BAB 2
LANDASAN TEORI

2.1 Citra

Dalam penelitian ini terdapat istilah "citra" yang dalam Bahasa Inggris disebut *image.* Rinaldi [8] berpendapat bahwa secara harafiah, citra (*image*) adalah gambar pada bidang dwimatra (dua dimensi). Berdasarkan keterangan tersebut, dapat disimpulkan bahwa contoh citra antara lain adalah foto dan lukisan.

Citra ada dua macam, yaitu citra kontinu dan citra diskrit. Citra kontinu dihasilkan dari sistem optik yang menerima sinyal analog, misalnya mata manusia dan kamera analog. Citra diskrit dihasilkan melalui proses digitalisasi terhadap citra kontinu. Beberapa sistem optik dilengkapi dengan fungsi digitalisasi sehingga ia mampu menghasilkan citra diskrit, misalnya kamera digital dan scanner. Citra diskrit disebut juga citra digital. Komputer digital yang umum dipakai saat ini hanya dapat mengolah citra digital [8]. Citra yang akan diolah dalam penelitian ini adalah citra digital.

2.1.1 Citra Digital

Agar dapat diolah dengan menggunakan komputer digital, maka suatu citra harus direpresentasikan secara numerik dengan nilai-nilai diskrit. Proses representasi citra kontinu menjadi nilai-nilai diskrit disebut digitalisasi. Citra digital adalah hasil proses digitalisasi.

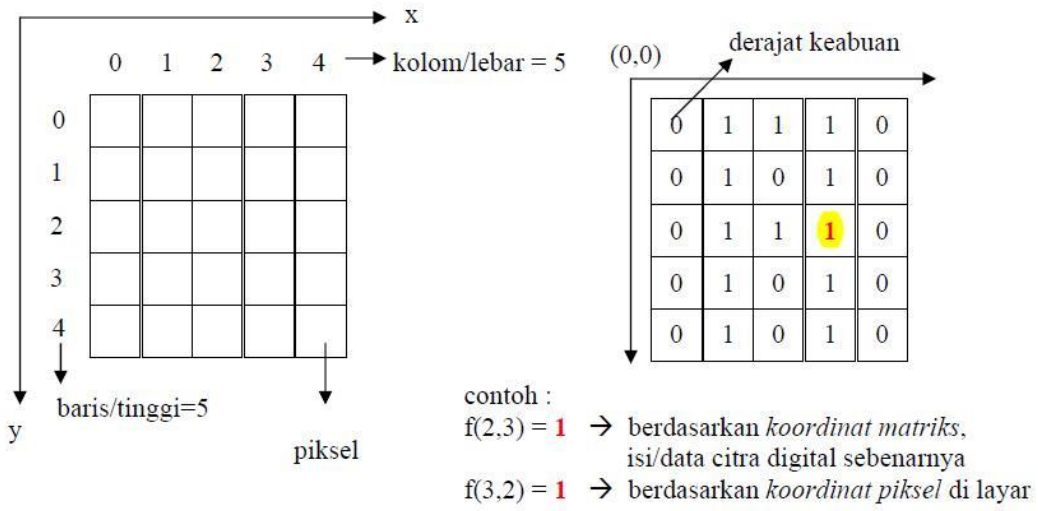
Suatu citra digital mempunyai fungsi dua dimensi f(x,y) yang memiliki ukuran M baris dan N kolom dimana x dan y adalah koordinat pada bidang dwimatra dan f(x,y) adalah intensitas cahaya (*brightness*) atau amplitudo atau derajat keabuan (*grey level*). Citra digital merupakan suatu matriks dimana indeks baris dan kolomnya menyatakan suatu titik pada citra tersebut dan elemen matriksnya (yang disebut sebagai elemen gambar atau *pixel*) menyatakan nilai derajat keabuan pada titik tersebut. Citra digital berukuran N x M (tinggi = N, lebar = M) dinyatakan dengan matriks N x M. Bentuk matriks citra digital dapat dilihat pada gambar 2.1.

N = jumlah baris $0 \leq y \leq N - 1$

M = jumlah kolom $0 \leq x \leq M - 1$ ….. (2.1)

L = maksimal warna intensitas $0 \leq f(x,y) \leq L - 1$

$$f(x,y) \approx \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,M-1) \\ f(1,0) & f(1,1) & \dots & f(1,M-1) \\ \vdots & \vdots & \vdots & \vdots \\ f(N-1,0) & f(N-1,1) & \dots & f(N-1,M-1) \end{bmatrix}$$



Gambar 2.1 Representasi Citra Digital 2 Dimensi

Citra digital pada umumnya berbentuk empat persegi panjang dengan dimensi ukurannya tinggi x lebar. Dimensi citra digital berisi blok-blok kecil yang berupa elemen gambar yang disebut pixel. Pixel-pixel tersebut memuat informasi warna yang menyusun suatu citra. Untuk menunjukkan tingkat pencahayaan pixel, digunakan bilangan bulat yang besarnya 8 *bit* (1 *byte*) untuk setiap pixelnya, dengan rentang antara 0-255, dimana 0 untuk warna hitam, 255 untuk warna putih, dan tingkat keabuan ditandai dengan nilai antara 0-255.

2.1.2 Tipe Citra Digital

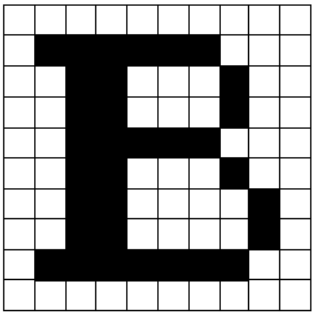
Berdasarkan format penyimpanan nilai warnanya, citra digital terbagi dalam tiga tipe yaitu :

1. Citra Biner

Citra biner (*binary image*) adalah citra yang hanya mempunyai dua nilai derajat keabuan: hitam dan putih. Pixel yang bernilai 1 melambangkan warna hitam, sedangkan pixel yang bernilai 0 adalah warna putih. Dinyatakan dalam suatu fungsi:

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

Berikut ini adalah contoh citra biner dan representasi nilai tiap pixel seperti pada gambar 2.2 [8]:



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

Gambar 2.2 Representasi Citra Biner

2. Citra *Grayscale* (Skala Keabuan)

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 :

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

Gambar 2.3 menunjukkan contoh citra skala keabuan 8-bit dengan ukuran 512 x 512 *pixel* [8].



Gambar 2.3 Citra Kapal

3. Citra Berwarna

RGB adalah suatu model warna yang terdiri dari merah, hijau, dan biru, digabungkan dalam membentuk suatu susunan warna yang luas. Setiap warna dasar, misalnya merah, dapat diberi rentang-nilai. Untuk monitor komputer, nilai rentangnya paling kecil = 0 dan paling besar = 255. Pilihan skala 256 ini didasarkan pada cara mengungkap 8 digit bilangan biner yang digunakan oleh mesin komputer. Dengan cara ini, akan diperoleh warna campuran sebanyak 256 x 256 x 256 = 16.777.216 jenis warna.

Sebuah jenis warna, dapat dibayangkan sebagai sebuah vektor di ruang 3 dimensi yang biasanya dipakai dalam matematika, koordinatnya dinyatakan dalam bentuk tiga bilangan, yaitu komponen-x, komponen-y dan komponen-z. Misalkan sebuah vektor dituliskan sebagai r = (x,y,z). Untuk warna, komponen-komponen tersebut digantikan oleh komponen R (*Red*), G (*Green*), B (*Blue*). Jadi, sebuah jenis warna dapat dituliskan sebagai berikut: warna = RGB(xxx, xxx, xxx). Putih = RGB (255,255,255), sedangkan untuk hitam= RGB(0,0,0). Gambar 2.4 menunjukkan contoh citra berwarna 8-bit. Representasi dalam citra digital dinyatakan dalam persamaan :

$$\begin{aligned} &fR(x,y)\ \Sigma\ [0\ldots255] \\ &fG(x,y)\ \Sigma\ [0\ldots255] \\ &fB(x,y)\ \Sigma\ [0\ldots255] \end{aligned} \qquad \ldots\ldots (2.4)$$



Gambar 2.4 Citra *Pepper*

2.2 Computer Vision

Pada hakikatnya, *computer vision* mencoba meniru cara kerja sistem visual manusia. Dalam proses penglihatan manusia, manusia melihat objek dengan menggunakan indera penglihatan yang berupa mata, lalu citra objek diteruskan ke otak untuk diinterpretasikan sehingga manusia mengerti objek apa yang tampak. Hasil interpretasi ini kemudian digunakan untuk pengambilan keputusan [8].

Tabel 2.1 Garis Besar *Human Vision* dan *Computer Vision*

Human Vision	Computer Vision
Menggunakan mata dan *visual cortex* di dalam otak.	Menggunakan kamera-kamera yang terhubung pada sistem komputer.
Menemukan dari gambar objek apa yang ada dalam penglihatan, dimana posisinya, bagaimana mereka bergerak, dan apa bentuknya.	Secara otomatis menginterpretasi gambar-gambar dan mencoba untuk mengerti isinya seperti pada human vision.

Secara garis besar, *computer vision* adalah sebuah teknologi mesin yang mampu mengenali objek yang diamati. Kemampuan untuk mengenali ini merupakan kombinasi dari pengolahan citra dan pengenalan pola. Pengolahan citra adalah proses awal dalam *computer vision* untuk menghasilkan citra yang lebih baik atau lebih mudah diinterpretasikan, sedangkan pengenalan pola adalah proses

identifikasi objek pada citra. Proses-proses dalam *computer vision* secara garis besar dapat dibagi menjadi:

1. Proses mengakuisisi citra digital (*Image Acquisition*)
2. Proses pengolahan citra (*Image Processing*)
3. Proses analisis data citra (*Image Analysis*)
4. Proses pemahaman data citra (*Image Understanding*)

Computer vision adalah sebuah tantangan untuk mencoba meniru indera manusia yang paling kuat. Bidang ini memiliki sejumlah aplikasi dalam dunia industri, antara lain:

1. Otomatisasi Proses Industri
2. Obat-obatan dan Diagnostik
3. Hiburan: Film dan Video
4. Keamanan dan Pengawasan (*Surveillance*)
5. Visualisasi dan *Augmented Reality*
6. Komunikasi
7. Interaksi Manusia dengan Komputer
8. Kemiliteran dan Penelitian Ruang Angkasa

2.2.1 Deteksi Manusia

Pendeteksian memiliki arti menemukan objek yang diinginkan dan menetukan posisinya dalam gambar. Pendeteksian disini bukan berasal dari dinamika sistem dan responnya tidak berdasarkan konsistensi sementara (*temporal consistency*) melainkan deteksi yang informasinya bersumber dari sebuah gambar. Contohnya adalah *face detection* dan *people detection* (deteksi manusia).

