# BAB II

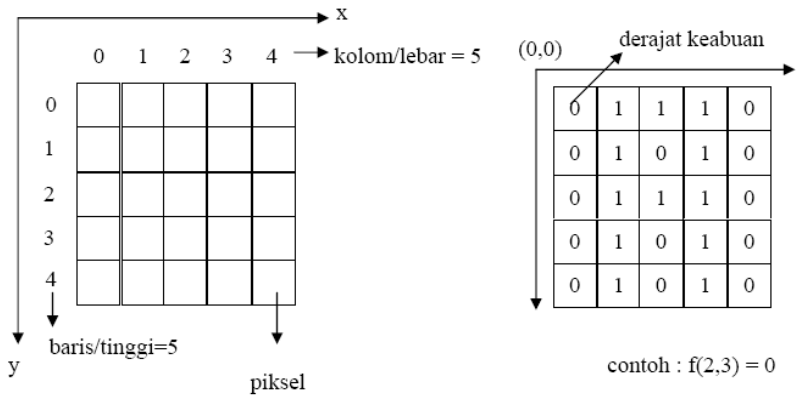
# LANDASAN TEORI

## 2.1 Citra Digital

Citra, menurut kamus *Webster*, adalah suatu representasi, kemiripan, atau imitasi dari suatu *objek* atau benda. Citra dapat dikelompokkan menjadi citra tampak dan citra tak tampak. Bayak contoh citra tampak dalam kehidupan sehari-hari, seperti foto, lukisan, serta apa yang nampak dilayar monitor dan televisi. Sedangkan citra tak nampak misalnya : data gambar dalam *file* (citra digital), dan citra yang direpresentasikan menjadi fungsi matematis. Hanya citra digital yang dapat diolah menggunakan komputer, jenis citra lain harus diubah terlebih dahulu menjadi citra digital misalnya dengan dipindai (*scan*) dengan *scanner*.

Citra digital adalah suatu fungsi dari intensitas cahay suatu objek dua dimensi yang dinotasikan dalam f (x,y), dimana dalam hal ini (x,y) menyatakan titik koordinat pada bidang dua dimensi, dan nilai f pada setiap titik (x,y) menyataknan intensitas , tingkat kecerahan atau derajat keabuan ( *brightness* atau *gray level* ). Citra digital merupakan suatu matriks dimana indeks baris dan kolomnya menyatakan suatu titik pada citra tersebut dan elemen matriksnya ( yang disebut sebagai elemen gambar atau *pixel* ) menyatakan tingkat keabuan pada titik tersebut. Citra digital berukuran N x M (tinggi = N, lebar = M) dinyatakan dengan matriks N x M.

| | | |
|---|---|---|
| N = jumlah baris | $0 \leq y \leq N - 1$ | ….. (2.1) |
| M = jumlah kolom | $0 \leq x \leq M - 1$ | |
| L = maksimal warna intensitas | $0 \leq f(x,y) \leq L - 1$ | |




Gambar 2.1 Ilustrasi Matriks Citra Digital

### 2.1.1 Karakteristik Citra Digital

Setiap citra digital memiliki beberapa karakteristik, antara lain yaitu :

a. Ukuran Citra Digital

Ukuran citra (image size) menyatakan ukuran banyaknya pixel penyusun citra raster yang dinyatakan dalam matrik 2 dimensi, yaitu (X × Y) Pixel, dimana X menyatakan ukuran banyaknya pixel perbaris pada arah horizontal sedangkan Y menyatakan ukuran banyaknya pixel perkolom pada arah vertikal. Sebagai contoh, Citra digital berukuran 800 × 600 pixel, terdiri dari 800 × 600 pixel = 480.000 pixel, dengan susunan 800 pixel setiap baris pada arah horizontal dan 600 pixel setiap kolom pada arah vertikal.

b. Resolusi

Atribut citra digital yang tak kalah pentingnya adalah resolusi (resolution), yang didefinisikan sebagai banyaknya pixel dalam setiap satuan panjang. Umumnya, resolusi dinyatakan dalam satuan dpi (dot per inchi). Sebagai contoh, citra digital yang memiliki resolusi 72 dpi, berarti terdiri dari 72 dot

(titik) pada setiap inchi. Semakin tinggi resolusi suatu citra digital, maka kualitasnya akan semakin baik.

### 2.1.2 Tipe Citra Digital

Berdasarkan format penyimpanan nilai warnanya Citra digital terbagi dalam tiga tipe yaitu :

1. Citra Biner

   Citra yang hanya memiliki dua nilai yaitu 1 dan 0. Dinyatakan dalam suatu fungsi:

   $$f(x,y) \Sigma \{0,1\} \quad \text{….. (2.2)}$$

2. Citra Grey-scale

   Citra yang terdiri dari satu layer warna dengan derajat keabuan tertentu. Untuk kebanyakan citra digital 8 bit, maka sistem Grey-scale diukur berdasarkan skala intensitas kecerahan, yang bernilai 0 – 255, dimana yang hitam pekat adalah 0 dan yang terputih adalah 255. Dinyatakan dalam suatu fungsi :

   $$f(x,y) \Sigma [0\ldots255] \quad \text{….. (2.3)}$$

3. Citra Berwarna

   Citra standar (Bitmap) dengan ukuran 512 × 512 dan mempunyai kedalaman warna 24 bit atau dikenal dengan istilah *true color,* dalam hal ini kedalaman warna yang dimaksud yaitu jumlah bit yang dipakai untuk merepresentasikan tiap titik pada gambar tersebut dinyatakan dalam bit per pixel . komputer mempunyai tiga komponen warna yaitu : merah, hijau, dan biru. Ketiga komponen dasar ini dikenal dengan istilah warna RGB (Red Green

Blue). Untuk citra digital dengan 8 bit per pixel mempunyai 256 warna sedangkan pada citra dengan 24 bit per pixel mempunyai 16 juta warna, tiap pixel dinyatakan dengan :

Bit ke-0 sampai ke-7 untuk warna merah

Bit ke-8 sampai ke-15 untuk warna hijau

Bit ke-16 sampai ke-23 untuk warna biru

Sehingga kombinasi warnanya adalah $256^3 + 256^2 + 256^1 = 16.843.008$.

