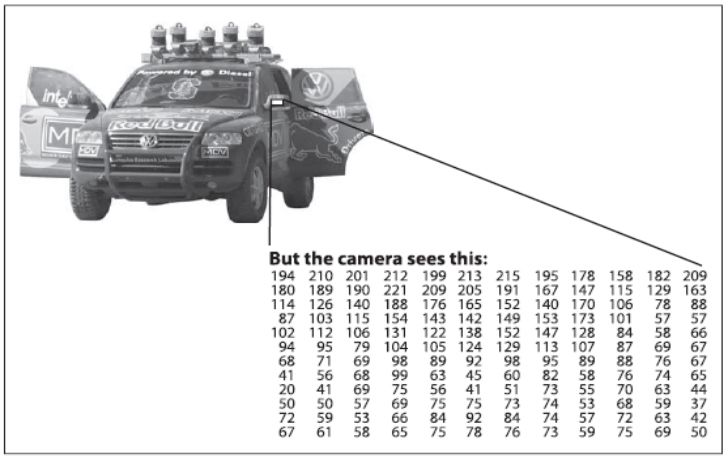
BAB 2
LANDASAN TEORI

2.1 Computer Vision

Computer Vision adalah suatu proses transformasi atau perubahan dari data yang berasal dari kamera *video* maupun foto/gambar kedalam sebuah hasil keputusan ataupun sebuah presentasi yang baru, dimana hasil dari kegiatan transformasi tersebut memiliki kepentingan untuk mencapai suatu tujuan. Data yang dimasukan kedalam kegiatan transformasi tersebut memungkinkan untuk memiliki beberapa informasi yang terkonstektual seperti halnya sebuah foto/gambar yang didalamnya terdapat berbagai objek. Dengan demikian akan didapatkan keputusan-keputusan yang akan diambil pada gambar, misalnya berupa "adakah telapak tangan seseorang pada gambar tersebut?" atau "siapa sajakah orang yang terdapat pada foto tersebut?". Adapun perubahan kedalam presentasi yang baru seperti perubahan gambar menjadi *grayscale* atau juga pemotongan objek pada gambar.

Berbeda halnya dengan manusia yang memiliki pemikiran untuk meneliti, memahami, dan membandingkan informasi pada objek secara langsung dengan informasi dari pengalaman-pengalaman yang didapatkan selama bertahun-tahun hidup didunia. Dalam sistem penglihatan mesin (*machine vision*) komputer hanya mampu mendapatkan informasi berupa kumpulan angka-angka dari media input data seperti kamera atau disket. Berikut contoh sebuah gambar untuk mengilustrasikannya :



Gambar 2.1 Ilustrasi penglihatan komputer

Dari gambar berikut terlihat bahwa kaca samping pada gambar mobil hanyalah berupa kumpulan angka-angka jika dilihat oleh computer. Tentunya angka-angka ini masih memiliki banyak *noise* atau gangguan sehingga informasi yang terkandung sangatlah sedikit. Hal ini disebabkan dari penglihatan komputer yang terbatas hanya 2 dimensi (2D) namun objek yang diproses ialah objek 3 dimensi (3D) dunia nyata, sehingga informasi yang didapat mengalami banyak sekali gangguan dan kendala dari berbagai fenomena di kehidupan nyata (cuaca, cahaya, bayangan, refleksi cahaya, dan gerakan). Dengan demikian hal tersebut menjadi permasalahan yang terus-menerus diteliti untuk ditemukan cara atau tehnik penyelesaiannya [2].

2.2 Sistem Pendeteksian Isyarat jari

Pendeteksian isyarat jari adalah proses yang bertujuan untuk melakukan seleksi area telapak tangan dari sebuah citra gambar yang dimasukan kedalam sistem. Proses ini bekerja dengan cara mendeteksi citra gambar yang telah dimasukan, apakah terdapat obek telapak tangan pada citra gambar tersebut atau tidak. Apabila objek telapak tangan telah ditemukan maka, citra telapak tangan tersebut akan ditandai oleh sistem dan akan dilakukan tahapan-tahapan selanjutnya misalnya pengekstraksian telapak tangan tersebut dari citra gambar sebelumnya. Selain itu, jika masukan pada sistem adalah bearbentuk *video* maka proses yang akan dilakukan adalah proses *hand tracking*. Secara umum, *proses*

hand tracking dan proses pendeteksian isyarat jari mempunyai fungsi yang sama. Pada dasarnya, meskipun sistem pengenalan isyarat jari didasari dengan metode-metode yang berbeda, sistem pengenalan siyarat jari memiliki tahapan-tahapan yang hampir sama, berikut tahapan-tahapan yang pada umumnya terdapat pada sistem pengenalan isyarat jari :

1. *Grayscaling* citra gambar
2. Pendeteksian jari tangan atau *tracking* jari
3. Pengenalan/pencocokan jari tangan

