2 BAB 2
LANDASAN TEORI

2.1 Logo

Kata logo berasal dari bahasa yunani *logos*, yang berarti kata, pikiran, pembicaraan dan akal budi. Logo sebenarnya merupakan penyingkatan dari *logotype*, namun kini istilah logo lah yang lebih populer. Tidak ada keseragaman istilah dalam mengartikan logo. Mengutip "*Design Dictionary"* dari *"Board of international Research In Design (BIRD)"*, logo biasanya mengandung teks, gambar, atau kombinasi keduanya. Elemen teks/tulisan pada logo biasanya disebut dengan *logotype*, sedangkan simbol/gambar yang mewakili sebuah kata/makna disebut *logogram*. Secara garis besar, logo adalah bentuk simbol visual dari suatu perusahaan, produk, tempat, servis, kampus, dan lain sebagainya yang dilihat oleh setiap orang, yang didalamnya terkandung unsur warna, garis, bentuk, dan elemen desain lainnya, yang sengaja dibuat ubtuk memberikan suatu arti tertentu terhadap objek yang diwakilkan.

2.2 Pengolahan Citra (*Image Processing*)

Pengolahan citra (*image processing*) adalah pengolahan suatu citra (gambar) dengan menggunakan komputer secara khusus, untuk menghasilkan suatu citra yang lain. Sesuai dengan perkembangan komputer itu sendiri, pengolahan citra mempunyai dua tujuan utama [3], yaitu sebagai berikut:

1. Memperbaiki kualitas citra, dimana citra yang dihasilkan dapat menampilkan informasi secara jelas. Hal ini berarti manusia sebagai pengolah informasi *(human perception).*
2. Mengekstraksi informasi ciri yang menonjol pada suatu citra, dimana hasilnya adalah informasi citra dimana manusia mendapatkan informasi ciri dari citra secara numerik atau dengan kata lain *computer* (mesin) melakukan interprestasi terhadap informasi yang ada pada citra melalui besaran-besaran data yang dapat dibedakan secara jelas (berupa besaran numerik).

Dalam perkembangan lebih lanjut, *image processing* dan *computer vision* digunakan sebagai pengganti mata manusia dengan perangkat *input image capture*

seperti kamera dan *scanner* dijadikan sebagai mata dan mesin komputer dijadikan sebagai otak yang mengolah informasi. Oleh sebab itu uncul beberapa pecahan bidang yang menjadi penting dalam *computer vision* antara lain, *pattern recognition* (pengenalan pola), *biometric* (pengenalan identifikasi manusia berdasarkan ciri-ciri biologis yang tampak pada badan manusia), *content based image and video retrieval* (mendapatkan kembali citra atau video dengan informasi tertentu), *video editing* dan lain-lain.

2.3 Citra Digital

Citra secara harafiah, adalah gambar pada bidang dua dimensi (dwimatra). Ditinjau dari sudut pandang matematis, citra merupakan fungsi menerus (*continue*) dari intensitas cahaya pada bidang dwimatra. Sedangkan citra digital adalah citra yang dapat diolah oleh komputer[3].

Sebuah citra digital dapat diwakili oleh sebuah matriks yang terdiri dari M kolom dan N baris, dimana perpotongan antara kolom dan baris disebut *pixel*, yaitu elemen terkecil dari sebuah citra. Sebuah citra digital dapat ditulis dalam bentuk fungsi 2.1.

$$f(x,y) = \begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,M-1) \\ f(1,0) & \cdots & \cdots & f(1,M-1) \\ \cdots & \cdots & \cdots & \cdots \\ f(N-1,0) & f(N-1,1) & \cdots & f(N1,M-1) \end{bmatrix} \quad (2.1)$$

Terdapat banyak format file dari citra digital, format file citra ini terdiri dari beberapa jenis dan setiap jenisnya memiliki karakteristik masing- masing.