Warna-warna ini merupakan warna dasar penyusun setiap pixel pada citra digital. Representasi dalam citra digital dinyatakan dalam persamaan :

$$\begin{aligned} f_R(x,y)\ \Sigma\ [0\ldots255] \\ f_G(x,y)\ \Sigma\ [0\ldots255] \\ f_B(x,y)\ \Sigma\ [0\ldots255] \end{aligned} \qquad \text{.....(2.4)}$$

### 2.1.3 Format File Citra Digital

Format file menentukan bagaimana informasi data dipresentasikan dalam suatu file. Informasi tersebut meliputi ada tidaknya kompresi, program aplikasi (feature) yang di support, penggunaan enkripsi dan lain-lain. Tiap format file memiliki kelebihan dan kelemahan pada masing-masing format tersebut. Dalam sistem operasi Windows biasanya format file dapat dibedakan dari namanya yaitu diakhiri titik dan diikuti dengan tiga atau empat huruf terakhir (misal .txt, .doc, .html dan lain-lain). Beberapa format file pada citra digital antara lain:

a. BMP (Bitmap) adalah standar file bitmap atau raster pada sistem operasi berbasis Windows. Gambar dengan format *.BMP (Bitmap) adalah gambar standar yang mempunyai informasi jumlah bit yang dipakai untuk merepresentasikan tiap titik pada citra digital yang dinyatakan dalam bit per

pixel. Format ini mampu menyimpan informasi dengan kualitas tingkat 1 bit samapi 24 bit. Biasanya mempunyai ukuran file yang relatif besar.

b. GIF (Graphics Interchange Format) menggunakan maksimal 8 bit warna ($2^8$ = 256 warna) pada gambar dan melakukan compresi dengan LZW compression yang merupakan kompresi loseless. Kompresi Loseless ini berarti tidak ada data yang dibuang. Sifatnya dapat menjadi lossy compression jika ada informasi warna pada gambar yang hilang. Format GIF mendukung gambar transparansi dan animasi.

c. JPEG (Joint Photographic Experts Group) menggunakan 24 bit warna ($2^{24}$ = 16 juta warna) dan melakukan kompresi dengan cara membuang data pada gambar (bersifat lossy compression). Semakin kecil file yang diinginkan semakin kecil data yang akan dibuang sehingga kualitasnya akan semakin menurun. Format JPEG tidak mendukung transparansi dan animasi.

d. PNG-8 menggunakan 8 bit warna, kurang kompatibel (tidak didukung ) oleh browser, tetapi biasanya mempunyai hasil kompresi yang lebih kecil dari format GIF. Berbeda dengan GIF yang telah di patenkan. Format PNG besifat bebas paten.

e. PNG-24 menggunakan 24 bit warna, hampir sama dengan PNG-8 tetapi mempunyai ukuran yang lebih besar dan warna yang lebih banyak.

Didalam penelitian ini format file yang digunakan adalah format file BMP dan PNG, alasan penggunaan kedua format file tersebut adalah bahwa format file BMP dan PNG adalah format file yang bersifat *lossless*, ini berarti bahwa tidak ada data informasi yang hilang sebagai bagian dari format file tersebut. Sebab dalam proses encode, jika menggunakan suatu format yang merugi (*lossy*)

seperti JPEG, maka kemungkin akan kehilangan sebagian dari informasi yang tersembunyi dan tidak akan bisa mendapatkan kembali pesan itu lagi. Oleh karena itu untuk mempertahankan informasi yang tersembunyi didalam file citra-stego, maka file *citra-stego* harus disimpan didalam sebuah format file citra digital yang bersifat *lossless*.

## 2.2 *Steganography*

*Steganography* sudah dikenal oleh Bangsa Yunani. Penguasa yunani dalam mengirimkan pesan rahasia menggunakan kepala budak atau prajurit sebagai media. Dalam hal ini, rambut budak dibotaki, lalu pesan ditulis pada kulit kepala budak. Ketika rambut budak tumbuh, budak tersebut diutus untuk membawa pesan rahasia.

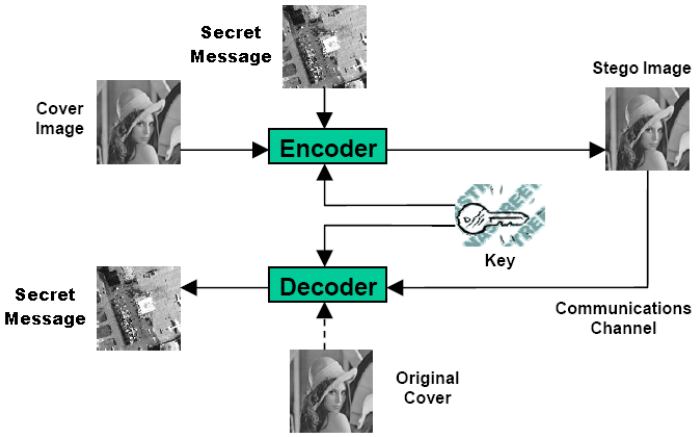
Bangsa Romawi mengenal steganography dengan menggunakan tinta tak nampak (*invisible ink*) untuk menulis pesan. Tinta tersebut dibuat dari campuran sari buah, susu, dan cuka. Jika tinta itu digunakan untuk menulis maka tulisannya tidak nampak. Tulisan diatas kertas dapat dibaca dengan cara memanaskan kertas tersebut.

