digunakan tanpa izin Pelaku Pertunjukan, Produser Fonogram, atau Lembaga Penyiaran.

Sanksi Pelanggaran Pasal 113

1. Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp 100.000.000 (seratus juta rupiah).
2. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp 500.000.000,00 (lima ratus juta rupiah).
3. Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp 1.000.000.000,00 (satu miliar rupiah).
4. Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp 4.000.000.000,00 (empat miliar rupiah).

Pengolahan Citra Digital dan Dasar Visi Komputer dengan Python

Aulia Akhrian Syahidi

POLIBAN PRESS

Pengolahan Citra Digital dan Dasar Visi Komputer dengan Python

Penulis :
Aulia Akhrian Syahidi

ISBN :
978-623-5259-11-6 (PDF)

Editor dan Tim Penyunting :
Reza Fauzan

Desain Sampul dan Tata letak :
Rahma Indera; Eko Sabar Prihatin

Profreader :
Prof. Dr. Eng. Fitri Utaminingrum, S.T., M.T.

Penerbit :
POLIBAN PRESS
Anggota APPTI (Asosiasi Penerbit Perguruan Tinggi Indonesia)
no.004.098.1.06.2019
Cetakan Pertama, 2024

Redaksi :
Politeknik Negeri Banjarmasin, Jl. Brigjen H. Hasan Basry,
Pangeran, Komp. Kampus ULM, Banjarmasin Utara
Telp : (0511)3305052
Email : press@poliban.ac.id

Diterbitkan pertama kali oleh :
Poliban Press, Banjarmasin, Januari 2024

KATA PENGANTAR

Dalam era revolusi teknologi, buku ini hadir sebagai panduan praktis yang sangat dibutuhkan dalam menggali seluk-beluk teknologi pengolahan citra digital dan visi komputer. Dengan cakupan yang komprehensif, buku ini memberikan pembaca, khususnya mahasiswa vokasi, pemahaman mendalam tentang konsep dasar, operasi, dan teknik pemrosesan lanjutan dalam domain ini.

Penulis tidak hanya menyajikan teori yang solid, tetapi juga mengintegrasikan kode program berbasis Python berdasarkan contoh kasus nyata, memberikan pengalaman praktis yang tak ternilai. Mulai dari konsep dasar pengolahan citra hingga tantangan dan perkembangan terkini dalam proyek visi komputer, buku ini memandu pembaca melalui perjalanan ilmiah yang membangun wawasan tentang aplikasi teknologi ini di berbagai sektor, seperti transportasi, keamanan publik, kesehatan, dan lingkungan.

Kami berharap buku ini tidak hanya menjadi panduan bermanfaat namun juga menjadi sumber inspirasi bagi pembaca untuk mengembangkan dan mengimplementasikan pengetahuan dalam proyek dan penelitian terapan di masa depan. Selamat membaca dan semoga buku ini membuka pintu menuju pemahaman yang lebih dalam tentang pengolahan citra digital dan visi komputer, serta memberikan kontribusi yang berarti dalam membangun smart city yang berkelanjutan dan industri 4.0 yang revolusioner.

Terima kasih kepada semua yang telah berkontribusi dalam merealisasikan buku ini.

November 2023

Penerbit

PRAKATA

Dalam era *smart city* dan industri 4.0, penggunaan citra dan analisis visual memiliki peran yang semakin penting dalam berbagai bidang seperti transportasi, keamanan publik, kesehatan, dan lingkungan. Buku ini dirancang sebagai panduan praktis dalam pengolahan citra digital dan visi komputer serta dilengkapi dengan teori dan panduan praktis, kode program berdasarkan contoh kasus menggunakan Bahasa Pemrograman Python, soal teori, dan soal praktik serta sangat cocok digunakan bagi mahasiswa vokasi.

Kami akan membawa para pembaca atau mahasiswa melalui pengenalan konsep dasar pengolahan citra, termasuk prinsip dasar, representasi citra, operasi dasar, serta teknik pemrosesan lanjutan seperti deteksi dan segmentasi objek. Selanjutnya, kami membahas konsep dasar visi komputer, termasuk fitur ekstraksi, deskripsi objek, dan pengenalan pola. Kami juga mengeksplorasi pemrograman dasar visi komputer seperti deteksi wajah, pengenalan objek, dan pelacakan objek. Terakhir, buku ini juga menyoroti tantangan dan perkembangan terkini proyek dalam pengolahan citra digital dan visi komputer yang semakin relevan dengan perkembangan *smart city* dan industri 4.0.

Kami berharap buku ini memberikan pemahaman yang komprehensif dan memperluas wawasan Anda tentang pengolahan citra digital, dasar visi komputer, serta kontribusinya dalam membangun *smart city* yang berkelanjutan dan industri 4.0 yang revolusioner. Rasa terima kasih disampaikan kepada semua pihak dan terutama untuk P3M Politeknik Negeri Banjarmasin yang telah memfasilitasi penerbitan buku ini. Semoga buku ini bermanfaat dan Anda dapat menerapkan pengetahuan yang diperoleh ke dalam proyek dan penelitian terapan di masa depan.

Penulis

DAFTAR ISI

BAB 1

PENGENALAN DAN KONSEP PENGOLAHAN CITRA DIGITAL

Capaian Pembelajaran:

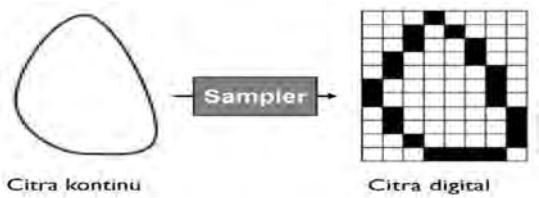
1. Mampu mengetahui dan memahami pengertian citra dan citra digital
2. Mampu mengetahui dan memahami pengertian pengolahan citra
3. Mampu mengetahui dan memahami hubungan pengolahan citra digital dengan disiplin ilmu lainnya
4. Mampu mengetahui dan memahami komponen-komponen pembentuk persepsi visual
5. Mampu mengetahui dan memahami formasi citra pada mata manusia
6. Mampu mengetahui dan memahami letak persepsi warna
7. Mampu mengetahui dan memahami tentang cahaya dan spektrum elektromagnetik
8. Mampu mengetahui dan memahami tentang penerapan pengolahan citra digital di berbagai bidang

1.1 Citra dan Citra Digital

Citra sering disebut juga sebagai gambar pada bidang dwimatra (2 Dimensi). Citra mengacu pada representasi visual dari objek atau *scene* yang dapat diamati oleh mata manusia atau perangkat sensor elektronik (Munir, 2019). Dimana manusia sendiri adalah makhluk visual. Manusia cukup memanfaatkan penglihatannya untuk memahami dunia di sekitarnya. Manusia ketika melihat sebuah benda tidak hanya untuk mengidentifikasi dan mengklasifikasi, akan tetapi juga dapat mengetahui perbedaan dan merasakan secara cepat. Mata manusia dapat dengan mudah beradaptasi dan menginterpretasikan sebuah objek untuk memperoleh informasi, dimana dalam dunia nyata sebuah objek dapat mengalami perubahan baik itu karena perbedaan siang dan malam; pengaruh cahaya

dan juga bayangan. Citra dapat berupa gambar dua dimensi atau tiga dimensi, tergantung pada dimensi objek yang direpresentasikan. Citra mempunyai karakteristik yang tidak dimiliki oleh data teks, yaitu citra kaya dengan informasi.

Dalam citra dibagi menjadi dua jenis yaitu citra analog dan digital. Citra analog atau kontinu yaitu citra yang diperoleh dari sistem optik yang dapat menerima sinyal analog, seperti mata manusia atau kamera analog. Citra digital, di sisi lain, adalah gambar dua dimensi yang dihasilkan dari gambar analog dua dimensi yang kontinu menjadi gambar diskrit melalui proses *sampling* (Lihat Gambar 1.1). Menurut Asmara (2018) menyebutkan bahwa citra digital (citra raster) merupakan representasi numerik dari citra dua dimensi. Citra ini direpresentasikan dalam format digital, yang terdiri dari piksel-piksel diskrit. Nilai numerik yang direpresentasikan ini umumnya adalah nilai biner 8 bit, yang kemudian disimpan pada elemen citra yang disebut sebagai piksel atau *pixel*. Piksel ini disimpan pada *memory computer* sebagai *map* raster yakni *array* dua dimensi bertipe integer.



Gambar 1.1. Transformasi Citra Analog menjadi Citra Digital

Citra digital terbentuk melalui proses pengambilan gambar menggunakan perangkat seperti kamera digital atau *scanner*, atau melalui proses transformasi citra analog menjadi format digital. Citra digital terdiri dari piksel-piksel kecil yang membentuk *grid* atau matriks. Setiap piksel mewakili sebuah titik dalam citra dan memiliki nilai numerik yang menggambarkan intensitas cahaya atau warna pada titik tersebut. Piksel-piksel ini dapat diolah secara digital menggunakan perangkat lunak pengolahan citra untuk mengubah atau memperbaiki citra, melakukan analisis citra, atau mengaplikasikan efek dan *filter* tertentu. Citra digital

a[m, n] merupakan citra dalam ruang diskrit 2D yang berasal dari citra analog *a(x,y)* di ruang kontinyu 2D melalui proses *sampling* yaitu yang biasa kita sebut sebagai digitalisasi (Young dkk., 1995).

Menurut Gonzalez & Woods (2018) menyebutkan bahwa suatu citra dapat didefinisikan sebagai fungsi dua dimensi *f(x,y)*. Dalam citra digital, koordinat spasial (bidang) yakni horizontal dan vertikal direpresentasikan oleh dua sumbu, *x* dan *y*, sedangkan nilai intensitas cahaya atau warna pada setiap titik dalam citra direpresentasikan oleh nilai *f* yang berhingga. Citra digital terbentuk melalui proses pengambilan sampel pada bayangan objek menggunakan perangkat seperti kamera digital atau *scanner*. Proses ini mengubah citra analog menjadi representasi diskrit dalam bentuk piksel-piksel.

Ketika *x, y,* dan nilai intensitas *f* semuanya berhingga dan diskrit, maka citra yang dihasilkan disebut sebagai bayangan digital. Piksel-piksel dalam citra digital memiliki posisi terdefinisi secara diskrit dan intensitasnya direpresentasikan oleh nilai numerik diskrit. Citra digital ini dapat dianalisis, dimanipulasi, atau disimpan menggunakan perangkat lunak pengolahan citra atau sistem komputer.

Setiap elemen pada larik *(array)* dinamakan sebagai piksel atau *picture element*. Dimensi citra biasanya ditulis dengan format panjang x tinggi (misalnya 1024 x 768 piksel). Namun, perlu diperhatikan bahwa secara matematis sebuah citra digital didefinisikan dengan ukuran tinggi *M* (misalnya 768) dan panjang *N* (misalnya 1024) dimana *M x N* menyatakan resolusi citra, serta *L* sebagai derajat keabuan. Definisi citra secara matematis terlihat pada Gambar 1.2, dimana *x* menunjukkan baris dan *y* menunjukkan kolom.

$$f(x,y)\begin{cases}0 \le x \le M\\ 0 \le y \le N\\ 0 \le f \le L\end{cases} \qquad f(x,y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1)\\ f(1,0) & f(1,1) & \cdots & f(1,N-1)\\ \vdots & \vdots & \vdots & \vdots\\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1)\end{bmatrix}$$

Gambar 1.2. Representasi Citra

Seperti pada layar monitor bahwa koordinat citra dimulai dari pojok kiri atas. Secara matematis dimulai dari (0,0) dan berakhir di (M-1, N-1). Perlu diingat bahwa untuk mengakses piksel citra, penulisan indeks secara matematis pada citra bersesuaian juga dengan penulisan indeks pada Bahasa Pemrograman Python.

Lebih lanjut, citra terdiri dari dua macam yaitu citra diam dan bergerak. Citra diam *(still image)* adalah sebuah citra tunggal. Sedangkan citra bergerak *(moving images)* adalah rangkaian citra diam yang ditampilkan secara beruntun (sekuensial) sehingga mampu memberikan kesan sebagai gambar bergerak atau animasi/video.

1.2 Pengolahan Citra

Citra perlu diolah atau diproses lebih lanjut, karena dalam bentuk aslinya terjadi penurunan kualitas citra yang disebabkan oleh pengaruh cahaya, citra sering kali mengandung *noise* (derau/sesuatu yang mengganggu), terlalu kontras, kurang tajam (terjadi *blur*), dan lain sebagainya sehingga berdampak pada hilangnya informasi yang ingin disampaikan atau informasi yang tidak relevan. Selain itu, pengolahan citra juga dapat membantu dalam memperbaiki kualitas citra, meningkatkan visualisasi, ekstraksi fitur, dan analisis informasi yang terkandung dalam citra.

Beberapa ahli mengemukakan tentang pengolahan citra yang khususnya tentang pengolahan citra digital (*digital image processing*) yaitu sebagai berikut.

1. Pengolahan citra yakni manipulasi citra oleh komputer dengan tujuan untuk meningkatkan kualitas citra, mendapatkan informasi yang berguna, atau mengambil keputusan berdasarkan citra tersebut (Gonzalez & Woods, 2018).
2. Pengolahan citra melibatkan serangkaian operasi matematika yang diterapkan pada citra digital untuk menghasilkan citra yang lebih baik, menghilangkan gangguan, atau mengekstrak informasi (Pratt, 2007).

3. Pengolahan citra melibatkan manipulasi piksel pada citra menggunakan teknik matematis dan algoritme komputer untuk menghasilkan suatu hasil yang diinginkan (Gonzalez dkk., 2011).
4. Pengolahan citra sebagai suatu disiplin yang berkaitan dengan pemrosesan, analisis, dan interpretasi citra digital untuk meningkatkan pemahaman manusia, dan mendukung pengambilan keputusan oleh mesin (Sonka dkk., 2015).

Pengolahan citra melibatkan manipulasi, analisis, dan ekstraksi informasi dari citra menggunakan teknik matematika dan algoritme komputer untuk mencapai berbagai tujuan, seperti perbaikan kualitas citra, pengenalan pola, pengambilan keputusan, dan pemrosesan *computer vision*.

Menurut Putra (2010) menyebutkan bahwa dalam pengolahan citra dapat dibagi menjadi tiga kategori yaitu sebagai berikut.

1. Kategori rendah yang melibatkan operasi-operasi sederhana seperti prapengolahan citra untuk mengurangi derau, pengaturan kontras, dan pengaturan ketajaman citra. Pengolahan kategori rendah ini memiliki *input* dan *output* berupa citra.
2. Kategori menengah yang melibatkan operasi-operasi seperti segmentasi dan klasifikasi citra. Proses pengolahan citra ini melibatkan *input* berupa citra dan *output* berupa atribut (fitur) citra yang dipisahkan dari citra *input*.
3. Kategori tinggi yang melibatkan proses pengenalan objek dan deskripsi citra, dimana hal ini merupakan tugas-tugas dari manusia yang mampu memanfaatkan *vision* (mata) manusia kemudian mengenali dan membedakannya.

Tujuan dari pengolahan citra tidak lain adalah untuk memperbaiki dan meningkatkan kualitas dari citra, namun terdapat tujuan lainnya diantaranya: 1) untuk memperbaiki tampilan citra *(image enhancement)*; 2) untuk mengurangi ukuran *file* citra dengan tetap mempertahankan kualitas citra *(image compression)*; 3) untuk memulihkan citra ke kondisi

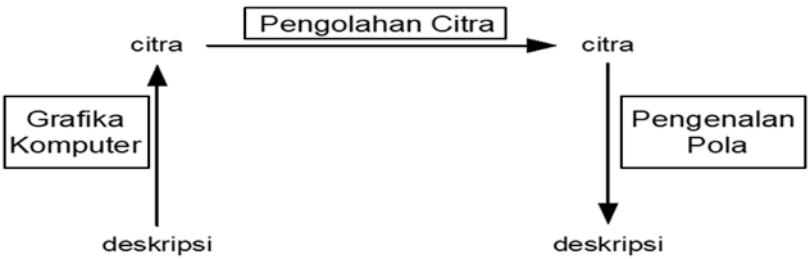
semula *(image restoration)*; 4) Untuk menyoroti ciri tertentu dari citra agar lebih mudah untuk dianalisis (Petrou & Petrou, 2010).

Operasi-operasi umum yang ada di dalam pengolahan citra digital diantaranya:

1. perbaikan kualitas citra (*image enhancement*),
2. pemampatan citra (*image compression*),
3. analisis citra (*image analysis*),
4. rekonstruksi citra (*image reconstruction*),
5. restorasi citra (*image restoration*), dan
6. segmentasi citra (*image segmentation*).

1.3 Hubungan Pengolahan Citra Digital dengan Disiplin Ilmu Lainnya

Terkait dengan citra, sebenarnya terdapat beberapa ilmu yang berkaitan (Lihat Gambar 1.3), namun tujuannya berbeda yaitu:



Gambar 1.3. Keterhubungan dengan Bidang Ilmu Lainnya

1. Grafika Komputer *(Computer Graphics)* adalah cabang ilmu informatika yang memproses/mengolah data berupa deskripsi menjadi citra. Deskripsi tersebut misalnya koordinat titik. Dari koordinat titik tersebut terbentuklah citra berupa garis, kotak, segitiga, lingkaran, elips, dan lain-lain. *Input* dari program grafika komputer adalah deskripsi dan *output* berupa citra.
2. Pengolahan Citra *(Image Processing)* adalah cabang ilmu informatika untuk memperbaiki kualitas citra agar kualitasnya lebih baik atau lebih mudah diinterpretasi oleh manusia maupun komputer. *Input* dari program pengolahan citra adalah citra dan *output* berupa citra pula.

3. Pengenalan Pola *(Pattern Recognition/Image Interpretation)* adalah cabang ilmu yang mempelajari pola-pola data baik numerik maupun simbolik sehingga bisa diambil satu atau lebih kesimpulan. *Input* dari program pengenalan pola ini adalah citra dan *output* berupa deskripsi objek.

1.4 Komponen-Komponen Pembentuk Persepsi Visual

Pengolahan citra digital harus dipahami secara lebih dasar yaitu bahwa semuanya bermula dari sebuah konsep sederhana yaitu persepsi visual. Artinya bahwa terlihatnya sebuah citra oleh manusia adalah persepsi yang dibentuk oleh sistem penglihatan manusia. Secara lebih umum, persepsi visual ini bisa dimodelkan menjadi Sumber Cahaya – Objek – Sensor. Terdapat tiga komponen sehingga terbentuk sebuah persepsi terhadap sebuah tampilan, yaitu Sumber Cahaya – Objek – Sensor. Jika ada salah satu saja komponen tidak ada atau rusak, maka persepsi visual tidak akan terjadi atau terganggu. Misalnya saja jika sensor yang digunakan adalah mata manusia. Jika mata mengalami buta, maka manusia tidak dapat melihat walaupun benda dan sumber cahayanya ada. Begitu juga jika dalam ruangan yang gelap, maka tidak bisa melihat benda apa pun.

1. Sumber Cahaya sebagai salah satu yang memengaruhi terhadap persepsi visual juga dapat memberikan kesan yang berbeda apabila menggunakan sumber cahaya yang berbeda.
2. Objek yang dilihat manusia dapat memengaruhi terhadap persepsi visual. Misalnya objek yang dilihat adalah bunga matahari. Bunga matahari memiliki warna kuning karena kelopak pada bunga matahari menyerap gelombang cahaya kuning dari sinar matahari. Kemudian kelopak bunga matahari akan memantulkan gelombang cahaya kuning dan diterima oleh mata manusia. Begitu pula dengan lukisan, cat, mobil, warna mata, dan objek-objek lainnya.
3. Sensor yang merupakan hal terakhir dalam memengaruhi persepsi visual, sensor sebagai media untuk menangkap visualisasi terhadap objek yang dilihat akan memberikan persepsi yang berbeda tergantung

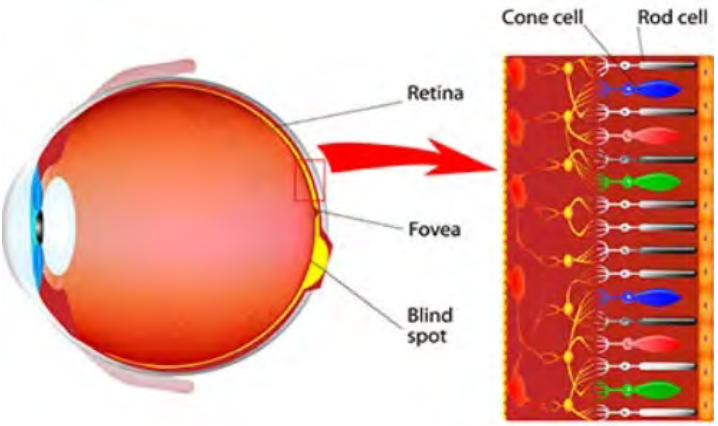
dengan sensor yang digunakan. Misalnya saja sensor yang digunakan adalah mata manusia, jika manusia melihat taman bunga dengan jenis bunga dan warna bunga yang beragam, tentunya kita akan melihat taman bunga yang memiliki warna yang bermacam-macam. Mata hewan ketika melihat suatu objek juga akan menampilkan persepsi yang berbeda (Lihat Gambar 1.4). Penggunaan sensor lain seperti sensor kamera akan memberikan kualitas gambar yang berbeda bergantung terhadap kualitas alat-alat di dalam kamera tersebut.

Dari ketiga komponen persepsi visual tadi diharapkan tumbuh kesadaran bahwa persepsi manusia terhadap objek yang ada di hadapannya tidak hanya bergantung pada objek tersebut, namun juga ada faktor lainnya yang memengaruhi.

Gambar 1.4. Perbedaan Persepsi Visual Mata Manusia dengan Mata Ular

1.5 Formasi Citra pada Manusia

Mata manusia sebagai sensor penglihatan utama bekerja secara sistematis yang disebut sebagai sistem penglihatan manusia. Cahaya yang dipantulkan oleh objek yang kita lihat akan ditangkap oleh mata manusia. Jika mata manusia bekerja secara normal, maka pantulan cahaya tersebut akan mengenai tepat pada retina. Pada retina terdapat dua reseptor yaitu *rods* (batang) dan *cones* (kerucut) (Lihat Gambar 1.5). *Rods* merupakan reseptor yang peka terhadap perbedaan intensitas cahaya (gelap/terang) sedangkan *cones* merupakan reseptor yang peka terhadap perbedaan warna. *Rods* beradaptasi lebih lambat dibandingkan *cones*. Itulah sebabnya ketika manusia memasuki ruangan yang minim penerangan, maka perlu waktu yang cukup lama bagi mata manusia untuk beradaptasi agar bisa melihat dengan jelas apa yang ada di sekitarnya.



Gambar 1.5. Sel Fotoreseptor di Mata Manusia

Masing-masing sel *cones* peka pada panjang gelombang tertentu. Kombinasi dari setiap tiga tipe sel *cones* tersebut diinterpretasikan sebagai sebuah warna tertentu oleh otak manusia. Sel *rod* mengukur jumlah energi yang diterima mata, yang berarti tingkat keabuan. Tipe L, M, dan S adalah kependekan dari *Long*, *Medium*, dan *Short* yang maksudnya adalah panjang gelombang cahaya.

Adanya reseptor yang berbeda di dalam retina ini juga menjelaskan kita akan fenomena buta warna. Buta warna parsial misalnya terjadi disebabkan ada masalah pada *cones* sehingga hanya sebagian saja yang aktif. Oleh karena itu, beberapa warna akan terlihat sama. Buta warna parsial mencakup buta warna merah dan hijau (protanopia dan deuteranopia) dan buta warna biru dan kuning (tritanopia).

1.6 Letak Persepsi Warna

Mata manusia memiliki kemampuan yang luar biasa sebagai sensor cahaya, namun sebenarnya persepsi terhadap cahaya tersebut terletak pada otak manusia dan bukan pada mata. Mata hanya memberikan data kepada otak tentang hasil penangkapannya, selanjutnya otaklah yang mengolah dan memaknai hasil penangkapan mata.

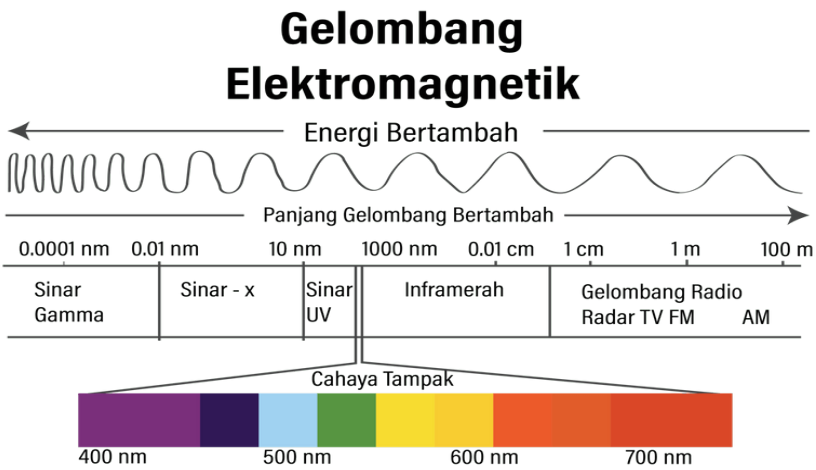
Selain itu kerusakan pada otak juga bisa berpengaruh terhadap cahaya yang ditangkap oleh mata. Ketika semisal terjadi kecelakaan pada kepala yang menyebabkan kebutaan padahal organ mata masih berfungsi dengan baik, maka sebenarnya saraf otak yang mengenali cahaya rusak sehingga walaupun mata bisa mengenali cahaya, namun otak akan mempersepsi bahwa tidak ada satupun cahaya yang masuk (gelap).

1.7 Cahaya dan Spektrum Elektromagnetik

Pada abad ke-18, Newton meneliti tentang cahaya. Dia melewatkan seberkas cahaya matahari ke dalam ruang gelap dan mengenakannya pada sebuah prisma. Ternyata cahaya itu terdifraksi dan terurai menjadi beberapa cahaya dengan warna yang berbeda yaitu merah, jingga, kuning, hijau, biru, nila, dan ungu (Lihat Gambar 1.6). Dari kondisi tersebut dapat disimpulkan bahwa cahaya putih ternyata merupakan gabungan dari berbagai berkas cahaya yang berbeda.

Gambar 1.6. Percobaan Newton terkait Cahaya

Cahaya sebenarnya merupakan gelombang elektromagnetik. Gelombang elektromagnetik memiliki berbagai bentuk seperti gelombang radio, inframerah, ultraviolet, dan lainnya. Cahaya adalah satu-satunya gelombang elektromagnetik yang dapat dilihat oleh mata. Jika diteliti lebih lanjut, maka cahaya hasil difraksi tersebut dapat diilustrasikan pada Gambar 1.7.



Gambar 1.7. Spektrum Gelombang Elektromagnetik (Sudut Pandang Spektrum Cahaya)

Gelombang elektromagnetik yang dapat dilihat oleh mata berwarna ungu sampai dengan merah dengan panjang gelombang diantara 380 – 780 nm, terdapat gelombang lain yang tidak dapat dilihat oleh mata manusia namun berguna untuk berbagai keperluan lain. Misalnya, sinar gamma digunakan untuk bidang astronomi yang dimanfaatkan untuk melihat objek-objek eksotis di luar angkasa, seperti supernova. Sinar-X dimanfaatkan untuk membuat foto *rontgen* yang sangat berguna untuk dunia kedokteran. Sinar ultraviolet dimanfaatkan untuk lampu penerangan dan pendeteksian uang palsu. Inframerah dimanfaatkan untuk *remote* dan kamera *Closed Circuit Television* (CCTV) di malam hari. Radar dimanfaatkan untuk bidang militer. Kemudian gelombang radio dan TV dimanfaatkan dalam bidang penyiaran.

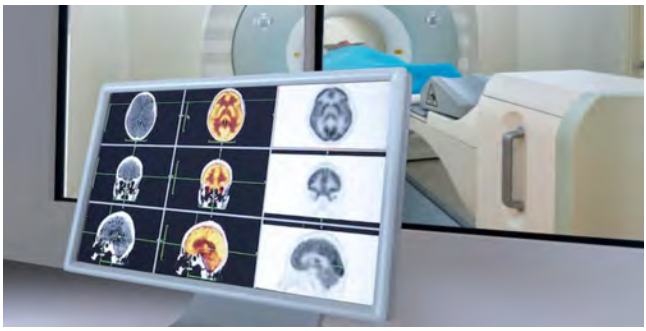
Beberapa prinsip yang perlu kita pahami adalah bahwa semakin kecil panjang gelombang maka semakin besar pula frekuensi yang artinya semakin besar energi yang dikandungnya. Sebagai contoh yaitu pada *laser pointer* berwarna hijau memiliki energi yang lebih besar dibandingkan dengan *laser pointer* berwarna merah. Oleh karena itu, *laser pointer* berwarna hijau dapat menempuh jarak yang lebih jauh dan memerlukan daya baterai yang lebih besar jika dibandingkan *laser pointer* berwarna merah.

Semakin besarnya frekuensi suatu gelombang maka akan berdampak besar terhadap tubuh manusia. Oleh karena itu tubuh manusia harus terlindungi ketika diterpa gelombang-gelombang energi tinggi seperti ultraviolet, sinar-X, sinar gamma, dan sinar kosmik. Jika terlalu sering terpapar maka akan berdampak pada kerusakan organ tubuh manusia. Misalnya saja kulit yang terlalu sering terkena sinar ultraviolet akan mengalami kerusakan. Itu juga yang menjadi alasan bahwa penggunaan sinar-X untuk membuat perekaman foto *rontgen* pada pasien secara beruntun dibatasi.

1.8 Penerapan Pengolahan Citra Digital di Berbagai Bidang

Pengolahan citra digital telah menjadi bagian penting dalam berbagai bidang di masa kini. Teknologi ini telah membawa dampak signifikan dalam berbagai industri dan memberikan manfaat yang sangat besar. Berikut adalah beberapa contoh penerapan pengolahan citra digital di beberapa bidang:

1. Bidang Kedokteran: Pengolahan citra digital digunakan dalam diagnosis medis, pemindaian tubuh, dan pemantauan pasien. Citra hasil pemindaian seperti *Computed Tomography (CT) Scan*, MRI *(Magnetic Resonance Imaging)* (Lihat Gambar 1.8), dan pemindaian *ultrasonography* dapat dianalisis untuk mendeteksi penyakit, tumor, dan kondisi medis lainnya. Pengolahan citra juga membantu dalam perencanaan perawatan dan intervensi bedah.



Gambar 1.8. Pemanfaatan pada Magnetic Resonance Imaging

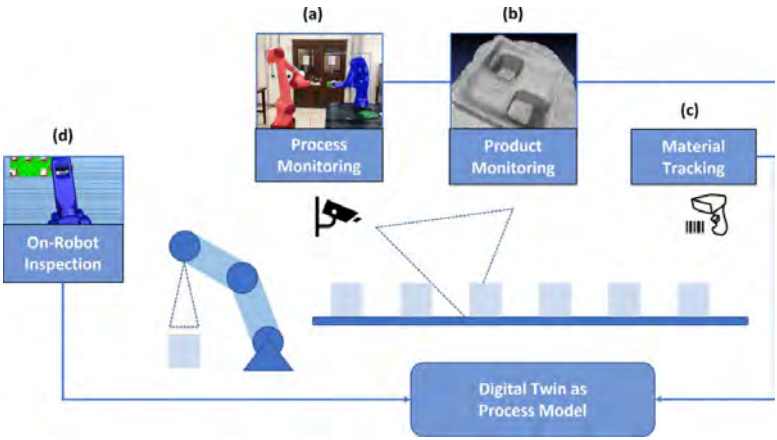
2. Bidang Keamanan: Pengolahan citra digunakan dalam sistem pengawasan dan keamanan, seperti pemantauan CCTV dan pengenalan wajah. Citra yang diperoleh dari kamera keamanan dapat dianalisis untuk mendeteksi kegiatan mencurigakan, mengidentifikasi orang atau objek tertentu, dan memberikan keamanan tambahan.

Gambar 1.9. Pemanfaatan *Security Detection* dengan CCTV

3. Bidang Otomotif: Pengolahan citra digital digunakan dalam pengenalan objek, pengolahan gambar kendaraan, dan penglihatan komputer dalam mobil otonom (*autonomous car*) (Lihat Gambar 1.10). Sistem pengolahan citra membantu mobil otonom dalam mendeteksi dan mengenali tanda lalu lintas, pejalan kaki, dan kendaraan lain di sekitarnya.

Gambar 1.10. Pemanfaatan pada *Autonomous Car*

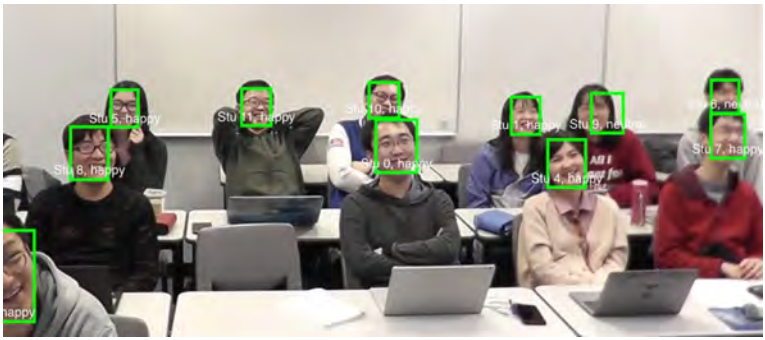
4. Bidang Manufaktur: Pengolahan citra digunakan dalam inspeksi kualitas produk, pengawasan proses produksi (Lihat Gambar 1.11), dan pengenalan pola. Citra produk yang diambil dapat dianalisis untuk mendeteksi cacat atau ketidaksesuaian dengan standar kualitas. Pengolahan citra juga dapat digunakan untuk mengontrol robot dalam proses produksi.



Gambar 1.11. Pemanfaatan pada Pengawasan Proses Produksi

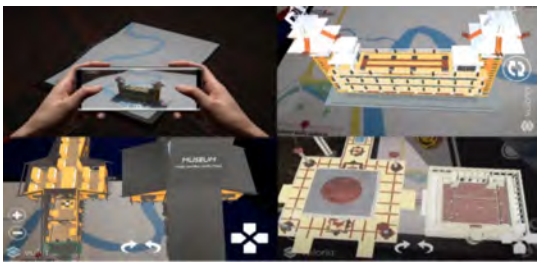
5. Bidang Perbankan dan Keuangan: Pengolahan citra digunakan dalam verifikasi identitas melalui pemindaian sidik jari dan pengenalan wajah. Citra wajah atau sidik jari dapat dianalisis dan dibandingkan dengan *database* untuk verifikasi keaslian identitas pelanggan. Pengolahan citra juga dapat digunakan dalam deteksi kecurangan dan tindakan yang mencurigakan dalam transaksi keuangan.
6. Bidang Arsitektur dan Desain: Pengolahan citra digunakan dalam visualisasi arsitektur, desain interior, dan *modeling 3D*. Citra digital dapat diubah menjadi representasi visual yang realistis dari bangunan atau desain interior yang membantu dalam perencanaan dan visualisasi proyek.
7. Bidang Pendidikan: Pengolahan citra digunakan dalam pendidikan untuk pengenalan karakter tulisan tangan, pemahaman teks, deteksi

plagiarisme, dan deteksi emosi siswa saat belajar (Lihat Gambar 1.12). Pengolahan citra juga dapat digunakan dalam *platform* pembelajaran *online* untuk memberikan umpan balik visual kepada siswa.



Gambar 1.12. Pemanfaatan pada Deteksi Emosi Siswa ketika Pembelajaran

8. Bidang Perjalanan dan Pariwisata: Dalam bidang perjalanan dan pariwisata, pengolahan citra digital dan visi komputer dapat memberikan manfaat yang signifikan. Berikut adalah beberapa contoh pemanfaatan pengolahan citra digital dan visi komputer dalam bidang perjalanan dan wisata:
   a. Pengenalan Objek Wisata: Dengan menggunakan teknik pengolahan citra dan visi komputer, dapat dilakukan pengenalan objek wisata seperti bangunan bersejarah, *landmark*, atau tempat-tempat populer. Ini dapat membantu pengunjung untuk mengidentifikasi dan mendapatkan informasi tentang tempat-tempat wisata yang mereka kunjungi.
   b. *Augmented Reality (AR)* dan *Virtual Reality (VR)*: Pengolahan citra dan visi komputer dapat digunakan untuk menciptakan pengalaman AR dan VR yang menarik dalam industri perjalanan dan pariwisata (Lihat Gambar 1.13). Dengan menggunakan teknologi ini, pengunjung dapat menggabungkan informasi digital dengan realitas fisik di sekitar mereka, atau merasakan pengalaman wisata virtual yang imersif.



Gambar 1.13. Aplikasi AR untuk Bidang Perjalanan dan Pariwisata (Syahidi dkk., 2022)

c. Alat penerjemah berbasis OCR *(Optical Character Recognition)*: dapat berguna dalam perjalanan dan pariwisata untuk membantu pengunjung dalam memahami teks dalam peta, tanda-tanda, atau panduan lokal yang mungkin tidak tersedia dalam bahasa yang mereka kuasai (Lihat Gambar 1.14).

Gambar 1.14. Pemanfaatan OCR dan AR sebagai Alat Penerjemah Bahasa Banjar (Syahidi dkk., 2018)

d. Deteksi Keramaian: Dalam bidang pariwisata, pengolahan citra dan visi komputer dapat digunakan untuk mendeteksi tingkat keramaian di tempat-tempat wisata. Hal ini dapat membantu dalam

pengelolaan dan perencanaan kunjungan wisata, serta memungkinkan pengunjung untuk memilih waktu yang lebih baik untuk menghindari kerumunan.

Ini hanya beberapa contoh penerapan pengolahan citra digital di berbagai bidang di masa kini. Dengan kemajuan teknologi dan perkembangan lebih lanjut, pengolahan citra digital memiliki potensi besar untuk digunakan dalam berbagai bidang lainnya.

1.9 Latihan

1. Jelaskan perbedaan antara citra analog dan citra digital yang mencakup pada sudut pandang representasi sinyal, akurasi, kemampuan manipulasi, penyimpanan, reproduksi, dan ketahanan terhadap degradasi!
2. Menurut Putra (2010), pengolahan citra dapat dibagi menjadi tiga kategori, sebutkan dan jelaskan!
3. Jelaskan, mengapa citra perlu diproses atau diolah lebih lanjut!
4. Sebutkan dan jelaskan masing-masing operasi-operasi umum yang ada di dalam pengolahan citra digital!
5. Menurut Anda, apa saja manfaat dari pengolahan citra digital?
6. Sebutkan dan jelaskan aplikasi-aplikasi yang pernah Anda gunakan atau ketahui, dimana mengadopsi teknologi pengolahan citra digital!