2.3.1 Karakteristik Citra Digital

Setiap citra digital memiliki beberapa karakteristik, antara lain yaitu :

a. Ukuran Citra Digital

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

b. Resolusi

Atribut citra digital yang tak kalah pentingnya adalah resolusi (resolution), yang didefinisikan sebagai banyaknya *pixel* dalam setiap satuan panjang. Umumnya, resolusi dinyatakan dalam satuan dpi (*dot per inch*). Sebagai contoh, citra digital yang memiliki resolusi 72 dpi, berarti terdiri dari 72 *dot* (titik) pada setiap inchi. Semakin tinggi resolusi suatu citra digital, maka kualitasnya akan semakin baik.

2.3.2 Tipe Citra Digital

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

1. Citra Biner

   Citra yang hanya memiliki dua nilai yaitu 1 dan 0. Dinyatakan dalam fungsi 2.2:

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

2. Citra Grey-scale

   Citra yang terdiri dari satu layer warna dengan derajat keabuan tertentu. Untuk kebanyakan citra digital 8 bit, maka sistem Grey-scale diukur berdasarkan skala intensitas kecerahan, yang bernilai 0 – 255, dimana yang hitam pekat adalah 0 dan yang terputih adalah 255. Dinyatakan dalam fungsi 2.3 :

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

3. Citra Berwarna

   Citra standar (Bitmap) dengan ukuran 512 × 512 dan mempunyai kedalaman warna 24 bit atau dikenal dengan istilah *true color*, dalam hal ini kedalaman warna yang dimaksud yaitu jumlah bit yang dipakai untuk merepresentasikan tiap titik pada gambar tersebut dinyatakan dalam bit per *pixel* . komputer mempunyai tiga komponen warna yaitu, merah, hijau, dan biru. Ketiga komponen dasar ini dikenal dengan istilah warna RGB (*Red Green Blue*). Untuk citra digital dengan 8 bit per *pixel* mempunyai 256

warna sedangkan pada citra dengan 24 bit per *pixel* mempunyai 16 juta warna, tiap *pixel* dinyatakan dengan :

Bit ke-0 sampai ke-7 untuk warna merah

Bit ke-8 sampai ke-15 untuk warna hijau

Bit ke-16 sampai ke-23 untuk warna biru

Sehingga kombinasi warnanya adalah $256^3 + 256^2 + 256^1 = 16.843.008$. Warna-warna ini merupakan warna dasar penyusun setiap *pixel* pada citra digital. Representasi dalam citra digital dinyatakan dalam persamaan 2.4, 2.5, dan 2.6:

$$f_R(x,y)\ \Sigma\ [0\ldots255] \quad (2.4)$$

$$f_G(x,y)\ \Sigma\ [0\ldots255] \quad (2.5)$$

$$f_B(x,y)\ \Sigma\ [0\ldots255] \quad (2.6)$$

2.3.3 Format File Citra Digital

Beberapa contoh format umum, pada citra digital yaitu *Bitmap (BMP)*, *Joint Photographic Group Experts (JPEG)*, *Graphics Interchange Format (GIF)*, dan *Portable Network Graphics (PNG)*.

1. *Bitmap (BMP)*

   Bitmap merupakan format baku citra pada sistem operasi *windows* dan IBM OS/2. Citra berformat BMP merupakan citra yang tidak terkompresi, sehingga pada umumnya citra berformat BMP mempunyai ukuran yang relatif lebih besar dibandingkan dengan forman citra lainnya. Intensitas pixel dari citra berformat BMP dipetakan ke sejumlah bit tertentu. Panjang setiap pixel pada *bitmap* yaitu 4 bit, 8 bit, sampai 24 bit yang merepresentasikan nilai intensitas *pixel*. Dengan demikian ada sebanyak 28 = 256 derajat keabuan, mulai dari 0 sampai 255[4].

