BAB II

LANDASAN TEORI

II.1 Computer Vision

Pengolahan citra adalah pengolahan suatu citra dengan menggunakan komputer secara khusus, untuk menghasilkan suatu citra yang lain. Sedangkan *computer vision* dapat didefinisikan setara dengan pengertian pengolahan citra yang dikaitkan dengan akusisi citra, pemrosesan, klasisfikasi, pengakuan, dan pencakupan keseluruhan, pengambilan keputusan yang diikuti pengindentifikasian citra.

Computer Vision adalah transformasi atau perubahan dari data-data yang dapat berupa gambar diam ataupun video kamera menjadi bentuk lain atau suatu representasi baru dan membantu dalam pengambilan keputusan[2]. Semua bentuk transformasi yang dilakukan di arsipkan atau disimpan untuk tujuan tertentu. Input data dapat berasal dari informasi yang berhubungan dengan objek. Pada representasinya baru dimungkinkan berarti merubah warna dari suatu citra menjadi citra yang *grayscale* atau menghilangkan pergerakan kamera dari suatu citra berurutan.

Manusia seperti halnya mahluk visual yang mudah sekali tertipu dan berfikir jika tugas *computer vision* itu mudah. Seperti halnya otak manusia membagi sinyal dalam banyak saluran/*channel* dimana mengalirkan informasi yang berbeda-beda kedalam otak. Otak manusia memiliki *attention system* mengidetifikasi dengan jalannya sendiri. Setelah diproses akan menghasilkan informasi balik dan akhirnya berhasil mengartikan visual yang dihasilkan oleh

input dari otot kontrol sensor dan indra lainnya yang memperbolehkan otak untuk mengasosiasikan secara bersilang segala informasi yang didapat. Proses ini berulang dengan perangkat keras sensor berupa mata yang memiliki mekanisme pengontrol cahya (*lightning*) melalui iris dan melakukan *setting* terhadap penerimaan permukaan oleh retina.

Dalam sistem *machine vision*, suatu komputer menerima kumpulan angka dari kamera atau tempat penyimpanan *disk*, dan itulah yang komputer lihat terhadap suatu citra. Dalam kumpulan angka yang dilihat komputer terdapat *noisy* yang besar sehingga memperkecil informasi yang didapat. Sehingga sebelum mengolah *image* atau citra perlunya dibuat pengolahan *noisy* ini agar mendapatkan persespsi yang baik dari kumpulan angka yang memiliki *noisy* tersebut.

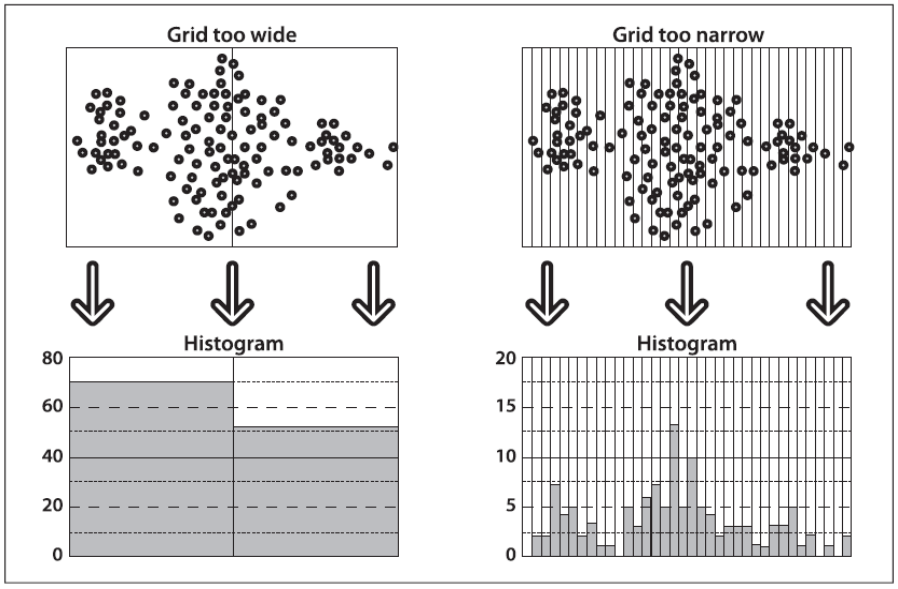
II.1.1 Histograms

Dalam proses analisis citra, objek, dan informasi video, biasanya dilakukan representasi apa yang dilihat oleh komputer melalui kamera ataupun *disk* sebagai histogram. Histogram dapat digunakan untuk merepresentasikan segala macam hal yang ada dalam suatu citra seperti distribusi warna pada suatu objek, garis batas dari suatu objek, dan distribusi probabilitas representasi hipotesis yang sedang digunakan mengenai lokasi objek.

Histogram banyak ditemukan dalam berbagai implementasi *computer vision*. Histogram digunakan untuk mendeteksi transisi gambar dalam video dengan menampilkan tanda/*mark* statistik dari *edge* dan warna dari setiap frame

ke frame. Identifikasi *interest points* dalam semua citra dengan memberikan tanda pada setiap *interest point* yang ada di pada histogram. Histogram dari *edges*, warna, *corners*, dan lainnya merupakan objek yang dilewatkan dalam proses klasifikasi untuk proses objek recognition.

Histogram diambil dengan membuat statistik berdasarkan kumpulan angka yang dilihat oleh komputer dan biasa dibuat dalam bentuk matriks dalam dimensi tertentu dan memiliki jumlah chanel tertentu. Berikut salah satu contohhistogram yang dipakai:



Gambar II.1 Sample Histogram

Dalam contoh gambar II.1, histogram diambil dari kumpulan titik yang mewakili atau representasi dari matriks suatu citra. Lalu dilakukan komputasi untuk melihat kepadatan dari titik-titik tersebut dengan membuat grid pada kumpulan titik tersebut sehingga dapat dilihat grid yang memiliki kepadatan tertentu. Dalam proses tersebut diperlukan grid yang tidak terlalu besar dan juga

tidak terlalu berdekatan dan banyak, bila terlalu besar lebar grid dari citra tersebut maka kemungkinan galat atau *error* atau kesalahan yang mungkin terjadi dalam proses komputasi dikarenakan *range* yang terlalu besar, sedangkan bila terlalu padat maka proses komputasi pun akan membutuhkan waktu yang lebih lama karena terlalu detail.

Dalam histogram dapat dilakukan proses *image processing*, *image transformation*, *contours detecting*, dan *tracking and motion*. Proses dilakukan dengan menggunakan algoritma tertentu guna untuk memanipulasi histogram yang didapat.

Dalam openCV histogram dapat dilihat sebagai multi-dimensional array. Setiap dimensi sesuai dengan fitur yang ada pada citra atau objek tertentu. Array tempat menyimpan nilai dari multi-dimensional histogram biasa disebut dengan *bin*.

Proses yang dapat dilakukan dalam memanipulasi yaitu normalisasi, pencarian nilai tertinggi dan terendah di dalam histogram, proses komputasi nilai atau besaran dari memmory yang dipakai oleh histogram, dan *comparing histogram.* Akan tetapi yang terpenting dalam *computer vision* adalah bagaimana membandingkan dua histogram dengan kriteria tingkat kesamaan atau kemiripan pola antar histogram tersebut. Sehingga dengan normalisasi tersebut andaikan terdapat perbedaan histogram yang dikarenakan transformasi masih dapat di deteksi dengan baik.

II.1.2 Tracking dan Motion

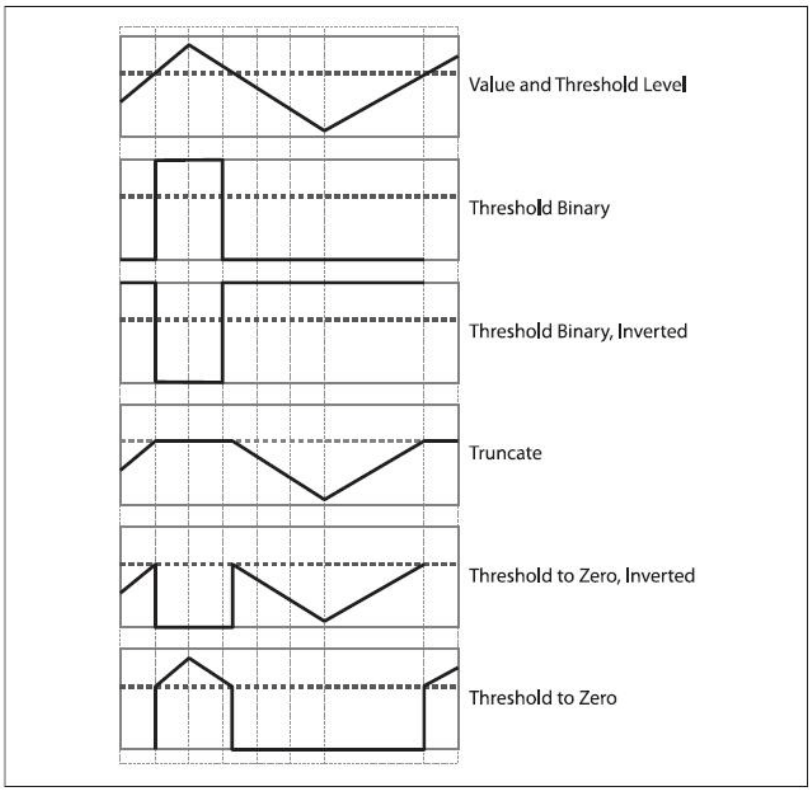
Dalam *computer vision* diperlukan suatu proses dimana komputer harus bisa mengikuti pergerakan objek yang ditentukan dan dengan sumber citra berupa video baik dari *disk* ataupun real time berasal dari input kamera. Identifikasi dilakukan untuk menemukan *object of interest* dari satu frame yang merupakan *subsequent* dari suatu video. *Tracking* merupakan istilah untuk melacak atau mencari suatu objek pada suatu citra, dalam prosesnya ada sesuatu yang penting yang terjadi pada objek tersebut dalam citra, ketika citra tersebut merupakan frame yang merupakan *subsequent* dari suatu video yaitu *motion*. *Motion* merupakan pergerakan atau arah pergerakan yang terlihat dilakukan objek atau yang terjadi terhadap objek akibat pergantian frame.