BAB 2

PEMROGRAMAN PYTHON PADA PENGOLAHAN CITRA DIGITAL

Capaian Pembelajaran:

1. Mampu mengetahui dan memahami bahasa pemrograman Python untuk pengolahan citra digital.
2. Mampu mengetahui dan memahami jenis-jenis pustaka Python dalam pengolahan citra digital.
3. Mampu mengetahui dan melakukan pemasangan (*install*) kebutuhan program Python untuk pengolahan citra digital.

2.1 Bahasa Pemrograman Python untuk Pengolahan Citra Digital

Pengolahan citra digital melibatkan proses dalam mengolah citra digital menggunakan komputer. Ini merupakan aspek yang sangat penting dalam berbagai aplikasi, termasuk pengolahan medis, pengenalan pola, pengolahan citra satelit, dan bidang lainnya. Salah satu alat yang umum dan kekinian digunakan dalam pengolahan citra digital adalah bahasa pemrograman Python.

Python merupakan bahasa pemrograman yang sangat terkenal dan banyak digunakan dalam pengolahan citra digital di masa kini. Python menyediakan berbagai pustaka yang dapat digunakan untuk memproses citra digital. Dimana pustaka tersebut menyediakan berbagai fungsi dan metode yang memungkinkan pengolahan citra digital, termasuk deteksi tepi, segmentasi, pengolahan morfologi, dan lain-lain sampai pada tahap lanjutan yang berkaitan dengan visi komputer *(computer vision).*

Salah satu penerapan pengolahan citra digital menggunakan bahasa pemrograman Python adalah deteksi wajah. Untuk melakukan deteksi wajah, dapat digunakan pustaka OpenCV yang menyediakan berbagai fungsi khusus untuk tujuan tersebut. Proses deteksi wajah melibatkan

beberapa tahap, seperti identifikasi wajah awal, penyesuaian ukuran wajah, dan pengenalan pola. Setelah berhasil mendeteksi wajah, berbagai proses lanjutan dapat dilakukan, seperti pengenalan ekspresi wajah, jenis kelamin, dan lain sebagainya.

Selain deteksi wajah, pengolahan citra digital menggunakan Python juga dapat digunakan untuk segmentasi citra. Segmentasi citra adalah proses membagi citra menjadi beberapa bagian atau wilayah yang berbeda. Untuk melakukan segmentasi citra, pustaka Scikit-Image menyediakan berbagai fungsi, seperti segmentasi warna, segmentasi tekstur, dan sejenisnya. Segmentasi citra memiliki peran penting dalam berbagai aplikasi, seperti pengolahan medis dan pengolahan citra satelit.

Dalam hal akurasi dan kecepatan pengolahan citra digital, Python memiliki keunggulan tersendiri. Python menggunakan *compiler just-in-time* (JIT) yang dapat secara signifikan meningkatkan kinerja pengolahan citra digital. Selain itu, Python juga mendukung paralelisasi dan distribusi, sehingga memungkinkan pengolahan citra digital dilakukan dengan menggunakan beberapa komputer secara bersamaan.

Dalam hal pembelajaran dan penggunaan, Python juga sangat mudah dipelajari dan digunakan oleh para pengembang dan mahasiswa. Python memiliki sintaks yang sederhana dan mudah dipahami, serta terdapat banyak dokumentasi dan tutorial *online* yang mudah diakses.

Bahasa pemrograman Python memiliki beberapa keunggulan yang membuatnya cocok untuk pengolahan citra digital diantaranya:

1. Kemudahan Penggunaan: Python memiliki sintaks yang sederhana dan mudah dipahami, sehingga lebih mudah bagi pemula untuk mempelajarinya. Selain itu, Python juga memiliki banyak pustaka dan modul yang telah dikembangkan dengan baik untuk pengolahan citra digital, seperti OpenCV, Pillow, dan Scikit-Image, yang memudahkan pengembang dalam memanipulasi dan memproses citra.
2. Ketersediaan Pustaka dan Ekosistem yang Kaya: Python menyediakan berbagai pustaka dan modul yang dioptimalkan khusus untuk pengolahan citra digital. Contohnya, OpenCV adalah salah satu

pustaka populer yang memiliki fungsi-fungsi lengkap untuk pemrosesan dan analisis citra. Pustaka lain seperti Pillow, Scikit-Image, dan NumPy juga menyediakan banyak fungsi dan metode yang berguna dalam pengolahan citra digital.

3. Performa dan Kecepatan: Meskipun Python merupakan bahasa pemrograman yang interpretatif, Python memiliki kemampuan untuk meningkatkan performa pengolahan citra digital dengan bantuan JIT seperti PyPy (interpreter bahasa pemrograman Python yang ditulis dengan Python dan dilengkapi JIT) atau menggunakan pustaka-pustaka yang ditulis dalam bahasa pemrograman C/C++ di bawahnya. Selain itu, penggunaan paralelisasi dan distribusi dalam Python memungkinkan pengolahan citra digital dilakukan secara efisien menggunakan beberapa komputer secara bersamaan.
4. Komunitas yang Aktif: Python memiliki komunitas pengembang yang besar dan aktif, terutama dalam bidang pengolahan citra digital. Dukungan dan kontribusi dari komunitas ini menyediakan berbagai sumber daya, seperti dokumentasi, tutorial, dan forum diskusi, yang memudahkan pengembang dalam belajar dan mengatasi masalah dalam pengolahan citra digital.
5. *Multiplatform* dan Keterjangkauan: Python dapat digunakan di berbagai *platform* seperti Windows, macOS, dan Linux. Selain itu, Python bersifat *open-source*, sehingga dapat diakses secara gratis dan digunakan untuk pengolahan citra digital tanpa biaya lisensi yang tinggi.

Dengan kombinasi keunggulan-keunggulan yang telah diketahui ini dan ditawarkan, Python telah menjadi salah satu bahasa pemrograman yang populer dan sering digunakan dalam pengolahan citra digital di masa kini.

Berikut adalah beberapa perintah-perintah dasar yang umum digunakan dalam pemrograman Python untuk pengolahan citra digital:

1. **import** adalah fungsi Python untuk memanggil modul di luar fungsi dasar yang disediakan.

2. **cv2** adalah nama modul Python yang digunakan untuk memanggil fungsi-fungsi OpenCV.
3. **cv2.imread** perintah yang digunakan pada modul cv2 untuk memanggil citra. **imread** merupakan singkatan dari *image read* yaitu perintah untuk membaca citra pada tempat penyimpanan citra.
4. **cv2.IMREAD_COLOR** perintah untuk memuat gambar berwarna. Setiap transparasi gambar akan terbengkalai.
5. **cv2.IMREAD_GRAYSCALE** perintah untuk memuat gambar dalam mode skala keabuan.
6. **cv2.IMREAD_UNCHANGED** perintah untuk memuat gambar dalam mode saluran alfa.
7. **cv2_imshow** perintah yang digunakan pada modul cv2 untuk menampilkan citra, dimana sebuah jendela akan terbuka secara otomatis akan untuk menampilkan citra dan kemudian menyesuaikan dengan ukuran dari citra tersebut. **imshow** merupakan singkatan dari *image show*.
8. **cv2.waitkey()** merupakan fungsi pengikat *keyboard* maksudnya adalah waktu dalam milidetik. Jika kita menekan sembarang tombol di *keyboard* sebuah program akan terus berjalan. Fungsi ini mengembalikan sebuah bilangan bulat yang merupakan kode kunci (*key code*) dimana menunggu *input keyboard* dalam aplikasi pemrosesan citra yang akan ditekan oleh pengguna.
9. **cv2.destroyAllWindows()** merupakan fungsi untuk menghancurkan atau menutup jendela/kotak dialog dari tampilan citra atau program.
10. **cv2.imwrite()** untuk menyimpan gambar bergantung pada format yang ditujukan.

2.2 Jenis-Jenis Pustaka Python untuk Pengolahan Citra Digital

Pustaka *(Library)* pada Python ini sendiri merupakan kumpulan modul terkait berisi kumpulan kode yang dapat digunakan berulang kali dalam program yang berbeda. Pustaka ini membuat pemrograman Python lebih sederhana dan nyaman bagi *programmer*. Berikut merupakan beberapa

pustaka Python yang umum digunakan dalam pengolahan citra digital dan visi komputer, beserta penjelasan kegunaan dan contoh kode programnya yaitu sebagai berikut:

1. OpenCV *(Open-Source Computer Vision Library)* adalah pustaka yang paling populer untuk pengolahan citra digital dan visi komputer yang dikembangkan pada tahun 2000. Pustaka ini menyediakan berbagai fungsi untuk memproses citra dan melakukan tugas-tugas seperti deteksi objek, deteksi wajah, segmentasi citra, pengenalan pola, dan banyak lagi. OpenCV (https://opencv.org/) juga mendukung pengolahan video dan memiliki algoritme yang dioptimalkan untuk kecepatan pemrosesan yang tinggi. Berikut contoh penggunaan pustaka ini pada kode program:

```
import cv2

filepath = 'data\\akhrian.jpg'
image = cv2.imread(filepath)
```

2. Pillow/PIL *(Python Imaging Library)* merupakan pustaka yang digunakan untuk memanipulasi gambar dalam Python. Pustaka ini menyediakan berbagai fungsi untuk membuka, menyimpan, dan mengubah ukuran gambar. Selain itu, Pillow juga mendukung operasi dasar seperti pemutaran, pergeseran, dan transformasi gambar. Berikut contoh penggunaan pustaka ini pada kode program:

```
from PIL import Image
with Image.open("akhrian.jpg") as im;

im.rotate(60).show()
```

Fungsi:

```
PIL.Image.open(fp, mode='r', formats=None)
```

Fungsi untuk membuka dan mengidentifikasi *file* gambar yang diberikan. Fungsi ini mengidentifikasi *file*, namun *file* tetap terbuka dan data gambar sebetulnya tidak dibaca dari *file* sampai pengguna mencoba memproses data atau memanggil **load()**.

3. NumPy *(Numerical Python)* adalah pustaka yang digunakan untuk melakukan manipulasi dan analisis data numerik (komputasi ilmiah), termasuk data citra. Pustaka ini menyediakan struktur data array multidimensi yang efisien dan berbagai fungsi matematika yang kuat, dimana citra pada dasarnya merupakan array nilai piksel yang mana setiap piksel diwakili oleh nilai 1 *(Grayscale)* atau 3 *(Red-Green-Blue)*. NumPy sangat berguna dalam pengolahan citra digital, terutama untuk operasi matematika seperti konvolusi, *filtering*, dan transformasi Fourier selain itu juga melakukan tugas untuk pemotongan gambar, penyembunyian, atau memanipulasi nilai piksel. Berikut contoh penggunaan pustaka ini pada kode program:

```
import numpy as np
import cv2

image = cv2.imread('akhrian.jpg')
```

Contoh pada himpunan array ketika proses *filter sharpening* pada konvolusi:

```
kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]])
sharpened_img = np.convolve(img_array, kernel, mode='same')
```

4. SciPy *(Scientific Python)* adalah pustaka yang memperluas kemampuan NumPy dengan menambahkan fungsi-fungsi ilmiah dan teknis. Pustaka ini menyediakan algoritme-algoritme untuk pengolahan citra seperti segmentasi, analisis tekstur, pengenalan pola, ekstraksi fitur, dan sebagainya, biasanya menggunakan *submodule* **scipy.ndimage**, fungsi tersebut menyediakan fungsi untuk beroperasi pada array Numpy n-dimensi. SciPy juga memiliki pustaka turunan seperti scikit-image dan scikit-learn yang lebih spesifik untuk pengolahan citra dan pembelajaran mesin *(machine learning)*. Berikut contoh penggunaan pustaka ini pada kode program:

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

```
image = plt.imread('akhrian.jpg')
```

Contoh lainnya untuk pengolahan citra multidimensi:

```
from scipy import misc,ndimage
from matplotlib import pyplot as plt

face = misc.face()
blurred_face = ndimage.gaussian_filter(face, sigma=3)
```

5. Scikit-Image adalah pustaka khusus untuk pengolahan citra digital yang memiliki beberapa bagian yang ditulis dalam Cython (Cython ialah bahasa pemrograman yang merupakan superset dari bahasa pemrograman Python yang dirancang untuk memiliki kinerja seperti bahasa pemrograman C) dalam rangka untuk mencapai kinerja yang baik. Pustaka ini menyediakan berbagai fungsi untuk operasi dasar seperti filtrasi, segmentasi, transformasi geometris, manipulasi ruang warna, pemrosesan dan analisis citra lanjutan. Scikit-Image juga memiliki algoritme-algoritme yang dioptimalkan untuk performa tinggi dalam pengolahan citra diantaranya *segmentation, morphology*, dan lainnya. Scikit-Image menggunakan array Numpy sebagai objek gambar. Berikut contoh penggunaan pustaka ini pada kode program:

```
from skimage import io, filters

image = io.imread('akhrian.jpg')
```

Selain itu, contoh berikut adalah untuk melakukan *filter* citra dengan pustaka ini:

```
from skimage import data, io, filters
image = data.coins()

edges = filters.sobel(image)
io.imshow(edges)
io.show()
```

6. Dlib adalah pustaka perangkat lunak yang populer untuk pengolahan citra digital dan visi komputer dimana berfokus pada pengenalan objek dan deteksi wajah. Pustaka ini ditulis dalam bahasa pemrograman C++

dan menyediakan antarmuka untuk bahasa pemrograman Python. Dlib menyediakan berbagai algoritme canggih dalam bidang pengolahan citra dan visi komputer, termasuk deteksi objek, deteksi wajah, deteksi *landmark*, pengenalan wajah, dan pelacakan objek.

Berikut adalah beberapa fitur utama dari pustaka Dlib dalam konteks pengolahan citra digital dan visi komputer:

a. Deteksi Wajah: Dlib menyediakan algoritme deteksi wajah yang sangat akurat dan cepat. Algoritme ini menggunakan metode deteksi wajah berbasis fitur yang dikenal sebagai *Histogram of Oriented Gradients* (HOG). Deteksi wajah dengan Dlib juga termasuk kemampuan untuk mengidentifikasi kotak pembatas *(bounding box)* wajah yang tepat.
b. Deteksi *Landmark*: Dlib memiliki kemampuan untuk mendeteksi *landmark* wajah, yaitu titik-titik penting seperti mata, hidung, mulut, dan sebagainya. Algoritme deteksi *landmark* yang disediakan oleh Dlib menggunakan *shape prediction models* yang dapat menghasilkan koordinat *landmark* dengan presisi tinggi.
c. Pelacakan Objek: Dlib juga menyediakan algoritme pelacakan objek *(object tracking)* yang kuat. Algoritme pelacakan ini dapat digunakan untuk melacak objek yang bergerak dalam sebuah video atau rangkaian citra. Pelacakan objek dengan Dlib menggunakan pendekatan *correlation filters* yang efisien dan akurat.
d. Pengenalan Wajah: Dlib memiliki modul pengenalan wajah *(face recognition)* yang dapat digunakan untuk mengenali identitas wajah dari sekumpulan data pelatihan. Pustaka ini menggunakan *deep learning* untuk melakukan pengenalan wajah dengan kualitas yang tinggi.
e. Pengolahan Citra dan Fitur Lainnya: Selain fitur-fitur utama yang telah disebutkan sebelumnya, Dlib juga menyediakan berbagai algoritma pengolahan citra dan fitur lainnya seperti ekstraksi fitur

dengan *Convolutional Neural Networks* (CNN), segmentasi citra, deteksi tepi, registrasi citra, dan banyak lagi.

Dlib adalah pustaka yang sangat populer dalam komunitas pengolahan citra dan visi komputer karena keakuratannya, kecepatannya, dan keandalannya. Pustaka ini memiliki dokumentasi yang kaya dan mendalam, serta dukungan yang kuat dari komunitas pengguna yang aktif. Dlib juga menyediakan model pre-trained yang siap digunakan untuk melaksanakan tugas-tugasnya. Berikut contoh penggunaan pustaka ini pada kode program:

```
import dlib
import cv2

image = cv2.imread('gambar.jpg')
detector = dlib.get_frontal_face_detector()
faces = detector(image)
```

7. Matplotlib: merupakan pustaka Python yang digunakan untuk visualisasi data. Pustaka ini menyediakan berbagai fungsi dan alat untuk membuat grafik, plot, diagram, dan visualisasi lainnya dalam bahasa pemrograman Python. Matplotlib sering digunakan bersama dengan pustaka lain seperti NumPy dan Pandas untuk menganalisis dan menggambarkan data secara interaktif. Dengan Matplotlib, Anda dapat membuat berbagai jenis visualisasi seperti grafik garis, *scatter plot*, histogram, diagram batang, dan banyak lagi. Berikut contoh penggunaan pustaka ini pada kode program:

```
import cv2
import matplotlib.pyplot as plt

image = cv2.imread('nama_citra.jpg', 0)  # Mode grayscale
histogram = cv2.calcHist([image], [0], None, [256], [0, 256])
```

8. TensorFlow dan Keras adalah pustaka yang digunakan untuk pembelajaran mesin dan *deep learning*. Meskipun mereka tidak khusus untuk pengolahan citra, pustaka-pustaka ini dapat digunakan dalam

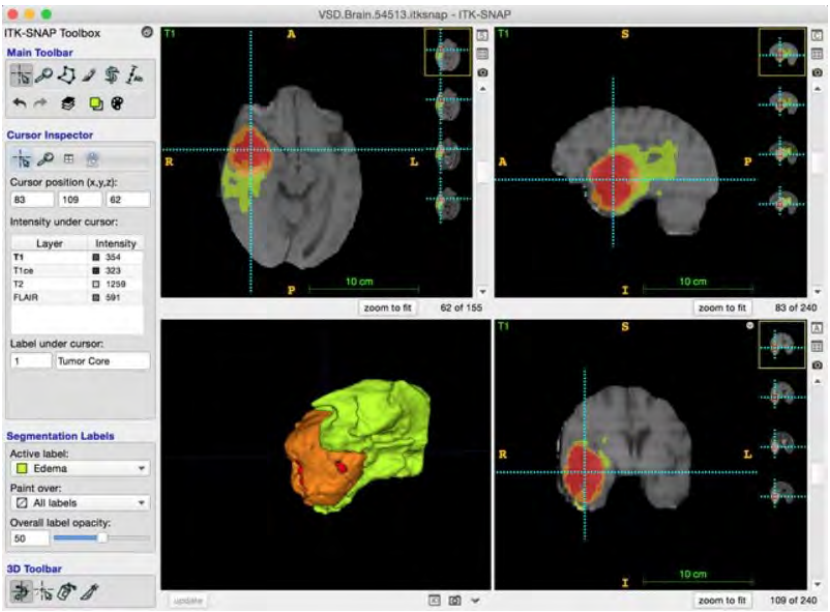
visi komputer untuk mengembangkan model pengenalan objek, segmentasi, dan klasifikasi citra yang canggih. Berikut contoh penggunaan pustaka ini pada kode program:

```
import tensorflow as tf
from tensorflow.keras.applications import EfficientNetB0
from tensorflow.keras.applications.efficientnet import
preprocess_input
from tensorflow.keras.preprocessing import image
import numpy as np

model = EfficientNetB0(weights='imagenet')
```

9. Mahotas: Mahotas merupakan pustaka pengolahan citra dan visi komputer lain yang dirancang untuk informatika *bioimage*. Pustaka ini membaca dan menulis citra dalam array NumPy, dan diimplementasikan dalam C++ dengan antarmuka Python. Pustaka ini juga dapat digunakan untuk berbagai tujuan seperti segmentasi, ekstraksi fitur, klasifikasi, analisis bentuk, dan banyak lagi. Mahotas juga diintegrasikan dengan pustaka-pustaka lain seperti NumPy dan SciPy, sehingga memungkinkan integrasi yang mudah dengan alat dan fungsi ilmiah lainnya dalam ekosistem Python. Berikut adalah fungsi Mahotas yang populer yaitu: *Watershed, Convex Points Calculations, Hit-and-Miss Transform, Local Binary Patterns, Morphological Processing, Speeded-Up Robust Features (SURF), Thresholding, Convolutional, Sobel Edge Detection*, dan lainnya. Berikut contoh penggunaan pustaka ini pada kode program untuk kasus *image distance*:

```
import mahotas
import numpy as np

image1 = mahotas.imread_grey('akhrian.png')
image2 = mahotas.imread_grey('auliaakhrianvisajapan.jpg')

distance = mahotas.bhattacharyya_dist(image1.ravel(),
image2.ravel())
```

10. SimpleITK *(Insight Segmentation and Registration Toolkit)*: sebuah pustaka yang digunakan untuk pemrosesan citra medis dan analisis citra. SimpleITK menyediakan berbagai algoritme untuk segmentasi, registrasi, rekonstruksi 3D, dan analisis citra medis lainnya. Pustaka ini dikembangkan oleh *National Library of Medicine* (NLM) dan digunakan secara luas dalam komunitas ilmu kedokteran dan penelitian medis. SimpleITK adalah sebuah pustaka yang menyediakan antarmuka Python. Pustaka SimpleITK memungkinkan pengguna untuk menggunakan fungsionalitas dengan mudah dan lebih familiar dalam lingkungan pemrograman Python. SimpleITK memungkinkan manipulasi citra medis, pemrosesan, dan analisis melalui sintaks Python yang lebih sederhana.



Gambar 2.1. *Segmentation of Multi-modality Medical Imaging Datasets with ITK-SNAP*

Dengan SimpleITK, Anda dapat membaca dan menulis citra medis dalam berbagai format, melakukan operasi pemrosesan citra seperti

filtrasi, segmentasi, registrasi, dan ekstraksi fitur. Selain itu, SimpleITK juga menyediakan alat visualisasi dan utilitas yang berguna dalam pemrosesan citra medis. Berikut contoh penggunaan pustaka ini pada kode program:

```
import SimpleITK as sitk

image = sitk.ReadImage('path/to/image.dcm')
segmenter = sitk.ConnectedThresholdImageFilter()
```

Pada Gambar 2.1 merupakan contoh hasil penggunaan SimpleITK pada segmentasi pencitraan medis yang bersumber dari Yushkevich dkk. (2019):

11. Pandas: merupakan pustaka Python yang digunakan untuk analisis data dan manipulasi data tabular. Meskipun Pandas tidak secara khusus digunakan untuk pengolahan citra digital, Anda dapat menggunakan Pandas untuk membaca metadata citra, mengelola data yang terkait dengan citra, atau menyimpan data hasil pengolahan citra. Berikut contoh penggunaan pustaka ini pada kode program:

```
import cv2
import pandas as pd

image = cv2.imread('akhrian.jpg')
height, width, channels = image.shape
metadata = pd.DataFrame({'Lebar': [width], 'Tinggi': [height], 'Channel': [channels]})
```

12. Pgmagick: pustaka Python yang digunakan untuk memanipulasi dan mengolah gambar menggunakan fungsi-fungsi dari pustaka ImageMagick. Pgmagick menyediakan antarmuka Python untuk ImageMagick, yang merupakan perangkat lunak sumber terbuka yang kuat untuk mengedit, mengubah ukuran, dan memanipulasi gambar. Dengan Pgmagick, Anda dapat membaca, menulis, mengubah ukuran, memutar, mengubah format, dan melakukan berbagai operasi pemrosesan citra lainnya menggunakan ImageMagick. Pgmagick menyediakan antarmuka yang mudah digunakan dalam bahasa Python

untuk mengakses fungsi-fungsi ImageMagick. Berikut contoh penggunaan pustaka ini pada kode program:

```
from pgmagick import Image

img = Image("input.jpg")
img.resize("400x300")
img.rotate(90)
img.write("output.jpg")
```

2.3 Kebutuhan dan Proses *Install* Program Python untuk Pengolahan Citra Digital

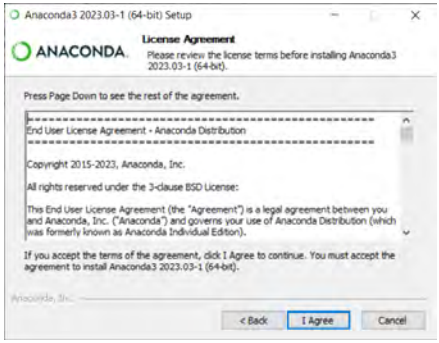
Pada buku ini, kami akan memfokuskan penggunaan perangkat lunak Sublime Text dan Anaconda Prompt. SublimeText (https://www.sublimetext.com/3) pada pengolahan citra digital ini digunakan sebagai *text editor* dalam melakukan pengkodean program dikarenakan sangat mudah dan juga efisien. Namun, penting untuk dicatat bahwa pemilihan *text editor* adalah preferensi pribadi. Selain Sublime Text, ada banyak *text editor* lainnya yang juga populer dalam pengembangan pengolahan citra digital, seperti Visual Studio Code, Atom, atau PyCharm.

Kemudian Anaconda melalui Anaconda Prompt (https://www.anaconda.com/download#windows) digunakan untuk melakukan pemanggilan direktori program, instalasi pustaka, mengelola paket-paket, memastikan kesesuaian versi yang diperlukan terkait pustaka dan paket, dan juga untuk menjalankan program pengolahan citra digital. Dalam rangka pengolahan citra digital, Anaconda Prompt menawarkan lingkungan pengembangan yang kuat dan fleksibel dengan dukungan manajemen paket yang baik. Silakan dalam melakukan praktikum ini untuk memasang perangkat lunak Sublime Text dan Anaconda Prompt dengan versi terbarunya.

Berikut adalah tahapan instalasi perangkat lunak Anaconda (Lihat Gambar 2.2 sampai dengan 2.6). Tahap pertama, silakan untuk membuka *file* Anaconda Installer, tunggu sejenak untuk proses *loading*, kemudian

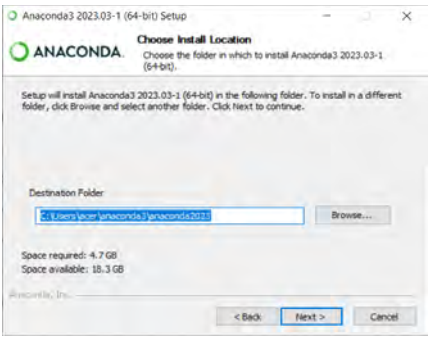
muncul kotak dialog yang terlihat pada Gambar 2.2., silakan untuk menekan tombol **Next**.

Gambar 2.2. Kotak Dialog Awal

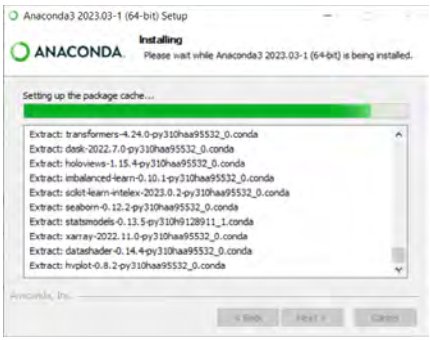


Gambar 2.3. Persetujuan Lisensi

Kemudian muncul kotak dialog persetujuan lisensi (Lihat Gambar 2.3), silakan untuk menyetujuinya dengan menekan tombol **I Agree**. Selanjutnya, akan muncul kotak dialog pemilihan direktori untuk menentukan letak program Anaconda yang akan dipasang pada sistem operasi kita, silakan pilih *destination folder* yang sesuai kemudian tekan tombol **Next** (Lihat Gambar 2.4).



Gambar 2.4. Pemilihan Direktori

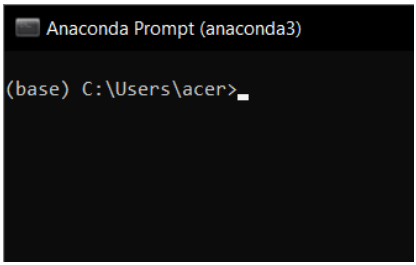


Gambar 2.5. Proses Instalasi

Pada Gambar 2.5, merupakan proses instalasi Anaconda, tunggu beberapa saat hingga selesai. Jika instalasi berhasil, kemudian akan muncul kotak dialog yang dapat dilihat pada Gambar 2.6, kemudian tekan tombol **Finish**. Proses instalasi Anaconda telah selesai dan dapat digunakan, silakan temukan program Anaconda Prompt pada bagian **Search Box** pada

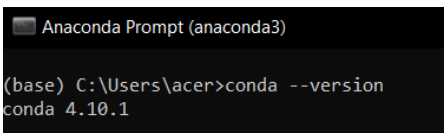
Windows dengan mengetikkan kata Anaconda dan kemudian kita tekan untuk menjalankannya, hasilnya dapat dilihat pada Gambar 2.7.

Gambar 2.6. Proses Instalasi Selesai

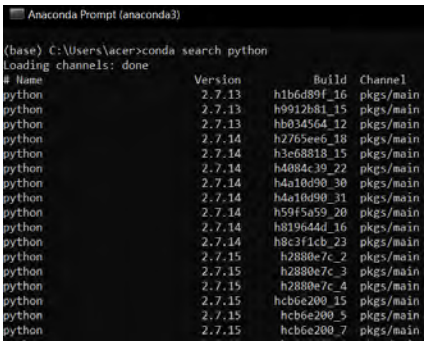


Gambar 2.7. Anaconda Prompt

Selanjutnya kita akan melakukan pengecekan pada Anaconda Prompt terkait Conda dan Python, apakah sudah terpasang dengan baik sesuai versinya. Cara melakukan pengecekan Conda dengan mengetikkan perintah **conda --version** (Lihat Gambar 2.8) kemudian tekan **Enter**, jika telah berhasil terpasang maka akan menghasilkan informasi versi dari Conda.



Gambar 2.8. Pengecekan Conda

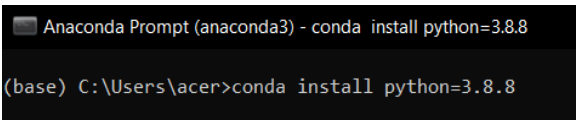


Gambar 2.9. Pengecekan Versi Python

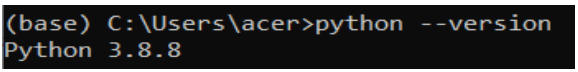
Selanjutnya, kita akan melakukan pengecekan untuk Python, sebelumnya kita lakukan pengecekan versi yang tersedia dari Python dengan perintah **conda search python** (Lihat Gambar 2.9) dengan syarat

harus terkoneksi dengan internet kemudian tekan **Enter**, maka akan muncul beberapa versi Python yang dapat kita pilih dengan melihat daftar versi yang tersedia sampai ke bagian akhir untuk versi terbarunya yang tersedia. Kami merekomendasikan untuk menggunakan Python versi 3.8 dan 3.9 agar tidak mengalami masalah.

Untuk melakukan instalasi Python, ketikan perintah **conda install python=3.8.8** (Lihat Gambar 2.10) dengan syarat harus terkoneksi dengan internet kemudian tekan **Enter**, tunggu beberapa saat untuk proses instalasi sampai selesai. Selanjutnya kita pastikan hasil instalasi berhasil dengan mengetikkan perintah **python --version** (Lihat Gambar 2.11) kemudian tekan **Enter** dan akan menghasilkan informasi versi Python yang berhasil dipasang tersebut.



Gambar 2.10. Proses Instalasi Versi Python



Gambar 2.11. Pengecekan Hasil Instalasi Python

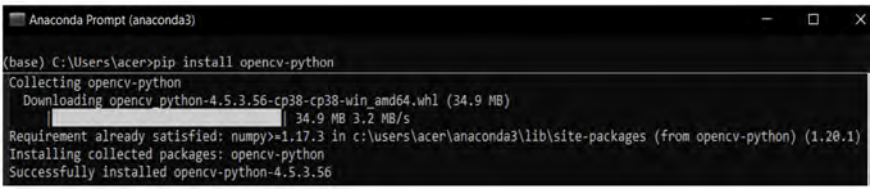
Kebutuhan selanjutnya adalah kita akan membuat direktori penyimpanan (*storage directory*). Direktori penyimpanan adalah lokasi dalam sistem *file* komputer di mana data, berkas, atau informasi dapat disimpan. Ini adalah folder atau direktori dimana *file-file* tersebut ditempatkan dan diorganisir. Dalam konteks pengolahan citra digital, direktori penyimpanan dapat digunakan untuk menyimpan gambar atau hasil pengolahan citra. Misalnya, Anda dapat memiliki direktori khusus di mana semua gambar yang akan diproses atau hasil pengolahan citra disimpan. Direktori penyimpanan biasanya dapat dipilih oleh pengguna atau ditentukan dalam kode program. Ketika menyimpan atau membuka *file*, Anda perlu memberikan jalur atau alamat *file* yang menunjukkan direktori penyimpanan yang diinginkan. Dengan cara ini, Anda dapat mengorganisir dan mengakses *file-file* tersebut dengan mudah.

Kami menyarankan pada praktikum ini untuk membuat direktori seperti pada Gambar 2.12 berikut dengan tatanan (C:\PCD\Praktik):

Gambar 2.12. Kebutuhan Direktori Penyimpanan

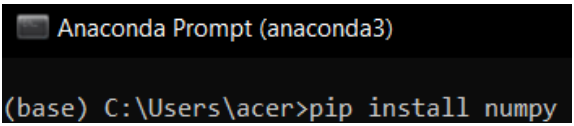
Folder **PCD** berisikan subfolder bernama **Praktik** yang digunakan untuk menyimpan *file* program Python dan keluaran dari citra yang tersimpan, selanjutnya di dalam subfolder Praktik terdapat subfolder selanjutnya yaitu **dataset** yang digunakan untuk menampung *dataset* citra maupun *dataset* lainnya yang berhubungan dengan data latih ataupun data uji serta juga untuk keluaran dari citra yang tersimpan.

Kebutuhan berikutnya, kita perlu melakukan instalasi pustaka umum yang sering digunakan diantaranya OpenCV, NumPy, dan Matplotlib, dikarenakan pustaka tersebut merupakan yang utama dan akan terus kita gunakan dengan menyesuaikan keperluan. Pertama kita akan menginstalasi pustaka OpenCV, silakan untuk mengetikkan perintah **pip install opencv-python** (Lihat Gambar 2.13) dengan syarat harus terkoneksi dengan internet kemudian tekan **Enter**, tunggu beberapa saat untuk proses instalasi sampai selesai dengan hasil adanya notifikasi sukses.



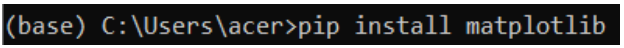
Gambar 2.13. Proses Instalasi Pustaka OpenCV

Selanjutnya, menginstalasi pustaka NumPy, silakan untuk mengetikkan perintah **pip install numpy** (Lihat Gambar 2.14) dengan syarat harus terkoneksi dengan internet kemudian tekan **Enter**, tunggu beberapa saat untuk proses instalasi sampai selesai.



Gambar 2.14. Proses Instalasi Pustaka NumPy

Terakhir, menginstalasi pustaka Matplotlib, silakan untuk mengetikkan perintah **pip install matplotlib** (Lihat Gambar 2.15) dengan syarat harus terkoneksi dengan internet kemudian tekan **Enter**, tunggu beberapa saat untuk proses instalasi sampai selesai.



Gambar 2.15. Proses Instalasi Pustaka Matplotlib

2.4 Latihan

1. Sebutkan keunggulan pemrograman Python yang dimanfaatkan untuk pengolahan citra digital!
2. Sebutkan dan ringkaslah masing-masing kegunaan pustaka yang dimiliki oleh Python untuk pengolahan citra digital!
3. Jelaskan kegunaan direktori penyimpanan dalam pengolahan citra digital!

BAB 3

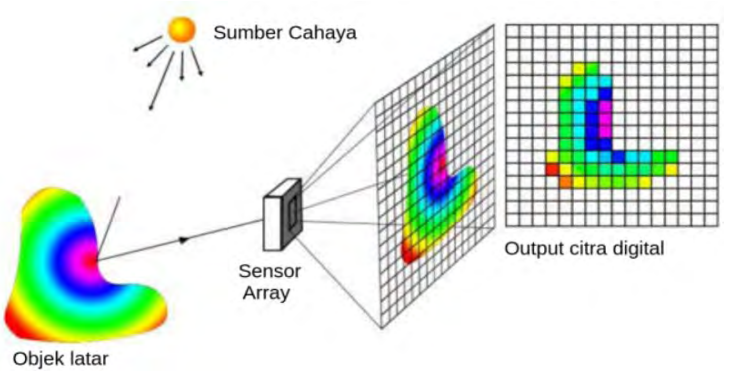
AKUISISI, JENIS, FORMAT FILE, DAN HUBUNGAN ANTARPIKSEL PADA CITRA

Capaian Pembelajaran:

1. Mampu mengetahui, memahami, dan mempraktikkan akuisisi citra.
2. Mampu mengetahui, memahami, dan mempraktikkan jenis warna pada citra.
3. Mampu mengetahui, memahami, dan mempraktikkan format *file* yang tersedia pada citra.
4. Mampu mengetahui, memahami, dan mempraktikkan keterhubungan antarpiksel pada citra.

3.1 Akuisisi Citra

Akuisisi citra *(image acquisition)* adalah proses untuk mengambil, mendapatkan, atau proses pencuplikan citra dari sumbernya, seperti kamera, *scanner*, atau sensor lainnya. Dalam konteks pengolahan citra digital, akuisisi citra merupakan langkah awal dalam memperoleh data citra yang akan diproses, pada Gambar 3.1 merupakan ilustrasi dari akuisisi citra dengan sensor array.



Gambar 3.1. Akuisisi Citra dengan Sensor Array

Berdasarkan Gambar 3.1, sensor array dapat menghasilkan gambar yang jelas dan detail dengan waktu akuisisi yang cepat karena setiap piksel dalam array dapat bekerja secara paralel untuk mendeteksi cahaya atau sinyal. Ini membuat sensor array cocok untuk aplikasi seperti fotografi, pemindaian dokumen, pengawasan, pemrosesan video, dan bidang lain yang membutuhkan akuisisi citra cepat dan akurat.

