BAB II

LANDASAN TEORI

2.1 Struktur Mata Manusia

Manusia membutuhkan informasi berupa rangsangan dari lingkungan luar sekitar untuk dapat menjalani hidupnya dengan baik. Agar rangsangan yang berasal dari luar tubuh dapat ditangkap dibutuhkan alat-alat tubuh tertentu yang bernama indera. Ada lima indra pada manusia, salah satunya adalah mata. Mata adalah indera yang digunakan untuk melihat lingkungan sekitarnya dalam bentuk gambar sehingga mampu dengan mengenali benda-benda yang ada di sekitarnya dengan cepat. Jumlah mata manusia ada dua buah yang bekerja saling menunjang satu sama lain. Bagian-bagian pada organ mata bekerjasama mengantarkan cahaya dari sumbernya menuju ke otak untuk dapat dicerna oleh sistem saraf manusia. Bagian-bagian tersebut adalah:

1. Kornea

Merupakan bagian terluar dari bola mata yang menerima cahaya dari sumber cahaya.

2. Pupil dan Iris

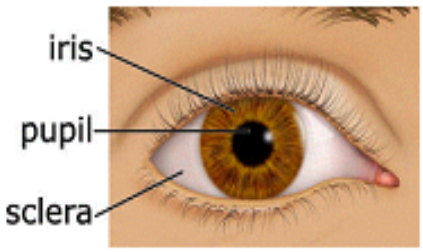
Dari kornea, cahaya akan diteruskan ke pupil. Pupil menentukan kuantitas cahaya yang masuk ke bagian mata yang lebih dalam. Pupil mata akan melebar jika kondisi ruangan yang gelap, dan akan menyempit jika kondisi ruangan terang. Lebar pupil dipengaruhi oleh iris di sekelilingnya.Iris berfungsi sebagai diafragma. Iris inilah terlihat sebagai bagian yang berwarna pada mata.

3. Lensa mata

Lensa mata menerima cahaya dari pupil dan meneruskannya pada retina. Fungsi lensa mata adalah mengatur fokus cahaya, sehingga cahaya jatuh tepat pada bintik kuning retina. Untuk melihat objek yang jauh (cahaya datang dari jauh), lensa mata akan menipis. Sedangkan untuk melihat objek yang dekat (cahaya datang dari dekat), lensa mata akan menebal.

4. Retina

Retina adalah bagian mata yang paling peka terhadap cahaya, khususnya bagian retina yang disebut bintik kuning. Setelah retina, cahaya diteruskan ke saraf optik.



Gambar 2.1 struktur mata manusia

2.2 Principal Componen Analysis

2.2.1 Pengertian Principal Componen Analysis (PCA)

Secara umum Principal Componen Analysis (PCA) dapat diartikan sebagai suatu cara untuk pengenalan pola terhadap suatu kumpulan data dan mengekspresikan kumpulan data tersebut terhadap kesamaan dan perbedaan diantaranya. PCA dapat digunakan untuk melakukan reduksi dimensi data tanpa mengurangi informasi-informasi dari data tersebut.

2.2.2 Ekstrasi Ciri Penting digunakan untuk Reduksi Dimensi

Dalam melakukan klasifikasi terhadap sekumpulan data, langkah yang terlebih dahulu dilakukan adalah melakukan reduksi demensi. Reduksi dimensi disini dapat diartikan sebagai suatu pemetaan terhadap suatu ruang dimensi multidimensi ke dalam dimensi yang lebih rendah. Hal ini dilakukan mengingat kumpulan data yang akan diklasifikasikan, dalam konteks dunia nyata, merupakan data yang *non linear separable*. Reduksi dimensi ini dapat dilakukan dengan melakukan ekstraksi ciri-ciri penting terhadap data yang akan di klasifikasikan. Ekstraksi ciri-ciri penting dilakukan dengan menciptakan suatu subset dari ciri-ciri baru dengan melakukan kombinasi dengan ciri-ciri yang telah ada.

Ekstrasi terhadap ciri-ciri penting suatu data dapat digambarkan dalam pemetaan matriks seperti berikut :

$$\begin{bmatrix} X_1 \\ X_2 \\ . \\ . \\ . \\ X_N \end{bmatrix} \longrightarrow \begin{bmatrix} X_1 \\ X_2 \\ . \\ . \\ . \\ X_m \end{bmatrix} = f \begin{bmatrix} Y_1 \\ Y_2 \\ . \\ . \\ . \\ Y_M \end{bmatrix}$$

Alasan PCA digunakan untuk melakukan ekstraksi ciri-ciri penting dikarenakan :

1. PCA memiliki efisiensi komputasi yang cukup tinggi.
2. PCA atau juga dikenal dengan metode Karhunen-Louve memilih proyeksi linear untuk melakukan reduksi dimensi terhadap sample data yang tersebar.

3. PCA dapat menangani berbagai ekspresi wajah dengan berbagai pose dan kondisi pencahayaan.

Dalam melakukan maksimasi proyeksi linear terhadap data sample tersebar, PCA melakukan pemilihan vector, W, yang mempunyai kontribusi terhadap distribusi daerah wajah di dalam suatu citra digital keseluruhan.

Setelah didapat vector-vektor yang merepresentasikan representasi ruang wajah, PCA melakukan proyeksi dimana dilakukan transformasi suatu ruang n-dimensi kedalam ruang m- dimensi dimana n > m :

$$\mathbf{y=W^t x}$$

2.3 Computer Vision

Terminologi lain yang berkaitan dengan pengolahan citra adalah *computer vision* atau *machine vision*. Pada hakikatnya, *computer vision* mencoba meniru cara kerja sistem visual manusia (*human vision*). *Human vision* sesungguhnya sangat kompleks. Manusia melihat objek dengan indera penglihatan (mata), lalu citra objek diteruskan ke otak untuk diinterpretasi sehingga manusia mengerti objek apa yang tampak dalam pandangan matanya. Hasil interpretasi ini mungkin digunakan untuk pengambilan keputusan. *Computer vision* merupakan proses otomatis yang mengintegrasikan sejumlah besar proses untuk persepsi visual, seperti akuisisi citra, pengolahan citra, klasifikasi, pengenalan (*recognition*), dan membuat keputusan. *Computer Vision* terdiri dari teknik-teknik untuk mengestimasi ciri-ciri objek di dalam citra,

pengukuran ciri yang berkaitan dengan geometri objek, dan menginterpretasi informasi geometri tersebut.

Proses-proses di dalam *computer vision* dapat dibagi menjadi tiga aktivitas:

1. Memperoleh atau mengakuisisi citra digital.
2. Melakukan teknik komputasi untuk memproses atau memodifikasi data citra (operasi-operasi pengolahan citra).
3. Menganalisis dan menginterpretasi citra dan menggunakan hasil pemrosesan untuk tujuan tertentu, misalnya memandu robot, mengontrol peralatan, memantau proses manufaktur, dan lain-lain.

Computer vision dalam melakukan proses-prosesnya tidak luput dari pengenalan pola, yakni mengelompokkan data numerik dan simbolik (termasuk citra) secara otomatis oleh mesin (dalam hal ini komputer). Tujuan pengelompokkan adalah untuk mengenali suatu objek di dalam citra. Manusia bisa mengenali objek yang dilihatnya karena otak manusia telah belajar mengklasifikasi objek-objek di alam sehingga mampu membedakan suatu objek dengan objek lainnya. Kemampuan visual manusia inilah yang dicoba ditiru oleh mesin. Komputer menerima masukan berupa citra objek yang akan diidentifikasi, memproses citra tersebut, dan memberikan keluaran berupa deskripsi ojek di dalam citra.



Gamba 2.2 Input dan Output Pengenalan Pola

2.4 Deteksi Mata (Eye Detection)

Tujuan pendeteksian mata adalah untuk menentukan ada atau tidak adanya suatu mata dalam suatu citra. Selanjutnya jika ada, kemudian dilanjutkan dengan mengadakan pencarian lokasi dan luas mata dalam citra tersebut. Dalam pendeteksian suatu mata ada beberapa faktor yang dapat mempengaruhi. Adapun faktor-faktor yang mempengaruhi tingkat kesulitan proses deteksi mata sebagai berikut :

1. Pose. Citra yang mengandung wajah yang beragam terkait dengan pose kamera-wajah, dan beberapa ciri wajah seperti mata atau hidung mungkin nampak sebagian.
2. Ada tidaknya struktur komponen. Ciri wajah seperti jenggot, kumis, dan kacamata mungkin atau tidak mungkin ada dan ini adalah masalah besar dalam hal keragaman di antara komponen wajah yang meliputi bentuk, warna, dan ukuran.
3. Kondisi-kondisi citra. Pada saat suatu citra dibentuk, faktor-faktor seperti pencahayaan (spektrum, intensitas dan sumber distribusi) dan karakteristik kamera (tanggapan sensor, lensa) mempengaruhi penampilan dari suatu wajah. Metoda pendeteksian citra tunggal digolongkan ke dalam empat kategori :

Tabel 2.1 metode pendeteksian citra tunggal

Metoda	Representative Works
Knowledge based method	⇒ Multiresolution rule-based Method
Feature invariant • Ciri wajah • Tekstur • Warna kulit • Ciri yang beragam	 ⇒ Pengumpulan sisi-sisi ⇒ Matriks Space Gray-Level Dependence (SGLD) pada pola wajah ⇒ Mixture of Gaussian ⇒ Integrasi antara warna kulit, ukuran dan bentuk
Template Matching • Predefined Face Template • Deformable Templates	 ⇒ Membentuk suatu template ⇒ Active Shape Model (ASM)
Appearance-based method • Eigenface • Distribution based method • Neural Network • Support Vector Machine • Naïve Bayes Classifier • Hidden Markov Model • Information-Theoretical Approach	 ⇒ Membentuk suatu template ⇒ Active Shape Model (ASM) ⇒ Eigenvector decomposition and clustering ⇒ Distribusi Gaussian dan multiplayer perceptron ⇒ Ensemble of neural network dan arbitration schemes ⇒ SVM dengan polynomial kernel ⇒ Penggabungan statistika penampilan local dan posisi ⇒ Urutan statistika yang lebih tinggi dengan HMM ⇒ Kullback relative information

2.4.1 Performasi Pendeteksian Mata

Ketika area wajah dideteksi, informasi lebar area wajah digunakan untuk mengkalkulasi ukuran baru dari citra training. Setelah melalui beberapa proses

pada langkah area wajah setelah itu proses pencarian dan kemudian area mata ditemukan.

