BAB 2
LANDASAN TEORI

2.1 Video Digital

Pada dasarnya video didefinisikan sebagai suatu rangkaian citra digital yang tersusun berdasarkan waktu, Video memiliki 3 dimensi karena dua dimensi yang pertama adalah baris dan kolom pada setiap citranya, sedangkan dimensi ketiga adalah waktu urutan *framenya*. Satu buah citra pada video dikenal sebagai *frame* [10].

Beberapa karakteristik yang dapat mempengaruhi kualitas dari suatu video antara lain resolusi dimensi, kuantisasi atau kedalaman bit, dan *frame rate*.

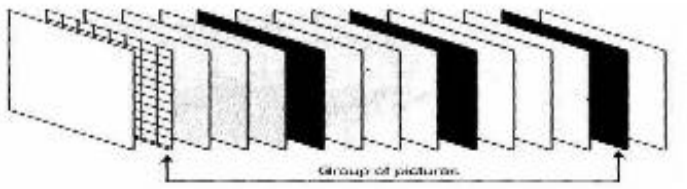
2.1.1 *Frame Rate*

Frame rate menyatakan banyaknya frame yang bergerak tiap detik. Semakin banyak *frame* yang bergerak tiap detiknya maka semakin bagus kualitas video. *Frame rate* dinyatakan dalam *frame per second* (fps). Ada dua karakteristik yang berkaitan dengan *frame rate* , yaitu kehalusan gerakan (*smooth motion*) dan kilatan (*flash*). Untuk mendapatkan gerakan yang halus , video digial setidaknya harus menampilkan *frame* sekitar 25 *frame per second*. Apabila *frame rate* kurang dari 20 *frame per second* biasanya akan memunculkan kilatan *flash*.

2.1.2 Ubah Video Kedalam Deretan Citra

Dalam sebuah video didapatkan frame per second (fps) yang mana dalam satu detik video tersebut terdapat beberapa frame. Misalkan pada sebuah video dengan frame rate per second = 15 dan durasi video tersebut adalah 10 detik, maka didapatkan total frame pada sebuah video tersebut adalah 150. Jadi terdapat 150 deretan citra pada video yang berdurasi 10 detik dengan frame per second 15.

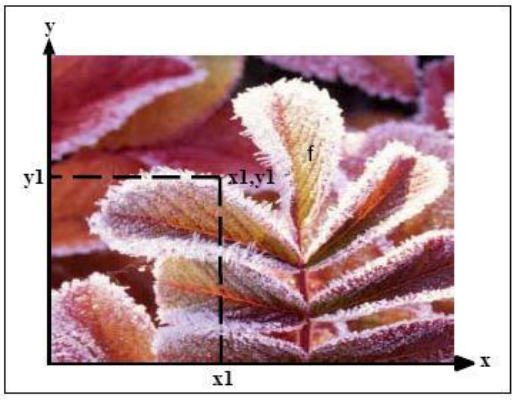
Berikut pada gambar Gambar 2.1 Rangkaian Pada Video sehingga didapatkan sebuah citra.



Gambar 2.1 Rangkaian Pada Video

2.2 Citra Digital

Citra digital adalah citra f(x,y) yang telah didigitalisasi baik kordinat area maupun *brightness level*. Persilangan antara baris dan kolom tertentu disebut dengan piksel [11]. Contohnya adalah gambar/ titik diskrit pada baris x1 dan kolom y1. Hal tersebut dapat diilustrasikan pada Gambar 2.2 **Gambar Citra (x,y)** dibawah ini.



Gambar 2.2 Gambar Citra (x,y)

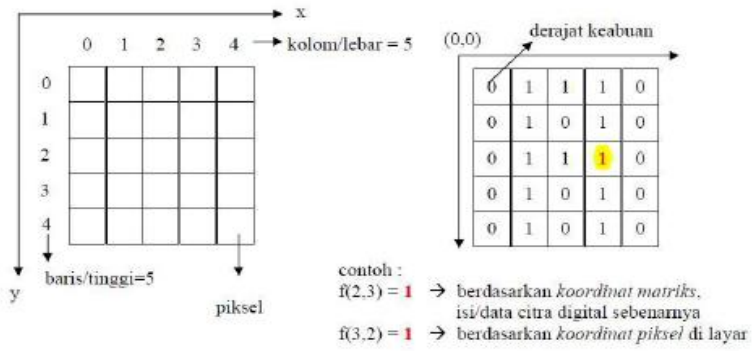
Masing-masing titik pada matriks f(x,y) memiliki nilai-nilai tertentu. Nilai-nilai inilah yang menunjukan besar intensitas cahaya yang mnyusun citra digital tersebut. Nilai intensitas digital tersebut dinamakan *picture elements* (*pixels*). Kumpulan piksel ini menentukan dimensi atau resolusi dari sebuah citra. Citra digital merupakan suatu matriks dimana indeks baris dan kolomnya menyatakan suatu titik pada citra tersebut dan elemen matriksnya menyatakan tingkat 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 2.3 dibawah ini.

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 2.3 Gambaran Citra Digital

2.2.1 Tipe Citra Digital

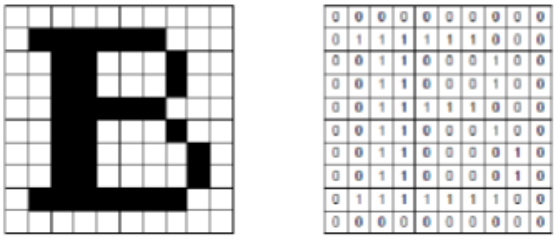
Berdasarkan format penyimapanan nilai warnanya, citra dibagi dalam tiga tipe yaitu :

1. Citra Biner

Pada citra digital dengan tipe biner, setiap piksel pada citra hanya memiliki dua nilai saja yaitu 0 dan 1. Nilai 0 mewakili warna hitam dan nilai 1 mewakili warna putih. Karena hanya memiliki 2 nilai yang mungkin untuk setiap piksel, maka setiap piksel hanya memiliki ukuran 1 bit saja. Citra dengan tipe biner seperti ini akan sangat efisien dalam proses penyimpanannya. Dinyatakan dalam suatu fungsi:

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

Berikut ini adalah contoh citra biner dan representasi nilai tiap pixel seperti pada Gambar 2.4 [12].