*Steganography* adalah teknik menyembunyikan data rahasia didalam wadah atau media (*cover*) digital sehingga keberadaan data rahasia tersebut tidak diketahui oleh orang. Hal yang fundamental yang diperlukan dalam metode *steganography* adalah tidak terlihat jelas, artinya pesan yang disisipkan pada media tersebut harus tidak terlihat secara kasat mata. Dua hal lain yang juga dibutuhkan adalah cara untuk memaksimalkan kapasitas pesan yang dapat disisipkan, dan juga menjamin keamanan (*security*) dari pesan yang disisipkan

tersebut. Gambar 2.2 menampilkan sebuah representasi sederhana dari proses *encoding* dan *decoding* secara umum didalam *steganography*.

Metode *steganography* akan menyisipkan pesan pada sebuah media *cover* dalam hal ini adalah citra berwarna 24 bit dan membangun sebuah *citra-stego*. Beberapa metode *steganography* menggunakan *stego-key* dalam menyisipkan pesan yang berguna untuk menjaga keamanan pesan yang disisipkan.

Setelah pesan disisipkan kedalam *citra-cover* oleh program *encoder* maka akan dihasilka *citra-stego*. *Citra-stego* yang dihasilkan ini seharusnya terlihat serupa dengan *citra-cover*, jika tidak pihak penyerang atau orang yang tidak berhak dapat melihat informasi yang disisipkan.




Gambar 2.2 Proses *encoding* dan *decoding* didalam *steganography*.

Setelah memproduksi *citra-stego*, kemudian akan dikirimkan via beberapa saluran komunikasi, seperti email. Kepada penerima diharapkan untuk memecahkan kode, penerima harus memecahkan kode (*decode*) *citra-stego* tersebut untuk memperoleh informasi rahasia tersebut kembali. Proses *decode* secara sederhananya adalah kebalikan dari proses *encode*. Dalam memecahkan kode proses (*decode*), dibutuhkan kunci (*stego-key*), yang dapat berupa *password* yang digunakan didalam proses *encode* sebelumnya. setelah proses *decode*

diselesaikan, informasi rahasia yang disisipkan dalam *citra-stego* kemudian bisa diekstrak dan dilihat.

Penyembunyian data atau pesan rahasia kedalam citra digital akan mengubah kualitas citra tersebut. kriteria yang harus diperhatikan dalam penyembunyian pesan data adalah :

1. *Fidelity*, yaitu mutu *citra-cover* tidak jauh berubah. Setelah penyisipan data, citra hasil atau *citra-stego* masih terlihat baik.
2. *Robustness*, yaitu data yang disembunyikan harus tahan (*robust*) terhadap berbagai operasi manipulasi yang dilakukan pada *citra-stego*, seperti pengubahan kontras, penajaman, pemampatan, rotasi, pemotongan, dan sebagainya.
3. *Recovery*, yaitu data yang disembunyikan harus dapat diungkap kembali (*reveal*) atau diekstrak kembali, karena tujuan *steganography* adalah *data-hiding*, maka sewaktu-waktu data rahasia didalam *citra-stego* harus dapat diambil kembali untuk digunakan lebih lanjut.

## 2.3 Teori *Least Significant Bits*

Secara matematis, metode *Least Significant Bits* pada citra digital merupakan perubahan nilai bit-bit yang paling rendah dari setiap nilai pixel pada citra digital dengan nilai bit-bit pesan yang akan disisipkan. Dengan demikian perubahan tersebut hanya megakibatkan perubahan yang tidak signifikan pada intensitas pada byte tertentu atau erat hubungannya dengan warna pada suatu citra. Dan juga memanfaatkan kelemahan mata manusia yang sekilas tidak dapat membedakan perubahan yang kecil. Contoh sederhana :

Pesan :

Karakter "I" = 108 *desimal* (ASCII), 1 *byte*

= 01101100 *biner*

| | | | | |
|---|---|---|---|---|
| Sebuah pixel dengan nilai | : ( | 10101010 | 10101010 | 10101010 ) |
| | | *Red* | *Green* | *Blue* |
| Jumlah bit | : 3 | | | |
| Sebuah pixel baru | : ( | 10101011 | 10101011 | 10101000 ) |
| | | *Red* | *Green* | *Blue* |

Dalam proses penyisipan pesan kedalam sebuah citra, dapat memanfaatkan metode penyisipan *Least Significant Bits* seperti yang telah dijelaskan sebelumnya. Dalam implementasinya, pesan yang akan disisipkan harus dikonversi terlebih dahulu kedalam bentuk sistem bilangan sehingga dapat melakukan proses penyisipan, tetapi pada dasarnya computer mengenal sebuah karakter memang sudah dalam bentuk sistem bilangan *biner* sehingga kita hanya perlu mengetahui cara memanggil atau fungsi yang digunakan bahasa pemrograman untuk mengembalikan karakter ke bentuk bilangan.

Untuk meningkatkan kapasitas total suatu citra dapat menampung pesan maka dalam implementasinya kapasitas per kanal warnanya diset 4 bit, karena citra yang digunakan citra 24 bit yang terdiri dari 3 kanal warna (R-G-B) maka kapasitas perpixelnya adalah 12 bit.