Terdapat dua tipe metode yang terdapat dalam teknik *tracking* dan *motion* ini yaitu *Dense Tracking Techniques* dan *Sparse Tracking Techniques*. Dalam penelitian ini metode yang dipakai dalam teknik *tracking* dan *motion* adalah *Phyramidal Lucas Kanade* yang termasuk kedalam *Sparse Tracking Techniques*.

II.1.3 Threshold

Threshold adalah *image processing* dimana melakukan pengolahan pixel pada suatu citra atau menghilangkan beberapa pixel dan juga mempertahankan beberapa pixel sehingga menghasilkan suatu citra baru hasil sortir pixel yang telah dilakukan. Dengan dilakukannya komputasi *threshold* maka dapat dengan mudah mendapatkan *edge*/tepi dari suatu citra.

Dalam *computer vision, threshold* dilakukan agar mempermudah dalam proses identifikasi ataupun perbandingan dari dua atau lebih citra. Terdapat 5 tipe threshold yaitu *binary threshold, binary inverted, truncate, threshold to zero dan threshold to zero inverted.*



Gambar II.2 Tipe-tipe Threshold

Dalam melakukan *threshold* sendiri dibutuhkan citra dalam bentuk 8 bit dan 2 *channel* atau *grayscale*. Setelah itu citra *grayscale* ini diruba menjadi bi-level (2 bit) atau *black-and-white.* Berikut adalah fungsi *threshold* terhadap suatu citra pada persamaan (1):

$$t(x,y) = \begin{cases} A\big(p(x,y)\big), f\big(x,y,p(x,y)\big) = true \\ B\big(p(x,y)\big), f\big(x,y,p(x,y)\big) = false \end{cases} \quad \text{.....(1)}$$

Fungsi f(x,y,p(x,y)) adalah representasi dari fungsi *pixel velue* (untuk mendapatkan nilai pixel dari suatu kordinat (x,y) lalu membandingkan tiga nilai pixel dari fungsi g, p dan h yaitu g(x,y) < p(x,y) < h(x,y). Maka dalam *threshold* sendiri dasarnya ada dua operasi. Pertama menggunakan fungsi prediksi dari suatu kordinat pada suatu citra yaitu sumbu g(x,y) dan h(x,y) yang konstan di lakukan di dalam citra. Lalu kedua melakukan perbandingan dari hasil perbandingan sebelumnya dengan fungsi h(x,y) (berfungsi untuk mengextrak daerah *brightness* dan *contrast*).

Hasil *threshold* dari suatu image biasanya akan dipakai dalam berbagai teknik dalam computer vision seperti pencarian *contour*, penggambaran *contour* dan *tracking* atau *motion.*

II.1.4 Segmentasi Citra

Segmentation merupakan teknik untuk memisahkan atau mengisolasi suatu objek atau sebagian dari objek dari suatu *image* (memisahkan foreground dengan *background*). Banyak situasi dalam *computer vision* yang membutuhkan segmentasi atau memisahkan *foreground* atau suatu objek terhadap backgroundnya guna melihat aktifitas *pixel* dari objek yang objek yang ingin kita lihat. Misalkan ketika suatu kamera keamanan dilengkapi pendeteksi pergerakan manusia atau objek lainnya yang menyerupai manusia dan beberapa binaang, maka kamera tersebut harus terlebih dahulu dapat melakukan segmentasi terhadap objek manusia ataupun binatang tertentu.

Segmentasi biasa dilakukan dengan maksud agar pencarian *pixel* tidak terlalu memakan waktu dan *memory*, sehingga ketika terjadi komputasi pencarian tidak memerlukan pencarian secara menyeluruh pada suatu citra, cukup pencarian pada daerah yang biasa disebut *region of interest* sehingga dapat menghemat waktu dan *memory*. Pencarian yang dimaksud biasanya berupa proses *tracking* dan *motion* atau pergerakan suatu pixel. Komputasi pendeteksian *motion* ini akan terjadi hanya pada setiap point yang telah di segmentasi atau bisa disebut pada objek yang tela dispesifikan.

Ada banyak metoda atau algoritma yang dapat digunakan dalam segmentasi bahkan dengan algoritma *image processing* biasa seperti *morphology*, *flood fill*, *threshold*, dan *pyramid* akan tetapi terdapat algoritma tersendiri yang dapat menghasilkan segmentasi yang cukup baik terutama dalam segmentasi kulit manusia yang dipakai dalam penelitian ini.

II.1.4.1 Background Subtraction

Background Subtraction merupakan operasi yang sangat fundamental dalam *image processing*. Biasanya dilakukan pembelajaran terhadap suatu *background* (biasanya dilakukan pada suatu model *background*. Suatu *background model* akan di *compare* atau di bandingkan terhadap citra yang dimiliki, sehingga dapat membedakan mana *background* dan mana *foreground*.

Background Subtraction menggunakan dua model dengan tingkat warna atau yang biasa dipakai adalah *brightness* tinggi dan rendah. Dua model yang memiliki nilai warna tinggi yang biasa disebut dengan model maximum dan yang

rendah disebut dengan model minimum. Model maksimum dan minimum digunakan untuk mengeliminasi background, dengan membandingkan setiap nilai pixel yang ada dan menggantinya dengan 0 dan 1 sehingga menghasilkan citra 1 *channel* dengan citra yang mirip dengan hasil threshold. Jika *pixel* pada suatu citra berada di antara range maximum dan minimum makan akan diganti dengan nilai 1 sedangkan jika tidak akan diberikan nilai 0.

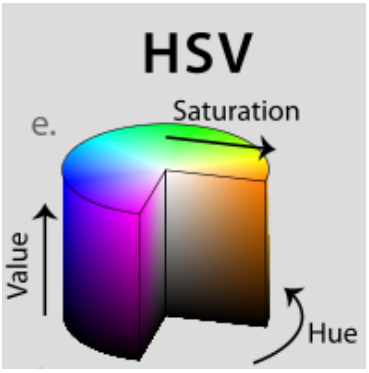
Namun ada dua cara yang dapat digunakan yaitu dengan menggunakan nilai range warna ataupun dengan menggunakan citra *background* yang normal (hanya ada *background* yang nanti dipakai dalam citra). Dengan menggunakan citra *background* biasanya dengan membandingkan citra *background* maximum dan minimum (biasanya menggunakan *brightness*) dan melakukan normalisasi citra sehingga dapat dilakukan pembandingan dengan citra yang ingin dicari objeknya atau citra yang ingin dikenakan segmentasi.

Sedangkan dengan menggunakan warna biasanya menggunakan warna maximum atau minimum dengan format *YCrCb* ataupun *HSV*. Penggunaan warna *YCrCb* atau *HSV* bertujuan agar lebih mudah dalam membandingkan warna, dengan menggunakan warna 1 channel. Seperti *HSV, H* pada warna *HSV* merupakan Hue yang terdiri dari satu nilai dari warna RGB, sehingga memudahkan dalam mendeteksi range warna.

II.1.5 HSV Color *Image*

HSV singkatan *hue*, *saturasi*, dan *value*, dan juga sering disebut HSB (B untuk kecerahan). Sebuah model ketiga, yang umum dalam aplikasi visi

komputer, HSI, untuk *hue*, saturasi, dan intensitas. Akan tetapi, sementara definisi ini tidak standar, dan setiap singkatan ini dapat digunakan untuk salah satu model lainnya tiga atau beberapa silinder yang terkait (selinder warna seperti RGB, HSL, YCrCb, dan lain-lain).



Gambar II.3 Selinder warna HSV

Dalam masing-masing silinder, sudut sekitar sumbu vertikal pusat sesuai dengan "*hue*", jarak dari sumbu sesuai dengan "*saturation*", dan jarak sepanjang sumbu sesuai dengan "*lightness*", "*value*" atau "*brightness*". Karena HSV transformasi sederhana dari perangkat-tergantung model RGB, warna fisik mereka menentukan tergantung pada warna primary merah, hijau, dan biru dari perangkat atau dari ruang RGB tertentu, dan di koreksi gamma digunakan untuk mewakili jumlah mereka primary. Nilai numerik HSV menggambarkan nilai-nilai warna yang berbeda untuk setiap ruang dasar RGB.

Kedua representasi digunakan secara luas dalam komputer grafis, dan satu atau yang lain dari mereka sering lebih mudah daripada RGB, tetapi HSV pun dikritik karena tidak cukup memisahkan warna membuat atribut, atau kurangnya keseragaman persepsi.

Berikut adalah konversi nilai RGB ke dalam HSV :

Dalam kasus ini citra dalam bentuk 8-bit dan 16-bit. R, G and B yang di konversi berformat floating-point dan dengan nilai dari 0 hingga 1.

$$V = \max(R, G, B) \quad \text{.....(2)}$$

$$S = \begin{cases} \frac{V - \min(R,G,B)}{V} & if\ V \neq 0 \\ 0 & otherwise \end{cases} \quad \text{.....(3)}$$

$$H = \begin{cases} 60(G-B)/S & if\ V = R \\ 120 - 60(B-R)/S & if\ V = G \\ 240 - 60(R-G)/S & if\ V = B \end{cases} \quad \text{.....(4)}$$

if H < 0 then H = H + 360

output-nya adalah 0 ≤V ≤ 1, 0 ≤ S ≤ 1, 0 ≤ H ≤ 360.