Dalam deteksi manusia, khususnya yang dilakukan oleh Dalal dan Triggs [1], terdapat beberapa kesulitan atau tantangan, antara lain:

1. Berbagai macam pose tubuh.
2. Penampilan dan pakaian yang bervariasi.
3. Latar belakang dan lingkungan yang kompleks.
4. Pencahayaan yang tidak dibatasi.
5. Ukuran yang berbeda.

Deteksi manusia berfungsi untuk mendeteksi objek manusia dalam sebuah citra. Dalam penelitian ini, definisi citra manusia adalah area persegi dalam citra yang memiliki wujud penampakan manusia. Dengan mempertimbangkan tantangan yang ada, maka disusunlah sebuah kriteria citra manusia dalam penelitian seperti terlihat pada tabel 2.2 berikut ini:

Tabel 2.2 Kriteria Citra Manusia dalam Penelitian

No	Kategori	Keterangan
1.	Wujud	yang dimaksud adalah manusia seutuhnya yang terlihat dari kaki hingga kepala
2.	Arah / Orientasi	dapat terlihat dari depan, belakang, maupun samping
3.	Posisi Gerak	bervariasi dari berjalan, diam, membawa barang, membaca, berlari, dan sebagainya dengan prioritas manusia sedang dalam posisi berdiri
4.	*Gender* / Jenis Kelamin	tidak terbatas pada laki-laki atau wanita saja, bisa keduanya
5.	Usia	prioritas deteksi pada manusia mulai dari usia remaja, dewasa, dan usia lanjut
6.	Postur Tubuh	dapat bervariasi dari kurus hingga gemuk
7.	Tinggi Badan	lebih diprioritaskan pada manusia dengan tinggi diatas 1 meter
8.	Pakaian	dapat terdiri dari berbagai macam jenis pakaian dan berbagai jenis warna yang umum digunakan sehari-hari
9.	Pencahayaan	diutamakan manusia berada dalam lingkungan yang terang / siang hari
10.	Lokasi	diutamakan berada pada tempat-tempat umum atau pusat keramaian

Beberapa contoh citra manusia dapat dilihat pada gambar 2.5 berikut ini:



Gambar 2.5 Contoh Gambar Dalam Database Penelitian

Dalam gambar 2.5 manusia dalam gambar selalu berdiri namun berada dalam berbagai variasi pose, penampilan, pakaian, pencahayaan, dan latar belakang.

Sedangkan gambar “citra manusia” yang dimaksud dalam penelitian ini adalah area persegi merah hasil deteksi seperti pada gambar 2.6 [1].



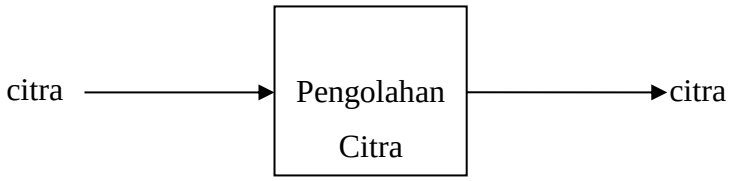
Gambar 2.6 Gambar Citra Manusia Penelitian Dalal dan Triggs

2.3 Pengolahan Citra Digital

Pengolahan citra adalah pemrosesan citra dengan menggunakan komputer agar kualitasnya lebih baik. Tujuannya agar citra dapat dengan mudah diinterpretasi oleh manusia atau komputer. Pada gambar 2.7 digambarkan bahwa teknik pengolahan citra memanipulasi citra menjadi citra lain. Jadi, masukannya adalah citra dan keluarannya juga citra. Namun citra keluaran mempunyai kualitas lebih baik daripada citra masukan

Pada umumnya, operasi-operasi pada pengolahan citra diterapkan pada citra bila [9]:

a. Perbaikan atau modifikasi citra perlu dilakukan untuk meningkatkan kualitas penampakan atau menonjolkan beberapa aspek informasi yang terkandung di dalam citra,
b. Elemen di dalam citra perlu dikelompokkan, dicocokkan, atau diukur,
c. Sebagian citra perlu digabung dengan citra yang lain.



Gambar 2.7 Pengolahan Citra

2.3.1 Metode Ubah Nilai RGB ke *Grayscale*

Untuk memudahkan pengolahan citra, biasanya citra RGB akan dikonversi menjadi citra skala keabuan (*grayscale*). Karena pada citra RGB pengolahan citra akan dilakukan dengan menghitung pada 3 *channel* warna, sedangkan pada citra skala keabuan cukup 1 *channel* warna sehingga memudahkan proses komputasi. Format citra skala keabuan memakai warna hitam sebagai warna minimal (0) dan warna putih (255) sebagai warna maksimalnya, sehingga warna antaranya adalah abu-abu seperti ditunjukkan pada gambar.

Berdasarkan catatan dokumentasi GIMP [10], dapat kita telusuri bahwa terdapat tiga macam metode algoritma untuk mengubah nilai R G B menjadi *grayscale* antara lain:

1. *Lightness*

   Algoritmanya adalah mencari nilai tertinggi dan terendah dari nilai R G B, kemudian nilai tertinggi dan terendah tersebut dijumlahkan lantas dikalikan dengan 0.5 (dibagi 2). Secara matematis dapat dirumuskan :

   Grayscale = (max(R,G,B) + min(R,G,B)) * 0.5 ….. (2.5)

2. *Average*

   Algoritmanya adalah dengan menjumlahkan seluruh nilai R G B, kemudian dibagi 3, sehingga diperoleh nilai rata-rata dari R G B, nilai rata-rata itulah yang dapat dikatakan sebagai grayscale. Rumus matematisnya adalah :

   Grayscale = (R + G + B) / 3 ….. (2.6)

3. *Luminosity*

   Algoritmanya adalah dengan mengalikan setiap nilai R G B dengan konstanta tertentu yang sudah ditetapkan nilainya, kemudian hasil perkalian seluruh nilai R G B dijumlahkan satu sama lain. Rumus matematisnya adalah :

   Grayscale = (0.21 * R) + (0.72 * G) + (0.07 * B) ….. (2.7)

Dari ketiga macam algoritma diatas, masing-masing metode walaupun sama-sama menghasilkan warna *grayscale*, namun tingkat derajat *grayscale*-nya berbeda-beda, sehingga setiap metode menghasilkan warna abu-abu yang tidak sama persis satu sama lain. Berikut gambaran perbedaan hasil gambar dengan ketiga metode *grayscale* :



(a) (b) (c) (d)

Gambar 2.8 Hasil Metode Grayscale

Gambar 2.8a adalah gambar asli yang berformat RGB. Gambar 2.8b, 2.8c, dan 2.8d masing-masing adalah hasil konversi grayscale dengan metode *Luminosity* (2.8b), *Lightness* (2.8c), dan *Average* (2.8d).

Untuk konversi RGB ke *grayscale* dengan metode *Luminosity*, konstanta yang digunakan lebih dititikberatkan pada *channel* warna hijau (*green*) karena mata manusia lebih sensitif pada warna hijau daripada warna lainnya. Rekomendasi nilai konstanta *Luminosity* yang umum digunakan pada perangkat pengolahan citra saat ini ada 2 tipe, yaitu BT.709 [11] dan BT.601 [12] yang direkomendasikan oleh *International Telecomuncation Union* (ITU). BT.709 adalah standar format untuk *high-definition television* (HDTV), sedangkan BT.601 adalah standar format untuk *standar-definition television* (SDTV).

Adapun nilai konstanta *Luminosity* yang diberikan pada rekomendasi BT.709 adalah seperti pada persamaan 2.7, sedangkan untuk rekomendasi BT.601 dapat dilihat pada rumus matematis berikut:

Grayscale = (0.299 * R) + (0.587 * G) + (0.114 * B) ….. (2.8)

2.3.2 Histogram Citra

Histogram citra adalah grafik yang menggambarkan penyebaran nilai-nilai intensitas *pixel* dari suatu citra atau bagian tertentu di dalam citra. Dari sebuah histogram dapat diketahui frekuensi kemunculan nisbi (*relative*) dari intensitas pada citra tersebut. Histogram juga dapat menunjukkan banyak hal tentang kecerahan (*brightness*) dan kontras (*contrast*) dari sebuah gambar [8].

Umumnya setiap *pixel* memiliki nilai derajat keabuan. Misalkan citra degan L derajat keabuan, yaitu dari nilai 0 sampai $L-1$. Secara matematis histogram citra dihitung dengan rumus

$$h_i = \frac{n_i}{n} \quad , i = 0,1,\ldots, L-1 \qquad \text{….. (2.9)}$$

yang dalam hal ini,

n_i = jumlah *pixel* yang memiliki derajat keabuan i

n = jumlah seluruh *pixel* di dalam citra

Sebagai contoh penghitungan histogram citra, misalkan matriks citra digital berukuran 8 x 8 *pixel* dengan derajat keabuan dari 0 sampai 15 (ada 16 buah derajat keabuan):

$$\begin{bmatrix} 3 & 7 & 7 & 8 & 10 & 12 & 14 & 10 \\ 2 & 0 & 0 & 0 & 1 & 8 & 15 & 15 \\ 14 & 6 & 5 & 9 & 8 & 10 & 9 & 12 \\ 12 & 12 & 11 & 8 & 8 & 10 & 11 & 1 \\ 0 & 2 & 3 & 4 & 5 & 13 & 10 & 14 \\ 4 & 5 & 0 & 0 & 1 & 0 & 2 & 2 \\ 15 & 13 & 11 & 10 & 9 & 9 & 8 & 7 \\ 2 & 1 & 0 & 10 & 11 & 14 & 13 & 12 \end{bmatrix}$$

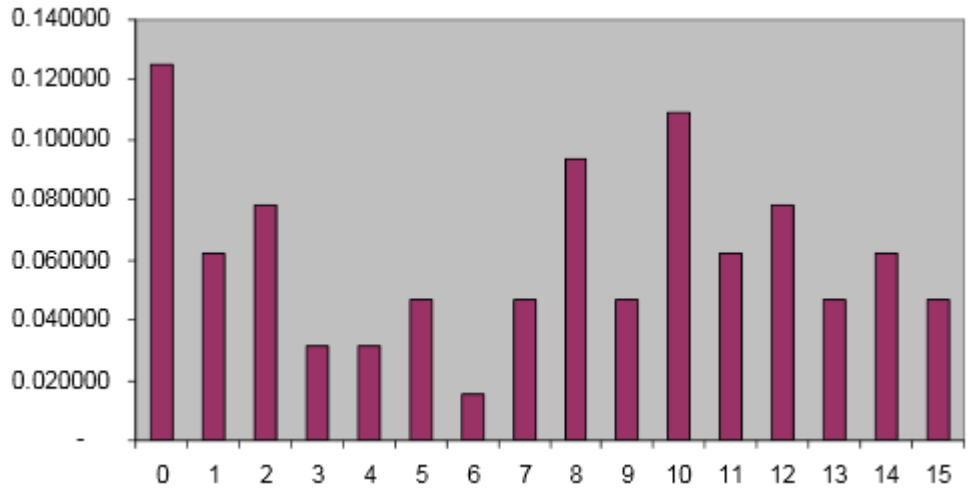
Gambar 2.9 Matriks Citra Digital

Tabulasi perhitungan histogramnya dapat dilihat pada tabel 2.3. Dapat dilihat bahwa semakin besar nilai n_i maka semakin besar pula nilai h_i [8].