Ada dua parameter penting di dalam metode adaptive *Eigeneye* yaitu :

1. Banyaknya citra training di dalam database yang digunakan disimbolkan dengan M
2. Banyaknya eigenvektor yaitu, *eigenfaces* adalah M'

Dalam teori probabilitas dan statistika, *variance* atau ragam dari peubah acak (atau distribusi probabilitas) adalah ukuran yang menunjukkan dispersi statistik (seberapa jauh data tersebar di sekitar rata-rata). Cek *variance* merupakan salah satu parameter penting selama pencarian mata, menggunakan nilai *variance* yang rendah, berarti lebih banyak kandidat untuk dekomposisi dan hasilnya mengakibatkan waktu perhitungan yang lebih lama. Pada sisi lain, menggunakan nilai *variance* yang tinggi berarti banyak dari area ditolak sebelum dekomposisi, maka mata mungkin ditolak sebagai bukan mata.

2.4.2 Efek Dari Jumlah Citra Training

Penggunaan jumlah citra training selama pendeteksian adalah suatu parameter yang sangat penting untuk metode *adaptive Eigeneye.* Setelah berhasil mendeteksi area wajah, untuk menyediakan skala *invarians* yang tergantung pada properti lebar wajah, database mata diukur kembali. jadi lebih banyaknya jumlah citra training akan meningkatkan komputational waktu. Banyaknya *eigeneyes* yang digunakan ketika diambil, sebaiknya sama dengan banyaknya citra training. Hal ini untuk memperoleh hasil pendeteksian yang akurat dan mengurangi kesalahan pada saat pendeteksian mata dilakukan. Untuk maksimum jarak bisa

diijinkan dari ruang mata, setelah dilakukan pengujian maka diperoleh nilai yang terbaik yaitu 1250. Jarak ini merupakan jarak yang ideal untuk melakukan pendeteksian. citra training dan area wajah ketika pencarian akan dilakukan telah diubah menjadi XYZ color space dan Z komponen dari citra telah digunakan metode eigeneye.

2.4.3 Efek Dari Jumlah Eigen Vektor

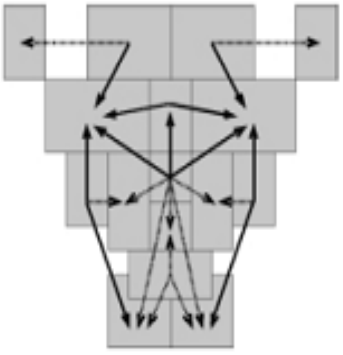
Jumlah maksimum *eigenvektor* yang dapat digunakan adalah banyaknya citra training M. Bagaimanapun dalam praktek suatu yang lebih kecil satuan *eigenvektor* cukup untuk proses pendeteksian karena *eigenvektor* dengan eigen nilai kecil mempunyai efek sangat kecil pada keakurasian pendeteksian. Setelah mengamati hasil bagian yang sebelumnya, banyaknya citra training yang digunakan telah diambil, yaitu 9 dan banyaknya *eigenvektor* adalah M'=9. Maksimal jarak yang diperbolehkan dari ruang mata terpilih 1250. citra training dan area wajah ketika pencarian akan dilakukan telah diubah menjadi XYZ *color space* dan Z komponen dari citra telah digunakan metode *eigeneye*.

2.5 Metode Template Matching

Prinsip dasar *template matching* melibatkan penempatan objek (yaitu wajah) di dalam suatu citra di mana terdapat korelasi yang kuat antara suatu template dan area yang ditetapkan dalam citra. *Template* adalah bagian dari sebuah citra yang berukuran tertentu, berisi sebuah obyek atau area yang ingin diproses. *Template* ditempatkan pada posisi obyek dalam citra input, kemudian dihitung nilai korelasi antar keduanya.

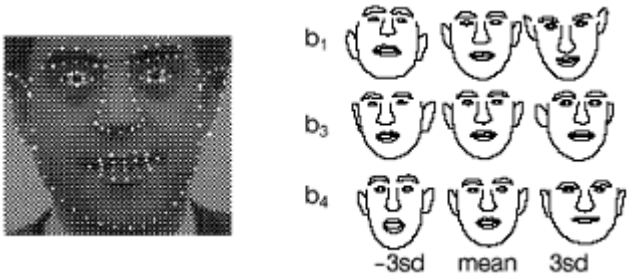
Template matching Merupakan proses pencarian lokasi sebuah bagian gambar dalam suatu gambar, yang paling mirip dengan suatu *template* dan berukuran berukuran sama. Teknik korelasi merupakan teknik pencocokan tradisional berdasarkan pada hubungan antara area kandidat wajah dari citra input dengan citra acuan (template). Jika template dan area kandidat wajah tersebut identik maka koefisien kolerasinya bernilai "1". Jika tidak terdapat kolerasi atau hubungan maka koefisien kolerasi sama dengan "0".

Pendekatan pencocokan template (*template matching*) menyajikan informasi sebuah wajah secara menyeluruh dalam sifat 2-D. *Template* wajah disimpan, juga template dari ciri-ciri wajah (bentuk kepala, hidung, mulut). Proses ekstraksi ciri dilakukan dengan mencocokan (mengkorelasikan) piksel-piksel dari citra wajah dengan template yang ada pada sebelumnya (*predefined template*) atau menggunakan fungsi dengan parameter tertentu. Gambar 2.3 menunjukkan bagaimana Scassellati mengadaptasi template wajah dengan membagi wajah menjadi 16 region dan 23 relasi.



Gambar 2.3 Segmentasi Wajah Menurut Scassellati

Lanitis menerapkan pendekatan lain yaitu dengan membuat aneka model *template* dari sebuah wajah. Gambar 2.4 menunjukkan ilustrasi metode yang dilakukan. Teknik pendekatan yang lain adalah *Hierarchical Template Matching*. Pendekatan ini diusulkan oleh Miao untuk menangani posisi wajah dalam berbagai sudut dan skala. Selain pendekatan di atas, dengan asumsi bahwa *template* sudah ada terlebih dahulu, dilakukan juga penelitian lain dengan asumsi *template* bisa berubah-ubah (*deformable templates*). Salah satu tekniknya dikembangkan oleh Yuille dimana ciri-ciri wajah dideskripsikan dengan parameter *template*.



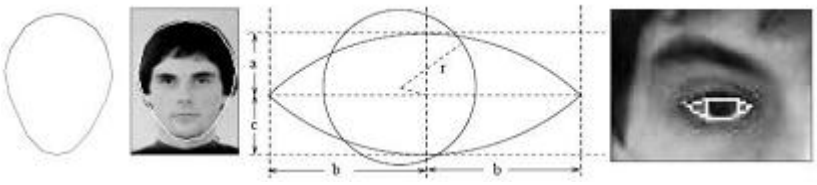
Gambar 2.4 model template dan variasinya

2.5.1 Analisis Template Matching

Teknik pendeteksian berbasiskan *template* (*Template-based Methods* - TBM) menggunakan *template* wajah (dalam posisi frontal pada umumnya) sebagai pola baku yang didefinisikan, baik secara manual (*predefined template*) maupun dibentuk menggunakan fungsi dengan parameter-parameter tertentu. Nilai korelasi antara citra masukan dan *template* dihitung berdasarkan komponen tertentu secara terpisah, misalnya kontur wajah, mata, hidung dan mulut. Keberadaan wajah ditentukan berdasarkan nilai dari korelasi tersebut. Pendekatan

ini memiliki keuntungan yaitu sederhana untuk diimplementasikan, namun kurang memadai dan terbatas mengingat variabilitas dari wajah dalam hal ukuran, pose dan bentuk. Usulan-usulan telah disampaikan melalui literatur dalam rangka mengurangi keterbatasan pendekatan ini antara lain melalui pendekatan multiresolusi, multiskala, bagian pola tiru (*subtemplate*) dan pola tiru yang dapat berubah (*deformable template*).

Teknik pencocokan template yang menggunakan template yang didefinisikan terlebih dahulu dapat diwakili oleh penelitian Craw dkk. (1992) yang menggunakan *template* bentuk (*shape*) yaitu bentuk wajah yang menyerupai ellips dan bentuk fitur wajah yaitu mata dan bolamata. Filter Sobel digunakan untuk mengekstraksi tepi-tepi dari citra, kemudian hasilnya dibandingkan dengan *template* yang sudah ditentukan sebelumnya. Langkah ini dilakukan untuk menentukan kontur wajah secara keseluruhan. Proses yang sama dilakukan dalam ukuran yang berbeda untuk melokalisasikan fitur-fitur seperti mata, alis dan bibir. Gambar 3.1 Menunjukkan template secara keseluruhan dan template untuk fitur sebuah mata.