Gambar 2.4 Gambaran Citra Biner

2. Citra Grayscale

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 :

$$f(x,y)\ \Sigma\ [0\ldots255] \qquad \ldots\ldots (2.3)$$

Gambar 2.5 menunjukkan contoh citra skala keabuan 8-bit dengan ukuran 512 x 512 *pixel* [12].



Gambar 2.5 Citra Kapal

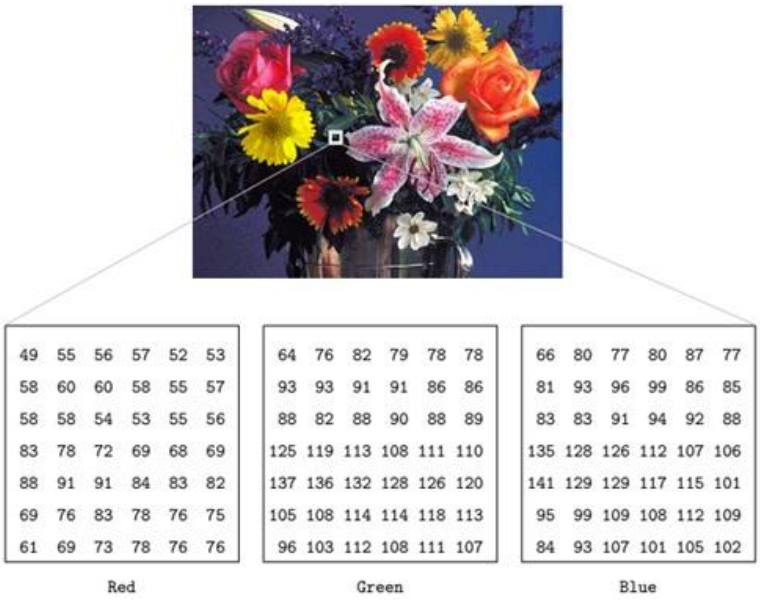
3. Citra Berwarna atau *True Color*

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 2.6 menunjukkan contoh citra berwarna 8-bit. Representasi dalam citra digital dinyatakan dalam persamaan :

$$fR(x,y)\ \Sigma\ [0\ldots255]$$
$$fG(x,y)\ \Sigma\ [0\ldots255] \qquad \ldots\ldots (2.4)$$
$$fB(x,y)\ \Sigma\ [0\ldots255]$$



Gambar 2.6 Representasi Citra RGB

2.2.2 Metode Konversi Nilai Citra RGB ke *Graycale*

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.

Berdasarkan catatan dokumentasi GIMP [13], 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 :

$$Grayscale = (\max(R,G,B) + \min(R,G,B)) * 0.5 \quad \text{..... (2.5)}$$

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 :

$$Grayscale = (R + G + B) / 3 \quad \text{..... (2.6)}$$

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 :

$$Grayscale = (0.21 * R) + (0.72 * G) + (0.07 * B) \quad \text{..... (2.7)}$$

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* :

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 [14] dan BT.601 [15] 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)

2.2.3 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 [16].

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

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

Keterangan :

ni = 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):

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

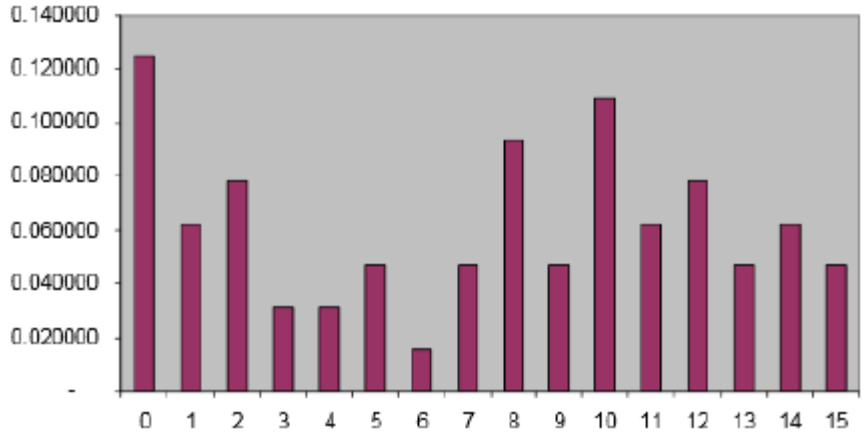
Gambar 2.7 Matriks Citra Digital

Perhitungan histogramnya dapat dilihat pada Tabel 2.1. Dapat dilihat bahwa semakin besar nilai *ni* maka semakin besar pula nilai *hi* [16].

Tabel 2.1 Perhitungan Histogram

i	***ni***	***hi* = *ni* / *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

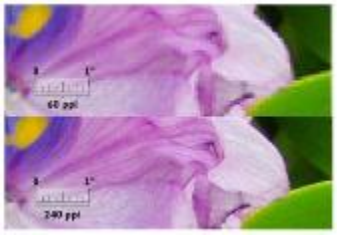
Histogram citra digital dari matriks citra digital diatas dapat dilihat pada Gambar 2.8 berikut ini:



Gambar 2.8 Histogram Citra Digital.

2.2.4 Resolusi

Resolusi dinyatakan dalam satuan piksel x piksel, yang menyatakan ukuran panjang kali lebar dari suatu frame video. Semakin tinggi resolusinya, semakin besar bit yang digunakan untuk menyimpan dan mentransmisikan datanya, dan semakin bagus kualitasnya. Misalnya pada format AVI (Audio Video Intereave) resolusi adalah 320 piksel x 240 piksel [11]. Berikut pada Gambar 2.9 merupakan resolusi citra.