Tabel 2.3 Perhitungan Histogram

i	n_i	$h_i = n_i / n$ (n=64)
0	8	0.125
1	4	0.0625
2	5	0.078125
3	2	0.03125
4	2	0.03125
5	3	0.046875
6	1	0.015625
7	3	0.0468875
8	6	0.09375
9	3	0.046875
10	7	0.109375
11	4	0.0625
12	5	0.078125
13	3	0.046875
14	4	0.0625
15	3	0.046875

Histogram citra digital dari matriks citra digital diatas dapat dilihat pada Gambar 2.10 berikut ini:



Gambar 2.10 Histogram Citra Digital

2.3.3 Konvolusi Fungsi Dua Dimensi

Salah satu operasi mendasar dalam pengolahan citra adalah operasi konvolusi. Konvolusi 2 buah fungsi f(x) dan g(x) didefinisikan sebagai berikut [8]:

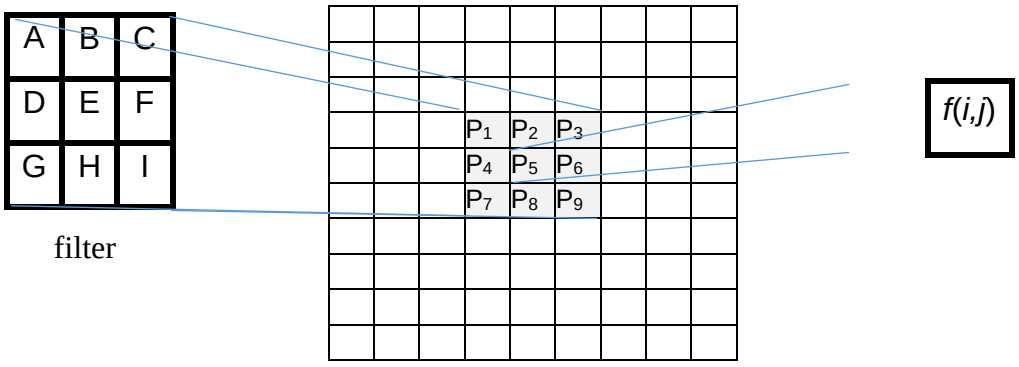
$$h(x) = f(x) * \mathrm{g(x)} = \int_{-\infty}^{\infty} f(a)g(x-a)da \qquad \text{..... (2.10)}$$

yang dalam hal ini, tanda * menyatakan operator konvolusi, dan peubah (*variable*) *a* adalah peubah bantu (*dummy variable*).

Untuk fungsi diskrit, konvolusi didefinisikan sebagai

$$h(x) = f(x) * g(x) = \sum_{a=-\infty}^{\infty} f(a)g(x-a) \qquad \text{..... (2.11)}$$

Pada persamaan 2.11, $g(x)$ disebut juga kernel konvolusi atau kernel penapis (*filter*). Dalam citra digital, filter konvolusi dinyatakan dalam bentuk matriks (umumnya 3x3, namun ada juga yang berukuran 2x2 atau 2x1 atau 1x2). Ukuran matriks ini, biasanya lebih kecil dari ukuran citra. Setiap elemen matriks, disebut koefisien konvolusi. Ilustrasi konvolusi, ditunjukkan pada gambar 2.11.



$f(i,j)$ = A p1 + B p2 + C p3 + D p4 + E p5 + F p6 + G p7 + H p8 + I p9

Gambar 2.11 Ilustrasi Konvolusi

Operasi konvolusi dilakukan dengan menggeser *filter* konvolusi *pixel* per *pixel*. Hasil konvolusi, disimpan di dalam matriks yang baru. Misalkan citra $f(x,y)$ yang berukuran 5 x 5 dan sebuah *mask* yang berukuran 3 x 3 masing – masing adalah sebagai berikut:

$$f(x,y) = \begin{bmatrix} 4 & 4 & 3 & 5 & 4 \\ 6 & 6 & 5 & 5 & 2 \\ 5 & 6 & 6 & 6 & 2 \\ 6 & 7 & 5 & 5 & 3 \\ 3 & 5 & 2 & 4 & 4 \end{bmatrix} \qquad g(x,y) = \begin{bmatrix} 0 & -1 & 0 \\ -1 & \bullet\, 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

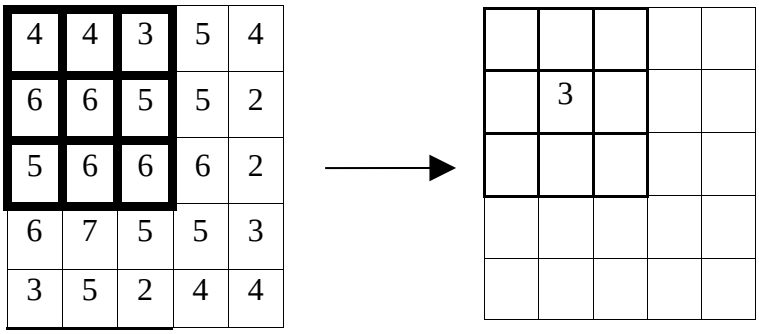
(Keterangan : Tanda • menyatakan posisi (0,0) dari *kernel*)

Operasi konvolusi antara citra f(x,y) dengan mask g(x,y),

$$f(x,y) * g(x,y) \qquad \text{….. (2.12)}$$

dapat diilustrasikan sebagai berikut :

(1) Tempatkan *mask* pada sudut kiri atas, kemudian hitung nilai *pixel* pada posisi (0,0) dari *mask* :

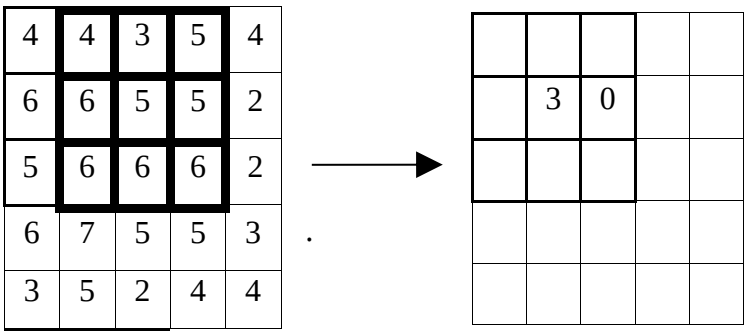


Gambar 2.12 Konvolusi Langkah Pertama

Hasil konvolusi = 3. Nilai ini dihitung dengan cara berikut :

(0 x 4) + (-1 x 4) + (0 x 3) + (-1 x 6) + (4 x 6) + (-1 x 5) + (0 x 5) + (-1 x 6) + (0 x 6) = 3

(2) Geser *mask* satu *pixel* ke kanan, kemudian hitung nilai *pixel* pada posisi (0,0) dari *mask* :

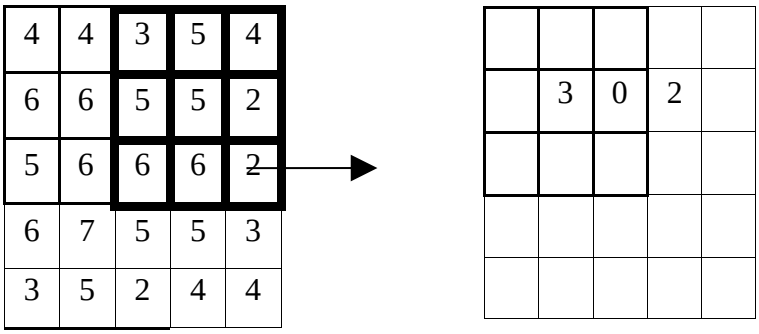


Gambar 2.13 Konvolusi Langkah Kedua

Hasil konvolusi = 0. Nilai ini dihitung dengan cara berikut :

(0 x 4) + (-1 x 3) + (0 x 5) + (-1 x 6) + (4 x 5) + (-1 x 5) + (0 x 6) + (-1 x 6) + (0 x 6) = 0

(3) Geser *mask* satu *pixel* ke kanan, kemudian hitung nilai *pixel* pada posisi (0,0) dari mask :



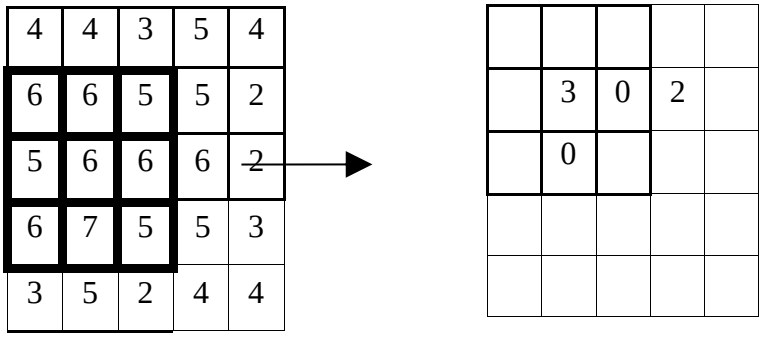
Gambar 2.14 Konvolusi Langkah Ketiga

Hasil konvolusi = 2. Nilai ini dihitung dengan cara berikut :

(0 x 3) + (-1 x 5) + (0 x 4) + (-1 x 5) + (4 x 5) + (-1 x 2) + (0 x 6) + (-1 x 6) + (0 x 2) = 2

(4) Selanjutnya, geser *mask* satu *pixel* ke bawah, kemudian lakukan konvolusi dari sisi kiri citra. Setiap kali konvolusi, geser *mask* satu *pixel* ke kanan :

(i)

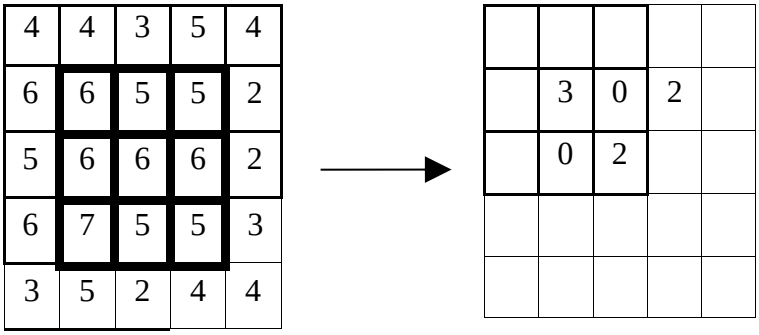


Gambar 2.15 Konvolusi Langkah Keempat

Hasil konvolusi = 0. Nilai ini dihitung dengan cara berikut :

(0 x 6) + (-1 x 6) + (0 x 5) + (-1 x 5) + (4 x 6) + (-1 x 6) + (0 x 6) + (-1 x 7) + (0 x 5) = 0

(ii)