Lalu nilai dikonversi ke data type tujuan :

8-bit images

$$\boldsymbol{V = 255V, S = 255S, H = H/2 (to\ fit\ to\ 0\ to\ 255)}$$

16-bit images (currently not supported)

$$\boldsymbol{V < -65535V, S < -65535S, H < -H}$$

32-bit images

H, S, V yang dihasilkan sudah 32-bit

II.1.6 YCrCb Color *Image*

YCbCr atau Y'CbCr, kadang-kadang ditulis YCbCr atau Y'CBCR, adalah keluarga ruang warna yang digunakan sebagai bagian dari pipa warna gambar

dalam video dan sistem fotografi digital. Y adalah komponen luma dan CB dan CR adalah perbedaan biru dan merah perbedaan komponen kroma. Y '(dengan prima) dibedakan dari Y yang terang, yang berarti bahwa intensitas cahaya adalah non-linear dikodekan menggunakan koreksi gamma.

Y'CbCr bukan merupakan ruang warna mutlak, tetapi merupakan cara pengkodean informasi RGB. Warna yang sebenarnya ditampilkan tergantung pada primary RGB yang sebenarnya digunakan untuk menampilkan sinyal. Oleh karena itu nilai dinyatakan sebagai Y'CbCr diprediksi hanya jika standar RGB kromatisitas warna primer yang digunakan.

Berikut adalah konversi warna RGB ke dalam YCrCb :

$$Y = 0.2999 \cdot R + 587 \cdot G + 114 \cdot B \quad \text{.....(5)}$$

$$Cr = (R - Y) \cdot 0.713 + delta \quad \text{.....(6)}$$

$$Cb = (B - Y) \cdot 0.564 + delta \quad \text{.....(7)}$$

$$R = Y + 1.403 \cdot (Cr + delta) \quad \text{.....(8)}$$

$$G = Y - 0.344 \cdot (Cr - delta) - 0.714 \cdot (Cb - delta) \quad \text{.....(9)}$$

$$B = Y + 1.773 \cdot (Cb - delta) \quad \text{.....(10)}$$

Dimana

$$delta = \begin{cases} 128 & \text{Untuk Citra 8 bit} \\ 32768 & \text{Untuk Citra 16 bit} \\ 0.5 & \text{Untuk Citra floating point} \end{cases}$$

Y, Cr dan Cb telah meliputi seluruh value range.

Pada gambar II.4 merupakan contoh citra yang memiliki format warna YCrCb, hasil konversi dari format warna RGB ke YCrCb.



Gambar II.4 Contoh dari citra dengan format warna YCrCb

II.1.7 Grayscale Image

Dalam sistem yang dibangun, banyak dilakukan konversi citra dari berbagai format warna seperti RGB, BGR, RGBA, YCrCb, HSV menjadi *grayscale*. Tujuannya adalah untuk melakukan pencarian batas atau *edge* atau tepi citra lalu menggambarkan tepi-tepi tersebut sehingga komputer akan dapat dengan mudah melihat bentuk atau kontur-kontur yang terdapat dengan citra. Setelah komputer dapat melihat bentuk tersebut maka komputer dapat dengan mudah melakukan berbagai proses komputasi terutama yang berhubungan dengan *recognizing, tracking,* dan *motion* pada suatu gambar bergerak baik itu dari *disk* atau pun *real-time* dengan sumber *input* menggunakan kamera atau *web-cam*.

Dalam sistem ini digunakan fungsi untuk melakukan konversi warna citra menjadi *grayscale* seperti pada persamaan (11):

$$Y = (0.299)R + (0.587)G + (0.114)B \quad \ldots\ldots(11)$$

II.1.8 Kontur

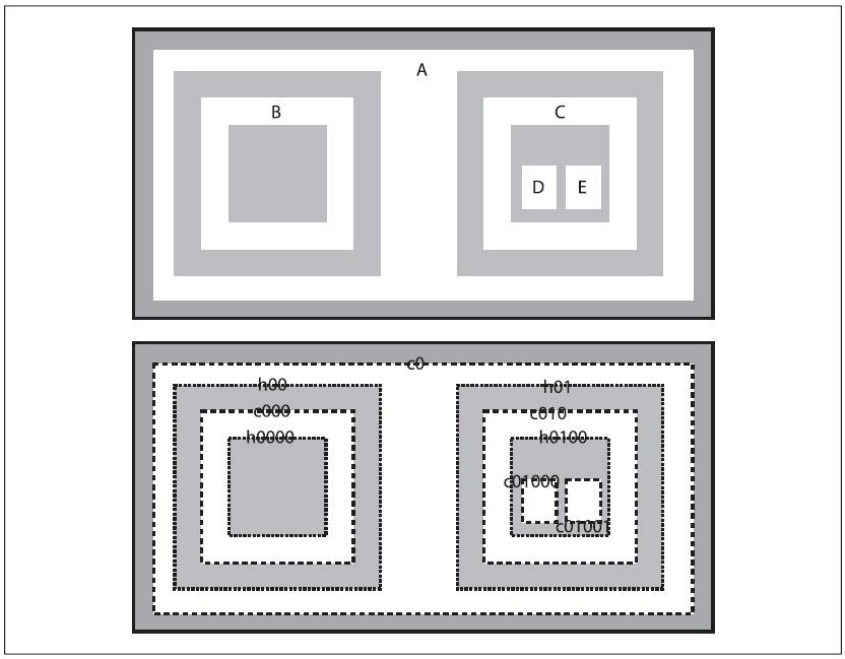
Kontur adalah suatu kumpulan poinatau titik yang didapatkan dari suatu komputasi atau perhitungan yang mewakili bentuk dari suatu batas objek pada suatu citra yang nantinya dihubungkan dengan garis dari setiap titik atau point yang telah didapat, sehingga menghasilkan kontur dari objek tersebut.

Kontur biasa dipakai untuk melihat garis-garis batas dari suatu objek, mirip dengan tepi (*edge*) hanya saja kontur lebih detil, tegas dan jelas. Misalkan melakukan pencarian kontur terhadap suatu citra untuk melihat apakan ada kontur yang berbentuk seperti sebuah mobil dala suatu citra.

Kontur didalam *openCV* membutuhkan suatu tempat memory khusus untuk menampung sequence yang berisikan titik atau point-point yang ditemukan. Maka di dalam *openCV* dibuat suatu struktur data yang merupakan linkedlist yang sequential tersendiri untuk menampung titik-titik kontur tersebut yang disebut dengan sequence atau disingkat cvSeq (*computer vision Sequential*).

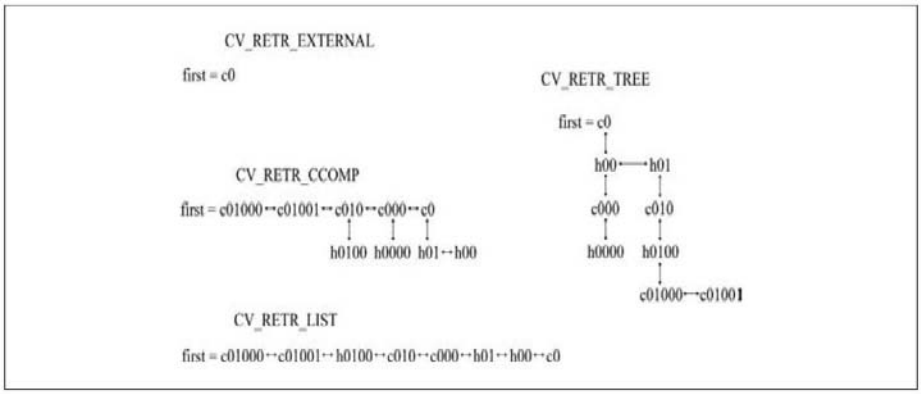
Dalam pembuatan kontur diperlukan rutin untuk melakukan proses pencarian kontur. Citra yang dipakai adalah citra hasil konversi menjadi *grayscale* dan dikenakan *threshold* sehingga menghasilkan citra *bi-level*. Dalam gambar II.5 dapat terlihat dari ilustrai pencarian kontur. Terdapat image/citra/gambar yang memiliki daerah putih dan abu-abu (dengan tingkat ketebalan yang berbeda) dan merupakan background, daerah putih diberi label *A* sampai *E* dengan background

gelap. Di bagian gambar paling bawah terdapat hasil dari rutin pencarian kontur dan diberi label *cX* dan *hX* (*X* merupakan nilai biner iterator). Untuk label *cX* merupakan kontur dari setiap objek yang ada pada gambar sedangkan label *hX* merupakan hole atau lubang yang nantinya merupakan bagian yang dilingkari oleh kontur, yang didalam gambar kontur di perlihatkan sebagai garis yang putus-putus.



Gambar II.5 Contoh test pencarian kontur pada suatu citra

Dalam pencarian kontur ada 4 metode yang dapat digunakan dalam pengambilan konturpada suatu citra yaitu *Retrieval External,Retrieval Components, Retriecal List dan Retrieval Tree.*



Gambar II.6 Bentuk Tipe Contour

1. *Retrieval External*

   Pengambilan kontur yang didasarkan hanya pada kontur-kontur yang terlihat jelas dan berada di bagian luar objek yang ada dalam citra.

2. *Retrieval List*

   Pengambilan seluruh kontur dan menyimpannya di dalam *list*. Tersimpan sebagai *linkedlist*.

3. *Retrieval Components*

   Pengambilan seluruh kontur dan menyimpannya pada dua level hirarki dimana hirarki paling atas/*top level boundaris* (kontur yang paling luar yang melingkari *hole*) adalah *external boundaries* dari komponen sedangkan level kedua adalah boundaris dari *hole*.

4. *Retrieval Tree*

Pengambilan seluruh kontur dan merekonstruksikannya ke dalam hirarki (*full herarchy*) menjadi kontur yang bersarang. Seperti pada gambar II-6.