2.2.1 Grayscaling

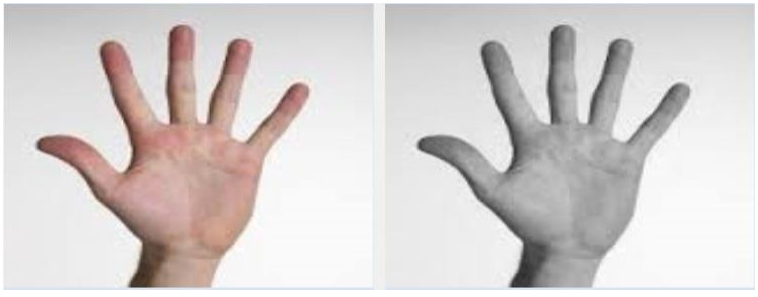
Grayscaling pada citra merupakan tahap pengkonversian citra gambar warna yang terdiri dari tiga parameter warna yaitu merah, hijau, dan biru (RGB). Jika citra gambar RGB diproses kedalam sistem pengenalan wajah, maka akan lebih sulit untuk diproses karena mengingat citra gambar RGB memiliki tiga parameter warna, oleh karena itu diperlukan suatu penyempitan parameter kedalam warna abu sehingga menjadi lebih mudah dalam hal penyamaan parameter yang akan digunakan pada implementasi sistem pengenalan isyarat jari. Untuk mengkonversikan citra yang memiliki warna RGB kedalam derajat keabuan dapat menggunakan persamaan (2.1) atau (2.2)

$$\boldsymbol{Gray = (R + G + B)/3} \qquad \textbf{(2.1)}$$

atau

$$\boldsymbol{Gray = 0.114 * R + 0.587 * G + 0.299 * B} \qquad \textbf{(2.2)}$$

Maka hasil dari proses *grayscaling* citra dapat dilihat pada gambar 2.2.

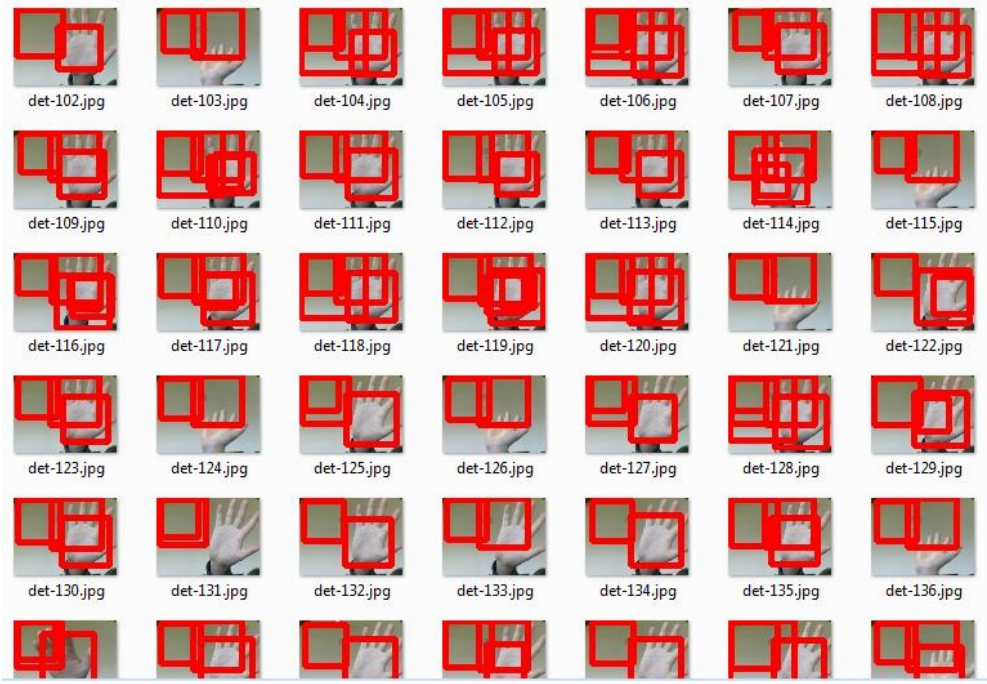


Gambar 2.2 *Grayscaling* citra

Nilai ambang batas pada setiap filter diatur setiap saat proses filter terjadi, sehingga nilai dari ambang batas bersifat dinamis. Filter pada setiap tingkat telah dilatih untuk mengklasifikasi citra yang telah melalui tahap sebelumnya. Saat proses pengklasifikasian, jika salah satu filter gagal terlewati maka citra dapat dikatakan sebagai daerah yang bukan wajah. Jika citra belum cukup untuk dikatakan sebagai strong classifier maka proses diulang sampai bobot terpenuhi dengan menaikkan nilai dari ambang batas. Ketika citra dapat melewati setiap filter yang ada di dalam rantai, maka dapat dikatakan daerah tersebut merupakan telapak tangan

2.2.2 Pendeteksian Isyarat Jari

Pendeteksian isyarat jari adalah proses yang bertujuan untuk melakukan seleksi area isyarat jari dari sebuah citra gambar yang dimasukan kedalam sistem. Proses ini bekerja dengan cara mendeteksi citra gambar yang telah dimasukan, apakah terdapat obek isyarat jari pada citra gambar tersebut atau tidak. Apabila objek isyarat jari telah ditemukan maka, citra isyarat jari tersebut akan ditandai oleh sistem dan akan dilakukan tahapan-tahapan selanjutnya misalnya pengekstraksian isyarat jari tersebut dari citra gambar sebelumnya. Selain itu, jika masukan pada sistem adalah berbentuk video maka proses yang akan dilakukan adalah proses *hand tracking*. Secara umum, proses *hand tracking* dan proses pendeteksian isyarat jari mempunyai fungsi yang sama. Berikut gambar 2.2 setelah dilakukannya proses pendeteksian telapak tangan.