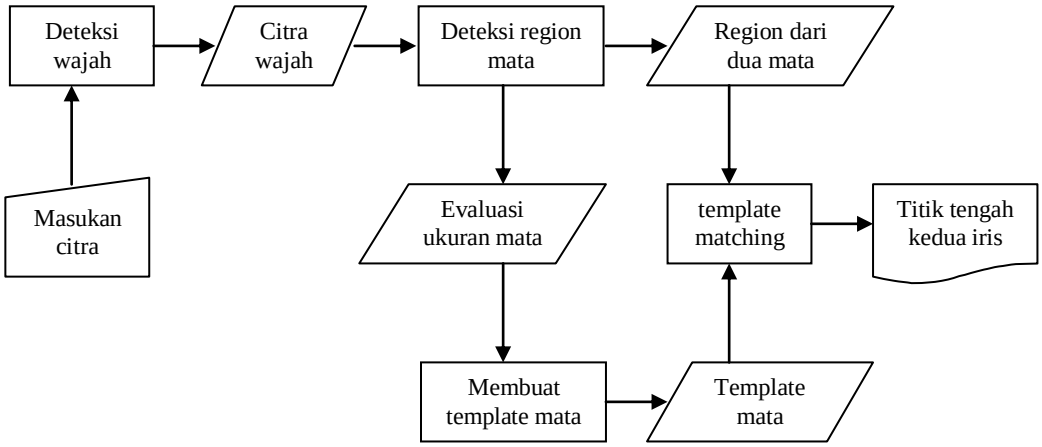
Gambar 2.5 Template Wajah Keseluruhan dan Template Fitur Mata

Kelebihan pendekatan dengan cara pencocokan *template* adalah Mudah dan sederhana karena berupa perbandingan biasa. Sedangkan kekurangannya adalah *template* harus dibuat serupa mungkin dengan wajah yang diinginkan,

Hampir tidak mungkin membuat banyak *template*, dengan pose berbeda -beda untuk setiap wajah.

2.5.1.1 Arsitektur

Banyak metode atau pendekatan yang begitu menjanjikan dengan kriteria dan keunggulan tertentu, yang digunakan untuk mendeteksi wajah. Kita mengasumsikan posisi atau lokasi wajah secara garis besar telah ditentukan sebelumnya, dengan ketentuan bahwa citra hanya memiliki satu wajah, dan mata yang terdapat da dalam wajah itu dapat dilihat tanpa menggunakan kaca mata atau sejenisnya yang melapisi mata.



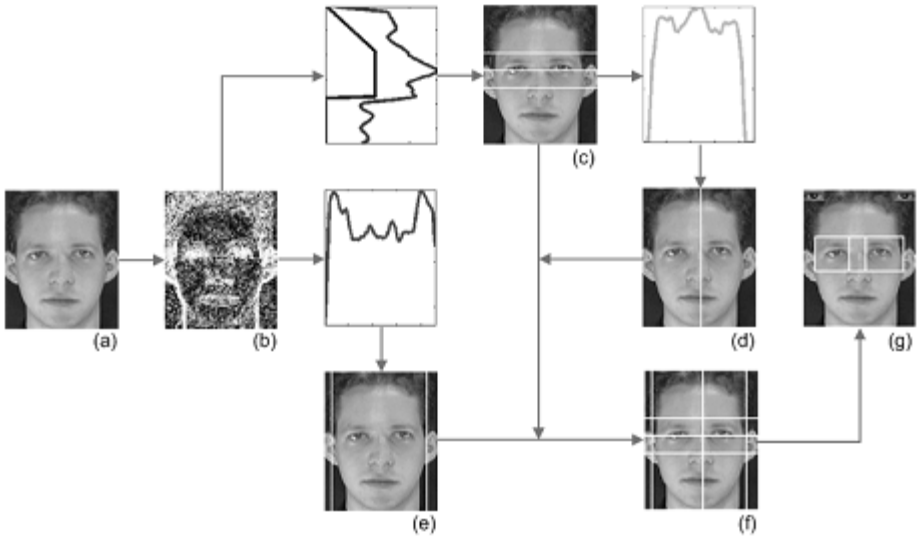
Gambar 2.6 proses deteksi mata

Ketika wajah atau citra yang mengandung wajah (*single face*) hadir di dalam sistem, maka pendeteksian wajah pertama-tama akan melakukan penentuan lokasi garis besar atau *region face* yang terdapat pada citra tersebut. Langkah kedua, dengan menggunakan metode segmentasi atau ekstrasi fitur, menentukan lokasi kedua mata secara garis besar yang terdapat di dalam wajah pada citra.

Pada waktu yang sama, dengan memanfaatkan informasi lokasi kedua mata tersebut, sistem mengevalusai ukuran kedua mata dan menentukan *template* yang digunakan dengan estimasi ukuran kedua mata yang diperoleh. Pada akhirnya sistem menentukan posisi pusat kedua selaput mata yang telah disesuaikan dengan *template* yang di peroleh yang telah diaplikasikan terhadap kedua region mata ini.

2.5.1.2 Pendeteksian Region-region Mata

Ketika region wajah telah ditentukan secara garis besar maka proses selanjutnya adalah menentukan posisi atau lokasi kedua mata. Adapun prosesnya dapat dilihat pada gambar 2.7 berikut:



Gambar 2.7 Pendeteksian Region-region Mata

Langkah pertama adalah mengalkulasi gradient citra (b) berdasarkan region wajah citra (a). Kemudian mengaplikasikan proyeksi horizontal terhadap gradient citra tersebut. Asumsi yang telah nyata adalah bahwa posisi mata terdapat pada bagian atas region wajah dan nilai piksel-piksel yang terdapat di sekitar lebih mudah berubah dibandingkan keadaan pikel-pikel bagian yang lain pada wajah.

Secara jelas tampak bahwa puncak proyeksi horizontal pada bagian atas dapat memberikan posisi horizontal mata. Berdasarkan posisi horizontal mata. Berdasarkan posisi horizontal dan total tinggi wajah, kita dapat menandakan region lokasi mata (c) secara horizontal.

Selanjutnya kita melakukan proyeksi vertikal terhadap keseluruhan piksel yang terdapat pada batas-batas horizontal di dalam region mata yang telah ditentukan sebagaimana citra (c), sehingga diperoleh posisi puncak proyeksi horizontal yang berada di dekat pusat vertikal citra wajah (d). Posisi puncak vertical ini dapat diasumsikan sebagai posisi vertical pusat wajah, karena area diantara kedua mata tersebut lebih cerah di dalam region horizontal.

Pada waktu yang sama, proyeksi vertical dapat dilakukan terhadap citra gradient (e), terdapat dua puncak pemroyeksian di dekat batas-batas citra wajah pada sebelah kiri dan sebelah kanan citra yang berkaitan sebagai batas kiri dan batas kanan citra. Dari batas kiri dan batas kanan ini maka dengan mudah kita mengestimasi lebar wajah di dalam citra tersebut.

Dengan mengombinasikan semua citra yang dihasilkan (c), (d) dan (e) maka diperoleh suatu citra yang telah disegmentasikan sebagaimana citra (f) dan perkiraan atau estimasi lebar wajah, serta kita dapat menandakan pula kedua region mata yang terdapat pada wajah di dalam citra wajah tunggal.

2.5.1.3 Penentuan Template Mata

Setelah kedua region mata berhasil ditentukan maka proses selanjutnya adalah menentukan *template* yang sesuai yang akan digunakan untuk mengalokasikan posisi pusat selaput mata sebelumnya. Karena region yang

diperoleh akan memiliki area yang lebih kecil dari keseluruhan area wajah yang ada, maka diperlukan suatu algoritma yang efisien.

Template untuk mata dapat diperoleh atau ditentukan dari citra wajah nyata, tetapi tidak dapat digunakan secara langsung, karena ukuran mata pada citra nyata tidak sama dengan ukuran *template* mata. Penyelesaian sederhana untuk permasalahan ini adalah dengan melakukan proses penyesuaian atau *matching* beberapa kali, dimana pada tiap-tiap waktu digunakan *template* dengan ukuran yang berbeda. Hal ini bukan cara yang begitu efektif, terlalu boros dalam pewaktuan diantaranya.

Berkaitan dengan algoritma yang ada, dan di dalam tatanan untuk peningkatan efisiensi, maka unuran mata akan diestimasi secara otomatis. Oleh karena itu proses *matching* atau penyesuaian akan diunjuk kerjakan hanya sekali. Dengan tahapan proses yang sudah dijelaskan diatas maka citra akhir (g) pada gambar 2.7 menunjukkan *template-templete* yang digunakan untuk kedua mata yang didasari oleh pengestimasian ukuran kedua mata tersebut.