Gambar 2.9 Resolusi Pada Citra

2.3 *Computer Vision*

Pada dasarnya *computer vision* mencoba meniru cara kerja sistem penglihatan manusia. Dalam proses penglihatan manusia, manusia melihat objek dengan menggunakan indera penglihatan yang berupa mata, lalu citra objek dilanjutkan ke otak untuk diinterpretasikan sehingga manusia mengerti objek apa yang tampak. Hasil interpretasi ini kemudian digunakan untuk pengambilan keputusan [12].

Tabel 2.2 Garis Besar *Human Vision* dan *Computer Vision*

Human Vision	Computer Vision
Menggunakan indera pengihatan 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 tersebut 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

2.3.1 *Human Detection* (Deteksi Manusia)

Arti dari pendeteksian yaitu 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).

Pada deteksi manusia, penelitian yang dilakukan oleh Dalal dan Triggs [16], 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.

Human detection berfungsi untuk mendeteksi objek manusia pada sebuah citra. Pada 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 2.3 berikut ini:

Tabel 2.3 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

Pada Gambar 2.10 adalah contoh pendeteksisan manusia yang dilakukan oleh Dalal dan Trigss [16].



Gambar 2.10 Citra Digital

2.4 Deteksi Tepi Sobel

Pada 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 [12].

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:

-1	0	1
-2	0	2
-1	0	1

x Filter

1	2	1
0	0	0
-1	-2	-1

y Filter

$Ix = I * S(x)$ dan $Ix = I * S(x)$ .....(2.10)

Ket : I = Citra yang akan dideteksi tepi.

S = Filter sobel

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

$$|G| = \sqrt{Gx^2 + Gy^2} \quad \text{.....(2.11)}$$

Sedangkan nilai arah gradien dapat dihitung dengan fungsi:

$$\theta = \arctan \frac{Gx}{Gy} \quad \text{.....(2.12)}$$

2.5 *Local Binary Pattern*

LBP adalah salah satu deskriptor tekstur terbaik dan telah banyak digunakan dalam berbagai aplikasi. LBP telah terbukti sangat diskriminatif dan 15 keuntungan utamanya, yaitu variasi untuk perubahan tingkat abu-abu monoton dan efisiensi komputasi, membuatnya cocok untuk tugas gambar menuntut analisis. *Local Binary Pattern* (LBP) didefinisikan sebagai ukuran tekstur *gray-scale* invarian, berasal dari definisi umum tekstur di daerah sekitar. Operator LBP dapat dilihat sebagai pendekatan kesatuan dengan model statistik dan struktur tradisional berbeda dari analisis tekstur. Secara sederhana, LBP adalah sebuah kode biner yang menggambarkan pola tekstur lokal. Hal ini dibangun dengan lingkungan batas dengan nilai abu-abu dari pusatnya [17]. Selain itu LBP juga memiliki komputasi yang cukup efisien. Adapun rumus dari LBP adalah sebagai berikut :

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(I_{p,R} - I_c)2^{P-1-p} \qquad s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \quad \text{....(2.13)}$$

Keterangan rumus :

P = jumlah banyaknya tetangga

R = radius antara titik pusat dan titik tetangga

LBPP,R = nilai desimal hasil konversi nilai biner

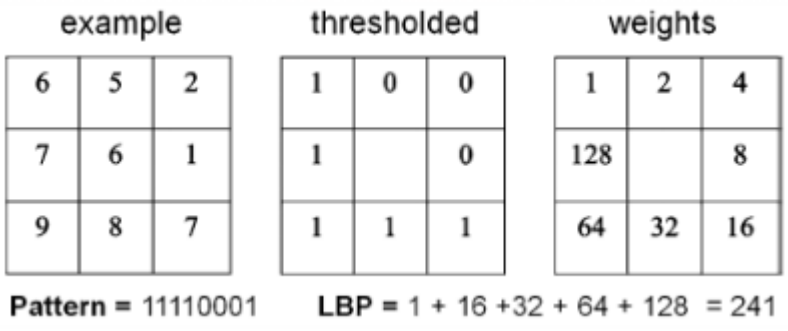
Ic = nilai intensitas piksel pusat

Rumus perbandingan piksel fungsi s(x) didefinisikan sebagai berikut:

s(x) = fungsi *thresholding*.

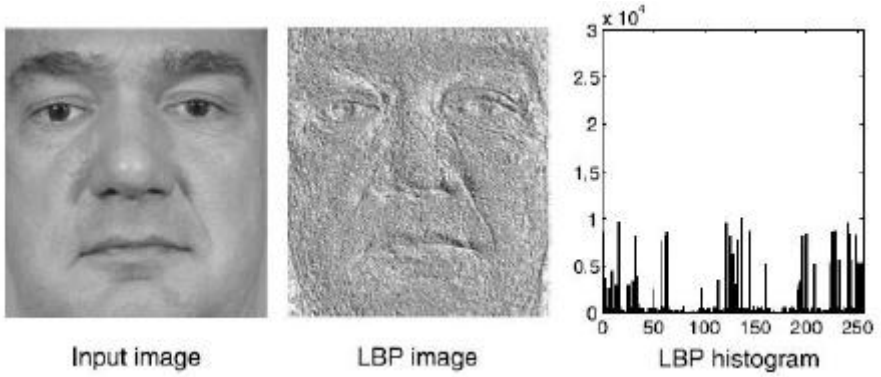
LBP menggunakan operator yang bekerja pada blok gambar dengan ukuran piksel 3x3 (biasa disebut kernel). Piksel-piksel yang terdapat pada blok tersebut lalu di threshold dengan nilai piksel tengah , dikalikan dengan bilangan pangkat

dua, dan lalu dijumlahkan untuk mendapatkan nilai label untuk menggantikan nilai piksel tengah sebelumnya [6].