2. *Joint Photographic Group Experts (JPEG)*

   Joint Photographic Group Experts (JPEG) merupakan standar kompresi file yang dikembangkan oleh *Group Joint Photographic Experts* menggunakan kombinasi DCT dan pengkodean *Huffman* untuk mengkompresikan citra. Citra JPEG merupakan citra terkompresi yang bersifat *lossy*, artinya citra tidak bisa dikembalikan ke bentuk aslinya. Citra

ini memiliki ukuran yang relatif lebih kecil dibandingkan dengan citra berformat BMP karena telah terkompresi.

3. *Graphics Interchange Format (GIF)*

   Graphics Interchange Format (GIF) merupakan format citra terkompresi. Ukuran dari citra berformat GIF merupakan yang terkecil dari semua format citra digital. Kombinasi warna yg tersedia sebanyak 256 warna. Citra jenis ini banyak dugunakan untuk keperluan website, untuk membuat sebuah icon, logo dan keperluan lainnya yang mengharuskan menggunakan citra dengan ukuran yang kecil. Citra jenis ini juga dapat dibuat animasi.

4. *Portable Network Graphics (PNG)*

   Portable Network Graphics (PNG) adalah salah satu format penyimpanan citra yang menggunakan metode kompresi yang tidak menghilangkan bagian dari citra tersebut (*lossless compression*). Citra berformat PNG merupakan salah satu format yang baik untuk digunakan pengolahan citra, karena format ini selain tidak menghilangkan bagian dari citra yang sedang diolah.

2.4 Citra Grayscale

Citra digital *grayscale* atau *greyscale* adalah suatu citra dimana nilai dari setiap piksel merupakan sample tunggal [5]. Citra yang ditampilkan dari citra jenis ini terdiri atas warna abu-abu, bervariasi pada warna hitam pada bagian yang intensitas terlemah dan warna putih pada intensitas terkuat. Citra *grayscale* berbeda dengan citra ”hitam-putih”, dimana pada konteks komputer, citra hitam putih hanya terdiri atas 2 warna saja yaitu ”hitam” dan ”putih” saja. Pada citra *grayscale* warna bervariasi antara hitam dan putih, tetapi variasi warna diantaranya sangat banyak. Citra *grayscale* seringkali merupakan perhitungan dari intensitas cahaya pada setiap piksel pada spektrum elektromagnetik *single band*.

Citra *grayscale* disimpan dalam format 8 bit untuk setiap sample piksel, yang memungkinkan sebanyak 256 intensitas. Format ini sangat membantu dalam pemrograman karena manupulasi bit yang tidak terlalu banyak. Pada aplikasi lain seperti pada aplikasi *medical imaging* dan *remote sensing* biasa juga digunakan format 10,12 maupun 16 bit.

Untuk mendapatkan nilai *grayscale* dari sebuah citra, dapat dilakukan dengan formula 2.7 seperti berikut :

$$Luminosity = 0.21 \times R + 0.71 \times G + 0.07 \times B \quad (2.7)$$

Dengan :

R = nilai piksel matriks R.

G = nilai piksel matrik G.

B = nilai piksel matrik B.

2.5 Citra Biner

Citra biner adalah citra dimana piksel-piakselnya hanya memiliki dua buah nilai intensitas yaitu bernilai 0 dan 1 dimana 0 menyatakan warna latar belakang *(background)* dan 1 menyatakan warna tinta/objek *(foreground)* atau dalam bentuk angka 0 untuk warna hitam dan angka 255 untuk warna putih [5]. Citra biner yang didapatkan dari citra *grayscale* dapat diperoleh dengan rumus :

$$T = \frac{fmaks+fmin}{2} \quad (2.8)$$

Dengan :

T = Nilai normalisasi.

$fmaks$ = Nilai intensitas maksimum pada citra.

$fmin$ = Nilai intensitas minimum pada citra.