Dalam proses peyisipan bit-bit pesan ke bit-bit *pixel* citra, menggunakan operator logika *and* dan *or*. Contoh : 1 *and* 1 = 1, 1 *and* 0 = 0, 0 *and* 0 = 0, 1 *or* 1 = 1, 1 *or* 0 = 1, 0 *or* 0 = 0. Dengan mengeset kapasitas penyisipan 4 bit per kanal warna, maka dalam proses penyisipan diperlukan suatu bilangan yang berfungsi mempertahankan 4 bit MSB (*Most Significant Bits*) nilai intensitas kanal warna pada suatu *pixel* citra karena hanya akan merubah 4 bit LSB-nya, bilangan yang dimaksud adalah F0 (*hexadecimal*) = 240 (*decimal*) = 11110000 (*binary*).

Sedangkan untuk mempertahankan bit-bit pesan, digunakan bilangan F = 15 = 00001111. Contoh ilustrasi : suatu citra dengan ukuran 2 × 2 *pixel* (gambar 2.3), akan disisipkan karakter 'K' = 01001011 (ASCII)

| Piksel 1 | | | Pikse 2 | | |
|---|---|---|---|---|---|
| Blue | Green | Red | Blue | Green | Red |
| 00110011 | 01011101 | 01000110 | 00110111 | 01011101 | 01000100 |
| **Piksel 3** | | | **Piksel 4** | | |
| Blue | Green | Red | Blue | Green | Red |
| 00110010 | 01011100 | 01001110 | 00110011 | 01011111 | 01100110 |

Gambar 2.3 Ilustrasi *pixel citra-cover*

Langkah 1 : Sisipkan 4 bit LSB karakter pada kanal warna Blue pada pixel 1

( 0011<u>0011</u> and 11110000 ) or ( 0100<u>1011</u> and 00001111 ) = 0011<u>1011</u>

Blue F0 'K' F hasil 1

Langkah 2 : Geser kekanan 4 bit MSB karakter yang belum diproses lalu sisipkan pada kanal warna Green pada pixel 1

( 01011101 and 11110000 ) or ( 00000100 and 00001111 ) = 01010100

Green F0 'K' setelah digeser F hasil 2

Sehingga menghasilkan perubahan intensitas pixel seperti gambar 2.4 berikut.

| Piksel 1 | | | Pikse 2 | | |
|---|---|---|---|---|---|
| Blue | Green | Red | Blue | Green | Red |
| 00111011 | 01010100 | 01000110 | 00110111 | 01011101 | 01000100 |
| **Piksel 3** | | | **Piksel 4** | | |
| Blue | Green | Red | Blue | Green | Red |
| 00110010 | 01011100 | 01001110 | 00110011 | 01011111 | 01100110 |

Gambar 2.4 Ilustrasi *pixel citra-stego*

## 2.4 Filter Pendeteksi Tepi (*Edge*)

Deteksi tepi (*edge detection*) merupakan salah satu proses pengolahan yang sering dibutuhkan pada analisa citra digital. Proses ini bertujuan untuk meningkatkan penampakan garis tepi pada suatu citra digital. Informasi tentang tepi merupakan informasi yang penting dalam pengenalan citra sehingga deteksi tepi merupakan proses yang sering dilakukan dalam pengolahan citra digital. Definisi tepi dibagi menjadi dua, yaitu :

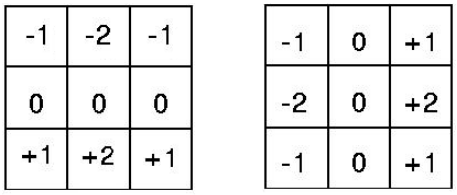
1. Citra kontinu : perubahan intensitas yang tajam antara titik yang bertetanggaan.
2. Citra diskrit : lokasi titik (*pixel*) dimana terjadi perubahan *gray level* secara mendadak, misalnya suatu titik hitam dikatakan tepi bila atau paling tidak mempunyai tetangga putih dan sebaliknya.

*Edge detection* adalah penelusuran gambar, dalam hal ini adalah *pixel* secara vertikal dan horizontal, sambil melihat terjadinya perubahan warna yang mendadak yang melebihi harga sensitifitas antara dua titik *pixel* yang berdekatan. Tujuan *edge detection* adalah untuk mengidentifikasi suatu daerah pada citra digital dimana terdapat perubahan yang besar dalam intensitas. Dengan deteksi tepi pada suatu sistem pemrosesan citra (berbasis komputer) akan menemukan tanda-tanda permukaan atau garis bentuk yang timbul dari suatu objek.

### 2.4.1 *Sobel*

*Sobel* adalah salah satu operator yang digunakan sebagai filter untuk pendeteksian tepi suatu citra digital. Pada operator *sobel* memiliki kernel (*Mask*) 3

× 3 untuk perhitungannya dalam setiap pixel. Operator *Sobel* yang digunakan sebagai berikut :

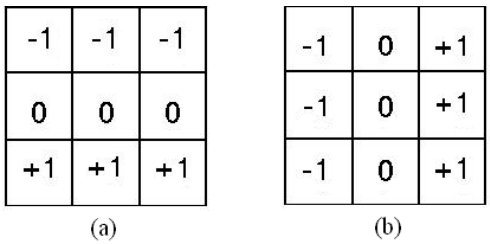



Gambar 2.5 *Mask* pada operator *Sobel*

Secara matematis, operator *sobel* menggunakan dua *mask*, *mask* yang pertama digunakan untuk menghitung selisih antara titik pada sisi vertical (a) dan mask yang kedua digunakan untuk menghitung selisih antara titik pada sisi horizontal (b).