2.5.1.4 Pelokasian Pusat Selaput

Kita misalkan, *template* yang telah dimiliki adalah g[i,j] dan template pemisalan yang akan dideteksi di dalam suatu citra adalah f[I,j]. maka hal pertama yang akan kita lakukan adalah menempatkan *template* pada suatu lokasi di dalam citra dan mendeteksi keberadaan titik-titik yang dimiliki dengan membandingkan nilai-nilai intensitas di dalam *template* dengan nilai-nilai terkait di dalam citra. Di karenakan nilai-nilai intensitas tersebut kecil sekali kemungkinan untuk sesuai secara tepat dengan nilai-nilai di dalam citra maka kita membutuhkan suatu

pengukuran ketidaksamaan antara nilai-nilai intensitas pada *template* dengan nilai-nilai terkait yang terdapat pada citra. Beberapa pengukuran didefinisikan sebagai berikut :

$$\max_{[i,j] \in R} | f - g |$$

$$\sum_{[i,j] \in R} | f - g | \qquad (2.1)$$

$$\sum_{[i,j] \in R} | f - g |^2$$

Dimana R merupakan region dari *template*. Jumlah kuadrat error merupakan suatu pengukuran yang umum digunakan. Di dalam kasus penyetaraan *template,* perhitungan tersebut dapat digunakan secara tidak langsung dengan biaya perhitungan yang minimasi. Kita dapat menyederhanakan:

$$\sum_{[i,j] \in R} (f - g)^2 = \sum_{[i,j] \in R} f^2 + \sum_{[i,j] \in R} g^2 - 2 \sum_{[i,j] \in R} f.g \qquad (2.2)$$

Sekarang kita asumsikan bahwa f dan g adalah tetap, maka $\sum fg$ akan memberikan suatu ukuran yang tidak setara. Trik atau strategi yang beralasan untuk memperoleh keseluruhan lokasi dan pemisalan *template* adalah dengan menggeser *template* awal dan menggunakan pengukuran untuk penyesuaian terhadap seluruh titik pada citra. Oleh karena itu, untuk template m x n, kita hitung:

$$M[i, j] = \sum_{k=1}^{m} \sum_{l=1}^{n} g[k,l] f[i+k, j+l] \quad (2.3)$$

dimana k dan 1 merupakan suatu *displacemen* yang cenderung terhadap *template* di dalam citra. operasi ini juga disebut *cross-correlation* antara f dan g.

Tujuan kita adalah menemukan suatu titik yang disebut *maxima local* atau pada batas atas tertentu. telah diketahui sebelumnya dengan mengasumsikan f dan g mempunyai suatu nilai konstan maka akan terdapat sedikit permasalahan dalam hal ketidaksetaraan ukuran. seandainya komputasi tersebut diaplikasikan pada suatu citra, dengan ketentuan yang diubah sedikit, dimana *template* g adalah konstan, dan nilai f bervariasi, maka nilai-nilai M akan sangat bergantung pada nilai f, dan hal ini tidak akan memberikan kita suatu indikasi penyesuaian yang benar pada lokasi-lokasi yang berbeda. untuk mengatasi hal ini kita perlu menormalisasi *cross-correlation* diatas, dan M dapat dikalkulasikan dengan:

$$C_{f.g}[i, j] = \sum_{k=1}^{m} \sum_{l=1}^{n} g[k,l] f[i+k, j+l] \quad (2.4)$$

$$M[i, j] = \frac{C_{f.g}[i, j]}{\left\{ \sum_{k=1}^{m} \sum_{l=1}^{n} f^{2}[i+k, j+l] \right\}^{1/2}} \quad (2.5)$$

Pada metode *template matching*, suatu pola wajah yang standar (pada umumnya frontal) secara manual sudah dikenal atau dinotasikan dalam parameter oleh suatu fungsi. Misal suatu citra input, nilai-nilai korelasi dengan pola

standar dihitung untuk sekeliling wajah, mata, hidung dan mulut dengan bebas. Keberadaan suatu wajah ditentukan berdasarkan pada nilai-nilai korelasi. Pendekatan ini sederhana untuk diterapkan. Gambar 2.8 dibawah menunjukkan contoh penggunaan *template matching* untuk melokasikan pusat selaput.



Gambar 2.8 Template Matching

Dibagian kiri pada baris pertama ditampilkan dua *template* mata yang diperoleh dari pengestimasian ukuran wajah. baris kedua adalah region-region yang diperoleh setelah *template matching* dilakukan. Hasil setelah diterapkannya *cross-correlation* berada pada baris ketiga, dan citra yang utuh menunjukkan hasil berupa posisi pusat mata.

2.6 Metode EigenEye

Eigeneye merupakan suatu metode untuk mengambil ciri-ciri penting dari sekumpulan data dengan melakukan dekomposisi terhadap data tersebut sehingga menghasilkan koefisien-koefisien yang tidak saling berkorelasi. *Eigeneye* juga dikenal dengan nama transformasi Karhunen-Loeve atau transformasi *Hotelling* atau teknik PCA. Tujuan dari *Eigeneye* adalah menangkap variasi total pada citra mata yang dilatihkan dan menjelaskan variasi tersebut dengan variabel-variabel yang jumlahnya lebih sedikit. Dengan kata lain

Eigeneye digunakan untuk merepresentasi data dalam dimensi rendah sehingga waktu komputasi dapat dikurangi dan kompleksitas dari mata yang tidak perlu dapat dihilangkan. *Eigeneye* menghasilkan vektor-vektor eigen atau vektor-vektor karakteristik yang nantinya akan digunakan untuk membentuk ruang eigen.

Dalam mereduksi dimensi data menjadi lebih rendah maka diperlukan penentuan vektor-vektor eigen yang dapat direduksi maupun vektor-vektor eigen yang tidak dapat direduksi dengan mengurutkan vektor eigen yang bernilai eigen terbesar sampai yang terkecil. Vektor-vektor eigen yang dapat direduksi adalah vektor yang mempunyai nilai eigen yang kecil. Hal ini dikarenakan nilai eigen yang kecil menandakan informasi yang dibawa oleh vektor eigen tersebut tidak terlalu penting sehingga dapat direduksi tanpa mempengaruhi informasi penting wajah tersebut.

2.6.1 Analisis EigenEye

Pendekatan *Eigenface* diterima menjadi metode yang mendekati *real-time* karena kecepatannya.. Dengan penggunaan tambahan informasi dari *face tracker* kita dapat menggunakan metode *eigeneye* dengan hanya satu database. Mempunyai *property* dari deteksi mata tanpa masalah skala. Sebelum mata melakukan pencarian diluar area wajah, database mata akan mengukur kembali dengan faktor skala yang di hitung berdasarkan dibawah ini :

$$scalefacto\ \ r = \frac{facewidth\ \ \times ideal\ \ \frac{eyewidth}{facewidth}\ ratio}{EyeWidthof\ \ \ TheDatabas\ \ \ e} \qquad (2.6)$$

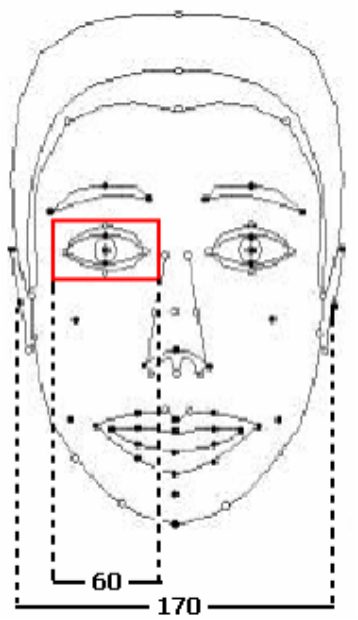
Keterangan :

Scale factor : berapa banyak lebar dan tinggi dari gambar mata di dalam database.

Face width : lebar dari wajah yang dilokasikan.

Ideal (eye width/face height) rasio : rasio lebar mata terhadap total lebar wajah yaitu ˜0,35.

Eye width of the database : panjang mata yang terdapat di dalam database.



Gambar 2.9 Model Wajah yang digunakan dalam Deteksi Mata

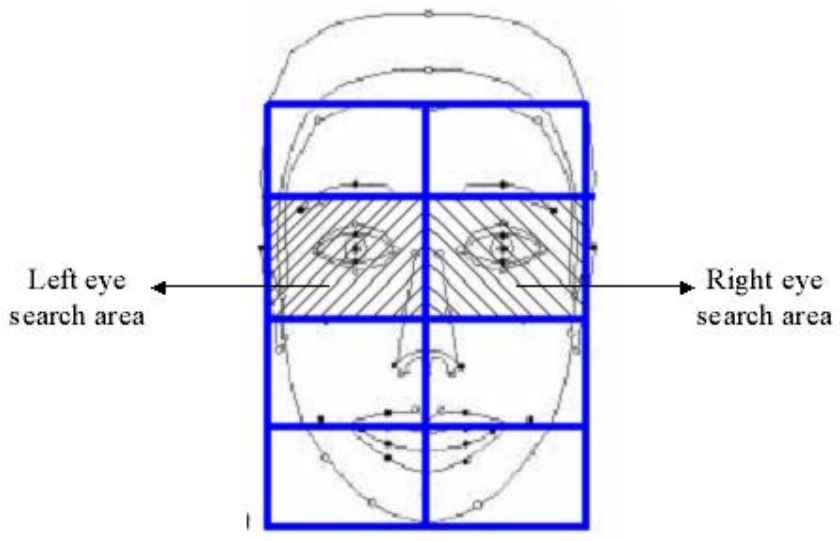
Pada gambar 2.9 mata dalam database diukur kembali menggunakan faktor skala menggunakan interpolasi. Metode interpolasi antara nilai piksel di sekitarnya dengan contoh statistik dari intensitas nilai lokal.

newWidth = scale factor X database eye width (2.7)

newHeight = scale factor X database eye height

deteksi mata menggunakan eigeneyes Rincian dari masing-masing langkahnya adalah sebagai berikut :

1. Untuk metode eigeneye, array gambar 2D harus di representasikan sebagai ID data set. *Image* di dalam gambar database adalah *image* berwarna, jadi untuk langkah pertama, kita harus di ubah dulu ke dalam form 2D. Dalam menggambarkan image berwarna sebagai image *grayscale* untuk metode *eigeneye*, kita dapat mencoba komponen yang berbeda dari *color space* seperti CIE XYZ, YCbCr, HSV, CIE Lab ini akan memperoleh hasil yang lebih baik dari pada menggunakan *grayscale*.
2. Setelah citra mata sudah di format menjadi gambar array 2D, citra di ukur ulang.
3. citra rata-rata, diperoleh menggunakan form ID dari citra mata, yang mana dinormalisasi menjadi nol dengan menggerakkan komponen DC, digunakan untuk menyusun matrik *covariance*. Kemudian *eigenvector* dihitung dan *eigenvector* dengan *eigenvalues* yang tinggi dipilih ke dalam bentuk U *eigenvector* matriks U. pemilihan angka yang tepat dari citra yang dicoba dan *eigenvector* merupakan *point critical* dalam deteksi *eigeneye*.
4. Calon area mata akan dimasukkan ke dalam *eigendecomposition*, untuk penyimpanan dengan menggunakan model *geometrical*. Pencarian area untuk kiri dan kanan mata dibatasi, dapat kita lihat pada gambar 2.10:



Gambar 2.10 area pencarian untuk kiri dan kanan mata

$$leftEyeX_{min} = 0 \; and \; leftEyeX_{max} = \frac{faceAreaWi \quad dth}{2}$$

$$leftEyeY_{min} = \frac{faceAreaHe \quad ight}{2} \; andleftEye \quad Y_{max} = \frac{faceAreaHe \quad ight \times 3}{4}$$

$$rightEyeX_{min} = \frac{faceAreaWi \quad dth}{2} \; andrightEy \quad eX_{max} = faceAreaWi \quad dth$$

$$rightEyeY_{min} = \frac{faceAreaHe \quad ight}{2} \; andrightEy \quad eY_{max} = \frac{faceAreaHe \quad ight \times 3}{4} \qquad (2.8)$$

5. Selama fase pembangunan dari sistem area pencarian mata, kadang-kadang ada area sederhana yang terdeteksi sebagai lokasi mata. Dikarenakan wilayah kulit mempunyai nilai variasi yang rendah dimana terdapat area dengan gradien nilai variasi mata yang lebih tinggi, Sebaiknya kita memeriksa area *variance* terlebih dahulu sebelum pendeteksian *eigeneye* dilakukan. Hal ini untuk mengurangi jumlah kesalahan pendeteksian. Masing-masing calon area dengan nilai *variance* yang tinggi dikandidatkan ke dalam ruang mata pada

database mata. Y_i adalah data ke kesekian, $\hat{Y}$ adalah rata-rata dan n adalah jumlah data.. Berikut ini adalah persamaan untuk menghitung nilai variance :

$$s = \frac{\sum_{i=1}^{n} (Y_i - \hat{Y})^2}{(n-1)} \quad (2.9)$$

Setelah nilai variance didapat maka selanjutnya nilai tersebut akan di cek untuk pendefinisan citra dalam area pencarian sebagai kandidat mata yang akhirnya koordinat mata akan didapat.

2.6.2 Pencarian Nilai Eigen

Persamaan *eigen* dapat ditulis sebagai berikut:

$$\mathbf{Ax - \lambda x = o} \quad (210)$$

Dengan mengingat, bahwa A berordo nxn dan **x** berordo nx1, maka dengan mengalikan dengan matrik identitas I yang berordo nxn, maka persamaan di atas dapat ditulis, sebagai:

$$\mathbf{Ax - \lambda Ix = o} \quad (2.11)$$

Atau

$$\mathbf{(A - \lambda I)x = o,} \quad (2.12)$$

dengan mengingat vektor eigen $\mathbf{x} \neq \mathbf{o}$, maka

$$\mathbf{det(A - \lambda I) = 0} \quad (2.13)$$

Persamaan $\det(A - \lambda I) = 0$ dikenal sebagai persamaan karakteristik.

2.7 Algoritma Camshift

Pada penelitian ini digunakan pendekatan algoritma yang berbasiskan warna obyek, yaitu algoritma *CamShift* yang diambil dari *Open Source Computer Vision Library*.

CamShift singkatan dari *Continuously Adaptive Mean-Shift*. Algoritma *CamShift* merupakan pengembangan dari algoritma dasar *Mean-Shift* yang dilakukan secara berulang untuk dapat melacak pergerakan dari obyek. Algoritma *Mean-Shift* beroperasi berdasarkan distribusi probabilitas dari citra. Untuk melacak citra berwarna pada *video sequence*, maka citra berwarna tersebut harus direpresentasikan dalam bentuk distribusi probabilitas citra dengan menggunakan distribusi histogram dari citra tersebut. Distribusi warna dari *video sequence* selalu berubah tiap waktu, maka algoritma *Mean-Shift* harus dikembangkan supaya dapat digunakan untuk melacak obyek. Perkembangan dari algoritma ini dinamakan algoritma *CamShift*.

Tahap pertama dari algoritma *CamShift* adalah dengan melakukan penghitungan terhadap nilai *hue* dari obyek sampel warna kulit manusia. Nilai histogram hue sebagai hasil perhitungan tersebut selanjutnya akan disimpan untuk digunakan sebagai *lookup table*.

Lookup table histogram tersebut dapat digunakan untuk semua orang dari berbagai ras (kecuali *albino*). Ada pendapat yang menyatakan bahwa jika citra model dengan citra yang akan dibandingkan berbeda ras warna, maka diperlukan *lookup table* histogram yang baru, misalnya: citra model berkulit putih sedangkan citra yang akan dibandingkan berkulit hitam. Pendapat tersebut sama sekali salah sebab meskipun warna kulit berbeda, nilai *hue*-nya tetap sama. Perbedaan warna ras yang ada disebabkan karena perbedaan *saturation*. Karena *lookup table* histogram menggunakan nilai *hue*, maka tidak diperlukan *lookup table* baru jika citra yang akan dibandingkan mempunyai warna yang berbeda.

Tahap kedua, dilakukan pemilihan lokasi awal dari *Mean-Shift* 2D *search window*. Kemudian dilakukan penghitungan nilai *histogram hue* yang menjadi nilai probabilitas untuk tiap-tiap *pixel* pada citra

Tahap ketiga, dijalankan algoritma *Mean-Shift* untuk mencari pusat dan besar *search window* yang baru. Lokasi tengah dan besar daerah yang dihasilkan disimpan. Lalu dilakukan *looping* ke tahap dua dimana lokasi awal dan *search window* menggunakan hasil algoritma *Mean-Shift* pada tahap tiga.

Tahap algoritma *Mean-Shift* dapat dijabarkan sebagai berikut:

1. Tentukan ukuran dari *search window.*
2. Tentukan lokasi awal dari *search window.*
3. Hitung lokasi *mean* dalam *search window.*
4. Pusatkan *search window* pada lokasi mean yang sudah dihitung dengan langkah ke-3.
5. Ulangi langkah ke-3 dan langkah ke-4 sampai *search window* konvergen, yaitu sampai *window* tersebut berpindah dengan jarak yang kurang dari nilai ambang yang sudah ditentukan (*preset threshold*)

Untuk mencari titik pusat (*moment*) dan besar daerah *search window* pada alogritma *Mean Shift*, dilakukan beberapa perhitungan.

1. Mencari titik pusat

$$M_{00} = \sum_x \sum_y I(x, y) \tag{2.14}$$

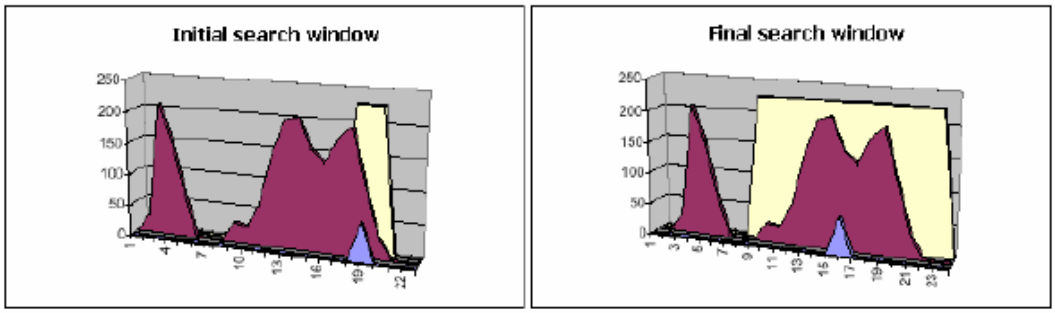
2. Mencari titik pertama untuk x dan y

$$M_{10} = \sum_x \sum_y xI(x, y); \qquad M_{01} = \sum_x \sum_y yI(x, y) \tag{2.15}$$

3. Kemudian lokasi *mean* dalam *search window.*

$$X_c = \frac{M_{10}}{M_{00}} and \qquad Y_c = \frac{M_{10}}{M_{00}} \tag{2.16}$$

Dimana *I(x,y)* adalah piksel (probabilitas) *value* pada posisi *(x,y)* dalam citra dan x dan y adalah jarak keseluruhan *search window.*



Gambar 2.11 Camshift Search Window

2.8 Model Warna (Color Space)

Model warna adalah metode untuk menjelaskan properti atau perilaku warna didalam beberapa konteks tertentu. Tidak ada warna tunggal yang dapat menjelaskan semua aspek warna, sehingga dibuat model yang berbeda untuk mendeskripsikan perbedaan karakteristik dari warna. Sumber cahaya dideskripsikan dengan istilah frekuensi domain (*hue*), *luminance* (*brighthness*) dan *purity* (saturasi). Istilah chromaticity digunakan untuk mengacu pada dua properti yang mendefinisikan karakteristik warna yaitu purity dan frekuensi yang dominan. Pada tugas akhir ini, untuk pendeteksian wajah terlebih dahulu dilakukan pemisahan skin region dengan non skin region

dengan mengacu pada model warna kulit. Dari dua komponen yang direpresentasikan oleh RGB *color space*, hanya nilai *chrominance* sebuah citra berwarna yang diambil, karena nilai *luminance* tidak penting dalam pemisahan *skin region* dan *non skin region*, sehingga pada kondisi pencahayaan normal tidak perlu dimasukkan dalam representasi warna kulit. Warna kulit dapat ditampilkan dalam *chromatic color space*. Warna chromatic didefinisikan sebagai berikut :

$$r = \frac{R}{(R + G + B)} \qquad b = \frac{B}{(R + G + B)}$$

Warna hijau tidak perlu dinormalisasi karena r + g + b = 1 . Selain itu, mata manusia kurang sensitif terhadap warna hijau sehingga komponen warna hijau tidak terlalu berpengaruh. Distribusi warna wajah dapat direpresentasikan dengan Model gauss N (m,C) :

$m = E\{x\}$

dengan $x = (r\ b)^T$

dan $\bar{r} = \frac{1}{N} \sum_{i=1}^{N} r_i$

$\bar{b} = \frac{1}{N} \sum_{i=1}^{N} b_i$

serta $C = E\{(x-m)(x-m)^T\}$

dimana, m : rata-rata nilai N buah komponen r dan b

r : rata-rata nilai N buah komponen r

b : rata-rata nilai N buah komponen

C : kovariansi komponen-komponen r dan b.