Gambar 2.3 Pendeteksian telapak tangan.

2.2.3 Pengenalan Isyarat Jari

Proses ini bertujuan untuk membandingkan citra isyarat jari yang dideteksi oleh sistem dengan citra isyarat jari yang terdapat didalam *database*. Dari proses perbandingan isyarat jari tersebut akan menghasilkan nilai jarak terdekat yang menandakan nilai isyarat jari citra yang terdeteksi hampir menyamai dengan nilai citra wajah pada *database*. Nilai jarak ini akan menjadi nilai masukan untuk nilai kemiripan citra. Nilai kemiripan citra merupakan nilai tingkat kemiripan citra yang dibandingkan, semakin besar nilainya menandakan bahwa orang yang sedang diamati adalah orang yang sama dengan orang yang citra isyarat jari telah disimpan dalam *database*. Setelah itu dilakukan proses pencocokan, yaitu kondisi dimana hasil dari perbandingan isyarat jari memberikan keputusan terhadap citra isyarat jari yang terdeteksi apakah cocok dengan citra isyarat jari yang telah dipilih dari *database*, sehingga hasil dari pencocokan adalah benar-benar hasil yang paling tepat dan akurat.

2.3 Metode *Viola Jon*es

Metode *Viola Jones* adalah metode yang memiliki kapabilitas untuk mendeteksi wajah pada suatu citra gambar dengan tingkat keakuratan yang cukup tinggi. Pada standarnya pengolahan citra pada gambar akan dilakukan dengan merubah skala ukuran gambar kedalam ukuran yang ditentukan dan melakukan pendeteksian dengan skala tetap pada gambar tersebut. Dengan hal tersebut simpulkan bahwa waktu untuk pendeteksian akan lebih cepat dibandingkan dengan melakukan pendeteksian terhadap citra gambar yang berukuran berbeda-beda.

Berbeda halnya dengan Metode *Viola Jones* yang terus merubah ukuran skala pendeteksian dibandingkan harus merubah ukuran dari skala gambar dan menjalankan banyak pendeteksian terhadap gambar dengan ukuran skala pendektesian yang berbeda tiap waktunya. Hal ini diperkirakan akan membutuhkan lebih banyak waktu, tapi Metode *Viola Jones* memiliki fitur pendeteksian yang mengkonstruksi gambar kedalam gambar integral dan beberapa fitur persegi yang simple yang disebut *Haar wavelets*. Sehingga hasil kalkulasi dalam skala pendeteksian akan sama berapapun ukuran dari gambar yang diproses [12].

Setelah melalu proses *grayscale* lalu gambar dihitung cintra integralnya dari pendeteksian ini yaitu dengan cara mengubah contoh gambar yang berdimensi 5x5 kedalam gambar integral. Yaitu dengan cara menjumlahkan tiap *pixel* dengan seluruh *pixel* yang terdapat di daerah sebelah kiri dan atas *pixel* tersebut, seperti yang terdapat pada contoh berikut :

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

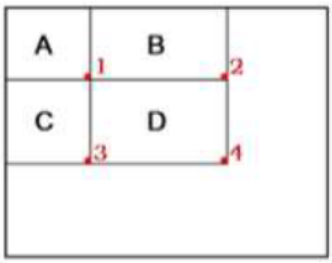
Input Image

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

Integral Image

Gambar 2.4 Input integral image

Dengan demikian, selanjutnya perhitungan dapat dilakukan dengan cara menjumlahkan 4 nilai *pixel* kedalam *pixel* yang telah dipilih. Berikut ilustrasi pada gambar :



Gambar 2.5 Ilustrasi posisi *pixel* pada gambar

Dimana D adalah nilai *pixel* kanan bawah, A adalah nilai *pixel* kiri atas, B adalah *pixel* atas dari *pixel* D dan C adalah nilai *pixel* kiri dari *pixel* D. Dengan rumusan seperti berkut : D = D + A - (B + C). Dengan *integral image* ini maka perhitungan *Haar* akan menjadi lebih cepat. Berikut adalah perbandingan perhitungan haar pada gambar tanpa integral dengan gambar integral, dimana total *pixel* hitam dikurangi dengan total *pixel* putih :

Gambar tanpa integral

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Input Image

Gambar 2.6 Input Image

Sehingga nilai fitur *Haar* adalah sebagai berikut :