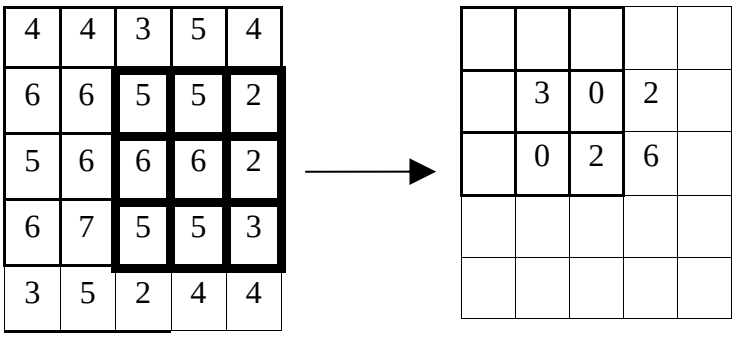


Gambar 2.16 Konvolusi Langkah Kelima

Hasil konvolusi = 2. Nilai ini dihitung dengan cara berikut :

(0 x 6) + (-1 x 5) + (0 x 5) + (-1 x 6) + (4 x 6) + (-1 x 6) + (0 x 7) + (-1 x 5) + (0 x 5) = 2

(iii)

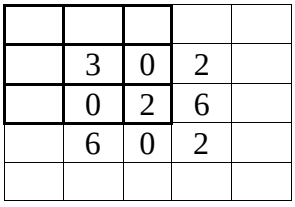


Gambar 2.17 Konvolusi Langkah Keenam

Hasil konvolusi = 6. Nilai ini dihitung dengan cara berikut :

(0 x 5) + (-1 x 5) + (0 x 2) + (-1 x 6) + (4 x 6) + (-1 x 2) + (0 x 5) + (-1 x 5) + (0 x 3) = 6

Dengan cara serupa, *pixel –pixel* pada baris ketiga dikonvolusi sehingga menghasilkan :



Gambar 2.18 Hasil Konvolusi

Sebagai catatan, jika hasil konvolusi menghasilkan nilai *pixel* negatif, maka nilai tersebut dijadikan 0. Sebaliknya, jika hasil konvolusi, menghasilkan nilai *pixel* lebih besar dari nilai keabuan maksimum, maka nilai tersebut dijadikan nilai keabuan maksimum.

Masalah timbul, apabila *pixel* yang dikonvolusi, adalah *pixel* pinggir *(border).* Karena, beberapa koefisien konvolusi, tidak dapat dapat diposisikan, pada *pixel-pixel* citra (efek menggantung). Seperti contoh di bawah ini:

4	4	3	5	4	?
6	6	5	5	2	?
5	6	6	6	2	?
6	*1*	5	5	3	
3	*5*	2	4	4	

Gambar 2.19 Efek Menggantung

Masalah menggantung seperti ini, selalu terjadi pada *pixel – pixel* pinggir kiri, kanan, atas dan bawah. Solusi untuk masalah ini adalah :

a. *Pixel – pixel* pinggir diabaikan, tidak di konvolusikan. Solusi ini banyak dipakai di dalam pustaka fungsi – fungsi pengolahan citra. Dengan cara seperti ini, maka *pixel – pixel* pinggir nilainya tetap sama seperti citra asal. Dalam hal ini, nilai *pixel – pixel* pinggir sama dengan nilai *pixel* semula.
b. Duplikasi elemen citra, misalnya elemen kolom pertama disalin ke kolom M+1, begitu juga sebaliknya, lalu konvolusi dapat dilakukan terhadap *pixel – pixel* pinggir tersebut.
c. Elemen yang ditandai dengan "?" diasumsikan bernilai 0 atau konstanta yang lain, sehingga konvolusi *pixel – pixel* pinggir dapat dilakukan.

Solusi, dengan ketiga pendekatan diatas mengasumsikan, bagian pinggir citra lebarnya sangat kecil (hanya satu pixel) relatif dibandingkan dengan ukuran citra. Sehingga, *pixel – pixel* pinggir, tidak memperlihatkan efek yang kasat mata.

4	4	3	5	4
6	4	0	8	2
5	0	2	6	2
6	6	0	2	3
3	5	2	4	4

Gambar 2.20 *Pixel* Pinggir (Tidak Diarsir) Tidak Dikonvolusi

Algoritma kovolusi, citra N x M dengan *filter* yang berukuran 3 x 3. *Pixel* yang dikonvolusi, adalah elemen (i,j). Delapan buah *pixel* yang bertetangga dengan *pixel* (i,j) diperlihatkan pada gambar 2.21.

i-1,j-1	i-1,j	i-1,j+1
i,j-1	i,j	i,j+1
i+1,j-1	i+1,j	i+1,j+1

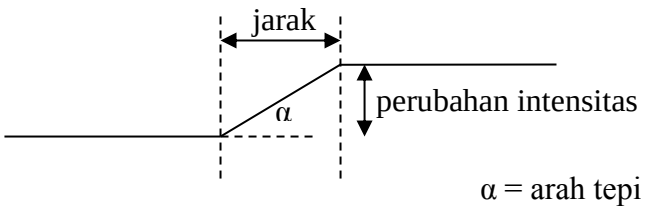
Gambar 2.21 *Pixel – pixel* Utama Dikonvolusi

Operasi konvolusi, merupakan komputasi pada aras lokal. Karena komputasi dilakukan, untuk suatu *pixel* tetangga pada citra masukannya. Konvolusi digunakan dalam proses pengolahan citra. Seperti perbaikan kualitas citra, penghalusan maupun pelembutan citra, penghilangan derau, deteksi tepi, penajaman tepi.

2.4 Pendeteksian Tepi

Dalam pengolahan citra terdapat satu istilah yaitu analisis citra. Analisis citra bertujuan mengidentifikasi parameter-parameter yang diasosiasikan dengan ciri (*feature*) dari objek di dalam citra, untuk selanjutnya parameter tersebut digunakan dalam menginterpretasi citra. Analisis citra pada dasarnya terdiri dari tiga tahapan: ekstraksi ciri (*feature extraction*), segmentasi, dan klasifikasi [8]. Dalam tahap ekstraksi ciri, kemampuan pendeteksian tepi (*edge*) dari objek dalam citra adalah faktor kuncinya.

Tepi *(edge),* adalah perubahan nilai intensitas derajat keabuan yang mendadak (besar) dalam jarak yang singkat (Gambar 2.20). Perbedaan intensitas inilah yang menampakkan rincian pada gambar. Tepi, biasanya terdapat pada batas, antara dua daerah berbeda pada suatu citra. Tepi, dapat diorientasikan dengan suatu arah dan arah ini berbeda-beda pula, bergantung pada perubahan intensitas.



Gambar 2.22 Model Tepi Satu Dimensi

Terdapat tiga macam tepi dalam citra digital, yaitu :

a. Tepi curam

   Tepi, dengan perubahan intensitas yang tajam. Arah tepi 90°.

b. Tepi landai

   Disebut juga tepi lebar, yaitu tepi dengan sudut arah yang kecil. Tepi landai, dapat dianggap terdiri dari sejumlah tepi – tepi lokal berdekatan lokasinya.

c. Tepi mengandung derau (*noise*)

   Umumnya, tepi yang terdapat pada aplikasi *computer vision* menggandung derau. Operasi peningkatan kualitas citra (*image enhancement*), dapat dilakukan terlebih dahulu sebelum pendeteksian tepi

Tujuan pendeteksian tepi adalah untuk meningkatkan penampakan garis batas suatu daerah atau objek dalam citra. Teknik yang dapat digunakan untuk mendeteksi tepi antara lain:

1. Operator gradien pertama (*Differential Gradient*)
2. Operator turunan kedua (*Laplacian*)
3. Operator kompas (*Compass Operator*)

Contoh pendeteksian tepi pada sebuah citra dapat dilihat pada Gambar 2.23 berikut. *Pixel* citra yang terdeteksi sebagai tepi digambarkan dengan warna putih sedangkan *pixel* yang terdeteksi bukan tepi digambarkan dengan warna hitam.

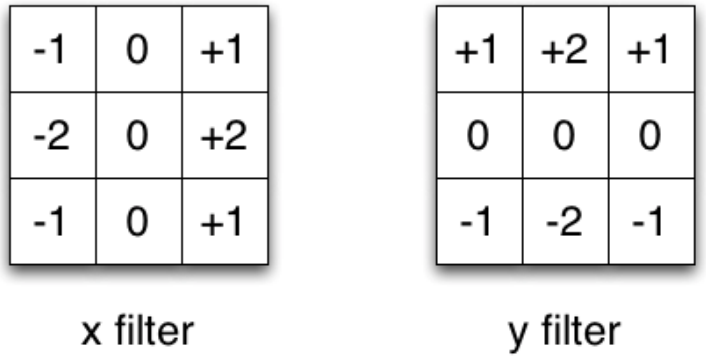


Gambar 2.23 Deteksi Tepi Citra

Pendeteksian tepi adalah salah satu elemen terpenting dalam analisa citra dan *computer vision*, karena deteksi tepi memainkan peran yang signifikan pada banyak penerapan pengolahan citra, terutama untuk *machine vision*. Berbagai metode *computer vision* mengandalkan pendeteksian tepi sebagai tahap *pre-processing*. Namun, tidak satupun algoritma pendeteksian tepi dapat dengan sukses menemukan tepi untuk berbagai macam citra dan pada saat ini tidak ada ukuran kuantitatif tertentu yang diberikan untuk kualitas dari pendeteksian tepi.