Gambar 2.11 Operator LBP

Setelah semua *pixel* selesai di LBP, selanjutnya gambar hasil tersebut dibuat ke dalam bentuk histogram. Histogram tersebut yang akan digunakan sebagai vektor ciri untuk mendeskripsikan tekstur dari gambar tersebut [6].



Gambar 2.12 Input Gambar, Hasil LBP, LBP *Histogram*

Pada histogram lbp terdapat 256 bin yang nantinya diambil *feature vectore* dari histogram tersebut. *Feature vector* tersebut didapatkan dengan membagi setiap bin dengan jumlah semua *vector* yang melakukan *voting* terhadap setiap bin tersebut.

2.6 *Histograms of Oriented Gradients*

Maksud 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 [16]. Adapun tahapan tahapan dari metode HOG adalah sebagai berikut : [18]

1. Mengkonversikan sebuah *image* menjadi bentuk *grayscale.*
2. Menghitung *edge degree* dan *strength per pixel*.
3. Memecah *image* ke dalam *cell areas*
4. Membuat *histogram* dari *edge degree* dan *strength per pixel* yang telah dilakukan pada tahapan nomor 2.
5. Melakukan proses *normalize the histogram* dengan *neighbour cells.* Rumus perhitungan dari normalisasi histogram adalah sebagai berikut:

$$b = \frac{b}{\sqrt{b^2+e}} \quad \text{.....(2.14)}$$

$$h = \frac{h}{||h||^2+e} \quad \text{.....(2.15)}$$

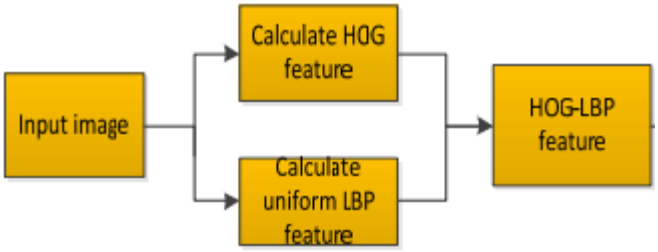
Nilai normalisasi fitur blok didapat dari Fitur blok dinormalisasi untuk mengurangi efek perubahan kecerahan obyek pada satu blok. Variabel b merupakan nilai blok fitur dan variabel e merupakan bilangan positif yang bernilai kecil untuk mencegah pembagian dengan 0. Nilai normalisasi tiap blok digabungkan menjadi satu vektor menjadi fitur vektor HOG. Kemudian fitur vektor HOG dilakukan normalisasi. Normalisasi dilakukan melalui (5). Variabel h merupakan nilai fitur HOG dan variabel e merupakan bilangan positif yang bernilai kecil untuk mencegah pembagian dengan 0.

6. Mengambil nilai feature vector *histogram* dari orientasi cell pada citra.

2.6.1 Gabungan *Feature Vector* HOG – LBP

Kombinasi dua jenis fitur (HOG dan LBP) sebagai fitur set untuk mendeteksi pejalan kaki. Kerangka detektor HOG-LBP ditunjukkan pada Gambar1. Pertama, gambar input dinormalisasi. Ukuran 64 * 128. Kedua, untuk setiap gambar kami menghitung fitur HOG dan fitur LBP berseragam Masing-masing. Ketiga,

vektor fitur HOG-LBP di konkatenasi menjadi single *vector* lalu dilatih oleh algoritma klasifikasi. Lalu HOG LBP Detektor bisa dilakukan untuk pengujian. Pada bagian berikut, kami memberikan gambaran umum tentang keduanya [19].



2.7 *Support Vector Machine*

Support Vector Machine (SVM) pertama kali diperkenalkan oleh Vapnik pada tahun 1992 sebagai rangakaian harmonis konsep-konsep unggulan dalam bidang pattern recognition. Konsep SVM dapat dijelaskan secara sederhana sebagai usaha mencari *Hyperplane* terbaik yang berfungsi sebagai pemisah dua buah *class* pada input *space* [20].

Dalam linear *Support Vector Machine permisah* merupakan fungsi linear. Data latih dinyatakan oleh (xi, yi) dan xi = { $x1$, $x2$, …,xiq} merupakan atribut (fitur) set untuk data latih kelas ke-i. Untuk yi) € {-1,1} menyatakan label kelas. pendefinisian persamaan suatu *hyperplane* pemisah yang dituliskan dengan:

$$wx_i+b=0 \quad .....(2.16)$$

Data xi yang terbagi ke dalam dua kelas, yang termasuk kelas -1 (sampel negatif) didefinisikan sebagai vektor yang memenuhi pertidaksamaan 2.17 berikut ini.

$$wxi+b<0\ untuk\ yi=1 \quad .....(2.17)$$

Sedangkan yang termasuk kelas +1 (sampel positif) memenuhi pertidaksamaan 2.18 berikut.

$$wxi+b>0\ untuk\ yi=+1 \quad ....(2.18)$$

Dimana:

xi = data input

yi = label yang diberikan

w = nilai dari bidang normal

b = posisi bidang relatif terhadap pusat koordinat

Parameter w dan b adalah parameter yang akan dicari nilainya. Bila label data yi = -1, maka pembatas menjadi persamaan 2.19 berikut ini.

$$wxi+b\leq -1 \quad \text{.....(2.19)}$$

Bila label data yi = +1, maka pembatas menjadi persamaan 2.20 berikut.

$$wxi+b\geq +1 \quad \text{.....(2.20)}$$

Margin terbesar dapat dicari dengan cara memaksimalkan jarak antar bidang pembatas kedua kelas dan titik terdekatnya, yaitu $\frac{2}{W}$. Hal ini dirumuskan sebagai permasalahan *quadratic programming* (QP) problem yaitu mencari titik minimal persamaan (2.21) dengan memperhatikan persamaan (2.22) berikut.