Terdapat dua jenis akuisisi citra yang umum, yaitu sebagai berikut:

1. Akuisisi Citra Diam (*Static Image Acquisition*): Akuisisi citra diam adalah proses untuk mengambil citra tunggal pada waktu tertentu. Dalam hal ini, kamera atau sumber citra mengambil satu citra pada suatu waktu dan menghasilkan *file* gambar statis. Citra yang diambil dapat berupa foto, gambar medis, atau gambar lainnya. Contoh aplikasi akuisisi citra diam meliputi pemindaian dokumen, fotografi digital, analisis medis, dan banyak lagi.
2. Akuisisi Citra Bergerak (*Video/Image Sequence Acquisition*): Akuisisi citra bergerak melibatkan pengambilan sekuens citra dalam urutan waktu. Dalam hal ini, kamera atau sumber citra mengambil serangkaian citra dengan kecepatan tertentu sehingga dapat membentuk video atau urutan gambar. Contoh aplikasi akuisisi citra bergerak meliputi pemantauan lalu lintas, pengenalan gerakan, pengawasan keamanan, dan pengolahan video.

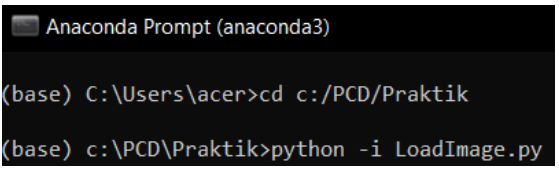
Dalam keduanya, teknologi kamera, sensor, atau perangkat lainnya digunakan untuk mendeteksi cahaya atau sinyal dari objek yang ingin diakuisisi, dan kemudian mengonversinya menjadi data citra yang dapat diolah secara digital. Perbedaan utama antara akuisisi citra diam dan bergerak adalah dalam cara pengambilan citra dan hasil akhirnya. Akuisisi citra diam menghasilkan satu citra tunggal pada satu waktu, sedangkan akuisisi citra bergerak menghasilkan serangkaian citra yang membentuk video atau urutan gambar.

Praktik 3.1. Akuisisi Citra dengan *Load Image*

Silakan kode program ini diketikkan pada teks editor misalnya menggunakan Sublime Text, simpanlah kode program ini dengan nama **LoadImage.py.**

```
import cv2

filepath = 'dataset\\aakhriansyahidi.jpg'#masukkan foto anda dimana sesuaikan dengan letak file foto anda

image = cv2.imread(filepath)

#menampilkan gambar
cv2.imshow("Ini Citra yang Diakuisisi", image)
cv2.waitKey(0)

#simpan gambar di dalam folder dataset yang sudah dibuat direktorinya
output = 'dataset\\GambarKamuNew.png'
cv2.imwrite(output, image)
#notifikasi jika gambar dengan nama file baru telah berhasil disimpan
print("[INFO] Gambar ini Telah Tersimpan di {}".format(output))
```

Untuk menjalankan program, maka Anda harus membuka Anaconda Prompt terlebih dahulu, kemudian ketikkan perintah berikut **cd c:/PCD/Praktik** untuk masuk ke dalam direktori terlebih dahulu, kemudian jalankan program dengan mengetikkan perintah **python -i LoadImage.py**, dimana **LoadImage.py** merupakan nama *file* dari kode program yang telah disimpan di dalam direktori.



Output Program:

Citra pada kotak dialog

Menyimpan kembali citra
dengan nama *file* yang berbeda

Praktik 3.2. Akuisisi Citra dengan *Video Capture*

Simpanlah kode program ini dengan nama **VideoCapture.py.**

```
from cv2 import *

cam_port = 0
cam = VideoCapture(cam_port)

result, image = cam.read()

if result:
```

```
    imshow("Akuisisi Citra", image)#teks pada dialog box
    imwrite("Inihasilakuisiscitra.png", image)#image hasil capture camera

    waitKey(0)
    destroyWindow("AkuisisiCitra")

else:
    print("Gambar tidak terdeteksi. Silakan coba lagi!")
```

cam_port = 0: Baris ini mendefinisikan variabel cam_port dan menginisialisasinya dengan nilai 0. Di sini, nilai 0 mengacu pada nomor port atau indeks yang digunakan untuk mengidentifikasi kamera yang akan diakses. Dalam hal ini, nilai 0 mengindikasikan bahwa kamera dengan indeks 0 akan digunakan. Jika ada beberapa kamera terhubung ke perangkat, indeks dapat berbeda-beda untuk setiap kamera.

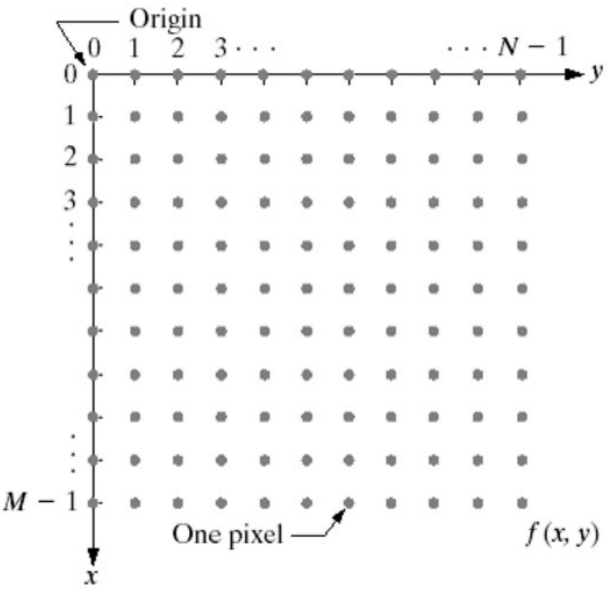
cam = VideoCapture(cam_port): Baris ini membuat objek cam menggunakan kelas VideoCapture dari library yang digunakan (seperti OpenCV). Objek ini digunakan untuk mengakses dan mengontrol kamera yang dipilih. Pada baris ini, objek cam diinisialisasi dengan menggunakan cam_port sebagai argumen. Dengan cara ini, kamera dengan indeks yang sesuai (dalam hal ini, indeks 0) akan diakses oleh objek cam yang telah dibuat.

Dengan kode tersebut, Anda dapat menggunakan objek cam untuk melakukan berbagai operasi pada kamera, seperti mengambil *frame* dari kamera, merekam video, mengakses properti kamera, dan sebagainya.

3.1.1. Piksel

Piksel *(pixel/picture element)* unsur gambar atau representasi sebuah titik terkecil dalam sebuah gambar grafis yang dihitung per inci (Lihat Gambar 3.2). Setiap piksel mewakili nilai kecerahan atau warna pada

posisi tertentu dalam citra. Nilai ini dapat berupa angka bilangan bulat dalam skala keabuan *(grayscale)* atau kumpulan nilai dalam ruang warna seperti RGB *(Red, Green, Blue)*. Piksel dalam citra digital dapat dikontrol dan dimanipulasi secara terpisah.



Gambar 3.2. Ilustrasi Piksel pada Citra

Pengolahan citra digital melibatkan manipulasi nilai piksel untuk melakukan berbagai operasi seperti:

1. Peningkatan Kualitas: Mengubah tingkat kecerahan atau kontras, mengurangi *noise*, menghaluskan citra, atau meningkatkan ketajaman.
2. Filtrasi: Menerapkan *filter* spasial atau frekuensi untuk menghilangkan *noise*, mempertajam kontur, atau memperoleh efek artistik.
3. Segmentasi: Memisahkan objek atau area yang berbeda dalam citra berdasarkan karakteristik seperti warna, tekstur, atau bentuk.
4. Ekstraksi Fitur: Mengidentifikasi fitur penting dalam citra seperti tepi, sudut, atau tekstur untuk tujuan analisis lebih lanjut.
5. Transformasi: Melakukan transformasi seperti rotasi, penskalaan, atau transformasi perspektif pada citra.

6. Rekonstruksi: Memadukan informasi dari beberapa piksel untuk menghasilkan citra yang lebih baik atau mengembalikan detail yang hilang.

Selain itu, piksel dalam citra digital dapat digunakan untuk pengenalan pola, kompresi citra, pengenalan wajah, deteksi objek, dan banyak lagi. Pada dasarnya, piksel adalah unit dasar yang memungkinkan pengolahan dan analisis lebih lanjut pada citra digital. Melalui manipulasi nilai piksel, berbagai operasi dan transformasi dapat dilakukan untuk menghasilkan hasil akhir yang diinginkan dalam pengolahan citra digital.

Selain itu, juga terdapat istilah resolusi. Resolusi pada citra digital merujuk pada jumlah piksel yang terdapat dalam citra dan menentukan tingkat detail yang dapat ditampilkan. Resolusi diukur dalam piksel atau dalam bentuk dimensi seperti lebar dan tinggi dalam piksel.

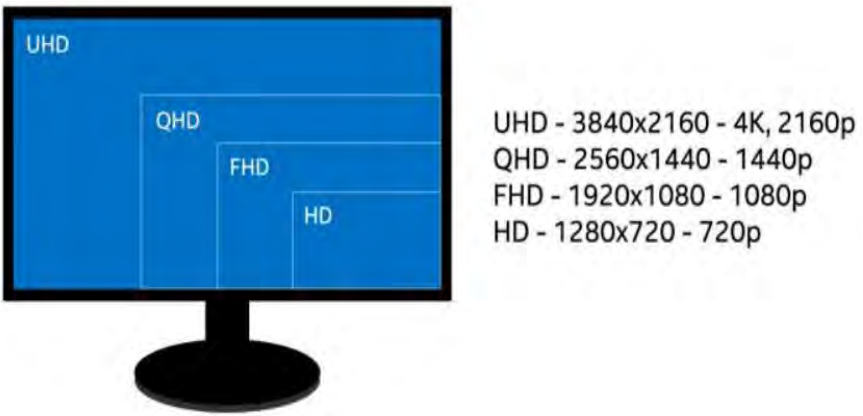
Resolusi horizontal mengacu pada jumlah piksel dalam arah horizontal atau lebar citra, sedangkan resolusi vertikal mengacu pada jumlah piksel dalam arah vertikal atau tinggi citra. Resolusi total citra dinyatakan sebagai jumlah total piksel yang ada dalam citra, yaitu hasil perkalian resolusi horizontal dengan resolusi vertikal.

Resolusi memengaruhi tingkat detail dan kualitas citra yang dapat dilihat. Semakin tinggi resolusi, semakin banyak piksel yang tersedia, sehingga citra dapat menampilkan lebih banyak detail dan kejelasan. Sebaliknya, resolusi yang rendah memiliki jumlah piksel yang lebih sedikit, sehingga citra mungkin tampak kabur atau kurang tajam (Lihat Gambar 3.3).

Gambar 3.3. Perbedaan Resolusi Citra

Resolusi juga memengaruhi ukuran *file* citra digital. Citra dengan resolusi tinggi cenderung memiliki ukuran file yang lebih besar karena menyimpan lebih banyak informasi piksel. Dalam aplikasi pengolahan citra digital, penting untuk mempertimbangkan resolusi citra yang sesuai dengan kebutuhan. Resolusi yang terlalu rendah mungkin mengakibatkan hilangnya informasi penting, sementara resolusi yang terlalu tinggi dapat memperlambat proses pengolahan dan membutuhkan penyimpanan yang lebih besar.

Resolusi juga berkaitan dengan tampilan citra pada perangkat output seperti monitor (Lihat Gambar 3.4) atau printer. Perangkat tersebut memiliki resolusi layar atau cetak sendiri, dan citra harus sesuai dengan resolusi tersebut agar dapat ditampilkan atau dicetak dengan baik. Monitor atau layar datar yang sering kita temui terdiri dari ribuan piksel yang terbagi dalam baris-baris dan kolom-kolom. Jumlah piksel yang terdapat dalam sebuah monitor dapat kita ketahui dari resolusinya. Resolusi maksimum yang disediakan oleh monitor adalah 3.840 x 2.160, dimana 3.840 piksel horizontal dan 2.160 piksel vertikal, maka jumlah piksel yang ada dalam layar monitor tersebut adalah 8,3 juta piksel. Semakin tinggi jumlah piksel yang tersedia dalam monitor, semakin tajam gambar yang mampu ditampilkan oleh monitor tersebut.



Gambar 3.4. Resolusi Monitor

Praktik 3.3. Mengetahui Skala pada Citra
Simpanlah kode program ini dengan nama **ImageScaling.py**.

```
import cv2

img = cv2.imread('dataset\\aakhriansyahidi.jpg',
cv2.IMREAD_UNCHANGED)

#mendapatkan dimensi gambar
dimensions = img.shape

#mengetahui nilai tinggi, lebar, jumlah saluran dalam gambar
height = img.shape[0]
width = img.shape[1]
channels = img.shape[2]

print('Image Dimension    : ',dimensions)
print('Image Height       : ',height)
print('Image Width        : ',width)
print('Number of Channels : ',channels) #RGB=3, CMYK=4,
Binary=2, Grayscale=0-1
```

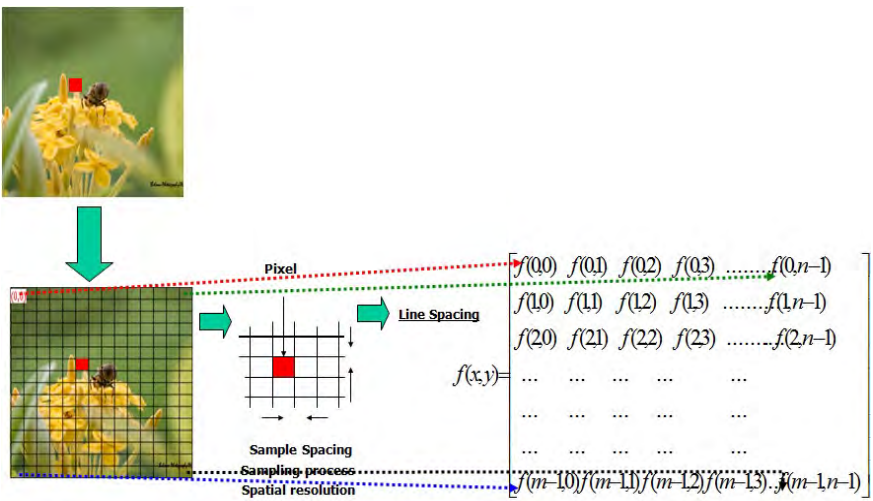
3.1.2. Image Sampling

Citra natural sebenarnya merupakan gambaran nyata dari objek yang kita rekam atau akuisisi. Misalkan suatu citra pemandangan di pinggir pantai yang menggambarkan adanya pantai, pasir, dan pohon kelapa yang benar-benar ada di dunia nyata. Citra nyata tersebut bisa direpresentasikan ke dalam bentuk fungsi kontinu *f(x,y)* (Lihat Gambar 3.5). Proses untuk mendigitalisasikan suatu fungsi kontinu ke dalam fungsi diskrit disebut sebagai *image sampling*. *Image sampling* merupakan salah satu proses digitalisasi untuk menghasilkan citra digital.

Citra kontinu adalah citra yang memiliki intensitas warna atau kecerahan yang kontinu dalam setiap titiknya, sedangkan citra digital

adalah representasi diskrit dari citra tersebut dengan menggunakan titik-titik sampel diskrit. Dalam *image sampling*, citra kontinu dipindahkan ke dalam domain diskrit dengan cara mengambil titik sampel di lokasi-lokasi tertentu. Titik sampel ini biasanya disebut sebagai piksel, yang merepresentasikan unit dasar dalam citra digital. Proses pengambilan sampel ini melibatkan dua parameter penting: resolusi spasial dan tingkat pencahayaan (*brightness level*).

Pada umumnya, perangkat akuisisi citra, seperti kamera digital, piksel diwakili oleh rata-rata dari sinyal di area tertentu dari sebuah pemandangan di hadapan kamera. Hasil *sampling* ditentukan oleh geometri dari elemen sensor yang ada pada perangkat akuisisi, jumlah piksel atau resolusi yang digunakan. Semakin bagus resolusinya, maka semakin presisi citra didigitalisasikan. *Output* dari *image sampling* ini adalah prediksi ukuran piksel/nilai resolusi dari citra tersebut (Lihat Gambar 3.5).

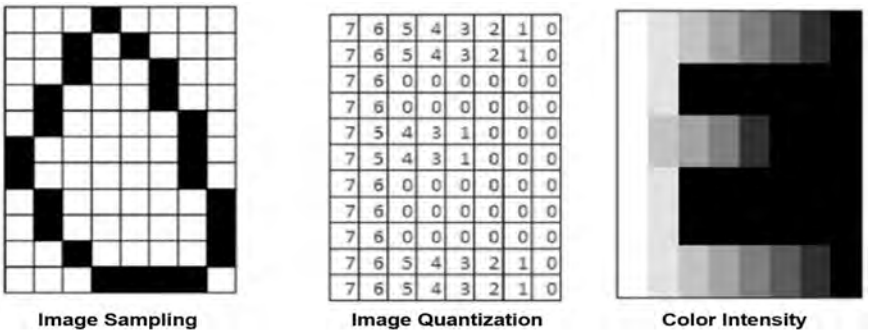


Gambar 3.5. *Image Sampling*

3.1.3. Image Quantization

Image quantization adalah proses pengurangan tingkat kecerahan (*brightness*) pada citra digital. Proses ini dilakukan dengan mengalokasikan nilai diskrit untuk mewakili intensitas warna atau

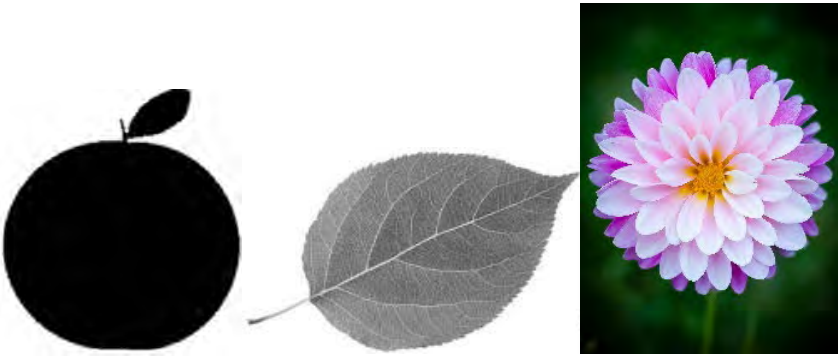
kecerahan pada setiap piksel dalam citra. Tujuan utama *dari image quantization* adalah untuk mengurangi jumlah bit yang diperlukan untuk menyimpan informasi piksel dalam citra, sehingga mengurangi ukuran *file* citra. Selain itu juga untuk menunjukkan banyaknya derajat nilai pada setiap piksel (menunjukkan jumlah bit pada citra digital, dimana *black & white* = 2 bit, *grayscale* = 8 bit, dan RGB = 24 bit). *Image quantization* juga erat kaitannya dengan memberikan nilai pada masing-masing piksel pada citra yang bergantung pada penentuan warna (Lihat Gambar 3.6).



Gambar 3.6. *Image Quantization*

Praktik 3.4. Mengetahui Skala pada Citra

Representasikanlah citra analog berikut menjadi citra digital dan kemudian proseslah ke dalam *image sampling* dan *image quantization* berbantuan aplikasi Ms Office Excel!



Silakan akses *file* citra pada tautan berikut: https://bit.ly/Praktik3-4

3.2 Jenis-Jenis Citra

3.2.1. Citra Berwarna

Citra berwarna merupakan citra yang mempunyai 3 buah kanal warna di dalamnya. Pada umumnya jenis citra ini terbentuk dari komponen merah/*red* (R), hijau/*green* (G), dan biru/*blue* (B) yang dimodelkan ke dalam ruang warna RGB. RGB adalah standar yang digunakan untuk menampilkan citra berwarna pada layar televisi maupun layar komputer. Namun, terdapat juga citra berwarna yang menggunakan ruang warna yang berbeda, seperti CMYK *(Cyan, Magenta, Yellow, Black)*, HSV *(Hue, Saturation, Value)*, YCbCr *(Luma, Chroma blue, Chroma red)*, dan Lab (L*a*b*), pada Gambar 3.7 merupakan contoh citra berwarna.

Gambar 3.7. Citra Berwarna

Citra berwarna atau *true color image* sering juga disebut dengan 24 bit *color image* karena setiap nilai pikselnya memerlukan penyimpanan sebesar 24 bit. Dimana masing-masing piksel pada setiap kanal mempunyai kemungkinan nilai sebanyak 256 kemungkinan, yaitu dengan nilai diantara 0-255. Hal tersebut membuktikan bahwa untuk setiap piksel pada satu kanal memerlukan 8 bit data. Karena citra berwarna memiliki 3 buah kanal, sehingga untuk satu piksel pada citra berwarna memerlukan 3 x 8 bit = 24 bit. Maka dari itu citra berwarna disebut juga sebagai 24 bit *color image*. Dengan kombinasi warna yang ada, maka citra berwarna mempunyai kemungkinan sebanyak 256 x 256 x 256 = 2^{24} = 16.777.216 variasi warna yang dihasilkan.

3.2.2. Citra Berwarna dengan Transparansi

Seiring dengan perkembangan dunia desain grafis, maka semakin diperlukan sebuah fitur baru dalam citra berwarna yaitu citra berwarna dengan transparasi. Citri ini biasanya digunakan untuk menghilangkan bagian *background* dari objek dalam sebuah citra. Citra ini terbentuk dari komponen RGB dan *Alpha* (A) yang dimodelkan ke dalam ruang warna RGBA. *Alpha* atau transparansi akan disimpan ke dalam satu kanal tambahan pada citra berwarna, sehingga total kanal yang digunakan sebanyak 4 kanal yaitu 3 kanal warna dan 1 kanal transparansi. Oleh karena itu, citra berwarna dengan transparansi disebut juga sebagai 32 bit *color image* karena untuk setiap nilai pikselnya memerlukan penyimpanan sebesar 32 bit.

Perlu diperhatikan bahwa tidak semua *file* citra mampu mendukung adanya transparansi. Sebagai contoh citra berekstensi JPG tidak mendukung transparansi sehingga jika ada citra dengan transparansi disimpan ke dalam ekstensi JPG, maka transparansinya akan hilang. Beberapa format yang mendukung citra ini adalah PNG dan GIF.

3.2.3. Citra Grayscale

Citra *grayscale* atau skala keabuan merupakan citra yang mempunyai 1 buah kanal sehingga yang ditampilkan hanyalah nilai intensitas atau dikenal juga dengan istilah derajat keabuan (Lihat Gambar 3.8). Karena citra jenis ini hanya mempunyai 1 kanal saja, maka citra ini mempunyai tempat penyimpanan yang lebih hemat. Jenis citra ini disebut juga sebagai 8 bit *color image* karena untuk setiap nilai piksel memerlukan penyimpanan sebesar 8 bit. Foto hitam putih maupun gambar yang ditampilkan oleh televisi hitam putih sebetulnya menggunakan citra *grayscale*, bukan dalam warna hitam dan warna putih. Namun di kalangan masyarakat istilah foto hitam putih maupun televisi hitam putih sudah terbiasa digunakan dalam kehidupan sehari-hari.

Gambar 3.8. Citra *Grayscale*

Secara teori terdapat beberapa cara dalam mengonversi citra berwarna menjadi citra skala keabuan. Cara yang paling mudah adalah dengan merata-ratakan semua nilai piksel RGB dengan persamaan:

$$\mathbf{y = \frac{1}{3}(R + G + B)} \quad (3.1)$$

Namun hasil konversi yang diberikan dengan menggunakan persamaan (3-1) tidak terlalu bagus. Untuk mendapatkan hasil yang lebih baik, maka dapat digunakan persamaan (3-2).

$$\mathbf{y = 0{,}299R + 0{,}587G + 0{,}144B} \quad (3.2)$$

3.2.4. Citra Biner

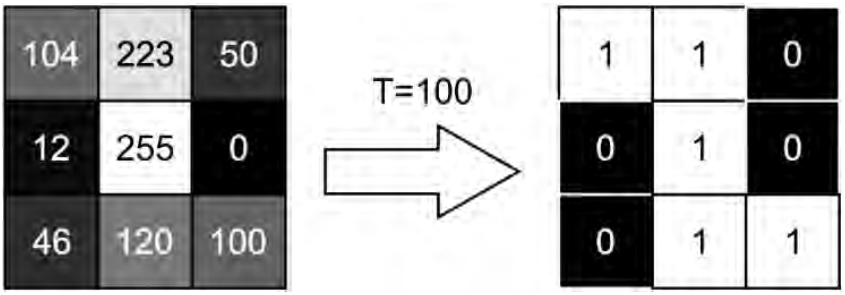
Citra biner atau citra hitam putih *(black & white image)* merupakan citra yang hanya memiliki 2 kemungkinan nilai untuk setiap pikselnya yaitu 0 atau 1. Nilai 0 akan tampil sebagai warna hitam sedangkan 1 akan tampil sebagai warna putih. Sehingga jenis citra ini hanya memerlukan 1 bit untuk menyimpan nilai pada setiap pikselnya. Jenis citra ini sering digunakan untuk proses *masking* ataupun proses segmentasi citra.

Untuk memperoleh suatu citra biner maka kita memerlukan citra *grayscale* yang dilakukan *thresholding* terhadapnya berdasarkan nilai ambang batas *(threshold)* yang ditentukan. Jika nilai piksel pada citra *grayscale* melebihi atau menyamai nilai *threshold*, maka nilai piksel

tersebut dikonversi menjadi nilai 1. Namun jika nilai piksel kurang dari nilai *threshold* maka nilai piksel tersebut dikonversi menjadi nilai 0 (Lihat Gambar 3.9). Proses konversi ini secara sederhana dapat menggunakan Persamaan 3.3.

$$g(x,y) = \begin{cases} 1, & jika\ f(x,y) \geq T \\ 0, & jika\ f(x,y) < T \end{cases} \tag{3.3}$$

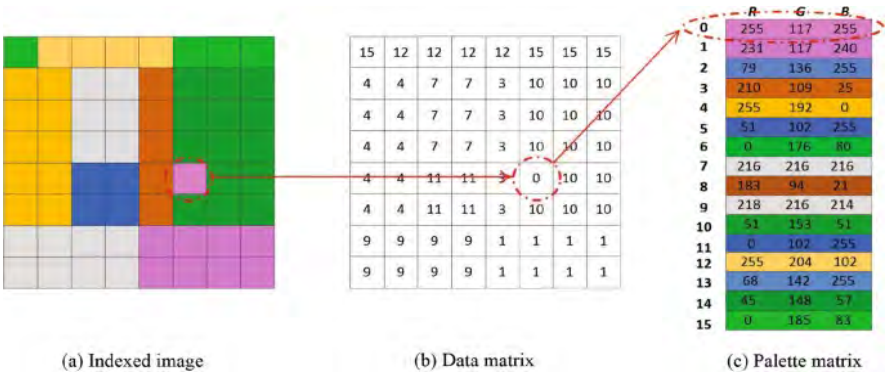
Menemukan nilai *threshold* untuk menghasilkan citra biner yang baik bukanlah hal yang sederhana. Bisa saja kita memilih nilai *threshold* sebesar 128 yang merupakan nilai tengah dari 0 sampai dengan 255, akan tetapi belum tentu citra biner yang dihasilkan merupakan hasil yang terbaik untuk semua citra. Otsu dalam penelitiannya mengusulkan sebuah metode dalam melakukan *threshold*. *Threshold* tersebut memiliki nilai yang dinamis bergantung terhadap nilai-nilai piksel dalam suatu citra. Metode Otsu termasuk metode terbaik dalam menentukan *thresholding* yang dapat digunakan untuk melakukan konversi citra *grayscale* menjadi citra biner.



Gambar 3.9. Simulasi *Threshold* pada Citra Biner secara Sederhana

3.2.5. Citra Terindeks

Citra terindeks merupakan salah satu jenis format citra yang digunakan dalam pengolahan citra digital. Format ini menggunakan sebuah palet warna yang terdiri dari kumpulan warna yang telah ditentukan sebelumnya (misalnya, 256 warna) untuk merepresentasikan gambar. Setiap piksel dalam citra terindeks dikodekan dengan nilai indeks yang mengacu pada posisi warna yang sesuai dalam palet (Lihat Gambar 3.10).



(a) Indexed image (b) Data matrix (c) Palette matrix

Gambar 3.10. Citra Terindeks

Berikut adalah beberapa karakteristik penting dari citra terindeks:

1. Palet Warna: Citra terindeks menggunakan palet warna, yang merupakan himpunan warna yang tersedia dalam citra. Palet ini dapat terdiri dari berbagai jumlah warna, tetapi dalam format yang umum digunakan seperti citra terindeks 8 bit, palet terdiri dari 256 warna.
2. Nilai Indeks: Setiap piksel dalam citra terindeks direpresentasikan oleh sebuah nilai indeks. Nilai indeks ini adalah angka diskrit yang berkisar dari 0 hingga N-1, di mana N adalah jumlah warna dalam palet. Nilai indeks tersebut menentukan warna yang digunakan untuk piksel tersebut sesuai dengan palet warna yang diberikan.
3. Efisiensi Penyimpanan: Citra terindeks cenderung lebih efisien dalam hal penyimpanan dibandingkan dengan citra berwarna penuh. Karena hanya perlu menyimpan nilai indeks untuk setiap piksel, ukuran file citra terindeks cenderung lebih kecil daripada citra dengan format lain yang menyimpan setiap nilai RGB untuk setiap piksel.
4. Reduksi Warna: Penggunaan palet terbatas dalam citra terindeks dapat menyebabkan reduksi warna atau kualitas warna yang lebih rendah dibandingkan dengan citra berwarna penuh. Terbatasnya jumlah warna dalam palet bisa membatasi kemampuan citra terindeks untuk mereproduksi rentang warna yang luas.

Citra terindeks sering digunakan dalam aplikasi di mana efisiensi penyimpanan menjadi faktor penting, seperti pada tampilan grafis di

komputer, web, dan game. Namun, dalam konteks pengolahan citra yang memerlukan representasi warna yang akurat dan detil, citra berwarna penuh (misalnya, format RGB) lebih umum digunakan.

Praktik 3.5. Melakukan perubahan citra dari RGB menjadi Biner
Simpanlah kode program ini dengan nama **RGBtoBiner.py**.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

#masukkan alamat direktori dari posisi penyimpanan foto anda
img = cv2.imread('dataset\\aakhriansyahidi.jpg',0)

#proses untuk mengubah data citra RGB menjadi skala biner/hitam putih
plt.imshow(img, cmap = 'binary', interpolation = 'bicubic')
plt.xticks([]), plt.yticks([])
plt.show()
```

Praktik 3.6. Melakukan perubahan citra dari RGB menjadi Grayscale
Simpanlah kode program ini dengan nama **RGBtoGrayscale.py**.

```
import numpy as np
import cv2

from matplotlib import pyplot as plt

#masukkan alamat direktori dari posisi penyimpanan foto anda
img = cv2.imread('dataset\\aakhriansyahidi.jpg',0)

#proses untuk mengubah data citra RGB menjadi skala keabuan/grayscale
plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
```

```
plt.xticks([]), plt.yticks([])
plt.show()
```

Praktik 3.7. Melakukan perubahan posisi citra dengan rotasi
Simpanlah kode program ini dengan nama **ImageRotating.py**.

```
import cv2
import numpy as np

filepath = 'dataset\\aakhriansyahidi.jpg'
image = cv2.imread(filepath)

#atur rotasi citra 90 derajat
image_norm = cv2.rotate(image, cv2.ROTATE_90_CLOCKWISE)

#tampilkan citra asli
cv2.imshow('original Image', image)

#tampilkan citra hasil rotasi
cv2.imshow('Rotated Image', image_norm)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Praktik 3.8. Melakukan perubahan posisi citra dengan mengubah ukuran pikselnya
Simpanlah kode program ini dengan nama **ImageResizing.py**.

```
import cv2

img = cv2.imread('dataset\\aakhriansyahidi.jpg')

#menghasilkan citra baru yang akan diatur ukurannya
newImg = cv2.resize(img, (450, 300))
```

```
#memunculkan hasil citra baru yang telah diatur ukurannya
cv2.imshow('Resized Image', newImg)

cv2.waitKey(0)
```

Praktik 3.9. Melakukan penurunan nilai bit pada citra grayscale
Simpanlah kode program ini dengan nama **GrayscaleLowBit.py**, silakan menggunakan dataset citra grayscale yang anda miliki.

```
import cv2
import numpy as np

imageSource = 'dataset\\Grayscale.jpg'
original_img = cv2.imread(imageSource,cv2.COLOR_BGR2GRAY)
cv2.imshow( "original", original_img )
result = original_img & 0b10000000
cv2.imshow( "out", result )
cv2.imwrite('out.jpg',result)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Praktik 3.10. Melakukan pengaburan *(blur)* pada citra
Simpanlah kode program ini dengan nama **BlurringImage.py**.

```
import cv2
import numpy as np

img = cv2.imread('dataset\\poliban.jpg',0)

# Gaussian Blur
Gaussian = cv2.GaussianBlur(img, (7, 7), 0)
cv2.imshow('Gaussian Blurring', Gaussian)
cv2.waitKey(0)
```

```
# Median Blur
Median = cv2.medianBlur(img, 5)
cv2.imshow('Median Blurring', Median)
cv2.waitKey(0)


# Bilateral Blur
Bilateral = cv2.bilateralFilter(img, 9, 75, 75)
cv2.imshow('Bilateral Blurring', Bilateral)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Praktik 3.11. Memberikan garis batas pada citra

Simpanlah kode program ini dengan nama **ColorBorder.py**.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img1 = cv2.imread('dataset\\poliban.jpg')

#membuat border pada citra
constant=
cv2.copyMakeBorder(img1,15,15,15,15,cv2.BORDER_CONSTANT)

#atur plot/tata letak untuk citra asli
plt.subplot(231),plt.imshow(img1,'gray'),plt.title('CITRA ASLI')

#atur plot/tata letak untuk citra hasil border
plt.subplot(232),plt.imshow(constant,'gray'),plt.title('CITRA
BORDER')
```

```
plt.show()
```

Praktik 3.12. Menggambar lingkaran di dalam citra

Simpanlah kode program ini dengan nama **DrawingCircles.py**.