II.1.8.1 Convexty dan Convexty Defects

Dalam menggambar *shape* atau suatu bentuk dalam suatu kontur yang sudah ditemukan dapat menggunakan metode *Convexity* and *Convexity defects*. Metode ini memberikan pemahaman pada komputer mengenai bentuk dari kontur yang ditemukan dengan memanfaatkan ketidak tepatan penggambaran shape yang mengelilingi objek, metode ini bukan hanya dapat mendapatkan karakteristik dari tangan tapi juga posisi tangan.

Convexity Defect berfungsi sebagai suatu metode untuk mendapatkan kontur yang berkarakter kuat dan pasti pada suatu objek misalkan tangan. sedangkan *convex hull* biasa dipakai untuk mendapatkan kontur besar yang mengelilingi suatu objek.

Convexity dan *convexity defects* menggambarkan kontur dengan dua metode yaitu convex hull dan *convexity defects*. *Convexity* adalah rasio perimeter *convexity-hull* , $\pi_{convex\ hull}$ ke perimeter batas bentuk, π_{shape} , mana convex hull adalah poligon convex minimum yang mencakup bentuk.

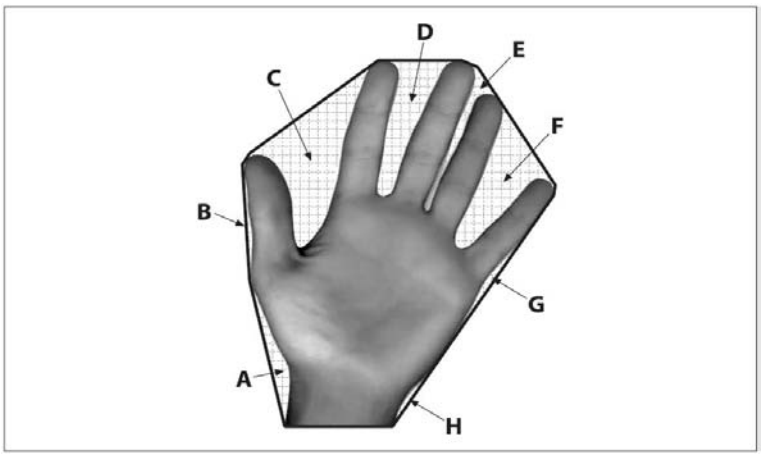
$$\varsigma = \frac{\pi_{convex\ hull}}{\pi_{shape}} \quad \text{.....(12)}$$

Sehingga dengan menggunakan convex hull kita akan mendapatkan bentuk boundary dari tangan dengan batas-batas minimal.

Sedangkan untuk mendapatkan kontur atau menggambarkan kontur tepat mengelilingi kontur tangan dapat menggunakan metode **convexity defects.** Berikut adalah persamaan yang dipakai dalam opencv sesuai dengan *advances visual computing 5th symposium*:

$$\theta = arccos\left(\frac{v_1.v_2}{|v_1||v_2|}\right) \quad \text{.....(13)}$$

Dimana v1,v2 adalah masing-masing vektor antara awal dan titik akhir pertama dan kedua *convexity defects*. Jikaangle lebih rendah dari threshold terkecil,makav1 dan v2 dari persamaan *convexity defects* tersebut dianggap sebagai kandidat yang mungkin untuk penyempitan (menjadi titik baru), selain itumaka *convexity defects* tidak dapat dipakai. Berikut contoh *convex hull* dan *convexity defects*.

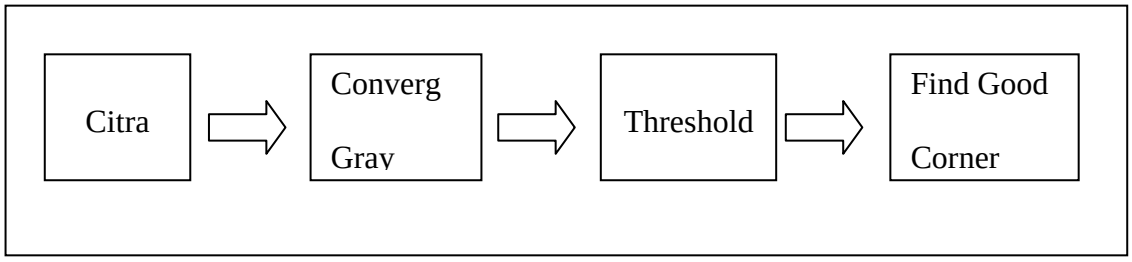


Gambar II.7 Implementasi Convexity dan Convexity Defect pada citra tangan manusia

Convexity defect terlihat merupakan garis yang tepat mengelilingi tangan dan merupakan shape atau bentuk yang minimalis, sedangkan garis kontur gelap adalah convex hull sekitar tangan.

II.1.9 Corner Finding

Corner finding adalah teknik yang dipakai untuk melakukan pencarian terhadap setiap sisidari suatu citra dan mencari sudut yang baik untuk dapat dipakai sebagai acuan untuk melakukan tracking terhadap corner tersebut pada setiap *frame*. Sehingga dalam prosesnya biasanya *corner finding* dilakukan terhadap terhadap citra yang sudah dikonversi menjadi 2 *channel* dan sudah dilakukan proses *threshold* guna menemukan seluruh *edge* yang ada dalam citra tersebut.



Gambar II.8 Proses *Corner Finding*

Ada banyak corak yang dapat di lacak. Sangat penting ketika memperhitungkan seperti apa sesungguhnya corak atau objek yang di lacak, seperti halnya ketika menentukan sebuah titik pada suatu citra dalam sebuah frame *sequent* dalam sebuah video lalau mencari titik yang sama pada frame selanjutnya dengan corak yang sama dengan corak titik sebelumnya. Terkadang corak dari objek yang dipilih pada citra sebelumnya cenderung bersifat unik sehingga pada frame berikutnya sulit menemuka titik yang mirip sekali coraknya dengan objek yang ada pada frame lainnnya.

Untuk mendeteksi suatu *corner* dapat menggunakan definisi yang dibuat oleh *Haris* [Haris88] yang menggunakan matriks dari turunan (∂2x , ∂2y , ∂x ∂y) yang dihasilkan dari intensitas citra. Turunan dari matriks suatu citra dan mengambil semua hasil turunan lalu menggunakannya untuk membuat citra yang baru atau biasa disebut dengan *Hessian Image*. Terminologi dari *Hessian Matrix* dalam mengolah point tersebut dengan menggunakan matriks dua dimensi, dan berikut persamaannya pada persamaan (15).

$$H(p) = \begin{bmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} \\ \frac{\partial^2 I}{\partial y \partial x} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \text{.....(14)}$$

Untuk menemukan *corner* dengan menggunakan definisi *Haris* atau biasa disebut dengan *HarrisCorner*, dengan korelasi turunan kedua matriks dari suatu citra dengan jendela/*window* yang di dalamnya terdapat *point-point* dapat menggunakan persamaan matriks persamaan (4).

$$M(X,Y) = \begin{bmatrix} \sum_{-K \le i,j \le K} w_{i,j} I_x^2(x+i, y+j) & \sum_{-K \le i,j \le K} w_{i,j} I_x^2(x+i, y+j) I_y(x+i, y+j) \\ \sum_{-K \le i,j \le K} w_{i,j} I_x^2(x+i, y+j) I_y(x+i, y+j) & \sum_{-K \le i,j \le K} w_{i,j} I_x^2(x+i, y+j) \end{bmatrix}$$

(15)

(disini wi,j adalah *weigh* yang memiliki nilai yang sama tetapi ini biasa digunakan untuk membuat tampilan jendela yang circular).

Definisi *Harris* pada dasarnya menggunakan nilai determinan fungsi H(p), atau substract dari H(p) (dengan nilai konstanta *weight*), lalu melakukan perbandingan dengan citra yang nilai matriksnya belum dilakukan determinan

treshold. Pada definisi lain disebutkan selain apa yang telah dilakukan dalam definisi *Harris* bahwa pencarian *corner* nilai dua eigen merupakan nilai terbesar dari nilai *treshold*.

II.1.10 Lucas Kanade

Dalam *computer vision,* metode *Lucas-Kanade* adalah metode diferensial banyak digunakan untuk estimasi *optical flow* yang dikembangkan oleh *Bruce D. Lucas* dan *Takeo Kanade.* Ini mengasumsikan bahwa *aliran/flow* pada dasarnya konstan/tetap di lingkungan lokal dari pixel yang dipertimbangkan, dan memecahkan persamaan aliran dasar optik untuk semua piksel di lingkungan itu, dengan kriteria kuadrat terkecil.

Dengan menggabungkan informasi dari beberapa piksel terdekat, metode *Lucas-Kanade* sering dapat menyelesaikan ambiguitas yang melekat pada persamaan aliran optik metode *point-wise*.

