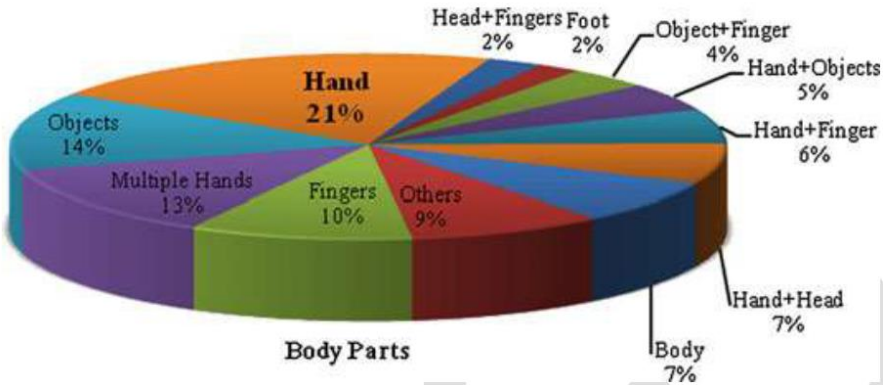
BAB 2
LANDASAN TEORI

2.1 Hand Gesture

Gesture merupakan komunikasi non verbal untuk mengekspresikan diri seseorang lewat gerakan. *Gesture* bisa digunakan lewat gerakan-gerakan tubuh atau salah satu bagian tubuh untuk berkomunikasi dengan seseorang. *Gesture* menjadi peranan penting dalam hal komunikasi manusia dan komputer. Penggunaan *gesture* atau gerakan tubuh juga memilki arti berbeda di setiap negara karena perbedaan budaya yang ada di setiap negara.

Dalam penelitiannya Karam, menyatakan bahwa manusia lebih banyak menggunakan *hand gesture* untuk komunikasi non verbal dalam kehidupan sehari-harinya jika dibandingkan bagian tubuh lainnya, bisa dilihat perbandinganya pada Gambar 2.1 dibawah ini[8].



Gambar 2.1 Perbandingan Penggunaan *Gesture* Tubuh

Sebuah sistem pengenalan *hand gesture* dapat dibagi menjadi dua tahap, yaitu tahap akuisisi *hand gesture* dan tahap pengenalan (*recognition*). Tahap akuisisi merupakan proses untuk mendapatkan pola *hand gesture* yang dilakukan oleh pengguna sistem tersebut. Untuk *hand gesture* dinamis tahapannya terdiri *dari hand detection, hand segmentation* dan *hand tracking*. Untuk proses *hand tracking* dapat dibagi menjadi dua macam, yaitu dalam bidang dua dimensi atau dalam bidang tiga

dimensi. Pada tahap pengenalan dilakukan proses identifikasi atau proses pengelompokkan (*clustering*) untuk menginterpre-tasikan *hand gesture* yang didapatkan dari tahap akuisisi[9].

2.2 Pengolahan Citra Digital

Secara umum, istilah pengolahan citra digital menyatakan "pemrosesan gambar berdimensi-dua melalui komputer digital. Foto adalah contoh gambar berdimensi dua yang dapat diolah dengan mudah. Setiap foto dalam bentuk citra digital (misalnya berasal dari kamera digital) dapat diolah melalui perangkat tertentu. Sebagai contoh, apabila hasil bidikan kamera terlihat agak gelap, citra dapat diolah menjadi lebih terang dimungkinkan pula untuk memisahkan foto orang dari latar belakangnya. Gambaran tersebut menunjukkan hal sederhana yang dapat dilakukan melalui pengolahan citra digital. Tentu saja banyak hal pelik lain yang dapat dilakukan melalui pengolahan citra digital[10].

Pengolahan citra menjadi bagian penting yang mendasari berbagai aplikasi nyata, seperti pengenalan pola, penginderaan jarak jauh melalui satelit atau pesawat udara, dan *machine vision*. Pada pengenalan pola, pengoahan citra antara lain berperan memisahkan objek dari latar belakang secara otomatis. Selanjutnya, objek akan diproses oleh pengklasifikasi pola. Pada pnginderaan jarak jauh, tekstur atau warna pada citra dapat dipakai untuk mengidentifikasi objek-objek yang terdapat dalam citra. Pada *machine vision* (sistem yang dapat "melihat" dan "memahami" yang dilihatnya), pengolahan citra berperan untuk mengenali bentuk-bentuk khusus yang dilihat oleh mesin.

2.3 Jenis Citra Digital

Berdasarkan jenis warnanya, citra digital umumnua dapat dibedakan menjadi tiga jenis yaitu citra berwarna, citra berskala abu-abu, dan citra biner. Citra berwarna atau RGB (*Red, Green, Blue*) merupakan citra yang nilai intensitas pikselnya tersusun oleh tiga kanal warna yaitu merah, hijau, dan biru. Citra berskala abu-abu atau *grayscale* adalah citra yang nilai intensitas pikselnya berdasarkan derajat keabuan. Sedangkan citra biner adalah citra yang hanya memiliki dua nilai intensitas yaitu 0 (hitam) dan 1 (putih)[10]. Selain itu juga ada citra HSV (*Hue,*

Saturation, Value) biasanya sering digunakan untuk segmentasi citra pada warna tertentu

2.3.1 Citra Berwarna (RGB)

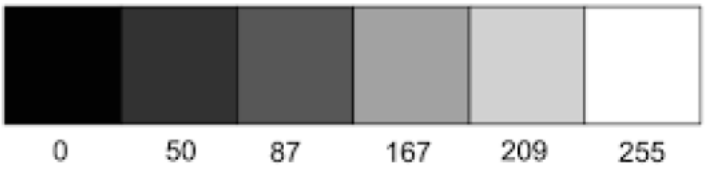
Citra berwarna, atau biasa dinamakan citra RGB, merupakan jenis citra yang menyajikan warna dalam bentuk komponen R(merah), G(hijau), dan B(biru). Tiap komponen warna menggunakan 8 bit (nilainya berkisar antara 0 sampai dengan 255). Dengan demikian, kemungkinan warna yang dapat disajikan mencapai 255 x 255 x 255 atau 16.581.375 warna. Tabel 2.1 menunjukkan contoh warna R, G dan B.