Nilai fiatur *Haar* = |(total *pixel* hitam)-(total *pixel* putih)|

=|(1+1+1+1+1+1+1+1)-(1+1+1+1+1+1+1+1)|

= |8-8|

= 0

Gambar integral

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25

Integral Image

Gambar 2.7 Integral Image

Sehingga nilai fitur *Haar* adalah sebagai berikut :

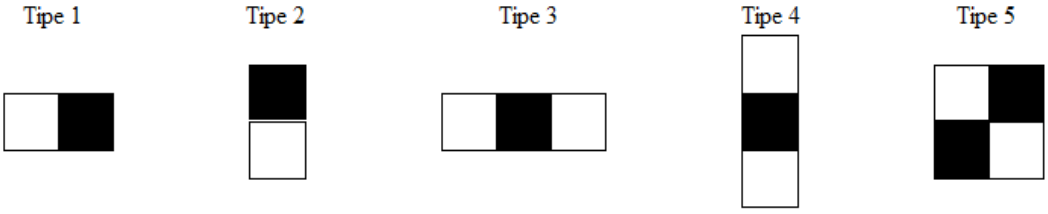
Nilai fiatur *Haar* = |(total *pixel* hitam)-(total *pixel* putih)|

=|(15+1-(3+5))-(25+3-(5+15))|

= |8-8|

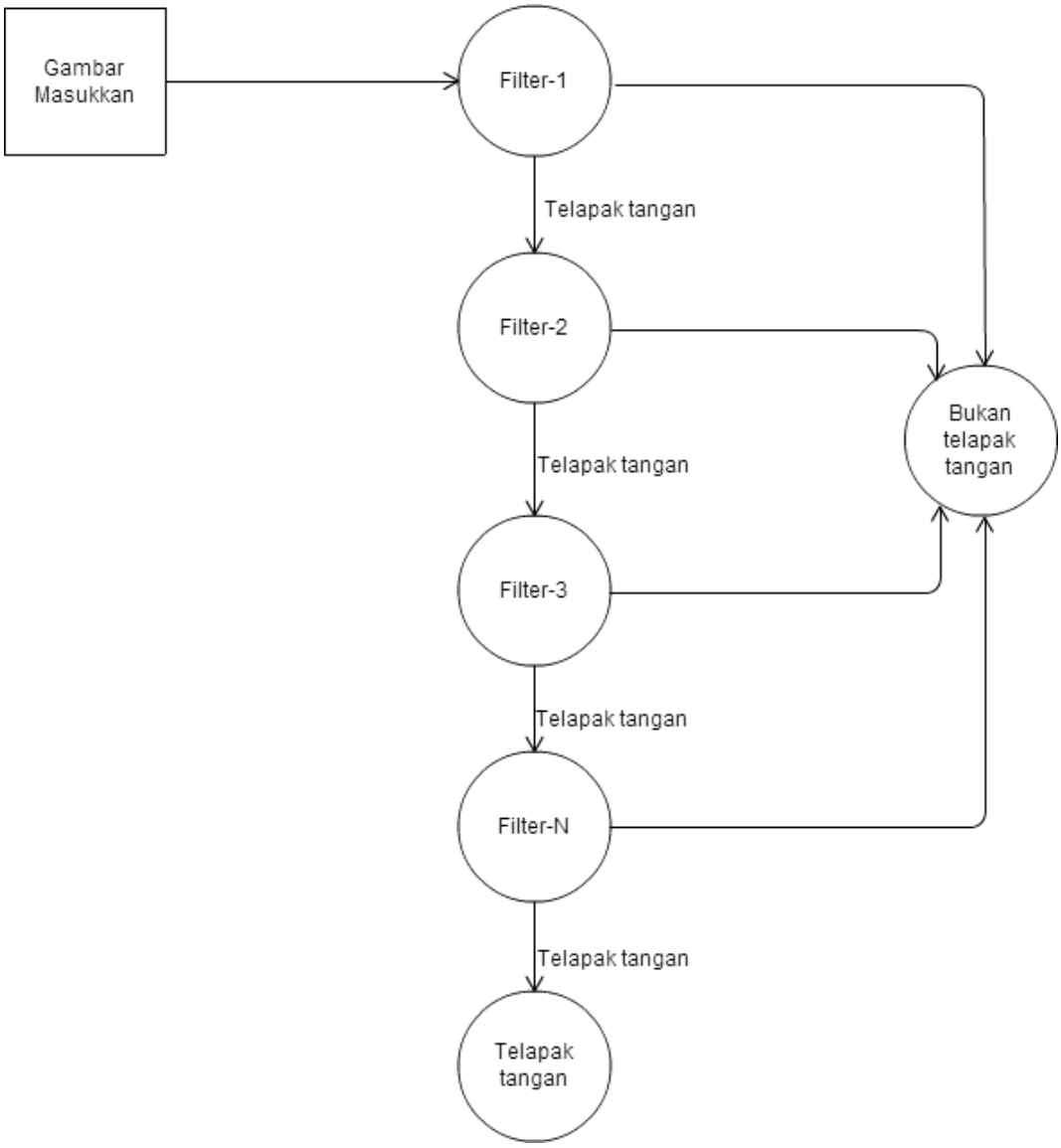
= 0

Kemudian hasil nilai tersebut akan menghasilkan fitur *sub-window* yang memiliki dua atau lebih kombinasi kotak hitam dan putih yang akan digunakan untuk tahap pendetaksian selanjutnya. Berikut contoh gambar untuk tipe-tipe dari kombinasi kotak :