$$\min \tau(w) = \frac{1}{2}||w||^2 \quad \text{.....(2.21)}$$

$$y_i(wx_i + b) - 1 \geq 0, (i = 1, \dots, n) \quad \text{.....(2.22)}$$

Permasalahan ini ini dapat dipecahkan dengan berbagi teknik komputasi. Lebih mudah diselesaikan dengan mengubah persamaan 2.21 ke dalam fungsi *Lagrangian* pada persamaan 2.23, dan menyederhanakannya menjadi persamaan 2.24 berikut.

$$L(w, a, b) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i(\, y_i((w^T x_i + b) - 1)) \quad \text{....(2.23)}$$

$$L(w, a, b) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i\, y_i(w^T x_i + b) + \sum_{i=1}^{n} a_i \quad \text{.....(2.24)}$$

Dimana ai adalah *lagrange multiplier* yang bernilai nol atau positif ($ai \geq 0$). Nilai optimal dari persamaan 2.24 dapat dihitung dengan meminimalkan L terhadap w, b, dan a. Dapat dilihat pada persamaan 2.25 sampai 2.26 berikut ini.

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} a_i\, y_i x_i = 0 \quad \text{.....(2.25)}$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} a_i\, y_i = 0 \quad \text{.....(2.26)}$$

$$\frac{\partial L}{\partial a} = w - \sum_{i=1}^{n} a_i\, y_i (w^T x_i + b) - \sum_{i=1}^{n} a_i = 0 \quad \text{.....(2.27)}$$

Maka masalah *Lagrange* untuk klasifikasi dapat dinyatakan pada persamaan 2.28 berikut.

$$Min\ L(w, a, b) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i\, y_i (w^T x_i + b) + \sum_{i=1}^{n} a_i \quad \text{....(2.28)}$$

Dengan memperhatikan persamaan 2.29 dan 2.30 berikut ini.

$$w - \sum_{i=1}^{n} a_i\, y_i x_i = 0 \quad \text{.....(2.29)}$$

$$\sum_{i=1}^{n} a_i\, y_i = 0 \quad \text{.....(2.30)}$$

Model persamaan 2.28 diatas merupakan model primal *Lagrange*. Sedangkan dengan memaksimalkan L terhadap a_i, persamannya menjadi persamaan 2.31 berikut ini.

$$Max \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1,j=1}^{n} a_i y_i y_j{}^T\, x_i x_j{}^T \quad \text{.....(2.31)}$$

Dengan memperhatikan persamaan 2.32 berikut ini.

$$\sum_{i=1}^{n} a_i y_i = 0, a_i > 0\ (i, j = 1, \dots, n) \quad \text{.....(2.32)}$$

Untuk mencari nilai x_i dan y_i dapat dilakukan ketika sudah didapatkan nilai tiap kata (*term*) dari pembobotan dari masing-masing metode seleksi fitur dan inisialisasi kelas . Hasil dari pembobotan diubah ke dalam bentuk format data svm (x), sedangkan data kelas menjadi label data svm (y).

Untuk mendapatkan nilai ai, langkah pertama adalah mengubah setiap *feature data training* menjadi nilai vektor (*support vector*) = $\begin{bmatrix} x \\ y \end{bmatrix}$. Kemudian nilai

vektor dari setiap *data training* dimasukkan ke persamaan 2.33 kernel *trick phi* (φ) berikut ini.

$$\varphi\begin{bmatrix}x\\y\end{bmatrix} = \begin{cases}\sqrt{x_n^2+y_n^2} > 2, maka & \begin{bmatrix}\sqrt{x_n^2+y_n^2}-x+|x-y|\\ \sqrt{x_n^2+y_n^2}-x+|x-y|\end{bmatrix}\\ \sqrt{x_n^2+y_n^2} < 2, maka & \begin{bmatrix}x\\y\end{bmatrix}\end{cases} \quad .....(2.33)$$

Nilai x didapatkan dari persamaan 2.34 kernel linear untuk x berikut.

$$\sum_{i=1,j=1}^{n} x_i^T x_j \; (i,j = 1,\dots,n) \quad .....(2.34)$$

Nilai y didapatkan dari persamaan 2.34 kernel linear untuk y berikut.

$$\sum_{i=1,j=1}^{n} y_i^T y_j \; (i,j = 1,\dots,n) \quad ....(2.35)$$

Untuk mendapatkan jarak tegak lurus yang optimal dengan mempertimbangkan vektor positif, maka hasil perhitungan dari substitusi nilai x dan nilai y ke persamaan 2.33 diberi nilai bias = 1 . Kemudian cari parameter ai, dengan terlebih dahulu mencari nilai fungsi setiap *data training* menggunakan persamaan 2.35, lalu mencari nilai ai pada persamaan linear menggunakan persamaan 2.36 dengan memperhatikan $i,j=1,\dots,n$ berikut

$$\sum_{i=1,j=1}^{n} a_i {T_i}^T T_j \quad .....(2.36)$$

$$\sum_{i=1,j=1}^{n} a_i {T_i}^T T_j = y_i \quad .....(2.37)$$

Setelah parameter ai didapatkan, kemudian masukkan ke persamaan 2.38 berikut.

$$W = \sum_{i=1}^{n} a_i T_i \quad .....(2.38)$$

Hasil yang didapatkan menggunakan persamaan 2.38, selanjutnya digunakan persamaan 2.39 untuk mendapatkan nilai w dan b yaitu sebagai berikut.

