BAB 2
LANDASAN TEORI

2.1 *Computer Vision*

Terminologi lain yang berkaitan erat dengan pengolahan citra adalah *Computer Vision* atau *Machine Vision*. Pada hakikatnya nya, *Computer Vision* mencoba meniru cara kerja sistem visual manusia (*Human Vision*). *Human Vision* sesungguhnya sangat kompleks. Manusia melihat objek dengan indra penglihatan (mata), lalu citra objek diteruskan ke otak untuk diinterpretasi sehingga manusia mengerti objek apa yang tampak dalam pandangan matanya. Hasil interpretasi ini memungkinkan digunakan untuk pengambilan keputusan (misalnya menghindar kalau melihat mobil melaju di depan) [10].

Computer vision merupakan kebalikan dari komputer grafik dimana pemahaman komputer terhadap citra (*Image Understanding*) secara AI (*Artificial Intelligence*) atau menganalisis perilaku (*behavior*) / pola citra, sensor untuk robotika dan emulasi komputer dari penglihatan manusia. *Computer vision* berbeda dengan *Human Vision*. Karakteristik *Human Vision* mempunyai ilusi *Adelson Checkerboard*, warna yang konstan (*Color constancy*), ukuran yang konstan (*Size constancy*) dan ilusi thatcher. Penglihatan (*vision*) prestasi terbesar dari kecerdasan alami (*natural intelligence*) manusia. *Vision cortex* menempati sekitar 50% dari bagian otak *Macaque* dan seakan – akan otak manusia dikhususkan untuk menangani urusan *vision*. Penglihatan adalah proses kontruktif dimana persepsi kesadaran dari yang kita lihat adalah ilusi yang dibuat oleh otak manusia (dengan proses yang luar biasa rumit) [3].

Computer vision mempunyai anatomi sebagai berikut :

1. Pengolahan citra digital
    a. *Image acquisition*
    b. *Image enhancement*
    c. *Image restoration*

d. *Morphological processing*
e. *Segmentation*
f. *Object recognition*
g. *Representation and description*
h. *Image compression*
i. *Color image processing*

2. Sistem pencahayaan (*Lightning system*)
    a. *Lighting*
    b. *Staging*
    c. *Lenses*
    d. *Cameras*
3. *Staging*
    a. Parameter – parameter penting dalam sistem pencitraan (*imaging system*)
4. Lensa dan kamera
    a. Transformasi geometri dari 2D menjadi 3D
5. Aplikasi perangkat lunak *vision*
    a. HALCON dari MVTEC
    b. COGNEX

2.2 OpenCV (*Open Source Computer Vision Library*)

OpenCV (*Open Source Computer Vision Library*) adalah sebuah *library* perangkat lunak yang ditujukan untuk pengolahan citra yang biasa digunakan secara *real-time* (pada waktu itu juga). *Library* ini dibuat oleh Intel dan merupakan *library* yang bebas digunakan dan berada dalam naungan sumber terbuka (*Open source*) dari lisensi. *Library* ini juga bisa digunakan diberbagai *platform* dan didedikasikan sebagian besar untuk pengolahan citra secara *real-time*. Umumnya *library* ini menggunakan bahasa pemrograman C/C++, tetapi akhir – akhir ini sudah dikembangkan keberbagai bahasa pemrograman seperti Phyton, Javascript dan Java.

Secara garis besar OpenCV mempunyai modul/subrutin yang ada pada *library* yang akan dijelaskan sebagai berikut [11]:

1. *Core* – sebuah modul/subrutin dasar dari struktur data, sudah termasuk array multi dimensi dan matriks.
2. *Imgproc* – sebuah modul/subrutin untuk pemprosesan citra seperti *image filtering*, *geometrical image*, *image transformation* dan *color space conversion*.
3. *Video* – sebuah modul/subrutin untuk analisis video termasuk *motion*, *background subtraction* dan *object tracking algorithm*.
4. *Calib3d* – sebuah modul/subrutin untuk *geometry algorithm*, kalibrasi kamera dan elemen untuk membangun gambar 3D (3-*Dimensional*).
5. *Features2d* – sebuah modul/subrutin untuk perhitungan konvolusi dan ekstraksi fitur.
6. *Objdetect* – sebuah modul/subrutin untuk deteksi objek dari kelas yang sudah ditentukan.
7. *Highgui* – sebuah modul/subrutin untuk menangkap kamera *webcam* dan *image and video codecs.*
8. *Ml* – sebuah modul/subrutin tambahan untuk melakukan perhitungan *Machine learning* seperti *K-nearest Neighbors*, *Support Vector Machine* dan *Decision tree*.

2.3 Citra Digital

Secara umum, istilah pengolahan citra digital menyatakan "pemrosesan gambar berdimensi-dua melalui komputer digital. Foto adalah contoh gambar berdimensi dua yang dapat diolah dengan mudah. Setiap foto dalam bentuk citra digital (misalnya berasal dari kamera digital) dapat diolah melalui perangkat tertentu. Sebagai contoh, apabila hasil bidikan kamera terlihat agak gelap, citra dapat diolah menjadi lebih terang dimungkinkan pula untuk memisahkan foto orang dari latar belakangnya. Gambaran tersebut menunjukkan hal sederhana yang dapat dilakukan melalui pengolahan citra digital. Tentu saja banyak hal pelik lain yang dapat dilakukan melalui pengolahan citra digital [12].

2.4 Jenis Citra Digital

Ada dua jenis citra yang umum digunakan dalam pemrosesan citra. Kedua jenis citra tersebut yaitu citra berwarna dan citra HSV. Jenis citra berwarna merupakan citra digital biasa yang digunakan dalam kamera *handphone* sedangkan untuk citra HSV (*Hue*, *Saturation* dan *Value*) biasanya digunakan untuk segmentasi pada warna tertentu.