Gambar 2.8 Tipe-tipe kombinasi kotak

Selanjutnya, setelah nilai fitur didapatkan *Viola-Jones* menggunakan metode *AdaBoost machine-learning* untuk mengetahui suatu fitur apakah merepresentasikan ada tidaknya wajah dalam suatu citra masukan. Metode ini menggabungkan banyak *classifier* lemah menjadi satu *classifier* kuat. *Classifier* adalah informasi ciri yang menandakan adanya suatu objek pada suatu citra misalnya ciri pada wajah. Sedangkan *classifier* lemah adalah suatu jawaban benar namun memiliki tingkat kebenaran yang kurang akurat, maka jika digabungkan *classifier* lemah tersebut akan menghasilkan suatu *classifier* kuat. Berikut adalah gambar dari metode *AdaBoost* yang digunakan untuk mendeteksi isyarat jari:



Gambar 2.9 Algoritma Adaboost

Gambar berikut menjelaskan penolakan ciri menggunakan *Viola Jones Classifier*, dimana ciri yang memiliki kemungkinan kecil sebagai isyarat jari akan ditolak terus menerus hingga *Classifier* akan menyisakan ciri isyarat jari yang asli pada citra gambar.

Tingkatan proses penyeleksian fitur diurutkan berdasarkan bobot terberat sampai teringan, sehingga dapat secepat mungkin untuk mengklasifikasikan bahwa tidak terdeteksi isyarat jari dalam citra masukan, pengurutan proses penyeleksian fitur ini disebut *cascade classifier*.



Gambar 2.10 Fitur tingkat pertama pada Metode *Viola Jones*

2.4 Model Isyarat Jari

Sistem ini menggunakan *webcam* yang mengambil citra pola jari tangan. Pola tersebut didapat dari bentuk gerakan jari telunjuk atau jempol. Jari tersebut nantinya menunjuk ke suatu arah atas.



Gambar 2.11 Model Isyarat jari untuk perintah *Play*



Gambar 2.12 Model Isyarat jari untuk perintah *Previous*



Gambar 2.13 Model Isyarat jari untuk perintah *Next*



Gambar 2.14 Model Isyarat jari untuk perintah *Pause*



Gambar 2.15 Model Isyarat jari untuk perintah *Stop*

Arah yang ditunjukkan oleh bentuk pola jari ini akan menjadi dasar masukan untuk perintah pada aplikasi pemutar musi*k* tanpa mengendalikan langsung dengan bantuan mouse. Pada dasarnya perintah aplikasi pemutar musik ada 5, yaitu perintah *play*, *pause*, *next*, *previous* dan *stop*. Hal ini agar pengguna tidak perlu menghafal bentuk isyarat jari. Pemrosesan citra pola jari dilakukan menggunakan bahasa C# dengan *library* OpenCV.

2.5 Pemrograman Berorientasi Objek

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya[12]. Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientas objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek,

perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek.

Pada saat ini, metode berorientasi objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat mentransformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekatan terstruktur, jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan pada saat ini beragam dengan *platform* yang berbeda-beda, sehingga menimbulkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut.

Keuntungan menggunakan metodologi berorientasi objek adalah sebagai berikut :

1. Meningkatkan produktivitas

   Karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*reusable*).
2. Kecepatan pengembangan

   Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean.
3. Kemudahan pemeliharaan

   Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dan pola-pola yang mungkin sering berubah-ubah.
4. Adanya konsistensi

   Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.
5. Meningkatkan kualitas perangkat lunak

   Karena pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

2.5.1 Konsep Dasar Berorientasi Objek

Pendekatan berorientasi objek merupakan suatu teknik atau cara pendekatan dalam melihat permasalahan dan sistem (sistem perangkat lunak, sistem informasi, atau sistem lainnya)[12]. Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.

Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut, mulai dan abstraksi objek, kelas, hubungan antar kelas sampai abstraksi sistem. Saat mengabstraksikan dan memodelkan objek, data dan proses-proses yang dipunyai oleh objek akan dienkapsulasi (dibungkus) menjadi suatu kesatuan. Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemrograman, dan pengujian perangkat lunak. Ada berbagai teknik yang dapat digunakan pada masing-masing tahap tersebut, dengan aturan dan alat bantu pemodelan tertentu.

Sistem berorientasi objek merupakan sebuah sistem yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah sistem yang komponennya dibungkus (dienkapsulasi) menjadi kelompok data dan fungsi. Setiap komponen dalam sistem tersebut dapat mewarisi atribut dan sifat dan komponen lainnya, dan dapat berinteraksi satu sama lain.