2.6 Jarak Euclidean (Euclidean *Distance*)

Jarak Euclidean dapat dianggap sebagai jarak yang paling pendek antar dua poin - poin, maka dari itu dalam proyek akhir ini digunakan fungsi jarak Euclidean dan pada dasarnya sama halnya dengan persamaan Pythagoras ketika digunakan di dalam dua dimensi. Secara matematis dapat dituliskan di dalam persamaan berikut.

$$d = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \quad (2.9)$$

Dengan :

d = Nilai jarak.

x_i = Nilai pada fitur 1.

y_i = Nilai pada fitur 2.

n = Banyak nilai.

2.7 Deteksi Tepi (Edge Detection)

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

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

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

2.7.1 Sobel

Sobel adalah salah satu operator yang digunakan sebagai filter untuk pendeteksian tepi suatu citra digital. Pada operator sobel memiliki kernel (*Mask*) 3×3 untuk perhitungannya dalam setiap *pixel*. Operator Sobel yang digunakan dapat dilihat pada gambar 2.1 :

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

(a)

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

(b)

Gambar 2.1 Kernel Horizontal Sobel (a) Dan Kernel Vertikal Sobel (b)

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

$$G(x, y) = G1(x, y)* G1(x, y) + G2(x, y) * G2(x, y) \quad (2.7)$$

Sebagai contoh untuk menyelesaikan sebuah citra digital mempunyai ukuran 5 × 5 *pixel* pada gambar 2.2 :

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

Gambar 2.2 Matrik 5 x 5 Citra Digital

Menggunakan proses pendeteksian tepi dengan menggunakan operator Sobel, dimana f(2,2) = 160 yaitu :

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

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

Maka h(2,2) bila menggunakan :

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

$$= \sqrt{(90*90) + (210*210)}$$

$$= 228{,}4 \approx 228$$

2.8 Proses Pencocokkan *(Matching Process)*

Matching Process digunakan untuk menghitung jarak antar tepi*(edge)*. Dimana dalam tahapan ini menentukan tingkat kemiripan dari tepi citra uji dan tepi citra latih. Semakin kecil jaraknya maka citra tersebut semakin mirip dengan citra uji. Pada penelitian ini, proses pencocokkan menggunakan perhitungan tepi. Perhitungan ini diimplementasikan dengan rumus euclidean sebagai berikut :

$$d = \sqrt{(j_1 - j_2)^2 + (i_1 - i_2)^2} \quad (2.10)$$

Dengan :

d = Jarak dari dua citra grayscale dengan perhitungan chamfer.

j_1 = data koordinat ke-j_1 pada tepi *(edge)* citra uji.

j_2 = data koordinat ke-j_2 pada tepi *(edge)* citra yang ada di latih.

i_1 = data koordinat ke-i_1 pada tepi *(edge)* citra uji.

i_2 = data koordinat ke-i_2 pada tepi *(edge)* citra yang ada di latih.

2.9 Chamfer Matching

Chamfer *matching* adalah metode yang dilakukan untuk mencari jarak terdekat dari titik tepi dari citra. Perhitungan ini diimplementasikan dengan rumus sebagai berikut :

$$\sum_{i=1}^{n} \min(\left|\left|p_{i,}q_1\right|\right|, ||p_i, q_2||, \cdots ||p_i, q_m||) \quad (2.11)$$

Dengan :

p_i = titik yang ada pada citra uji.

q_j = titik yang ada pada citra latih.

2.10 Unified Modelling Language (UML)

Unified Modelling Language (UML) adalah sebuah bahasa pemodelan standar yang memiliki sintaks dan semantic. Pemodelan ini sangat cocok digunakan untuk merancang dan memodelkan sistem berorientasi objek [6]. Diagram pada UML dibagi menjadi dua bagian yaitu *structural* diagram dan *behavior* diagram. *Structural* diagram digunakan untuk mendeskripsikan relasi antar kelas. *Tools* yang digunakan pada bagian ini yaitu *class diagram*. Sedangkan *behavior* diagram digunakan untuk mendeskripsikan interaksi antara aktor dan sebuah *use case* (bagaimana seorang aktor menggunakan sistem). *Tools* yang digunakan pada bagian ini yaitu *Use case* diagram, *Sequence* diagram, dan *Activity* diagram.