### *2.4.2 Prewitt*

Seperti Operator *Sobel*, Operator *Prewit* juga diterapkan dalam dua buah mask. *Mask* yang pertama digunakan untuk menghitung selisih antara titik pada sisi horizontal, sehingga perhitungan yang dilakukan adalah megalikan matriks tetangga (*neighbor*) dengan *matriks mask* pertama yang hasilnya berupa penelusuran vertical (a), kemudian dilakukan perkalian antara *matriks neighbor* dengan *matriks mask* kedua yang hasilnya berupa penelusuran horizontal (b). Hasil dari kedua perkalian ini kemudian dijumlahkan sehingga menghasilkan penelusuran vertikal dan horizontal.




Gambar 2.6 *Mask* pada operator *Prewitt*

Secara matematis, operator *prewitt* menggunakan dua *mask* 3×3 yang digunakan dengan citra asli untuk mengkalkulasi perkiraan menyangkut suatu nilai *derivative* untuk perubahan horisontal dan vertikal.

Kombinasi antar kedua hasil operasi dengan mask tersebut bisa dilakukan dengan mengambil hasil penjumlahan, nilai maksimum, rerata atau rerata geometri.

$$G(x,y) = (|G1(x,y)| + |G2(x,y)|) \quad .....(2.6)$$

$$G(x,y) = \max(|G1(x,y)|, |G2(x,y)|) \quad .....(2.7)$$

$$G(x,y) = (|G1(x,y)| + |G2(x,y)|)/2 \quad .....(2.8)$$

$$G(x,y) = \sqrt{G1(x,y) * G1(x,y) + G2(x,y) * G2(x,y)} \quad .....(2.9)$$

Dengan G1(x,y) dan G2(x,y) adalah hasil operasi dengan mask 1 dan mask 2. Dalam praktek, formula (1) dan (2) biasanya lebih disukai dan lebih mudah dikerjakan karena mengandung jumlah operasi aritmetika yang lebih sedikit.

### *2.4.3 Laplacian of Gaussian*

Operator lain yang digunakan sebagai *filter* untuk mendeteksi tepi (*edge*) adalah operator *laplacian*. Berbeda dengan operator *gradient* yang menggunakan turunan pertama, operator *laplacian* menggunakan turunan kedua (*Zero Crossing Operator*). Operator ini dapat digunakan untuk horisontal dan vertikal. bentuk *mask* pada operator *laplacian* untuk operasi konvolusi seperti pada gambar dibawah ini.

| | | |
|---|---|---|
| 0 | −1 | 0 |
| −1 | 4 | −1 |
| 0 | −1 | 0 |

Laplacian 5 titik

| | | |
|---|---|---|
| −1 | −1 | −1 |
| −1 | 8 | −1 |
| −1 | −1 | −1 |

Laplacian 9 titik I

| | | |
|---|---|---|
| −2 | 1 | −2 |
| 1 | 4 | 1 |
| −2 | 1 | −2 |

Laplacian 9 titik II

Gambar 2.7 Mask pada operator Laplacian

Sebagai contoh untuk menyelesaikan sebuah citra digital mempunyai ukuran 5 × 5 pixel dengan nilai sebagai berikut :

| | | | | |
|---|---|---|---|---|
| 250 | 240 | 200 | 200 | 180 |
| 240 | 200 | 180 | 150 | 150 |
| 180 | 160 | ***160*** | 150 | 120 |
| 180 | 140 | 120 | 120 | 100 |
| 160 | 130 | 100 | 80 | 60 |

Menggunakan proses pendeteksian tepi dengan menggunakan operator Sobel, Prewit, dan *Laplacian of Gaussian,* dimana f(2,2) = 160 yaitu :

**a. Prewitt**

Horisontal = G1(x,y) = | (–1*200) + (–1*160) + (–1*140) + (1*150) + (1*150) + (1*120) |
= | – 80 | = 80

Vertikal = G2(x,y) = | (–1*200) + (–1*180) + (–1*150) + (1*140) + (1*120) + (1*120) |
= | – 150 | = 150

Maka h(2,2) bila menggunakan :

G(x,y) = ( | G1(x,y) | + | G2(x,y) | ) = 80 + 150 = **230**

G(x,y) = max ( | G1(x,y) | , | G2(x,y) | ) = **150**

G(x,y) = ( | G1(x,y) | + | G2(x,y) | ) / 2 = (80 + 150)/2 = **115**

$$G(x,y) = \sqrt{G1(x,y)*G1(x,y) + G2(x,y)*G2(x,y)}$$
$$= \sqrt{(80*80) + (150*150)}$$
$$= 170$$

**b. Sobel**

Horisontal = G1(x,y) = | (–1*200) + (–2*160) + (–1*140) + (1*150) + (2*150) + (1*120) |
= | – 90 | = 90

Vertikal = G2(x,y) = | (–1*200) + (–2*180) + (–1*150) + (1*140) + (2*120) + (1*120) |
= | – 210 | = 210

Maka h(2,2) bila menggunakan :