Berikut ini adalah beberapa konsep dasar yang harus dipahami tentang metodologi berorientasi objek:

1. Kelas (*Class*)

   Kelas adalah sekumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statis dan himpunan objek yang sama yang mungkin lahir atau diciptakan dan kelas tersebut. Sebuah kelas akan mempunyai sifat (atribut), kelakuan (metode/operasi), hubungan (*relationship*)dan arti. Suatu kelas dapat diturunkan dan kelas yang lain, dimana atribut dan kelas semula dapat diwariskan ke kelas yang baru. Secara teknis kelas adalah sebuah struktur dalam pembuatan perangkat lunak. Kelas merupakan bentuk struktur pada kode program yang menggunakan metodologi berorientasi objek.

2. Objek (*object*)

   Objek adalah abstraksi dari sesuatu yang mewakili dunia nyata benda, manusia, satu organisasi, tempat, kejadian, struktur, status, atau hal-hal lain yang bersifat abstrak. Objek merupakan suatu *entitas* yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi, dan dihancurkan.

Secara teknis, sebuah kelas saat program dieksekusi maka akan dibuat sebuah objek. Objek dilihat darisegi teknis adalah elemen pada saat *runtime* yang akan diciptakan, dimanipulasi, dan dihancurkan saat eksekusi sehinga sebuah objek hanya ada saat sebuah program dieksekusi. Jika masih dalam bentuk kode, disebut sebagai kelas jadi pada saat *runtime* (saat sebuah program dieksekusi), yang kita punya adalah objek, di dalam teks program yang kita lihat hanyalah kelas.

3. Metode (*method*)

   Operasi atau metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada metodologi struktural. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri. Operasi atau metode merupakan fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek.

Metode atau operasi dapat berasal dari *event*, aktifitas atau aksi keadaan, fungsi, atau kelakuan dunia nyata. Contoh metode atau operasi misalnya *Read*, *Write*, *Move*, *Copy*, dan sebagainya.

4. Atribut (*attribute*)

   Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut dipunyai secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya.

5. Abstraksi (*abstraction*)

   Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.

6. Enkapsulasi (*encapsulation*)

   Pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerja-nya.

7. Pewarisan (*inheritance*)

   Mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dan dirinya.

8. Antarmuka (*interface*)

   Antarmuka sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah *interface* dapat diimplementasikan oleh kelas lain.

9. *Reusability*

   Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.

10. Generalisasi dan Spesialisasi

    Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus. Misalnya kelas yang lebih umum (generalisasi) adalah kendaraan darat dan kelas khususnya (spesialisasi) adalah mobil, motor, dan kereta.

11. Komunikasi Antar objek

    Komunikasi antar objek dilakukan lewat pesan (*message*) yang dikirim dari satu objek ke objek lainnya.

12. Polimorfisme (*polymorphism*)

    Kemampuan suatu objek digunakan di banyak tujuan yang berbeda dengan nama yang sehingga menghemat baris program.

13. *Package*

Package adalah sebuah kontainer atau kemasan yang dapat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam *package* yang berbeda.

2.5.2 Pengenalan UML

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

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

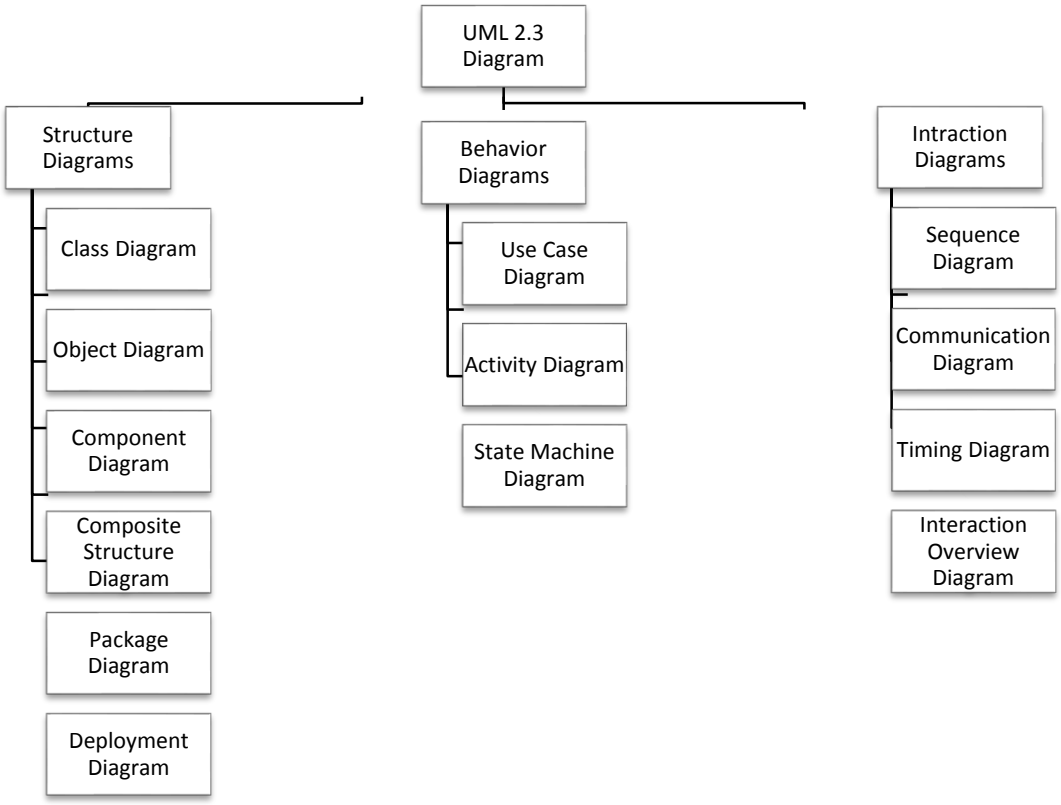
Pada perkembangan teknik pemrograman berorientasi objek, munculah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual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