```
import cv2

image = cv2.imread('dataset\\poliban.jpg')

cv2.circle(image,(100, 23), 25, (0,255,0))
cv2.circle(image,(40, 100), 25, (0,0,255))
cv2.circle(image,(100, 76), 52, (255,0,0), 3)
cv2.circle(image,(100, 24), 25, (0,255,0))
cv2.circle(image,(40, 220), 25, (0,0,255))
cv2.circle(image,(200, 76), 52, (255,0,0), 3)

cv2.imshow('Menggambar Lingkaran di dalam Citra',image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Praktik 3.13. Menuliskan teks di dalam citra

Simpanlah kode program ini dengan nama **WriteTextinImage.py**.

```
import cv2

img = cv2.imread('dataset\\poliban.jpg')

cv2.putText(img, "Politeknik Negeri Banjarmasin", (0, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1.0,
        (0, 0, 0), 4)

cv2.imshow('Menuliskan Teks pada Citra', img)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3.3 Format File pada Citra

3.3.1. Perbandingan Citra Bitmap dan Citra Vektor

Citra bitmap dan citra vektor adalah dua format citra yang berbeda dalam pengolahan citra digital. Berikut adalah perbandingan antara kedua format tersebut:

1. Representasi:

   Citra Bitmap: Citra bitmap (atau raster) direpresentasikan sebagai kumpulan piksel-piksel diskret yang tersusun dalam grid berbaris dan berkolom. Setiap piksel dalam citra bitmap memiliki nilai intensitas warna atau kecerahan yang spesifik. Format bitmap menyimpan informasi warna untuk setiap piksel secara langsung, termasuk untuk setiap detail dan nuansa kecil dalam citra.

   Citra Vektor: Citra vektor direpresentasikan menggunakan objek-objek geometris matematis seperti garis, kurva, dan poligon. Format vektor menyimpan instruksi-instruksi dan parameter-parameter untuk menggambar bentuk-bentuk tersebut, seperti posisi, panjang, sudut, dan warna. Citra vektor tidak menyimpan informasi piksel-piksel secara langsung, tetapi instruksi-instruksi untuk menghasilkan citra ketika diperlukan.

2. Skalabilitas:

   Citra Bitmap: Citra bitmap memiliki resolusi yang tetap, artinya jumlah piksel dan detail citra ditentukan pada saat pembuatan citra. Ketika citra bitmap diperbesar, jumlah piksel tetap, sehingga dapat menghasilkan efek buram jika diperbesar terlalu besar.

   Citra Vektor: Citra vektor memiliki skalabilitas yang tinggi. Karena citra vektor didasarkan pada objek geometris matematis, mereka dapat diperbesar atau diperkecil tanpa kehilangan kualitas atau kejelasan. Bentuk-bentuk dalam citra vektor dapat diubah ukurannya dengan proporsi yang tetap dan tepat.

3. Ukuran *File*:

   Citra Bitmap: Citra bitmap cenderung memiliki ukuran *file* yang besar. Setiap piksel dalam citra bitmap membutuhkan informasi intensitas warna secara langsung, sehingga semakin besar resolusi citra, semakin besar ukuran *file* yang dihasilkan.

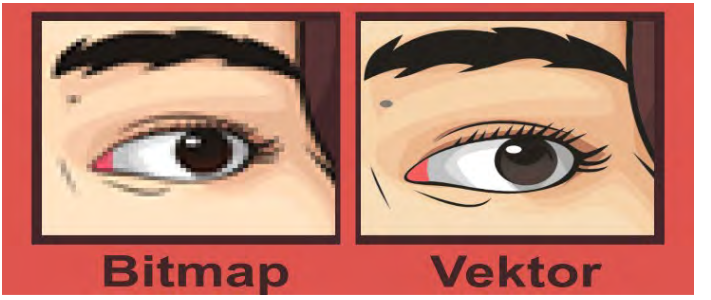
   Citra Vektor: Citra vektor cenderung memiliki ukuran *file* yang lebih kecil dibandingkan citra bitmap. Karena citra vektor hanya menyimpan instruksi-instruksi untuk menggambar bentuk-bentuk geometris, ukuran *file* cenderung lebih kecil dan tidak bergantung pada resolusi citra.

4. Kompleksitas Gambar:

   Citra Bitmap: Citra bitmap lebih cocok untuk merepresentasikan gambar dengan konten yang kompleks dan detail yang tinggi, seperti foto-foto atau citra realistis. Format bitmap dapat mereproduksi berbagai nuansa warna dan tingkat kecerahan yang halus.

   Citra Vektor: Citra vektor lebih cocok untuk merepresentasikan gambar dengan bentuk-bentuk geometris sederhana dan tajam, seperti logo atau ilustrasi. Format vektor tidak baik dalam mereproduksi tekstur kompleks atau gradasi warna yang halus.

Pemilihan format citra tergantung pada jenis gambar yang ingin direpresentasikan, penggunaan yang dimaksud, dan kebutuhan kualitas, skalabilitas, dan ukuran *file*. Lihat Gambar 3.11 untuk mengetahui perbedaan citra bitmap dan vektor.



Gambar 3.11. Citra Bitmap vs Citra Vektor

3.3.2. Ekstensi File pada Citra

Ekstensi *file* pada citra merupakan bagian dari nama *file* yang menunjukkan format *file* citra tersebut. Ekstensi *file* biasanya terdiri dari beberapa karakter yang mengikuti titik (.) setelah nama *file* utama. Ekstensi *file* memberikan petunjuk kepada sistem operasi atau perangkat lunak mengenai format dan jenis data yang ada dalam *file* citra. Berikut adalah beberapa ekstensi file yang umum digunakan untuk citra:

1. JPEG (.jpg atau .jpeg): Format JPEG *(Joint Photographic Experts Group)* adalah format yang paling umum digunakan untuk menyimpan citra yang dikompresi dengan kehilangan. Format ini memungkinkan ukuran file yang relatif kecil dengan kualitas gambar yang baik, dan umum digunakan untuk foto digital dan gambar berwarna.
2. PNG (.png): Format PNG *(Portable Network Graphics)* adalah format yang umum digunakan untuk citra dengan kualitas tinggi dan kompresi tanpa kehilangan. Format ini mendukung lapisan transparansi dan palet warna terindeks. PNG sering digunakan untuk grafis web, ikon, dan gambar dengan latar belakang transparan.
3. GIF (.gif): Format GIF *(Graphics Interchange Format)* adalah format yang mendukung animasi dan memiliki palet warna terindeks. GIF menggunakan kompresi tanpa kehilangan, tetapi terbatas pada 256 warna. Format ini sering digunakan untuk animasi sederhana, grafik web, dan ikon.
4. BMP (.bmp): Format BMP (Bitmap) adalah format dasar yang menggunakan representasi langsung dari piksel-piksel dalam citra tanpa kompresi. Format ini mendukung citra dengan kualitas tinggi dan bebas dari kompresi dengan kehilangan, sehingga file BMP cenderung lebih besar dibandingkan format kompresi lainnya. BMP umumnya digunakan dalam aplikasi industri atau keperluan khusus.
5. TIFF (.tiff atau .tif): Format TIFF *(Tagged Image File Format)* adalah format yang serbaguna dan mendukung berbagai jenis data citra, termasuk citra berwarna, citra *grayscale*, dan citra hitam putih. Format

ini sering digunakan dalam aplikasi profesional seperti fotografi, pencitraan medis, dan percetakan.

3.3.3. Ukuran File Citra

Ukuran *file* citra akan tergantung dengan jenis citra yang digunakan. Jenis citra *grayscale* akan mempunyai ukuran *file* lebih kecil jika dibandingkan dengan citra berwarna. Sedangkan jenis citra berwarna akan mempunyai ukuran lebih kecil jika dibandingkan dengan citra berwarna dengan transparansi. Berikut adalah contoh prediksi ukuran *file* pada setiap jenis citra yang setiap pikselnya bernilai dengan rentang 0 – 255 (8 bit). Perlu diingat bahwa 8 bit = 1 byte.

1. Sebuah citra *grayscale* dengan resolusi 320 x 240 piksel akan mempunyai prediksi ukuran *file* sebesar 320 x 240 x 1 kanal x 8 bit = 614.400 bit setara dengan konversi 76.800 byte.
2. Sebuah citra berwarna dengan resolusi 320 x 240 piksel akan mempunyai prediksi ukuran *file* sebesar 320 x 240 x 3 kanal x 1 byte = 230.400 byte.
3. Sebuah citra berwarna dengan transparansi mempunyai resolusi 320 x 240 piksel akan mempunyai prediksi ukuran *file* sebesar 320 x 240 x 4 kanal x 1 byte = 307.200 byte.

3.4 Hubungan Antarpiksel pada Citra

3.4.1. Ketetanggaan

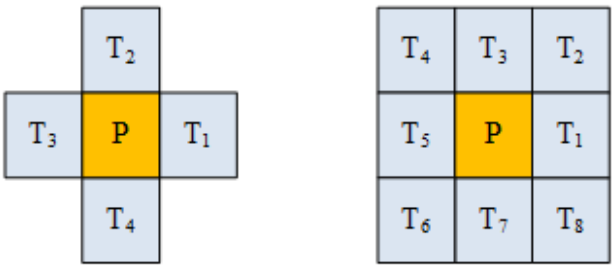
Ketetanggaan pada citra merujuk pada sejauh mana piksel-piksel dalam citra memiliki keterkaitan spasial. Konsep ini sering kali terkait dengan ruang warna dalam citra dan kemungkinan adanya pola atau struktur yang berulang. Terdapat dua jenis ketetanggaan utama dalam konteks pengolahan citra:

1. Ketetanggaan Spasial: Ketetanggaan spasial mengacu pada keterkaitan antara piksel-piksel yang berdekatan dalam citra. Piksel-piksel yang berdekatan biasanya memiliki kesamaan dalam intensitas warna, dan pola tersebut dapat digunakan untuk mengidentifikasi dan

mengekstrak fitur-fitur dari citra. Misalnya, dalam citra *grayscale*, piksel-piksel tetangga dengan intensitas yang serupa dapat membentuk tepi atau garis, sementara dalam citra berwarna, piksel-piksel tetangga dengan warna yang serupa dapat membentuk tekstur atau pola.

2. Ketetanggaan Kontekstual: Ketetanggaan kontekstual mengacu pada keterkaitan antara piksel saat ini dengan piksel-piksel di sekitarnya dalam konteks yang lebih luas. Hal ini terkait dengan penggunaan informasi dari piksel-piksel tetangga untuk membantu dalam analisis dan pengolahan citra. Misalnya, dalam teknik pengurangan noise seperti filter median, piksel yang sedang diproses diubah nilainya menjadi median dari nilai piksel-piksel tetangga di sekitarnya. Ketetanggaan kontekstual juga digunakan dalam teknik segmentasi citra, seperti region growing, di mana piksel-piksel yang terkait dikelompokkan bersama berdasarkan kesamaan kontekstual mereka.

Ketetanggaan dalam citra memiliki peran penting dalam berbagai aplikasi pengolahan citra, seperti pengurangan *noise*, segmentasi citra, pemulihan citra, deteksi objek, dan lainnya. Dengan memanfaatkan informasi ketetanggaan dalam citra, dapat diperoleh hasil yang lebih akurat dan berkualitas dalam analisis dan manipulasi citra. Pada Gambar 3.12 merupakan ilustrasi ketetanggaan pada citra dengan 4 dan 8 ketetanggaan.



Gambar 3.12. Ketetanggan Antarpiksel pada Citra

3.4.2. Jarak Antarpiksel

Jarak antarpiksel dalam citra merujuk pada jarak atau perbedaan antara dua piksel yang berdekatan dalam citra. Jarak ini diukur dalam unit spasial

seperti piksel atau satuan panjang yang relevan (misalnya, milimeter atau inci). Jarak antarpiksel dapat dihitung dalam berbagai konteks dalam pengolahan citra misalnya Euclidean, d_4 *(city block)*, dan d_8 *(checkboard)*. Berikut ini adalah beberapa penggunaan umum dari konsep jarak antarpiksel:

1. Resolusi Spasial: Jarak antarpiksel dalam citra dapat digunakan untuk mengukur resolusi spasial citra. Resolusi spasial mengacu pada sejauh mana citra dapat membedakan detail kecil atau objek yang terpisah secara spasial. Semakin kecil jarak antarpiksel dalam citra, semakin tinggi resolusi spasialnya, dan sebaliknya. Resolusi spasial yang tinggi penting dalam aplikasi seperti deteksi objek, pemrosesan gambar medis, dan pengenalan wajah.
2. *Filter* Spasial: Dalam teknik pengolahan citra, *filter* spasial sering digunakan untuk menerapkan operasi pada piksel-piksel dalam citra. *Filter* tersebut dapat berupa *filter* rata-rata, *filter sharpening*, *filter edge detection*, dan lain-lain. Jarak antarpiksel dalam konteks ini digunakan untuk menghitung koefisien yang tepat dalam operasi *filter*. *Filter* spasial sering kali beroperasi dengan mengambil piksel tetangga dalam radius tertentu, dan jarak antarpiksel digunakan untuk menentukan radius tersebut.
3. Transformasi Spasial: Dalam beberapa transformasi citra, seperti transformasi Fourier, jarak antarpiksel memiliki peran penting. Transformasi Fourier mengubah citra dari domain spasial menjadi domain frekuensi, yang memungkinkan analisis lebih lanjut terhadap karakteristik frekuensi citra. Jarak antarpiksel dalam citra spasial berkaitan langsung dengan jarak dalam domain frekuensi setelah transformasi Fourier. Transformasi ini sering digunakan dalam kompresi citra, ekstraksi fitur, dan pemrosesan sinyal.

Perlu dicatat bahwa jarak antarpiksel dalam citra dapat bervariasi tergantung pada resolusi dan skala citra. Semakin tinggi resolusi citra, semakin kecil jarak antarpikselnya. Selain itu, dalam citra dengan skala berbeda, jarak antarpiksel dapat diukur dalam unit yang berbeda. Misalnya,

dalam citra medis, jarak antarpiksel dapat diukur dalam satuan panjang fisik seperti milimeter atau inci, sementara dalam citra digital, jarak antarpiksel diukur dalam satuan piksel.

3.5 Latihan

1. Mengapa citra perlu diakuisisi?
2. Jelaskan perbedaan *Static Image Acquisition* dengan *Video/Image Sequence Acquisition*!
3. Mengapa untuk memperoleh citra digital memerlukan proses *image sampling* dan *image quantization*?
4. Pengolahan citra digital melibatkan manipulasi nilai piksel untuk melakukan berbagai operasi, sebutkan!
5. Apa kegunaan *thresholding*?
6. Hitunglah ukuran citra melalui prediksi perhitungan ukuran citra jika sebuah citra grayscale, berwarna, dan berwarna dengan transparansi mempunyai resolusi 1024 x 360 piksel!

BAB 4

OPERASI DASAR PADA CITRA DIGITAL

Capaian Pembelajaran:

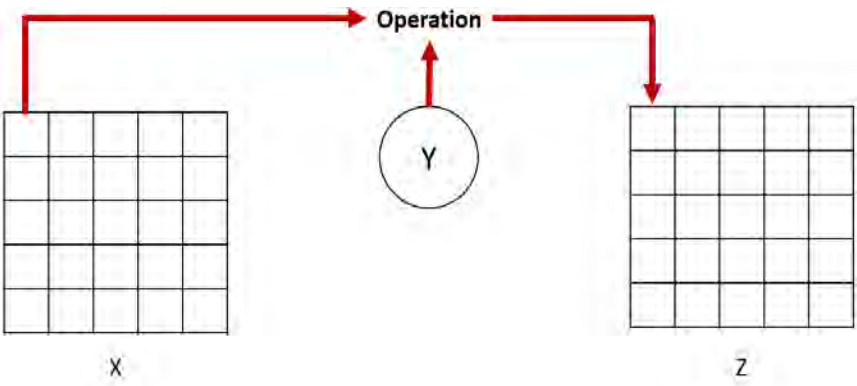
1. Mampu mengetahui dan memahami operasi aritmatika pada citra.
2. Mampu mengetahui, memahami, dan mempraktikkan operasi logika pada citra.

4.1 Operasi Aritmatika pada Citra

4.1.1. Penambahan

Apabila nilai piksel pada suatu citra ditambahkan, maka sebenarnya efek yang terjadi adalah citra tersebut akan bertambah *grayscale*-nya sehingga tampak lebih terang. Untuk citra berwarna, penambahan nilai piksel berarti penambahan derajat untuk masing-masing komponen citra.

Nilai maksimal pada citra adalah 255, sehingga jika hasil penambahan tersebut melebihi 255 maka akan ditetapkan menjadi 255 saja. Misalkan suatu komponen piksel pada citra bernilai 200 kemudian ditambahkan dengan 75 maka hasilnya adalah 275 sehingga akan menjadi 255. Gambar 4.1 merupakan ilustrasi pengoperasian.



Gambar 4.1. Ilustrasi Pengoperasian Citra

4.1.2. Pengurangan

Jika nilai piksel pada suatu citra dikurangi, maka sebenarnya efek yang terjadi ialah citra tersebut akan berkurang *grayscale*-nya, sehingga tampak lebih gelap.

Nilai minimum piksel pada sebuah citra adalah 0, sehingga jika hasil pengurangan tersebut lebih kecil dari 0, maka akan ditetapkan menjadi nilai 0. Misalkan suatu komponen piksel pada citra bernilai 42 kemudian dikurangkan dengan 50 maka hasilnya adalah -8 sehingga akan menjadi 0.

4.2 Operasi Logika pada Citra

Operasi logika ini sering kali digunakan pada citra biner yang nilainya hanya memiliki kondisi 0 atau 1. Operasi ini akan bekerja sesuai dengan prinsip operator logika untuk nilai Boolean *(true* yaitu 1 atau *false* yaitu 0*)*.

4.2.1. AND

Operator AND akan menghasilkan *true* (1) jika kedua piksel yang dioperasikan memiliki nilai piksel *true* (1). Jika menggunakan teori himpunan, ini adalah operasi irisan. Untuk praktik 4.1 sampai dengan 4.4 silakan menggunakan citra yang dapat diunduh pada tautan berikut: https://bit.ly/CitraOperasiLogika.

Praktik 4.1. Operasi AND pada Citra

Simpanlah kode program ini dengan nama **ANDBitwiseOperation.py**.

```
import cv2
import numpy as np

img1 = cv2.imread('dataset\\input1.png')
img2 = cv2.imread('dataset\\input2.png')

dest_and = cv2.bitwise_and(img2, img1, mask = None)

cv2.imshow('Operasi AND pada Citra', dest_and)
```

```
if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
```

4.2.2. OR

Operator OR menghasilkan *true* (1) apabila minimal satu piksel yang dioperasikan bernilai *true* (1). Pada teori himpunan dinamakan dengan operasi gabungan.

Praktik 4.2. Operasi OR pada Citra

Simpanlah kode program ini dengan nama **ORBitwiseOperation.py**.

```
import cv2
import numpy as np

img1 = cv2.imread('dataset\\input1.png')
img2 = cv2.imread('dataset\\input2.png')

dest_or = cv2.bitwise_or(img2, img1, mask = None)

cv2.imshow('Operasi OR pada Citra', dest_or)

if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
```

4.2.3. NOT

Operator NOT akan menegasikan/membalik nilai *true* (1) menjadi *false* (0) dan *false* (0) menjadi *true* (1). Dalam teori himpunan disebut dengan operasi komplemen.

Praktik 4.3. Operasi NOT pada Citra

Simpanlah kode program ini dengan nama **NOTBitwiseOperation.py**.

```
import cv2
import numpy as np
```

```
img1 = cv2.imread('dataset\\input1.png')
img2 = cv2.imread('dataset\\input2.png')

dest_not1 = cv2.bitwise_not(img1, mask = None)
dest_not2 = cv2.bitwise_not(img2, mask = None)

cv2.imshow('Operasi NOT pada Citra 1', dest_not1)
cv2.imshow('Operasi NOT pada Citra 2', dest_not2)

if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
```

4.2.4. XOR

Operasi logika XOR mengharuskan salah satu nilai piksel pada citra bernilai *true* (1) maka akan menghasilkan *true* (1), jika kedua-duanya bernilai *true* (1) atau *false* (0) makan akan menghasilkan *false* (0).

Praktik 4.4. Operasi XOR pada Citra

Simpanlah kode program ini dengan nama **XORBitwiseOperation.py**.

```
import cv2
import numpy as np

img1 = cv2.imread('dataset\\input1.png')
img2 = cv2.imread('dataset\\input2.png')

dest_xor = cv2.bitwise_xor(img1, img2, mask = None)

cv2.imshow('Operasi XOR pada Citra', dest_xor)

if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
```

Praktik 4.5. Simulasi Operasi Logika

Representasikanlah citra 1 dan 2 yang merupakan inputan pada praktik 4.1 sampai dengan 4.4 kemudian proseslah ke dalam *image sampling* dan *image quantization* berbantuan aplikasi Ms Office Excel, setelah itu lakukan operasi AND, OR, NOT, dan XOR, serta bandingkan hasilnya dengan *output* program 4.1 sampai dengan 4.4.

4.3 Latihan

1. Diketahui data citra *grayscale* A dan B dengan 4 x 4 piksel sebagai berikut:

150	109	180	71
142	140	142	142
200	220	7	8
90	100	111	112

Citra A

49	59	111	222
43	58	112	233
22	57	114	250
12	56	124	251

Citra B

   a. Lakukan proses penjumlahan pada citra *grayscale* A dan B dengan nilai 155!
   b. Lakukan proses pengurangan pada citra *grayscale* A dan B dengan nilai 160!
2. Operasi apa yang digunakan untuk memperoleh negatif pada sebuah citra?
3. Diketahui data citra biner A dan B dengan 6 x 6 piksel sebagai berikut:

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
0	0	1	1	0	0
0	1	0	0	1	0
1	0	0	0	0	1

Citra A

0	0	1	1	0	0
1	1	1	1	1	1
1	0	0	0	0	1
1	0	0	0	0	1
1	0	0	0	0	1
0	1	1	1	1	0

Citra B

a. Lakukan operasi AND pada citra biner A dan B!
b. Lakukan operasi OR pada citra biner A dan B!
c. Lakukan operasi NOT pada citra biner A!
d. Lakukan operasi XOR pada citra biner A dan B!

BAB 5

HISTOGRAM CITRA

Capaian Pembelajaran:

1. Mampu mengetahui, memahami, dan menjelaskan konsep histogram.
2. Mampu membuat kode program histogram citra.
3. Mampu memahami kandungan informasi dalam histogram citra.
4. Mampu melakukan manipulasi pada histogram citra untuk memperoleh citra yang diinginkan.

5.1 Histogram

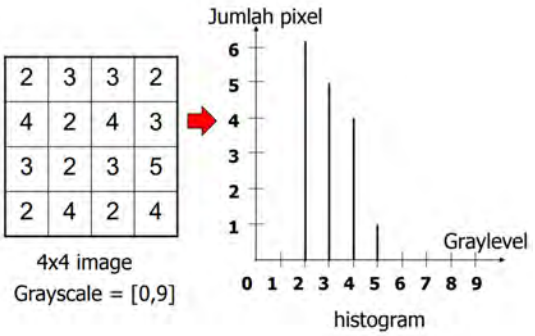
Dalam pengolahan citra digital, histogram merupakan representasi grafis dari distribusi intensitas piksel dalam sebuah citra. Histogram menggambarkan jumlah piksel dengan intensitas tertentu pada sumbu Y (frekuensi) terhadap intensitas piksel pada sumbu X.

Secara umum, histogram membantu kita memahami bagaimana intensitas piksel tersebar dalam citra. Hal ini dapat memberikan informasi tentang kontras, kecerahan, distribusi nilai piksel, dan keadaan pencahayaan dalam citra tersebut. Pada histogram, sumbu X biasanya mewakili rentang nilai intensitas piksel (misalnya dari 0 hingga 255 untuk citra *grayscale* dengan 8 bit depth), sedangkan sumbu Y menunjukkan jumlah piksel dengan intensitas tersebut.

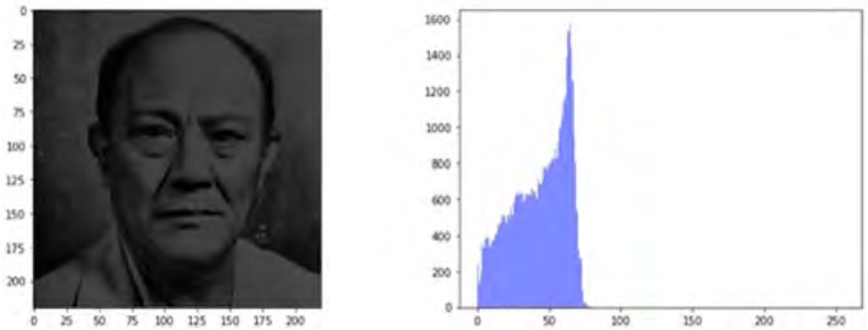
Dengan menggunakan histogram, kita dapat melihat apakah citra cenderung memiliki intensitas yang terkonsentrasi pada rentang tertentu atau apakah distribusinya lebih merata. Histogram dapat digunakan untuk berbagai tujuan dalam pengolahan citra digital, seperti penyesuaian kontras, segmentasi citra, deteksi tepi, ekualisasi histogram, dan lain sebagainya. Dengan menganalisis histogram, kita dapat membuat keputusan dan melakukan operasi pengolahan yang sesuai untuk memperbaiki atau memanipulasi citra sesuai kebutuhan kita.

5.2 Histogram Citra Grayscale

Histogram *grayscale* merupakan histogram yang digunakan untuk menganalisis dan menggambarkan distribusi intensitas piksel dalam citra *grayscale*. Citra *grayscale* adalah citra yang hanya memiliki satu saluran warna atau tingkat keabuan, di mana setiap pikselnya direpresentasikan oleh satu nilai intensitas tunggal. Histogram *grayscale* mencerminkan seberapa sering intensitas piksel tertentu muncul dalam citra. Histogram *grayscale* biasanya memiliki sumbu X yang mewakili rentang intensitas piksel, misalnya dari 0 hingga 255 untuk citra dengan kedalaman warna 8 bit (Lihat Gambar 5.1 dan Gambar 5.2). Sumbu Y pada histogram *grayscale* menunjukkan jumlah piksel dalam citra dengan intensitas tertentu. Dengan menganalisis histogram *grayscale*, kita dapat mendapatkan informasi tentang distribusi intensitas piksel dalam citra.



Gambar 5.1. Contoh Pembuatan Histogram Citra



Gambar 5.2. Histogram Citra Grayscale

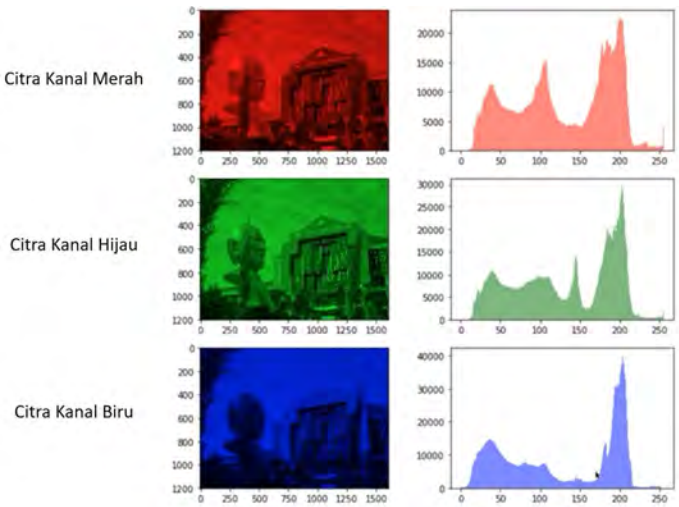
Misalnya, histogram yang memiliki puncak di sekitar nilai intensitas tinggi menunjukkan bahwa citra tersebut cenderung lebih cerah, sementara histogram yang merata menunjukkan distribusi intensitas yang merata di seluruh rentang. Histogram *grayscale* juga digunakan dalam berbagai teknik pengolahan citra, seperti penyesuaian kontras, ekualisasi histogram, segmentasi citra, dan operasi lainnya yang melibatkan manipulasi intensitas piksel.

5.3 Histogram Citra Berwarna

Histogram citra berwarna merupakan histogram yang digunakan untuk menganalisis dan menggambarkan distribusi intensitas warna *(luminance)* dalam citra berwarna. Citra berwarna terdiri dari tiga saluran warna utama, yaitu merah *(red)*, hijau *(green)*, dan biru *(blue)*, yang dikombinasikan untuk menghasilkan berbagai warna. Histogram citra berwarna sering kali ditampilkan dalam bentuk histogram 2D atau 3D, tergantung pada jumlah saluran warna yang digunakan. Histogram 2D menggambarkan distribusi intensitas dua saluran warna (misalnya, hijau vs merah) pada sumbu X dan Y, sementara histogram 3D menggambarkan distribusi intensitas ketiga saluran warna pada sumbu X, Y, dan Z.

Dalam histogram citra berwarna, setiap saluran warna memiliki rentang intensitasnya sendiri, biasanya dari 0 hingga 255 untuk citra dengan kedalaman warna 8 bit per saluran warna. Histogram menggambarkan jumlah piksel dalam citra yang memiliki kombinasi intensitas warna tertentu dalam saluran-saluran tersebut. Dengan menganalisis histogram citra berwarna, kita dapat memperoleh pemahaman tentang distribusi intensitas warna dalam citra. Hal ini berguna untuk penyesuaian kontras, peningkatan kecerahan, deteksi objek atau fitur berwarna, segmentasi citra berdasarkan warna, dan berbagai teknik pengolahan citra lainnya yang melibatkan manipulasi intensitas warna.

Sebuah citra berwarna bisa dekomposisi menjadi citra yang banyaknya sesuai dengan jumlah kanal citra. Masing-masing citra hasil dekomposisi tersebut mempunyai histogramnya masing-masing (Lihat Gambar 5.3).

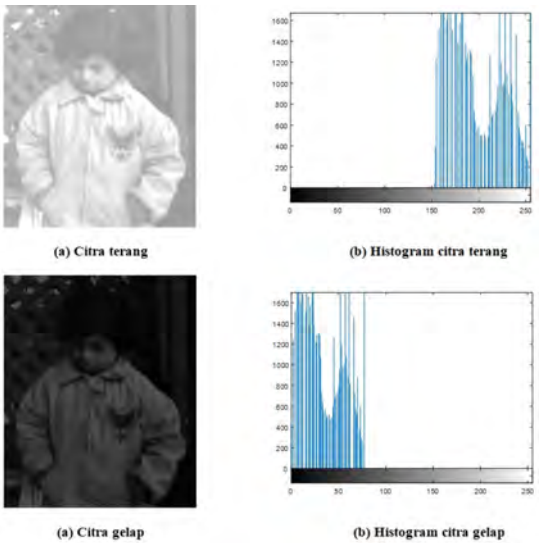


Gambar 5.3. Histogram Citra Grayscale

5.4 Kandungan Informasi dalam Histogram Citra

5.4.1. Intensitas Citra Keseluruhan

Pada sebuah histogram dapat diketahui intensitas citra secara keseluruhan sehingga dapat disimpulkan citra tersebut gelap atau terang (Lihat Gambar 5.4).

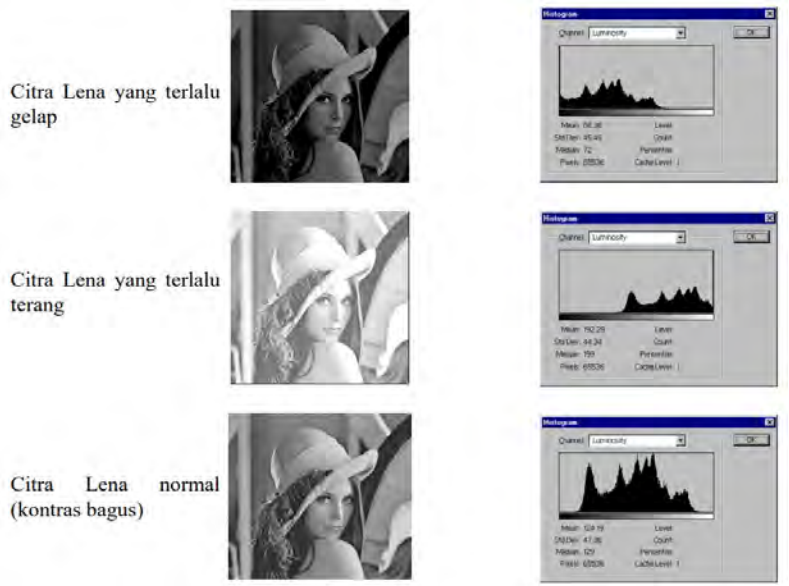


Gambar 5.4. Ilustrasi Histogram pada Citra Terang dan Citra Gelap

Dari Gambar 5.4, dapat disimpulkan bahwa citra dengan intensitas terang akan mempunyai sebaran histogram yang cenderung ke kanan dan sedangkan citra intensitas gelap akan mempunyai sebaran histogram yang cenderung ke kiri. Terangnya cahaya dari sebuah citra dapat diukur dengan menghitung nilai rata-rata dari histogram. Semakin tinggi nilai rata-rata, maka semakin tinggi kecerahan suatu citra.

5.4.2. Kontras Citra

Sebuah citra yang mempunyai kontras baik/tinggi maka nilai-nilainya terdistribusi dengan merata, sedangkan citra yang mempunyai kontras buruk/rendah maka nilai-nilainya berkumpul pada satu sisi tertentu (Lihat Gambar 5.5). Citra dengan standar deviasi lebih besar mempunyai kontras lebih tinggi dibandingkan citra dengan standar deviasi kecil.

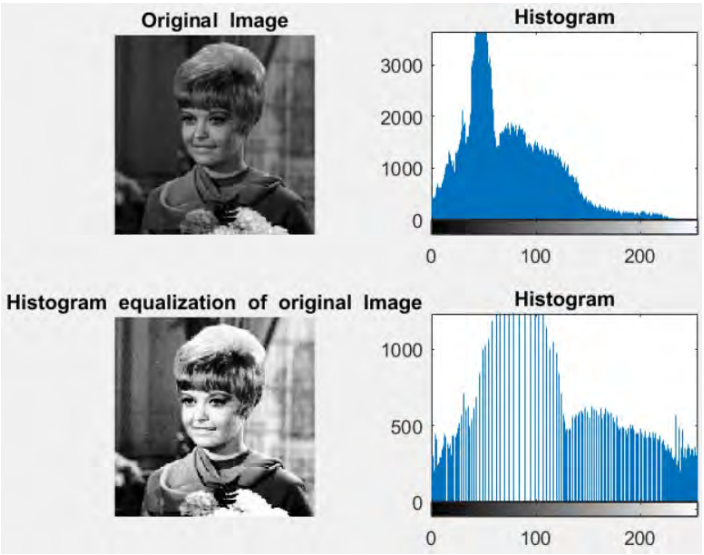


Gambar 5.5. Perbandingan Nilai Rata-Rata dan Nilai Standar Deviasi pada Citra

5.5 Histogram Equalization

Histogram *equalization* salah satu teknik yang digunakan dalam pengolahan citra digital untuk memperbaiki kontras dan meningkatkan kualitas citra. Tujuan utama dari histogram *equalization* adalah untuk

mendistribusikan secara merata nilai intensitas piksel dalam rentang dinamik yang tersedia (Lihat Gambar 5.6).



Gambar 5.6. Contoh Penerapan Histogram Equalization

Fungsi utama dari histogram *equalization* adalah sebagai berikut:

1. Peningkatan Kontras: Histogram *equalization* dapat meningkatkan kontras dalam citra dengan mengubah distribusi intensitas piksel. Dengan meratakan distribusi intensitas piksel di seluruh rentang dinamik, perbedaan intensitas antara piksel-piksel yang berdekatan diperbesar, menghasilkan citra yang lebih tajam dan kontras yang lebih baik.
2. Normalisasi Citra: Histogram *equalization* juga dapat digunakan untuk mengubah citra sehingga memiliki distribusi intensitas yang lebih normal atau mendekati distribusi Gaussian. Ini berguna dalam situasi di mana citra mungkin memiliki distribusi intensitas yang tidak merata atau terdistorsi.
3. Meningkatkan Detail Citra: Dengan meratakan distribusi intensitas, histogram *equalization* dapat membantu meningkatkan penampilan detail dalam citra. Hal ini terutama berguna ketika ada detail yang

tersembunyi atau redup di dalam citra karena distribusi intensitas yang tidak merata.

4. Peningkatan Visualisasi: Histogram *equalization* juga digunakan sebagai teknik pra-pemrosesan untuk meningkatkan visualisasi citra. Dengan meningkatkan kontras dan meratakan distribusi intensitas, citra yang disesuaikan dapat lebih mudah dilihat dan dianalisis oleh mata manusia.

Namun, perlu diingat bahwa histogram *equalization* juga memiliki beberapa batasan. Misalnya, teknik ini dapat memperkuat derau atau noise yang ada dalam citra, karena mengubah distribusi intensitas piksel. Oleh karena itu, seringkali digunakan bersama dengan teknik pengurangan derau untuk memperoleh hasil yang optimal.

Untuk lebih mudah dalam memahami histogram *equalization*, berikut merupakan contoh penerapannya pada citra dengan resolusi 6 x 6 piksel dan dengan rentang 0-7.

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

Dari citra di atas, maka n_G (jumlah tingkat keabuan di dalam citra) = 8 dan n (jumlah seluruh piksel di dalam citra), sehingga: s_k (Histogram *Equalization*) = $\frac{8-1}{36}\sum_{j=0}^{k} nrj$

r_j	n_{rj}	$\sum_{j=0}^{k} nrj$	S_k
0	5	5	1
1	5	10	2
2	8	18	4
3	2	20	4
4	1	21	5
5	2	23	5
6	9	32	7
7	4	36	7

Maka keluaran citra yang dihasilkan adalah:

7	7	4	4	7	7
7	7	5	5	7	7
7	1	2	2	4	4
7	1	2	1	4	4
4	1	2	1	5	4
7	7	4	7	2	4

5.6 Histogram Specification

Histogram *specification* merupakan teknik dalam pengolahan citra digital yang digunakan untuk mengubah distribusi intensitas piksel dalam citra menjadi distribusi yang diinginkan. Teknik ini memanipulasi histogram citra dengan mengacu pada histogram referensi yang telah ditentukan sebelumnya.

Fungsi utama dari histogram *specification* adalah sebagai berikut:

1. Mencocokkan Distribusi Intensitas: Histogram *specification* digunakan untuk mencocokkan distribusi intensitas citra yang tidak sesuai dengan distribusi intensitas referensi. Dengan merujuk pada histogram referensi, intensitas piksel dalam citra asli diubah sedemikian rupa sehingga histogram hasil cocok dengan histogram referensi yang diinginkan. Hal ini berguna untuk mengubah citra menjadi memiliki karakteristik intensitas yang diharapkan.
2. Pemetaan ke Tingkat Intensitas yang Diketahui: Dalam beberapa kasus, kita mungkin memiliki citra dengan distribusi intensitas yang tidak diinginkan, tetapi kita tahu persis distribusi intensitas yang diinginkan. Dalam situasi ini, histogram *specification* memungkinkan kita untuk secara tepat memetakan intensitas piksel dalam citra asli ke tingkat intensitas yang diinginkan. Ini berguna dalam aplikasi seperti pemrosesan warna atau koreksi warna, di mana kita ingin mengontrol secara presisi karakteristik intensitas citra.
3. Mencapai Efek Visual yang Diinginkan: Histogram *specification* juga digunakan untuk mencapai efek visual tertentu pada citra. Misalnya, dengan menggunakan histogram referensi yang spesifik, kita dapat mengubah citra untuk mencapai tampilan yang lebih cerah, lebih

gelap, atau memperoleh karakteristik intensitas yang unik. Hal ini membuka peluang untuk manipulasi kreatif dalam pengolahan citra.

Histogram *specification* adalah alat yang berguna dalam mengontrol dan memanipulasi distribusi intensitas dalam citra digital. Dengan mengubah histogram citra, kita dapat mencocokkan atau memetakan intensitas piksel ke distribusi yang diinginkan, sehingga mencapai tujuan pengolahan dan efek visual yang diinginkan.

Agar lebih mudah dalam memahami histogram *specification*, berikut ini adalah contoh penerapannya pada citra dengan resolusi 6 x 6 piksel dan dengan rentang nilai 0 – 7. Diasumsikan spesifikasi *grayscale* yang diharapkan adalah {0: 5%, 1: 5%, 2: 10%, 3: 10%, 4: 25%, 5: 5%, 6: 25%, 7: 15%}.

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

Kemudian lakukan proses berikut ini untuk langkah awal:

x	p_x	$\int_0^x px(u)du$
0	5/36	5/36=0,138
1	5/36	10/36=0,277
2	8/36	18/36=0,500
3	2/36	20/36=0,555
4	1/36	21/36=0,583
5	2/36	23/36=0,638
6	9/36	32/36=0,888
7	4/36	36/36=1,000

Langkah berikutnya jika tingkat keabuan dari citra telah tersedia, maka proses dengan tabel berikut:

x	pz	$\int_0^z pz(u)du$
0	5%=0,05	0,05
1	5%=0,05	0,10
2	10%=0,10	0,20
3	10%=0,10	0,30

4	25%=0,25	0,55
5	5%=0,05	0,60
6	25%=0,25	0,85
7	15%=0,15	1,00

Maka pemetaan masukan tingkat keabuan ke dalam keluaran tingkat keabuan berdasarkan spesifikasi yang ditentukan yaitu sebagai berikut:

x	Pemetaan	z
0	0,138≈0,10	1
1	0,277≈0,30	3
2	0,500≈0,55	4
3	0,555≈0,55	4
4	0,583≈0,60	5
5	0,683≈0,60	5
6	0,888≈0,85	6
7	1,000≈1,00	7

Maka keluaran citra yaitu sebagai berikut:

6	6	4	4	7	7
6	6	5	5	7	6
6	1	3	3	4	4
7	1	3	1	4	4
4	1	3	1	5	4
6	6	4	6	3	4

Untuk praktik 5.1 sampai dengan 5.5 silakan menggunakan citra yang dapat diunduh pada tautan berikut: https://bit.ly/CitraHistogram.

Praktik 5.1. Histogram Citra *Grayscale*

Simpanlah kode program ini dengan nama **GrayscaleImageHistogram.py**.

```
import cv2

from matplotlib import pyplot as plt

# membaca citra di dalam dataset
img = cv2.imread('dataset\\CitraGrayscale.jpg',0)
```

```
# temukan frekuensi piksel dalam rentang 0-255
histr = cv2.calcHist([img],[0],None,[256],[0,256])

# menunjukkan grafik plotting dari suatu gambar
plt.plot(histr)
plt.show()
```

Praktik 5.2. Histogram Citra Berwarna

Simpanlah kode program ini dengan nama **RGBImageHistogram.py**.

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

def plot_histogram(image):
    # Pisahkan kanal RGB
    r, g, b = cv2.split(image)

    # Hitung histogram untuk setiap kanal
    hist_r, bins_r = np.histogram(r.flatten(), bins=256, range=[0, 256])
    hist_g, bins_g = np.histogram(g.flatten(), bins=256, range=[0, 256])
    hist_b, bins_b = np.histogram(b.flatten(), bins=256, range=[0, 256])

    # Plot histogram
    plt.figure(figsize=(10, 6))
    plt.subplot(311)
    plt.hist(r.flatten(), bins=256, range=[0, 256], color='r')
    plt.title('Red Channel')
    plt.subplot(312)
    plt.hist(g.flatten(), bins=256, range=[0, 256], color='g')
    plt.title('Green Channel')
    plt.subplot(313)
    plt.hist(b.flatten(), bins=256, range=[0, 256], color='b')
```

```
    plt.title('Blue Channel')
    plt.tight_layout()
    plt.show()

# membaca citra di dalam dataset
image = cv2.imread('dataset\\CitraRGB.jpg',0)

# Konversi citra dari BGR ke RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Tampilkan histogram
plot_histogram(image_rgb)
```

Praktik 5.3. Histogram Citra *Low Contrast*

Simpanlah kode program ini dengan nama **LowImageHistogram.py**.