G(x,y) = ( | G1(x,y) | + | G2(x,y) | ) = 90 + 210 = **300**

$G(x,y) = \max ( | G1(x,y) | , | G2(x,y) | ) = \mathbf{210}$

$G(x,y) = ( | G1(x,y) | + | G2(x,y) | ) / 2 = (90 + 210)/2 = \mathbf{150}$

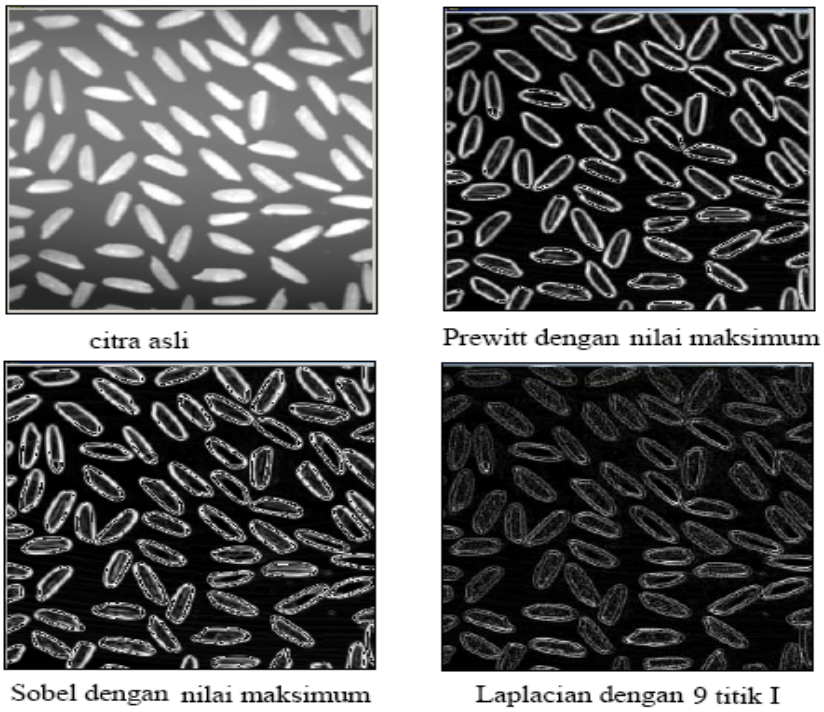
$$G(x,y) = \sqrt{G1(x,y) * G1(x,y) + G2(x,y) * G2(x,y)}$$
$$= \sqrt{(90*90) + (210*210)}$$
$$= 228{,}4 \approx 228$$

**c. Laplacian 9 titik 1**

$G1(x,y) = | (-1*200) + (-1*180) + (-1*150) + (-1*160) + (8*160) + (-1*150) + (-1*140) + (-1*120) + (-1*120) |$

$= \mathbf{60}$

Maka h(2,2) = **60**



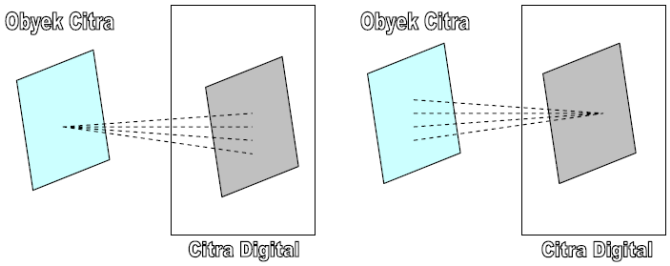

Gambar 2.8 Hasil dari proses pendeteksian tepi

## 2.5 Operasi Bertetanggaan

Operasi bertetanggaan **(***Neighborhood Operation***)** pada dasarnya adalah hubungan antara citra dengan sebuah filter (mask atau kernel). Nilai dari filter mask merupakan bobot kontribusi titik bertetanggaan terhadap operasi

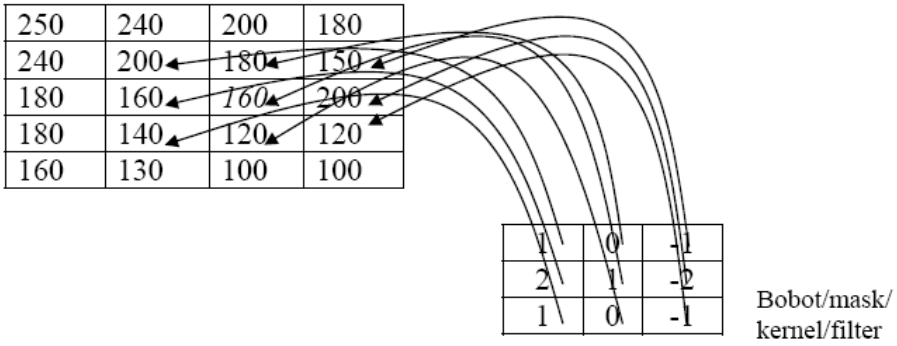
bertetanggaan. Sebuah citra yang ideal, apabila mampu mencerminkan kondisi sesungguhnya dari suatu obyek. Mempunyai hubungan satu-satu (one to one), satu titik pada obyek dipetakan tepat satu pixel di citra digital.

Tetapi pada kenyataannya, hubungan yang ada antara titik dalam obyek dengan titik pada citra digital adalah hubungan satu ke banyak (one to many) dan banyak ke satu (many to one). Ini dikarenakan sinyal yang dikirim oleh obyek citra mengalami penyebaran (divergensi), sehingga yang diterima oleh sensor atau detector tidak lagi berupa suatu titik, namun berupa luasan, atau sebaliknya satu titik pada sensor atau detector dapat menerima banyak sinyal dari beberapa bagian.




Gambar 2.9 Hubungan sesungguhnya antara obyek citra dan citra digital

Operasi citra digital yang berhubungan dengan kondisi diatas disebut operasi bertetangga (neighborhood operation). salah satu pengolahan citra yang berkaitan dengan operasi ini adalah : deteksi tepi (Edge Detection).