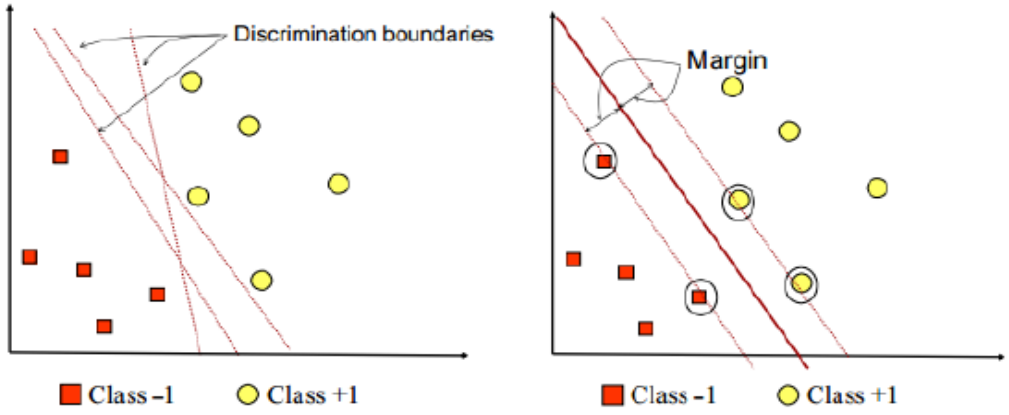
$$y=wx+b \quad .....(2.39)$$

Sedemikian sehingga didapatkanlah nilai w dan nilai b atau nilai *hyperplane* untuk mengklasifikasikan kedua kelas.

Berikut ini fungsi *kernel* yang dapat mengubah data set *non-linear* menjadi *linier* dalam *space* baru.

Tabel 2.4 Kernel Trik SVM

Kernel Type	Value
Linier	K($xixj$)= x^Tx
Polynomial	K($xixj$)=(x^Txi+1)^P
Gaussian	K($xixj$)=exp− $\|\|x1-x2\|\|2/2\sigma$



Gambar 2.13 SVM Mencari* Hyperlane *Terbaik

Gambar pada ruas sebelah kiri menunjukan beberapa *pattern* yang merupakan anggota dari dua buah *class*: +1 dan -1. Pattern yang tergabung pada *class* -1 disimbolkan dengan warna merah (bentuk kotak), sedangkan pattern class -1 disimbolkan dengan warna kuning (bentuk lingkaran). Permasalahan pada klasifikasi objek dapat diterjemahkan dengan cara menemukan garis (*hyperplane*) yang memisahkan antara kedua kelompok tersebut. *Hyperplane* pemisah terbaik diantara kedua *class* didapat dengan cara mengukur margin *hyperplane* tersebut dan mencari titik maksimalnya. Margin adalah jarak antara *hyperplane* tersebut dengan pattern terdekat dari masing-masing *class*. *Pattern* terdekat inilah yang disebut sebagai support vector. Gambar kanan menunjukkan *hyperplane* terbaik yang

terletak pada tengah-tengah kedua *class*, sedangkan titik merah dan kuning yang berada dalam lingkaran hitam merupakan *support vector*. Usaha untuk mencari lokasi *hyperplane* ini yang merupakan inti proses pembelajaran pada SVM.

2.8 Sliding Window

Proses untuk menggeser piksel demi piksel dengan ukuran jendela yang telah ditentukan baik arah horizontal maupun arah vertikal yang selanjutnya tiap jendela tersebut akan dibandingkan dengan nilai klasifikasi. Adapun langkah-langkah nya sebagai berikut [21] :

1. Scan gambar di semua skala dan lokasi.
2. Ekstrak fitur pada setiap jendela.
3. Jalankan klasifikasi SVM pada setiap jendela yang sudah diekstraksi fitur.
4. Eliminasi kotak yang bertumpuk dengan mengambil nilai klasifikasi paling tinggi.

Berikut pada Gambar 2.14 adalah detection window dengan sliding windows.



Gambar 2.14 Detection Window Dengan Sliding Windows

2.9 Non-Maximum Supression

Metode ini yaitu untuk meminimasi kotak kotak setelah dilakukan klasifikasi pada sliding windows. Pada dasarnya setelah dilakukan klasifikasi pada ekstraksi hog akan terdeteksi beberapa kotak yang mengelilingi objek tersebut. Mungkin hal yang tidak mungkin untuk melakukan perhitungan pada satu objek manusia dihitung lebih dari satu orang. Metode ini yaitu menggabungkan semua kotak hasil klasifikasi dengan cara mengeliminasi kotak yang bertumpuk dengan cara menghapus nilai kotak yang rendah, kotak dengan nilai klasifikasi paling tinggi diantara kotak yang bertumpuklah yang diambil [22].

Berikut adalah contoh implementasi Non-Maximum Supression yang dilakukan oleh Sanja Fidler [21]. Pada Gambar 2.15 hasil dari pendeteksian menggunakan sliding windows.



Gambar 2.15 Pendeteksian Dengan Sliding Windows

Setelah itu dilakukan normalisasi untuk menghilangkan kotak kotak yang tumpang tindih pada suatu objek. Pada Gambar 2.16 merupakan hasil dari normalisasi kotak yang saling tumpang tindih.



Gambar 2.16 Hasil Non Maxima Supression

2.10 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:

1. Usulan kredit : terpuji atau tidak terpuji
2. Kelulusan : cukup, baik, sangat baik

Pada klasifikasi terdapat dua tahap yaitu :

1. Learning (training) : Pembelajaran menggunakan data training
2. Testing: Menguji metode atau model menggunakan data testing

Dalam proses klasifikasi, terdapat suatu Confusion Matriks seperti pada tabel 2.5.