Parameter-parameter tersebut dapat digunakan untuk menentukan tingkat kemiripan setiap piksel dalam suatu gambar, dengan hubungan :

$P(r,b) = \exp[-0.5 (x-m)^T C^{-1} (x-m)]$

Sehingga dengan mentransformasi komposisi nilai r dan b dari setiap piksel dalam suatu gambar berwarna dengan hubungan diatas, diperoleh gambar yang setiap pikselnya menunjukkan tingkat kemiripan setiap piksel dengan warna kulit, dimana dengan proses ini akan menghasilkan gambar *skin likelihood.*

2.8.1 Gray Color Space

Representasi citra dengan hanya menggunakan satu warna abu-abu (*gray*) yang berbeda intensitasnya. Citra abu-abu dapat dihasilkan dari citra warna RGB dengan mengalikan ketiga komponen warna RGB dengan suatu koefisien yang jumlahnya satu.

$Y = a \times R + b \times G + c \times B$

dimana :

Y = citra abu-abu (*grayscale image*)

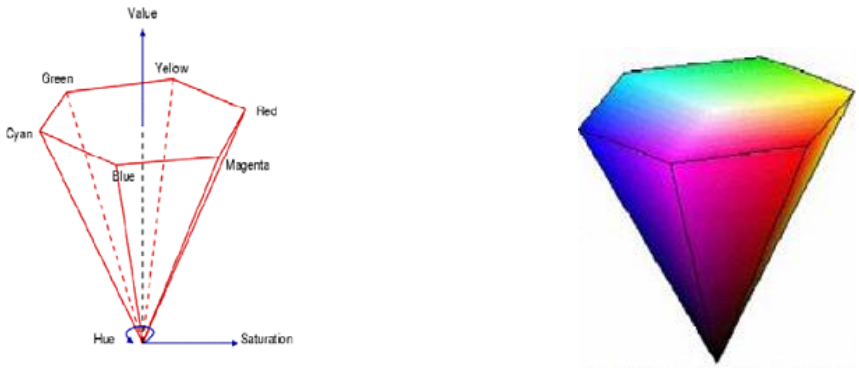
a,b,c merupakan variable koefisien

dengan nilai $a + b + c = 1$

Dalam pengolahan citra, *grayscale image* biasanya banyak digunakan. Hal ini disebabkan karena *grayscale image* memiliki sturuktur yang lebih sederhana daripada citra warna RGB sehingga untuk komputasi dan pengolahan citra dapat lebih cepat dan efisien. Intensitas *grayscale* biasanya disimpan sebagai data citra 8 bit atau 256 intensitas warna *gray* dari nilai 0 (hitam) dan 255 (putih).

2.8.2 HSV Color Space

Model warna HSV merupakan hasil penggabungan dari *Hue, Saturation* dan *Value*. Hue (corak warna) merupakan sensasi pengelihatan manusia berdasarkan pada kemiripan suatu daerah tampak seperti daerah yang lain sesuai dengan warna yang diterimanya, merah, kuning, hijau dan biru atau kombiinasi keduanya. *Saturation* adalah kekayaan warna pada suatu daerah sesuai dengan proporsi gelap-terangnya. Kita bias menemukan warna biru langit sampai biru tua dengan mengubah nilai dari komponen saturation ini. Sedangkan *Value* merupakan persepsi suatu daerah warna yang tampak ketika menerima sedikit atau banyak cahaya dengan referensi warna putih. Dari gambar 2.6 dapat divisualisasikan model warna HSV



Gambar 2.12 Model Warna HSV

2.8.3 XYZ Color Space

Diciptakan oleh *Commission Internationale de l'Eclairage* (CIE) pada tahun 1931, CIE XYZ merupakan salah satu standar yang cukup terkenal, tetapi sudah kuno. Kelemahan *color space* ini adalah sulitnya untuk mengatur *brightness* (Gareth Loy:2002). Untuk mempresentasikan warna, standar ini menggunakan kombinasi penamhahan nila X, Y, Z. Ketiga nilai ini selalu bernilai positif, dan diubah menjadi nilai RGB melalui matriks transformasi.

X	0.41211	0.357585	0.1810454	R
Y=	0.212649	0.715169	0.072182	xG
Z	0.019332	0.119195	0.950390	B

Menurut eksperimen yang sudah dilakukan diketahui bahwa menggunakan Z komponen dari XYZ *color space* adalah untuk suatu pendeteksian gambar wajah, *eigenvalue* untuk pendeteksian mata. Ini memberi hasil yang lebih baik sebagai pengganti Versi *grayscale*.

2.8.4 Lab Color Space

Lab, disebut juga seperti CIELAB, adalah model warna yang lengkap dengan menggunakan konvensional, untuk menguraikan semua warna dan dapat terlihat oleh mata manusia. Tiga parameter di dalam model adalah L adalah *luminancy*, a adalah suatu posisi antar hijau dan merah, b adalah posisi antar biru dan kuning. Rumusan Konversi dari RGB *color space* ke lab *color space* adalah:

X	0.43310	0.376220	0.189860	R / 255
Y =	0.212649	0.715169	0.072182 x	G / 255

Z	0.017756	0.109478	0.872915	B / 255

$L = 116xY^{1/3}$ *for* $Y > 0.008856$

$L = 903.3xY$ *for* $Y <= 0.008856$

$a = 500x(f(x)- f(y))$

$a = 200x(f(x)- f(z))$

Dimana

$f(t) = t^{1/3}$ *for* $t > 0.008856$

$f(t) = 7.787x\ t + 16/116$ *for* $t > 0.008856$

2.8.5 YCBRC Color Space

YCbCr *color space* adalah dasar pada *luminance* dan *chrominance*. diperoleh dari RGB representasi menggunakan :

Y	0.299	0.587	0,114	R
Cb=	-0.169	0,331	0,5	*G
Cr	0,5	-0,149	-0,081	B

Dimana Y adalah *luminance* atau komponen *brigthness* dan Cb dan Cr adalah komponen dari biru dan merah berturut-turut.

2.9 Image Thresholding

Unsur yang paling esensial dalam sistem deteksi gerakan adalah pemisahan antara objek bergerak dengan latar belakang. Kinerja suatu sistem deteksi gerakan ikatakan baik apabila sistem ini mampu benar-benar memisahkan antara objek bergerak dan latar belakangnya.

Cara yang paling tepat dalam memisahkan obyek dari latar belakangnya adalah menentukan nilai threshold T[3] . Untuk sembarang piksel (x,y) dimana f(x,y) > T disebut sebagai objek, sedangkan untuk piksel bernilaif(x,y) T disebut sebagai latar belakang. Dengan f(x,y) merupakan fungsi nilai selisih piksel absolut, sehingga persamaannya adalah:

$g(x,y)$ = 1 jika $f(x,y)>T$

0 jika $f(x,y)<=T$

dengan,

$f(x,y) = |I_1(x,y)-I_2(x,y)$

dimana,

I_1 = citra ke-1

I_2 = citra ke-2

Secara keseluruhan nilai *bilevel*s menunjukan hasil yang dapat di klasifikasikan menjadi hitam(0) dan putih(1). Adanya kekasaran (*noise)* dan efek cahaya (*illumination)* akan mempengaruhi nilai yang dihasilkan dari pemrosesan.



Gambar 2.13 Contoh Proses Image Thresholding

2.10 Active Contour Models (Snakes)

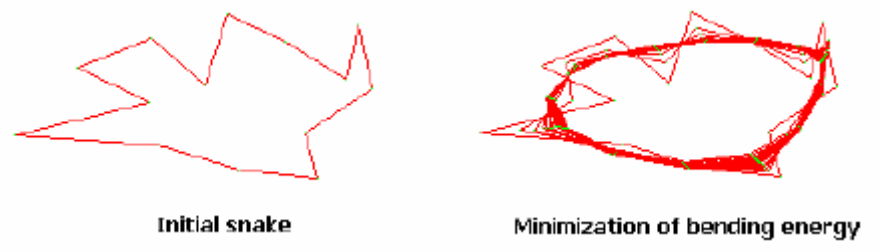
Sejak *Active Contour Model* (*Snakes*) muncul, sejak itu telah dengan sukses menerapkan berbagai solusi permasalahan dalam *computer vision* dan analisa gambar. Seperti *edge,* dan pendeteksian *subjective contours*, *motion tracking and segmentasi*. Suatu *active contour* adalah suatu energi untuk memperkecil *spline*. Fungsi Energi suatu *snakes* melibatkan terminologi untuk memperindah *snakes* pada corak yang diinginkan dalam sebuah gambar, *snakes* mengubah bentuk secara terus-menerus untuk memperkecil fungsi energi itu, sehingga dapat dianggap sebagai suatu kasus yang khusus dari suatu teknik umum untuk *matching*, yaitu mengubah bentuk model bagi suatu gambar.