2.4.1 Citra Berwarna

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

Tabel 2.1 Warna dan nilai penyusun warna

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

2.4.2 Citra HSV

HSV mendefinisikan warna dalam terminologi *Hue*, *Saturation* dan *Value*. Keuntungan HSV adalah terdapat warna-warna yang sama dengan yang ditangkap oleh indra manusia. Sedangkan warna yang dibentuk model lain seperti RGB

merupakan hasil campuran dari warna-warna primer. Karakteristik dari ketiga terminologi tersebut adalah sebagai berikut :

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

Berikut ini merupakan perhitungan nilai HSV dari pengubahan citra RGB menjadi citra HSV [13]:

$$r = \frac{R}{(R+G+B)}, g = \frac{G}{(R+G+B)}, b = \frac{B}{(R+G+B)} \quad \text{.....(2.1)}$$

$$V = max(r, g, b) \quad \text{.....(2.2)}$$

$$S = \begin{cases} 0, \ jika\, V = 0 \\ 1 - \frac{\min(r,g,b)}{V}, \ jika\, V > 0 \end{cases} \quad \text{.....(2.3)}$$

$$H = \begin{cases} 0, \ jika\, S = 0 \\ \frac{60*(g-b)}{(S*V)}, \ jika\, V = r \\ 60 * \left[2 + \frac{(b-r)}{(S*V)}\right], \ jika\, V = g \\ 60 * \left[4 + \frac{(r-g)}{(S*V)}\right], \ jika\, V = b \end{cases} \quad \text{.....(2.4)}$$

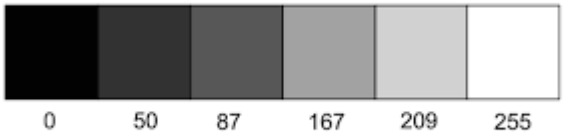
$$H = H + 360, \ jika\, H < 0 \quad \text{.....(2.5)}$$

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

2.4.3 Citra Berskala Keabuan

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

menyatakan putih. Berikut merupakan penjelasan nilai citra skala keabuan terdapat pada Gambar 2.1



Gambar 2.1 Nilai Citra Skala Keabuan

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

$$Grayscale = (0.299 * R) + (0.587 * G) + (0.114 * B)$$ .....(2.6)

2.5 Morfologi Pengolahan Citra

Operasi morfologi merupakan operasi yang digunakan pada citra biner (hitam-putih) untuk mengubah struktur bentuk objek yang terkandung dalam citra [12]. Selain itu terdapat operasi morfologi lainnya yaitu seperti *Erode* (Erosi), *Dilate* (Dilasi), *Opening* (Erosi-Dilasi), *Closing*(Dilasi-Erosi).

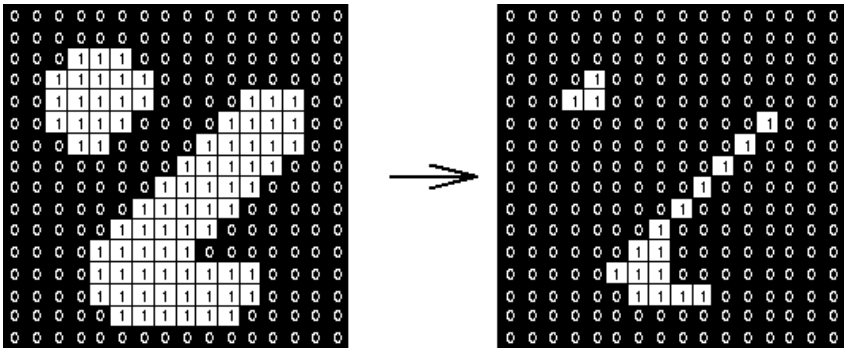
2.5.1 Erode

Operasi erode (Erosi) mempunyai efek memperkecil struktur citra. Operasi erosi dapat dirumuskan sebagai berikut [12]:

$$A \ominus B = \{ p \in z^2 \mid (a + b) \in |, \ untuk\ setiap\ b \in B \}$$.....(2.7)

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

Hasil erosi biasanya merupakan operasi nalar AND dari setiap koordinat A dan B. Berikut merupakan hasil dari operasi erode terdapat pada Gambar 2.2



Gambar 2.2 Hasil Operasi Erode/Erosi

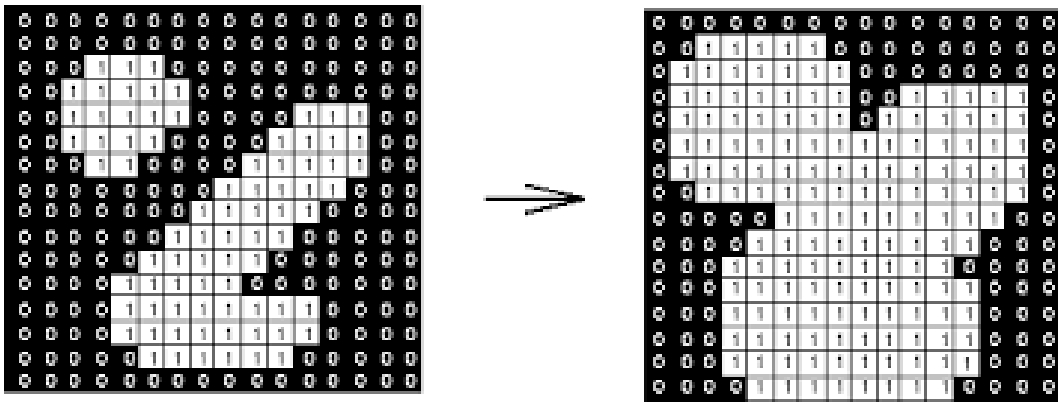
2.5.2 Dilate

Operasi dilate (Dilasi) biasa dipakai untuk mendapatkan efek pelebaran terhadap piksel bernilai 1. Operasi dilasi dapat dirumuskan sebagai berikut [12]:

$$A \oplus B = \{ z \mid z = a + b,\ dengan\ a \in A\ \ dan\ b \in B \} \quad \text{.....(2.8)}$$

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

Hasil dilasi berupa penjumlahan seluruh pasangan koordinat dari A dan B. Berikut merupakan hasil dari operasi dilate terdapat pada Gambar 2.3



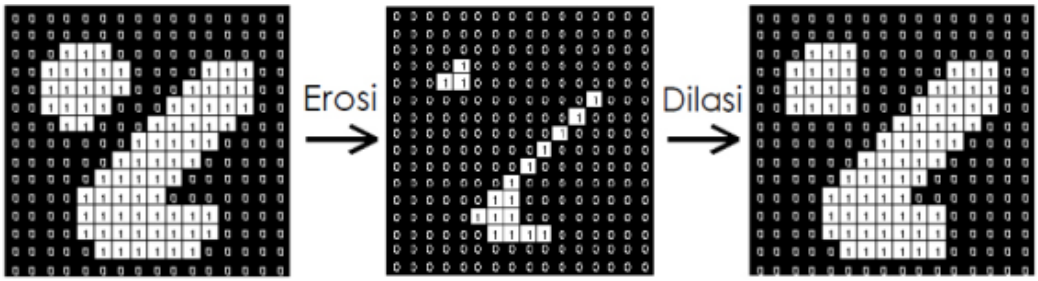
Gambar 2.3 Hasil Operasi Dilate/Dilasi

2.5.3 *Opening* (Erosi-Dilasi)