Tabel 2.1 Tabel Warna RGB

Warna	R	G	B
Merah	255	0	0
Hijau	0	255	0
Biru	0	0	255
Hitam	0	0	0
Putih	255	255	255
Kuning	0	255	255

2.3.2 Citra Skala Keabuan (*Grayscale*)

Sesuai dengan nama yang melekat, citra jenis ini menangani gradasi warna hitam dan putih, yang tentu saja menghasilkan efek warna abu-abu (*Grayscale*). Pada jenis gambar ini, warna dinyatakan dengan intensitas. Dalam hal ini, intensitas berkisar antara 0 sampai dengan 255. Nilai 0 menyatakan hitam dan nilai 255 menyatakan putih. Berikut merupakan penjelasan nilai citra skala keabuan terdapat pada Gambar 2.2



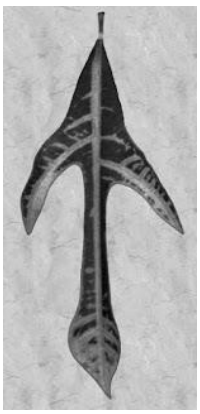
Gambar 2.2 Nilai Citra Skala Keabuan

Untuk mendapatkan nilai citra skala keabuan dapat dilakukan perhitungan sebagai berikut :

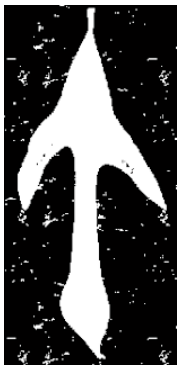
$$Grayscale = (0.299 * R) + (0.587 * G) + (0.114 * B) \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (2.1)$$

2.3.3 Citra Biner

Citra biner adalah citra dengan setiap piksel hanya dinyatakan dengan sebuah nilai dari dua kemungkinan (yaitu nilai 0 dan 1). Nilai 0 menyatakan warna hitam dan nilai 1 menyatakan warna putih. Citra jenis ini banyak dipakai dalam pemrosesan citra, misalnya untuk kepentingan memperoleh tepi bentuk suatu objek. Sebagai contoh perhatikan pada Gambar 2.3. Bagian kiri menyatakan atas beraras keabuan, sedangkan bagian bawah adalah hasil konvesi ke citra biner.



a) Citra daun berskala abu-abu



b) Citra biner

Gambar 2.3 Hasil konversi citra *grayscale* ke biner

Untuk mendapatkan nilai citra skala keabuan dapat dilakukan perhitungan sebagai berikut :

$$g(x,y) = \begin{cases} 1, jika\ f(x,y) \geq T \\ 0, jika\ (f,y) < T \end{cases} \quad \text{………………………………………………. (2.2)}$$

Dimana :

f(x,y) : Nilai citra grayscale

g(x,y) : Nilai citra biner

T : Nilai threshold

2.3.4 Citra HSV

HSV mendefinisikan warna dalam terminologi *Hue, Saturation* dan *Value*. Keuntungan HSV adalah terdapat warna-warna yang sama dengan yang ditangkap oleh indra manusia. Sedangkan warna yang dibentuk model lain seperti RGB merupakan hasil campuran dari warna-warna primer. Karakteristik dari ketiga terminologi tersebut adalah sebagai berikut :

1. *Hue*, menyatakan warna sebenarnya seperti merah, violet dan kuning. Digunakan untuk menentukan kemerahan (*Redness*), kehijauan (*Greeness*) dan lain sebagainya.
2. *Saturation,* kadang disebut sebagai Chroma adalah kemurnian atau kekuatan warna.
3. *Value*, kecerahan dari warna. Nilainya berkisar antara 0-100%. Apabila nilainya 0 maka warnanya akan menjadi hitam, semakin besar nilai maka semakin cerah dan muncul variasi- variasi baru dari warna tersebut.

Berikut ini merupakan perhitungan nilai HSV dari pengubahan citra RGB menjadi citra HSV[10]:

$$r = \frac{R}{(R+G+B)}, g = \frac{R}{(R+G+B)}, b = \frac{R}{(R+G+B)} \quad \text{………………… (2.3)}$$

$$V = \max(r, g, b) \quad \text{.......} \quad (2.4)$$

$$S = \begin{cases} 0, & jika\, V = 0 \\ 1 - \dfrac{\min(r, g, b)}{V}, & V > 0 \end{cases} \quad \text{.......} \quad (2.5)$$

$$H = \begin{cases} 0, jika\, S = 0 \\ \dfrac{60 * (g - b)}{S * V}, jika\, V = r \\ 60 * \left[2 + \dfrac{b - r}{S * V}\right], jika\, V = g \\ 60 * \left[4 + \dfrac{r - b}{S * V}\right], jika\, V = b \end{cases} \quad \text{.......} \quad (2.6)$$

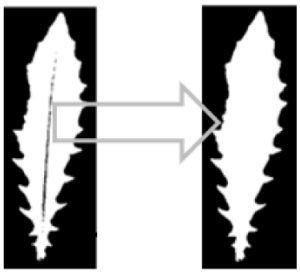
$$H = H + 360, jika\, H < 0 \quad \text{.......} \quad (2.7)$$

R,G,B adalah nilai dari citra RGB yang akan dikonversikan ke citra HSV.

2.4 Morfologi Pengolahan Citra

Operasi morfologi merupakan operasi yang umum dikenakan pada citra biner (hitam-putih) untuk mengubah struktur bentuk objek yang terkandung dalam citra. Sebagai contoh, lubang pada daun dapat ditutup melalui operasi morfologi sebagaimana ditunjukkan di Gambar 2.4. Beberapa contoh lain aplikasi morofologi adalah sebagai berikut[10] :

a. Membentuk filter spasial
b. Memperoleh skeleton (rangka) objek
c. Menentukan letak objek di dalam citra
d. Memperoleh bentuk struktur objek