Metode Lucas-Kanade mengasumsikan bahwa perpindahan dari isi gambar antara dua instants dekat (*frame*) yang kecil dan sekitar konstan dalam lingkungan dari *p* titik dalam pertimbangan. Jadi persamaan aliran optik dapat dianggap berlaku untuk semua piksel dalam jendela berpusat di *p*. Yakni, aliran gambar lokal (kecepatan) vektor (Vx, Vy) harus memenuhi persamaan (5).

$$I_x(q_1)V_x + I_y(q_1)V_y = -I_t(q_1) \quad \text{……..……(16)}$$
$$I_x(q_2)V_x + I_y(q_2)V_y = -I_t(q_2)$$
$$\vdots$$
$$I_x(q_n)V_x + I_y(q_n)V_y = -I_t(q_n)$$

Mana$q_1, q_2, \ldots, q_n$ adalah piksel dalam jendela, dan $I_x(q_i)$,$I_y(q_i)$,$I_t(q_i)$ adalah derivatif parsial dari gambar sehubungan dengan posisi x, y dan waktu t, dievaluasi pada titik qi dan pada waktu saat ini. Persamaan ini dapat ditulis dalam bentuk matriks Av = b, di mana

$$A = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix},\ v = \begin{bmatrix} V_x \\ V_y \end{bmatrix},\ and\ b = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_{1n}) \end{bmatrix} \text{.....(17)}$$

Sistem ini memiliki persamaan lebih dari yang diketahui dan dengan demikian biasanya over-diterminan. Metode Lucas-Kanade mendapatkan solusi dengan prinsip kuadrat terkecil. Yaitu, memecahkan sistem 2 × 2

$A^TAv = A^Tb$ or

$v = (A^TA)^{-1}A^Tb$

dimana A^T adalah matriks transpos *A*, dengan perhitungan :

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_x(q_i)I_y(q_i) \\ \sum_i I_x(q_i)I_y(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix} \text{.....(18)}$$

dengan jumlah berjalan dari i = 1 sampai n.ATA matriks sering disebut tensor struktur gambar pada titik p.

II.1.11 Algoritma Pyramidal Lucas Kanade

Pada dasarnya algoritma *phyramidal lucas kanade* merupakan pengembangan dari algoritma *lucas kanade*. Metoda ini di adaptasikan dengan metode *phyramidal* guna untuk memberikan kemampuan untuk memperluas

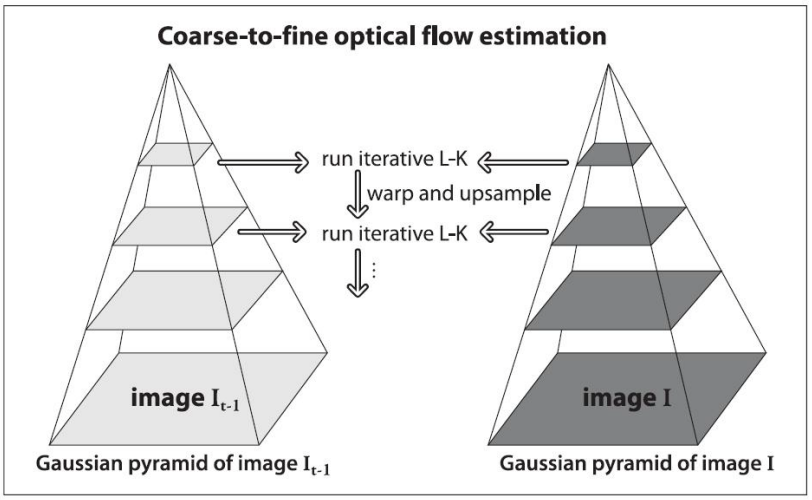
pendeteksian gerakan agar gerakan-gerakan yang besar dalam *windows*(jendela yang membaca gerakan input dari kamera *web*) yang kecil. Misalkan ada gerakan objek ke kanan akan mudah terdeteksi dibandingkan dengan gerakan objek yang menyerong ke kanan dan kebawah secara bersamaan di depan kamera dengan *windows* yang berukuran kecil tidak akan bias terdeteksi.

Pada dasarnya metode lucas kanade terdiri dari tiga asumsi :

1. *Brightness Constancy* : *brightness* dari sebuah pixel dari sebuah citra dalam sebuah layar tidak berubah (bentuk dalm sebuah *grayscale*).
2. *Temporalpersistenceor* "*small movements*" : gerakan citra bergerak secara perlahan. Ini berarti kecepatan tergantung dari skala perpindahan objek yang ada dalam *frame* tidak berpindah terlalu banyak.
3. *Spatial coherence* : point kordinat yang ada pada tampilan citra yang sama memiliki kemiripan objek.

Hanya saja dengan asumsi di atas gerakan besar yang terjadi tidak dapat dideteksi, sedangkan salah satu solusinya adalah menggunakan *windows* yang besar, akan tetapi ini mengakibatkan adanya gerakan yang tidak koheren. Algoritma *Pyramidal Lucas-Kanade* menyelesaikan permasalahan tersebut tanpa menghilangkan asumsi gerakan yang koheren.

Pendekatan ini menggunakan prinsip piramida, yaitu bekerja dari detil citra paling rendah hingga detil citra paling tinggi.



Gambar II.9 Pyramidal Lucas Kanade

Algoritma *Pyramidal Lucas-Kanade* pertama bekerja pada layer piramida paling tinggi. Kemudian hasilnya digunakan sebagai titik awal untuk bekerja pada layer dibawahnya. Hal ini berlanjut hingga mencapai level paling rendah. Berdasarkan makalah yang diajukan oleh Bouquet (Bouquet, 2000), algoritma *Pyramidal Lucas-Kanade* dapat dijelaskan sebagai berikut: Misalkan *I* dan *J* adalah dua citra *gray scale* 2D. Dua nilai *I(x) = I(x;y)* dan *J(x) = J(x;y)* adalah nilai gray scale dari dua citra pada lokasi $x = [x;y]_T$, dengan *x* dan *y* merupakan koordinat dua pixel dari point citra generik *x*. Citra *I* akan ireferensikan sebagai citra pertama, dan citra *J* sebagai citra kedua[2].

Anggap sebuah titik $\mathbf{u} = [u_x\ u_y]_T$ pada citra pertama *I*. Tujuan dari *tracking fiture* adalah untuk menemukan lokasi $\mathbf{v} = \mathbf{u} + \mathbf{d} = [u_x + d_x\ u_y + d_y]_T$ pada citra kedua *J* dengan *I*(**u**) dan *J*(**v**) adalah serupa. Vektor $\mathbf{d} = [d_x\ d_y]_T$ merupakan kecepatan citra pada *x*, juga diketahui sebagai aliran optis pada *x*. Hal ini penting untuk mendefinisikan dugaan kesamaan dalam ketetanggaan 2D. Misalkan wxdan wy_y

dua integer yang mendefinisikan setengah ukuran dari window integrasi di atas *x*. Didefinisikan kecepatan citra **d** sebagai vektor yang meminimasi fungsi yang didefinisikan sebagai persamaan (8)[1].

$$\varepsilon(d) = \varepsilon(d_x, d_x) \quad \text{.....(19)}$$

Sedangkan untuk summary algoritma *pyramidal tracking* dengan *lucas kanade* secara matematis terdiri dari persamaan-persamaan matematik yang diambil berdasarkan journal nomor [3] pada daftar pustaka, berikut summary algoritma *pyramidal lucas kanade*:

(*Goal* : jadikan **u** menjadi titik kordinat awal pada gambar. menemukanlokasi**v** yang sesuaipadagambar **J)**

Build *pyramid representations of* $\boldsymbol{I}$ *and* $\boldsymbol{J}$:

$$\{I^L\}_{L=0,.,L_m} \ and \ \{J^L\}_{L=0,.,L_m} \quad \text{.....(20)}$$

Initiation *of pyramidal guess*: $g^{L_m} = [g^{L_m}\, g^{L_m}]^T = [0\ 0]^T$ .....(21)

For $L = L_m$ **down to 0 with step of -1** (or until $\|\bar{\eta}^k\|$< accuracy threshold)

Location of point $\boldsymbol{u}$ *on image* I^l : $\boldsymbol{u}^L = [p_x\ p_y]^T = \boldsymbol{u}/2^L$ .....(22)

Derivative of I^L *with respect to x* : $I_x(x,y) = \frac{I^L(x+1,y) - I^L(x-1,y)}{2}$ ....(23)

Derivative of I^L *with respect to y* : $I_y(x,y) = \frac{I^L(x,y+1) - I^L(x,y-1)}{2}$ ...(24)

Spatial gradient matrix :

$$G = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \begin{bmatrix} I_x^2(x,y) & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_x^2(x,y) \end{bmatrix} \text{.....(25)}$$

*Initialization*dari iterasi L-K : $\overline{v^0} = [0 \;\; 0]^T$ .....(26)

For k = 1 **to**K **with step of** 1 (or until $\|\bar{\eta}^k\|$< accuracy threshold

Image difference :

$$\delta I_k(x,y) = I^L(x,y) - J^L(x + g_x^L + v_x^{k-1}, y + g_y^L + v_y^{k-1}) \text{ .....(27)}$$

Image mismatch vector :

$$\bar{b}_k = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{x=p_y-w_y}^{p_y+w_y} \begin{bmatrix} \delta I_k(x,y) I_x(x,y) \\ \delta I_k(x,y) I_y(x,y) \end{bmatrix} \text{.....(28)}$$

Optical flow (Lucas-Kanade) : $\bar{\eta}^k = G^{-1}\bar{b}_k$ .....(29)

Guess for next iteration : $\bar{v}^k = \bar{v}^{k-1} + \bar{\eta}^k$ .....(30)

End of for-loop on k

Final optical at level L : $\boldsymbol{d}^L = \bar{v}^K$ .....(31)

Gues for next level L-1 :

$$\boldsymbol{g}^{L-1} = \begin{bmatrix} g_x^{L-1} & g_y^{L-1} \end{bmatrix}^T = 2(\boldsymbol{g}^L + d^L) \text{ ..(32)}$$

End for for-loop onk

Final optical flow vector : $\boldsymbol{d} = \boldsymbol{g^0} + \boldsymbol{d^0}$.....(33)

Location of point on J : **v = u + d**.....(34)

Solution : titik yang dicari atau di tracking adalah titik **v** pada citra **J**

Dan **psedeocode** yang di ambil dari algoritma di atas adalah sebagai berikut :

```
#
build pyramid representations of I and JILm, JLm
Initialization of pyramidal guess gLm
for L = Lm to 0
     uL =location of u on ILm = u/2^L
     Ix = derivative of IL wrt x
     Iy = derivative of IL wrt y
     G  = spatial gradient matrix
     init iterative L-K (guess V0)
     for k = 1 to K or until optical flow < treshold
          Vk = iterative L-K loop estimation
     end for
     final optical flow at level L dL = Vk
     calculate guess for next level gLm-1
end for
final optical flow vector V = g0 + d0
return v = u +d
#
#
```

Sehingga akhirnya ditemukan titik perubahan adalah **v** dan dengan vector **V.**

II.2 OpenCV (Open Computer Vision)

OpenCV merupakan *framework* untuk membantu dalam membangun sistem yang menggunakan teknologi Computer Vision yang *support* terhadap *Intel Image Processing Library (IPL).* OpenCV di desain untuk mengefisiensikan komputasi dengan focus terhadap aplikasi yang bersifat *real-time*. OpenCV dibangun oleh Intel pertamkali dikembangkan tahun 1999 dan versi alpha yang direalease ke public *IEEE Conference on Computer Vision and Pattern Recognition* pada tahun 2000, pada pertengahan tahun 2008 disupport oleh *Willow Garage* dan pada tahun 2009 di release OpenCv versi 2.