Operasi *opening* (Erosi-Dilasi) merupakan kombinasi antara operasi erosi dan dilasi yang dilakukan secara berurutan, tetapi citra asli dierosi terlebih dahulu baru kemudian hasilnya didilasi. Operasi ini digunakan untuk memutus bagian-bagian dari objek yang hanya terhubung dengan 1 atau 2 buah titik saja, atau menghilangkan objek – objek kecil yang secara umum membuat *smooth* batas dari objek besar tanpa mengubah area objek secara signifikan. *Opening* adalah *idempotent* yaitu apabila operasi *opening* diulang-ulang tidak akan memberikan dampak yang berkelanjutan. Operasi *opening* dapat dirumuskan sebagai berikut [12]:

$$A \circ B = (A \ominus B) \oplus B \text{.....(2.9)}$$

Berikut merupakan hasil dari operasi *opening* terdapat pada Gambar 2.4



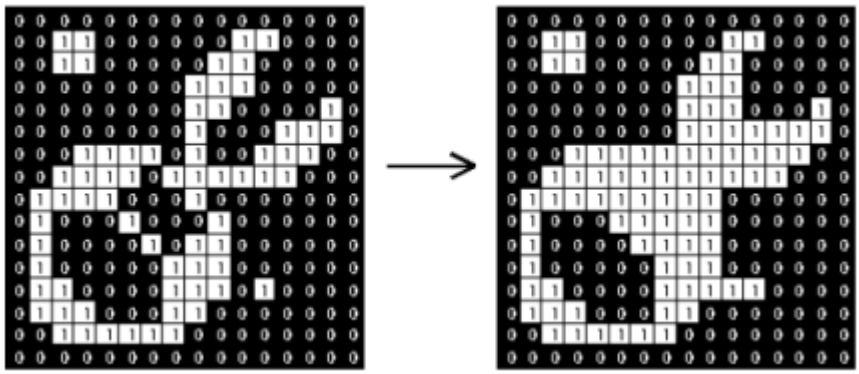
Gambar 2.4 Hasil Operasi *Opening* (Erosi-Dilasi)

2.5.4 *Closing* (Dilasi-Erosi)

Operasi *closing* (Dilasi-Erosi) adalah kombinasi antara operasi dilasi dan erosi yang dilakukan secara berurutan. Citra asli didilasi terlebih dahulu, kemudian hasilnya dierosi. Operasi ini digunakan untuk menutup atau menghilangkan lubang-lubang kecil yang ada dalam segmen objek, menggabungkan objek yang berdekatan dan secara umum membuat *smooth* batas dari objek besar tanpa mengubah objek secara signifikan. Operasi *closing* dapat dirumuskan sebagai berikut [12]:

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

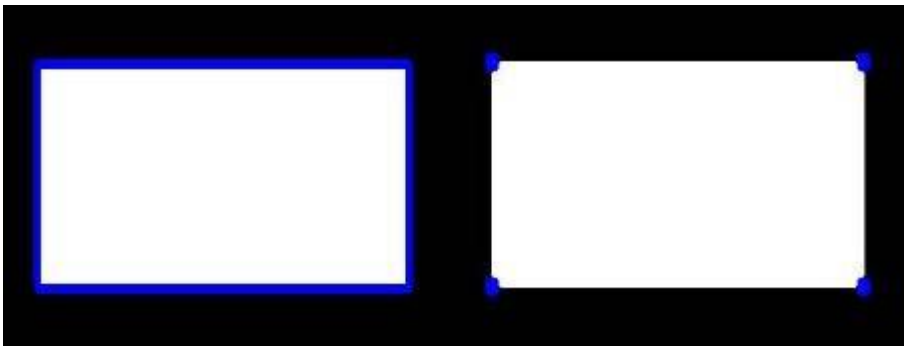
Berikut merupakan hasil dari operasi *closing* terdapat pada Gambar 2.5



Gambar 2.5 Hasil Operasi Erode/Erosi

2.6 *Countours*

Countours bisa dijelaskan lebih simpelnya merupakan gabungan kurva – kurva dari titik yang berkelanjutan (disepanjang daerah) dimana mempunyai warna dan intensitas yang sama. Kontur ini akan berguna sebagai alat untuk mendeteksi, pengenalan dan menganalisis bentuk dari objek [14]. Berikut ini merupakan contoh dari *Contours* dapat dilihat pada Gambar 2.6



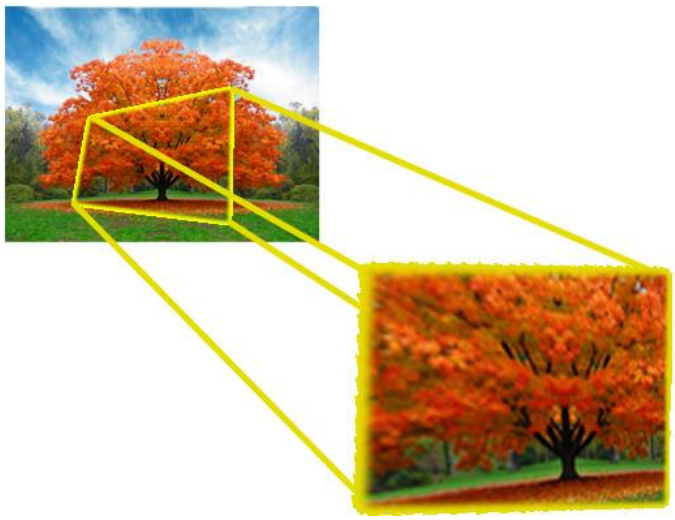
Gambar 2.6 Contoh *Countours*

2.7 Segmentasi Citra

Dalam pengolahan citra, terkadang kita menginginkan pengolahan hanya pada obyek tertentu. Oleh sebab itu, perlu dilakukan proses segmentasi citra yang bertujuan untuk memisahkan antara objek (foreground) dengan *background*. Pada umumnya keluaran hasil segmentasi citra adalah berupa citra biner di mana objek (*foreground*) yang dikehendaki berwarna putih (1), sedangkan *background* yang ingin dihilangkan berwarna hitam (0). Sama halnya

pada proses perbaikan kualitas citra, proses segmentasi citra juga bersifat eksperimental, subjektif, dan bergantung pada tujuan yang hendak dicapai.