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

def plot_histogram(image):
    # Hitung histogram citra
    hist, bins = np.histogram(image.flatten(), bins=256, range=[0, 256])

    # Plot histogram
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Citra Asli')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.plot(hist, color='black')
    plt.title('Histogram')
```

```
    plt.xlabel('Intensitas Piksel')
    plt.ylabel('Jumlah Piksel')
    plt.tight_layout()
    plt.show()

# Baca citra
image = cv2.imread('dataset\\lowcontrast.jpg',
cv2.IMREAD_GRAYSCALE)

# Tampilkan histogram citra low contrast
plot_histogram(image)
```

Praktik 5.4. Histogram Citra *High Contrast*
Simpanlah kode program ini dengan nama **HighImageHistogram.py**. Gantilah citra dengan dataset citra *high contrast* dengan menggunakan kode program yang sama pada Praktik 5.3.

Praktik 5.5. Histogram *Equalization*
Simpanlah kode program ini dengan nama **ImageEqualization.py**.

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

def histogram_equalization(image):
    # Konversi citra ke skala abu-abu jika citra berwarna
    if len(image.shape) > 2:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Hitung histogram citra
    hist, bins = np.histogram(image.flatten(), bins=256, range=[0, 256])

    # Hitung probabilitas distribusi piksel
```

```
    prob = hist / float(image.size)

    # Hitung nilai kumulatif distribusi probabilitas
    cum_prob = prob.cumsum()

    # Hitung transformasi histogram
    equalized_image = np.interp(image.flatten(), bins[:-1], cum_prob *
255)
    equalized_image =
equalized_image.reshape(image.shape).astype(np.uint8)

    # Hitung histogram citra hasil equalisasi
    equalized_hist, _ = np.histogram(equalized_image.flatten(), bins=256,
range=[0, 256])

    # Plot histogram sebelum dan setelah equalisasi
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.imshow(image, cmap='gray')
    plt.title('Citra Asli')
    plt.axis('off')
    plt.subplot(1, 2, 2)
    plt.imshow(equalized_image, cmap='gray')
    plt.title('Hasil Equalisasi Histogram')
    plt.axis('off')
    plt.tight_layout()

    # Plot histogram
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.plot(hist, color='black')
    plt.title('Histogram Sebelum Equalisasi')
```

```
    plt.xlabel('Intensitas Piksel')
    plt.ylabel('Jumlah Piksel')
    plt.subplot(1, 2, 2)
    plt.plot(equalized_hist, color='black')
    plt.title('Histogram Setelah Equalisasi')
    plt.xlabel('Intensitas Piksel')
    plt.ylabel('Jumlah Piksel')
    plt.tight_layout()

    plt.show()

# Baca citra
imalge = cv2.imread('dataset\\ImageEqualization.jpg',
cv2.IMREAD_GRAYSCALE)

# Proses histogram equalization
equalized_image = histogram_equalization(image)
```

5.7 Latihan

1. Apa manfaat dari penerapan histogram pada citra?
2. Sebutkan tujuan dari histogram *equalization* dan histogram *specification*!
3. Buatlah histogram dari citra dengan resolusi 4 x 4 piksel berikut:

122	123	124	124
121	121	123	125
121	120	119	126
120	120	119	126

4. Berikut ini adalah potongan citra dengan ukuran 10 x 10 piksel dengan rentang nilai 0-7:

6	6	6	7	7	7	7	7	7	7
6	6	7	7	7	7	7	7	7	7

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

a. Lakukanlah histogram *equalization* pada citra tersebut!
b. Lakukanlah histogram *specification* pada citra tersebut dengan spesifikasi: {0: 5%, 1: 5%, 2: 10%, 3: 15%, 4: 20%, 5: 25%, 6: 15%, 7: 5%}!

BAB 6

KONVOLUSI DAN FILTER SPASIAL

Capaian Pembelajaran:

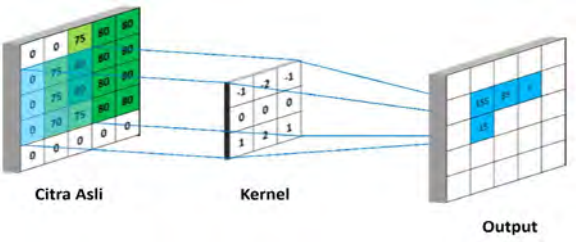
1. Mampu mengetahui, memahami, dan menjelaskan konsep dasar konvolusi citra
2. Mampu memahami dan melakukan operasi konvolusi citra.
3. Mampu mengetahui, memahami, dan menjelaskan frekuensi rendah dan tinggi pada citra.
4. Mampu mengetahui, memahami, dan menjelaskan penapisan dan penapisan spasial beserta jenisnya
5. Mampu mensimulasikan kode program konvolusi

6.1 Teori Konvolusi

Konvolusi pada citra *(Image Convolutional)* adalah teknik pemrosesan gambar yang penting dan umum digunakan dalam pengolahan citra digital. Konvolusi digunakan untuk mengubah suatu citra dengan menerapkan suatu kernel atau filter pada setiap piksel di dalamnya. Kernel adalah matriks kecil yang dapat digunakan untuk mengubah nilai piksel pada citra (Lihat Gambar 6.1). Tetapi dengan adanya konvolusi, ukuran dari citra tetap sama, tidak berubah.

Proses konvolusi melibatkan menggeser kernel di seluruh citra dengan tujuan untuk melakukan operasi matematis pada setiap piksel. Operasi ini menggabungkan nilai piksel dan bobot kernel yang sesuai untuk menghasilkan nilai piksel baru pada posisi yang sesuai di citra hasil konvolusi.

Contoh *filter* yang umum digunakan dalam konvolusi adalah *filter* rata-rata *(average filter)*, *filter gaussian*, *filter sharpening*, dan *filter* deteksi tepi *(edge detection)*. Setiap *filter* ini memiliki pengaruh yang berbeda pada citra, seperti penghalusan, pengasahan, atau penekanan fitur tertentu.



Gambar 6.1. Ilustrasi Proses Konvolusi

Konvolusi merupakan dasar dari banyak teknik pengolahan citra, termasuk deteksi objek, segmentasi citra, pengenalan pola, dan banyak lagi. Penerapan konvolusi memungkinkan kita untuk mengambil informasi penting dari citra dan menghasilkan representasi yang lebih berguna untuk analisis lebih lanjut atau pengolahan lanjutan.

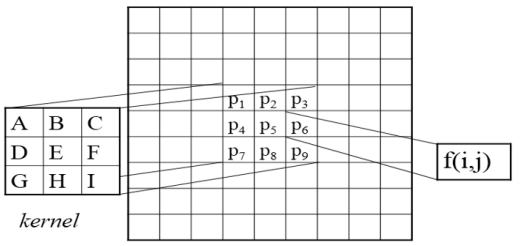
6.2 Operasi Konvolusi

Operasi konvolusi (Lihat Gambar 6.2) pada citra melibatkan tiga langkah utama:

Langkah 1: Tempatkan kernel pada titik piksel tertentu di dalam citra.

Langkah 2: Lakukan perhitungan hasil konvolusi dengan mengalikan setiap elemen kernel dengan nilai piksel yang sesuai dalam citra dan menjumlahkannya.

Langkah 3: Tempatkan hasil perhitungan di atas pada titik yang sesuai dalam citra hasil konvolusi.



$$f(i,j) = Ap_1 + Bp_2 + Cp_3 + Dp_4 + Ep_5 + Fp_6 + Gp_7 + Hp_8 + Ip_9$$

Gambar 6.2. Operasi Konvolusi

Contoh pengoperasian konvolusi pada citra:

Misalnya sebuah citra f dengan ukuran 6 x 6 piksel akan dikonvolusi menggunakan kernel g dengan ukuran 3 x 3 piksel.

$$f(x, y) = \begin{pmatrix} 4 & 7 & 1 & 9 & 8 & 5 \\ 3 & 8 & 5 & 1 & 3 & 9 \\ 2 & 2 & 8 & 2 & 5 & 7 \\ 3 & 3 & 4 & 6 & 7 & 9 \\ 2 & 5 & 5 & 1 & 4 & 8 \\ 7 & 3 & 6 & 3 & 8 & 5 \end{pmatrix}$$

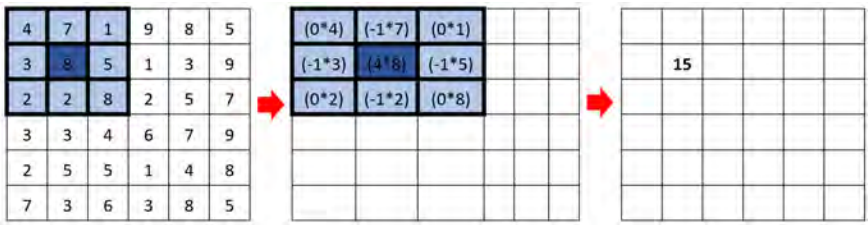
Citra Asli

$$g(x, y) = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Kernel

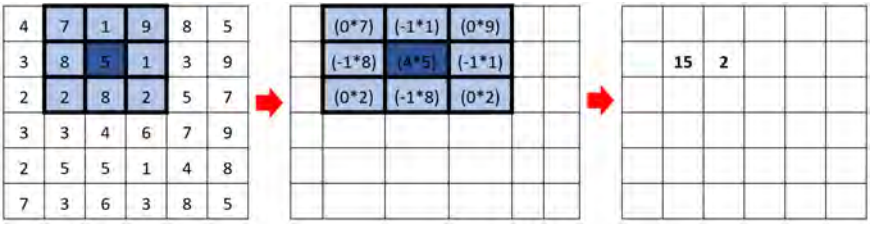
Penyelesaian:

Tempatkan kernel pada sudut kiri atas, kemudian hitung nilai piksel pada posisi (0, 0) dari kernel:



Hasil konvolusi iterasi pertama yaitu 15. Nilai ini diperoleh dengan menggunakan rumus *f(i, j)* = (0 x 4) + (-1 x 7) + (0 x 1) + (-1 x 3) + (4 x 8) + (-1 x 5) + (0 x 2) + (-1 x 2) + (0 x 8) = 15. Nilai 15 ini akan menggantikan nilai piksel pada citra asil yang sebelumnya adalah 8.

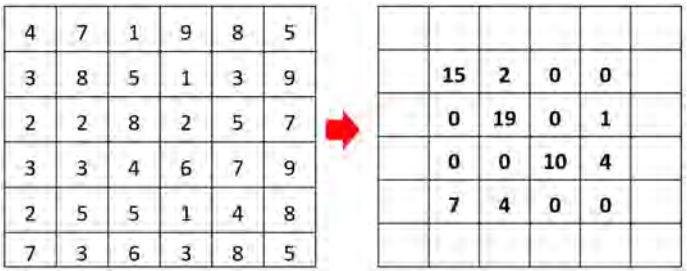
Iterasi berikutnya dengan cara menggeser kernel satu piksel ke kanan, kemudian lakukan perhitungan nilai piksel pada posisi (0, 0) dari kernel, proses sama seperti iterasi pertama:



Hasil konvolusi iterasi pertama yaitu 2. Nilai ini diperoleh dengan menggunakan rumus *f(i, j)* = (0 x 7) + (-1 x 1) + (0 x 9) + (-1 x 8) + (4 x 5) + (-1 x 1) + (0 x 2) + (-1 x 8) + (0 x 2) = 2. Nilai 2 ini akan menggantikan nilai piksel pada citra asil yang sebelumnya adalah 5.

Proses ini diulang untuk setiap piksel dalam citra, kecuali untuk piksel-piksel di tepi citra di mana kernel tidak sepenuhnya tertanam dalam citra. Biasanya, untuk menangani piksel-piksel di tepi ini, ada beberapa metode seperti *"zero-padding"* (menambahkan nilai piksel nol di luar tepi citra) atau *"mirror-padding"* (merefleksikan citra di tepiannya) yang digunakan. Penerapan *filter* konvolusi yang berbeda pada citra dapat menghasilkan efek yang berbeda, seperti penghalusan, deteksi tepi, atau penguatan fitur tertentu dalam citra. Setiap *filter* atau kernel memiliki bobot yang berbeda untuk mencapai efek yang diinginkan. Misalnya, untuk menghasilkan filter rata-rata, semua elemen kernel akan memiliki bobot yang sama dan dijumlahkan, sedangkan untuk *filter* deteksi tepi, bobotnya akan menonjolkan perbedaan nilai piksel di sekitar tepi objek.

Demikian juga untuk iterasi selanjutnya sampai akhir, dimana setiap nilai pada matriks dilakukan operasi konvolusi sehingga diperoleh keluaran berikut:



6.2 Citra Berfrekuensi Rendah dan Tinggi

Dalam konteks pengolahan citra, istilah "citra berfrekuensi rendah" dan "citra berfrekuensi tinggi" merujuk pada dua jenis citra yang dibedakan berdasarkan distribusi frekuensi spasialnya.

1. Citra Berfrekuensi Rendah: Citra berfrekuensi rendah merupakan citra dimana intensitas perubahan spasialnya lambat

(kecenderungannya bahwa di mana piksel-piksel pada citra memiliki kemiripan nilai dengan piksel tetangganya). Dalam citra berfrekuensi rendah, banyak informasi memiliki komponen rendah dan cenderung berubah secara perlahan dari piksel ke piksel. Citra berfrekuensi rendah umumnya terdiri dari gambar-gambar dengan elemen yang lembut, seperti citra yang terdiri dari banyak daerah berwarna seragam, halus, atau pola yang tidak berubah secara tiba-tiba.

2. Citra Berfrekuensi Tinggi: Citra berfrekuensi tinggi adalah citra dimana intensitas perubahan spasialnya tajam dan cepat (cenderungnya suatu piksel pada citra memiliki nilai dengan kemiripan yang jauh dengan piksel tetangganya). Dalam citra berfrekuensi tinggi, banyak informasi memiliki komponen frekuensi tinggi dan cenderung berubah secara tajam dari piksel ke piksel. Citra berfrekuensi tinggi umumnya terdiri dari gambar-gambar dengan tepi yang tajam, detil yang kompleks, atau kontras yang tinggi.

Pembedaan ini berdasarkan frekuensi spasial, yang menggambarkan bagaimana intensitas citra berubah dalam domain spasialnya (ukuran dan posisi piksel). Frekuensi spasial tinggi terkait dengan perubahan cepat dalam intensitas citra, sementara frekuensi spasial rendah terkait dengan perubahan lambat atau lebih halus.

Perbedaan ini menjadi sangat relevan dalam berbagai aplikasi pengolahan citra. Misalnya, dalam teknik penghalusan *(smoothing), filter* frekuensi rendah digunakan untuk menghilangkan noise atau menghaluskan citra. Sebaliknya, dalam deteksi tepi atau penguatan fitur, *filter* frekuensi tinggi digunakan untuk menyoroti tepi atau fitur penting dalam citra.

6.3 Filter dan Filter Spasial

Filter (penapisan) pada citra adalah suatu metode atau operasi pengolahan citra yang digunakan untuk mengubah atau memanipulasi informasi dalam citra dengan cara tertentu. *Filter* biasanya diterapkan dengan menggunakan kernel atau matriks kecil yang diaplikasikan pada

setiap piksel dalam citra. *Filter* ini berfungsi untuk mengubah nilai piksel atau fitur dalam citra, menghilangkan noise, meningkatkan kontras, mengidentifikasi tepi objek, dan banyak lagi.

Tujuan utama dari penggunaan *filter* pada citra adalah:

1. Penghalusan *(Smoothing)*: Mengurangi *noise* atau ketidaksempurnaan pada citra dengan menghaluskan intensitas piksel. Filter ini membantu menciptakan citra yang lebih bersih dan lebih mudah diolah.
2. Deteksi Tepi *(Edge Detection)*: Menyoroti dan mengidentifikasi tepi atau batas antara objek dalam citra. *Filter* ini membantu dalam deteksi fitur penting dalam citra, seperti tepi objek.
3. Penguatan Tepi *(Edge Enhancement)*: Meningkatkan kontras tepi dalam citra, membuat tepi objek menjadi lebih tajam dan mudah terlihat.
4. Peningkatan Kontras *(Contrast Enhancement)*: Meningkatkan seluruh kontras citra sehingga detail dan fitur dapat terlihat dengan lebih jelas.
5. Pengurangan atau Pencocokan Fitur *(Feature Reduction or Matching)*: Mengurangi dimensi fitur dalam citra, yang membantu dalam analisis dan pengenalan pola.
6. Penghilangan *Noise (Noise Removal)*: Menghilangkan komponen noise yang tidak diinginkan dari citra untuk mendapatkan hasil yang lebih bersih dan lebih akurat.
7. Segmentasi Citra *(Image Segmentation)*: Membagi citra menjadi beberapa bagian atau wilayah berdasarkan karakteristik tertentu seperti warna, tekstur, atau intensitas piksel.

Filter pada citra dapat diimplementasikan menggunakan berbagai teknik matematis seperti konvolusi, transformasi Fourier, atau teknik-teknik pemrosesan sinyal lainnya. Pilihan *filter* yang tepat bergantung pada tujuan pengolahan citra dan tugas spesifik yang ingin dicapai, sehingga hasilnya dapat lebih sesuai dengan kebutuhan analisis atau aplikasi tertentu.

Filter spasial pada citra merupakan suatu teknik pengolahan citra yang melibatkan perubahan atau manipulasi nilai piksel dalam citra berdasarkan

hubungan spasial (posisi piksel) dengan piksel-piksel sekitarnya. *Filter* spasial umumnya menggunakan suatu kernel atau matriks yang diaplikasikan pada citra untuk menghasilkan citra yang telah diproses.

Filter spasial diterapkan pada citra melalui proses konvolusi yang telah dijelaskan sebelumnya, yaitu dengan menggeser kernel pada setiap piksel dalam citra dan mengalikan nilai piksel dengan bobot kernel yang sesuai. Hasil dari operasi konvolusi adalah citra baru yang telah diproses sesuai dengan tujuan *filter* yang digunakan. Penerapan *filter* spasial merupakan langkah penting dalam pengolahan citra untuk menghasilkan citra yang lebih baik, meningkatkan kualitas visual, dan memfasilitasi analisis lebih lanjut dalam berbagai aplikasi, termasuk pengenalan objek, analisis medis, dan pengolahan gambar pada umumnya.

6.4 Lowpass Filter

Lowpass filter merupakan suatu proses pada citra untuk meloloskan data pada frekuensi rendah dan akan mengurangi atau menolak data pada frekuensi tinggi (Lihat Gambar 6.3). Salah satu contoh *lowpass filter* adalah *mean filter* dengan kernel berikut:

$$g = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

Dimana nilai *g(i, j)* haruslah bernilai positif dan jumlah dari semua nilai = 1. Tujuan dilakukan *lowpass filter* yaitu untuk menghilangkan *noise* dan membuat citra menjadi lebih halus. Contoh dari *filter* ini adalah *mean filter* dan *Gaussian filter*.

6.5 Highpass Filter

Highpass filter merupakan suatu proses pada citra untuk meloloskan data pada frekuensi tinggi dan akan mengurangi atau menolak data pada

frekuensi tinggi (Lihat Gambar 6.3). Untuk melakukan *highpass filter* menggunakan fungsi *filter* berikut:

$$g = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Dimana nilai *g(i, j)* dapat bernilai positif, negatif, atau nol serta jika dijumlah keseluruhan nilainya bernilai 0. *Filter* ini digunakan untuk melakukan deteksi tepi.

6.6 Bandpass Filter

Bandpass filter merupakan penapisan yang digunakan untuk meloloskan data pada frekuensi di rentang tertentu dan menolak data pada frekuensi di luar rentang tersebut.

$$g = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Dimana nilai *g(i, j)* dapat bernilai positif, negatif, atau 0 serta jika dijumlah keseluruhan nilainya tidak sama dengan 0. *Filter* ini digunakan untuk penajaman bagian tepi objek *(sharpening)*.

Gambar 6.3. *Lowpass* dan *Highpass Filtering* pada Citra *Grayscale*

Untuk praktik 6.1 silakan menggunakan citra yang dapat diunduh pada tautan berikut: https://bit.ly/CitraKonvolusi.

Praktik 6.1. Konvolusi Citra

Simpanlah kode program ini dengan nama **ConvolutionalImage.py**.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('dataset\\semangka.jpg',0)

kernel = np.ones((5,5),np.float32)/25
dst = cv.filter2D(img,-1,kernel)
plt.subplot(121),plt.imshow(img),plt.title('Citra Asli')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Konvolusi/Pelembutan Citra')
plt.xticks([]), plt.yticks([])
plt.show()
```

Praktik 6.2. Teori Konvolusi Citra

Simpanlah kode program ini dengan nama **ConvolutionalTheory.py.**

```
import cv2 as cv
import scipy.signal as sig
import numpy as np
b=np.asarray([[4,7,1,9],
              [3,55,5,1],
              [2,2,8,2],
              [3,3,4,6]
              ],dtype=np.uint8)

w=np.asarray([[0,-1],
              [-1,4]],dtype=np.uint8)
```

```
w_r=np.asarray([[0,-1],
            [-1,4]
              ],dtype=np.uint8)

print(sig.convolve2d(b,w,mode="same"))
kernel_r=np.asarray([[1,1,1],[1,1,2],[2,1,1]])
print("-------")
print(cv.filter2D(b,-1,w_r))
```

6.7 Latihan

1. Jelaskan konsep tentang konvolusi dan sebutkan manfaatnya dalam pengolahan citra digital!
2. Apa pengaruh frekuensi terhadap citra?
3. Sebutkan tujuan dari penapisan pada citra!
4. Sebutkan perbedaan antara *Lowpass* dan *Highpass Filtering*!
5. Diketahui citra asli dengan ukuran 5 x 5 piksel:

17	11	9	8	8
50	11	7	7	9
15	11	5	21	11
15	23	26	34	32
32	22	17	17	6

Lakukan operasi konvolusi dengan kernel berikut ini:

a. $\begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$ b. $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

BAB 7

TRANSFORMASI FOURIER

Capaian Pembelajaran:

1. Mampu mengetahui dan memahami operasi transformasi terkhusus transformasi fourier.
2. Mampu mengetahui dan memahami operasi fast fourier transform.
3. Mampu mensimulasikan kode program fast fourier transform.

7.1 Operasi Transformasi

Proses *image enhancement* berbasis transformasi citra dilakukan dengan:

a. Mentransformasi domain citra asal ke dalam domain lain yang sesuai bagi proses *enhancement*.
b. Melakukan proses *enhancement* pada domain baru tersebut.
c. Mengembalikan citra ke dalam domain spasial untuk ditampilkan/diproses lebih lanjut.

Salah satu metode transformasi yang populer dalam aplikasi pengolahan citra digital adalah *Fast Fourier Transform* (FFT).

7.2 Transformasi Fourier

Transformasi Fourier pada citra merupakan teknik penting dalam pengolahan citra yang memungkinkan kita untuk menganalisis citra dalam domain frekuensi. Transformasi Fourier mengubah citra dari domain spasial menjadi domain frekuensi, di mana informasi tentang berapa banyak frekuensi yang ada dalam citra diungkapkan dengan lebih jelas. Kita dapat mendigitalisasikan bentuk citra kontinue menjadi citra diskrit.

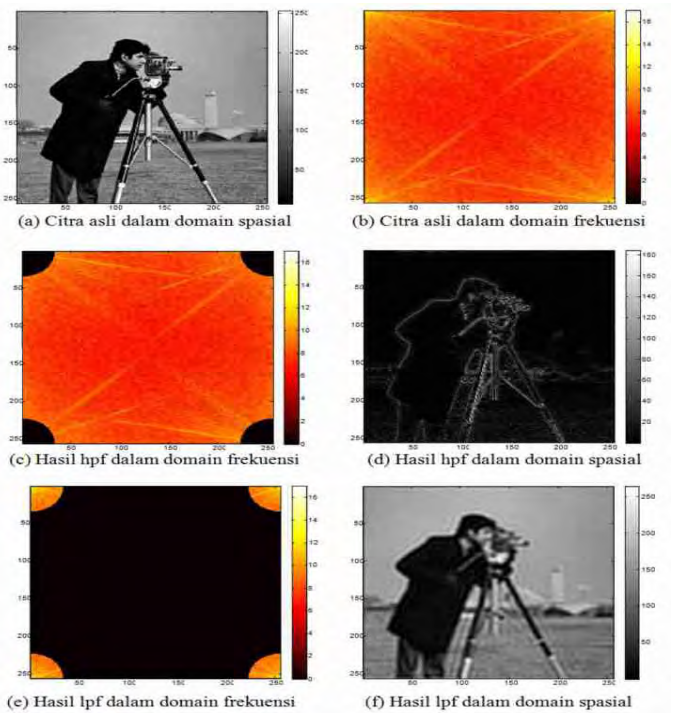
Proses Transformasi Fourier pada citra melibatkan penerapan *Discrete Fourier Transform* (DFT) atau FFT pada nilai piksel citra. DFT mengambil citra yang berupa matriks dari piksel dan menghitung koefisien frekuensi

yang menggambarkan bagaimana piksel berkontribusi terhadap frekuensi-frekuensi tertentu.

Dalam domain frekuensi, komponen rendah (frekuensi rendah) dari citra mewakili informasi mengenai gambaran besar atau komponen utama dari citra tersebut, seperti warna dasar atau bentuk objek besar. Sementara itu, komponen tinggi (frekuensi tinggi) menggambarkan detail atau fitur halus dalam citra, seperti tepi, garis, atau tekstur.

Beberapa hal yang dapat dilakukan dengan Transformasi Fourier pada citra termasuk:

1. *Lowpass Filtering*: Menghilangkan frekuensi tinggi dan mengurangi noise dalam citra untuk menghasilkan hasil yang lebih halus (Lihat Gambar 7.1).
2. *Highpass Filtering*: Menghilangkan frekuensi rendah dan meninggalkan detail atau fitur yang lebih tajam dalam citra (Lihat Gambar 7.1).



Gambar 7.1. FFT pada Citra *Grayscale*

3. Analisis Spektrum: Menganalisis spektrum frekuensi citra untuk mengidentifikasi frekuensi dominan atau fitur khusus dalam citra.
4. Rekonstruksi Citra: Mengembalikan citra dari domain frekuensi ke domain spasial setelah dilakukan operasi pengolahan dalam domain frekuensi.

Transformasi Fourier pada citra telah menjadi dasar bagi banyak teknik pengolahan citra yang lebih lanjut. Dengan menganalisis citra dalam domain frekuensi, kita dapat mengidentifikasi dan memanipulasi fitur-fitur penting dari citra dengan lebih efisien dan efektif. Transformasi Fourier menjadi salah satu alat yang sangat berguna dalam analisis dan pengolahan citra digital.

7.3 Fast Fourier Transform

FFT merupakan algoritme yang digunakan untuk menghitung DFT dengan cepat. DFT adalah metode yang digunakan untuk mengubah sinyal dari domain spasial (misalnya, data piksel pada citra) menjadi domain frekuensi. FFT sangat penting dalam pengolahan sinyal dan pengolahan citra karena dapat mempercepat proses perhitungan DFT secara signifikan. Fungsi utama dari FFT adalah untuk mengubah sinyal atau data dari domain spasial menjadi domain frekuensi. Dengan melakukan transformasi ini, kita dapat memahami dan menganalisis sinyal atau data dengan cara yang berbeda.

Beberapa fungsi utama dari FFT dalam pengolahan citra adalah:

1. *Filtering*: FFT memungkinkan kita untuk menerapkan filter pada citra dalam domain frekuensi. Dengan melakukan filtering di domain frekuensi, kita dapat dengan cepat menghapus frekuensi yang tidak diinginkan atau merubah fitur-fitur tertentu pada citra.
2. Deteksi Tepi: Dengan menggunakan FFT, kita dapat menyoroti tepi dan fitur penting lainnya dalam citra.

3. Penghalusan dan Penguatan Citra: FFT dapat digunakan untuk menghaluskan atau memperkuat citra dengan mengatur koefisien frekuensi tertentu.
4. Transformasi Kompresi Citra: FFT digunakan dalam beberapa teknik kompresi citra, seperti Transformasi Wavelet, untuk mengurangi jumlah data yang diperlukan untuk menyimpan citra.
5. Rekonstruksi Citra: Setelah dilakukan operasi dalam domain frekuensi, FFT dapat digunakan untuk mengembalikan citra dari domain frekuensi ke domain spasial.

Praktik 7.1. *Fast Fourier Transform 1*

Simpanlah kode program ini dengan nama **FourierTransform1.py**.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

#filter rata-rata sederhana tanpa parameter penskalaan
mean_filter = np.ones((3,3))

#membuat filter gaussian
x = cv.getGaussianKernel(5,10)
gaussian = x*x.T

#filter pendeteksi tepi dengan jenis yang berbeda-beda
#scharr (x)
scharr = np.array([[-3, 0, 3],
                   [-10,0,10],
                   [-3, 0, 3]])

#sobel (x)
sobel_x= np.array([[-1, 0, 1],
                   [-2, 0, 2],
```

```
                [-1, 0, 1]])

# sobel in y direction
sobel_y= np.array([[-1,-2,-1],
                [0, 0, 0],
                [1, 2, 1]])

# metode laplacian
laplacian=np.array([[0, 1, 0],
                 [1,-4, 1],
                 [0, 1, 0]])

filters = [mean_filter, gaussian, laplacian, sobel_x, sobel_y, scharr]
filter_name = ['mean_filter', 'gaussian','laplacian', 'sobel_x', \
            'sobel_y', 'scharr_x']
#variabel fft adalah Fast Fourier Transform pada citra
fft_filters = [np.fft.fft2(x) for x in filters]
fft_shift = [np.fft.fftshift(y) for y in fft_filters]
mag_spectrum = [np.log(np.abs(z)+1) for z in fft_shift]
for i in range(6):
   plt.subplot(2,3,i+1),plt.imshow(mag_spectrum[i],cmap = 'gray')
   plt.title(filter_name[i]), plt.xticks([]), plt.yticks([])
plt.show()
```

Untuk praktik 7.2 silakan menggunakan citra yang dapat diunduh pada tautan berikut: https://bit.ly/Fourier_Transform.

Praktik 7.2. *Fast Fourier Transform 2*

Simpanlah kode program ini dengan nama **FourierTransform2.py**.

```
import cv2 as cv
import numpy as np
```

```
from matplotlib import pyplot as plt

img = cv.imread('dataset\\sobel.jpg',0)
dft = cv.dft(np.float32(img),flags = cv.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)
magnitude_spectrum =
20*np.log(cv.magnitude(dft_shift[:,:,0],dft_shift[:,:,1]))

plt.subplot(121),plt.imshow(img, cmap = 'gray')
plt.title('Citra Masukan'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(magnitude_spectrum, cmap = 'gray')
plt.title('Hasil Magnitude Spectrum'), plt.xticks([]), plt.yticks([])

plt.show()
```

7.4 Latihan

1. Mengapa citra perlu untuk dilakukan operasi transformasi? Jelaskan!
2. Jelaskan secara ringkas sejarah dari transformasi fourier dan dimanfaatkan untuk kasus seperti apa saja?
3. Apa perbedaan antara *Discrete Fourier Transform* dan *Fast Fourier Transform*?

BAB 8

PERBAIKAN KUALITAS CITRA

Capaian Pembelajaran:

1. Mampu memahami dan menjelaskan konsep dan tujuan perbaikan kualitas citra.
2. Mampu memahami macam-macam filter untuk memperbaiki kualitas citra.
3. Mampu menerapkan filter-filter yang sudah tersedia di dalam kode program.
4. Mampu menerapkan filter-filter untuk mencapai suatu kualitas tampilan citra tertentu.
5. Mampu memahami contoh derau/noise.
6. Mampu memahami filter untuk digunakan dalam pengurangan derau/noise.

8.1 Konsep Perbaikan Kualitas Citra

Perbaikan kualitas citra merupakan proses mengoptimalkan atau meningkatkan kualitas gambar atau foto secara visual. Tujuan dari perbaikan kualitas citra adalah untuk mencapai hasil yang lebih baik dalam hal detail, ketajaman, kontras, warna, dan penampilan keseluruhan. Hal ini dapat dicapai dengan menggunakan berbagai teknik dan algoritme pemrosesan citra yang bertujuan untuk mengurangi distorsi atau gangguan yang mungkin ada dalam citra tersebut.

Berikut adalah beberapa tujuan utama dari perbaikan kualitas citra:

1. Meningkatkan ketajaman: Teknik-teknik seperti peningkatan tajam, atau pengurangan noise, dapat membantu meningkatkan kejelasan dan detail gambar sehingga objek dalam gambar menjadi lebih terlihat.
2. Mengurangi *noise*: *Noise* adalah gangguan acak yang dapat muncul dalam citra, misalnya pada gambar yang diambil dalam kondisi

pencahayaan rendah atau ketika menggunakan sensitivitas ISO tinggi. Mengurangi *noise* membantu menjaga atau mengembalikan detail asli yang mungkin tersembunyi oleh *noise* tersebut.

3. Koreksi warna: Teknik pemrosesan citra dapat digunakan untuk mengoreksi ketidaksempurnaan warna pada gambar, sehingga mencapai reproduksi warna yang lebih akurat dan realistis.
4. Peningkatan kontras: Peningkatan kontras membantu membedakan objek dari latar belakang dan membuat citra lebih menarik secara visual.
5. Restorasi citra: Jika citra mengalami kerusakan, seperti kerusakan fisik atau degradasi, perbaikan kualitas citra dapat membantu memulihkan atau merekonstruksi citra tersebut agar lebih mudah diinterpretasikan atau dianalisis.
6. Peningkatan deteksi dan pengenalan objek: Dalam aplikasi pengenalan objek atau pengolahan citra berbasis komputer, perbaikan kualitas citra dapat membantu meningkatkan akurasi sistem dalam mengenali dan mendeteksi objek tertentu dalam gambar.
7. Penyempurnaan visual: Perbaikan kualitas citra dapat membuat gambar atau foto terlihat lebih menarik dan estetis, misalnya dengan menghilangkan efek merah mata, memperbaiki ekspresi wajah, dan lain-lain.

Secara keseluruhan, perbaikan kualitas citra memiliki beragam aplikasi, mulai dari pemrosesan fotografi pribadi hingga penggunaan dalam industri, kedokteran, analisis citra, keamanan, dan berbagai bidang lainnya di mana kualitas visual dari gambar sangat penting.

Menurut (Hidayatullah, 2017) menyebutkan bahwa perbaikan kualitas citra mempunyai beberapa tujuan diantara:

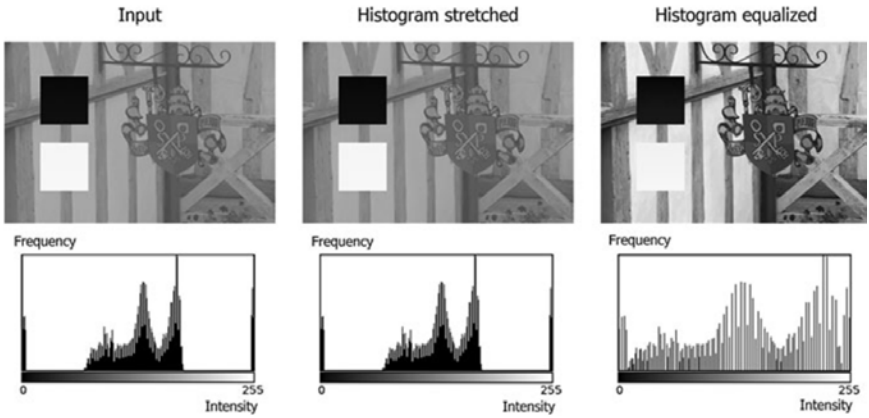
1. Agar citra mempunyai tampilan yang lebih baik menurut selera manusia.
2. Agar citra lebih mudah untuk dianalisis oleh proses otomatisasi berbasis citra.

3. Menghilangkan artefak-artefak pengganggu yang tidak diinginkan atau yang lebih dikenal dengan istilah derau.

Proses-proses yang termasuk ke dalam perbaikan citra yaitu perubahan kecerahan citra *(image brightness)*, perbaikan kontras, pengurangan derau, pelembutan citra *(image smoothing)*, dan penajaman citra *(image sharpening)*.

8.2 Perbaikan Kontras

Perbaikan kontras dapat dilakukan dengan beberapa teknik. Yang pertama adalah histogram *stretching*. Histogram ditarik sedemikian rupa sehingga lebih merata distribusinya. Teknik yang kedua adalah histogram *equalization*. Mirip dengan teknik pertama, teknik ini juga bertujuan membuat distribusi pada histogramnya tersebar lebih merata (Lihat Gambar 8.1).



Gambar 8.1. Perbaikan Kontras

8.3 Pengurangan Derau (Noise)

Noise atau derau secara sederhana merupakan suatu gangguan atau kecacatan pada citra yang mengakibatkan citra tidak jelas dan kehilang informasi. Pengurangan *noise* pada citra adalah proses menghilangkan atau mengurangi gangguan acak *(noise)* yang terdapat pada gambar atau citra

digital. *Noise* adalah sinyal acak yang diakibatkan oleh berbagai faktor diantaranya:

1. Proses akuisisi citra yang kurang baik, seperti kurang atau lebihnya cahaya yang tertangkap.
2. Sensor yang digunakan.
3. Lingkungan, seperti terdapat awan ketika pengambilan citra melalui citra satelit.
4. Transmisi, terjadi *data loss* ketika pengiriman citra melalui sebuah sinyal yang mengakibatkan citra yang diterima kurang atau tidak sama dengan citra yang dikirim.