Struktur snakes yang sederhana terdiri dari satu set *point control* yang dihubungkan oleh garis lurus dan membentuk *form* pengulangan keluar. Masing-Masing control point mempunyai suatu posisi yang diberi oleh (X,Y) yaitu suatu koordinat pada gambar, Penyesuaian *snakes* dibuat oleh gerakkan *control point* ini menurut fungsi energi *snakes*. fungsi untuk suatu snakes terdiri atas dua bagian, diantaranya energi internal dan energi eksternal.

$$E_{snakes} = E_{internal} + E_{external}$$

Energi Internal = Energi internal adalah bagian yang tergantung pada intrinsik properti dari *snake* seperti lengkungan atau panjang.

Energi Eksternal = Energi eksternal tergantung pada faktor seperti struktur gambar dan prosedur initialisasi .



Gambar 2.14 Minimasi bending energy

2.11 Pemrograman Berorientasi Objek

Pemrograman berorientasi objek (*object-oriented programming* disingkat OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Bandingkan dengan logika pemrograman terstruktur. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya.

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

2.11.1 Konsep dasar dari Pemrograman Berorientasi Objek

Pemrograman orientasi-objek menekankan konsep berikut :

1. kelas - kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *'class of dog'* adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada

berbagai macam perilaku/turunan dari anjing. Sebuah *class* adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi objek. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang non-programmer sekalipun terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah *class* sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP).

2. Objek - membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer; objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.
3. Abstraksi - Kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya, dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.
4. Enkapsulasi - Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak; hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya. Setiap objek mengakses interface yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

5. Polimorfisme melalui pengiriman pesan. Tidak bergantung kepada pemanggilan subrutin, bahasa orientasi objek dapat mengirim pesan; metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesa tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tungal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan, dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melalui penggunaan fungsi kelas-pertama.
6. Inheritas- Mengatur polimorfisme dan enkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis khusus dari objek yang sudah ada - objek-objek ini dapat membagi (dan memperluas) perilaku mereka tanpa haru mengimplementasi ulang perilaku tersebut (bahasa berbasis-objek tidak selalu memiliki inheritas.)
7. Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen

yang memiliki manager, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bagian administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bag administrasi untuk mengambilnya. Pada kasus tersebut seorang manager tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa mendapatkan data tersebut melalui objek petugas adminiistrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri.

2.12 .Net dengan Visual C++

.NET *Platform* merupakan satu set kumpulan teknologi yang memungkinkan teknologi *Internet* ditransformasikan ke dalam *platform distributed computing* dengan skalabilitas dan kompatibilitas tinggi. Secara teknikal, *.NET Platform* menyediakan konsep pemrograman dengan *library* dan modul-modul baru yang konsisten, terlepas dari jenis bahasa pemrograman yang digunakan.

.NET Platform menyediakan hal-hal berikut bagi para *developer* :

1) *Language independent*, dengan programming model yang konsisten di semua tier aplikasi yang dibangun.
2) *Interoperability* dan kompatibilitas antar aplikasi.
3) Kemudahan migrasi dari teknologi yang ada saat ini.
4) Dukungan penuh terhadap berbagai teknologi standar yang digunakan dalam *platform internet*, antara lain HTTP, XML, SOAP dan HTML.

Teknologi inti .NET secara umum terdiri dari 4 area pokok :

1. .NET Framework
2. .NET Building Block Services
3. Visual Studio .NET
4. .Net Enterprise Server

2.13 .NET Framework

.NET Framework adalah teknologi inti yang menyediakan berbagai *library* untuk digunakan oleh aplikasi di atasnya. Komponen inti *.NET Framework* adalah *Common Language Runtime* (CLR) yang menyediakan *run time environment* untuk aplikasi yang dibangun menggunakan *Visual Studio .NET*, terlepas dari jenis bahasa pemrogramannya.



Gambar 2.15 struktur .net frame work

Dengan adanya CLR tersebut, *programmer* dapat menikmati *consistent object model* dalam mengakses berbagai komponen *library*. Dengan demikian penggunaan bahasa pemrograman dalam dunia .NET adalah lebih ke masalah selera atau taste, dan bukan pada kelebihan maupun kekurangan masing-masing bahasa. Mengapa ? Karena semua bahasa pemrograman yang mensupport .NET mengakses *library* yang sama di dalam .NET Framework, dengan objek model

yang konsisten, dengan *run time file* yang sama. Pada VB6, tentu kita tahu bahwa diperlukan distribusi *run time library* khusus ketika menginstal aplikasi. Demikian pula ketika anda menginstal aplikasi yang dibangun dengan Visual C++ maupun Delphi. Dalam dunia .NET, hal tersebut sudah tidak diperlukan lagi, selama *.NET Framework* telah terinstal di komputer sasaran.

2.13.1 Visual Studio .NET

Visual Studio .NET menyediakan *tools* bagi para *developer* untuk membangun aplikasi yang berjalan di *.Net Framework*. *VS.Net* membawa perubahan besar dalam gaya pemrograman, karena setiap programmer dituntut untuk memahami .NET *object model* dan *Object Oriented Programming* dengan baik, jika tidak ingin menghasilkan aplikasi dengan performa rendah.

VS.Net juga semakin mempertipis jarak antara *Windows Programmer* dengan *Web Programmer*. Dunia *scripting* yang akrab bagi *programmer web* akan sulit ditemukan dalam .NET, karena pemrograman web sudah bersifat *full object oriented,* dengan fasilitas *event driven* programming sebagaimana layaknya *windows programming*. Pemrograman *web* menjadi lebih mudah dan menyenangkan bagi para *programmer windows,* sedangkan anda para veteran *scripting language* sudah saatnya untuk beralih ke *ASP.NET*, yang dapat diprogram menggunakan *VB, C#, C++* maupun *Phyton* dan *COBOL* sekalipun.

Microsoft masih menyediakan *Jscript.NET* bagi anda para pecandu *JavaScript* dan *JScript*, sedangkan versi *.Net* dari *VBScript* belum diketahui apakah akan disediakan atau tidak. Para veteran *VB.Script* disarankan untuk mempelajari *VB*, sehingga dapat menggunakan *VB.Net* untuk membangun aplikasi

web. Bahasa pemrograman yang terdapat di *VS.NET* adalah *VB.NET, C#, C++ .NET, J#,* dan *Jscript .NET*. Dalam masa mendatang akan terus ditambah berbagai bahasa pemrograman lain.

2.13.2 Visual C++

Visual C++ merupakan perangkat pengembangan aplikasi yang menggunakan *C++* sebagai bahasa pemrograman dan dapat digunakan untuk membuat aplikasi berbasis *Windows* maupun berbasis teks (aplikasi konsol). Perangkat ini menyediakan lingkungan yang dinamakan IDE (*Intergrated Development Environment*) yang memungkinkan pemakai membuat, mengkompilasi, menggabungkan, dan menguji program dengan cepat dan mudah. Beberapa komponen penting dalam *Visual C++* adalah sebagai berikut :

a. Editor

Editor menyediakan sarana bagi pemrogram untuk menuliskan program. Yang menarik, editor yang disediakan mampu mengenali kata-kata tercadang *C++* dan akan memberi warna tersendiri terhadap kata-kata seperti itu. Keuntungannya adalah program menjadi lebih mudah dibaca dan jika Anda melakukan kesalahan dalam menuliskan kaata-kata itu maka akan lebih cepat terdeteksi.

b. Kompiler

Kompiler adalah perangkat lunak yang berfungsi untuk menterjemahkan kode sumber atau source kode ke dalam bentuk bahasa mesin. Peranti ini juga memberikan pesan-pesan kesalahan jika terjadi kesalahan kaidah dalam penulisan

program yang terdeteksi pada tahap proses kompilasi. Hasil dari kompilasi akan berupa kode objek yang disimpan dalam berkas berekstensi .obj (disebut berkas objek).

c. Linker

Linker adalah perangkat lunak yang berfungsi menggabungkan berbagai modul yang dihasilkan oleh kompiler dan modul kode dari berbagai pustaka *C++*, serta membentuk menjadi kode yang dapat dieksekusi. Sama seperti kompiler, *linker* juga dapat mendeteksi kesalahan. Kesalahan yang terjadi sewaktu proses linking bisa disebabkan karena ada bagian pustaka atau bagian program yang tidak ditemukan.

d. Pustaka

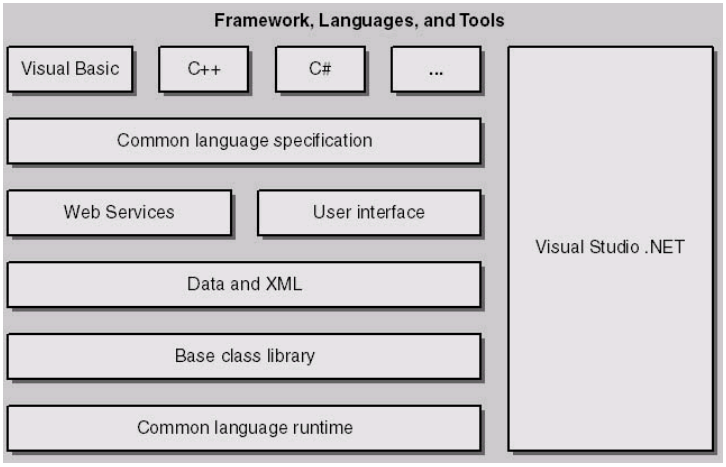
Visual C++ menyediakan berbagai pustaka (*library*) yang memudahkan pemrogram dalam melakukan berbagai operasi seperti menghitung akar kuadrat dan mengakses database. Pustaka-pustaka yang tersedia antara lain berupa: *Standart* C++ *Library* (berisi semua rutin yang tersedia pada kebanyakan kompiler C++).