Daerah yang kita inginkan tersebut disebut dengan *Region of Interest* (ROI). Proses untuk mendapatkan ROI salah satunya adalah dengan cara melakukan cropping pada suatu citra [15]. Berikut ini merupakan contoh dari *Region of Interest* (ROI) yang terdapat pada Gambar 2.7



Gambar 2.7 Contoh dari *Region of Interest* (ROI)

2.7.1 *Hand Tracking*

Hand tracking adalah sistem untuk mengikuti pergerakan tangan. *Hand tracking* dapat dimanfaatkan untuk berbagai macam hal seperti, alat penggerak mouse, alat kontrol game, dll. Sistem *hand tracking* yang baik adalah secara *real time*, namun yang menjadi permasalahan adalah pemilihan metode yang digunakan agar proses *tracking* dapat dilakukan secara *real rime* dengan akurasi yang sangat baik (> 90%) [16]. Berikut merupakan contoh gambaran *Hand Tracking* terdapat pada Gambar 2.8

Gambar 2.8 Contoh gambaran *Hand Tracking*

2.8 *Scale Invariant Feature Transform*

Pencocokan gambar (*Image Matching*) adalah aspek fundamental dari berbagai masalah yang ada pada *Computer Vision*, termasuk pengenalan objek, membangun gambar 3D dari beberapa gambar, *Stereo correspondence*, dan *motion tracking*. Jumlah dari ekstraksi fitur yang dihasilkan akan dikurangi dengan menyaring pendekatan ini menggunakan operasi yang lebih baik pada saat menetapkan lokasi titik kunci pada gambar. Berikut ini tahapan penting proses komputasi untuk menghasilkan sejumlah ekstraksi fitur pada gambar :

1. *Scale-space extrema detection* : tahapan pertama komputasi untuk mencari semua *scale* dan lokasi titik poin pada gambar. Pada tahapan ini akan menggunakan fungsi *Difference of Gaussian* yang efektif untuk mencari potensial titik kunci pada gambar yang *scale* dan orientasinya beragam (*invariant*).
2. *Keypoint localization* : pada setiap kandidat lokasi, sebuah model akan diterapkan untuk mencari lokasi dan *scale*. Titik kunci didapatkan berdasarkan hasil pengukuran tahapan sebelumnya.
3. *Orientation assignment* : satu atau lebih orientasi akan dihasilkan pada setiap titik kunci berdasarkan lokasi lokal gambar dan arah gradien gambar. Semua operasi dilakukan pada gambar yang dimana relatif sudah bertransformasi.

4. *Keypoint descriptor* : gradien lokal gambar sudah dihitung untuk disetiap area titik kunci untuk kemudian ditebarkan. Tahapan ini menebarkan titik punci untuk mendeskripsikan area gambar mana yang sudah berganti bentuk dan iluminansi nya.

Pendekatan ini dinamakan *Scale Invariant Feature Transform* (SIFT) yaitu mengubah data gambar menjadi koordinat *scale-invariant* terhadap fitur lokal gambar [17].

2.8.1 *Difference of Gaussian*

Untuk menggunakan fungsi *Difference of Gaussian* sebelumnya harus dilakukan proses *Gaussian blur* untuk mengkaburkan sebuah gambar. Secara matematika proses *blurring* ini sebenarnya berdasarkan proses konvolusi pada gambar dengan menggunakan operator gaussian. Berikut ini merupakan rumus untuk perhitungan *Gaussian blur* pada *Diffence of Gaussian* [18]:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \quad (2.11)$$

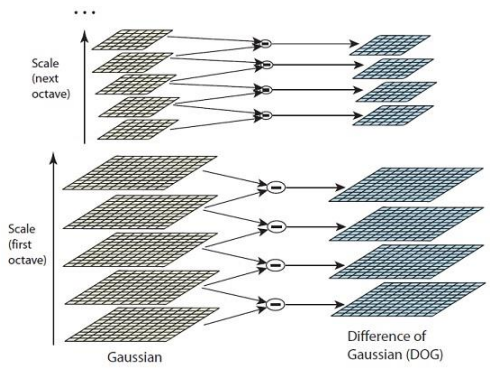
Dimana G merupakan *kernel* dari *Gaussian blur* :

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} \quad (2.12)$$

I merupakan gambar, tanda * adalah proses konvolusi, x dan y merupakan koordinat gambar, σ merupakan ukuran dari kaburnya suatu gambar dan L merupakan hasil proses *blurring*. Selanjutnya untuk menggunakan fungsi *Difference of Gaussian* dapat digunakan rumus untuk perhitungan sebagai berikut :

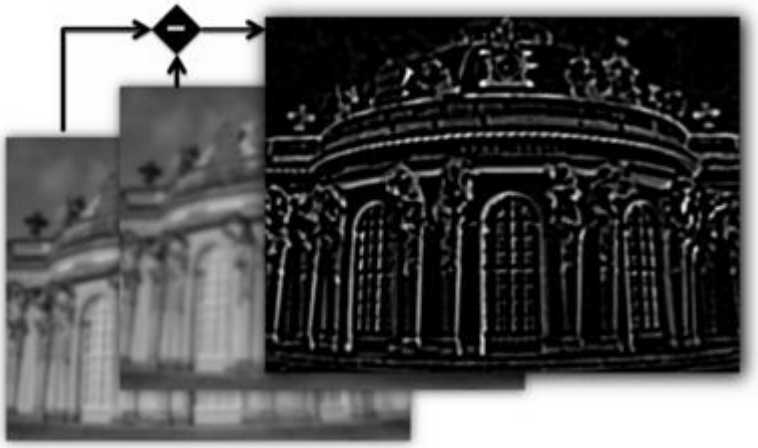
$$DoG = L(x, y, \sigma) - L'(x, y, \sigma) \quad (2.13)$$

Operasi *Difference of Gaussian* akan melakukan duplikasi pada gambar *blur* yang sama untuk dilakukan pengurangan antara gambar *blur* dimana L' merupakan gambar dari hasil *blur* sebelumnya. Berikut merupakan gambaran dari fungsi *Difference of Gaussian* terdapat pada Gambar 2.9



Gambar 2.9 Fungsi *Difference of Gaussian*

Hasil dari *Difference of Gaussian* merupakan deteksi tepi seperti yang terlihat pada Gambar 2.10



Gambar 2.10 Hasil *Difference of Gaussian*

2.9 *Histogram of oriented Gradients*

Histogram of oriented Gradients merupakan cara yang umum dilakukan untuk memperoleh deskriptor untuk deteksi objek tertentu. Sebagai contoh, untuk mendeteksi keberadaan manusia (*Human detection*) seperti penelitian yang pernah dilakukan oleh Dalal dan Trigs [19]. Proses dari algoritma ini dapat dijelaskan sebagai berikut [1] :

1. *Preprocessing*, melakukan intensitas normalisasi atau dengan kata lain mengubah citra gambar menjadi *greyscale* (Citra berskala keabuan).