II.2.1 IPLImage dan Math

OpenCV memiliki beberapa tipe data primitif. Jenis data tidak primitif dari sudut pandang C, tetapi mereka semua struktur sederhana, dan opencv akan menganggap mereka sebagai atom. Tipe data dasar tersebut dapat diperiksa dalam rincian struktur yang diuraikan dalam header cxtypes.h file (dan juga lainnya struktur), yang ada di *.../OpenCV/cxcore/* include direktori dari *OpenCV* menginstal.

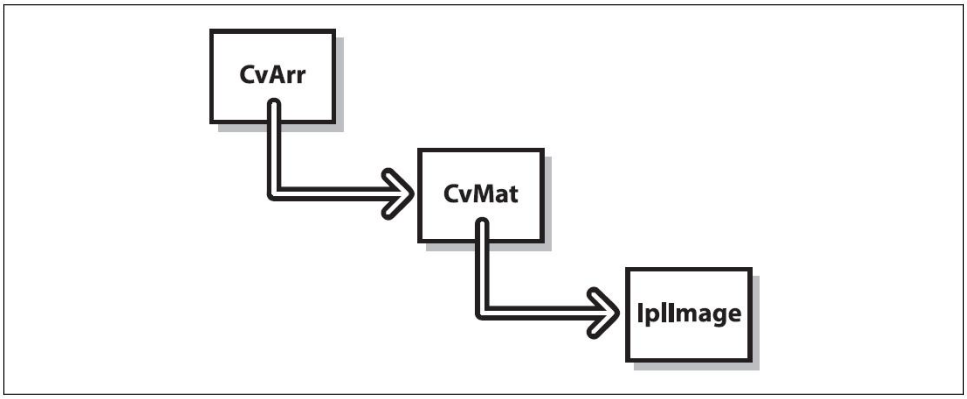
Tipe sederhana jenis ini adalah *CvPoint*. *CvPoint* adalah struktur sederhana dengan dua bilangan bulat anggota, x dan y. *CvPoint* memiliki dua saudara kandung: *CvPoint2D32f* dan *CvPoint3D32f*. Tipe data *CvPoint2D32f* dan *CvPoint3D32f* memiliki dua anggota yang sama x dan y, yang keduanya *floating-point*. Terakhir juga mengandung elemen ketiga, z.

CvSize lebih mirip sepupu *CvPoint*. Anggota-anggotanya adalah lebar dan tinggi, yangkedua bilangan bulat. Jika Anda ingin memiliki *CvSize* dengan tipe data purba dalam C/C++ floating-point, gunakan *CvSize2D32f* sepupu *CvSize*.*CvRect* merupakan anak/turunan dari *CvPoint* dan *CvSize*; itu berisi empat anggota: x, y, lebar, dan tinggi. Terakhir namun tidak kalah pentingnya adalah *CvScalar*, yang merupakan satu set empat nomor ganda-presisi. Ketika memori tidak memiliki masalah (mencukupi), *CvScalar* sering digunakan untuk mewakili satu, dua, atau tiga bilangan real. *CvScalar* memiliki anggota tunggal val, yang merupakan pointer ke array yang berisi empat ganda presisi floating-titik angka. Semua tipe data memiliki metode konstruktor dengan nama seperti *cvSize()*

(umumnya * konstruktor mempunyai nama yang sama dengan tipe struktur tetapi dengan karakter pertama tidak dikapitalisasi). Ingatlah bahwa ini adalah C dan tidak C++, jadi ini "konstruktor" hanya fungsi inline yang mengambil daftar argumen dan mengembalikan struktur yang diinginkan dengan nilai ditetapkan dengan benar.

II.2.1.1 *Matrix* dan Tipe-tipe *Image*

Dalam GambarII.10 menunjukan hirarki class atau struktur dari ketiga jenis foto. Dalam penggunaan *OpenCV*, tipe data *IplImage* adalah struktur dasar yang digunakan untuk mengkodekan suatu citra. Gambar dimungkinkan memiliki format *grayscale*, warna, empat-*channel* (RGB +*alpha*), dan masing-masing saluran mungkin berisi salah satu dari beberapa jenis *integer*atau *floating-point*. Oleh karena itu, jenis ini lebih *general* dari pada tiga saluran dimana citra memiliki 8-bit RGB. *OpenCV* menyediakan banyak operator yang berguna yang bertindak atas suatu gambar, termasuk alat untuk mengubah ukuran gambar, ekstrak saluran individu, mencar inilai terbesar atau terkecil saluran tertentu, tambahkan dua foto, ambang gambar, dan sebagainya.



Gambar II.10 Meskipun *OpenCV* diimplementasikan dalam C, struktur yang digunakan dalam *OpenCV* memiliki

Ada dua hal yang perlu diketahui sebelum terjun kedalam pengolahan matriks. Pertama, tidak ada "vektor" dalam *OpenCV*. Setiap kali terdapat vektor, *OpenCV* hanya menggunakan matriks dengan satu kolom (atau satu baris, jika ingin berpindah atau vektor *conjugate*). Kedua, konsep matriks di OpenCV agak lebih abstrak dari konsep yang dipelajari dalam aljabar linear pada mata kuliah biasanya. Secara khusus, unsur-unsur matriks tidak perlu diri mereka nomor sederhana. Sebagai contoh, rutin yang menciptakan matriks dua dimensi baru memiliki prototipe sebagai berikut:

```
cvMat * cvCreateMat (int baris, int cols, tipe int);
```

Di sini salah satu jenis dari daftar panjang jenis *predefined* bisa dalam bentuk: *CV_<bit_depth>(S|U|F)C<number_of_channels>*. Matriks dalam *OpenCV* dapat terdiri dari 32-bit floating-point (*CV_32FC1*), dari unsigned *integer* 8-bit kembar tiga (*CV_8UC3*), atau unsur-unsur lain yang tak terhitung jumlahnya. Sebuah elemen dari *CvMat* belum tentu satu hanya memiliki satu nilai di dalamnya sehingga dapat mewakili beberapa nilai untuk entri tunggal dalam

matriks memungkinkan untuk melakukan hal-hal seperti merupakan saluran beberapa warna dalam sebuah gambar RGB. Untuk citra sederhana yang berisi jalur merah, hijau dan biru, operator citra akan diterapkan untuk setiap channel secara terpisah (kecuali jika dinyatakan lain).

II.2.1.2 Struktur Data IPlImage

Pada intinya obyek ini merupakan *CvMat* tetapi dengan beberapa variabel tambahan di dalamnya untuk membuat matriks ditafsirkan sebagai gambar. Struktur ini awalnya didefinisikan sebagai bagian dari Intel Image.

Beberapa hal mungkin kecil tidak teralalu signifikan untuk di pelajari, tetapi banyak yang sangat penting untuk memahami bagaimana OpenCV menafsirkan sesuatu dan bagaiman OpenCV bekerja dengan gambar. Pada *struct* di atas, terdapat variabel lebar (*width*) dan tinggi (*height*), kedalaman (*depth*) dan *Channels* dimana nilai tersebut nantinya diambil berdasarkan informasi yang didapat dari suatau citra yang berbentuk matriks. Variabel kedalaman mengambil salah satu set nilai yang didefinisikan dalam *ipl.h*, dan nilai-nilai ini tidak persis seperti yang dilihat ketika melihat matriks. Hal ini karena untuk foto atau image cenderung untuk berurusan dengan kedalaman dan jumlah channel terpisah (sedangkan dimatrik yang sering digunakan cenderung untuk merujuk secara bersamaan). Kedalaman yang mungkin dipakai dalam tipe data *image* yang dapat digunakan Tabel2-1.

Tabel II-1 Tabel kedalaman tipe data iplimage yang disediakan

Macro Image pixel type	Image pixel type
IPL_DEPTH_8U	Unsigned 8-bit integer (8u)
IPL_DEPTH_8S	Signed 8-bit integer (8s)
IPL_DEPTH_16S	Signed 16-bit integer (16s)
IPL_DEPTH_32S	Signed 32-bit integer (32s)
IPL_DEPTH_32F	32-bit fl oating-point single-precision (32f)
IPL_DEPTH_64F	64-bit fl oating-point double-precision (64f)

II.2.2 Lucas kanade Dalam OpenCV

Rutin yang mengimplementasikan algoritma *Lucas-Kanade nonpyramidal* pada *optical-flow*adalah:

```
void cvCalcOpticalFlowLK (
     const CvArr * Imga,
     const CvArr * imgB,
     CvSize winSize,
     CvArr * velx,
     CvArr * vely
        );
```

Gambar II.11 Gambar kedalaman tipe data iplimage yang disediakan

Hasilnya array untuk OpenCV rutin akan diisi hanya oleh piksel untuk yang mampu menghitung kesalahan minimum. Untuk pixel yang error ini (dan dengan demikian perpindahan) tidak dapat diandalkan dihitung, kecepatan terkait akan ditetapkan ke 0. Dalam kebanyakan kasus, Anda tidak akan ingin menggunakan rutin ini. Metode berbasis piramida lebih baik untuk situasi paling banyak memakan waktu.