Gambar 2.4 Tulang daun dapat dianggap sebagai bagian melalui morfologi

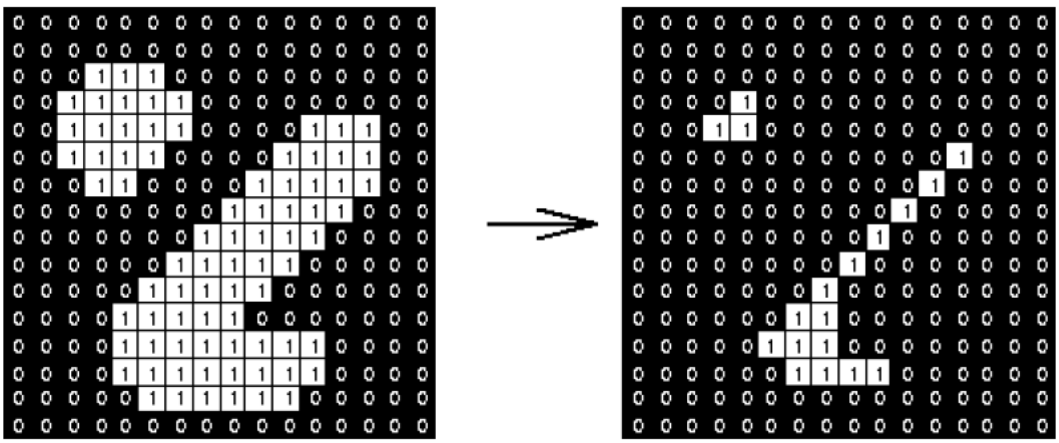
2.4.1 Erosi

Operasi *erode* (Erosi) mempunyai efek memperkecil struktur citra. Operasi erosi dapat dirumuskan sebagai berikut[10]:

$$A \ominus B = \{ p \in z^2 \mid (a + b) \in |, untuk\ setiap\ b \in B \} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad (2.8)$$

Dimana A merupakan *f(x,y)* dari citra asli dan B adalah elemen penstruktur atau biasa disebut strel. Elemen penstruktur yang biasa digunakan dalam operasi erosi adalah bentuk kotak. Bentuk elemen penstruktur lainnya ada yang berupa elipse, garis, piringan dan lainnya.

Hasil erosi biasanya merupakan operasi nalar **AND** dari setiap koordinat A dan B. Berikut merupakan hasil dari operasi *erode* terdapat pada Gambar 2.5



Gambar 2.5 Hasil operasi erosi

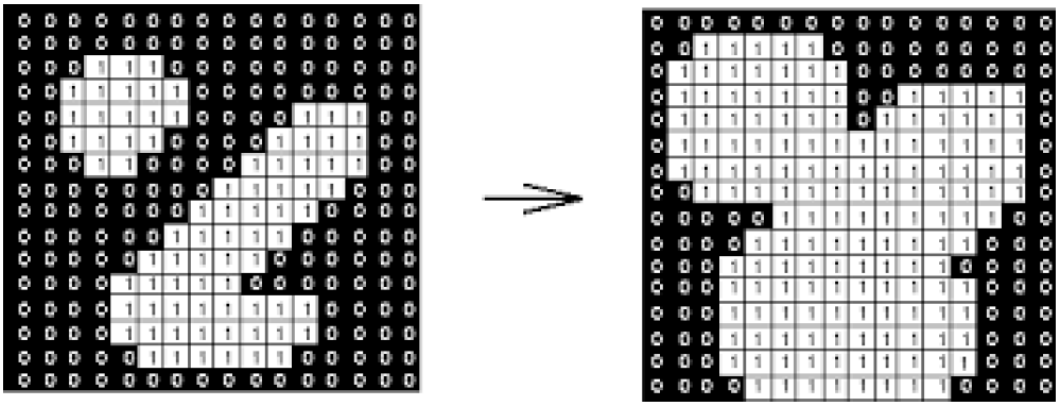
2.4.2 Dilasi

Operasi *dilate* (Dilasi) biasa dipakai untuk mendapatkan efek pelebaran terhadap piksel bernilai 1. Operasi dilasi dapat dirumuskan sebagai berikut[10]:

$$A \oplus B = \{ z \mid z = a + b, dengan\ a\ \in A\ dan\ b\ \in B \} \quad \ldots\ldots\ldots\ldots\ldots\ldots (2.9)$$

Dimana A merupakan *f(x,y)* dari citra asli dan B adalah elemen penstruktur atau biasa disebut strel. Elemen penstruktur yang biasa digunakan dalam operasi dilasi juga biasanya adalah berbentuk kotak.

Hasil dilasi berupa penjumlahan seluruh pasangan koordinat dari A dan B. Berikut merupakan hasil dari operasi dilasi terdapat pada Gambar 2.6



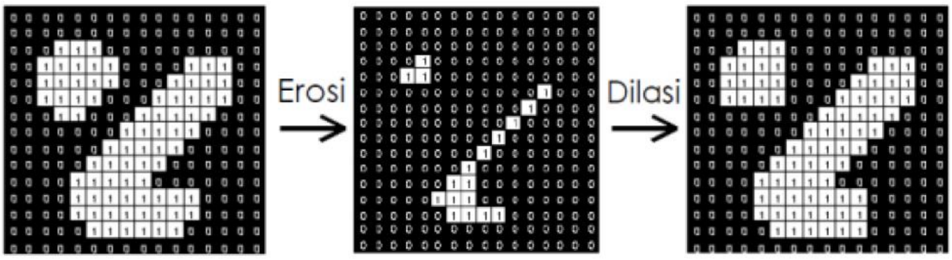
Gambar 2.6 Hasil operasi dilasi

2.4.3 Opening (Erosi - Dilasi)

Operasi *opening* (Erosi-Dilasi) merupakan kombinasi antara operasi erosi dan dilasi yang dilakukan secara berurutan, tetapi citra asli dierosi terlebih dahulu baru kemudian hasilnya didilasi. Operasi ini digunakan untuk memutus bagian-bagian dari objek yang hanya terhubung dengan 1 atau 2 buah titik saja, atau menghilangkan objek – objek kecil yang secara umum membuat *smooth* batas dari objek besar tanpa mengubah area objek secara signifikan. *Opening* adalah *idempotent* yaitu apabila operasi opening diulang-ulang tidak akan memberikan dampak yang berkelanjutan. Operasi opening dapat dirumuskan sebagai berikut[10].