2. Menghitung *edge map*. Mengestimasi arah x dan y pada gambar lalu menghitung gradien *magnitudes* dan gradien *angle* untuk setiap pixel gambar. Berikut ini merupakan rumus perhitungan untuk gradien *magnitudes* dan gradien *angle* :

   $|G| = \sqrt{I_x^2 + I_y^2}$ .....(2.14)

   Dimana |G| adalah bilangan absolut dari gradien *magnitudes* dan I adalah citra gambar yang sudah *greyscale* dari hasil tahapan sebelumnya. I_x merupakan matrik terhadap sumbu-x dan I_y merupakan matrik terhadap sumbu-y. I_x dan I_y dapat dihitung dengan rumus perhitungan sebagai berikut :

   $I_x = I * D_x \; dan \; I_y = I * D_y$ .....(2.15)

   D_x adalah *mask* $[-1\ 0\ 1]$, sedangkan D_y adalah *mask* $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$ masing – masing dihitung dengan cara konvolusi (tanda *). Kemudian gradien *angle* ke dalam koordinat sumbu dengan sudut diantara 0 sampai 180 dapat dihitung dengan rumus perhitungan sebagai berikut :

   $\theta = \arctan(\frac{I_x}{I_y})$ .....(2.16)

3. *Spatial binning*. Tahapan selanjutnya adalah melakukan perhitungan *histogram* dari gradien *angle* ke tiap-tiap *cell*. Setiap pixel dalam sebuah *cell* mempunyai nilai *histogram* nya sendiri – sendiri berdasarkan nilai yang dihasilkan dalam perhitungan gradien yang kemudian dilakukan normalisasi pada setiap blok. *Cell* memiliki ukuran 8x8 pixel pada sebuah citra. Sedangkan blok memiliki ukuran 2x2 *cell*.

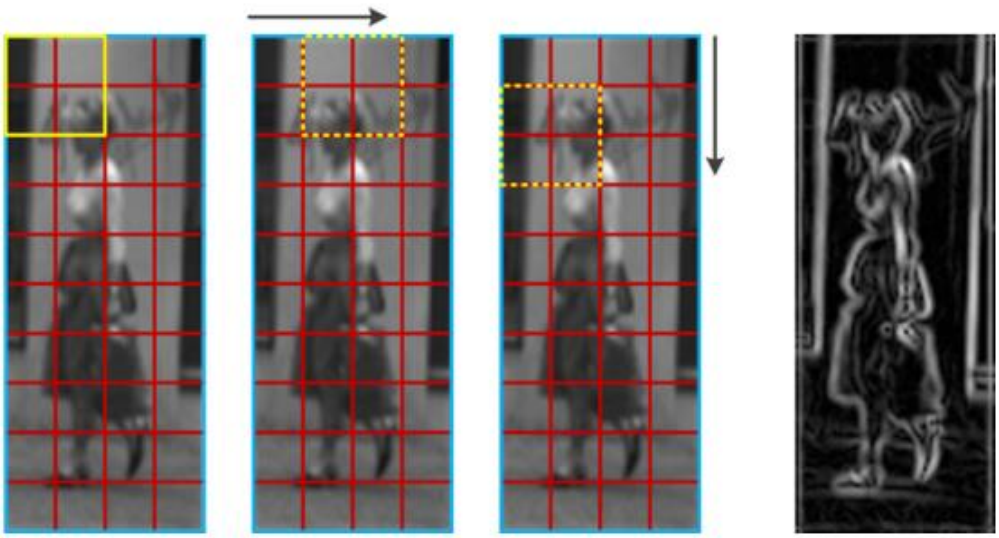
4. *Normalize voting values for generating a descriptor*. Nilai normalisasi fitur blok selanjutnya didapat dengan rumus perhitungan sebagai berikut [20] :

   $norm = \frac{v(n)}{\sqrt{(\sum_{k=1}^{block*l} v(k)^2)+1}}$ .....(2.17)

Nilai *v* merupakan nilai gradien *magnitudes* sedangkan *n* adalah jumlah *bins* dan *block* (2x2 *cell*) merupakan jumlah *cell* sedangkan *l* merupakan jumlah blok yang tidak *overlap.* Fitur blok dinormalisasi untuk mengurangi efek perubahan kecerahan obyek pada satu blok.

5. *Augment all block vectors consecutively*. Setelah fitur blok dinormalisasi, nilai normalisasi setiap blok akan digabungkan menjadi satu vektor (vektor 1 dimensi) hasil satu vektor inilah bisa disebut sebagai fitur vektor *Histogram of oriented Gradients*.

Berikut ini merupakan ilustrasi dari cara kerja *Histogram of oriented Gradients* terdapat pada Gambar 2.11



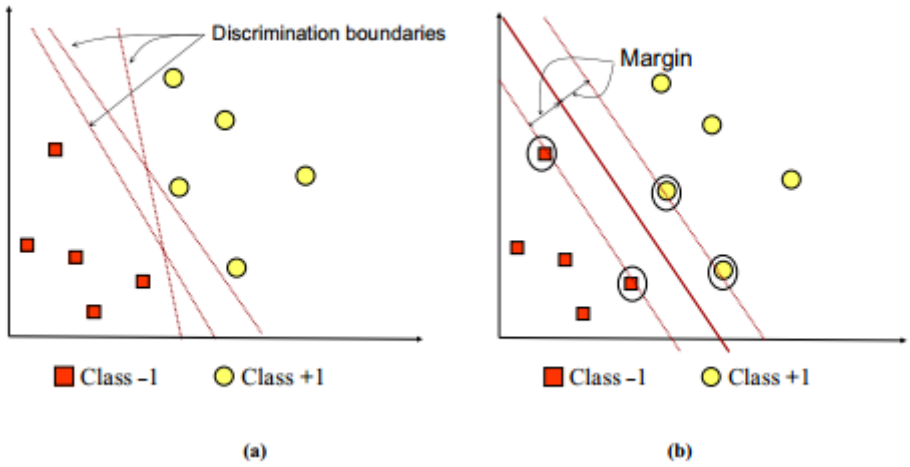
Gambar 2.11 Cara kerja *Histogram of oriented Gradients*

2.10 *Support Vector Machine*

Support Vector Machine (SVM) diperkenalkan oleh Vapnik pada tahun 1992 sebagai suatu teknik klasifikasi yang efisien untuk masalah nonlinier. SVM berbeda dengan teknik klasifikasi di era 1980-an, seperti *decision tree* dan ANN, yang secara konsep kurang begitu jelas dan seringkali terjebak pada optimum lokal. SVM memiliki konsep yang jauh lebih matang, lebih jelas secara matematis, dibanding teknik – teknik klasifikasi sebelum era 1990-an. SVM berusaha menemukan *hyperplane* dengan memaksimalkan jarak antar kelas. Dengan cara ini, SVM dapat

menjadmin kemampuan generalisasi yang tinggi untuk data – data yang akan datang [21].