Tabel 2.5 Confussion Matriks

	Classfied Positive	**Classified Negatif**
Actual Positif	True Positif	False Negatif
Actual Negatif	False Positif	True Negatif

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:

$$Akurasi = \frac{TP+FN}{TP+TN+FP+FN} \quad .....(2.40)$$

2.11 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak (*software engineering*) merupakan pembangunan dengan menggunakan prinsip atau konsep rekayasa dengan tujuan menghasilkan perangkat lunak yang bernilai ekonomi, dipercaya dan bekerja secara efisien menggunakan mesin. Rekayasa perangkat lunak lebih fokus pada praktik pengembangan perangkat lunak dan mengirimkan perangkat lunak yang bermanfaat bagi pengguna. Rekayasa perangkat lunak fokus pada bagaimana membangun perangkat lunak yang memenuhi kriteria berikut:

1. Dapat terus dipelihara setelah perangkat lunak selesai dibuat seiring berkembangnya teknologi dan lingkungan.
2. Dapat diandalkan dengan proses bisnis yang dijalankan dan perubahan yang terjadi.
3. Efisien dari segi sumber daya dan penggunaan.
4. Kemampuan untuk dipakai sesuai dengan kebutuhan.

Pekerjaan yang terkait dengan rekayasa perangkat lunak dapat dikategorikan menjadi tiga buah kategori umum tanpa melihat area dari aplikasi, ukuran proyek atau kompleksitas perangkat lunak yang akan dibangun, yaitu:

1. Fase Pendefinisian (*Definition Phase*)

Fase pendefinisian fokus pada "*what*" yang artinya harus mencari tahu atau mengidentifikasi informasi apa yang harus diproses, seperti apa fungsi dan performansi yang diinginkan, seperti apa prilaku sistem yang diinginkan, apa kriteria validasi yang dibutuhkan untuk mendefinisikan sistem.

2. Fase Pengembangan (*Development Phase*)

Fase pengembangan fokus pada "*how*" yang artinya selama tahap pengembangan perangkat lunak seorang perekayasa perangkat lunak (*software engineer*) berusaha untuk mendefinisikan bagaimana data diinstrukturkan dan bagaimana fungsi-fungsi yang dibutuhkan diimplementasikan, bagaimana tampilan karakter antarmuka, bagaimana desain ditranslasikan ke dalam bahasa pemrograman dan bagaimana pengujian dijalankan.

3. Fase Pendukung (*Support Phase*)

Fase pendukung fokus pada perubahan yang terasosiasi pada perbaikan kesalahan (*error*), adaptasi yang dibutuhkan pada lingkungan perangkat lunak yang terlibat dan perbaikan yang terjadi akibat perubahan kebutuhan pengguna. Fase pendukung terdiri dari empat tipe perubahan antara lain:

a. Koreksi (*Correction*), yaitu pemeliharaan dengan melakukan perbaikan terhadap kecacatan perangkat lunak.
b. Adaptasi (*Adaptation*), yaitu merupakan tahap untuk memodifikasi perangkat lunak guna mengakomodasi perubahan lingkungan luar dimana perangkat lunak dijalankan.
c. Perbaikan (*Enchancement*), pemeliharaan perfektif atau penyempurnaan melakukan eksekusi atau penambahan pada kebutuhan fungsional sebelumnya.
d. Pencegahan (*Prevention*), pencegahan atau sering disebut juga dengan rekayasa ulang sistem (*software reengineering*) harus dikondisikan untuk mempu melayani kebutuhan yang diinginkan pemakainya.

2.12 UML (Unifield Modeling Language)

Unified Modelling Language (UML) adalah sekumpulan spesifikasi yang dikeluarkan oleh OMG. UML terbaru adalah UML 2.3 yang terdiri dari 4 macam spesifikasi, yaitu : *Diagram Interchange Spesification*, UML *Infrastrukture,* UML *Superstrukture, dan Object Constraint Language* (OCL). Pada UML 2.3 terdiri 13 macam diagram yang dikelompokan pada 3 kategori, yaitu [24].

A. Struktur Diagram, yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan. Pada Struktur Diagram dibagi menjadi 6 bagian [24].

1. Diagam Kelas

Diagram kelas menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut attribut dan metode atau operasi.

2. Diagram Objek

Diagram objek menggambarkan struktur sistem dari segi penamaan objek dan jalannya objek dalam sistem.

3. Diagram Komponen

Diagram komponen dibuat untuk menunjukan organisasi dan ketergantungan diantara kumpulan komponen dalam sebuah sistem.

4. Composite Structure Diagram

Composite Structure Diagram bru muali ada pada UML versi 2.0. diagram ini dapat digunakan untuk menggambarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan struktur pada saat berjalan (*runtime*) .

5. Package Diagram

Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML. Hampir semua diagram dalam UML dapat dikelompokan menggunakan *Package Diagram*.

6. Deployment Diagram

Deployment diagram menunjukan konfigurasi komponen dalam proses eksekusi aplikasi.

B. *Behavior Diagram*, yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem. Pada *Behavior Diagram* dibagi menjadi 3 bagian :

1. Use Case Diagram

Use case diagram merupakan pemodelan untuk kelakuan (behavior) sistem informasi yang akan dibuat,. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat.

2. Activity Diagram

Activity diagram menggambarkan *workflow* atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak.

3. State Machine Diagram

State machine diagram digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem atau objek.

C. *Interactions Diagram*, yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi antar subsistem pada suatu sistem. Pada Interactions Diagram dibagi menjadi 4 bagian :

1. Sequence Diagram

Sequence diagram menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan message yang dikirimkan dan diterima antar objek.

2. Communication Diagram

Communication diagram menggambarkan interaksi antar bojek/bagian dalam bentuk urutan pengiriman pesan. Diagram komunikasi merepresentasikan informasi yang diperoleh dari diagram kelas, diagram *sequence*, dan diagram *use case* untuk mendeskripsikan gabungan antara struktur statis dan tingkah laku dinamis dari suatu sistem.

3. Timing Diagram

Timing diagram merupakan diagram yang fokus pada penggambaran terkait batas waktu.

4. Interaction Overview Diagram

Interaction overview diagram mirip dengan diagram aktivitas yang berfungsi untuk menggarbarkan sekumpulan urutan aktivitas, diagram ini adalah bentuk aktivias diagram yang setiap titik merepresentasikan diagram interaksi.