$$A \circ B = (A \ominus B) \oplus B \quad \text{...} \quad (2.10)$$

Berikut merupakan hasil dari operasi opening terdapat pada Gambar 2.7



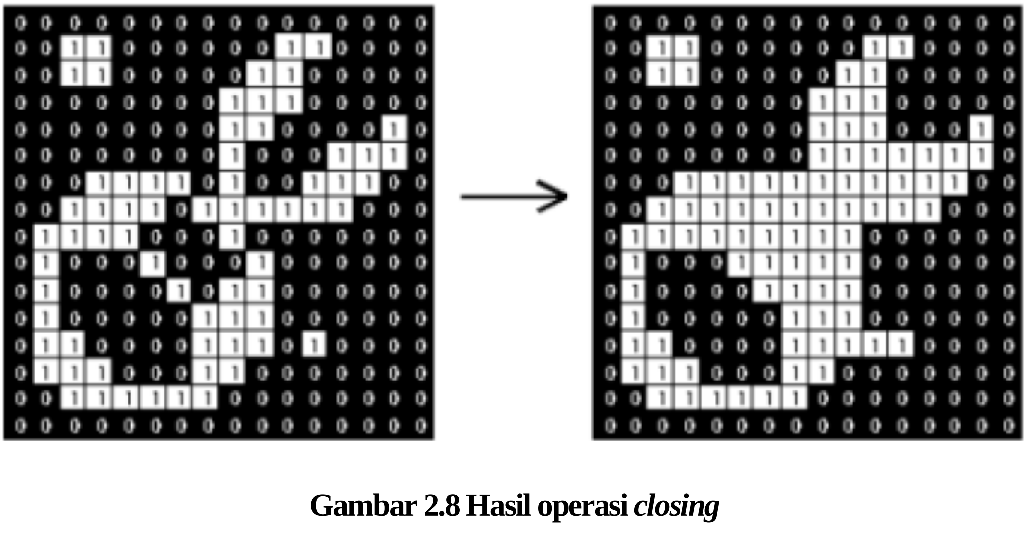
Gambar 2.7 Citra hasil *opening*

2.4.4 Closing (Dilasi - Erosi)

Operasi *closing* (Dilasi-Erosi) adalah kombinasi antara operasi dilasi dan erosi yang dilakukan secara berurutan. Citra asli didilasi terlebih dahulu, kemudian hasilnya dierosi. Operasi ini digunakan untuk menutup atau menghilangkan lubang-lubang kecil yang ada dalam segmen objek, menggabungkan objek yang berdekatan dan secara umum membuat *smooth* batas dari objek besar tanpa mengubah objek secara signifikan. Operasi *closing* dapat dirumuskan sebagai berikut[10].

$$A \bullet B = (A \oplus B) \ominus B \quad \text{...} \quad (2.11)$$

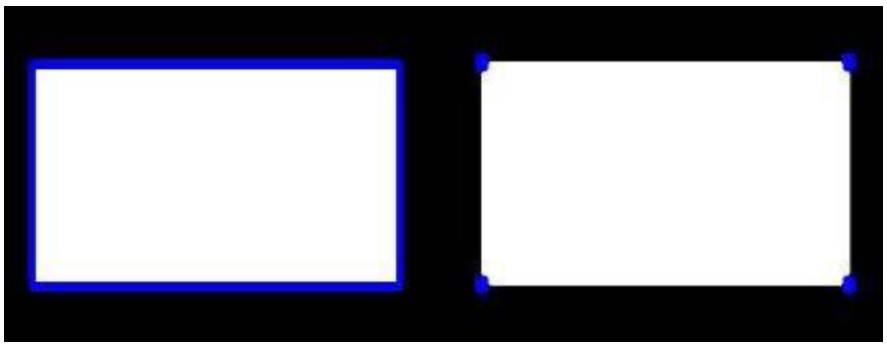
Berikut merupakan hasil dari operasi *closing* terdapat pada Gambar 2.8



Gambar 2.8 Hasil operasi *closing*

2.4.5 *Contours*

Countours bisa dijelaskan lebih simpelnya merupakan gabungan kurva – kurva dari titik yang berkelanjutan (disepanjang daerah) dimana mempunyai warna dan intensitas yang sama. Kontur ini akan berguna sebagai alat untuk mendeteksi, pengenalan dan menganalisis bentuk dari objek. Berikut ini merupakan contoh dari Contours dapat dilihat pada Gambar 2.9

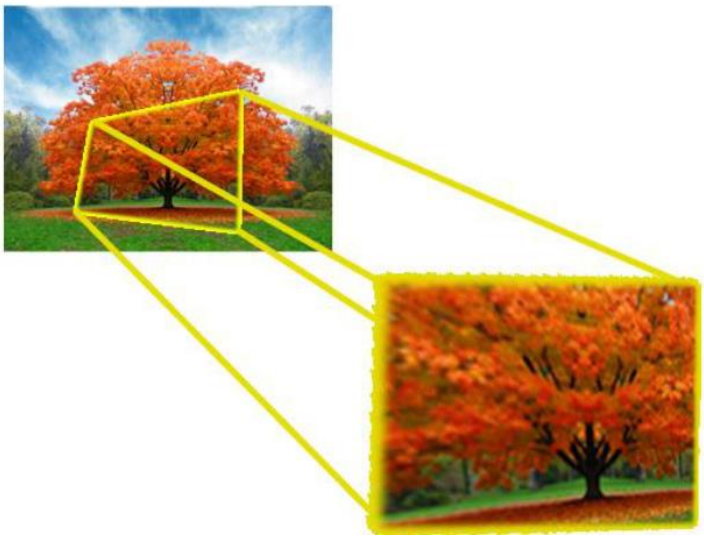


Gambar 2.9 Hasil citra *contours*

2.5 Segmentasi Citra

Segmentasi citra merupakan proses yang ditujukan untuk mendapatkan objek-objek yang terkandung di dalam citra atau membagi citra ke dalam beberapa daerah dengan setiap objek atau daerah memiliki kemiripan atribut. Pada citra yang mengandung sejumlah objek, prose suntuk memilah semua objek tentu saja lebih kompleks. Contoh penerapan segmentasi citra untuk membuat fasilitas semacam "*Magic Wand*", yang biasa terdapat pada aplikasi pengedit foto.[10]

ROI (*Region of Interest*) adalah proses mengambil bagian tertentu pada suatu citra dengan nilai kontur terbesar. Berikut ini merupakan contoh dari *Region of Interest* (ROI) yang terdapat pada Gambar 2.10



Gambar 2.10 Citra hasil *Region of Interest*

2.6 *Scale-Invariant Feature Transform* (SIFT)

Pada tahun 1999, David G. Lowe seorang peneliti dari *University of British Columbia* memperkenalkan suatu metode baru dalam ekstraksi fitur dari suatu citra. Metode ekstraksi fitur ini disebut sebagai SIFT. Dengan menggunakan SIFT ini, suatu citra akan di ubah menjadi vektor fitur lokal yang kemudian akan digunakan sebagai salah satu pendekatan untuk melakukan deteksi dan ekstraksi deskriptor fitur lokal tersebut[11].