Gambar 2.10 Operasi bertetanggaan

Formula yang dipakai *SUM OF PRODUCTS* :

$$h(x,y) = \sum_{x=-M}^{M} \sum_{y=-N}^{N} f(u,v).g(x+u,y+v) \qquad \text{.....(2.5)}$$

x,y,u,v : posisi titik di dalam citra

m,n : batas titik tetangga yang masih memberikan pengaruh ke titik yang sedang ditinjau untuk arah horisontal dan vertikal.

Misal dari ilustrasi diatas, nilai pixel semula **160** menjadi :

h(x,y) = 1*200 + 0*180 – 1*150 + 2*160 + 1*160 – 2*200 + 1*140 + 0*120 – 1*120

= 150

## 2.6 *Steganalysis*

Seperti halnya teknik pandai telah dipikirkan untuk menyembunyikan informasi, sejumlah teknik pandai yang sama telah dirancang untuk mendeteksi informasi yang tersembunyi. Teknik ini dikenal sebagai *steganalysis. Steganalysis* adalah seni dalam mendeteksi pesan yang tersembunyi pada citra hasil *steganography*.

*Sample Pairs* dan *RS Analysis*, kedua teknik *steganalysis* tersebut adalah sangat akurat dalam memprediksi penambahan nilai angka atas *citra-stego* yang menggunakan penyisipan *Least Significant Bits*. *RS Analysis* membuat modifikasi kecil pada bit yang paling sedikit signifikan didalam suatu gambar kemudian menggunakan modifikasi ini dan sebuah fungsi diskriminasi untuk menggolongkan kelompok *pixels*. Penyebutan kelompok berdasar pada modifikasi yang mengijinkan kalkulasi dari suatu penambahan nilai angka yang diperkirakan.

*RS Analysis* adalah suatu kasus yang khusus dari *Sample Pairs Analysis*, yang juga menggunakan *bit* yang paling sedikit signifikan untuk membantu mengkalkulasi suatu penambahan nilai angka yang diperkirakan. *Sample Pairs Analysis* menggunakan *finite state machines* untuk menggolongkan kelompok pixels yang dimodifikasi dengan sebuah pola yang diberikan.

Citra yang tidak berisi steganography sering mempunyai suatu penambahan nilai angka yang alami sampai 3%, sedangkan gambar yang berisi informasi tersembunyi pada umumnya mempunyai penambahan nilai angka yang diperkirakan secara teliti mencerminkan jumlah informasi yang tersembunyi.

## 2.7 Kriteria Penilaian *Citra-Stego*

Jika sebuah citra digital disisipi dengan pesan (artinya, *citra-stego*), maka citra tersebut pasti mengalami *error* yang diakibatkan penyisipan pesan tersebut. Untuk menentukan tingkat kualitas suatu *citra-stego* antara citra sebelum dan sesudah disisipi pesan, maka diperlukan suatu kriteria yang dapat digunakan untuk menilai apakah *citra-stego* tersebut dapat menyembunyikan pesan yang dikandungnya dengan efektif sehingga kasat mata.

Maka adapun kriteria penilaian tersebut menggunakan kriteria obyektif yaitu yang didasarkan pada batas error yang diperbolehkan untuk citra yang diolah. Untuk citra asal p(x,y) dan citra yang telah disisipkan pesan p~(x,y) dengan ukuran yang sama M × N. parameter yang digunakan sebagai kriteria penilaian secara obyektif pada penelitian ini adalah :

*1. Average Absolute Difference*

Menghitung nilai rata-rata perbedaan absolut antara citra asli dengan *citra-stego*. Rata-Rata perbedaan absolut adalah:

AD = (1 / XY) * sum(|p(x,y) - p~(x,y)|) …..(2.10)

2. *Correlation Quality*

   Menghitung kualitas hubungan antara citra asli dengan *citra-stego*.

   Rumus yang digunakan adalah:

   CQ = sum(p(x,y)*p~(x,y)) / sum(p(x,y)) …..(2.11)

3. *LpNorm*

   Menghitung nilai menyangkut Lpnorm antara citra asli dengan *citra-stego*.

   Lp = ((1 / XY) * sum( (|p(x,y) - p~(x,y)|) ^ p)) ^ 1/p …..(2.12)

4. *Mean Squared Error* (MSE)

   Menghitung nilai rata-rata dari jumlah kuadrat *absolute error* antara citra asli dengan *citra-stego*. Rumus yang digunakan adalah:

   MSE = (1 / XY) * sum( (p(x,y) - p~(x,y))^2) …..(2.13)

5. *Normalised Cross Correlation*

   Menghitung nilai menyangkut hubungan antara citra asli dengan *citra-stego*. Rumus yang digunakan adalah:

   NC = sum(p(x,y)*p~(x,y)) / sum(p(x,y)^2) …..(2.14)

6. *Peak Signal To Noise Ratio* (PSNR)

   Menghitung nilai dari sinyal tertinggi untuk perbandingan *noise* antara citra asli dengan *citra-stego*. PSNR dihitung menggunakan rumusan berikut:

   PSNR = (XY * max( p(x,y)^2)) / sum( (p(x,y) - p~(x,y))^2) …..(2.15)

## *2.8* Java

Java merupakan bahasa pemrograman berorientasi objek dan bebas *platform*, dikembangkan oleh *SUN Micro System* dengan sejumlah keunggulan yang memungkinkan Java dijadikan sebagai bahasa pengembangan *enterprise*. Berikut ini beberapa keunggulan dari Java yaitu :

a. Berorientasi objek

Konsep pemrograman berorientasi objek tak lain dirancang agar dapat memandang pemrograman sebagai suatu kehidupan nyata. Ini membuat pengembangan software menjadi lebih mudah, karena seolah-olah berhubungan dengan kehidupan nyata, bukan lagi kehidupan 'dunia lain'. Java merupakan salah satu bahasa yang memiliki dukungan penuh terhadap konsep pemrograman berorientasi objek ini.