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

Seperti yang kita ketahui di dunia sistem informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan dan konteksnya. Begitu juga dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunaannya. Jadi, belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin digambarkan. Secara analogi jika dengan bahasa yang digunakan sehari-hari, belum tentu penyampaian bahasa dengan puisi adalahhal yang salah. Sistem informasi bukanlah ilmu pasti, maka jika ada banyak perbedaan dan interpretasi di dalam bidang sistem informasi merupakan hal yang sangat wajar.

2.5.3 Diagram UML

Pada *UML* 2.3 terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada gambar ini :



Gambar 2.16 Diagram UML

Berikut ini penjelasan singkat dari pembagian kategori tersebut :

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

2.5.4 *Use Case Diagram*

Use Case atau diagram *Use Case* merupakan pemodelan untuk kelakuan (*behavior*) sistem informasi yang akan dibuat. *Use Case*mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *Use Case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu.

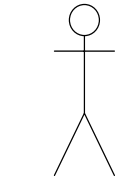
Syarat penamaan pada *Use Case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *Use Case*yaitu pendefinisian apa yang disebut aktor dan *Use Case.*

2.5.4.1 Aktor

Aktor adalah sesuatu (entitas) yang berhubungan dengan sistem dan berpartisipasi dalam *use case*. Aktor menggambarkan orang, sistem atau *entitas eksternal* yang secara khusus membangkitkan sistem dengan *input* atau masukan kejadian-kejadian, atau menerima sesuatu dari sistem. Aktor dilukiskan dengan peran yang mereka mainkan dalam *use case*, seperti Staff, Kurir dan lain-lain.

Dalam *use case diagram* terdapat satu aktor pemulai atau *initiator actor* yang membangkitkan rangsangan awal terhadap sistem, dan mungkin sejumlah aktor

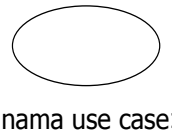
lain yang berpartisipasi atau *participating actor*. Akan sangat berguna untuk mengetahui siapa aktor pemulai tersebut.



Gambar 2.17 Aktor

2.5.4.2 *Use Case*

Use case yang dibuat berdasarkan keperluan aktor merupakan gambaran dari "apa" yang dikerjakan oleh sistem, bukan "bagaimana" sistem mengerjakannya. *Use case* diberi nama yang menyatakan apa hal yang dicapai dari interaksinya dengan aktor.

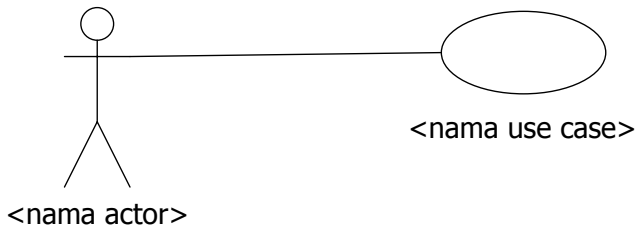


Gambar 2.18 Use case

2.5.4.3 *Relationship*

Relasi (*relationship*) digambarkan sebagai bentuk garis antara dua simbol dalam *use case diagram*. Relasi antara *actor* dan *use case* disebut juga dengan asosiasi (*association*). Asosiasi ini digunakan untuk menggambarkan bagaimana hubungan antara keduanya.

Relasi-relasi yang terjadi pada *use case diagram* bisa antara *actor* dengan *use case* atau *use case* dengan *use case*.



Gambar 2.19 Relationship

2.5.4.4 *Activity Diagram*

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

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

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

2.5.4.5 *Sequence Diagram*

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

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada

diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.

Penomoran pesan berdasarkan urutan interaksi pesan. Penggambaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah pesan yang berjalan terlebih dahulu.

2.5.4.6 *Class Diagram*

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

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

1) Kelas Main

   Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.

2) Kelas yang menangani tampilan sistem

   Kelas yang mendefinisikan dan mengatur tampilan ke pemakai.