Menurut penelitian yang dilakukan oleh Anto Satriyo Nugroho, Arief Budi Witarto, dan Dwi Handoko [22] konsep SVM dapat dijelaskan secara sederhana sebagai usaha mencari *hyperplane* terbaik yang berfungsi sebagai pemisah dua buah *class* pada *input space*. Gambar 2.12-a memperlihatkan beberapa *pattern* yang merupakan anggota dari dua buah class : +1 dan –1. *Pattern* yang tergabung pada *class* –1 disimbolkan dengan warna merah (kotak), sedangkan *pattern* pada *class* +1, disimbolkan dengan warna kuning(lingkaran).



Gambar 2.12 SVM mencari *Hyperplane* terbaik

Masalah klasifikasi dapat diterjemahkan dengan usaha menemukan garis (*hyperplane*) yang memisahkan antara kedua kelompok tersebut. Berbagai alternatif garis pemisah (*discrimination boundaries*) ditunjukkan pada Gambar 2.12-a. *Hyperplane* pemisah terbaik antara kedua *class* dapat ditemukan dengan mengukur *margin hyperplane* tsb. dan mencari titik maksimalnya. *Margin* adalah jarak antara *hyperplane* tersebut dengan pattern terdekat dari masing-masing *class*. *Pattern* yang paling dekat ini disebut sebagai *support vector*.

Garis solid pada Gambar 2.12-b menunjukkan *hyperplane* yang terbaik, yaitu yang terletak tepat pada tengah-tengah kedua *class*, sedangkan titik merah dan

kuning yang berada dalam lingkaran hitam adalah *support vector*. Usaha untuk mencari lokasi *hyperplane* ini merupakan inti dari proses pembelajaran pada SVM. Data yang tersedia dinotasikan sebagai $\overrightarrow{x_i} \in \Re^d$ sedangkan label masing-masing dinotasikan yi ∈ {-1,+1}untuk i = 1,2,...n ,yang mana n adalah banyaknya data. Diasumsikan kedua *class* –1 dan +1 dapat terpisah secara sempurna oleh *hyperplane* berdimensi *d* , yang didefinisikan sebagai berikut :

$$\vec{w} \,.\, \vec{x} + b = 0 \quad \text{.....(2.18)}$$

Pattern $\overrightarrow{x_i}$ yang termasuk *class* -1 (sampel negatif) dapat dirumuskan sebagai *pattern* yang memenuhi pertidaksamaan sebagai berikut :

$$\vec{w} \,.\, \vec{x} + b \leq -1 \quad \text{.....(2.19)}$$

Sedangkan *pattern* $\overrightarrow{x_i}$ yang termasuk *class* +1 (sampel positif) adalah sebagai berikut :

$$\vec{w} \,.\, \vec{x} + b \geq +1 \quad \text{.....(2.20)}$$

Margin terbesar dapat ditemukan dengan memaksimalkan nilai jarak antara *hyperplane* dan titik terdekatnya, yaitu $1 \,/\, \|\vec{w}\|$. Hal ini dapat dirumuskan sebagai *Quadratic Programming (QP) problem*, yaitu mencari titik minimal persamaan (2.21), dengan memperhatikan *constraint* persamaan (2.22)

$$\min_{\vec{w}} \tau(w) = \frac{1}{2} \|\vec{w}\|^2 \quad \text{.....(2.21)}$$

$$y_i(\overrightarrow{x_i} \,.\, \vec{w} + b) - 1 \geq 0, \forall i \quad \text{.....(2.22)}$$

Problem ini dapat dipecahkan dengan berbagai teknik komputasi, di antaranya *Lagrange Multiplier*.

$$L(\vec{w}, b, \alpha) = \frac{1}{2} \|\vec{w}\|^2 - \sum_{i=1}^{l} \alpha_i \left(y_i \left((\vec{x}_i \,.\, \vec{w} + b) - 1 \right) \right), \; i = 1,2 \ldots l \quad \text{.....(2.23)}$$

α_i adalah *Langrange multipliers*, yang bernilai nol atau positif ($\alpha_i \geq 0$). Nilai optimal dari persamaan (2.22) dapat dihitung dengan meminimalkan *L* terhadap $\vec{w}$ dan *b*, dan memaksimalkan *L* terhadap α_i. Dengan memperhatikan sifat bahwa pada

titik optimal *gradient* L = 0, persamaan (2.22) dapat dimodifikasi sebagai maksimalisasi *problem* yang hanya mengandung α_i, sebagaimana persamaan (2.24) dibawah

Maximize :

$$\sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j \vec{x}_i . \vec{x}_j \quad \text{.....(2.24)}$$

Dimana $\alpha_i \geq 0$ ($i = 1,2, \ldots . l$) dan $\sum_{i=1}^{l} \alpha_i y_i = 0$. Dari hasil perhitungan ini diperoleh α_i yang kebanyakan bernilai positif. Data yang berkorelasi dengan α_i yang positif inilah yang disebut *support vector*.

Untuk mendapatkan nilai α_i, langkah pertama adalah mengubah setiap *feature* menjadi nilai vektor (*support vector*) = $\frac{x}{y}$. Kemudian vektor akan dimasukkan kedalam persamaan *kernel trick phi* φ yaitu sebagai berikut :

$$\varphi \begin{bmatrix} x \\ y \end{bmatrix} = \begin{cases} \sqrt{x_n^2 + y_n^2} > 2, maka \begin{bmatrix} 4 - x + |\, x - y \,| \\ 4 - x + |x - y \,| \end{bmatrix} \\ \sqrt{x_n^2 + y_n^2} \leq 2, maka \begin{bmatrix} x \\ y \end{bmatrix} \end{cases} \quad \text{.....(2.25)}$$

Kemudian untuk mencari nilai α_i didapatkan dari persamaan sebagai berikut :

$$\sum_{i=1,j=1}^{n} a_i T_i^T T_j \ldots \ldots (2.26)$$

Dan selanjutnya menggunakan persamaan sebagai berikut :

$$W = \sum_{i=1}^{n} a_i T_i \; dan \sum_{i=1,j=1}^{n} a_i T_i = y_i \ldots \ldots (2.27)$$

Terakhir akan dicari nilai w dan b untuk menemukan *hyperplane* sebagai patokan proses klasifikasi dengan persamaan sebagai berikut :

$$y = wx + b \,, \qquad w = \sum_{i}^{n} a_i s_i \;\; \ldots \ldots (2.28)$$

Nilai s_i merupakan nilai *support vector* yang telah dihitung sebelumnya. Dengan demikian proses klasifikasi selesai dengan memperhatikan *hyperplane* nya.