Sebagai metode ekstraksi fitur pada pengenalan objek, SIFT ini memiliki kelebihan-kelebihan sebagai berikut:

1. Hasil ekstraksi fitur bersifat invariant terhadap ukuran, tranlasi dan rotasi dua dimensi.
2. Hasil ekstraksi fitur bersifat invariant sebagian terhadap perubahan iluminasi dan perubahan sudut pandang tiga dimensi.
3. Mampu meng-ekstrak banyak keypoint dari citra yang tipikal.
4. Hasil ekstraksi fitur benar-benar mencirikan secara khusus (*distinctive*).

Dengan kelebihan-kelebihan tersebut, penggunaan metode SIFT banyak dikembangkan untuk aplikasi pengenalan objek.

Secara garis besar, algoritma yang digunakan pada metode SIFT terdiri dari empat tahap, antara lain [12] :

1. Mencari Nilai Ekstrim Pada Skala Ruang

Pencarian nilai ekstrim pada skala ruang merupakan tahap awal dalam penentuan *keypoint* dari suatu citra. Dengan menggunakan fungsi *Gaussian-Blurred*, citra pada skala ruang dapat didefinisikan sebagai fungsi *L(x,y,σ)*, yang diperoleh dari hasil konvolusi skala-variabel *Gaussian*, *G(x,y,σ)*, dengan citra masukan *I(x,y)*, sehingga diperoleh:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad \text{....} \qquad (2.12)$$

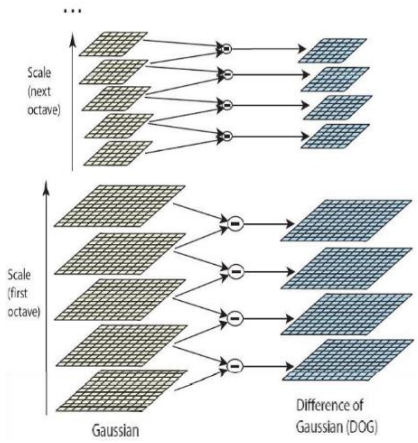
Dimana * adalah operasi konvolusi antara *x* dan *y* dan *G(x,y,σ)* adalah skala-skala variabel *Gaussian*.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2 + y^2)/2\sigma^2} \quad \text{....} \qquad (2.13)$$

Citra hasil *Difference-of-Gaussian*, *D(x,y,σ),* diperoleh dengan melakukan operasi konvolusi pada citra masukan dengan *filter Difference-of-Gaussian*, maka:

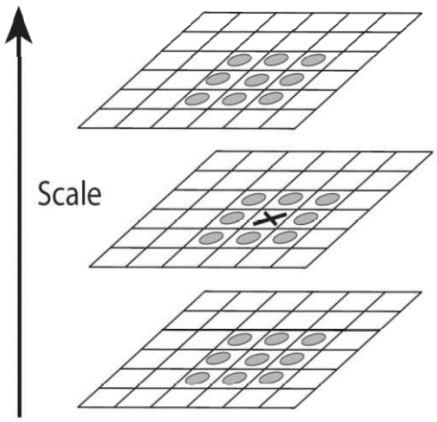
$$\begin{aligned} D(x, y, \sigma) &= \big(G(x, y, k\sigma) - G(x, y, k\sigma)\big) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \quad \text{....} \end{aligned} \qquad (2.14)$$

Dari persamaan (2.14) terlihat bahwa citra hasil *Difference-of-Gaussian* (DoG), sebenarnya merupakan selisih antara citra hasil pengkaburan *Gaussian* dengan nilai skala k yang berbeda. Proses ini diilustrasikan pada Gambar 2.1



Gambar 2.11 Proses konvolusi citra dengan *filter* DoG

Citra hasil konvolusi dikelompokkan berdasarkan *octave* (satu *octave* setara dengan penggandaan besarnya nilai σ), nilai k ditetapkan di awal sehingga diperoleh jumlah citra kabur yang sama pada setiap *octave* serta diperoleh citra hasil DoG yang sama untuk setiap *octave*. Setelah diperoleh citra DoG pada setiap *octave*, maka langkah selanjutnya ialah mencari kandidat *keypoint*. Kandidat *keypoint* dideteksi sebagai titik maksimum lokal atau titik maksimum local dari citra hasil DoG. Untuk mencari nilai maksimum dan minimum lokal maka masing-masing *pixel* pada citra hasil DoG akan dibandingkan dengan 8 *pixel* disekitarnya yang berada pada skala yang sama dengan 9 *pixel* yang bersesuaian dengannya (pada Gambar 2.12). Jika *pixel* tersebut merupakan maksimum atau minimum lokal, maka *pixel* tersebut akan dijadikan sebagai kandidat *keypoint*.



Gambar 2.12 Ilustrasi pencarian maksimum atau minimum *local*

2. Menentukan Keypoint

Setelah kandidat *keypoint* ditentukan melalui tahapan pertama, maka langkah selanjutnya ialah untuk mengambil detail dari kandidat *keypoint* tersebut. Detail yang diambil merupakan lokasi, skala dan rasio kelengkungan inti dari kandidat *keypoint*. Pada tahap ini akan terjadi pengurangan jumlah kandidat *keypoint*. Dimana setiap kandidat *keypoint* yang dianggap sangat rentan terhadap gangguan (*noise*) akan dihilangkan, yaitu kandidat *keypoint* yang memiliki nilai kontras yang rendah dan kandidat *keypoint* yang kurang jelas dan terletak di sepanjang tepi.