Noise dapat menyebabkan gambar terlihat buram, mengaburkan detail, dan mengurangi kualitas gambar secara keseluruhan. Manfaat dari pengurangan *noise* pada citra antara lain:

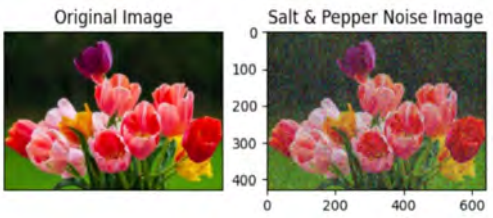
1. Peningkatan kualitas gambar: Dengan mengurangi *noise*, gambar akan terlihat lebih tajam dan jelas, karena detail yang tersembunyi oleh *noise* dapat diperjelas.
2. Meningkatkan ketepatan analisis: Ketika menggunakan citra untuk analisis atau pengolahan lebih lanjut, seperti deteksi objek atau pengenalan pola, mengurangi *noise* dapat membantu memastikan hasil analisis yang lebih akurat.
3. Mengurangi gangguan visual: *Noise* dapat mengganggu pengalaman visual pengamat saat melihat gambar. Dengan menghilangkan *noise*, citra akan terlihat lebih bersih dan lebih mudah dipahami.
4. Peningkatan hasil akhir: Jika gambar tersebut akan dicetak atau digunakan dalam publikasi, pengurangan *noise* akan meningkatkan kualitas hasil akhir, menjadikan gambar lebih menarik dan profesional.
5. Peningkatan efisiensi kompresi: Sebelum melakukan kompresi gambar untuk menyimpannya dalam format *file* yang lebih kecil, pengurangan *noise* dapat membantu meningkatkan efisiensi kompresi, karena *noise* yang berlebihan seringkali tidak berguna dan hanya meningkatkan ukuran *file*.

6. Perbaikan pengenalan pola: Pengurangan *noise* juga dapat membantu sistem pengenalan pola, seperti OCR, mengenali karakter atau bentuk dengan lebih baik dan akurat.

Pengurangan *noise* pada citra bisa dilakukan menggunakan berbagai teknik pemrosesan citra, seperti *filter* median, *filter* Gaussian, metode *denoising* berbasis transformasi (seperti Transformasi Gelombang Diskrit), atau menggunakan teknik pembelajaran mesin seperti penggunaan jaringan syaraf tiruan untuk *denoising* citra. Pemilihan metode yang tepat tergantung pada jenis *noise* yang ada dalam citra dan tujuan dari pengurangan *noise* yang diinginkan.

Berdasarkan bentuk dan karakteristiknya derau terbagi menjadi tiga yaitu:

1. **Derau *Salt and Pepper*:** Derau ini juga dikenal sebagai derau impuls adalah salah satu jenis *noise* yang umum terjadi pada gambar atau citra digital. Derau ini ditandai dengan kemunculan piksel dengan nilai intensitas yang ekstrem seperti adanya bintik-bintik pada citra, yaitu piksel dengan nilai paling rendah (seringkali disimbolkan sebagai "garam" atau "*salt*") atau piksel dengan nilai paling tinggi (disimbolkan sebagai "lada" atau "*pepper*") (Lihat Gambar 8.2).



Gambar 8.2. Citra dengan Derau *Salt and Pepper*

Untuk praktik 8.1 sampai dengan 8.6 silakan menggunakan citra yang dapat diunduh pada tautan berikut: https://bit.ly/Citra_Noise.

Praktik 8.1. Derau *Salt and Pepper*
Simpanlah kode program ini dengan nama **SaltandPepperNoise.py**.

```
import cv2
import numpy as np
from skimage.util import random_noise
from matplotlib import pyplot as plt

# Load citra yang akan digunakan
img = cv2.imread('dataset\\BungaTulip.jpg')
ori_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Menambahkan salt & pepper noise ke citra original.
noise_img = random_noise(ori_img, mode='s&p', amount=0.3)

# Fungsi diatas menghasilkan citra dengan nilai float
# yang berada pada rentang nilai [0,1], sehingga
# perlu diubah menjadi format uint8 dengan rentang
# nilai [0,255]
noise_img = np.array(255 * noise_img, dtype='uint8')

# Menampilkan citra dengan noise
plt.subplot(121), plt.imshow(ori_img), plt.title('Citra Asli')
plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(noise_img), plt.title('Citra dengan Derau
Salt & Pepper')
plt.show()
```

2. **Derau Gaussian:** Derau yang terdistribusi pada citra dengan distribusi normal (Gaussian), dimana kepadatan deraunya bergantung pada fungsi kepadatan probabilitas. Derau ini disebabkan oleh pencahayaan yang kurang, suhu yang sangat tinggi, dan kesalahan dalam transmisi citra.

Praktik 8.2. Derau Gaussian
Simpanlah kode program ini dengan nama **GaussianNoise.py**.

```
import cv2
import numpy as np
from skimage.util import random_noise
from matplotlib import pyplot as plt

# Load citra yang akan digunakan
img = cv2.imread('dataset\\BungaTulip.jpg')
ori_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Menambahkan gaussian noise ke citra original.
noise_img = random_noise(ori_img, mode='gaussian')

# Fungsi diatas menghasilkan citra dengan nilai float
# yang berada pada rentang nilai [0,1], sehingga
# perlu diubah menjadi format uint8 dengan rentang
# nilai [0,255]
noise_img = np.array(255 * noise_img, dtype='uint8')

# Menampilkan citra dengan noise
plt.subplot(121), plt.imshow(ori_img), plt.title('Original Image')
plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(noise_img), plt.title('Gaussian Noise Image')
plt.show()
```

3. **Derau Periodik:** Jenis derau atau kebisingan yang muncul secara berulang atau berpola pada citra. Derau periodik dapat menyebabkan distorsi atau ketidakjelasan pada gambar dan dapat mengganggu analisis atau pengolahan lebih lanjut pada citra tersebut (Lihat Gambar 8.3). Derau periodik dapat disebabkan oleh berbagai faktor, seperti gangguan elektronik dalam perangkat perekam, ketidaksempurnaan dalam perangkat keras citra, atau gangguan sinyal saat mentransfer atau

menyimpan citra. Untuk mengatasi derau ini dapat menggunakan median *filter* atau Transformasi Fourier.

Gambar 8.3. Derau Periodik pada Citra Lena

8.3.1. Median Filter

Median filter merupakan salah satu teknik pemrosesan citra yang digunakan untuk mengatasi derau atau kebisingan dalam citra. Teknik ini bertujuan untuk menghilangkan derau atau *noise* dari citra dengan cara menggantikan nilai piksel yang terpengaruh oleh derau dengan nilai median dari lingkungan piksel tertentu. *Median filter* sangat efektif dalam mengurangi jenis derau yang bersifat impulsif (dikenal juga sebagai *"Salt-and Pepper Noise"*), dimana piksel tertentu dalam citra mengalami perubahan drastis nilainya menjadi sangat terang *(salt)* atau sangat gelap *(pepper)*. *Filter* ini juga dapat membantu menghilangkan beberapa jenis derau periodik, seperti derau gelombang atau derau listrik dalam citra.

Praktik 8.3. *Median Filter*

Simpanlah kode program ini dengan nama **MedianFilter.py**.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('dataset\\NoiseImage.jpg')

blur = cv2.medianBlur(img,5)
```

```
plt.subplot(121),plt.imshow(img),plt.title('Citra Asli')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Citra Hasil Median Filter')
plt.xticks([]), plt.yticks([])
plt.show()
```

8.3.2. Mean Filter

Mean filter merupakan salah satu teknik pemrosesan citra yang digunakan untuk mengatasi derau atau kebisingan dalam citra. Prinsip dasar *mean filter* adalah menggantikan nilai piksel yang terpengaruh oleh derau dengan nilai rata-rata dari lingkungan piksel tertentu.

Mean filter cocok untuk mengatasi derau yang bersifat Gaussian (distribusi normal) atau derau yang memiliki karakteristik seperti *white noise* karena derau tersebut memiliki distribusi yang cenderung acak dan merata. Dengan mengambil rata-rata dari lingkungan piksel, nilai derau dapat dihaluskan sehingga citra terlihat lebih jernih. Namun, *mean filter* kurang efektif dalam mengatasi derau impulsif seperti *Salt and Pepper Noise* karena nilai rata-rata dapat sangat dipengaruhi oleh nilai ekstrim seperti piksel yang sangat terang atau sangat gelap.

Praktik 8.4. *Mean Filter*

Simpanlah kode program ini dengan nama **MeanFilter.py**.

```
import numpy as np
from matplotlib import pyplot as plt
import cv2

ori_img = cv2.imread('dataset\\GaussianNoiseImage.jpg')



def box_kernel(size):
    k = np.ones((size, size), np.float32) / (size ** 2)
    return k
```

```
kernel_size = 5
filtered_img = cv2.filter2D(ori_img, -1, box_kernel(kernel_size))

plt.subplot(121), plt.imshow(ori_img), plt.title('Citra Asli')
plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(filtered_img), plt.title('Citra yang
Diperhalus')
plt.xticks([]), plt.yticks([])
plt.show()
```

8.3.3. Gaussian Filter

Gaussian filter merupakan salah satu jenis *filter* yang digunakan dalam pengolahan citra untuk mengurangi derau atau kebisingan. *Filter* ini didasarkan pada fungsi Gaussian yang merupakan fungsi matematis. Fungsi Gaussian digunakan untuk menghasilkan kernel atau *mask* yang akan diterapkan pada citra untuk memperhalus atau mengaburkan gambar. *Filter* ini beroperasi dengan cara melakukan konvolusi citra dengan kernel Gaussian dengan ukuran tertentu dari pojok kiri atas sampai dengan pojok kanan bawah citra.

Filter ini cocok untuk mengatasi derau yang bersifat Gaussian (distribusi normal) atau derau yang cenderung merata. Derau Gaussian sering kali terjadi secara alami dalam berbagai situasi dan memiliki karakteristik distribusi yang halus. Namun, *Gaussian filter* juga dapat digunakan untuk mengatasi beberapa jenis derau non-Gaussian, terutama ketika parameter sigma diatur dengan bijak. *Filter* ini membantu menghaluskan gambar, mengurangi ketajaman detail, dan memperhalus transisi antara objek dalam citra.

Gaussian filter tidak efektif dalam mengatasi derau impulsif seperti *Salt and Pepper Noise* karena distribusi Gaussian tidak cocok untuk

mengidentifikasi atau menangani nilai ekstrim seperti nilai piksel yang sangat terang atau sangat gelap.

Praktik 8.5. *Gaussian Filter*

Simpanlah kode program ini dengan nama **GaussianFilter.py**.

```
import numpy as np
from matplotlib import pyplot as plt
import cv2

img = cv2.imread('dataset\\GaussianNoiseImage.jpg')
ori_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)



def gaussian_kernel(size, sigma=1):
    size = int(size) // 2
    x, y = np.mgrid[-size:size + 1, -size:size + 1]
    normal = 1 / (2.0 * np.pi * sigma ** 2)
    g = np.exp(-((x ** 2 + y ** 2) / (2.0 * sigma ** 2))) * normal
    return g



sigma = 1
size = 5
filtered_img_1 = cv2.filter2D(ori_img, -1, gaussian_kernel(size, sigma))

filtered_img_2 = cv2.GaussianBlur(ori_img, (size, size), 0)



plt.subplot(131), plt.imshow(ori_img), plt.title('Citra Asli')
plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(filtered_img_1), plt.title('Penghalusan Citra
Tahap 1')
```

```
plt.xticks([]), plt.yticks([])
plt.subplot(133), plt.imshow(filtered_img_2), plt.title('Penghalusan Citra Tahap 2')
plt.xticks([]), plt.yticks([])
plt.show()
```

8.4 Penajaman Citra

Penajaman citra merupakan salah satu bentuk *bandpass filter*. Itu artinya adalah yang diloloskan oleh *filter* ini merupakan citra berfrekuensi tertentu. Pada penajaman citra maka tepi-tepi objek yang ada di dalam citra akan terlihat jelas atau lebih tegas. Penajaman citra dapat dilakukan dengan melakukan konvolusi menggunakan kernel *bandpass filter*.

Praktik 8.6. Penajaman Citra

Simpanlah kode program ini dengan nama **ImageSharpening.py**.

```
import cv2
import numpy as np

def denoise_median(image, window_size):
    return cv2.medianBlur(image, window_size)

def denoise_gaussian(image, kernel_size, sigma):
    return cv2.GaussianBlur(image, (kernel_size, kernel_size), sigma)

# Ubah "nama_citra.jpg" sesuai dengan nama file citra yang ingin Anda proses
nama_file_citra = "dataset\\CitraLena.jpg"

# Baca citra menggunakan OpenCV
citra_asli = cv2.imread(nama_file_citra, cv2.IMREAD_COLOR)

# Konversi citra menjadi grayscale
```

```
citra_gray = cv2.cvtColor(citra_asli, cv2.COLOR_BGR2GRAY)

# Atur ukuran jendela median filter (gunakan bilangan ganjil)
window_size_median = 5

# Lakukan denoising menggunakan Median Filter
citra_hasil_median = denoise_median(citra_gray, window_size_median)

# Atur ukuran kernel dan nilai sigma untuk Gaussian Filter
kernel_size_gaussian = 5
sigma = 1.5

# Lakukan denoising menggunakan Gaussian Filter
citra_hasil_gaussian = denoise_gaussian(citra_gray, kernel_size_gaussian,
sigma)

# Tampilkan citra-citra hasil denoising
cv2.imshow("Citra Asli", citra_gray)
cv2.imshow("Citra Hasil Denoising (Median)", citra_hasil_median)
cv2.imshow("Citra Hasil Denoising (Gaussian)", citra_hasil_gaussian)

# Tunggu hingga tombol keyboard ditekan dan tutup jendela jika ditekan
cv2.waitKey(0)
cv2.destroyAllWindows()
```

8.5 Latihan

1. Mengapa citra perlu ditingkatkan kualitasnya dan mengapa derau yang diupayakan untuk dikurangi atau bahkan dihilangkan? Jelaskan!
2. Apa perbedaan antara *Salt and Pepper Noise* dengan *Gaussian Noise*?
3. Kapan menggunakan *Median Filter, Mean Filter,* dan *Gaussian Filter*? Apa saja syarat citra derau yang harus dipenuhi pada masing-masing *filter* agar kualitas citra menjadi lebih baik dan efektif?

BAB 9

OPERASI MORFOLOGI

Capaian Pembelajaran:

1. Mampu mengetahui, memahami, dan menjelaskan konsep operasi morfologi.
2. Mampu menerapkan operasi morfologi pada citra melalui kode program.

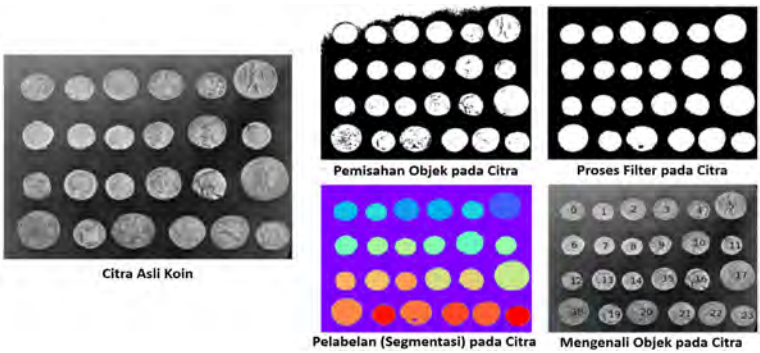
9.1 Konsep Operasi Morfologi

Operasi morfologi merupakan serangkaian teknik pengolahan citra yang dilakukan pada piksel-piksel citra berdasarkan bentuk, ukuran, dan struktur objek dalam citra. Teknik ini berfokus pada manipulasi bentuk dan geometri piksel untuk mengubah atau meningkatkan fitur-fitur objek dalam citra serta dalam rangka meningkatkan kualitas citra. Operasi morfologi merupakan operasi yang umum dikenakan pada citra biner (hitam-putih) untuk mengubah struktur bentuk objek yang terkandung dalam citra. Tujuan dari operasi morfologi adalah untuk memperbaiki hasil segmentasi.

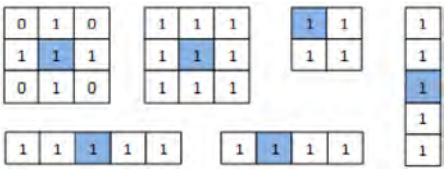
Contoh pengaplikasian dari operasi morfologi (Lihat Gambar 9.1) diantaranya: menutup lubang pada citra, memisahkan objek, membentuk *filter* spasial, memperoleh skeleton (rangka) objek, menentukan letak objek di dalam citra, dan memperoleh bentuk struktur objek. Beberapa operasi morfologi umum yang sering digunakan adalah Dilasi *(Dilation)* dan Erosi *(Erosion)*, serta kombinasi kedua operasi ini seperti *Opening* dan *Closing*.

Operasi morfologi umumnya melibatkan dua masukan yaitu:

- Citra masukan yang akan dikenai operasi morfologi
- Kernel atau *structuring element* dengan sebuah titik pusat *(hotspot)*, contoh kernel dengan titik pusat silakan lihat Gambar 9.2.



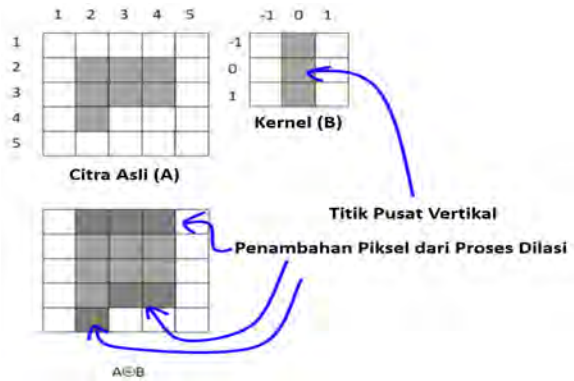
Gambar 9.1. Contoh Penerapan Operasi Morfologi



Gambar 9.2. Kernel dengan Sebuah Titik Pusat

9.2 Operasi Dilasi

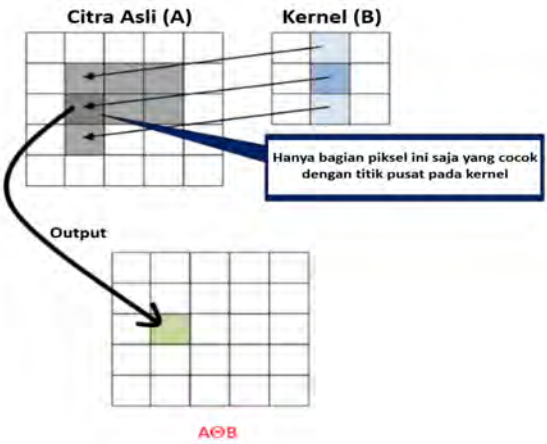
Dilasi *(Dilation)* mengembangkan area objek dalam citra. Operasi ini dilakukan dengan melebarkan batas objek dengan menggunakan elemen struktur (kernel) tertentu (Lihat Gambar 9.3). Hasilnya adalah objek menjadi lebih besar dan lebih tebal. Dilasi berguna untuk mengisi celah-celah kecil antara bagian-bagian objek atau untuk menghubungkan objek yang hampir saling berdekatan.



Gambar 9.3. Proses Dilasi

9.3 Operasi Erosi

Erosi *(Erosion)* menyusutkan area objek dalam citra. Operasi ini dilakukan dengan menyusutkan batas objek menggunakan elemen struktur (kernel) tertentu. Hasilnya adalah objek menjadi lebih kecil dan lebih tipis. Erosi berguna untuk menghilangkan tepi-tepi yang tidak diinginkan atau memisahkan objek yang saling berhubungan (Lihat Gambar 9.4).



Gambar 9.4. Proses Erosi

9.4 Operasi Opening

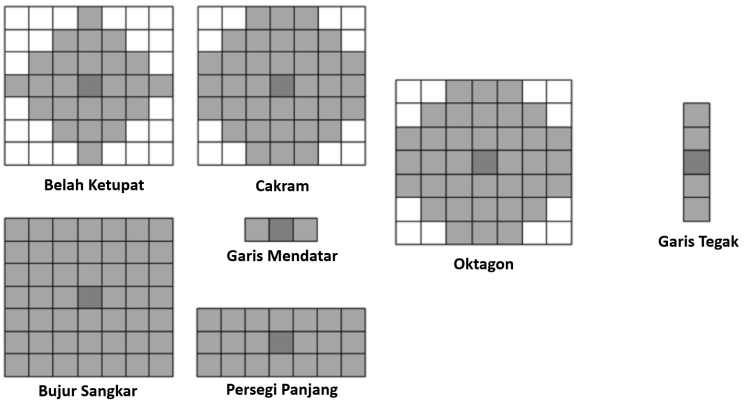
Opening adalah gabungan dari operasi Erosi diikuti oleh Dilasi. Operasi ini berguna untuk menghilangkan *noise* pada citra dan menghaluskan tepi objek tanpa mengubah ukuran objek utama.

9.5 Operasi Closing

Closing adalah gabungan dari operasi Dilasi diikuti oleh Erosi. Operasi ini berguna untuk mengisi celah-celah kecil dalam objek atau menghubungkan bagian-bagian objek yang terputus.

9.6 Elemen Penstruktur

Ukuran dan bentuk elemen penstruktur *(structure element)* juga menentukan hasil operasi morfologi. Bentuk yang umum digunakan pada operasi morfologi adalah cakram dan lingkaran (Lihat Gambar 9.5).



Gambar 9.5. Elemen Penstruktur pada Morfologi

Untuk praktik 9.1 sampai dengan 9.8 silakan menggunakan citra yang dapat diunduh pada tautan berikut: https://bit.ly/Citra_Morfologi.

Praktik 9.1. Erosi

Simpanlah kode program ini dengan nama **Erosion.py**.

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

img = cv2.imread('dataset\\DilasidanErosiCitra.jpg', 0)

# binerkan citra
biner = cv2.threshold(img, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

# menentukan kernel
kernel = np.ones((5, 5), np.uint8)

# membalikkan citra
invert = cv2.bitwise_not(biner)
```

```
# mengikis citra
erosion = cv2.erode(invert, kernel,
                                              iterations=1)

plt.imshow(erosion, cmap='gray')
```

Praktik 9.2. Dilasi

Simpanlah kode program ini dengan nama **Dilation.py**.

```
import cv2

img = cv2.imread('dataset\\DilasidanErosiCitra.jpg', 0)

biner = cv2.threshold(img, 0, 255,
cv.THRESH_BINARY+cv.THRESH_OTSU)[1]

kernel = np.ones((3, 3), np.uint8)

invert = cv2.bitwise_not(biner)

dilation = cv2.dilate(invert, kernel, iterations=1)

plt.imshow(dilation, cmap='gray')
```

Praktik 9.3. Erosi dan Dilasi

Simpanlah kode program ini dengan nama **ErosionDilation.py**.

```
import cv2
import numpy as np

img = cv2.imread('dataset\\DilasidanErosiCitra.jpg',0)

# Mengambil matriks ukuran 5x5 sebagai kernel
kernel = np.ones((5,5), np.uint8)
```

```
#proses erosi dengan 1 kali iterasi pengikisan
img_erosion = cv2.erode(img, kernel, iterations=1)

#proses dilasi dengan 1 kali iterasi pengikisan
img_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Citra Asli', img)
cv2.imshow('Hasil Erosion', img_erosion)
cv2.imshow('Hasil Dilation', img_dilation)

cv2.waitKey(0)
```

Praktik 9.4. *Opening*

Simpanlah kode program ini dengan nama **OpeningMorphology.py**.

```
import cv2

img = cv2.imread('dataset\\ImageNoise.jpg', 0)

biner = cv2.threshold(img, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

kernel = np.ones((3, 3), np.uint8)

opening = cv2.morphologyEx(biner, cv2.MORPH_OPEN, kernel,
iterations=1)

plt.imshow(opening, cmap='gray')
```

Praktik 9.5. *Closing*

Simpanlah kode program ini dengan nama **ClosingMorphology.py**.

```
import cv2
```

```
img = cv2.imread('dataset\\ImageNoise.jpg', 0)

binr = cv2.threshold(img, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

kernel = np.ones((3, 3), np.uint8)

closing = cv2.morphologyEx(binr, cv2.MORPH_CLOSE, kernel,
iterations=1)

plt.imshow(closing, cmap='gray')
```

Praktik 9.6. Gradien Morfologi

Simpanlah kode program ini dengan nama **MorphologyGradient.py**.

```
import cv2

img = cv2.imread('dataset\\ImageNoise.jpg', 0)

binr = cv2.threshold(img, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]

kernel = np.ones((3, 3), np.uint8)

invert = cv2.bitwise_not(binr)

# menggunakan morph gradient
morph_gradient = cv2.morphologyEx(invert, cv2.MORPH_GRADIENT,
kernel)

plt.imshow(morph_gradient, cmap='gray')
```

Praktik 9.7. Penstrukturan Elemen

Simpanlah kode program ini dengan nama **StructureElementMorphology.py**.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from skimage.morphology import diamond # import diamond pada
skimage morfologi

# Melakukan generate struktur elemen untuk 2D diamond
struc_2d = {
    "diamond(1)": diamond(1),
    "diamond(2)": diamond(2),
    "diamond(3)": diamond(3),
    "diamond(4)": diamond(4),
    "diamond(5)": diamond(5),
    "diamond(6)": diamond(6)}

# Memvisualisasikan masing-masing elemen
fig = plt.figure(figsize=(10,6))

idx = 1
for title, struc in struc_2d.items():
    ax = fig.add_subplot(2, 3, idx)
    ax.imshow(struc, cmap="summer", vmin=0, vmax=2,zorder=2)
    for i in range(struc.shape[0]):
        for j in range(struc.shape[1]):
            ax.text(j, i, struc[i, j], ha="center", va="center",
color="w",zorder=3)
    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
```

```
    ax.grid()
    ax.set_axisbelow(True)
    ax.set(xlim=(-1,struc.shape[0]),ylim=(-1,struc.shape[0]))
    ax.set_title(title)
    idx += 1

fig.tight_layout()
plt.savefig('diamond.png',dpi=100) # simpan gambar dalam bentuk file
.png
plt.show()
```

Praktik 9.8. Skeleton

Simpanlah kode program ini dengan nama **SkeletonMorphology.py**.

```
from skimage.morphology import skeletonize
from skimage import data
import matplotlib.pyplot as plt
from skimage.util import invert

# Akuisisi data gambar kuda
image = invert(data.horse())

# pengaturan kinerja skeletonization
skeleton = skeletonize(image)

# memunculkan visualisasi
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(8, 4),
                         sharex=True, sharey=True)

ax = axes.ravel()

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].axis('off')
```

```
ax[0].set_title('citra asli', fontsize=20)

ax[1].imshow(skeleton, cmap=plt.cm.gray)
ax[1].axis('off')
ax[1].set_title('citra skeleton', fontsize=20)

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

9.7 Latihan

1. Operasi morfologi pada pengolahan citra digital ditujukan untuk apa saja? Jelaskan dan sebutkan!
2. Sebutkan pengaplikasian operasi morfologi yang secara umum dapat dilakukan!
3. Apa perbedaan antara dilasi, erosi, *opening*, dan *closing*?
4. Apa tujuan dari citra dijadikan dalam bentuk skeleton?

BAB 10

DETEKSI TEPI

Capaian Pembelajaran:

1. Mampu memahami dan menjelaskan konsep pendeteksian tepi.
2. Mampu memahami dan menjelaskan jenis-jenis pendeteksian tepi.
3. Mampu menerapkan pendeteksian tepi menggunakan berbagai jenis deteksi tepi melalui kode program.

10.1 Pendeteksian Tepi

Pada dasarnya pendeteksian tepi ini merupakan pemrosesan citra berbasis biner. Penentuan tepian suatu objek dalam citra merupakan salah satu wilayah pengolahan citra digital yang paling awal dan paling banyak diteliti. Proses ini seringkali ditempatkan sebagai langkah pertama dalam aplikasi segmentasi citra, yang bertujuan untuk mengenali objek-objek yang terdapat dalam citra ataupun konteks citra secara keseluruhan. Deteksi tepi *(edge detection)* berfungsi untuk mengidentifikasi garis batas *(boundary)* dari suatu objek yang terdapat pada citra.

Pendeteksian tepi merupakan teknik untuk menemukan garis tepi dari suatu objek pada citra dengan cara mendeteksi perubahan tingkat kecerahan yang signifikan atau mempunyai diskontinuitas.

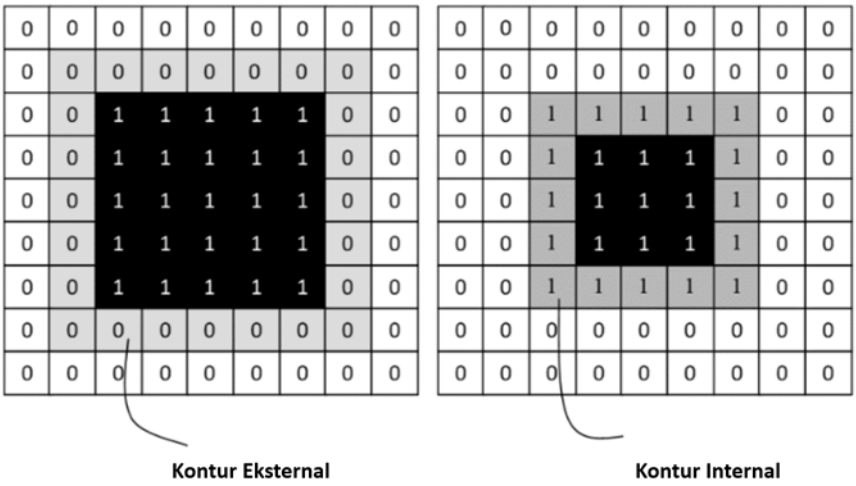
Terdapatnya diskontinuitas lokal dalam nilai piksel yang melebihi ambang batas *(threshold)* yang diberikan disebut sebagai tepi *(edge)*. Tepi juga bisa didefinisikan sebagai posisi citra dimana terjadi perubahan intensitas lokal yang terlihat jelas di sepanjang orientasi tertentu. Semakin besar perubahan intensitas lokal, maka semakin tinggi bukti yang menyatakan terdapat suatu tepi pada posisi tersebut. Tepi di suatu citra dapat dilihat dengan mengamati perbedaan nilai pikselnya.

Pendeteksian tepi dapat dilakukan dengan menggunakan *highpass filter* karena tepi termasuk ke dalam bagian citra berfrekuensi tinggi.

Pendeteksian tepi bisa digunakan untuk segmentasi citra dan ekstraksi data untuk keperluan pengolahan citra, visi komputer, dan visi mesin (Luhat Gambar 10.1 terkait dengan contoh pendeteksian tepi).

Gambar 10.1. Contoh Pengaplikasian Deteksi Tepi

Konsep dari algoritme deteksi tepi pada dasarnya memanfaatkan 8 ketetanggaan. Jika sekeliling piksel P bernilai sama (semua=1 atau semua=0), maka dapat diasumsikan bahwa piksel P tidak berada di tepi objek. Pada Gambar 10.3 merupakan deteksi tepi yang mengikuti kontur.



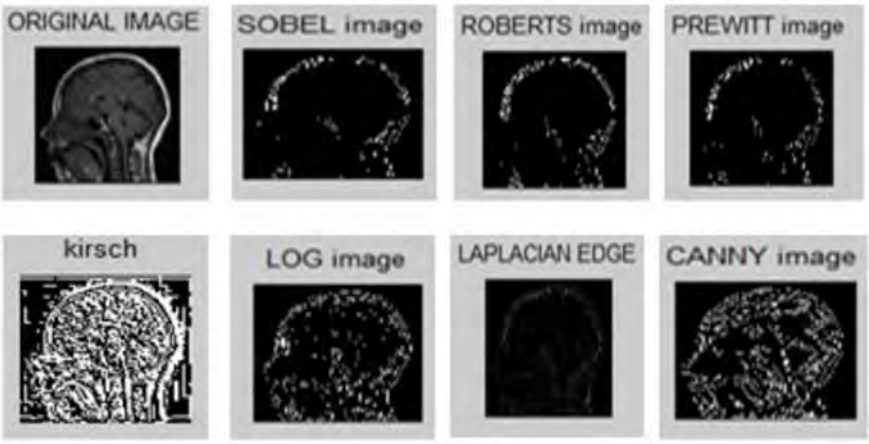
Gambar 10.2. Deteksi Tepi yang Mengikuti Kontur

10.2 Jenis-Jenis Pendeteksian Tepi

Ada beberapa metode yang umum digunakan untuk mendeteksi tepi dalam pengolahan citra digital. Berikut adalah beberapa di antaranya:

1. Metode Deteksi Tepi Sobel: Metode ini menggunakan operator Sobel untuk mendeteksi tepi dengan menghitung gradien citra dalam arah horizontal dan vertikal. Operator Sobel mengkonvolusi citra dengan matriks *filter* yang menyoroti perubahan intensitas yang tajam dalam kedua arah ini.
2. Metode Deteksi Tepi Prewitt: Serupa dengan Sobel, metode Prewitt juga menggunakan operator yang menghitung gradien citra dalam arah horizontal dan vertikal, tetapi menggunakan matriks *filter* yang berbeda.
3. Metode Deteksi Tepi Canny: Metode Canny adalah salah satu teknik deteksi tepi paling populer. Metode ini melibatkan beberapa tahap, termasuk menghaluskan citra dengan *filter* Gauss, menghitung gradien citra, menentukan arah tepi, dan menghubungkan tepi dengan menggunakan hysteresis. Metode ini merupakan deteksi tepi paling teliti.
4. Metode Deteksi Tepi *Laplacian*: Metode ini menggunakan operator *Laplacian* untuk menyoroti perubahan intensitas di citra. Ini sering digunakan dalam deteksi tepi tepi tinggi.
5. Metode Deteksi Tepi Roberts: Metode Roberts menggunakan dua operator kernel 2 x 2 untuk menghitung gradien citra dalam arah diagonal. Ini adalah pendekatan yang sederhana tetapi efektif untuk deteksi tepi.
6. Metode Deteksi Tepi LoG *(Laplacian of Gaussian)*: Metode ini menggabungkan *filter* Gaussian dan operator *Laplacian* untuk mendeteksi tepi pada berbagai skala.
7. Metode Deteksi Kirsch: Metode ini menggunakan 8 buah kernel 3 x 3. Kedelapan buah kernel tersebut mewakili setiap arah mata angin. Metode ini memberikan tepian paling tebal jika dibandingkan dengan yang lain.

Untuk perbandingan semua metode dalam deteksi tepi dapat dilihat pada Gambar 8.3 yang merupakan hasil penelitian dari Farzana & Sathik (2016).



Gambar 8.3. Perbandingan Masing-Masing Metode Deteksi Tepi

Untuk praktik 10.1 sampai dengan 10.6 silakan menggunakan citra yang dapat diunduh pada tautan berikut: https://bit.ly/Citra_DeteksiTepi.

Praktik 10.1. Deteksi Tepi
Simpanlah kode program ini dengan nama **EdgeDetection.py**.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

img = cv.imread('dataset\\platmobil.jpg',0)

edges = cv.Canny(img,100,200)
plt.subplot(121),plt.imshow(img,cmap = 'gray')
plt.title('Citra Asli'), plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(edges,cmap = 'gray')
plt.title('Hasil Deteksi Tepi'), plt.xticks([]), plt.yticks([])
plt.show()
```

Praktik 10.2. Deteksi Tepi Sobel
Simpanlah kode program ini dengan nama **SobelEdgeDetection.py**.

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

from skimage import filters
from skimage.data import camera
from skimage.util import compare_images

image = camera()
edge_sobel = filters.sobel(image) # penyaringan dengan sobel

fig, axes = plt.subplots(ncols=2, sharex=True, sharey=True,
                         figsize=(6, 3)) # atur tampilan gambar

axes[1].imshow(edge_sobel, cmap=plt.cm.gray)
axes[1].set_title('Metode Deteksi Tepi Sobel')

for ax in axes:
    ax.axis('off')

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

Praktik 10.3. Deteksi Tepi Prewitt

Simpanlah kode program ini dengan nama **PrewittEdgeDetection.py**.

```
import cv2
import numpy as np

img = cv2.imread('dataset\\platmobil.jpg')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_gaussian = cv2.GaussianBlur(gray,(3,3),0)

#canny
```

```
img_canny = cv2.Canny(img,100,200)

#sobel
img_sobelx = cv2.Sobel(img_gaussian,cv2.CV_8U,1,0,ksize=5)
img_sobely = cv2.Sobel(img_gaussian,cv2.CV_8U,0,1,ksize=5)
img_sobel = img_sobelx + img_sobely

#prewitt
kernelx = np.array([[1,1,1],[0,0,0],[-1,-1,-1]])
kernely = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
img_prewittx = cv2.filter2D(img_gaussian, -1, kernelx)
img_prewitty = cv2.filter2D(img_gaussian, -1, kernely)

cv2.imshow("Citra Asli", img)
cv2.imshow("Canny", img_canny)
cv2.imshow("Sobel X", img_sobelx)
cv2.imshow("Sobel Y", img_sobely)
cv2.imshow("Sobel", img_sobel)
cv2.imshow("Prewitt X", img_prewittx)
cv2.imshow("Prewitt Y", img_prewitty)
cv2.imshow("Prewitt", img_prewittx + img_prewitty)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Praktik 10.4. Deteksi Tepi Canny

Simpanlah kode program ini dengan nama **CannyEdgeDetection.py**.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage as ndi
from skimage.util import random_noise
from skimage import feature
```

```
# Menghasilkan gambar persegi yang memiliki noise
image = np.zeros((128, 128), dtype=float)
image[32:-32, 32:-32] = 1

image = ndi.rotate(image, 15, mode='constant')
image = ndi.gaussian_filter(image, 4)
image = random_noise(image, mode='speckle', mean=0.1)

# Hitung filter Canny untuk dua nilai sigma
edges1 = feature.canny(image)
edges2 = feature.canny(image, sigma=3)

# Menampilkan visualisasi hasil Canny
fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(8, 3))

ax[0].imshow(image, cmap='gray')
ax[0].set_title('Citra Asli Noise', fontsize=10)

ax[1].imshow(edges1, cmap='gray')
ax[1].set_title(r'Hasil Filter Deteksi Canny, $\sigma=1$', fontsize=10)

ax[2].imshow(edges2, cmap='gray')
ax[2].set_title(r'Hasil Filter Deteksi Canny, $\sigma=3$', fontsize=10)

for a in ax:
    a.axis('off')

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

Praktik 10.5. Deteksi Tepi Laplacian

Simpanlah kode program ini dengan nama **LaplacianEdgeDetection.py**.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img0 = cv2.imread('dataset\\platmobil.jpg',)

# mengubah ke dalam skala keabuan
gray = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY)

# menghilangkan noise
img = cv2.GaussianBlur(gray,(3,3),0)

# melakukan pemrosesan dengan kernel yang tepat
laplacian = cv2.Laplacian(img,cv2.CV_64F)
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)  # x
sobely = cv2.Sobel(img,cv2.CV_64F,0,1,ksize=5)  # y

plt.subplot(2,2,1),plt.imshow(img,cmap = 'gray')
plt.title('Citra Asli'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(laplacian,cmap = 'gray')
plt.title('Laplacian'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(sobelx,cmap = 'gray')
plt.title('Sobel X'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(sobely,cmap = 'gray')
plt.title('Sobel Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

Praktik 10.6. Deteksi Tepi Roberts

Simpanlah kode program ini dengan nama **RobertsEdgeDetection.py**.

```
import cv2
```

```
import numpy as np
from scipy import ndimage

roberts_cross_v = np.array( [[1, 0 ],
                                                    [0,-1 ]] )

roberts_cross_h = np.array( [[ 0, 1 ],
                                                    [ -1, 0 ]] )

img = cv2.imread('dataset\\platmobil.jpg',0).astype('float64')
img/=255.0
vertical = ndimage.convolve( img, roberts_cross_v )
horizontal = ndimage.convolve( img, roberts_cross_h )

edged_img = np.sqrt( np.square(horizontal) + np.square(vertical))
edged_img*=255
cv2.imwrite("Hasil Deteksi Tepi Roberts.jpg",edged_img)
```

10.3 Latihan

1. Bagaimana cara menentukan tepian objek?
2. Apa manfaat dari citra ketika dilakukan pendeteksian tepi?
3. Sebutkan dan jelaskan secara ringkas perbedaan masing-masing metode pada deteksi tepi!
4. Carilah contoh kasus perhitungan dalam memanfaatkan metode canny, prewitt, dan sobel!