II.2.2.1 Piramida Lucas-Kanade Dalam OpenCV

Untuk algoritma dalam *OpenCV* yang menghitung *Lucas-Kanade flowoptic* dalam piramida, adalah `cvCalcOpticalFlowPyrLK()`. Fungsi ini *flow optic* memanfaatkan *"good fitur to track"* dan juga melakukan indikasi kembali seberapa baik masing-masing pelacakan point melanjutkan.

```
void cvCalcOpticalFlowPyrLK (
        const CvArr * Imga,
        const CvArr * imgB,
        CvArr * Pyra,
        CvArr * pyrB,
        CvPoint2D32f * featuresA,
        CvPoint2D32f * featuresB,
        int hitung,
        CvSize winSize,
        int level,
        status char *,
        float * track_error,
        CvTermCriteria kriteria,
        int flag
        );
```

Gambar II.12 Rutin Pyramidal Lucas Kanade Optical Flow

Fungsi ini memiliki banyak masukan, dua argument pertama `cvCalcOpticalFlowPyrLK ()` adalah gambar awal dan akhir; keduanya harus citra *single-channel*, 8-bit. Kedua argumen berikutnya adalah *buffer* yang dialokasikan untuk menyimpan gambar piramida. Ukuran dari *buffer* harus setidaknya (*img.width+ 8*) *img.height* * / 3 byte, * dengan satu *buffer* tersebut untuk masing-masing dari dua gambar input (*PyrA* dan *pyrB*). Jika kedua pointer diatur ke *NULL* maka rutin akan mengalokasikan, penggunaan, dan tidak perlu mengalokasikan memori yang sesuai saat dipanggil, tetapi ini tidak begitu baik

dalm hal kinerja. F*eaturesAarray* berisi poin yang gerak tersebut dapat ditemukan, dan *featuresB* adalah *array* yang sama ke lokasi yang baru dihitung dari titik-titik dari *featuresA* harus ditempatkan, hitung adalah jumlah titik dalam daftar *featuresA*. Jendela yang digunakan untuk menghitung gerakan koheren lokal diberikan oleh *winSize*. Karena membangun gambar piramida, tingkat argumen yang digunakan untuk mengatur kedalaman tumpukan gambar.

Jika tingkat diatur ke 0 maka piramida tidak digunakan. Status array hitung panjang; pada saat penyelesaian rutin, setiap entri dalam status akan baik 1 (jika yang sesuai titik ditemukan pada gambar kedua) atau 0 (jika tidak). Parameter *track_error* adalah opsional dan dapat dimatikan dengan menetapkan ke NULL. Jika *track_error* aktif maka akan berbentuk array angka, satu untuk setiap titik dilacak, sama dengan *erence diff* antara patch sekitar titik dilacak pada gambar terlebih dulu, patch di sekitar lokasi yang titik yang dilacak pada gambar kedua. Anda dapat menggunakan *track_error* untuk memangkas pergi poin yang lokal perubahan terlalu banyak patch tampilan sebagai memindahkan titik.

Untuk menghasilkan struktur yang butuhkan biasanya menggunakan `cvTermCriteria()` berfungsi. Argumen pertama fungsi ini adalah `CV_TERMCRIT_ITER` atau `CV_TERMCRIT_EPS,` yang menentukan algoritma yang dipilih untuk menentukan setelah beberapa jumlah iterasi atau ketika konvergensi metrik mencapai beberapa nilai kecil (masing-masing). Kedua argumen berikutnya menetapkan nilai-nilai di mana satu, yang lain, atau kedua kriteria ini harus mengakhiri algoritma.

Alasan memiliki kedua pilihan adalah agar dapat menetapkan jenis untuk `CV_TERMCRIT_ITER | CV_TERMCRIT_EPS` dan dengan demikian rutin memiliki kondisi berhenti dalam proses yang memerlukan perulangan(ini adalah apa yang dilakukan di sebagian besar nyata code). Akhirnya, bendera memungkinkan untuk beberapa kontrol halus dari pembukuan internal rutin itu, mungkin ditetapkan untuk setiap atau semua (menggunakan OR bitwise) dari berikut.

1. *CV_LKFLOW_PYR_A_READY* Gambar piramida untuk frame terlebih Fdulu dihitung sebelum panggilan tersebut dan disimpan dalam Pyra.
2. *CV_LKFLOW_PYR_B_READY* Gambar piramida untuk frame kedua dihitung sebelum panggilan tersebut dan disimpan dalam pyrB.
3. *CV_LKFLOW_INITIAL_GUESSES* Array B sudah berisi menebak awal untuk fitur koordinat saat rutin disebut.

Flag ini sangat berguna saat menangani video berurutan. Gambar piramida agak mahal untuk menghitung (penggunaan *memory*), sehingga recomputing harus dihindari sebisa mungkin. Frame akhir untuk pasangan frame sebelumnya (source awal) akan dihitung menjadi bingkai awal untuk pasangan akan dighitung berikutnya. Jika rutin diberi informasikan bahwa informasi ini sudah dihitung maka tidak akan menghitung ulang (*recomputing*). Demikian pula, jika menghitung gerak suatu titik dari frame sebelumnya maka menjadi posisi yang

baik untuk membuat awal yang baik tebakan untuk frame berikutnya. Jadi rencana dasar yang sederhana: menyertakan foto, daftar poin yang ingin dilacak di *featuresA,* dan panggilan rutin. Ketika rutin memberikan return value, periksa status array untuk melihat poin berhasil dilacak dan kemudian periksa *featuresB* untuk menemukan yang baru lokasi titik-titik. Hal ini membawa kembali ke masalah bagaimana untuk memilih mana fituryang baik untuk melacak. Sebelumnya OpenCV memniliki rutin *cvGoodFeaturesToTrack* (), yang awalnya menggunakan metode yang diusulkan oleh Shi dan Tomasi untuk memecahkan ini masalah dengan cara yang dapat diandalkan. Dalam kebanyakan kasus, hasil yang baik yang diperoleh dengan menggunakan kombinasi dari *cvGoodFeaturesToTrack*() dan *cvCalcOpticalFlowPyrLK* (). Tentu saja dapat juga menggunakan kriteria sendiri untuk menentukan poin untuk melacak.

II.3 C/C++

Tahun 1969, laboratorium Belll AT&T di *Muray Hill*, *New Jersey* menggunakan bahasa *assembly* ini untuk mengembangkan sistem operasi UNIX. Maksudnya adalah untuk membuat sistem operasi yang dapat bersifat *programmer-friendly.* Setelah UNIX berjalan, Ken Thompson, seorang pengembang sistem di laboratorium tersebut mengembangkan bahasa baru dengan nama bahasa B. huruf B ini diambil BCPL. Bahasa B ini kemudian digunakan untuk menulis ulang atau merevisi sistem operasi UNIX. Oleh karena bahasa B masih bersifat interpret dan lambat, maka pada tahun 1971, sistem operasi UNIX kemudian ditulis ulang dengan menggunakan bahasa C, yaitu bahasa

pemrograman yang dikembangkan oleh *Dennis Ritchie*, seorang pengembang sistem di laboratorium yang sama.

Sampai sekarang basaha C masih digunakan untuk melakukan pengembangan-pengembangan program dan siste-sistem operasi, diantaranya sistem operasi *Windows*. Alasan itulah yang menjadikan bahasa C sangat populer di dunia pemrograman, khususnya untuk industri perangkat lunak.

Bjarne Stroustrup pada *Bell labs* pertama kali mengembangkan C++ pada awal 1980-an, Untuk mendukung fitur-fitur pada C++, dibangun efisiensi dan sistem support untuk pemrograman tingkat rendah (low level coding). Pada C++ ditambahkan konsep-konsep baru seperti class dengan sifat-sifatnya seperti inheritance dan overloading. Salah satu perbedaan yang paling mendasar dengan bahasa C adalah dukungan terhadap konsep pemrograman berorientasi objek (*Object Oriented Programming*).

Perbedaan Antara Bahasa pemrograman C dan C++ meskipun bahasa-bahasa tersebut menggunakan sintaks yang sama tetapi merekakeduanya memiliki perbedaan, C merupakan bahasa pemrograman prosedural, dimana penyelesaian suatu masalah dilakukan dengan membagi-bagi masalah tersebut kedalam su-submasalah yang lebih kecil, Selain itu, C++ merupakan bahasa pemrograman yang memiliki sifat Pemrograman berorientasi objek, Untuk menyelesaikan masalah, C++ melakukan langkah pertama dengan menjelaskan class-class yang merupakan anak class yang dibuat sebelumnya sebagai abstraksi dari *object-object* fisik, *Class* tersebut berisi keadaan *object*, anggota-anggotanya dan

kemampuan dari objectnya, Setelah beberapa *Class* dibuat kemudian masalah dipecahkan dengan *Class*.

II.4 Eclipse Helios

Eclipse adalah sebuah IDE (*Integrated Development Environment*) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua platform (*platform-independent*). Berikut ini adalah sifat dari Eclipse:

1. ***Multi-platform***: Target sistem operasi Eclipse adalah Microsoft Windows, Linux, Solaris, AIX, HP-UX, dan Mac OS X.
2. ***Mulit-language***: Eclipse dikembangkan dengan bahasa pemrograman Java, akan tetapi Eclipse mendukung pengembangan aplikasi berbasis bahasa pemrograman lainnya, seperti C/C++, Cobol, Phyton, Perl, PHP, dan lain sebagainya.
3. ***Multi-role***: Selain sebagai IDE untuk pengembangan aplikasi, Eclipse pun bisa digunakan untuk aktivitas dalam siklus pengembangan perangkat lunak, seperti dokumentasi, test perangkat lunak, pengembangan web, dan lain sebagainya.