Untuk setiap kandidat keypoint akan:

a. Dilakukan interpolasi dengan data terdekat di sekitarnya untuk menentukan posisi yang tepat.
b. Dibuang *keypoint* dengan kontras yang rendah.
c. Diberikan orientasi tertentu pada *keypoint* tersebut

3. Penentuan Orientasi

Pada tahap ini, masing-masing *keypoint* yang diperoleh akan diberikan suatu orientasi yang tetap berdasarkan sifat-sifat lokal pada citra. Dengan adanya proses ini maka *keypoint* yang diperoleh dapat direpresentasikan relative terhadap orientasi ini sehingga *keypoint* yang dihasilkan tidak terpengaruh terhadap adanya rotasi pada citra. Untuk menentukan orientasi dari masing-masing keypoint maka dilakukan perhitungan terhadap besar nilai *gradient*, *m(x,y)*, dan arah orientasi, *θ(x,y)*, dilakukan menggunakan persamaan berikut:

$$m(x,y) = \sqrt{(L((x+1,y) - L(x-1,y))2 - (L(x,y+1) - L(x,y-1))^2} \quad \ldots(2.15)$$

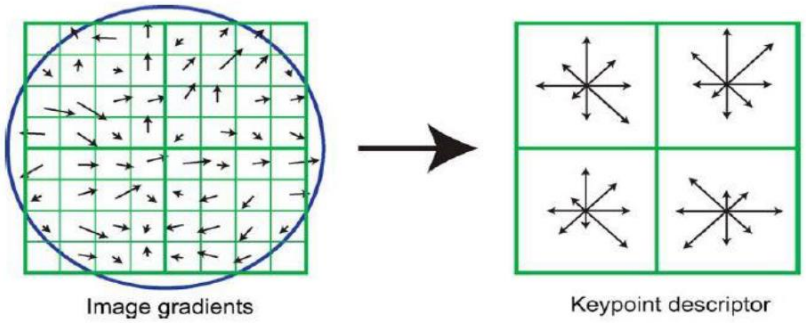
$$\theta(x,y) - tan^1\left(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\right) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots.. \quad (2.16)$$

4. Deskriptor *Keypoint*

Pada proses ini, masing-masing *keypoint* yang telah diorientasikan akan diberikan pencirian khusus (deskriptor). Proses ini bertujuan untuk mendapatkan

keypoint yang invariant terhadap perubahan intensitas cahaya atau perubahan sudut pandang tiga dimensi.

Deskriptor akan diukur sebagai suatu *histogram* orientasi pada wilayah *pixel* dengan ukuran 4x4. Nilai orientasi diperoleh dari citra *Gaussian* yang memiliki skala terdekat dengan skala *keypoint* yang akan dihitung. Agar *keypoint* yang diperoleh *invariant* terhadap orientasi, maka koordinat dari deskriptor dan *gradient* orientasi akan di rotasi *relative* terhadap orientasi dari *keypoint*. Kemudian fungsi pembebanan *Gaussian*, dengan besar nilai σ satu setengah kali dari besar jendela deskriptor, akan digunakan sebagai pembeban pada setiap besaran nilai dari titik sampel. Proses ini ditunjukkan pada lingkaran yang terdapat pada Gambar 2.13 sebelah kiri.



Gambar 2.13 Keypoint Descriptor

Deskriptor *keypoint* pada Gambar 2.13 menunjukkan adanya 8 arah pada masingmasing histogram orientasi dengan panjang masing-masing anak panah sesuai dengan besar nilai dari histogram asal. Selanjutnya deskriptor *keypoint* yang telah diperoleh akan dinormalisasi untuk mengatasi pengaruh perubahan cahaya.

Keypoint ini yang kemudian menjadi fitur-fitur lokal pada suatu citra dan akan dicocokkan dengan *keypoint-keypoint* yang terdapat pada citra *query* untuk menyesuaikan dengan objek yang tersedia menggunakan *matrix homography*.

2.7 K-Nearest Neighbor (KNN)

K-Nearest Neighbor sangat sering digunakan dalam klasifikasi dengan tujuan dari algoritma ini adalah untuk mengklasifikasi objek baru berdasarkan atribut dan *training samples*[13]. Algoritma *K-nearest neighbor* (KNN) adalah sebuah metode

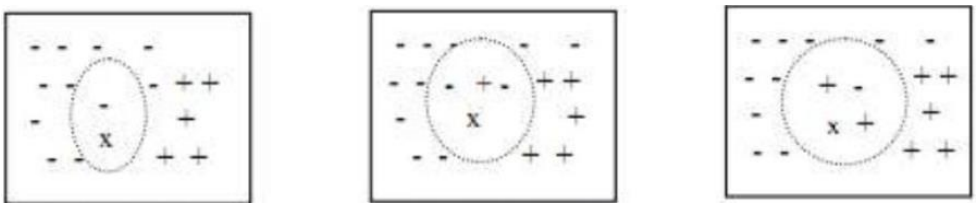
untuk melakukan klasifikasi terhadap objek berdasarkan data pembelajaran yang jaraknya paling dekat dengan objek tersebut. Teknik ini sangat sederhana dan mudah diimplementasikan. Tujuan dari algoritma ini adalah mengklasifikasikan objek baru berdasarkan atribut dan *training sample*.

KNN memiliki beberapa kelebihan yaitu ketagguhan terhadap *training data* yang memiliki banyak noise dan efektif apabila training datanya besar. Sedangkan kelemahan KNN adalah KNN perlunya menentukan nilai parameter K (jumlah dari tetangga terdekat), training berdasarkan jarak tidak jelas mengenai jenis jarak apa yang harus digunakan untuk mendapatkan hasil terbaik, dan biaya komputasi sangat tinggi karena diperlukan perhitungan jarak dari tiap *query instance* pada keseluruhan *training sample*.

Ketepatan algoritma KNN sangat dipengaruhi oleh ada atau tidak adanya fitur-fitur yang tidak relevan atau jika bobot fitur tersebut setara dengan relevansinya terhadap klasifikasi. Riset terhadap algoritma ini sebagian besar membahas bagaimana memilih dan memberi bobot terhadap fitur agar performa klasifikasi menjadi lebih baik.