BAB 11

CITRA BERWARNA

Capaian Pembelajaran:

1. Mampu memahami dan menjelaskan konsep dasar warna.
2. Mampu memahami dan menjelaskan macam-macam model warna.
3. Mampu memahami dan menjelaskan masing-masing kode warna.
4. Mampu menerapkan pemilihan dan kesesuaian model warna melalui kode program.

11.1 Konsep Dasar Warna

Konsep dasar warna dan citra berwarna dalam pengolahan citra digital berhubungan dengan cara representasi dan pemrosesan informasi warna dalam citra digital. Pemahaman tentang konsep ini penting dalam pengolahan citra, fotografi digital, grafika komputer, dan berbagai aplikasi visual lainnya. Persepsi visual citra berwarna *(color images)* umumnya lebih kaya jika dibandingkan dengan citra *grayscale* dikarenakan citra berwarna lebih disenangi daripada citra *grayscale*. Citra berwarna menampilkan warna objek seperti warna aslinya.

Warna yang dilihat oleh manusia mempunyai beberapa persepsi, sebenarnya manusia melihat warna karena adanya cahaya yang dipantulkan oleh objek. Sebagai contoh, suatu objek berwarna hijau karena objek tersebut memantulkan sinar biru dengan panjang gelombang 450 sampai 490 nanometer (nm). Warna sinar yang direspon oleh mata manusia adalah sinar tampak *(visible spectrum)* dengan panjang gelombang berkisar dari 400 nm (ungu) sampai 700 nm (merah). Berikut adalah beberapa konsep dasar yang perlu dipahami:

1. Model Warna: cara kita mewakili dan menggambarkan warna. Beberapa model warna yang umum digunakan dalam pengolahan citra digital adalah: RGB, CMYK, HSV, dan CIELab.

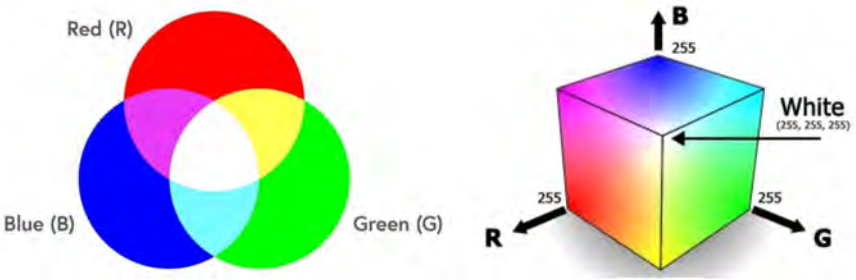
2. Representasi Citra Berwarna: dapat direpresentasikan dalam format raster, yang merupakan kumpulan piksel dengan informasi warna untuk setiap titik di gambar. Pada citra RGB, setiap piksel diwakili oleh tiga nilai yang menyatakan intensitas warna merah, hijau, dan biru.
3. Kanal Warna: dalam citra RGB, setiap saluran warna (merah, hijau, dan biru) disebut kanal warna. Setiap kanal mewakili komponen warna tertentu dalam citra. Misalnya, kanal merah akan menampilkan distribusi intensitas warna merah di seluruh gambar.
4. Histogram Warna: grafik yang menunjukkan distribusi intensitas piksel dalam setiap kanal warna. Histogram ini memberikan gambaran tentang seberapa banyak warna tertentu dalam citra dan membantu dalam analisis dan penyesuaian warna.
5. Pemrosesan Warna: melibatkan berbagai operasi untuk meningkatkan, mengubah, atau mengganti warna dalam gambar. Beberapa teknik pemrosesan warna meliputi penyesuaian kecerahan dan kontras, penyeimbangan warna, peningkatan saturasi, dan lainnya.
6. Segmentasi Berdasarkan Warna: teknik untuk memisahkan objek atau area dalam citra berdasarkan warna. Misalnya, untuk mendeteksi bola sepak dalam lapangan, kita dapat menggunakan metode segmentasi berwarna untuk memisahkan warna bola dari warna latar belakang.
7. *Filtering* Warna: digunakan untuk memisahkan atau mempertahankan piksel dengan intensitas warna tertentu dalam citra. Ini dapat digunakan untuk efek artistik atau tujuan analisis lainnya.

11.2 Macam-Macam Model Warna

11.3.1. Warna RGB

RGB *(Red, Green, Blue)* merupakan warna yang diwakili oleh campuran tiga saluran warna utama, yaitu merah, hijau, dan biru. Kombinasi intensitas ketiga saluran ini menciptakan berbagai warna atau disebut juga sebagai model aditif (saling menambahkan), dimana jika

merah, hijau, dan biru digabungkan menghasilkan warna putih (Lihat Gambar 11.1).

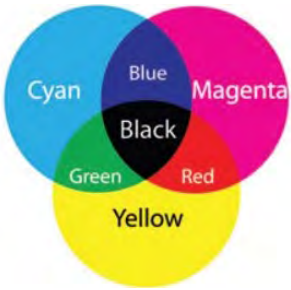


Gambar 11.1. Warna RGB

Setiap saluran (R, G, B) direpresentasikan oleh bilangan bulat dalam rentang 0 hingga 255. Contoh pengkodean warna: (255, 0, 0) untuk merah murni, (0, 255, 0) untuk hijau murni, dan (0, 0, 255) untuk biru murni.

11.3.2. Warna CMYK

CMYK *(Cyan, Magenta, Yellow, Key/Black)* biasa digunakan dalam percetakan. Warna ini diwakili sebagai kombinasi dari empat saluran warna utama (Lihat Gambar 11.2).

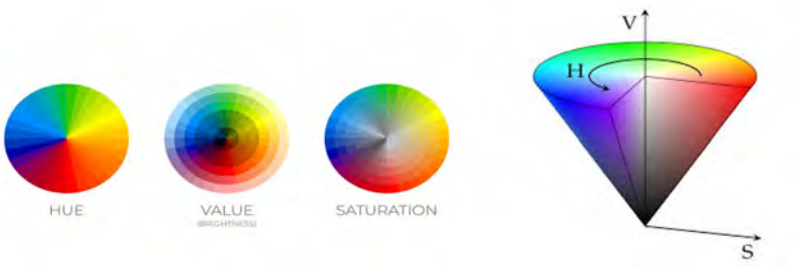


Gambar 11.2. Warna CMYK

Setiap saluran (C, M, Y, K) direpresentasikan oleh bilangan bulat dalam rentang 0 hingga 100 atau bilangan pecahan antara 0 hingga 1 (persentase). Contoh pengkodean warna: (100, 0, 0, 0) untuk cyan murni, (0, 100, 0, 0) untuk magenta murni, (0, 0, 100, 0) untuk kuning murni, dan (0, 0, 0, 100) untuk hitam murni.

11.3.3. Warna HSV

HSV *(Hue, Saturation, Value)* dimana model warna ini memisahkan warna menjadi informasi *hue* (tonalitas/warna yang sesuai dengan panjang gelombang), *saturation* (kejenuhan), dan *value* (nilai kecerahan) (Lihat Gambar 11.3). Menurut beberapa hasil penelitian menyatakan model warna HSV merupakan model yang lebih baik untuk digunakan dalam berbagai keperluan pengolahan citra dan visi komputer.

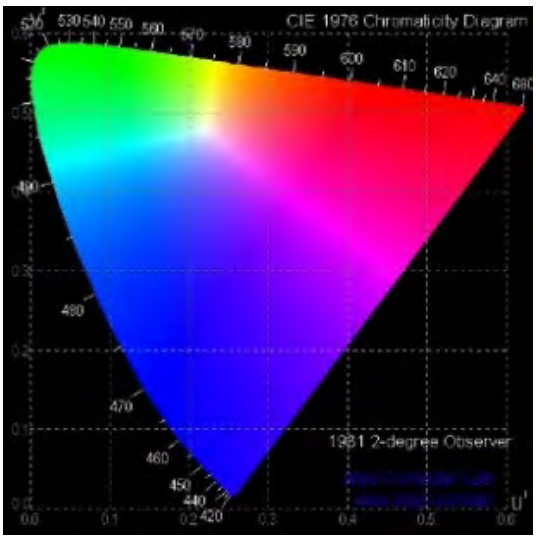


Gambar 11.3. Warna HSV

Hue direpresentasikan oleh sudut dalam rentang 0 hingga 360 derajat. *Saturation* dan *Value* biasanya direpresentasikan dalam rentang 0 hingga 100 atau bilangan pecahan antara 0 hingga 1 (persentase). Contoh pengkodean warna: (0°, 100%, 100%) untuk merah murni, (120°, 100%, 100%) untuk hijau murni, dan (240°, 100%, 100%) untuk biru murni.

11.3.4. Warna CIELab

Pada tahun 1970-an, CIE mengadopsi model warna CIELab sebagai standar internasional. Model warna ini mempunyai kelebihan yakni dekat dengan sistem penglihatan manusia (Lihat Gambar 11.4). Model ini digunakan untuk membedakan antara satu warna dengan warna lainnya. Jarak antara dua titik warna pada model ini diukur dengan metode Euclidean berkesesuaian dengan persepsi sistem penglihatan manusia terhadap warna tersebut.



Gambar 11.4. Warna CIELab

Praktik 11.1. Model Warna RGB

Simpanlah kode program ini dengan nama **RGBColorPicker.py**.

```
import cv2
import numpy as np

def nothing(x):
    pass

# Interaksi Trackbar
cv2.namedWindow("frame")
cv2.createTrackbar("H", "frame", 0, 179, nothing)
cv2.createTrackbar("S", "frame", 255, 255, nothing)
cv2.createTrackbar("V", "frame", 255, 255, nothing)

img_hsv = np.zeros((250, 500, 3), np.uint8)

while True:
    h = cv2.getTrackbarPos("H", "frame")
```

```
    s = cv2.getTrackbarPos("S", "frame")
    v = cv2.getTrackbarPos("V", "frame")

    img_hsv[:] = (h, s, v)
    img_bgr = cv2.cvtColor(img_hsv, cv2.COLOR_HSV2BGR)

    cv2.imshow("frame", img_bgr)
    key = cv2.waitKey(1)
    if key == 27:
        break

cv2.destroyAllWindows()
```

Praktik 11.2. Model Warna RGB to CMYK (Formula)
Simpanlah kode program ini dengan nama **CMYKtoRGBColor.py**.

```
# Rumus untuk mengubah RGB ke CMY
def rgb_to_cmy(r, g, b):

# Nilai RGB dibagi 255
# untuk membawa mereka antara 0 hingga 1.
        c = 1 - r / 255
        m = 1 - g / 255
        y = 1 - b / 255
        return (c, m, y)

# Contoh nilai RGB.
r = 0
g = 169
b = 86

# cetak hasil
print(rgb_to_cmy(r, g, b))
```

Praktik 11.3. Model Warna HSV

Simpanlah kode program ini dengan nama **HSVColorPicker.py**.

```
import cv2
import numpy as np

def nothing(x):
    pass

# Interaksi Trackbar
cv2.namedWindow("frame")
cv2.createTrackbar("H", "frame", 0, 179, nothing)
cv2.createTrackbar("S", "frame", 255, 255, nothing)
cv2.createTrackbar("V", "frame", 255, 255, nothing)

img_hsv = np.zeros((250, 500, 3), np.uint8)

while True:
    h = cv2.getTrackbarPos("H", "frame")
    s = cv2.getTrackbarPos("S", "frame")
    v = cv2.getTrackbarPos("V", "frame")

    img_hsv[:] = (h, s, v)
    img_bgr = cv2.cvtColor(img_hsv, cv2.COLOR_HSV2BGR)

    cv2.imshow("frame", img_bgr)
    key = cv2.waitKey(1)
    if key == 27:
        break

cv2.destroyAllWindows()
```

11.4 Latihan

1. Sebutkan dan jelaskan macam-macam cara dalam melakukan kombinasi warna yang berkesesuaian!
2. Apa dimaksud dengan psikologi warna dan buatlah penjabarannya bahwa setiap warna memengaruhi psikologi dan emosi!
3. Apa tujuan dari adanya kode warna dalam pengolahan citra digital? Jelaskan!

BAB 12

EKSTRAKSI FITUR DAN KONTUR

Capaian Pembelajaran:

1. Mampu memahami dan menjelaskan ekstraksi fitur.
2. Mampu memahami dan menjelaskan kontur.
3. Mampu menerapkan ekstraksi fitur dan kontur melalui kode program.

12.1 Esktraksi Fitur

Ekstraksi fitur *(Feature Extraction)* merupakan proses mengidentifikasi dan mengekstrak informasi penting atau karakteristik khusus dari citra. Fitur-fitur ini dapat mencakup garis, tekstur, pola, bentuk, atau atribut lain yang relevan dengan tujuan analisis atau pengolahan selanjutnya. Tujuan utama dari ekstraksi fitur adalah untuk mengurangi dimensi data dan menyajikan citra dalam representasi yang lebih sederhana dan berarti.

Contoh ekstraksi fitur meliputi:

- Deteksi tepi: Mengekstrak garis dan tepi objek dalam citra.
- Ekstraksi tekstur: Mengidentifikasi pola tekstur seperti kasar, halus, atau berbutir. Terdiri dari beberapa metode yaitu Statistis (*Gray Level Co-occurrence Matrix* (GLCM) dan Tamura); Struktural (*Shape Grammar*); dan Spektral (Fourier, Gabor, Laws).
- Ekstraksi bentuk: Mendeteksi bentuk geometris seperti lingkaran, persegi, atau segitiga.
- Ekstraksi warna: Mendapatkan informasi tentang distribusi warna dalam citra.
- Ekstraksi ciri ukuran: Untuk membedakan ukuran objek satu dengan objek lainnya dapat menggunakan parameter luas dan keliling. Luas merupakan banyaknya piksel yang menyusun suatu objek. Sedangkan keliling merupakan banyaknya piksel yang mengelilingi suatu objek.

Fitur-fitur ini kemudian dapat digunakan untuk berbagai tujuan, seperti pengenalan pola, klasifikasi objek, deteksi objek, atau aplikasi lain dalam bidang pengolahan citra, visi komputer, dan kecerdasan buatan.

12.2 Kontur

Kontur *(Contour)* adalah garis atau tepi yang mengelilingi batas atau perubahan intensitas dalam citra. Kontur mengidentifikasi perbedaan intensitas piksel antara area yang berbeda dalam gambar. Ketika dua area memiliki perbedaan intensitas yang cukup signifikan, kontur akan terbentuk di antara keduanya.

Kontur dapat digunakan untuk berbagai tujuan, termasuk segmentasi objek dalam citra, deteksi objek, pengenalan pola, dan analisis bentuk. Dalam analisis kontur, kita dapat menghitung fitur geometris dari kontur, seperti panjang, luas, atau bentuknya, untuk mengidentifikasi objek tertentu atau karakteristik dari citra.

Berbagai metode pengolahan citra seperti operasi morfologi, pendeteksian tepi (seperti operator Sobel atau Canny), atau transformasi Hough dapat digunakan untuk mengidentifikasi dan mengekstraksi kontur dari citra.

Untuk praktik 12.1 sampai dengan 12.5 silakan menggunakan citra yang dapat diunduh pada tautan berikut: https://bit.ly/Citra_Kontur.

Praktik 12.1. Kontur 1

Simpanlah kode program ini dengan nama **ContourDetection1.py**.

```
import cv2 as cv
import numpy as np

image = cv.imread('dataset\\elang.jpg')

#matriks kosong
blank = np.zeros(image.shape, dtype='uint8')
```

```
#mengkonversi ke grayscale
gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

#deteksi tepi
canny = cv.Canny(gray_image, 215, 275)

#mengidentifikasi contour
contours, hierarchies = cv.findContours(canny,cv.RETR_LIST,
        cv.CHAIN_APPROX_NONE)

#menggambar kontur pada gambar kosong
cv.drawContours(blank, contours, -1, (0,  255,0), 1)

#tampilkan garis kontur pada gambar
cv.imshow("Hasil Citra Deteksi Contour",blank)

cv.waitKey(0)
```

Praktik 12.2. Kontur 2

Simpanlah kode program ini dengan nama **ContourDetection2.py**.

```
import cv2 as cv
import numpy as np

image = cv.imread('dataset\\ImageObject.jpg')

gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
blurred = cv.GaussianBlur(gray, (3, 3), 0)
edged = cv.Canny(blurred, 10, 100)

# mendefinisikan (3, 3) struktur elemen
kernel = cv.getStructuringElement(cv.MORPH_RECT, (3, 3))
```

```
# menerapkan operasi dilation ke gambar tepi/edge detection
dilate = cv.dilate(edged, kernel, iterations=1)

# temukan kontur pada gambar yang didilasi/dilation
contours, _ = cv.findContours(dilate, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
image_copy = image.copy()
# cetak kontur pada salinan gambar aslinya
cv.drawContours(image_copy, contours, -1, (0, 255, 0), 2)
print(len(contours), "Objek yang ditemukan dalam citra masukan ini.")

cv.imshow("Citra yang Dilasi", dilate)
cv.imshow("Hasil dari Contour Detection", image_copy)
cv.waitKey(0)
```

Praktik 12.3. Transformasi Garis Hough pada Citra

Simpanlah kode program ini dengan nama **LineTransformDetection1.py**.

```
import sys
import math
import cv2 as cv
import numpy as np
def main(argv):

    default_file = 'dataset\\TableImage.jpg'
    filename = argv[0] if len(argv) > 0 else default_file

    src = cv.imread(cv.samples.findFile(filename),
cv.IMREAD_GRAYSCALE)
    # Periksa apakah citra dimuat dengan baik
    if src is None:
        print ('Kesalahan dalam membuka file citra!')
        print ('Usage: hough_lines.py [image_name -- default '
```

```
        + default_file + '] \n')
    return -1

dst = cv.Canny(src, 50, 200, None, 3)

# Salin tepi ke citra yang akan menampilkan hasilnya dalam BGR
cdst = cv.cvtColor(dst, cv.COLOR_GRAY2BGR)
cdstP = np.copy(cdst)

lines = cv.HoughLines(dst, 1, np.pi / 180, 150, None, 0, 0)

if lines is not None:
    for i in range(0, len(lines)):
        rho = lines[i][0][0]
        theta = lines[i][0][1]
        a = math.cos(theta)
        b = math.sin(theta)
        x0 = a * rho
        y0 = b * rho
        pt1 = (int(x0 + 1000*(-b)), int(y0 + 1000*(a)))
        pt2 = (int(x0 - 1000*(-b)), int(y0 - 1000*(a)))
        cv.line(cdst, pt1, pt2, (0,0,255), 3, cv.LINE_AA)


linesP = cv.HoughLinesP(dst, 1, np.pi / 180, 50, None, 50, 10)

if linesP is not None:
    for i in range(0, len(linesP)):
        l = linesP[i][0]
        cv.line(cdstP, (l[0], l[1]), (l[2], l[3]), (0,0,255), 3,
            cv.LINE_AA)
```

```
    cv.imshow("Citra Asli", src)
    cv.imshow("Standard Hough Line Transform with Line Detection",
cdst)
    cv.imshow("Probabilistic Line Transform with Line Detection", cdstP)

    cv.waitKey()
    return 0

if __name__ == "__main__":
    main(sys.argv[1:])
```

Praktik 12.4. Menemukan Pola Garis pada Citra

Simpanlah kode program ini dengan nama **LineTransformDetection2.py**.

```
import cv2 as cv
import numpy as np

img = cv.imread(cv.samples.findFile('dataset\\TableImage.jpg'))
gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
edges = cv.Canny(gray,50,150,apertureSize = 3)
lines = cv.HoughLinesP(edges,1,np.pi/180,100,minLineLength=100,
    maxLineGap=10)
for line in lines:
    x1,y1,x2,y2 = line[0]
    cv.line(img,(x1,y1),(x2,y2),(0,255,0),2)

cv.imwrite('deteksigariscitra.jpg',img)
```

Praktik 12.5. Menemukan Pola Lingkaran pada Citra

Simpanlah kode program ini dengan nama **HoughCircleTransform.py**.

```
import numpy as np
import cv2 as cv
```

```
img = cv.imread('dataset\\circleimage.png',0)
img = cv.medianBlur(img,5)
cimg = cv.cvtColor(img,cv.COLOR_GRAY2BGR)
circles = cv.HoughCircles(img,cv.HOUGH_GRADIENT,1,20,
                          param1=50,param2=30,minRadius=0,maxRadius=0)
circles = np.uint16(np.around(circles))
for i in circles[0,:]:
    # gambar lingkaran luar
    cv.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
    # gambar pusat lingkaran
    cv.circle(cimg,(i[0],i[1]),2,(0,0,255),3)

cv.imshow('Hasil Deteksi Pola Lingkaran pada Citra',cimg)
cv.waitKey(0)
cv.destroyAllWindows()
```

12.3 Latihan

1. Jelaskan secara ringkas konsep dari ekstraksi fitur!
2. Sebutkan dan jelaskan masing-masing contih dari ekstraksi fitur!
3. Apa manfaat dari adanya kontur dalam pengolahan citra digital?

BAB 13

SEGMENTASI

Capaian Pembelajaran:

1. Mampu memahami dan menjelaskan konsep segmentasi pada citra.
2. Mampu memahami dan menjelaskan konsep pengambangan pada citra.
3. Mampu menerapkan segmentasi pada citra dan macam-macam pengambangan pada citra melalui kode program.

13.1 Segmentasi Citra

Segmentasi Citra *(Image Segmentation)* merupakan salah satu tahap penting dalam pengolahan citra digital yang bertujuan untuk membagi citra menjadi beberapa wilayah atau segmen *(region)* yang homogen berdasarkan atribut tertentu, seperti warna, tekstur, atau intensitas. Tujuan utama dari segmentasi adalah untuk memudahkan analisis dan pengolahan lebih lanjut pada bagian-bagian yang lebih kecil dan relevan dalam citra (Lihat Gambar 13.1).

Gambar 13.1. Contoh Segmentasi Citra

Metode segmentasi citra dapat bervariasi tergantung pada sifat dan karakteristik citra yang ingin diolah. Beberapa teknik segmentasi citra yang umum digunakan meliputi:

- *Thresholding*: Memisahkan piksel menjadi dua kelompok berdasarkan nilai ambang *(threshold)* tertentu. Piksel dengan intensitas di atas ambang akan dianggap sebagai bagian dari satu wilayah, sedangkan piksel dengan intensitas di bawah ambang akan dianggap sebagai bagian dari wilayah lain.
- *Clustering*: Mengelompokkan piksel berdasarkan kesamaan atribut, seperti warna atau tekstur, ke dalam kelompok-kelompok yang homogen.
- *Watershed Segmentation*: Menggunakan paradigma pemodelan air mengalir untuk memisahkan daerah-daerah dengan "air" yang berbeda dalam citra.
- *Region Growing*: Memperluas wilayah dari piksel awal yang telah dipilih berdasarkan kriteria tertentu hingga mencapai batas yang diinginkan.
- Pendeteksian Tepi: Menggunakan operator tepi, seperti operator Sobel atau Canny, untuk menyoroti garis tepi yang memisahkan objek dalam citra.
- Pendeteksian Kontur: Mendeteksi dan mengekstraksi kontur dari objek dalam citra.
- Metode Pemisahan Warna: Memisahkan objek atau wilayah berdasarkan komponen warna tertentu dalam citra (misalnya, pemisahan merah, hijau, dan biru).

Segmentasi citra digunakan dalam berbagai aplikasi, termasuk pengenalan objek, analisis medis, deteksi gerakan, perhitungan ukuran objek, dan banyak lagi. Ketepatan segmentasi sangat penting karena dapat mempengaruhi kualitas dan akurasi dari hasil analisis dan pemrosesan citra selanjutnya.

13.2 Pengambangan Citra

Image Thresholding atau Pengambangan Citra adalah salah satu teknik penting dalam pengolahan citra digital yang digunakan untuk mengkonversi citra *grayscale* atau citra berwarna menjadi citra biner.

Thresholding merupakan salah satu metode segmentasi citra yang memisahkan antara objek dengan background dalam suatu citra berdasarkan pada perbedaan tingkat kecerahannya atau gelap-terangnya. *Region* citra yang cenderung gelap akan dibuat semakin gelap (hitam sempurna dengan nilai intensitas sebesar 0), sedangkan *region* citra yang cenderung terang akan dibuat semakin terang (putih sempurna dengan nilai intensitas sebesar 1).

Oleh karena itu, keluaran dari proses segmentasi dengan *metode thresholding* adalah berupa citra biner dengan nilai intensitas piksel sebesar 0 atau 1. Setelah citra sudah tersegmentasi atau sudah berhasil dipisahkan objeknya dengan background, maka citra biner yang diperoleh dapat dijadikan sebagai *masking* (penapis) utuk melakukan proses *cropping* sehingga diperoleh tampilan citra asli tanpa *background* atau dengan *background* yang dapat diubah-ubah.

Tujuan dari *thresholding* adalah untuk memisahkan piksel menjadi dua kelompok berdasarkan nilai intensitasnya, yaitu nilai di atas atau di bawah nilai ambang *(threshold)*. Proses *thresholding* sangat sederhana dan efektif untuk mengubah citra ke dalam format biner, yang memiliki beberapa manfaat penting dalam pengolahan citra digital diantaranya:

- Segmentasi Objek: *Thresholding* memungkinkan segmentasi objek dengan warna atau intensitas yang berbeda dalam citra. Misalnya, dalam citra medis, *thresholding* dapat digunakan untuk memisahkan area tumor dari jaringan normal, atau dalam aplikasi *vision*, *thresholding* dapat membantu mendeteksi objek tertentu dalam citra.
- Deteksi Tepi: *Thresholding* dapat digunakan untuk mendeteksi tepi objek dalam citra. Dengan memilih nilai ambang yang tepat, kita dapat menyoroti tepi objek dengan kontras yang tinggi, yang sangat berguna dalam analisis struktural citra.
- Pemrosesan Citra Lanjutan: Citra biner hasil *thresholding* sering digunakan sebagai langkah awal dalam berbagai teknik pemrosesan citra lanjutan. Misalnya, citra biner dapat digunakan dalam operasi

morfologi (erosi dan dilasi) untuk memperbaiki atau mengubah bentuk objek.

- Peningkatan Kualitas Citra: Dengan memilih nilai ambang yang tepat, *thresholding* dapat membantu meningkatkan kualitas citra dengan meningkatkan kontras atau memperjelas detail tertentu dalam citra.
- Ekstraksi Fitur: Citra biner hasil *thresholding* dapat digunakan untuk mengidentifikasi fitur-fitur tertentu dalam citra seperti luas daerah tertentu, jumlah objek, atau atribut lainnya yang relevan dengan analisis.
- Komputasi Efisien: Citra biner yang dihasilkan dari *thresholding* hanya memerlukan satu bit per piksel (0 atau 1), sehingga memungkinkan komputasi yang lebih efisien dibandingkan dengan citra berwarna atau *grayscale* yang memerlukan lebih banyak bit per piksel.

Untuk praktik 13.1 sampai dengan 13.6 silakan menggunakan citra yang dapat diunduh pada tautan berikut: https://bit.ly/Segmentation_Thresholding.

Praktik 13.1. Segmentasi Citra

Simpanlah kode program ini dengan nama **ImageSegmentation.py**.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('dataset\\ImageSegmentation.jpg')
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
plt.figure(figsize=(8,8))
plt.imshow(img,cmap="gray")
plt.axis('off')
plt.title("Citra Asli")
plt.show()
```

```
#mengkonversi ke citra grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(8,8))
plt.imshow(gray,cmap="gray")
plt.axis('off')
plt.title("Citra GrayScale")
plt.show()

#mengkonversi ke Binary Inverted Image
ret, thresh = cv2.threshold(gray, 0, 255,cv2.THRESH_BINARY_INV
+cv2.THRESH_OTSU)
plt.figure(figsize=(8,8))
plt.imshow(thresh,cmap="gray")
plt.axis('off')
plt.title("Citra Threshold")
plt.show()

#proses segmentasi citra
kernel = np.ones((3, 3), np.uint8)
closing = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE,kernel,
iterations = 15)
bg = cv2.dilate(closing, kernel, iterations = 1)
dist_transform = cv2.distanceTransform(closing, cv2.DIST_L2, 0)
ret, fg = cv2.threshold(dist_transform, 0.02*dist_transform.max(), 255, 0)
cv2.imshow('image', fg)
plt.figure(figsize=(8,8))
plt.imshow(fg,cmap="gray")
plt.axis('off')
plt.title("Segmentasi Citra")
plt.show()
```

```
#hasil akhir segmentasi (rekapan proses)
plt.figure(figsize=(10,10))

plt.subplot(2,2,1)
plt.axis('off')
plt.title("Citra Asli")
plt.imshow(img,cmap="gray")

plt.subplot(2,2,2)
plt.imshow(gray,cmap="gray")
plt.axis('off')
plt.title("Citra GrayScale")

plt.subplot(2,2,3)
plt.imshow(thresh,cmap="gray")
plt.axis('off')
plt.title("Citra Threshold")

plt.subplot(2,2,4)
plt.imshow(fg,cmap="gray")
plt.axis('off')
plt.title("Segmentasi Citra")

plt.show()
```

Praktik 13.2. *Simple Thresholding*

Simpanlah kode program ini dengan nama **SimpleThresholding.py**.

```
import cv2
import numpy as np

img = cv2.imread('dataset\\DaunGanja.jpg')
```

```
grayscaled = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
retval, threshold = cv2.threshold(grayscaled, 110, 255,
cv2.THRESH_BINARY)
cv2.imshow('citra asli',img)
cv2.imshow('hasil threshold',threshold)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Praktik 13.3. *Basic Image Thresholding*

Simpanlah kode program ini dengan nama **BasicImageThresholding.py**.

```
import cv2
import numpy as np

image1 = cv2.imread('dataset\\DaunGanja.jpg')

# cv2.cvtColor digunakan untuk melakukan konversi color space
# untuk mengonversi gambar dalam skala abu-abu
img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

# menerapkan ambang batas yang berbeda
# teknik pada gambar input
# semua nilai piksel di atas 120 akan disetel ke 255
ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(img, 120, 255,
cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img, 120, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img, 120, 255,
cv2.THRESH_TOZERO_INV)

cv2.imshow('Binary Threshold', thresh1)
cv2.imshow('Binary Threshold Inverted', thresh2)
```

```
cv2.imshow('Truncated Threshold', thresh3)
cv2.imshow('Set to 0', thresh4)
cv2.imshow('Set to 0 Inverted', thresh5)

if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
```

Praktik 13.4. *Adaptive Thresholding*

Simpanlah kode program ini dengan nama **AdaptiveThresholding.py**.

```
import cv2
import numpy as np

image1 = cv2.imread('dataset\\InputanAdaptiveThresholding.jpg')

img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

# menerapkan ambang batas yang berbeda dengan adaptive mean dan
gaussian
thresh1 = cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_MEAN_C,
        cv2.THRESH_BINARY, 199, 5)
thresh2 = cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY, 199, 5)

cv2.imshow('Adaptive Mean', thresh1)
cv2.imshow('Adaptive Gaussian', thresh2)

if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
```

Praktik 13.5. *Otsu Thresholding*

Simpanlah kode program ini dengan nama **OtsuThresholding.py**.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('dataset\\naskahkuno.jpg',0)

ret1,th1 = cv.threshold(img,127,255,cv.THRESH_BINARY)

# proses Otsu's thresholding
ret2,th2 =
cv.threshold(img,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# Thresholding Otsu setelah penyaringan dengan rumus Gaussian
blur = cv.GaussianBlur(img,(5,5),0)
ret3,th3 =
cv.threshold(blur,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)

# plot semua gambar dan histogramnya
images = [img, 0, th1,
        img, 0, th2,
        blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global Thresholding
(v=127)',
        'Original Noisy Image','Histogram',"Otsu's Thresholding",
        'Gaussian filtered Image','Histogram',"Otsu's Thresholding"]
for i in range(2):
    plt.subplot(3,3,i*3+1),plt.imshow(images[i*3],'gray')
    plt.title(titles[i*3]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),256)
    plt.title(titles[i*3+1]), plt.xticks([]), plt.yticks([])
    plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],'gray')
```

```
    plt.title(titles[i*3+2]), plt.xticks([]), plt.yticks([])
plt.show()
```

Praktik 13.6. *Multi Otsu Thresholding*

Simpanlah kode program ini dengan nama **MultiOtsuThresholding.py**.

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

from skimage import data #Library Scikit-Image (Skimage)
from skimage.filters import threshold_multiotsu

# Mengatur ukuran font untuk semua plot
matplotlib.rcParams['font.size'] = 9

# Memasukkan citra secara otomatis
image = data.camera()

# Menerapkan ambang batas nilai untuk multi-Otsu dengan nilai default
# sehingga menghasilkan tiga kelas
thresholds = threshold_multiotsu(image)

# Menggunakan nilai ambang batas, kami menghasilkan tiga wilayah
regions = np.digitize(image, bins=thresholds)

fig, ax = plt.subplots(nrows=1, ncols=3, figsize=(10, 3.5))

# Merencanakan gambar asli
ax[0].imshow(image, cmap='gray')
ax[0].set_title('Original')
ax[0].axis('off')
```

```
# Memplot histogram dan dua ambang yang diperoleh dari multi-Otsu
ax[1].hist(image.ravel(), bins=255)
ax[1].set_title('Histogram')
for thresh in thresholds:
    ax[1].axvline(thresh, color='r')

# Merencanakan hasil Multi Otsu
ax[2].imshow(regions, cmap='jet')
ax[2].set_title('Multi-Otsu result')
ax[2].axis('off')

plt.subplots_adjust()

plt.show()
```

13.3 Latihan

1. Jelaskan secara ringkas tujuan dari segmentasi citra!
2. Sebutkan dan jelaskan macam-macam teknik dalam segmentasi citra!
3. Jelaskan tujuan dari pengambangan citra!
4. Jelaskan apa perbedaan antara *Adaptive Thresholding* dengan *Otsu Thresholding*?

BAB 14

PENGENALAN DAN KONSEP VISI KOMPUTER

Capaian Pembelajaran:
Mampu memahami dan menjelaskan konsep visi computer.

14.1 Visi Komputer

Visi Komputer *(Computer Vision)* merupakan salah satu jenis kecerdasan artifisial/buatan yang mampu membuat komputer untuk memperoleh informasi yang bermanfaat dari gambar digital, video, ataupun berbagai sistem sensor visual lainnya. Manusia mampu memahami dunia sekitar dengan sistem penglihatan yang diproses oleh otak. Komputer dapat diprogram untuk dapat meniru sistem penglihatan manusia dengan menggunakan algoritme visi komputer. Dengan ini, komputer atau mesin akan mampu mengidentifikasi, mendeteksi, dan melacak berbagai objek dengan akurat serta dapat menarik kesimpulan dari beberapa gambar yang berurutan (Kahlil dkk., 2023).

Selain itu menurut Santa (2023) menyebutkan bahwa visi komputer adalah cabang ilmu komputer yang memungkinkan komputer untuk memproses, menganalisis, dan memahami gambar dan atau video. Dalam visi komputer, komputer dilatih untuk memproses data gambar dan atau video yang diakuisisi dari dunia nyata, selanjutnya menghasilkan informasi yang bermanfaat untuk keperluan tertentu. Misalnya pada sistem pengawasan lalu lintas, dimana visi komputer digunakan untuk mengenali nomor plat kendaraan yang melanggar aturan berlalu lintas.

Visi komputer bertujuan untuk memberikan kemampuan kepada komputer untuk “melihat” dan memahami dunia fisik melalui citra dan video. Visi komputer berupaya untuk menggabungkan teknik pengolahan citra digital, kecerdasan buatan, dan pengenalan pola untuk

memungkinkan komputer untuk mendeteksi, mengidentifikasi, dan menginterpretasi objek, wajah, gerakan, dan lainnya dari citra atau video.