*b. Multiplatform*

Java merupakan bahasa pemrograman yang bisa dijalankan pada komputer platform yang berbeda, asalkan JVM telah terintall untuk *platform* itu. Jika membuat program Java di atas Windows, maka ia bisa dijalankan pada Linux, Unix, Macintosh. Selain itu, Java juga bisa digunakan untuk membuat program yang berjalan di atas berbagai *mobile device* (dengan J2ME), PDA, *embeded system*, dan lain-lain.

*c. Multithread*

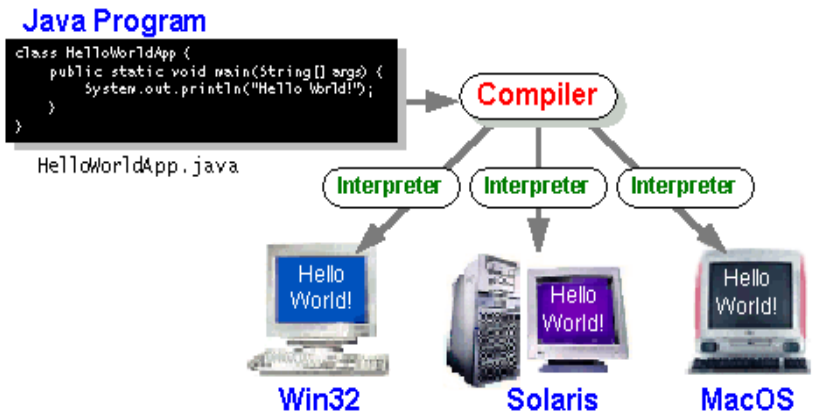
Fitur *multithread* digunakan untuk menjalankan sejumlah proses secara bersamaan. Dengan menggunakan fasilitas ini, dapat membuat program Java yang menjalankan beberapa perintah sekaligus, sehingga tidak perlu menunggu sebuah perintah selesai dikerjakan untuk mengerjakan perintah lain.

### *2.8.1* JVM *dan Byte Code*

Mengapa sebuah program Java bisa dijalankan pada semua sistem operasi? Jawabannya adalah karena keberadaan *Java Virtual Machine* (JVM) dan konsep byte-code. Karena hal inilah mengapa ada slogan khusus untuk Java, *write one, run anywhere*. Artinya hanya dengan membuat dan mengkompile sebuah program Java, program tersebut bisa dijalankan dimana saja.

Program Java yang ingin dijalankan (termasuk program Java lain yang diperlukan oleh program tersebut) harus dikompile terlebih dahulu dengan javac. Proses kompilasi program Java akan menghasilkan sejumlah file class dalam bentuk *byte-code*, file dengan ekstension ".class".

*Byte code* Java tidak bisa dijalankan seperti *binary code*. *Byte code* harus dijalankan pada JVM, yaitu mesin Java yang mengerti perintah-perintah yang ada *byte code*. JVM inilah yang selanjutnya berhubungan dengan sistem operasi untuk menjalankan perintah pada *byte code* tersebut Untuk setiap sistem operasi memiliki JVM yang berbeda. JVM harus sudah terinstall pada sistem agar dapat menjalankan program Java yang berbentuk *byte code* itu. Gambar berikut memperihatkan bagaimana sebuah file Java bisa dijalankan pada sistem operasi yang berbeda-beda, misalnya Windows, Solaris dan Macintos.




Gambar 2.11 Java dengan Sistem Operasi

### *2.8.2* J2SE *(Standard Edition)*

Yaitu fitur teknologi java yang dirancang untuk berjalan diatas PC dan workstation yang dapat berjalan diplatform sistem operasi linux, macintosh, windows, dll. SUN microsystem secara resmi membagi J2SE menjadi dua bagian besar yaitu J2SE *core* dan J2SE *desktop*. J2SE *core* memiliki teknologi turunan yang meliputi teknologi *security,* teknologi *debugging*, teknologi dokumentasi, teknologi database ,dan banyak teknologi turunan lainnya. Sedangkan J2SE *desktop* memiliki beberapa teknologi meliputi JRE (*Java Runtime Environment*) ,JFC (*Java Foundation Classes*), *Java Sound API*, dll. Berikut ini adalah langkah-langkah dalam menghasilkan suatu program dengan bahasa java, yaitu :

1. Mengkompile Program Java

Kompiler yang digunakan untuk mengkompile semua program Java, baik program aplikasi atau applet, adalah javac. Program ini terdapat pada direktori bin dari direktori dimana J2SDK diinstall. Sintaks untuk mengkompile program Java dengan javac:

javac <nama-file-java.java>

2. Menjalankan Program Java

Java memiliki 2 program untuk menjalankan program Java, yaitu: java dan appletviewer. Java digunakan untuk menjalankan program aplikasi dan appletviewer digunakan untuk menjalankan applet. Program java dan appletviewer akan menjalankan Java Virtual Machine (JVM) untuk menjalankan kedua bentuk program Java.

a. java

Sintaks untuk menjalankan program Java (*byte code*) dengan java:

java <nama-file-java>

Perhatikanlah bahwa pada saat menjalankan program Java tidak boleh mengikutkan nama ekstensi dari file class (.class).

b. appletviewer

Sintaks untuk menjalankan applet dengan appletviewer:

appletviewer <nama-file-html>