BAB 2
LANDASAN TEORI

2.1 Angkutan Kota (Angkot)

Transportasi atau pengangkutan dapat dikelompokan menurut macam atau moda atau jenisnya (*modes of transportation*) yang dapat ditinjau dari segi geografis transportasi itu berlangsung transportasi dapat dibagi menjadi[4]:

a. Angkutan antar benua, misalnya dari Asia ke Eropa.
b. Angkutan antar continental, misalnya dari Francis ke Swiss dan seterusnya sampai ke Timur Tengah.
c. Angkutan antar pulau, misalnya dari Pulau Jawa ke Pulau Sumatera.
d. Angkutan antar kota, misalnya dari Jakarta ke Bandung.
e. Angkutan antar daerah, misalnya dari Jawa Barat ke Jawa Timur.
f. Angkutan di dalam kota, misalnya kota Medan, Surabaya dan lain-lain.

Angkutan kota adalah angkutan dari suatu tempat ke tempat lain dalam wilayah kota dengan mempergunakan mobil bus umum atau mobil penumpang umum yang terikat dalam trayek tetap dan teratur[1]. Data trayek angkot berdasarkan Keputusan Walikota Bandung No.551/Kep.055-Huk/2008 tahun 2008 di kategorikan contoh sebagai berikut[5].

a. Nama Trayek, Cicaheum – Ledeng.
b. Kode Trayek, 05.
c. Jumlah Armada, 214.
d. Jarak (Km), 14.25 km.
e. Warna Kendaraan, Hijau Hitam.

2.2 Computer Vision

Computer Vision merupakan bidang ilmu yang mempelajari bagaimana komputer dapat merekonstruksi, menginterpretasikan, dan memahami suatu tampilan 3 dimensi dalam tampilan 2 dimensi berdasarkan sifat dari struktur tampilan tersebut. *Computer Vision* juga mempelajari bagaimana komputer dapat

mengenali obyek yang diamati. Namun visualisasi data lebih ke arah pemanipulasian gambar (visual) secara digital.

Bentuk visualisasi data yaitu Pengolahan Citra (*image processing*) dan Pengenalan Pola (*Pattern Recognition*). Pengolahan Citra (*image processing*) merupakan bidang dalam *Computer Vision* yang berhubungan dengan proses transformasi citra atau gambar. Proses ini bertujuan untuk mendapatkan kualitas citra yang lebih baik. Sedangkan Pengenalan Pola berhubungan dengan proses identifikasi obyek pada citra atau interpretasi citra. Proses ini bertujuan untuk mengekstrak informasi atau pesan yang disampaikan oleh gambar atau citra[6].

2.3 Citra Digital

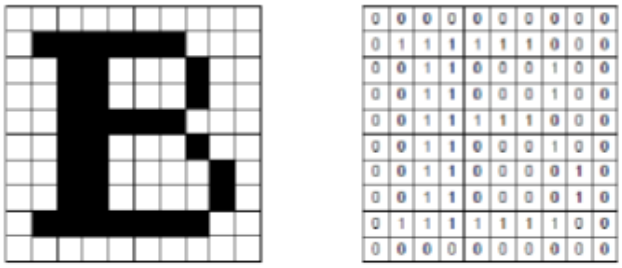
Citra Digital adalah representasi dari sebuah citra dua dimensi sebagai sebuah kumpulan nilai digital yang disebut elemen gambar atau piksel. Piksel adalah elemen terkecil yang menyusun citra dan mengandung nilai yang mewakili kecerahan dari sebuah warna pada sebuah titik tertentu. Umumnya citra digital berbentuk persegi panjang atau bujur sangkar (pada beberapa sistem pencitraan ada pula yang berbentuk segienam) yang memiliki lebar dan tinggi tertentu. Ukuran ini biasanya dinyatakan dalam banyaknya piksel sehingga ukuran citra selalu bernilai bulat. Setiap piksel memiliki koordinat sesuai posisinya dalam citra. Koordinat ini biasanya dinyatakan dalam bilangan bulat positif, yang dapat dimulai dari 0 atau 1 tergantung pada sistem yang digunakan. Setiap piksel juga memiliki nilai berupa angka digital yang merepresentasikan informasi yang diwakili oleh piksel tersebut[7].