2.10.1 *Kernel trick*

Feature space dalam prakteknya biasanya memiliki dimensi yang lebih tinggi dari vektor input (*input space*). Hal ini mengakibatkan komputasi pada *feature space* mungkin sangat besar, karena ada kemungkinan *feature space* dapat memiliki jumlah *feature* yang tidak terhingga. Selain itu, sulit mengetahui fungsi transformasi yang tepat. Untuk mengatasi masalah ini, pada SVM digunakan "*kernel trick*". Fungsi kernel yang umum digunakan adalah sebagai berikut [23]:

1. *Kernel linier*

   $K(x_i, x) = x_i^T x$ .....(2.29)
2. *Polynomial kernel*

   $K(x_i, x) = (\gamma . x_i^T x + r)^p, \gamma > 0$ .....(2.30)
3. *Radial basis function (RBF)*

   $K(x_i, x) = \exp(-\gamma \, |x_i - x|^2), \gamma > 0$ .....(2.31)
4. *Sigmoid kernel*

   $K(x_i, x) = \tanh(\gamma x_i^T x + r)$ .....(2.32)

2.11 *Multiclass Support Vector Machine*

SVM saat pertama kali diperkenalkan oleh Vapnik, hanya dapat mengklasifikasikan data ke dalam dua kelas (klasifikasi biner). Namun, penelitian lebih lanjut untuk mengembangkan SVM sehingga bisa mengklasifikasi data yang memiliki lebih dari dua kelas, terus dilakukan. Ada dua pilihan untuk mengimplementasikan *multiclass* SVM yaitu dengan menggabungkan beberapa SVM biner atau menggabungkan semua data yang terdiri dari beberapa kelas ke dalam sebuah bentuk permasalah optimasi. Namun, pada pendekatan yang kedua permasalahan optimasi yang harus diselesaikan jauh lebih rumit. Berikut ini adalah metode yang umum digunakan untuk mengimplementasikan *multiclass* SVM dengan pendekatan sebagai berikut [23] :

1. Metode *one-against-all*

   Dengan menggunakan metode ini, akan dibangun k buah model SVM biner (k adalah jumlah kelas). Contohnya, terdapat permasalahan klasifikasi dengan 4 buah kelas. Untuk pelatihan digunakan 4 buah SVM biner seperti pada Tabel 2.2 dan penggunanya dalam mengklasifikasi kelas pada data baru dapat dilihat pada persamaan sebagai berikut [24]:

   $$Kelas\ x = \arg\max_{i=1..k}(\left(w^{(i)}\right)^{T}.\varphi(x) + b^{(i)}) \quad \text{..... (2.33)}$$

   Dengan menentukan *hyperplane* terbesar pada nilai x maka akan mengklasifikasikan kelas tersebut.

Tabel 2.2 Contoh 4 SVM biner dengan metode *One-against-all*

$y_i = 1$	$y_i = -1$	Hipotesis
Kelas 1	Bukan kelas 1	$f^1(x) = (w^1)x + b^1$
Kelas 2	Bukan kelas 2	$f^2(x) = (w^2)x + b^2$
Kelas 3	Bukan kelas 3	$f^3(x) = (w^3)x + b^3$
Kelas 4	Bukan kelas 4	$f^4(x) = (w^4)x + b^4$

2.12 *Image Moments*

Image moments merupakan sebuah operasi dalam *Computer Vision* yang salah satunya untuk mencari *central* atau titik pusat dari sebuah *history* gambar. *Moments* melakukan *summarize* pada bentuk gambar yang diberikan. Berikut merupakan rumus perhitungan dari *Zeroth moments* untuk menemukan titik pusat dari sebuah *history* gambar [25]:

$$\mu_{0,0} = \sum_{x=0}^{w}\sum_{y=0}^{h} f(x,y) \quad \text{.....(2.34)}$$

Dimana *w* dan *h* merupakan *width* dan *height* dari suatu gambar dan f(x,y) merupakan nilai yang didapatkan dari pencarian kontur koordinat x dan y pada gambar biasanya berbentuk *array* (larik).

Dan untuk menemukan *central moments* atau *centroid* dari suatu gambar akan dilakukan penjumlahan seluruh koordinat *x* dan *y* dapat dilakukan dengan rumus perhitungan sebagai berikut :

$$sum_x = \sum\sum x\, f(x,y)\ dan\ sum_y = \sum\sum y\, f(x,y) \quad \text{.....(2.35)}$$

Selanjutnya akan diteruskan perhitungan rumus sebagai berikut :

$$centroid = \left(\frac{\mu_{1,0}}{\mu_{0,0}}, \frac{\mu_{0,1}}{\mu_{0,0}}\right) dimana\ \mu_{1,0} = \frac{sum_x}{\mu_{0,0}}\ dan\ \mu_{0,1} = \frac{sum_y}{\mu_{0,0}} \quad \text{.....(2.36)}$$

Hasil dari $\frac{\mu_{1,0}}{\mu_{0,0}}$ dan $\frac{\mu_{0,1}}{\mu_{0,0}}$ akan dibulatkan.

2.13 Webcam

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

Webcam juga merupakan sebuah periferal berupa kamera sebagai pengambil citra/gambar dan mikropon (optional) sebagai pengambil suara/audio yang dikendalikan oleh sebuah komputer atau oleh jaringan komputer. Gambar yang diambil oleh Webcam ditampilkan ke layar monitor, karena dikendalikan oleh komputer maka ada interface atau port yang digunakan untuk menghubungkan Webcam dengan komputer atau jaringan. Ada beberapa orang mengartikan Webcam sebagai *Web pages* + *Camera*, karena dengan menggunakan Webcam untuk mengambil gambar video secara aktual bisa langsung di upload bila komputer yang mengendalikan terkoneksi internet [26]. Berikut merupakan contoh webcam terdapat pada Gambar 2.13