2.10.1 Diagram Use Case

Diagram *use case* digunakan untuk mendeskripsikan kejadian-kejadian apa saja yang dapat dilakukan oleh *user*/aktor dan fungsionalitas-fungsionalitas apa saja yang diharapkan dari sistem yang akan dibangun, tanpa mendeskripsikan bagaimana sistem menyelesaikannya [5]. Sebuah *use case* menggambarkan suatu

urutan interaksi antara satu atau lebih aktor dan sistem. Dalam fase requirements, model *use case* mengambarkan sistem sebagai sebuah kotak hitam dan interaksi antara aktor dan sistem dalam suatu bentuk naratif, yang terdiri dari *input user* dan respon-respon sistem.

Setiap *use case* menggambarkan perilaku sejumlah aspek sistem, tanpa mengurangi struktur internalnya. Selama pembuatan model *use case* secara pararel juga harus ditetapkan obyek-obyek yang terlibat dalam setiap *use case*.

2.10.2 Diagram Sequence

Diagram *sequence* menggambarkan interaksi antar objek di dalam dan di sekitar sistem yang menekankan pada pengiriman pesan dalam suatu waktu tertentu [6]. *Sequence* diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

2.10.3 Diagram Activity

Diagram *activity* menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi, dan bagaimana mereka berakhir [6]. *Activity* diagram merupakan *state* diagram khusus, di mana sebagian besar state adalah action dan sebagian besar transisi di-trigger oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity* diagram tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari *level* atas secara umum.

2.10.4 Diagram Class

Diagram *class* digunakan untuk menggambarkan keadaan suatu sistem dengan menjelaskan keterhubungan antara suatu *class* dengan *class* yang lain yang terdapat pada sistem[6]. Sebuah *class* terdiri dari nama, atribut dan *method.* Atribut dan *method* dari sebuah kelas mempunyai visibility, Ada tiga jenis visibility yang digunakan yaitu *private*, *public*, dan *protected*. Setiap *class* pada *class* diagram

mempunyai hubungan dengan *class* lainnya, ada beberapa jenis hubungan *class*, yaitu *dependency*, asosiasi, agregasi, komposisi dan generalisasi.

a. *Dependency*

   Dependency merupakan hubungan terlemah antar *class*. *Dependency* bermakna satu *class* menggunakan atau memiliki pengetahuan terhadap *class* lain, namun hubungannya hanya sementara dan tidak ada batas waktu yang jelas.

b. Asosiasi

   Asosiasi mempunyai hubungan yang lebih kuat dari hubungan *dependency*, dimana suatu *class* tetap berhubungan dengan *class* lain seterusnya. Hubungan asosiasi dibagi menjadi dua jenis yaitu *directional* dan *bidirectional*.

c. Agregasi

   Agregasi merupakan bentuk hubungan yang mengimplikasikan kepemilikan suatu *class*. Agregasi juga merupakan bentuk yang lebih kuat dari asosiasi. Hubungan agregasi dinyatakan dengan simbol *diamond* pada pemilik kelas dan garis utuh berpanah kekelas yang dimiliki.

d. Komposisi

   Komposisi merupakan bentuk hubungan antar *class* yang paling kuat. Komposisi digunakan untuk mengambil seluruh bagian dari *class* yang berhubungan. Aturan dari komposisi yaitu hanya boleh ada satu komposisi dalam satu waktu.

e. Generalisasi

   Generalisasi merupakan bentuk hubungan antar *class*, dari *class* yang umum dengan *class* yang lebih khusus. Contoh *class* hewan memiliki hubungan generalisasi dengan *class* kucing, karena kucing merupakan *class* khusus dari *class* hewan yang lebih umum.