Classifier tidak menggunakan model apapun untuk dicocokkan dan hanya berdasarkan pada memori. Diberikan titik *query*, akan ditemukan sejumlah K obyek atau titik *training* yang paling dekat dengan titik *query*. Klasifikasi menggunakan voting terbanyak diantara klasifikasi dari K obyek algoritma KNN menggunakan klasifikasi ketetanggaan sebagai nilai prediksi dari *query instance* yang baru. Mirip dengan teknik *clustering*, pengelompokkan suatu data baru berdasarkan jarak data baru itu ke beberapa data/tetangga (*neighbor*) terdekat. Dalam hal ini jumlah data/tetangga terdekat ditentukan oleh *user* yang dinyatakan dengan K. Misalkan ditentukan K=5, maka setiap data *testing* dihitung jaraknya terhadap data *training* dan dipilih 5 data *training* yang jaraknya paling dekat ke data *testing*. Lalu periksa *output* atau labelnya masing-masing, kemudian tentukan *output* mana yang frekuensinya paling banyak. Lalu masukkan suatu data *testing* ke kelompok dengan *output* paling banyak. Misalkan dalam kasus klasifikasi dengan 3 kelas, lima data tadi terbagi atas tiga data dengan *output* kelas 1, satu data dengan *output* kelas 2

dan satu data dengan *output* kelas 3, maka dapat disimpulkan bahwa *output* dengan label kelas 1 adalah yang paling banyak. Maka data baru tadi dapat dikelompokkan ke dalam kelas 1. Prosedur ini dilakukan untuk semua data testing[13]. Gambar berikut ini adalah bentuk representasi KNN dengan 1, 2 dan 3 tetangga data terhadap data baru x.



a) 1-*nearest neighbor* b) 2-*nearest neighbor* c) 3-*nearest neighbor*

Gambar 2.14 Ilustrasi 1-, 2-, 3-nearest neighbor terhadap data baru (x)

Untuk mendefinisikan jarak antara dua titik yaitu titik pada data *training* (*x*) dan titik pada data testing (*y*) maka digunakan rumus *Euclidean*:

$$d_i = \sqrt{\sum_{i=1}^{p}(x_{2i} - x_{1i})^2} \quad \text{.....} \quad (2.17)$$

Dengan :

x_1 = sampel data

x_2 = data uji

i = variabel data

d = jarak

p = dimensi data

Langkah-langkah untuk menghitung metode *K-Nearest Neighbor* :

1. Menentukan parameter K (jumlah tetangga paling dekat).
2. Menghitung kuadrat jarak *Euclidean* (*query instance*) masing-masing obyek terhadap data sampel yang diberikan.

3. Kemudian mengurutkan objek-objek tersebut kedalam kelompok yang mempunyai jarak *Euclidean* terkecil.
4. Mengumpulkan kategori Y (Klasifikasi *nearest neighbor*).
5. Dengan menggunakan kategori *nearest neighbor* yang paling mayoritas maka dapat dipredisikan nilai *query instance* yang telah dihitung.

2.8 *Web Camera*

Webcam merupakan gabungan dari kata web dan camera. *Webcam* sendiri sebutan bagi kamera *real-time* (bermakna keadaan pada saat ini juga) yang gambarnya bisa diakses atau dilihat melalui internet , program *instant messaging* seperti Yahoo Messenger , AOL Instant Messenger (AIM), Windows Live Messenger , dan Skype, dan lainnya. Istilah "*webcam*" sendiri mengarah pada jenis kamera yang digunakan untuk kebutuhan layanan berbasis web. *Webcam* sendiri biasanya digunakan untuk keperluan konferensi jarak jauh atau juga sebagai kamera pemantau.

Webcam juga merupakan sebuah periferal berupa kamera sebagai pengambil citra/gambar dan mikropon (*optional*) sebagai pengambil suara/audio yang dikendalikan oleh sebuah komputer atau oleh jaringan komputer. Gambar yang diambil oleh Webcam ditampilkan ke layar monitor, karena dikendalikan oleh komputer maka ada *interface* atau *port* yang digunakan untuk menghubungkan *Webcam* dengan komputer atau jaringan. Ada beberapa orang mengartikan *Webcam* sebagai *Web pages + Camera*, karena dengan menggunakan *Webcam* untuk mengambil gambar video secara aktual bisa langsung di upload bila komputer yang mengendalikan terkoneksi internet[14]. Berikut merupakan contoh webcam terdapat pada Gambar 2.15.

Gambar 2.15 Webcam Logitech C270

2.9 Netbeans IDE

Netbeans merupakan sebuah aplikasi *Integrated Development Environment* (IDE) yang berbasiskan Java dari *Sun Microsystems* yang berjalan di atas *swing*. *Swing* merupakan sebuah teknologi Java untuk pengembangan aplikasi desktop yang dapat berjalan pada berbagai macam platform seperti windows, linux, Mac OS X dan Solaris. Sebuah IDE merupakan lingkup pemrograman yang di integrasikan ke dalam suatu aplikasi perangkat lunak yang menyediakan *Graphic User Interface* (GUI), suatu kode editor atau text, suatu *compiler* dan suatu *debugger*[13].

2.10 OpenCV (*Open Source Computer Vision Library*)

OpenCV (*Open Source Computer Vision Library*) adalah *sebuah library open source* yang dikembangkan oleh intel yang fokus untuk menyederhanakan *programming* terkait citra digital. Di dalam OpenCV sudah mempunyai banyak fitur, antara lain : pengenalan wajah, pelacakan wajah, deteksi wajah, Kalman filtering, dan berbagai jenis metode AI (*Artificial Intellegence*). OpenCV juga menyediakan berbagai algoritma sederhana terkait *Computer Vision* untuk *low level* API.