2.13 Matlab

Matlab dikembangkan sebagai bahasa pemograman sekaligus alat visualisasi, yang menawarkan banyak kemampuan untuk menyelesaikan berbagai kasus yang berhubungan langsung dengan disiplin ilmu matematika. Matlab dibangun dari bahasa induknya yaitu bahasa C, namun tidak dapat dikatakan sebagai varian dari bahasa C, karena dalam sintak maupun cara kerjanya sama sekali berbeda dari bahasa C. Namun dengan hubungan langsungnya dengan bahasa C, Matlab memiliki kelebihan-kelebihan dari bahasa C bahkan mampu berjalan pada semua *platform* Sistem Operasi tanpa mengalami perubahan sintak sama sekali. Bahasa ini mengintegrasikan kemampuan komputasi, visualisasi dan pemograman dalam sebuah lingkungan yang tunggal dan mudah digunakan. Matlab memberikan sistem interaktif yang menggunakan konsep array/matrik sebagai standar variabel elemennya tanpa membutuhkan pendeklarasian array seperti pada bahasa lainnya.

Matlab dikembangkan oleh MathWorks, yang pada awalnya dibuat untuk memberikan kemudahan mengakses data matrik pada proyek LINPACK dan EISPACK. Selanjutnya menjadi sebuah aplikasi untuk komputasi matrik. Dari sejak awal dipergunakan, matlab memperoleh masukan ribuan pemakai. Dalam lingkungan pendidikan ilmiah menjadi alat pemrograman standar bidang matematika, rekayasa dan keilmuan terkait. Dan dalam lingkungan industri dapat menjadi pilihan paling produktif untuk riset, pengembangan dan analisa.

2.14 Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan suatu investigasi yang dilakukan untuk mendapatkan informasi mengenai kualitas dari produk atau layanan yang sedang diuji. Pengujian perangkat lunak juga memberikan pandangan mengenai perangkat lunak secara obyektif dan *independen*, yang bermanfaat dalam operasional bisnis untuk memahami tingkat risiko pada implementasinya. Teknik-

teknik pengujian mencakup, namun tidak terbatas pada, proses mengeksekusi suatu bagian program atau keseluruhan aplikasi dengan tujuan untuk menemukan "*bug*" perangkat lunak. *Bug* merupakan suatu kesalahan desain pada suatu perangkat keras komputer atau perangkat lunak komputer yang menyebabkan peralatan atau program itu tidak berfungsi semestinya. *Bug* umumnya lebih umum dalam dunia perangkat lunak dibandingkan dengan perangkat keras [9].

Pengujian perangkat lunak merupakan suatu tahapan penting dalam pembangunan perangkat lunak. Pengujian dilakukan dengan cara mengevaluasi konfigurasi perangkat lunak yang terdiri dari spesifikasi kebutuhan, deskripsi perancangan, dan program yang dihasilkan. Hasil evaluasi kemudian dibandingkan dengan hasil uji yang diharapkan. Jika ditemukan kesalahan, maka perbaikan perangkat lunak harus dilakukan untuk kemudian diuji kembali.

Pengujian perangkat lunak adalah proses menjalankan dan mengevaluasi sebuah perangkat lunak secara manual maupun otomatis untuk menguji apakah perangkat lunak sudah memenuhi persyaratan atau belum, atau untuk menentukan perbedaan antara hasil yang diharapkan dengan hasil sebenarnya. Pelaksanaan pengujian perangkat lunak biasanya disesuaikan dengan metodologi pembangunan perangkat lunak yang digunakan.

Pengujian dikatakan berhasil apabila pengujian yang dilakukan memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya.

2.14.1 Teknik Pengujian

Teknik pengujian yang dapat digunakan dalam menguji perangkat lunak sebagai berikut :

a. Pengujian *White Box*

Pengujian *white box* adalah pengujian yang didasarkan pada pengecekan terhadap detail perancangan, menggunakan struktur kontrol dari desain program secara procedural untuk membagi pengujian ke dalam beberapa kasus pengujian.

Secara sekilas dapat diambil kesimpulan *white box testing* merupakan petunjuk untuk mendapatkan program yang benar secara 100%.

Penggunaan metode pengujian *white box* dilakukan untuk :

1) Memberikan jaminan bahwa semua jalur independen suatu modul digunakan minimal satu kali
2) Menggunakan semua keputusan logis untuk semua kondisi true atau false
3) Mengeksekusi semua perulangan pada batasan nilai dan operasional pada setiap kondisi.
4) Menggunakan struktur data internal untuk menjamin validitas jalur keputusan.

Persyaratan dalam menjalankan strategi *White Box Testing* sebagai berikut:

1) Mendefinisikan semua alur logika
2) Membangun kasus untuk digunakan dalam pengujian
3) Mengevaluasi semua hasil pengujian
4) Melakukan pengujian secara menyeluruh

b. Pengujian *Black Box* :

Pengujian *black box* digunakan untuk menguji fungsi-fungsi khusus dari perangkat lunak yang dirancang. Pada teknik ini, kebenaran perangkat lunak yang diuji hanya dilihat berdasarkan keluaran yang dihasilkan dari data atau kondisi masukan yang diberikan untuk fungsi yang ada tanpa melihat bagaimana proses untuk mendapatkan keluaran tersebut. Dari keluaran yang dihasilkan, kemampuan program dalam memenuhi kebutuhan pemakai dapat diukur sekaligus dapat diiketahui kesalahan-kesalahannya.

Beberapa jenis kesalahan yang dapat diidentifikasi :

1) Fungsi tidak benar atau hilang
2) Kesalahan antar muka
3) Kesalahan pada struktur data (pengaksesan basis data)
4) Kesalahan inisialisasi dan akhir program
5) Kesalahan performasi

Walaupun sulit untuk menelusuri kesalahan yang mungkin didapat, teknik pengujian *black box* lebih sering dipilih untuk menguji perangkat lunak karena kemudahan dalam pelaksanaannya.