2.13.3 Keuntungan Visual C++

Ada beberapa keuntungan menggunakan teknologi *.Net* dalam membangun sebuah aplikasi, baik aplikasi *web service* maupun aplikasi biasa yang sering digunakan seperti *Windows Form*, diantaranya.

1. Multi Language

Arsitektur *.NET* bersifat terbuka, sehingga memungkinkan berbagai bahasa pemrograman mengakses CLR dengan mulus. Banyak kalangan menyebut *.NET* sebagai "*open source*" versi *Microsoft*. Saat ini *.NET* dapat diprogram menggunakan *Visual Basic.NET, C++.NET, Visual C#, Jscript*, dan *J#.* Berbagai *third Party* yang dapat digunakan adalah *COBOL, Eiffel, Smalltalk, Perl, Phyton, ML, Pascal,* dan *Delphi.* Para veteran *Pascal* mungkin tidak pernah bermimpi membuat aplikasi *web* dengan bahasa "kuno" tersebut, tetapi kehadiran *.Net Framework* merealisasikan hal tersebut. Bagan berikut menjelaskan kedudukan bahasa pemrograman terhadap *.NET Framework* dengan CLR sebagai intinya.



Gambar 2.16 Kedudukan Bahasa Pemrograman Terhadap .NET

2. No DLL Hell

DLL merupakan blok atau modul-modul obyek dari sebuah aplikasi. Peranannya sangat penting, sekaligus memusingkan. Sering terjadi dalam dunia *windows,* kompatibilitas dan registrasi DLL di masing-masing *Workstation* menjadi isu besar dalam *deployment* aplikasi.

3. Strong Typing dan Type Safety

Bila anda pernah menggunakan VB6, pendefinisian tipe data bukanlah sesuatu yang mutlak wajib dilakukan karena VB akan mendefinisikan *primitive data type*, suatu *type default* untuk masing-masing angka atau karakter yang terdapat dalam variabel. Hal ini sebenarnya kurang baik karena dapat memboroskan memory dan merupakan sumber *bug*. *.NET* menyediakan *strong typing*, dimana setiap variabel wajib didefiniskan scope dan tipe datanya. Demikian pula dengan fasilitas *type safety* yang sangat bermanfaat untuk membantu dalam *coding* pemrograman, terutama fasilitas *intellisense* yang membimbing pemrogram dalam menentukan *property*, *method*, maupun *function* yang akan dipakai.

4. Cross Platform Possibility

.Net menyimpan dan mengirim data dalam bentuk XML yang merupakan format data universal di internet. Dengan demikian integrasi data antar *platform* lebih mudah dilakukan, selama platform tersebut mendukung XML. Representasi konsep ini adalah dataset, suatu *cache* data yang berbentuk XML dan dapat diakses dengan mudah. Sebuah data dapat diparsing antar tier aplikasi, baik dari database, *middle tier*, maupun aplikasi klien dalam format XML. Manipulasi format data dalam bentuk XML, .txt, maupun .rtf merupakan sesuatu yang menantang para *programmer* untuk membuat aplikasi lintas *platform.*

5. Code Once, More Application

Interface pemrograman bersifat konsisten, dengan objek model yang sama pada setiap bahasa yang digunakan. Suatu objek baik berbentuk *class*, *library*, maupun *web services* dapat diakses dengan mudah oleh berbagai aplikasi *windows* maupun *web*. Hal ini lebih menghemat waktu para *developer*, dimana sebuah objek dapat dibuat sekaligus untuk aplikasi *Web, Windows,* dan bahkan *console application* berbasis *DOS.*

2.14 Unified Modelling Process (UML)

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa berorientasi objek seperti *C++, Java, C#* atau *VB.NET*. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam *VB* atau *C.*

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax* semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut

dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch *OOD* (*Object-Oriented Design*), Jim Rumbaugh *OMT (Object Modeling Technique),* dan Ivar Jacobson *OOSE (Object-Oriented Software Engineering).*

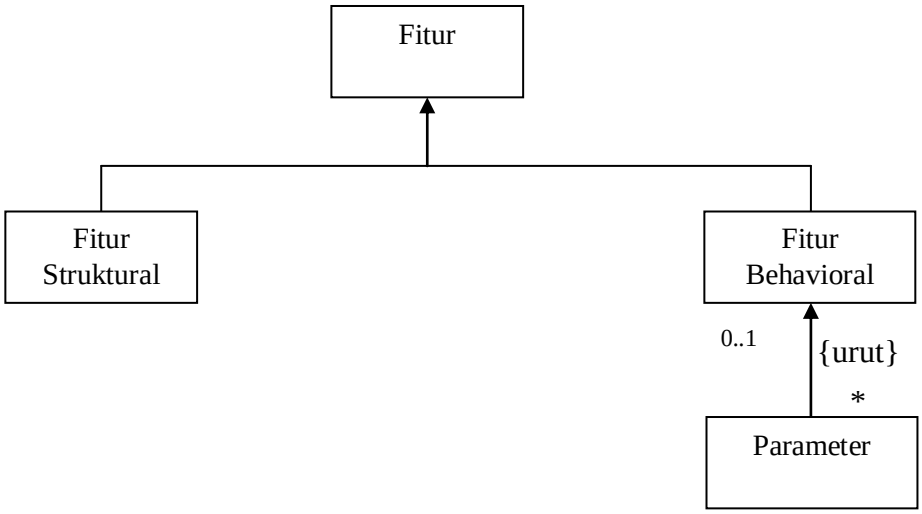
2.14.1 Notasi dan Meta-Model

UML, dalam posisinya saat ini, membedakan notasi dan meta-model. Notasi tersebut adalah grafik yang anda lihat di model, merupakan sintaks grafis dari bahasa pemodelan. Contohnya, notasi *class diagram* menentukan bagaimana pokok-pokok dan konsep, seperti *class, association*, dan *multiplicity* digambarkan. Tentu saja, hal ini memicu timbulnya pertanyaan tentang apa sebenarnya yang dimaksud dengan sebuah *association* atau *multiplicity* atau bahkan sebuah *class*. Pemakaian yang umum memberi beberapa definisi informal tentang hal tersebut, tetapi banyak yang masih mengharapkan definisi yang lebih detil.

Pikiran spesifikasi yang rinci dan bahasa desain paling umum terjadi dibidang metode formal. Dalam teknik ini, desain dan spasifikasi oleh beberapa *derivative* kalkulus predikat. Definisi ini secara matematis sudah rinci dan tidak mempunyai ambiguitas. Meskipun demikian, nilai definisi ini tidak *universal.*

Kebanyakan bahasa pemodelan grafis memiliki sedikit detil, notasi mereka cenderung berupa intuisi dari pada definisi formal, tetapi banyak orang yang menganggap mereka berguna, dan kegunaan inilah yang paling penting. Akan tetapi, para metodologis berusaha mencari cara-cara untuk memperbaiki detil metode ini tanpa mengorbankan kegunaannya. Salah satu cara

melakukannnya adalah mendefinisikan meta-model sebuah diagram, biasanya *class diagram*, yang mendefinisikan konsep bahasa tersebut.



Gambar 2.17 sebagian kecil Meta-Model UML

Banyak orang yangterlibat dalam perkembangan UML hanya tertarik pada model-model, khususnya karena hal ini penting dalam penggunaan UML dan sebagai bahasa pemrograman. Masalah-masalah notasi sering menjadi masalah nomor dua, hal ini penting untuk diingat jika anda akan mengakrabkan diri dengan dokumen-dokumen standar.

2.14.2 Diagram-Diagram UML

UML terdiri dari 8 jenis diagram, diantaranya adalah :

1. Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor

dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya.

Seorang / sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case* diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-extend *use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

2. Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok :

1. Nama (dan *stereotype*)
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public*, dapat dipanggil oleh siapa saja

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*. Sesuai dengan perkembangan *class model*, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*.

3. Statechart Diagram

Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari stimuli yang diterima. Pada umumnya statechart diagram menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart diagram*).

Dalam UML, state digambarkan berbentuk segiempat dengan sudut membulat dan memiliki nama sesuai kondisinya saat itu. Transisi antar *state* umumnya memiliki kondisi *guard* yang merupakan syarat terjadinya transisi yang bersangkutan, dituliskan dalam kurung siku. *Action* yang dilakukan sebagai akibat dari *event* tertentu dituliskan dengan diawali garis miring. Titik awal dan akhir digambarkan berbentuk lingkaran berwarna penuh dan berwarna setengah.

4. Activity Diagram

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *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.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan *behaviour* pada kondisi tertentu. Untuk mengilustrasikan proses-

proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

5. 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. Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*.

6. Collaboration Diagram

Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence diagram*, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap message memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1. *Messages* dari level yang sama memiliki prefiks yang sama.

7. Component Diagram

Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi code, baik berisi *source code* maupun *binary code*, baik *library* maupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*.

8. Deployment Diagram

Deployment/physical diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, *server* atau piranti keras apa), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi *server*, dan hal-hal lain yang bersifat fisikal Sebuah node adalah *server*, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya. Hubungan antar node (misalnya TCP/IP) dan *requirement* dapat juga didefinisikan dalam diagram ini.