OpenCV mempunyai banyak fitur yang dapat dimanfaatkan, berikut ini adalah fitur utama dari OpenCV antara lain[15]:

a. *Image and video I/O*

   Dengan antar muka ini kita dapat membaca data gambar dari file, atau dari umpan video langsung. Dan juga dapat menciptakan file gambar maupun video.

b. *Computer Vision* secara umum dan pengolahan citra digital (untuk *low* dan *mid level* API)

   Dengan antar muka ini kita dapat melakukan experimen uji coba dengan berbagai standar algoritma *computer vision*. Termasuk juga deteksi garis, tepi, pucuk, proyeksi elips, *image pyramid* untuk pemrosesan gambar multi skala,

pencocokan template, dan berbagai *transform* (*Fourier, cosine diskrit, distance transform*) dan lain lain.

c. Modul *computer vision high level*

Di dalam OpenCV juga termasuk kemampuan "*high level*", seperti kemampuan tambahan untuk deteksi wajah, pengenalan wajah, termasuk *optical flow*

d. Metode untuk AI dan *machine learning*
Aplikasi *computer vision* sering kali memerlukan *machine learning* atau metode AI lainnya, beberapa metode tersebut tersedia dalam paket OpenCV *machine learning*.

e. Sampling gambar dan transformasi
Di dalam OpenCV sudah terdapat antar muka untuk substraksi *subregion* dari gambar, random sampling, rotating, dan lain lain.

f. Metode untuk menciptakan dan menganalisa gambar biner

g. Metode untuk memperhitungkan pemodelan 3D
Fungsi ini sangat bermanfaat untuk *mapping* dan *localization*, baik untuk *stereo camera* ataupun satu kamera dengan berbagai sudut pandang.

2.11 UML (*Unified Modelling Language*)

UML (*Unified Modeling Language*) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem. *Unified Modeling Language* (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat tool untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP.

UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM[7]. UML adalah suatu bahasa yang digunakan untuk menentukan,

memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. 25 UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan system. UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu:

1. *Use Case Diagram*

   Use case diagram adalah gambaran graphical dari beberapa atau semua Aktor, use-case, dan interaksi diantara komponen-komponen tersebut yang memperkenalkan suatu sistem yang akan dibangun. Use-case diagram menjelaskan manfaat suatu sistem jika dilihat menurut pandangan orang yang berada di luar sistem. Diagram ini menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem tersebut berinteraksi dengan dunia luar.

   Use-case diagram dapat digunakan selama proses analisis untuk menangkap requirement sistem dan untuk memahami bagaimana sistem seharusnya bekerja. Selama tahap desain, *use case diagram* berperan untuk menetapkan perilaku (*behavior*) sistem saat diimplementasikan. Dalam sebuah model mungkin terdapat satu atau beberapa use-case diagram. Kebutuhan atau *requirements system* adalah fungsionalitas apa yang harus disediakan oleh sistem kemudian didokumentasikan pada model use-case yang menggambarkan fungsi sistem yang diharapkan (*use case*), dan yang mengelilinginya (Aktor), serta hubungan antara Aktor dengan use case (*use case diagram*) itu sendiri.

2. *Conceptual Diagram*

   Sebuah diagram konseptual merupakan representasi visual dari cara di mana konsep-konsep abstrak terkait. Hal ini digunakan sebagai bantuan dalam memvisualisasikan proses atau sistem tingkat tinggi melalui serangkaian garis yang unik dan bagan. Diagram konseptual secara luas digunakan dalam segala bidang seperti bisnis, ilmu pengetahuan, dan manufaktur.

3. *Sequence Diagram*

   Sequence Diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa message yang digambarkan terhadap waktu. Sequence Diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

   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.

4. *Collaboration Diagram*

   Collaboration diagram yaitu diagram yang mengelompokkan pesan pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram tersebut terdapat method yang dijalankan antara objek yang satu dan objek lainnya. Di diagram kolaborasi ini, objek harus melakukan sinkronisasi pesan dengan serangkaian pesan-pesan lainnya. *Collaboration Diagram* lebih menekankan kepada peran setiap objek dan bukan pada waktu penyampaian pesan.

5. *State Diagram*

   State Diagram adalah diagram untuk menggambarkan behavior, yaitu perubahan state di suatu *Class* berdasarkan event dan pesan yang dikirimkan dan diterima oleh *Class* tersebut. Setiap diagram state hanya boleh memiliki satu start state (*initial state*) dan boleh memiliki satu atau lebih dari satu stop states (*final state*).

6. *Activity Diagram*

   Activity Diagram memiliki pengertian yaitu lebih fokus kepada menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Dipakai pada business modeling untuk memperlihatkan urutan aktifitas proses bisnis. Memiliki struktur diagram yang mirip *flowchart* atau data *flow* diagram pada perancangan terstruktur. Memiliki pula manfaat yaitu apabila kita membuat diagram ini terlebih dahulu dalam memodelkan sebuah proses untuk membantu

memahami proses secara keseluruhan. Dan *activity* dibuat berdasarkan sebuah atau beberapa use case pada *use case diagram.*

7. *Class Diagram*

   Class Diagram adalah sebuah *Class* yang menggambarkan struktur dan penjelasan Class, paket, dan objek serta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* diagram juga menjelaskan hubungan antar Class dalam sebuah sistem yang sedang dibuat dan bagaimana caranya agar mereka saling berkolaborasi untuk mencapai sebuah tujuan.

8. *Object Diagram*

   Object Diagram adalah diagram yang memberikan gambaran struktur model sebuah sistem, dalam kurun waktu tertentu. Diagram objek yang berasal dari diagram kelas sehingga diagram objek tergantung pada diagram kelas. Objek Diagram, kadang-kadang disebut sebagai *diagram instance* sangat mirip dengan diagram kelas. Seperti diagram kelas *object diagram* juga menunjukkan hubungan antara obyek, tetapi object diagram menggunakan contoh-contoh dunia nyata. *Object diagram* digunakan untuk menunjukkan bagaimana sistem akan terlihat seperti pada waktu tertentu. Karena ada data yang tersedia di objek-objek diagram sering digunakan untuk menjelaskan hubungan yang kompleks antara objek.

9. *Component Diagram*

   Component Diagram adalah diagram UML yang menampilkan komponen dalam system dan hubungan antara mereka. Pada component *View*, akan difokuskan pada organisasi fisik system. Pertama, diputuskan bagaimana kelas-kelas akan diorganisasikan menjadi kode pustaka. Kemudian akan dilihat bagaimana perbedaan antara berkas eksekusi, berkas *dynamic link library* (DDL), dan berkas runtime lainnya dalam system.

10. *Deployment Diagram*

    Deployment Diagram adalah diagram yang menggambarkan detail bagaimana komponen disebar kedalam infrastruktur sistem, dimana komponen akan terletak (pada mesin, node, server atau piranti keras apa), bagaimana

kemampuan jaringan pada lokasi tersebut, spesifikasi server, dan hal-hal lain yang bersifat fisikal.