2.3.1 Jenis Citra Digital

2.3.1.1 Citra biner

Citra Biner atau citra hitam putih (*black and white image*) adalah citra yang hanya memiliki 2 kemungkinan nilai untuk setiap pikselnya, yaitu atau 1. Nilai 0 akan tampil sebagai warna hitam dan nilai 1 akan tampil sebagai warna putih. Maka dari itu, jenis citra ini hanya membutuhkan 1-bit untuk menyimpan nilai pada setiap pikselnya. Jenis citra ini sering digunakan untuk proses *masking* ataupun proses

segmentasi citra[7]. Berikut contoh citra biner dan representasi nilai tiap piksel pada Gambar 2.1.



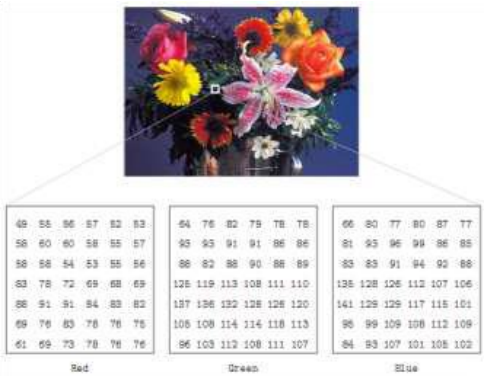
Gambar 2.1 Gambaran Citra Biner

2.3.1.2 Citra berwarna

Citra berwarna adalah citra yang memiliki 3 buah kanal warna di dalamnya. Pada umumnya jenis citra ini berbentuk dari komponen merah/*red* (R), hijau/*green* (G), dan biru/*blue* (B) yang dimodelkan kedalam ruang warna RGB.

1. RGB

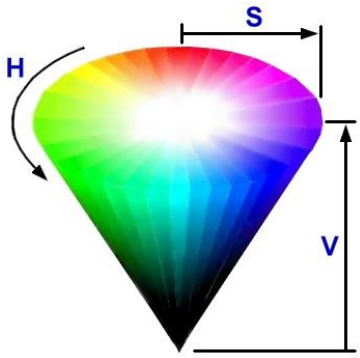
RGB adalah standar yang digunakan untuk menampilkan citra berwarna paa layer televisi maupun layer komputer. Namun terdapat juga (*Cyan, Magenta, Yellow, Black*), HSV (*Hue, Saturation, Value*), YCbCr (*Luma, Chroma blue, Chroma red*), dan Lab (L*a*b). Berikut ini contoh dari citra berwarna representasi citra RGB pada Gambar 2.2.



Gambar 2.2 Gambaran Citra Berwarna

2. HSV

HSV adalah model warna yang lebih baik untuk digunakan untuk berbagai keperluan pengolahan citra dan *computer vision*. Misalnya saja pada object tracking berdasarkan warna, segmentasi citra dsb. *Hue* (H) adalah ukuran dari jenis warna seperti warna merah, kuning, hijau dan seterusnya. Representasinya dalam bentuk deraja dengan nilai 0 – 360. *Saturation* (S) adalah keberwarnaan suatu warna. Semakin berwarna sebuah warna berarti semakin besar nilai saturasinya. Namun apabila suatu warna pucat, itu berarti saturasinya rendah. *Value* (V) adalah nilai kecerahan sebuah warna. Warna cerah memiliki nilai *Value* tinggi dan sebaliknya untuk warna yang gelap[7]. Berikut ini merupakan gambar Model Warna HSV pada Gambar 2.3 dan langkah perhitungan untuk konversi dari citra RGB menjadi citra HSV:

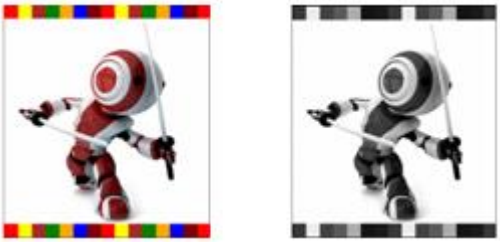


Gambar 2.3 Model Warna HSV

2.4 Citra Grayscale

Citra *Grayscale* adalah citra yang hanya memiliki 1 buah kanal sehingga yang ditampilkan hanyalah nilai intensitas aau dikenal juga dengan istilah derajat keabuan. Karena jenis citra ini hanya memiliki 1 kanal saja, maka cira *grayscale* memiliki tempat penyimpanan yang lebih hemat. Jenis ini disebut juga 8-*bit image* karena untuk setiap nilai pikselnya memerlukan penyimpanan sebesar 8-*bit*. Foto hitam putih maupun gambar yang ditampilkan oleh televisi hitam putih sebenarnya menggunakan citra *grayscale*, bukan dalam warna hitam dan putih. Namun dikalangan masyarakat istilah foto hitam putih maupun televisi hitam putih sudah

terbiasa digunakan dalam kehidupan sehari-hari[7]. Berikut contoh perbedaan antara citra berwarna dengan citra *grayscale* pada Gambar 2.4.



Gambar 2.4 Perbedaan Citra Berwarna dan Citra *Grayscale*

2.5 Akuisisi Citra

Tahap akuisisi citra dilakukan untuk menyeragamkan ukuran piksel pada citra yang diambil melalui *scanner*. Penggunaan *scanner* bertujuan untuk agar teknik tersebut bisa direplikasi dengan harga relatif murah dan tidak banyak membutuhkan persyaratan teknis[8]. Dalam menyeragamkan ukuran piksel, dilakukan *cropping* citra dengan ukuran 128x256 piksel, kemudian di *resize* menjadi 64x128. Citra angkot yang di-*cropping* dan *resize* dapat dilihat pada Gambar 2.5 (a) Citra yang di-*crop* dan (b) Citra yang di-*resize*.

` (a) (b)

Gambar 2.5 Citra di-*crop* dan Citra di-*resize*

2.6 *Histograms of Oriented Gradients*

Histograms of Oriented Gradients (HOG) adalah salah satu metode ekstraksi ciri yang digunakan dalam *image processing* untuk mendeteksi suatu objek. HOG berasal dari sebuah asumsi yang menyatakan bahwa suatu objek dapat di representasikan dengan baik berdasarkan bentuk. Untuk memperoleh informasi pembeda maka gambar akan dibagi menjadi *cell* dan setiap *cell* akan dihitung sebagain *Histograms of Oriented Gradient*. Setiap piksel dalam *cell* berkontribusi

pada saat dilakukan *voting* bobot untuk membangun sebuah histogram yang berorientasi pada nilai-nilai gradien yang dihitung[9].

Dalam pemrosesan *Histograms of Oriented Gradient* diperlukan beberapa langkah, diantaranya[10].

1. Merubah citra RGB menjadi citra *grayscale* (konversi citra).
2. Menghitung nilai gradien orientasi pada setiap piksel dalam citra *grayscale*.
3. Membuat histogram dengan 9 *bin* orientasi (0-20º, 20-40º, 40-60º, 60-80º, 80-100º, 100-120º, 120-140º, 140-160º, 160-180º).
4. Menormalisasikan nilai tiap bin orientasi tersebut.

2.5.1 Konversi nilai RGB ke Grayscale

Secara teori ada beberapa cara dalam mengonversi citra berwarna RGB ke dalam citra *grayscale*. Cara yang paling mudah adalah dengan merata-ratakan semua nilai piksel RGB sesuai dengan persamaan 2.1 berikut[7]:

$$y = \frac{1}{3}(R + G + B) \tag{2. 1}$$

Namun hasil konversi yang diberikan dengan menggunakan persamaan 2.1 tidak terlalu bagus. Untuk mendapatkan hasil konversi yang lebih baik, persamaan (2.2) berikut dapat digunakan[7].