2.4.1 Pendeteksian Tepi Operator Sobel

Salah satu teknik yang digunakan untuk mendeteksi tepi adalah dengan menghitung turunan pertama (*gradient*). Salah satu operator gradien pertama yang dapat digunakan untuk proses deteksi tepi dalam citra adalah operator gradien Sobel. Berikut adalah *mask* konvolusi dari filter Sobel:



Gambar 2.24 Filter Sobel

Dengan menggunakan filter Sobel pada sumbu x dan y, maka nilai gradien (|G|) dapat dihitung dengan menggunakan rumus:

$$|G| = \sqrt{Gx^2 + Gy^2} \qquad \ldots(2.13)$$

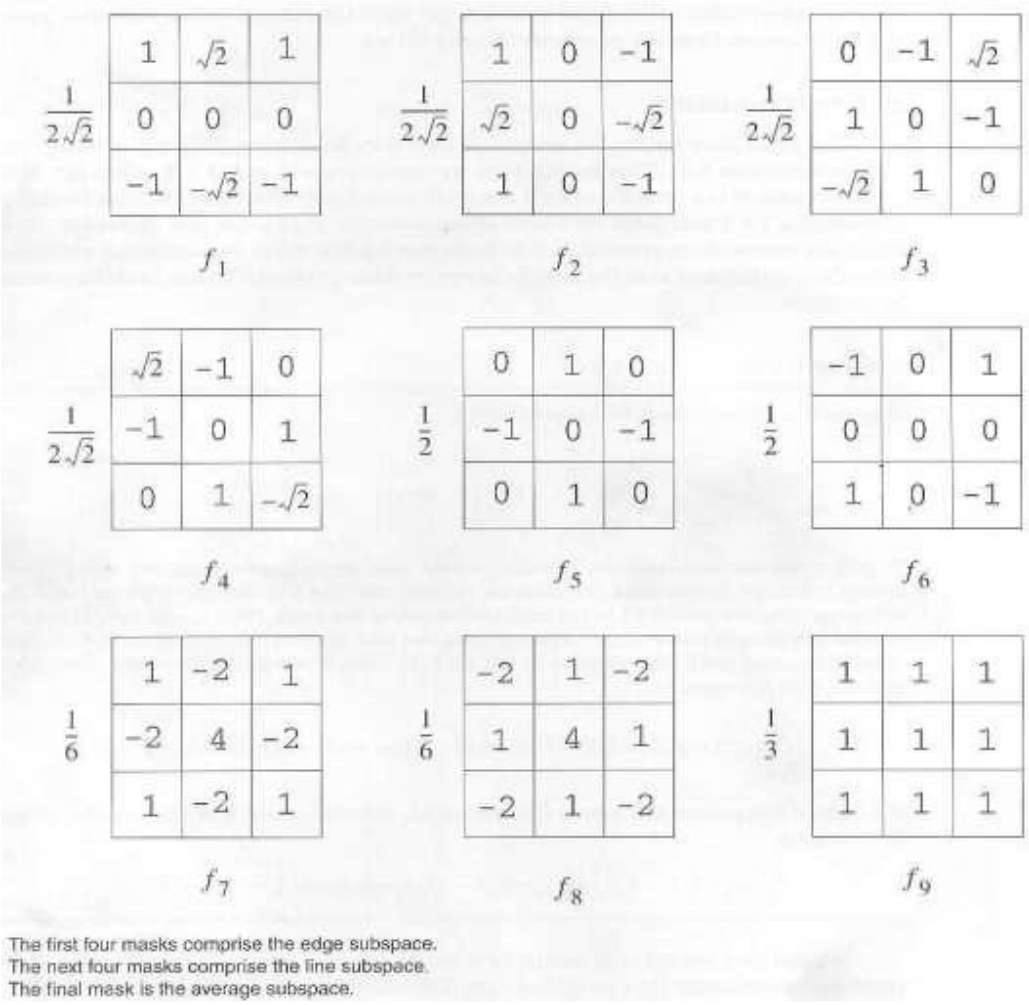
Sedangkan nilai arah gradien dapat dihitung dengan fungsi:

$$\theta = \arctan \frac{G_x}{G_y} \qquad \ldots(2.14)$$

2.4.2 Pendeteksian Tepi Frei-Chen

Deteksi tepi yang menggunakan Frei Chen *mask* yang diimplementasikan oleh pemetaan vektor intensitas menggunakan transformasi linear dan kemudian mendeteksi tepi berdasarkan sudut antara intensitas vektor dan diproyeksikan kedalam ruang bagian tepi.

Frei Chen mask adalah *mask* yang unik, dimana semua berisi basis vektor. Mask ini diimplementasikan pada area gambar 3x3 piksel dan diwakili dengan sejumlah sembilan Frei Chen mask.



Gambar 2.25 *Frei-Chen Mask*

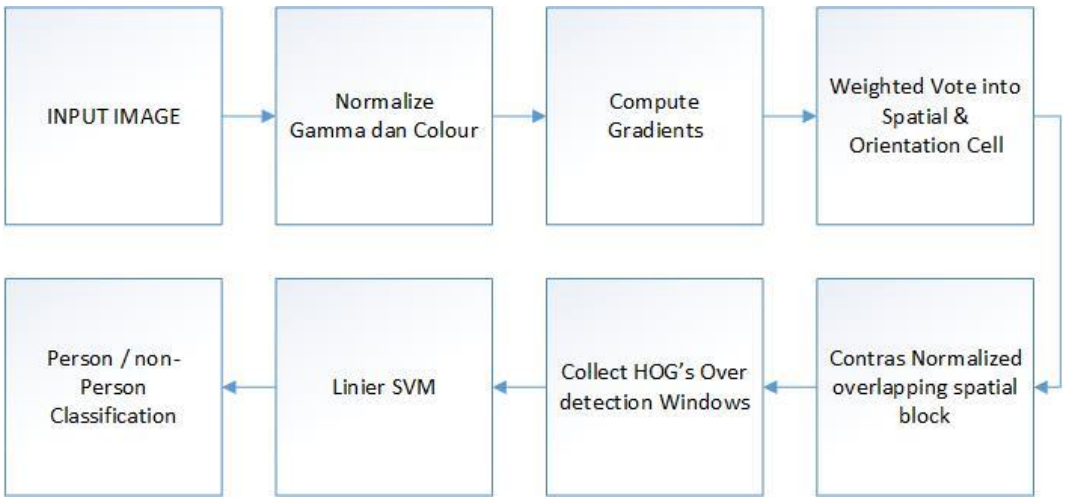
Dalam hal ini, kernel 1 sampai 4 digunakan untuk *subspace* tepi. 4 kernel selanjutnya untuk menghasilkan *subspace* garis. *Mask* terakhir untuk rata-rata *subspace*.

Jadi citra digital yang akan di deteksi tepi akan diproyeksikan pada *mask frei-chen* kemudian dilakukan perhitungan, hasil dari perhitungan tersebut diambil nilai bobotnya yang tidak bernilai nol, sehingga didapatkan hasil deteksi tepi [4].

2.5 Histograms of Oriented Gradients

Tujuan dari proses analisis citra adalah mengidentifikasi parameter-parameter yang diasosiasikan dengan ciri (*feature*) dari objek di dalam citra, untuk selanjutnya parameter tersebut digunakan dalam menginterpretasi citra. Tahap pertama analisis citra yaitu ekstraksi ciri (*feature extraction*).

Histograms of Oriented Gradients (HOG) adalah pendeskripsi ciri (*feature descriptor*) yang digunakan dalam *computer vision* dan *image processing* untuk pendeteksian objek. Metode HOG diperkenalkan pertama kali oleh Dalal dan Triggs pada tahun 2005 [1]. Metode HOG dalam penelitian ini akan digunakan untuk menguji apakah dalam citra uji terdapat manusia atau tidak. Gambar 2.23 menunjukkan langkah kerja proses ekstraksi fitur dari metode HOG.



Gambar 2.26 Proses Ekstraksi Fitur Metode HOG

Pertama, proses *normalize gamma and colour* atau normalisasi gamma dan warna pada citra. Tahap ini adalah tahap opsional yang berfungsi untuk mengurangi pengaruh efek pencahayaan pada citra.

Proses kedua adalah *compute gradients* atau perhitungan gradien pada frame citra. Pada tahap kedua ini, citra pertama kali dikonversi kedalam bentuk citra skala keabuan. Kemudian perhitungan gradien dilakukan dengan menerapkan metode sobel pada citra secara vertikal dan horisontal dengan filter kernel:

$$D_2(x)\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{dan} \quad D_2(y)\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad \text{..... (2.15)}$$

Dengan melambangkan *image* sebagai *I*, kita akan menghitung nilai *x* dan *y* menggunakan operasi konvolusi :

$$I_x = I * D_x \qquad \text{dan} \qquad I_y = I * D_y \qquad \text{..... (2.16)}$$

Nilai *magnitude* dan orintasi gradien dapat dihitung setelah mendapatkan nilai *x, y derivatives* (I_x dan I_y) dengan persamaan:

$$|G| = \sqrt{I_x^2 + I_y^2} \qquad \text{dan} \qquad \theta = \arctan\frac{I_y}{I_x} \qquad \text{..... (2.17)}$$

Langkah berikutnya yaitu *weighted vote into spatial and orientation cells.* Dalam tahap ini, gambar dibagi dalam ruang-ruang kecil yang disebut *cell. Cell* adalah sebuah ruang yang berukuran 4x4 pixel. Setiap pixel dalam *cell* dikelompokkan dalam *bin* berdasarkan nilai orientasi yang didapat dari perhitungan gradien. Tiap-tiap *cell* tersebut akan dibuat sebuah histogram dengan 9 channel histogram yang didistribusikan kedalam orientasi sudut 0° - 180°.

Selanjutnya tahap normalisasi *block*. Pada saat proses *compute gradient* di peroleh nilai gradien yang berbeda-beda. Sehingga perlu pengelompokkan tiap *cells*. *Cells* perlu dikelompokkan lagi menjadi kelompok yang lebih besar yang disebut *block*. Performa terbaik yang dihasilkan dalam deteksi manusia adalah dengan menggunakan 3x3 *cell* per *block* [1]. Sedangkan setelah pengelompokkan menjadi *block*, *block* ini biasanya terjadi *overlap* atau tumpang tindih sehingga harus dilakukan normalisasi. Teknik normalisasi yang sering digunakan antara lain:

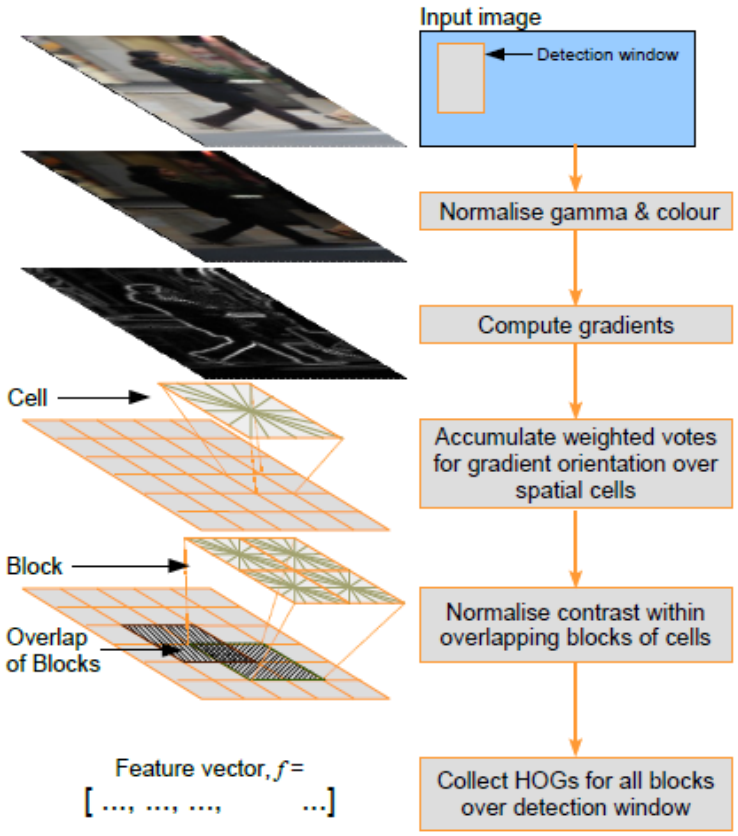
L2–norm: $$v \rightarrow \frac{v}{\sqrt{(\|v\|_2^2 + \varepsilon^2)}} \qquad \text{..... (2.18)}$$

L1-norm: $$v \rightarrow \frac{v}{\sqrt{(\|v\|_1 + \varepsilon)}} \qquad \text{..... (2.19)}$$

L1-sqrt: $$v \rightarrow \frac{v}{(\|v\|_1 + \varepsilon)} \qquad \text{..... (2.20)}$$

Langkah kelima yaitu *collect HOG over detection window*. Semua histogram untuk setiap *block* akan digabungkan untuk menghasilkan sebuah *feature vector*. *Feature vector* ini akan digunakan dalam *Support Vector Machine* untuk proses pendeteksian manusia.