2.11 Java

Java adalah sebuah teknologi yang diperkenalkan oleh *Sun Microsystems* pada pertengahan tahun 1990. Menurut definisi dari *Sun*, *Java* adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada

komputer *stand alone* ataupun pada lingkungan jaringan. *Java* berdiri di atas sebuah mesin *interpreter* yang diberi nama *Java Virtual Machine* (JVM). JVM inilah yang akan membaca *bytecode* dalam *file .class* dari suatu program sebagai representasi langsung program yang berisi bahasa mesin.

Oleh karena itu, bahasa *Java* disebut sebagai bahasa pemrograman yang *portable* karena dapat dijalankan pada berbagai sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM. *Platform Java* terdiri dari kumpulan *library*, JVM, kelas-kelas *loader* yang dipaket dalam sebuah lingkungan rutin *Java*, dan sebuah *compiler*, *debuger*, dan perangkat lain yang dipaket dalam *Java Development Kit* (JDK) [7].

Java merupakan salah satu bahasa pemrograman yang menggunakan paradigma pemrograman berbasis objek. Paradigma ini yaitu menggunakan objek untuk membungkus atribut dan operasi yang mungkin pada objek tersebut. Java mempunyai kelebihan dan kekurangan yaitu:

a. Kelebihan Java

Beberapa kelebihan dari java yaitu sebagai berikut:

1. Multiplatform

   Kelebihan utama dari *java* ialah dapat dijalankan di beberapa *platform*/sistem operasi komputer, sesuai dengan prinsip tulis sekali, jalankan dimana saja. Kelebihan ini memungkinkan sebuah program berbasis java dikerjakan diatas sistem operasi *linux* tetapi dijalankan dengan baik di atas *Microsoft Windows*.

2. OOP (*Object Oriented Programming*)

   Java merupakan salah satu bahasan pemrograman berbasis objek secara murni. Semua tipe data diturunkan dari kelas dasar yang disebut objek. Hal ini sangat memudahkan pemrogram untuk mendesain, membuat, mengembangkan, dan mengalokasi kesalahan sebuah program dengan basis *java* secara cepat, tepat, mudah dan terorganisir.

3. *Library* yang lengkap

   Java terkenal dengan kelengkapan *library*/perpustakaan (kumpulan program-program yang disertakan dalam pemrogram *java*) yang

sangat memudahkan dalam penggunaan oleh para pemrogram untuk membangun aplikasinya. Kelengkapan perpustakaan ini ditambah dengan keberadaan komunitas java yang besar yang terus menerus membuat perpustakaan-perpustakaan baru untuk melingkupi seluruh kebutuhan pembangunan aplikasi.

4. Bergaya C++

   Java memiliki sintaks seperti bahasa pemrograman C++ sehingga menarik banyak pemrogram C++ untuk pindah ke *java*. Saat ini pengguna *java* sangat banyak, sebagian besar adalah pemrogram C++ yang pindah ke *java*.

5. Pengumpulan sampah otomatis

   Java memiliki fasilitas pengaturan penggunaan memori sehingga para pemrogram tidak perlu melakukan pengaturan memori secara langsung (seperti halnya dalam bahasa C++ yang dipakai secara luas) [6].

b. Kekurangan Java

Beberapa kekurangan Java antara lain :

1. Mudah didekompilasi

   Dekompilasi adalah proses membalikkan dari kode jadi menjadi kode sumber. Ini memungkinkan karena kode jadi *java* merupakan *bytecode* yang menyimpan banyak atribut bahasa tingkat tinggi seperti nama-nama kelas, metode dan tipe data.

2. Penggunaan memori yang banyak

   Penggunaan memori untuk program berbasis *java* jauh lebih besar daripada bahasa tingkat tinggi generasi sebelumnya seperti C/C++ dan *Pascal* (lebih spesifik lagi *delphi* dan Objek *Pascal*) [7].