3) Kelas yang diambil dari pendefinisian *usecase*

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

4) Kelas yang diambil dari pendefinisian data

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

Jenis-jenis kelas diatas juga dapat digabungkan satu sama lain sesuai dengan pertimbangan yang dianggap baik asalkan fungsi-fungsi yang sebaiknya ada pada struktur kelas tetap ada. Susunan kelas juga dapat ditambahkan kelas utilitas seperti koneksi ke basis data, membaca *file* teks, dan lain sebagainya sesuai kebutuhan.

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

2.6 *Open CV*

OpenCV adalah sebuah *library open source* untuk visi komputer yang bisa didapatkan dari http://SourceForge.net/projects/opencvlibrary. *Library* ini ditulis dengan bahasa C dan C++, serta dapat dijalankan dengan Linux, Windows, dan Mac OS X. OpenCV dirancang untuk efisiensi komputasional dan dengan fokus yang kuat pada aplikasi *real-time.*

Salah satu tujuan OpenCV adalah untuk menyediakan infrastruktur visi komputer yang mudah digunakan yang membantu orang-orang dalam membangun aplikasiaplikasi visi yang *sophisticated* dengan cepat. *Library* pada OpenCV berisi lebih dari 500 fungsi yang menjangkau berbagai area dalam permasalahan visi, meliputi inspeksi produk pabrik, pencitraan medis, keamanan, antarmuka pengguna, kalibrasi kamera, visi stereo, dan robotika. Karena visi komputer dan pembelajaran mesin seringkali berkaitan, OpenCV juga memiliki *Machine Learning Library* (MLL). *Sublibrary* ini berfokus pada pengenalan pola statistik dan *clustering*. MLL sangat berguna untuk tugas-tugas visi yang berada dalam misi inti OpenCV, tetapi MLL cukup umum digunakan untuk permasalahan pembelajaran mesin.

Lisensi open source pada OpenCV telah distrukturisasi sehingga pengguna dapat membangun produk komersial menggunakan seluruh bagian pada OpenCV. Tidak ada kewajiban untuk meng-open source produk tersebut atau untuk memberikan peningkatan ke domain publik. Sebagian karena peraturan lisensi liberal ini, maka terdapat komunitas pengguna dalam jumlah yang sangat besar, termasuk di dalamnya orang-orang dari perusahaan besar (seperti IBM, Microsoft,

Intel, SONY, Siemens, dan Google) serta pusat-pusat penelitian (seperti Stanford, MIT, CMU, Cambridge, dan INRIA).

Sejak peluncuran pertamanya pada Januari 1999, OpenCV telah digunakan pada banyak aplikasi, produk, dan usaha-usaha penelitian. Aplikasi-aplikasi ini meliputi penggabungan citra pada peta web dan satelit, *image scan alignment*, pengurangan *noise* pada citra medis, sistem keamanan dan pendeteksian gangguan, sistem pengawasan otomatis dan keamanan, sistem inspeksi pabrik, kalibrasi kamera, aplikasi militer, serta kendaraan udara tak berawak, kendaraan darat, dan kendaraan bawah air. OpenCV juga telah digunakan untuk pengenalan suara dan musik, dimana teknik pengenalan visi diaplikasikan pada citra spektogram suara.

OpenCV adalah singkatan dari *Open Computer Vision*, yaitu *library open source* yang dikhususkan untuk melakukan pengolahan citra. Tujuannya adalah agar komputermempunyai kemampuan yang mirip dengan cara pengolahan visual pada manusia. *Library* ini dibuat untuk bahasa C/C++ sebagai optimasi aplikasi *real-time*. OpenCVmemiliki API (*Application Programming Interface*) untuk pengolahan tingkat tinggimaupun tingkat rendah. Pada OpenCV juga terdapat fungsi-fungsi siap pakai untuk me*load*,menyimpan, serta mengakuisisi gambar dan video.

Library OpenCVmemiliki fitur-fitur sebagai berikut:

1. Manipulasi data gambar (mengalokasi memori, melepaskan memori, menduplikasi gambar, mengatur serta mengkonversi gambar)
2. *Image*/Video I/O (bisa menggunakan kamera yang sudah didukung oleh *library* ini)
3. Manipulasi matriks dan vektor, serta terdapat juga *routines* aljabar linear (*products*, *solvers*, *eigenvalues*, SVD)
4. Pengolahan citra dasar (penapisan, pendeteksian tepi, *sampling* dan interpolasi, konversi warna, operasi morfologi, histogram, piramida citra)
5. Analisis structural
6. Kalibrasi kamera

7. Pendeteksian gerakan
8. Pengenalan objek
9. GUI dasar (menampilkan gambar/video, mengontrol *mouse*/*keyboard*, *scrollbar*)
10. *Image labelling* (garis, kerucut, poligon, penggambaran teks)

Libraries OpenCV menyediakan banyak algoritma visi komputer dasar, dengankeuntungan bahwa fungsi-fungsi tersebut telah diuji dengan baik dan digunakan oleh para peneliti di seluruh dunia. *Libraries* OpenCV juga menyediakan sebuah modul untuk pendeteksian objek yang menggunakan metode Viola Jones.

2.7 *Microsoft Visual 2010*

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

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