Untuk lebih jelasnya mengenai istilah-istilah seperti *cells*, *blocks*, *overlap block*, dan lain-lain dapat dilihat pada gambar 2.27.



Gambar 2.27 Gambaran Umum Metode HOG

2.6 Support Vector Machine

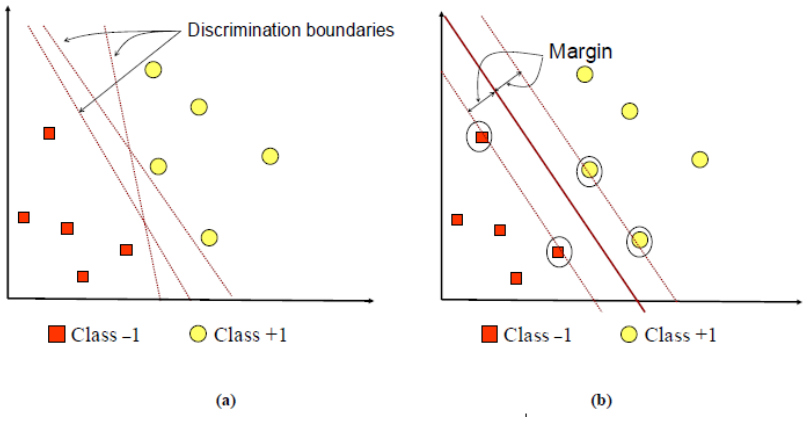
Support Vector Machine (SVM) pertama kali diperkenalkan oleh Vapnik pada tahun 1992 sebagai rangkaian harmonis konsep-konsep unggulan dalam bidang pattern recognition. SVM adalah metode learning machine yang bekerja atas prinsip *Structural Risk Minimization* (SRM) dengan tujuan menemukan *hyperplane* terbaik yang memisahkan dua buah *class* pada *input space* [13].

SVM dalam penelitian ini termasuk dalam rangkaian metode HOG yang merupakan alat untuk menetapkan apakah dalam *detection window* terdapat sosok manusia atau tidak. SVM *classifier* berfungsi untuk memisahkan *class* manusia dan nonmanusia.

Pada gambar 2.25a terdapat beberapa *pattern* yang terbagi dalam dua buah *class*-1 dan *class*+1. *Pattern* yang tergabung dalam *class*-1 dilambangkan dengan kotak warna merah, sedangkan *pattern* yang ada dalam *class*+1 dilambangkan

dengan lingkaran warna kuning. Masalah klasifikasi dapat diterjemahkan dengan usaha mencari garis (*hyperlane*) yang memisahkan antara kedua kelompok tersebut. Beberapa alternatif garis pemisah ditampilkan dalam gambar 2.25a.

Gambar 2.25b menunjukkan *hyperlane* yang memisahkan *class*-1 dengan *class*+1 yang memiliki margin tertentu. Margin adalah jarak antara *hyperlane* dengan *pattern* terdekat dari masing2 *class*. *Pattern* terdekat ini disebut sebagai *support vector*.



Gambar 2.28 SVM Mencari *Hyperlane* Terbaik

Data yang tersedia dinotasikan sebagai $\vec{x}_i \in \Re^d$ sedangkan label masing-masing dinotasikan $y_i \in \{-1,+1\}$ untuk i = 1,2,… ,l, yang mana l adalah banyaknya data. Diasumsikan kedua *class*-1 dan *class*+1 dapat terpisah secara sempurna oleh *hyperplane* berdimensi d , yang didefinisikan

$$\vec{w}.\vec{x}_i + b = 0 \qquad \text{....(2.21)}$$

Maka untuk pattern $\vec{x}_i$ yang termasuk kelas-1 dirumuskan sebagai pattern yang memenuhi pertidaksamaan:

$$\vec{w}.\vec{x}_i + b \leq -1 \qquad \text{....(2.22)}$$

Dari persamaan diatas kita mendapatkan persamaan baru yang *equal* yaitu:

$$y_i(\vec{x}_i \vec{w} + b) \leq 1 \qquad \text{....(2.23)}$$

sedangkan pattern $\vec{x}_i$ yang termasuk kelas+1 dirumuskan sebagai pattern yang memenuhi pertidaksamaan:

$$\vec{w}.\vec{x}_i + b \geq +1 \quad \text{....(2.24)}$$

Dari persamaan diatas kita mendapatkan persamaan baru yang *equal* yaitu:

$$y_i(\vec{x}_i \vec{w} + b) \leq 1 \quad \text{....(2.25)}$$

dari persamaan diatas dapat disimpulkan bahwa nilai dari support vector akan selalu sesuai dengan persamaan:

$$y_i(\vec{x}_i \vec{w} + b) - 1 = 0 \quad \text{....(2.26)}$$

Untuk menemukan margin dari kelas+1 dan kelas-1 dapat dihitung dengan cara menghitung jarak dari x_+ dan x_- sebagai berikut:

$$Width = (x_+ - x_-).\frac{\vec{w}}{\|w\|} \quad \text{....(2.27)}$$

nilai x_+ dan $-x_-$ masing-masing adalah:

$$x_+ = 1 - b \quad \text{....(2.28)}$$

$$-x_- = 1 + b \quad \text{....(2.29)}$$

maka didapat persamaan

$$Width = (x_+ - x_-).\frac{\vec{w}}{\|w\|} = \frac{2}{\|w\|} \quad \text{....(2.30)}$$

Untuk memaksimalkan margin maka dilakukan dengan memaksimalkan nilai *width:*

$$\max \frac{2}{\|w\|} \quad \text{....(2.31)}$$

Karena nilai $\max \frac{2}{\|w\|}$ adalah untuk jarak antar *support vector*, maka nilai dari *hyperlane* adalah $\max \frac{1}{\|w\|}$ atau dapat dengan meminimalkan nilai $\|w\|$. Hal ini dapat dirumuskan dengan *Qudratic Programming Problem* dengan mencari:

$$\min \frac{1}{2}\|w\|^2 \quad \text{....(2.32)}$$

Problem ini dapat diselesaikan dengan *Langrange Multiplier*:

$$L = \frac{1}{2}\| w \|^2 - \sum_{i=1}^{i} \alpha_i (\mathrm{y}_i((\vec{\mathrm{x}}_i . \vec{\mathrm{w}} + \mathrm{b}) - 1)) \quad \text{....(2.33)}$$

α_i adalah *Langrange Multiplier*, yang bernilai nol atau positif ($\alpha_i \geq 0$). Nilai optimal dari persamaan dapat dihitung dengan meminimalkan L terhadap $\vec{w}$ dan b, dan memaksimalkan L terhadap α_i. Dengan memperhatikan sifat bahwa titik optimal gradien L=0, persamaan dapat dimodifikasi sebagai maksimalisasi problem yang hanya mengandung α_i saja, sebagaimana persamaan dibawah:

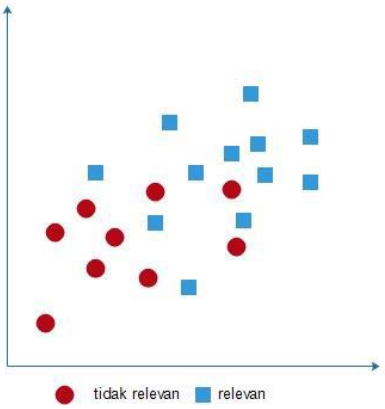
$$L = \sum_{i=1}^{i} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \vec{x}_i \vec{x}_j \quad \text{....(2.34)}$$

Subject to:

$$\alpha_i \geq 0 (i = 1, 2, \cdots, l) \qquad \sum_{i=1}^{l} \alpha_i y_i = 0 \quad \text{....(2.35)}$$

Dari hasil perhitungan ini diperoleh α_i yang kebanyakan bernilai positif. Data yang berkorelasi dengan α_i yang positif inilah yang disebut *support vector*.

Dot product $\vec{x}_i \vec{x}_j$ dari persamaan 2.34 sering diganti dengan simbol K. K adalah matrik kernel. Matrik kernel ini digunakan untuk membuat data set yang bersifat *non-linear* menjadi *linear*. Contoh dataset yang bersifat *non linear* dapat dilihat pada Gambar 2.29.



Gambar 2.29 Dataset Nonlinear

Oleh karena itu dalam SVM terdapat fungsi kernel yang dapat mengubah data set yang tidak linear menjadi linier dalam space baru.

Tabel 2.4 Kernel Trik pada SVM

Kernel Type	**Definisi**
Linear	$K(x_i, x_j) = x^T x$
Polynomial	$K(x_i, x_j) = (x^T x_i + 1)^p$
Gaussian	$K(x_i, x_j) = \exp(-\frac{\| x_i - x_j \|^2}{2\sigma^2})$

Pemilihan fungsi kernel yang tepat adalah hal yang sangat penting, karena fungsi kernel ini akan menentukan *feature space* dimana *classifier* akan dicari.