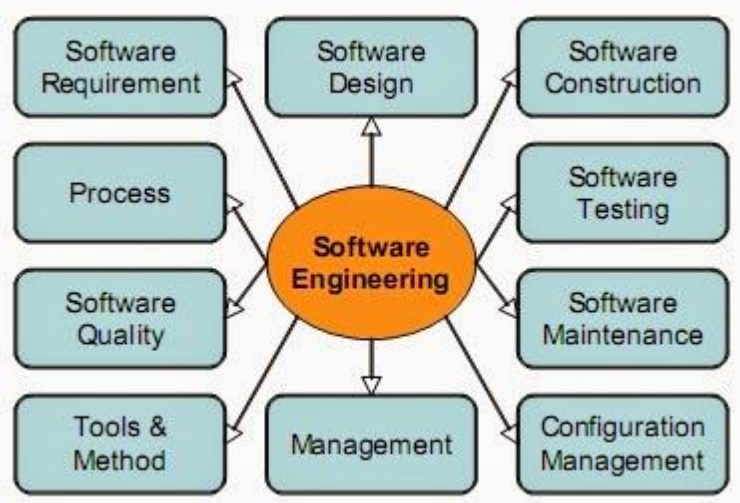
Gambar 2.13 Contoh webcam

2.14 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak telah berkembang sejak pertama kali diciptakan pada tahun 1940-an hingga kini. Fokus utama pengembangannya adalah untuk mengembangkan praktek dan teknologi untuk meningkatkan produktivitas para praktisi pengembang perangkat lunak dan kualitas aplikasi yang dapat digunakan oleh pemakai. Istilah Rekayasa Perangkat Lunak (RPL) secara umum disepakati sebagai terjemahan dari istilah *Software engineering*. Istilah *Software Engineering* mulai dipopulerkan pada tahun 1968 pada software engineering Conference yang diselenggarakan oleh NATO. Tujuan dari Rekayasa Perangkat Lunak adalah sebagai berikut :

1. Memperoleh biaya produksi perangkat lunak yang rendah
2. Menghasilkan perangkat lunak yang kinerjanya tinggi, andal dan tepat waktu
3. Menghasilkan perangkat lunak yang dapat bekerja pada berbagai jenis platform
4. Menghasilkan perangkat lunak yang biaya perawatannya rendah

Sesuai dengan definisi yang telah disampaikan sebelumnya, maka ruang lingkup RPL dapat dilihat pada Gambar 2.14



Gambar 2.14 Ruang Lingkup RPL

Berikut ini merupakan penjelasan dari ruang lingkup RPL [27]:

1. *Software Requirements* berhubungan dengan spesifikasi kebutuhan dan persyaratan perangkat lunak
2. *Software Design* mencakup proses penampilan arsitektur, komponen, antar muka, dan karakteristik lain dari perangkat lunak
3. *Software Construction* berhubungan dengan detail pengembangan perangkat lunak, termasuk algoritma, pengkodean, pengujian dan pencarian kesalahan
4. *Software Testing* meliputi pengujian pada keseluruhan perilaku perangkat lunak
5. *Software Maintenance* mencakup upaya-upaya perawatan ketika perangkat lunak telah dioperasikan
6. *Software Configuration* Management berhubungan dengan usaha perubahan konfigurasi perangkat lunak untuk memenuhi kebutuhan tertentu
7. *Software Engineering Management* berkaitan dengan pengelolaan dan pengukuran RPL, termasuk perencanaan proyek perangkat lunak
8. *Software Engineering Tools And Methods* mencakup kajian teoritis tentang alat bantu dan metode RPL
9. *Software Engineering Process* berhubungan dengan definisi, implementasi pengukuran, pengelolaan, perubahan dan perbaikan proses RPL

10. *Software Quality* menitik beratkan pada kualitas dan daur hidup perangkat lunak

2.15 NetBeans IDE

Bagi pengembang aplikasi berbasis Java maupun PHP mungkin sudah familiar dengan menggunakan NetBeans IDE. Biasanya NetBeans IDE ini digunakan oleh pengembang aplikasi untuk melakukan pemrograman, kompilasi, mencari kesalahan, dan menjalankan aplikasi yang telah dibuat. NetBeans IDE sendiri dibuat dengan menggunakan bahasa pemrograman Java, namun untuk membuat aplikasi yang dapat digunakan dalam sebuah perangkat komputer, maupun mobile, IDE ini pun mampu mendukung bahasa lain. Beberapa bahasa yang didukung oleh NetBeans IDE ini terdiri dari Java, C/C++, PHP, XML, HTML, Javadoc, Javascript, JSP, dan masih banyak lagi. Selain itu, pembaca dapat memasang *plugin*, maupun modul yang bisa didapatkan di komunitas untuk mendukung bahasa lain agar dapat dijalankan di NetBeans IDE ini.

Tampilan antarmuka dari NetBeans IDE ini bisa dibilang cukup memudahkan pengembang aplikasi dalam melakukan interaksi secara grafikal yang tidak hanya berupa kode saja sehingga dapat dilihat hasil kodenya secara langsung dalam bentuk grafis. Selain itu, library yang disediakan dalam NetBeans ini bisa dibilang cukup lengkap untuk beberapa bahasa pemrograman sehingga pengembang aplikasi pemula pun dapat mempelajari langsung *library* yang dibutuhkan dalam membuat aplikasi.

NetBeans IDE terbaru telah memasuki versi 8.0, dan telah mendukung Java 8. Selain itu, IDE ini bisa berjalan di sistem operasi Windows, Mac OS, dan Linux 32/64 bit. Untuk mengunduh NetBeans IDE ini pengguna dapat langsung mengunjungi halaman resmi unduh NetBeans yang nantinya akan diberikan pilihan paket unduhan NetBeans IDE yang terdiri dari paket JAVA SE, JAVA EE, C/C++, HTML5 & PHP, atau paket keseluruhan [28].

2.16 *Unified Modelling Language* (UML) 2.0

Diagram UML adalah sekumpulan alat yang digunakan untuk melakukan abstraksi terhadap sebuah sistem atau perangkat lunak berbasis objek. UML merupakan singkatan dari Unified Modeling Language. Dalam UML sendiri terdapat beberapa diagram yang wajib dikuasai yaitu [29]:

1. *Structural Diagram*
    a. *Class Diagram*, diagram ini terdiri dari *class*, *interface*, *association*, dan *collaboration*. Diagram ini menggambarkan objek - objek yang ada di sistem.
    b. *Deployment Diagram*, diagram ini menggambarkan kumpulan node dan hubungan antar node. Node adalah entitas fisik dimana komponen di-deploy. Entitas fisik ini dapat berupa server atau perangkat keras lainnya.
2. *Behavioral Diagram*
    a. *Use case Diagram*, diagram ini menggambarkan kumpulan use case, aktor, dan hubungan mereka. *Use case* adalah hubungan antara fungsionalitas sistem dengan aktor internal/eksternal dari sistem.
    b. *Sequence Diagram*, diagram ini menggambarkan interaksi yang menjelaskan bagaimana pesan mengalir dari objek ke objek lainnya.
    c. *Statechart Diagram*, diagram ini menggambarkan bagaimana sistem dapat bereaksi terhadap suatu kejadian dari dalam atau luar. Kejadian (*event*) ini bertanggung jawab terhadap perubahan keadaan sistem.
    d. *Activity Diagram*, menggambarkan aliran kontrol sistem. Diagram ini digunakan untuk melihat bagaimana sistem bekerja ketika dieksekusi