Eclipse pada saat ini merupakan salah satu IDE favorit dikarenakan gratis dan *open source*, yang berarti setiap orang boleh melihat kode pemrograman perangkat lunak ini. Selain itu, kelebihan dari Eclipse yang membuatnya populer adalah kemampuannya untuk dapat dikembangkan oleh pengguna dengan komponen yang dinamakan.

II.5 CDT (C Development Tool)

CDT adalah suatu project yang berisikan seluruh fungsional C dan C++ untuk mengintegrasikannya dengan *Development Environtment* Eclipse. Fasilitas yang dimasukan kedalam eclipse adalah support untuk pembuatan project dan memanage hasil *build*untuk berbagai macam peralatan, mendukung menggunakan '*make*' *build,* navigasi *source,* dan berbagai peralatan pendukung untuk *source knowloedge,* seperti hirarki, *call graph,include browser, macro definition browser, code editor* dengan *syntax highlighting*, *folding* dan *hyperlink*, navigasi, *source code refactoring* dan *code generation, visual debugging tools, including memory, register,* dan *disassembly viewers*.

II.6 GCC (GNU Compiler Collection) dan G++

Richard Stallman memulai GCC pada tahun 1985. Dia diperpanjang kompiler yang ada untuk mengkompilasi C. kompilator awalnya dikompilasi Pastel, sebuah dialek, diperpanjang *nonportable* Pascal, dan ditulis dalam Pastel. Ini ditulis ulang di C oleh Len Tower dan Stallman, dan dirilis pada tahun 1987 sebagai compiler untuk Proyek GNU, dalam rangka untuk memiliki sebuah kompiler yang tersedia yang perangkat lunak bebas. Pengembangannya dibawah oleh *Free Software Foundation* (FSF). Pada 1991, 1.x GCC telah mencapai titik stabilitas, tetapi keterbatasan arsitektur dicegah banyak perbaikan yang diinginkan, sehingga FSF mulai bekerja pada GCC 2.x.

Sebagai GCC merupakan perangkat lunak bebas, programer yang ingin bekerja di arah lain-terutama yang menulis antarmuka untuk bahasa lain selain C-

bebas untuk mengembangkan garpu mereka sendiri compiler. Beberapa garpu terbukti tidak efisien dan berat, namun, dan kesulitan dalam mendapatkan pekerjaan yang diterima oleh proyek GCC resmi sangat frustrasi bagi banyak orang. FSF terus kontrol yang dekat seperti pada apa yang telah ditambahkan ke versi resmi dari GCC GCC 2.x yang digunakan sebagai salah satu contoh dari model "katedral" pembangunan di esai s Eric S. Raymond 'The Cathedral dan Bazaar. Dengan rilis 4.4BSD pada tahun 1994, GCC menjadi compiler default untuk sistem BSD.

GNU Compiler Collection (GCC) adalah suatu sistem *compiler* yang dihasilkan oleh Proyek GNU mendukung berbagai bahasa pemrograman. GCC adalah komponen kunci dari GNU toolchain. Selain sebagai compiler resmi dari sistem operasi GNU belum selesai, GCC telah diadopsi sebagai compiler standar oleh sebagian besar sistem komputer modern lain operasi mirip Unix, termasuk Linux, keluarga BSD dan Mac OS X.

GCC telah porting ke berbagai arsitektur prosesor, dan secara luas digunakan sebagai alat dalam lingkungan pengembangan perangkat lunak sumber komersial, eksklusif dan tertutup. GCC juga tersedia untuk platform tertanam kebanyakan, misalnya Symbian, AMCC dan Power Freescale Arsitektur berbasis chip Compiler. Dapat menargetkan berbagai platform, termasuk videogame konsol seperti PlayStation 2 dan Dreamcast. Beberapa perusahaan membuat bisnis keluar dari penyediaan dan mendukung port GCC untuk berbagai platform, dan

produsen chip hari ini mempertimbangkan sebuah port GCC hampir penting bagi keberhasilan arsitektur.

Awalnya bernama *GNU C Compiler*, karena hanya menangani bahasa C pemrograman, GCC 1.0 dirilis pada tahun 1987, dan *compiler* ini diperpanjang untuk mengkompilasi C + + di bulan Desember tahun itu Front berakhir itu. Kemudian dikembangkan untuk Fortran, Pascal, Objective-C, Java, dan Ada, antara lain.

Free Software Foundation (FSF) mendistribusikan GCC di bawah GNU General Public License (GNU GPL). GCC telah memainkan peranan penting dalam pertumbuhan perangkat lunak bebas, baik sebagai alat dan contoh.

G++ merupakan prangkat lunak yang berfungsi sebagai compiler C++, sama halnya dengan GCC, G++ berlisensi open source dan biasa di pakai dalam sistem operasi berbasis linux. Dan G++ maupun GCC dapat di intgrasikan dengan IDE lain termasuk dengan eclipse dengan menggunakan bantuan *toolkit* yang bernama CDT.

II.7 QT Framework

Qt adalah aplikasi *cross-platform* kerangka yang banyak digunakan untuk mengembangkan perangkat lunak atau aplikasi dengan antarmuka pengguna grafis (GUI) (Qt disebut juga sebagai widget toolkit), dan juga digunakan untuk mengembangkan program-program non-GUI seperti tool baris perintah dan konsol untuk server. Qt yang pun telah digunakan dalam pengembangan perangkat lunak pada Autodesk Maya, Dassault DraftSight, Google Earth, KDE,

Adobe Photoshop Album, European Space Agency, OPIE, Siemens, Volvo, Walt Disney Animation Studios, Skype, VLC media player, Samsung, Philips, Panasonic, VirtualBox dan Mathematica.

Hal ini dihasilkan oleh Nokia divisi Pengembangan Qt Frameworks, yang muncu setelah Nokia meng-akuisisi perusahaan Trolltech Norwegia, produsen asli dari Qt. Nokia mengumumkan bahwa itu akan menjatuhkan teknologi Symbian dan basis smartphone masa depan mereka pada Microsoft platform bukannya pada Februari 2011, menimbulkan pertanyaan tentang nasib Qt's. Satu bulan kemudian Nokia mengumumkan akuisisi lisensi komersial Qt dan layanan profesional dengan Digia PLC, meskipun Nokia akan tetap menjadi kekuatan pengembangan utama di balik kerangka kerja.

Qt menggunakan standar C + + tetapi membuat ekstensif menggunakan generator kode khusus (disebut Meta Object Compiler, atau moc) bersama dengan beberapa macro untuk memperkaya bahasa. Qt juga dapat digunakan dalam beberapa bahasa pemrograman lain melalui bindings bahasa. Ia berjalan di semua platform utama dan memiliki dukungan internasionalisasi yang luas. Non-GUI fitur termasuk akses SQL database, parsing XML, manajemen benang, dukungan jaringan, dan terpadu lintas-platform API untuk menangani file.

Didistribusikan menurut ketentuan dari GNU Lesser General Public License (antara lain), Qt adalah perangkat lunak bebas dan open source. Semua edisi mendukung berbagai kompiler, termasuk kompiler GCC C + + dan Visual Studio suite.

II.8 UML

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

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

Saat ini UML paling banyak digunakan dengan cara pertama dan kedua. Khusus dalam metode agile (cepat dan ringan), UML digunakan dengan cara pertama.

UML menyediakan 10 macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu:

1. *Use Case Diagram* untuk memodelkan proses bisnis.
2. *Conceptual Diagram* untuk memodelkan konsep-konsep yang ada di dalam aplikasi.
3. *Sequence Diagram* untuk memodelkan pengiriman pesan (message) antar objects.
4. *Collaboration Diagram* untuk memodelkan interaksi antar objects.
5. *State Diagram* untuk memodelkan perilaku objects di dalam sistem.
6. *Activity Diagram* untuk memodelkan perilaku Use Cases dan objects di dalam system.
7. *Class Diagram* untuk memodelkan struktur kelas.
8. *Object Diagram* untuk memodelkan struktur object.
9. *Component Diagram* untuk memodelkan komponen object.
10. *Deployment Diagram* untuk memodelkan distribusi aplikasi.

Berikut akan dijelaskan 4 macam diagram yang paling sering digunakan dalam pembangunan aplikasi berorientasi object, yaitu use case diagram, sequence diagram, collaboration diagram, dan class diagram.

5. Use Case Diagram

Use case diagram digunakan untuk memodelkan bisnis proses berdasarkan perspektif pengguna sistem. *Use case diagram* terdiri atas diagram untuk use *case* dan *actor*. *Actor* merepresentasikan orang yang akan mengoperasikan atau orang yang berinteraksi dengan sistem aplikasi.

*Use cas*e merepresentasikan operasi-operasi yang dilakukan oleh actor. *Use case* digambarkan berbentuk elips dengan nama operasi dituliskan di dalamnya. *Actor* yang melakukan operasi dihubungkan dengan garis lurus ke *use case*.

6. *Sequence Diagram*

Sequence diagram menjelaskan secara detail urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari *use case*: interaksi yang terjadi antar *class*, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi.

7. Collaboration Diagram

Collaboration diagram dipakai untuk memodelkan interaksi antar *object* di dalam sistem. Berbeda dengan sequence diagram yang lebih menonjolkan kronologis dari operasi-operasi yang dilakukan, *collaboration* diagram lebih fokus pada pemahaman atas keseluruhan operasi yang dilakukan oleh *object*.

8. *Class Diagram*

Class diagram merupakan diagram yang selalu ada di permodelan sistem berorientasi objek. *Class diagram* menunjukkan hubungan antar *class* dalam sistem yang sedang dibangun dan bagaimana mereka saling berkolaborasi untuk mencapai suatu tujuan.