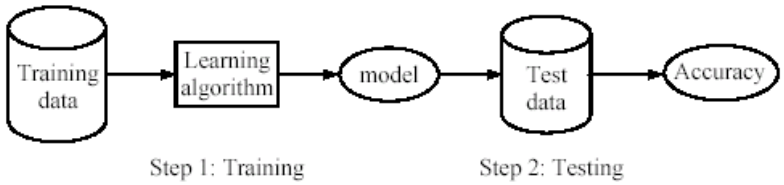
2.7 Pengukuran Akurasi Klasifikasi

Klasifikasi adalah proses pembelajaran secara terbimbing. Untuk melakukan klasifikasi, dibutuhkan training set sebagai data pembelajaran. Setiap sampel dari training set memiliki atribut dan klas label. Contoh klas label:

- Usulan kredit : diterima atau ditolak
- Kelulusan : pujian, sangat memuaskan, memuaskan

Dalam klasifikasi terdapat dua tahap yaitu:

- Learning (training) : Pembelajaran menggunakan data training
- Testing: Menguji metode atau model menggunakan data testing



Gambar 2.30 Alur Proses Klasifikasi

Dalam proses klasifikasi, terdapat suatu Confusion Matriks sebagai berikut:

Tabel 2.5 Confussion Matriks

	Classified Positive	***Classified Negative***
Actual Positive	*True Positive*	*False Negative*
Actual Negative	*False Positive*	*True Negative*

True Positive adalah citra manusia yang diklasifikan sebagai manusia, False Positive adalah citra non-manusia yang diklasifikasikan sebagai manusia, False Negative adalah citra non-manusia yang diklasifikasikan sebagai non-manusia, dan True Negative adalah citra manusia yang diklasifikasikan sebagai non-manusia. Setiap kolom dari confusion matrix merupakan contoh di kelas yang telah diprediksi, sedangkan setiap baris mewakili contoh di kelas yang sebenarnya. Setelah didapat *true positive*, *false positive*, *true negative* dan *false negative*, selanjutnya hitung untuk menghitung nilai akurasinya. Berikut persamaan untuk menghitung akurasi:

$$Akurasi = \frac{\text{TP+FN}}{\text{TP+TN+FP+ FN}}\ x\ 100\% \qquad \ldots(2.36)$$

2.8 OpenCV

2.8.1 Sekilas OpenCV

OpenCV (*Open Computer Vision*) adalah *library* pemrograman untuk *computer vision* yang bersifat *open source*. *Library* ini dapat diunduh secara gratis dari web OpenCV [14]. *Library* ini ditulis dalam bahasa C dan C++ dan dapat berjalan di *Linux, Windows* dan *Mac OS X*. Library OpenCV dapat digunakan dalam berbagai bahasa pemrograman, diantaranya *python,* C++, dan *Matlab.*

OpenCV dirancang untuk efisiensi komputasional dan dengan fokus yang kuat pada aplikasi *real-time* pada bidang pengolahan citra. Tujuannya adalah agar komputer mempunyai kemampuan yang mirip dengan cara pengolahan visual pada manusia.

OpenCV dapat digunakan untuk membuat program komputer visi sehingga program komputer mampu mendeteksi wajah dari manusia, lalu diproses oleh komputer untuk kemudian melakukan aksi seperti mengenali wajah atau mengikuti wajah manusia tersebut. OpenCV ini adalah program utama antara kamera dengan komputer dalam bidang *computer vision.* OpenCV sangat disarankan untuk *programmer* yang akan berkutat pada bidang *computer vision,* karena *library* ini mampu menciptakan aplikasi yang handal, kuat dibidang *digital vision,* dan

mempunyai kemampuan yang mirip dengan cara pengolahan penglihatan pada manusia.

Karena sifatnya yang *open source*, maka terdapat komunitas pengguna dalam jumlah yang sangat besar yaitu lebih dari 47 ribu orang dari komunitas pengguna termasuk di dalamnya orang-orang dari perusahaan besar serta pusat-pusat penelitian. Sejak peluncuran pertamanya pada Januari 1999, OpenCV telah digunakan pada banyak aplikasi, produk, dan usaha-usaha penelitian. Aplikasi-aplikasi ini meliputi penggabungan citra pada peta web dan satelit, *image scan alignment*, pengurangan *noise* pada citra medis, sistem keamanan dan pendeteksian gangguan, sistem pengawasan otomatis dan keamanan, sistem inspeksi pabrik, kalibrasi kamera, aplikasi militer, kendaraan udara tak berawak, kendaraan darat, serta kendaraan bawah air. OpenCV juga telah digunakan untuk pengenalan suara dan musik, dimana teknik pengenalan visi diaplikasikan pada citra spektogram suara. Bidang penerapan OpenCV antara lain:

1. 2D dan 3D fitur *toolkit*
2. Estimasi *Egomotion*
3. Sistem pengenalan wajah
4. Pengenalan gerakan
5. Interaksi manusia-komputer (HCI)
6. *Mobile* robotika
7. Pemahaman gerak
8. Identifikasi objek
9. Segmentasi dan pengenalan
10. Persepsi kedalaman dari 2 kamera: visi stereopsis stereo
11. *Structure from motion* (SFM)
12. Pelacakan gerak
13. *Augmented reality*

Library OpenCV menyediakan banyak algoritma dasar dalam bidang *computer vision*, dengan keuntungan bahwa fungsi-fungsi tersebut telah diuji dengan baik dan digunakan oleh para peneliti di seluruh dunia. *Libraries* OpenCV juga telah menyediakan modul pendeteksian manusia dengan metode HOG.

2.8.2 Fitur OpenCV

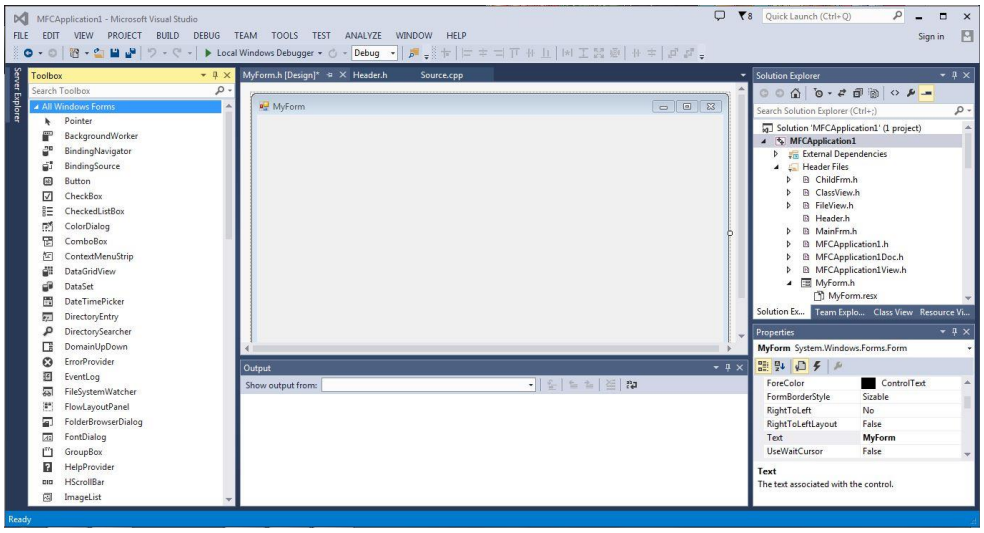
Library pada OpenCV berisi lebih dari 500 fungsi yang menjangkau berbagai area dalam permasalahan visi, meliputi inspeksi produk pabrik, pencitraan medis, keamanan, antarmuka pengguna, kalibrasi kamera, visi stereo, dan robotika. Berbagai macam fitur disediakan dalam library OpenCV, antara lain:

1. Manipulasi data gambar (mengalokasi memori, melepaskan memori, menduplikasi gambar, mengatur serta mengkonversi gambar)
2. *Image/Video Input/Output* (bisa menggunakan kamera yang sudah didukung oleh *library* ini)
3. Manipulasi matriks dan vektor, serta terdapat juga *routines* aljabar linear (*products*, *solvers*, *eigenvalues*, SVD)
4. *Image processing* dasar (penapisan/*filtering*, pendeteksian tepi, *sampling* dan interpolasi, konversi warna, operasi morfologi, histogram, *image pyramids/*piramida citra)
5. Analisis struktural
6. Pendeteksian gerak
7. Pengenalan objek
8. *Basic* GUI (menampilkan gambar/video, mengontrol *mouse*/*keyboard*, *scrollbar*)
9. Image labelling (*line* / garis, *conis* / kerucut, poligon, *test drawing*)

2.9 Microsoft Visual Studio 2013

Microsoft Visual Studio [15] merupakan sebuah perangkat lunak lengkap (suite) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi console, aplikasi Windows, ataupun aplikasi web. *Visual Studio* mencakup kompiler, SDK, *Integrated Development Environment* (IDE), dan dokumentasi (umumnya berupa MSDN Library). Kompilator yang dimasukkan ke dalam paket *Visual Studio* antara lain *Visual* C++, *Visual* C#, Visual *Basic*, *Visual Basic* .NET, *Visual InterDev*, *Visual* J++, *Visual* J#, *Visual FoxPro*, dan *Visual SourceSafe*.

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed code* (dalam bentuk *Microsoft Intermediate Language* di atas .NET *Framework*). Selain itu, *Visual Studio* juga dapat digunakan untuk mengembangkan aplikasi *Silverlight*, aplikasi *Windows Mobile* (yang berjalan di atas .NET *Compact Framework*).



Gambar 2.31 Tampilan Microsoft Visual Studio 2013

2.10 UML

Pada perkembangan teknologi perangkat lunak, diperlukan adanya bahasa yang digunakan untuk memodelkan perangkat lunak yang akan dibuat dan perlu adanya standarisasi agar orang di berbagai negara dapat mengerti pemodelan perangkat lunak. Seperti yang kita ketahui bahwa menyatukan banyak kepala untuk menceritakan sebuah ide dengan tujuan untuk memahami, merupakan hal yang tidak mudah, oleh karena itu diperlukan sebuah bahasa pemodelan perangkat lunak yang dapat dimengerti oleh banyak orang.

Banyak orang yang telah membuat bahasa pemodelan pembangunan perangkat lunak sesuai dengan teknologi pemrograman yang berkembang pada saat itu, misalnya yang sempat berkembang dan digunakan banyak pihak adalah *Data Flow Diagram* (DFD) untuk memodelkan perangkat lunak yang menggunakan pemrograman prosedural atau struktural, kemudian juga ada *State Transition Diagram* (STD) yang digunakan untuk memodelkan sistem *real time* (waktu nyata).

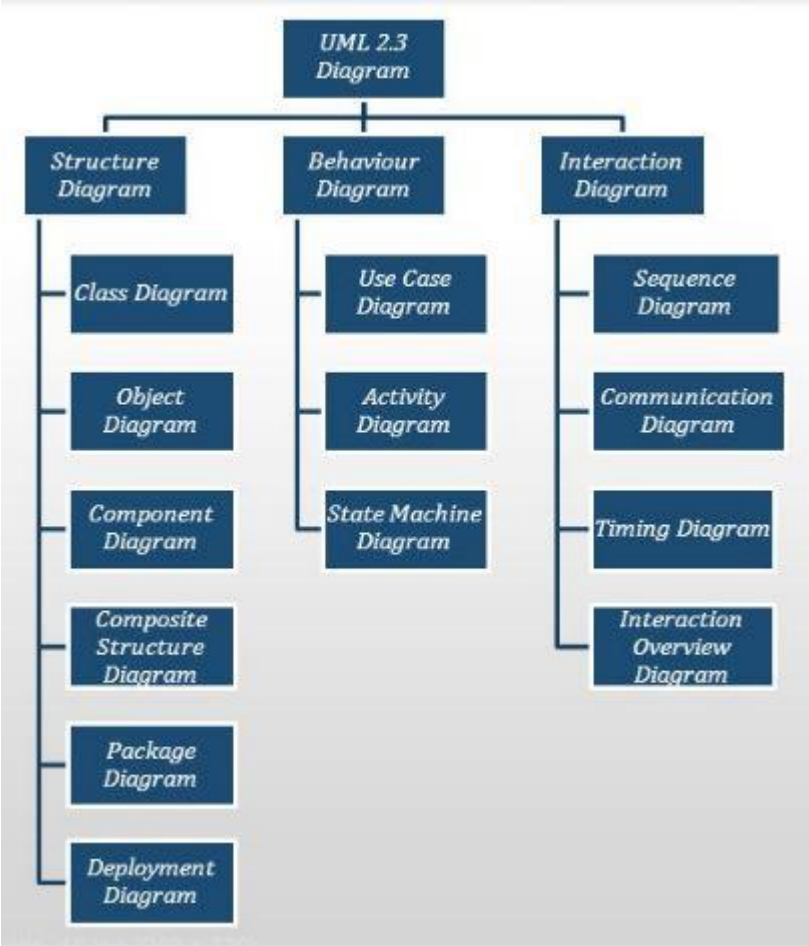
Pada perkembangan teknik pemrograman berorientasi objek, munculah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung [16].

UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek.

Seperti yang kita ketahui di dunia sistem informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan dan konteksnya. Begitu juga dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunaannya. Jadi, belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin digambarkan.

2.10.1 Diagram UML

Pada UML 2.3 terdiri dari 13 macam diagram yang dikelompokkan dalam kategori-kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada gambar ini :



Gambar 2.32 Diagram UML 2.3

Berikut ini penjelasan singkat dari pembagian kategori tersebut :

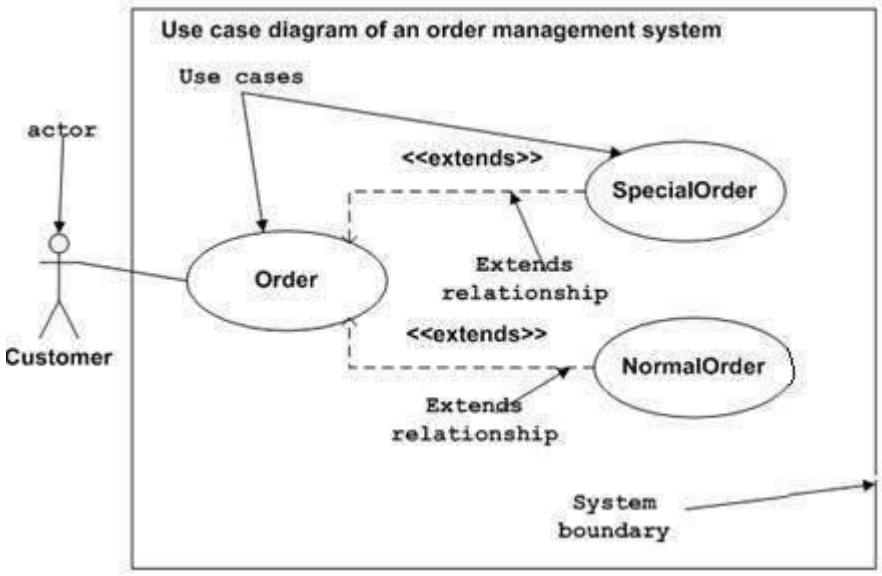
1. *Structure diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
2. *Behavior diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.
3. *Interaction diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antarsubsistem pada suatu sistem.

2.10.1.1 *Use Case* Diagram

Use case atau diagram use case merupakan pemodelan untuk kelakukan (*behaviour*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah

interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi itu [16].

Use case diagram dapat digunakan untuk menggambarkan kebutuhan sistem. Kebutuhan sistem adalah fungsionalitas apa saja yang harus ada dalam sistem (*use case*) serta aktor yang berkaitan dengan kerja sistem. Dengan kata lain *use case* dapat memberitahukan "apa" yang bisa dilakukan sistem, dan bukan "bagaimana". Gambar 2.33 berikut ini adalah contoh diagram *use case*:



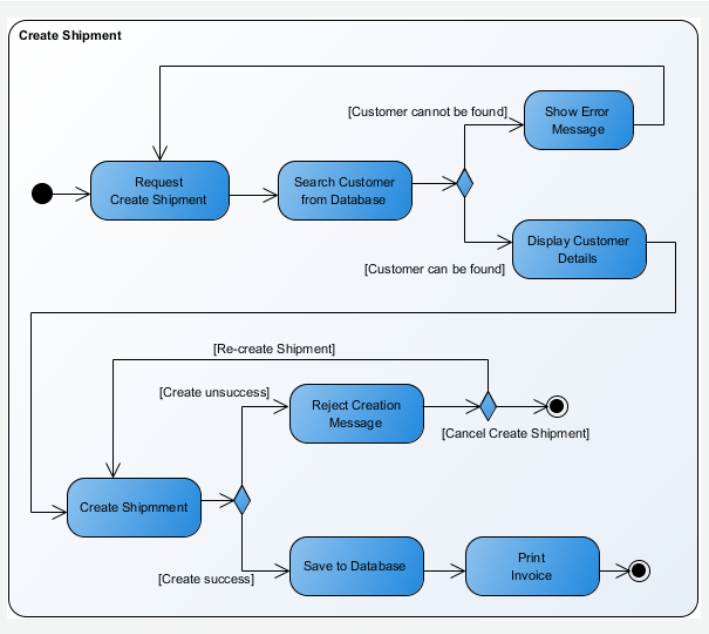
Gambar 2.33 Contoh *Use Case Diagram*

2.10.1.2 *Activity* Diagram

Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem. Contoh diagram aktivitas ada pada gambar 2.34 [17].

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut ini:

1. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan atau pengelompokan tampilan dari sistem/user interface dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.



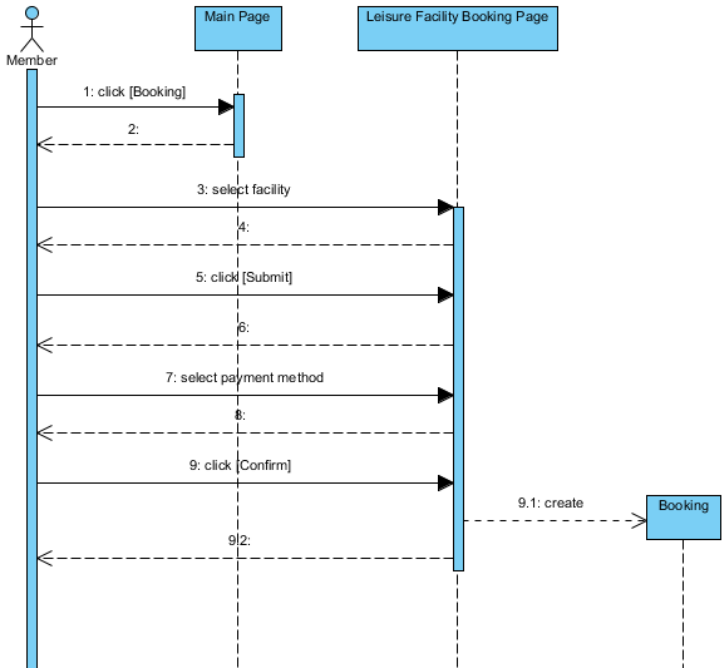
Gambar 2.34 Contoh *Activity Diagram*

2.10.1.3 *Sequence* Diagram

Diagram *sequence* menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah use *case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.

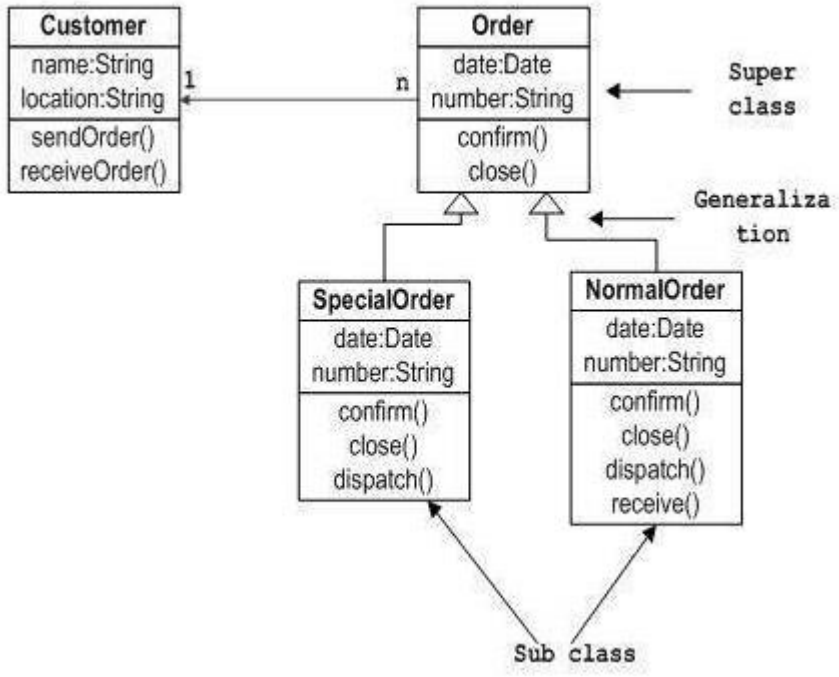
Penomoran pesan berdasarkan urutan interaksi pesan. Penggambaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah pesan yang berjalan terlebih dahulu. Gambar 2.35 adalah contoh diagram *sequence* [17].



Gambar 2.35 Contoh *Sequence Diagram*

2.10.1.4 *Class* Diagram

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas. Gambar 2.36 adalah contoh diagram kelas.



Gambar 2.36 Contoh *Class Diagram*

Kelas-kelas yang ada pada struktur sistem harus dapat melakukan fungsi-fungsi sesuai dengan kebutuhan sistem. Susunan struktur kelas yang baik pada diagram kelas sebaiknya memiliki jenis-jenis kelas berikut :

1) Kelas Main

   Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.

2) Kelas yang menangani tampilan sistem

   Kelas yang mendefinisikan dan mengatur tampilan ke pemakai.

3) Kelas yang diambil dari pendefinisian *use case*

   Kelas yang menangani fungsi-fungsi yang harus ada diambil dari pendefinisian *use case*.

4) Kelas yang diambil dari pendefinisian data

   Kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data.

Jenis-jenis kelas diatas juga dapat digabungkan satu sama lain sesuai dengan pertimbangan yang dianggap baik asalkan fungsi-fungsi yang sebaiknya ada pada

struktur kelas tetap ada. Susunan kelas juga dapat ditambahkan kelas utilitas seperti koneksi ke basis data, membaca *file* teks, dan lain sebagainya sesuai kebutuhan.

Dalam mendefinisikan metode yang ada di dalam kelas perlu memperhatikan apa yang disebut dengan *cohesion* dan *coupling*. *Cohesion* adalah ukuran seberapa dekat keterkaitan instruksi di dalam sebuah metode terkait satu sama lain sedangkan *coupling* adalah ukuran seberapa dekat keterkaitan instruksi antara metode yang satu dengan metode yang lain dalam sebuah kelas. Sebagai aturan secara umum maka sebuah metode yang dibuat harus memiliki kadar *cohesion* yang kuat dan kadar *coupling* yang lemah.