Kaitan visi komputer dengan pengolahan citra digital lanjutan sangat erat, karena visi komputer sebagian besar bergantung pada pengolahan citra untuk ekstraksi fitur, segmentasi, deteksi objek, dan berbagai tugas lainnya. Beberapa contoh kaitan antara visi komputer dan pengolahan citra digital lanjutan adalah sebagai berikut:

- Deteksi Objek *(Object Detection)*: sisi komputer menggunakan teknik pengolahan citra digital lanjutan, seperti segmentasi dan deteksi tepi, untuk mengidentifikasi objek tertentu dalam citra atau video. Metode seperti R-CNN, YOLO, dan SSD adalah contoh deteksi objek menggunakan visi komputer.
- Pengenalan Wajah *(Face Recognition)*: pengenalan wajah merupakan bagian penting dari visi komputer. Teknik seperti deteksi wajah, ekstraksi fitur wajah, dan pengenalan pola digunakan dalam pengolahan citra digital untuk mengenali wajah individu dalam gambar atau video.
- Pengenalan Pola *(Pattern Recognition)*: visi komputer seringkali melibatkan pengenalan pola untuk mengklasifikasikan objek atau mengidentifikasi pola tertentu dalam citra. Pengenalan pola memanfaatkan teknik pengolahan citra lanjutan untuk mengidentifikasi dan mengekstraksi fitur-fitur relevan.
- Klasifikasi Citra *(Image Classification)*: proses memprediksi kelas tertentu, atau label, untuk sesuatu yang didefinisikan oleh sekumpulan titik data. Klasifikasi gambar merupakan bagian dari masalah klasifikasi, di mana seluruh gambar diberi label. Sebagai contoh mungkin sebuah gambar akan diklasifikasikan sebagai pemotretan siang atau malam hari. Selain itu, dengan cara yang sama, gambar mobil dan sepeda motor juga akan secara otomatis ditempatkan ke dalam kelompoknya sendiri.
- Pencocokan Fitur *(Feature Matching)*: bagian dari banyak aplikasi visi komputer seperti pendaftaran gambar, kalibrasi kamera, dan

pengenalan objek, adalah tugas untuk membuat korespondensi antara dua gambar dari pemandangan atau objek yang sama. Pendekatan umum untuk *feature matching* ini terdiri dari mendeteksi satu set titik minat masing-masing terkait dengan deskriptor gambar dari data gambar. Setelah fitur dan deskriptornya diekstraksi dari dua atau lebih gambar, langkah selanjutnya adalah membuat beberapa kecocokan fitur awal antara gambar-gambar ini.

- Pemrosesan Citra Medis *(Medical Image Processing)*: pengolahan citra digital lanjutan dalam visi komputer digunakan dalam bidang medis untuk mendeteksi dan mendiagnosis penyakit, mengidentifikasi struktur anatomi, dan melacak perubahan pada citra medis.
- Pemrosesan Citra Video *(Video Processing)*: pengolahan citra digital lanjutan digunakan dalam analisis video untuk deteksi gerakan, pelacakan objek, dan pemahaman konteks temporal dari peristiwa yang terjadi dalam video dan atau rekaman video secara *real-time*.

Visi komputer dan pengolahan citra digital lanjutan telah membawa dampak besar dalam berbagai aplikasi kehidupan sehari-hari, seperti kendaraan otonom, pengawasan keamanan, pengenalan wajah pada perangkat pintar, perawatan kesehatan, dan banyak lagi. Kemajuan di dalam kedua bidang ini terus mendorong inovasi dan penerapan teknologi yang semakin canggih dan relevan dalam masyarakat.

14.2 Latihan

1. Jelaskan secara ringkas keterkaitan antara visi komputer dengan pengolahan citra digital!
2. Carilah dua buah artikel jurnal penelitian yang membahas tentang studi kasus dalam visi komputer, kemudian jelaskan bagaimana visi komputer dapat menyelesaikan kasus tersebut. Selain itu *screenshoot* pengaplikasinnya dan prosesnya!

BAB 15

KASUS-KASUS PROGRAM DASAR VISI KOMPUTER

Capaian Pembelajaran:

1. Mampu memahami dan menjelaskan pengenalan objek.
2. Mampu memahami dan menjelaskan deteksi objek.
3. Mampu memahami dan menjelaskan pelacakan objek.
4. Mampu menerapkan kasus-kasus program dasar visi komputer melalui kode program.

15.1 Pengenalan Objek *(Object Recognition)*

Pengenalan Objek adalah tugas untuk mengidentifikasi dan mengklasifikasikan objek tertentu dalam citra atau video ke dalam kategori atau kelas tertentu. Ini berarti mengenali objek tanpa perlu mengetahui jumlah atau lokasi tepat dari objek tersebut. Misalnya, dalam pengenalan wajah, tujuannya adalah untuk mengidentifikasi wajah dalam citra tanpa perlu mengetahui posisi wajah secara eksak.

15.2 Deteksi Objek (*Object Detection*)

Deteksi Objek adalah tugas untuk mendeteksi keberadaan objek dalam citra atau video dan menentukan lokasi dan batas-batas dari objek tersebut. Berbeda dengan *object recognition* yang hanya mengenali objek tanpa informasi lokasi, *object detection* memberikan informasi lebih detail tentang di mana objek berada dalam gambar. Misalnya, dalam deteksi kendaraan pada citra jalan, tujuannya adalah untuk mendeteksi lokasi dan batas-batas dari setiap kendaraan yang ada.

15.3 Pelacakan Objek (*Object Tracking*)

Pelacakan Objek adalah tugas untuk melacak pergerakan objek dari *frame* ke *frame* dalam urutan video. Ini berarti mengidentifikasi dan

melacak posisi objek dari waktu ke waktu saat objek bergerak dalam video. *Object tracking* digunakan untuk memantau objek dalam pergerakan, seperti pelacakan kendaraan di jalanan atau pelacakan wajah dari *frame* ke *frame* dalam rekaman video dan terkadang rekaman secara *real-time* melalui CCTV.

Untuk praktik 15.1 sampai dengan 15.8 silakan menggunakan citra atau datasetnya yang dapat diunduh pada tautan berikut: https://bit.ly/KasusProgramDasarVisiKomputer.

Praktik 15.1. Pencocokan *Template* pada Citra
Simpanlah kode program ini dengan nama **TemplateMatching.py**.

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

#Masukkan gambar pasfoto anda
img = cv.imread('dataset\\akhrian.png',0)
img2 = img.copy()

#Masukkan gambar pasfoto anda dengan nama file berbeda
#dan hanya bagian wajah saja
template = cv.imread('dataset\\akhriancrop.jpg',0)
w, h = template.shape[::-1]

methods = ['cv.TM_CCOEFF', 'cv.TM_CCOEFF_NORMED',
'cv.TM_CCORR',
        'cv.TM_CCORR_NORMED', 'cv.TM_SQDIFF',
'cv.TM_SQDIFF_NORMED']
for meth in methods:
    img = img2.copy()
    method = eval(meth)
```

```
    # Penerapan template Matching
    res = cv.matchTemplate(img,template,method)
    min_val, max_val, min_loc, max_loc = cv.minMaxLoc(res)
    # Jika menggunakan TM_SQDIFF atau TM_SQDIFF_NORMED,
maka diambil nilai minimum
    if method in [cv.TM_SQDIFF, cv.TM_SQDIFF_NORMED]:
        top_left = min_loc
    else:
        top_left = max_loc
    bottom_right = (top_left[0] + w, top_left[1] + h)
    cv.rectangle(img,top_left, bottom_right, 255, 2)
    plt.subplot(121),plt.imshow(res,cmap = 'gray')
    plt.title('Matching Result'), plt.xticks([]), plt.yticks([])
    plt.subplot(122),plt.imshow(img,cmap = 'gray')
    plt.title('Detected Point'), plt.xticks([]), plt.yticks([])
    plt.suptitle(meth)
    plt.show()
```

Praktik 15.2. Deteksi Bentuk pada Citra

Simpanlah kode program ini dengan nama **ShapeDetection.py**.

```
import cv2 as cv
image = cv.imread('dataset\\Shape.png')

#mengubah gambar menjadi mode skala keabuan
gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

#temukan image thresholding
_, thrash = cv.threshold(gray_image, 240,255, cv.THRESH_BINARY)
contours ,_ = cv.findContours(thrash, cv.RETR_TREE,
cv.CHAIN_APPROX_NONE)
```

```
for contour in contours:
        shape = cv.approxPolyDP(contour, 0.01*cv.arcLength(contour,
True), True)
        x_cor = shape.ravel()[0]
        y_cor = shape.ravel()[1]

        #deteksi pola gambar segitiga
        if len(shape) ==3:
                cv.drawContours(image, [shape], 0,(0,0,0),4)
                cv.putText(image,"ini adalah segitiga", (x_cor,y_cor),
                        cv.FONT_HERSHEY_COMPLEX, 0.5, (0,0,0))

        #deteksi pola gambar pentagon
        if len(shape) ==5:
                cv.drawContours(image, [shape], 0,(0,0,0),4)
                cv.putText(image,"ini adalah pentagon", (x_cor,y_cor),
                        cv.FONT_HERSHEY_COMPLEX, 0.5, (0,0,0))

        #deteksi pola gambar lingkaran
        if len(shape) >12:
                cv.drawContours(image, [shape], 0,(0,0,0),4)
                cv.putText(image,"ini adalah lingkaran", (x_cor,y_cor),
                        cv.FONT_HERSHEY_COMPLEX, 0.5, (0,0,0))

        #deteksi pola gambar bintang
        if len(shape) ==10:
                cv.drawContours(image, [shape], 0,(0,0,0),4)
                cv.putText(image,"ini adalah bintang", (x_cor,y_cor),
                        cv.FONT_HERSHEY_COMPLEX, 0.5, (0,0,0))

        #deteksi pola gambar trapesium
        if len(shape) ==4:
```

```
                x,y,w,h = cv.boundingRect(shape)
                aspect_ratio = float(w)/h
                if aspect_ratio >=1.2 and aspect_ratio <=2.0:
                        cv.drawContours(image, [shape], 0,(0,0,0),4)
                        cv.putText(image,"ini adalah trapesium",
(x_cor,y_cor),
                                cv.FONT_HERSHEY_COMPLEX, 0.5,
(0,0,0))

cv.imshow("Hasil Deteksi Bentuk", image)
cv.waitKey(0)
cv.destroyAllWindows()
```

Soal Tantangan:

1. Pada praktikum 15.2, silakan modifikasi kode program untuk mendeteksi objek persegi panjang dan persegi pada citra, selanjutnya carilah citra yang berisi objek persegi panjang dan persegi sebagai dataset citra!

Praktik 15.3. Pelacakan Bentuk secara *Real-Time*

Simpanlah kode program ini dengan nama **RealTimeShapeTracking.py**.

```
import cv2
import numpy as np

def nothing(x):

    pass

#akuisisi citra secara real-time
cap = cv2.VideoCapture(0)

cv2.namedWindow("Trackbars")
```

```
cv2.createTrackbar("L-H", "Trackbars", 0, 180, nothing)
cv2.createTrackbar("L-S", "Trackbars", 66, 255, nothing)
cv2.createTrackbar("L-V", "Trackbars", 134, 255, nothing)
cv2.createTrackbar("U-H", "Trackbars", 180, 180, nothing)
cv2.createTrackbar("U-S", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("U-V", "Trackbars", 243, 255, nothing)

font = cv2.FONT_HERSHEY_COMPLEX

while True:
    _, frame = cap.read()
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    l_h = cv2.getTrackbarPos("L-H", "Trackbars")
    l_s = cv2.getTrackbarPos("L-S", "Trackbars")
    l_v = cv2.getTrackbarPos("L-V", "Trackbars")
    u_h = cv2.getTrackbarPos("U-H", "Trackbars")
    u_s = cv2.getTrackbarPos("U-S", "Trackbars")
    u_v = cv2.getTrackbarPos("U-V", "Trackbars")

    lower_red = np.array([l_h, l_s, l_v])
    upper_red = np.array([u_h, u_s, u_v])

    mask = cv2.inRange(hsv, lower_red, upper_red)
    kernel = np.ones((5, 5), np.uint8)
    mask = cv2.erode(mask, kernel)

    # Deteksi Contours
    if int(cv2.__version__[0]) > 3:

        contours, _ = cv2.findContours(mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

```
    else:

        _, contours, _ = cv2.findContours(mask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)

    for cnt in contours:
        area = cv2.contourArea(cnt)
        approx = cv2.approxPolyDP(cnt, 0.02*cv2.arcLength(cnt, True),
True)
        x = approx.ravel()[0]
        y = approx.ravel()[1]

        if area > 400:
            cv2.drawContours(frame, [approx], 0, (0, 0, 0), 5)

    #Proses pelacakan bentuk secara real-time
            if len(approx) == 3:
                cv2.putText(frame, "Segitiga", (x, y), font, 1, (0, 0, 0))
            elif len(approx) == 4:
                cv2.putText(frame, "Persegi Panjang", (x, y), font, 1, (0, 0, 0))
            elif 10 < len(approx) < 20:
                cv2.putText(frame, "Lingkaran", (x, y), font, 1, (0, 0, 0))

    cv2.imshow("Hasil Deteksi", frame)
    cv2.imshow("Lapisan Mask Image", mask)

    key = cv2.waitKey(1)
    if key == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Praktik 15.4. Pelacakan Warna secara *Real-Time*

Simpanlah kode program ini dengan nama **ShapeDetection.py**.

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

while 1:
    ret,frame =cap.read()
    into_hsv =cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

    #Proses pembacaan warna pola RGB
    L_limit=np.array([98,50,50])
    U_limit=np.array([139,255,255])

    b_mask=cv2.inRange(into_hsv,L_limit,U_limit)
    blue=cv2.bitwise_and(frame,frame,mask=b_mask)

    cv2.imshow('Akuisisi Foto Asli pada Camera',frame)
    cv2.imshow('Hasil Pelacakan Warna Biru',blue)

    if cv2.waitKey(1)==27:
        break

cap.release()
cv2.destroyAllWindows()
```

Soal Tantangan:

1. Pada praktikum 15.4, silakan modifikasi kode program untuk melacak warna hijau dan kuning!

Praktik 15.5. Pelacakan Wajah secara *Real-Time* 1
Simpanlah kode program ini dengan nama **FaceTracking1.py**.

```
import cv2
import numpy as np

#masukkan dataset haarcascade_frontalface_default
faceDetect =
cv2.CascadeClassifier('dataset\\haarcascade_frontalface_default.xml');

#Melakukan akuisisi citra
camera = cv2.VideoCapture(0);
while(True):
 ret,img =camera.read();
 gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
 faces = faceDetect.detectMultiScale(gray,1.3,5);
 for (x,y,w,h) in faces:

#Cetak blok persegi berwarna hijau muda dengan kode RGB 0, 255, 0
#Cetak blok di area wajah saja
 cv2.rectangle(img,(x,y), (x+w,y+h),(0,255,0),2)

#Tampilkan hasil pelacakan
 cv2.imshow("Hasil Pelacakan Wajah",img);
 if(cv2.waitKey(1) ==ord('q')):
  break;

  camera.release()
cv2.destroyAllWindows()
```

Praktik 15.6. Pelacakan Wajah secara *Real-Time* 2

Simpanlah kode program ini dengan nama **FaceTracking2.py**.

```
import cv2
import os

#masukkan dataset haarcascade_frontalface_default
cascPath=os.path.dirname(cv2.__file__)+"/dataset/haarcascade_frontalfa
ce_default.xml"
faceCascade = cv2.CascadeClassifier(cascPath)

video_capture = cv2.VideoCapture(0)

while True:
    # menangkap frame-by-frame
    ret, frames = video_capture.read()

    gray = cv2.cvtColor(frames, cv2.COLOR_BGR2GRAY)

    faces = faceCascade.detectMultiScale(
        gray,
        scaleFactor=1.1,
        minNeighbors=5,
        minSize=(30, 30),
        flags=cv2.CASCADE_SCALE_IMAGE
    )

#Cetak blok persegi berwarna biru dengan kode RGB 0, 0, 255
#Cetak blok di area wajah saja
    for (x, y, w, h) in faces:
        cv2.rectangle(frames, (x, y), (x+w, y+h), (0, 0, 255), 2)

    # Tampilkan frame pelacakan yang dihasilkan
```

```
    cv2.imshow('Hasil Pelacakan Wajah', frames)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

        video_capture.release()
cv2.destroyAllWindows()
```

Praktik 15.7. Pelacakan Wajah dan Mata secara *Real-Time*
Simpanlah kode program ini dengan nama **FaceandEyeTracking.py**.

```
import numpy as np
import cv2

#masukkan dataset wajah
face_cascade=cv2.CascadeClassifier('dataset\\haarcascade_frontalface_d
efault.xml')
#masukkan dataset mata
eye_cascade=cv2.CascadeClassifier('dataset\\haarcascade_eye.xml')

cap=cv2.VideoCapture(0)

while 1:
    ret, img=cap.read()
    gray=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces=face_cascade.detectMultiScale(gray, 1.3, 5)

#proses pelacakan area wajah
#blok area wajah dengan bentuk persegi berwarna biru
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        roi_gray=gray[y:y+h, x:x+w]
        roi_color=img[y:y+h, x:x+w]
```

```
#proses pelacakan area mata
#blok area mata dengan bentuk persegi berwarna hijau
        eyes=eye_cascade.detectMultiScale(roi_gray)
        for (ex,ey,ew,eh) in eyes:
            cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)

    cv2.imshow('Hasil Deteksi Wajah dan Mata',img)
    k=cv2.waitKey(30) & 0xff
    if k==27:
        break

cap.release()
cv2.destroyAllWindows()
```

Praktik 15.8. Pelacakan Titik Api secara *Real-Time*

Simpanlah kode program ini dengan nama **FireTracking.py**.

```
import cv2

def detect_fire_live():

    #Masukkan dataset api dari file .xml
    fire_cascade = cv2.CascadeClassifier('dataset\\fire_tracking.xml');

    #akuisisi citra dengan open camera
    live_Camera = cv2.VideoCapture(0)

    while True:
        #Baca setiap frame dari pengambilan video
        ret, frame = live_Camera.read()

        if not ret:
```

```
            break

        #Konversi frame menjadi skala keabuan
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        #proses pelacakan api
        fires = fire_cascade.detectMultiScale(gray, scaleFactor=1.1,
minNeighbors=5, minSize=(30, 30))

        #blok area api yang terlacak dengan bentuk persegi berwarna merah
        for (x, y, w, h) in fires:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)

        #tampilkan hasil pelacakan
        cv2.imshow("Hasil Deteksi Titik Api", frame)

        #akhiri perulangan program dengan menekan tombol q
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    #Lepaskan tangkapan video dan tutup semua jendela
    live_Camera.release()
    cv2.destroyAllWindows()

#Panggil fungsi untuk melacak api dari video secara real-time
detect_fire_live()
```

BAB 16

TANTANGAN DAN PERKEMBANGAN TERKINI APLIKASI PENGOLAHAN CITRA DIGITAL DAN VISI KOMPUTER

Capaian Pembelajaran:
Mampu memahami dan menjelaskan tantangan dan perkembangan terkini pengaplikasian pengolahan citra digital dan visi computer.

16.1 Tantangan Pengaplikasian

Pengaplikasian pengolahan citra digital dan visi komputer masa kini menghadapi beberapa tantangan yang perlu diatasi untuk mencapai hasil yang lebih baik dan akurat. Beberapa tantangan utama termasuk:

1. Pengolahan Citra Besar dan *Real-Time*: Dalam beberapa aplikasi, seperti kendaraan otonom *(self-driving)* atau sistem pengawasan, pengolahan citra dan visi komputer harus dilakukan secara *real-time* dengan data citra yang sangat besar. Hal ini memerlukan kemampuan komputasi yang tinggi dan efisien untuk memproses data dalam waktu nyata.
2. Variasi Pencahayaan dan Kondisi Lingkungan: Citra dapat diambil dalam berbagai kondisi pencahayaan dan lingkungan yang berbeda, seperti kondisi cahaya rendah, pantulan cahaya, atau gangguan lainnya. Pengolahan citra dan visi komputer harus tahan terhadap variasi ini untuk memberikan hasil yang konsisten dan andal.
3. Deteksi dan Pengenalan Objek yang Kompleks: Deteksi dan pengenalan objek yang kompleks, seperti manusia, kendaraan, atau benda lain dengan banyak variasi bentuk dan posisi, merupakan tantangan besar dalam visi komputer. Klasifikasi objek yang tepat memerlukan teknik yang canggih dan akurat.

4. *Data Training* dan Anotasi: Pembuatan model visi komputer yang handal memerlukan dataset yang besar dan representatif untuk pelatihan. Memperoleh *data training* yang lengkap dan memiliki anotasi yang tepat bisa menjadi tugas yang rumit dan memakan waktu.
5. Keamanan dan Privasi: Dalam beberapa aplikasi, seperti pengenalan wajah atau sistem pengawasan, masalah keamanan dan privasi menjadi sangat penting. Tantangan ini mencakup perlindungan data citra dan penggunaan teknologi visi komputer yang etis dan adil.
6. Interpretasi Semantik: Memahami konteks dan interpretasi semantik dalam citra atau video adalah tantangan kompleks. Misalnya, dalam pengenalan objek, pemahaman tentang hubungan antar objek dan konteks lingkungan sangat penting untuk hasil yang lebih baik.
7. Pembelajaran dan Adaptasi Konstan: Teknologi visi komputer terus berkembang dengan cepat, dan tugas adaptasi konstan diperlukan untuk menghadapi perkembangan baru dan perubahan dalam lingkungan aplikasi.

Untuk mengatasi tantangan-tantangan ini, riset dan pengembangan terus berlanjut untuk meningkatkan algoritma, infrastruktur perangkat keras, dan dataset yang digunakan dalam pengolahan citra digital dan visi komputer. Dengan terus berlanjutnya inovasi dan kemajuan teknologi, diharapkan visi komputer dapat memberikan dampak yang semakin besar dalam berbagai bidang aplikasi di masa depan.

16.2 Perkembangan Terkini Pengaplikasian

16.2.1. Pengaplikasian secara Umum

Pengolahan citra digital dan visi komputer telah diaplikasikan dalam berbagai bidang dan aplikasi masa kini. Berikut adalah beberapa contoh pengaplikasiannya:

1. Kendaraan Otonom: Dalam industri otomotif, teknologi visi komputer digunakan dalam kendaraan otonom untuk membantu sistem navigasi dan pengambilan keputusan. Kamera dan sensor pada kendaraan dapat

mendeteksi rambu lalu lintas, penghalang, pejalan kaki, dan kendaraan lain untuk menghindari tabrakan dan mengemudi secara mandiri.

2. Pengawasan Keamanan dan CCTV: Pengolahan citra dan visi komputer digunakan dalam sistem pengawasan keamanan dan *Closed-Circuit Television* (CCTV) untuk mendeteksi kejadian mencurigakan, pencurian, atau perilaku abnormal. Sistem ini memungkinkan identifikasi dan peringatan dini terhadap potensi ancaman.
3. Pengenalan Wajah dan Identifikasi Biometrik: Aplikasi pengenalan wajah digunakan dalam keamanan perangkat pintar, seperti ponsel cerdas dan perangkat pintar lainnya, serta dalam sistem keamanan untuk mengenali individu berdasarkan ciri wajah uniknya.
4. Pengenalan Teks dalam Citra: Teknologi visi komputer digunakan untuk mengenali dan mengekstraksi teks dari gambar atau video. Aplikasinya termasuk OCR untuk mengubah teks dalam dokumen fisik menjadi teks digital yang dapat diakses.
5. Pengenalan Pola dalam Kesehatan: Dalam bidang kedokteran, visi komputer digunakan untuk mendeteksi dan mendiagnosis penyakit berdasarkan citra medis seperti MRI, CT scan, atau sinar-X. Ini termasuk deteksi tumor, analisis gambar mikroskopis, dan lainnya.
6. *Augmented Reality* (AR) dan *Virtual Reality* (VR): Visi komputer berperan dalam pengalaman AR dan VR dengan mengidentifikasi dan mengintegrasikan objek virtual ke dalam lingkungan fisik pengguna, memberikan interaksi yang lebih realistis dan immersif.
7. Industri Manufaktur dan Inspeksi Kualitas: Dalam industri manufaktur, teknologi visi komputer digunakan untuk inspeksi kualitas dan pengawasan produksi. Ini dapat mengidentifikasi cacat dalam produk atau komponen untuk memastikan standar kualitas yang tinggi.
8. Pengenalan Aksi dalam Video: Visi komputer dapat digunakan untuk mengenali aksi atau gerakan dalam video, seperti mendeteksi gerakan manusia, mengenali aktivitas manusia, atau pelacakan objek dalam video.

9. Pendeteksian Emosi dan Sentimen: Dalam analisis emosi atau analisis sentimen, visi komputer digunakan untuk mengenali ekspresi wajah atau ekspresi tubuh manusia untuk menilai emosi atau sentimen mereka.

16.2.1. Pengaplikasian dalam Area *Smart City*

Pengolahan citra digital juga memiliki penerapan yang luas dalam pengembangan konsep *smart city*. *Smart city* adalah konsep yang menggabungkan teknologi informasi dan komunikasi untuk meningkatkan efisiensi, kualitas hidup, dan keberlanjutan kota. Berikut adalah beberapa contoh penerapan pengolahan citra digital dan visi komputer dalam konteks *smart city*:

1. Pengelolaan Lalu Lintas: sistem pengawasan lalu lintas untuk mendeteksi dan mengontrol lalu lintas di jalan raya. Kamera CCTV yang terhubung dengan sistem dapat mendeteksi kepadatan lalu lintas, kecelakaan, pelanggaran lalu lintas, dan mengatur pengaturan lampu lalu lintas secara adaptif berdasarkan data citra yang diperoleh.
2. Keamanan Publik: sistem digunakan dalam pemantauan keamanan publik untuk mendeteksi aktivitas mencurigakan, pengenalan wajah, dan identifikasi pelaku kejahatan. Kamera CCTV yang terhubung dengan sistem dapat membantu polisi dalam menjaga keamanan kota dengan mengidentifikasi individu yang dicurigai atau mencari tersangka berdasarkan fitur wajah mereka.
3. Pengelolaan Limbah: sistem pengelolaan limbah untuk mengoptimalkan proses pengumpulan sampah. Kamera yang dipasang pada tong sampah dapat mengidentifikasi tingkat isi sampah, mengirimkan informasi ke pusat pengendalian, dan mengatur rute pengumpulan sampah yang efisien berdasarkan data citra.
4. Pemantauan Kualitas Udara: dimana mampu mendeteksi polusi udara. Sensor dan kamera yang terhubung dapat mengambil citra atmosfer dan menganalisis kualitas udara secara *real-time*. Data citra ini dapat membantu otoritas kota dalam mengambil langkah-langkah untuk

mengurangi polusi udara dan meningkatkan kualitas udara bagi warganya.

5. Pemantauan Parkir: untuk membantu pengguna mencari tempat parkir yang tersedia. Kamera yang terhubung dengan sistem dapat mengidentifikasi ruang parkir kosong dan mengirimkan informasi ke aplikasi atau papan pengumuman yang memberitahu pengguna tentang ketersediaan parkir di area tertentu.
6. Pemantauan Kualitas Air: digunakan dalam pemantauan kualitas air di sungai, danau, atau pantai. Citra satelit atau *drone* digunakan untuk mendapatkan gambaran tentang kejernihan air, polusi, dan perubahan lingkungan. Data citra ini dapat membantu otoritas kota dalam mengambil tindakan untuk menjaga keberlanjutan dan kebersihan sumber daya air.
7. Sistem Irigasi Cerdas: menggunakan kombinasi antara teknologi visi komputer dan *Internet of Things* (IoT) untuk memantau tingkat kelembaban tanah, pola cuaca, dan kesehatan tanaman, serta mampu memberikan umpan balik waktu nyata kepada para petani untuk mengoptimalkan irigasi dan mengurangi limbah air.

Penerapan pengolahan citra digital dan visi komputer dalam *smart city* memberikan kemungkinan besar untuk mengoptimalkan berbagai aspek kehidupan perkotaan, termasuk transportasi, keamanan, pengelolaan sumber daya, dan lainnya.

DAFTAR PUSTAKA

A. A. Syahidi, H. Tolle, A. A. Supianto, and K. Arai, “BandoAR: Real-Time Text Based Detection System Using Augmented Reality for Media Translator Banjar Language to Indonesian with Smartphone,” Proc. of the 5th IEEE ICETAS 2018, pp. 1-6, 2018.

A. A. Syahidi, Joniriadi, N. H. Waworuntu, Subandi, and K. Kiyokawa, “Tour Experience with Interactive Map Simulation based on Mobile Augmented Reality for Tourist Attractions in Banjarmasin City,” International Journal of Informatics and Computer Science, vol. 6, no. 1, pp. 22-31, 2022.

A. Farzana and M. M. Sathik, “Survey of Various Edge Detection Techniques over Brain MRI,” International Journal of Trend in Research and Development, vol. 4, no. 1, pp. 41-43, 2016.

D. Putra, “Pengolahan Citra Digital,” Yogyakarta: Penerbit ANDI, 2010.

I. T. Young, J. Gerbrands, and L. J. van Vliet, “Fundamentals of Image Processing,” Delf: TU Delft, Faculty of Applied Physics, Pattern Recognition Group, 1995.

K. Santa, “Buku Ajar Pengenalan Computer Speech and Vision,” Surabaya: Jakad Media Publishing, 2023.

Kahlil dkk., “Computer Vison Berbasis Deep Learning untuk Aplikasi Pertanian: Teori dan Praktik,” Aceh: Syiah Kuala University Press, 2023.

M. M. P. Petrou and C. Petrou, “Image Processing: The Fundamentals – 2nd Edition,” United States: Wiley, 2010.

M. Sonka, V. Hlavac, and R. Boyle, “Image Processing, Analysis, and Machine Vision,” United States: Cengage Learning, 2015.

P. A. Yushkevich, A. Pashchinskiy, I. Oguz, S. Mohan, J. E. Schmitt, J. M. Stein, D. Zuki´c, J. Vicory, M. McCormick, N. Yushkevich, N. Schwartz, Y. Gao, and G. Gerig, “User-Guided Segmentation of

Multi-modality Medical Imaging Datasets with ITK-SNAP," Neuroinformatics, vol. 17, no. 1, pp. 83-102, 2019.

P. Hidayatullah, "Pengolahan Citra Digital: Teori dan Aplikasi Nyata," Bandung: Penerbit Informatika, 2017.

R. A. Asmara, "Pengolahan Citra Digital – Teori, Praktek, dan Latihan-Latihan," Malang: POLINEMA PRESS, 2018.

R. C. Gonzalez and R. E. Woods, "Digital Image Processing – 4th Edition," England: Pearson Education Limited, 2018.

R. C. Gonzalez, R. E. Woods, and S. L. Eddins, "Digital Image Processing Using MATLAB," United States: McGraw-Hill Higher Education, 2011.

R. Munir, "Interpretasi dan Pengolahan Citra," Bandung: Institut Teknologi Bandung, Sekolah Teknik Elektro dan Informatika, 2019.

W. K. Pratt, "Digital Image Processing: PIKS Scientific Inside – 4th Edition," United States: Wiley-Interscience, 2007.

GLOSARIUM

A

Akuisisi Citra	Proses untuk mengambil, mendapatkan, atau proses pencuplikan citra dari sumbernya, seperti kamera, scanner, atau sensor lainnya.

C

Citra	Disebut juga sebagai gambar yang merupakan representasi visual dari objek atau scene yang dapat diamati oleh mata manusia atau perangkat sensor elektronik.

D

Derau	Pengotor pada citra disebut sebagai *noise* yang memiliki beragam penyebab, yang menyebabkan citra menjadi turun kualitasnya atau cacat.
Deteksi Tepi	Teknik untuk menemukan garis tepi dari suatu objek pada citra dengan cara mendeteksi perubahan tingkat kecerahan yang signifikan atau mempunyai diskontinuitas.

H

Histogram	Representasi grafis dari distribusi intensitas piksel dalam sebuah citra dimana menggambarkan jumlah piksel dengan intensitas tertentu pada sumbu Y (frekuensi) terhadap intensitas piksel pada sumbu X.

K

Kernel	Matriks kecil yang dapat digunakan untuk mengubah nilai piksel pada citra.

Kontur	Rangkaian nilai yang menunjukkan batas dari sebuah objek dalam citra.
Konvolusi	Mengubah suatu citra dengan menerapkan suatu kernel atau filter pada setiap piksel di dalamnya.
M	
Morfologi	Serangkaian teknik pengolahan citra yang dilakukan pada piksel-piksel citra berdasarkan bentuk, ukuran, dan struktur objek dalam citra.
P	
Pengambangan Citra	Salah satu teknik penting dalam pengolahan citra digital yang digunakan untuk mengkonversi citra *grayscale* atau citra berwarna menjadi citra biner.
Pengolahan Citra Digital	Manipulasi citra oleh komputer dengan tujuan untuk meningkatkan kualitas citra, mendapatkan informasi yang berguna, atau mengambil keputusan berdasarkan citra tersebut.
Piksel	Unit terkecil yang membentuk gambar digital atau citra, dimana setiap piksel merupakan titik individu dalam citra yang memiliki lokasi atau koordinat tertentu, dan memiliki nilai numerik yang mewakili intensitas warna atau tingkat kecerahan untuk citra berwarna atau citra *grayscale*.
S	
Segmentasi	Membagi citra menjadi beberapa wilayah atau segmen *(region)* yang homogen berdasarkan atribut tertentu, seperti warna, tekstur, atau intensitas.

V

Visi Komputer	Teknologi yang memberikan kemampuan kepada komputer untuk “melihat” dan memahami dunia fisik melalui citra dan video.

TENTANG PENULIS

Aulia Akhrian Syahidi, M.Kom. merupakan dosen tetap PNS di Program Studi Sarjana Terapan Sistem Informasi Kota Cerdas dan D3 Teknik Informatika, Jurusan Teknik Elektro, Politeknik Negeri Banjarmasin. Pendidikan terakhir ditempuh di Program Studi Magister (S2) Ilmu Komputer, Jurusan Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya Malang melalui beasiswa RISET *(Research Intensification for Sustainable Education and Training)* dari Universitas Brawijaya Malang, kemudian pada Semester III saat menjalani S2 juga berhasil berkolaborasi penelitian dengan Prof. Kohei Arai, B.S., M.S., Ph.D. (*Saga University*, Jepang) untuk bidang penelitian *Augmented Reality* dan *Character Recognition*, serta juga menjadi asisten peneliti dan berkolaborasi penelitian dengan Prof. Tsukasa Hirashima, B.E., M.E., Ph.D. (*Learning Engineering Lab.*, *Hiroshima University*, Jepang) untuk bidang penelitian *Educational Data Mining* dan *Interactive Educational Media Environment.*

Bidang keahlian beliau diantaranya adalah *Augmented Reality*, *Virtual Reality*, *Game Development, Mobile Application, Human-Computer Interaction (User Interface, User Experience, and Usability Evaluation), Digital Image Processing, Computer Vision, Internet of Things, Educational Data Mining, Artificial Intelligence*, dan *Information System*. Mata kuliah yang pernah diampu diantaranya Pengantar Sistem Informasi, Struktur Data, Dasar *Internet of Things*, Pengolahan Citra Digital, Kecerdasan Buatan, Metode Numerik, Desain Interaksi dan Antarmuka Pengguna, dan Interaksi Manusia dan Komputer. Pernah memperoleh prestasi internasional seperti *Best Paper Award* pada konferensi internasional 5th IEEE ICETAS di *Asian Institute of Technology*, Thailand Tahun 2018 dan 5th IEEE ICCED 2019 di *Nanyang Technological University*, Singapura serta peraih ide terbaik pada program IYIS 2018 di *Tokyo International Exchange Centre*, Tokyo, Jepang. Sekarang, beliau juga aktif sebagai tim *reviewer* dan *editor* pada banyak jurnal nasional maupun internasional di bidang Sistem Informasi, Teknik Informatika, Ilmu Komputer, Pendidikan Teknologi Informasi, dan Desain Komunikasi Visual serta *IEEE Transactions on Visualization and Computer Graphics*. Selain itu aktif sebagai asisten peneliti profesor di *Cybernetics and Reality Engineering Laboratory* (Prof. Kiyoshi Kiyokawa, B.E., M.E., Ph.D./Kiyokawa Sensei Lab.), *Division of Information Science*, *Nara Institute of Science and Technology* (NAIST), Jepang pada bidang *Augmented and Virtual Reality*.

Email penulis: aakhriansyahidi@poliban.ac.id / aakhriansyahidi@gmail.com

Buku ini adalah panduan komprehensif yang membahas tentang pengolahan citra digital dan dasar visi komputer dengan menggunakan bahasa pemrograman Python. Dengan penjelasan yang jelas dan contoh yang praktis serta soal teori, buku ini cocok untuk pemula yang ingin memahami konsep dasar hingga para praktisi yang ingin menguasai teknik-teknik canggih. Buku ini memang diperuntukan khusus untuk mahasiswa perguruan tinggi vokasi untuk program sarjana terapan dan diploma 3, akan tetapi juga sangat relevan bagi mahasiswa lainnya yang sedang menempuh mata kuliah pengolahan citra digital yang ingin memahami dasar visi komputer.

Isi buku mencakup topik tentang penting diantaranya (1) Pengenalan dan Konsep Pengolahan Citra Digital; (2) Pemrograman Python pada Pengolahan Citra Digital; (3) Akuisisi, Jenis, Format File, dan Hubungan Antarpiksel pada Citra; (4) Operasi Dasar pada Citra Digital; (5) Histogram Citra; (6) Konvolusi dan Filter Spasial; (7) Transformasi Fourier; (8) Perbaikan Kualitas Citra; (9) Operasi Morfologi; (10) Deteksi Tepi; (11) Citra Berwarna; (12) Ekstraksi Fitur dan Kontur; (13) Segmentasi; (14) Pengenalan dan Konsep Visi Komputer; (15) Kasus-Kasus Program Dasar Visi Komputer; (16) Tantangan dan Perkembangan Terkini Aplikasi Pengolahan Citra Digital dan Visi Komputer yang langsung dikaitkan dengan kasus-kasus dalam proyek pengembangan kota cerdas (smart city). Buku ini juga mencakup proyek-proyek praktis yang memungkinkan pembaca untuk langsung mengaplikasikan pengetahuan yang telah dipelajari dalam konteks dunia nyata. Proyek-proyek tersebut mencakup pengenalan wajah, pengenalan pola, deteksi objek, dan masih banyak lagi dimana dapat diimplementasikan dengan mudah menggunakan bahasa pemrograman Python yang juga dilengkapi dengan pustaka (library) pendukungnya.

Dengan berfokus pada bahasa pemrograman Python yang populer dan ramah pemula, buku ini membantu pembaca merasa lebih percaya diri dalam mengembangkan aplikasi dan sistem dasar visi komputer. Dalam buku ini, para pembaca akan menemukan solusi praktis untuk berbagai tugas pemrosesan citra digital dan pemahaman mendasar tentang visi komputer.

Penerbit Poliban Press
Redaksi :
Politeknik Negeri Banjarmasin, Jl. Brigjen H. Hasan Basry, Pangeran, Komp. Kampus ULM, Banjarmasin Utara
Telp : (0511)3305052
Email : press@poliban.ac.id

ISBN 978-623-5259-11-6 (PDF)
9 786235 259116