$$y = 0.299R + 0.587G + 0.144B \tag{2. 2}$$

Keterangan: *y = Grayscale*

R=Red

G=Green

B=Blue

2.5.2 Hitung Gradien Citra

Perhitungan gradien dilakukan dengan perhitungan gradient yang merupakan operator dua komponen *Ix* dan *Iy* dengan persamaan 2.3 dan 2.4 sebagai berikut[3]:

$$Ix(r, c) = I(r, c + 1) - I(r, c - 1) \tag{2. 3}$$

$$Iy(r, c) = I(r - 1, c) - I(r + 1, c) \tag{2. 4}$$

Keterangan: $Ix = nilai\ gradien\ citra\ horizontal$

$Iy = nilai\ gradien\ citra\ vertikal$

$r = baris$

$c = kolom$

2.5.3 *Magnitude* dan *Angle* gradien

Dari nilai gradien akan dikalkulasikan *magnitude* gradien. Rumus untuk mencari nilai magnitude diperoleh dengan menggunakan persamaan 2.5 berikut[3]:

$$\mu = \sqrt{I_{x^2} + I_{y^2}} \quad (2.5)$$

Keterangan: $\mu = magnitude\ gradien$

$I_x = nilai\ gradien\ citra\ horizontal$

$I_y = nilai\ gradien\ citra\ vertikal$

Angle gradien digunakan untuk mencari nilai orientasi. Rumus untuk mencari *angle* gradient menggunakan persamaan 2.6 sebagai berikut[3]:

$$\theta = \frac{180}{\pi}\ (tan_2^{-1}(Iy, Ix)\ mod\ \pi) \quad (2.6)$$

Keterangan: $\theta = angle\ gradien$

$I_y = gradien\ citra\ vertikal$

$I_x = gradien\ citra\ horizontal$

$\pi = phi\ dengan\ nilai\ angka\ 3{,}14$

2.5.4 Orientasi *Bin*

Tahap selanjutnya adalah melakukan perhitungan histogram dari orientasi gradien tiap *cell*. Setiap piksel dalam sebuah *cell* mempunyai nilai histogram sendiri-sendiri berdasarkan nilai yang dihasilkan dalam perhitungan gradien yang kemudian dilakukan normalisasi pada setiap blok. *Cell* memiliki ukuran 8x8 piksel ada sebuah citra. Sedangkan blok memiliki ukuran 2x2 *cell*. Setiap histogram memiliki sejumlah bin yang sama, yang menentukan ketelitiannya. *Bin* merepresentasikan orientasi gradien (*angle*) dan harus sama ditempatkan pada 0° − 180° untuk gradien "*unsigned*" atau 0° − 360° untuk gradien "*signed*". Satu histogram tiap *cell* dihitung, tiap piksel pada *cell* berkontribusi pada penambahan

histogram dimana histogram tersebut bergantung pada nilai *magnitude*-nya yang berhubungan dengan orientasi gradien, artinya sebuah bin dipilih berdasarkan arah (*angle*) dan vote (nilai yang akan diberikan pada *bin*) dipilih berdasarkan magnitude. Vote bisa jadi fungsi dari *magnitude*, tetapi terbukti bahwa menggunakan secara langsung nilai *magnitude* itu sendiri dapat memberikan hasil yang lebih baik[11]. Pada penelitian ini, dilakukan perhitungan yang didapat dengan persamaan 2.7 sampai 2.10 sebagai berikut[3]:

$$w = \frac{180}{B} \tag{2. 7}$$

$$ci = w\left(i + \frac{1}{2}\right) \tag{2. 8}$$

$$v_j = \mu\frac{c_{j+1} - \theta}{w} \Rightarrow binj = \left[\frac{\theta}{w} - \frac{1}{2}\right] mod\ B \tag{2. 9}$$

$$v_{j+1} = \mu\frac{\theta - c_j}{w} \Rightarrow (j+1)\ mod\ B \tag{2. 10}$$

Keterangan: $v_j = voting\ number$

$ci = bin\ center$

$\mu = nilai\ magnitude$

$\theta = nilai\ orientasi\ bin$

$i = 0,1, \dots, B-1$

$w = rentang\ nilai\ pada\ setiap\ bin$

$B = banyaknya\ bin$

2.5.5 Normalisai *Block*

Fitur blok dinormalisasi untuk mengurangi efek perubahan kecerahan obyek pada sebuah blok. Nilai normalisasi fitur blok selanjutnya didapatkan dengan rumus persamaan 2.11 sebagai berikut[3]:

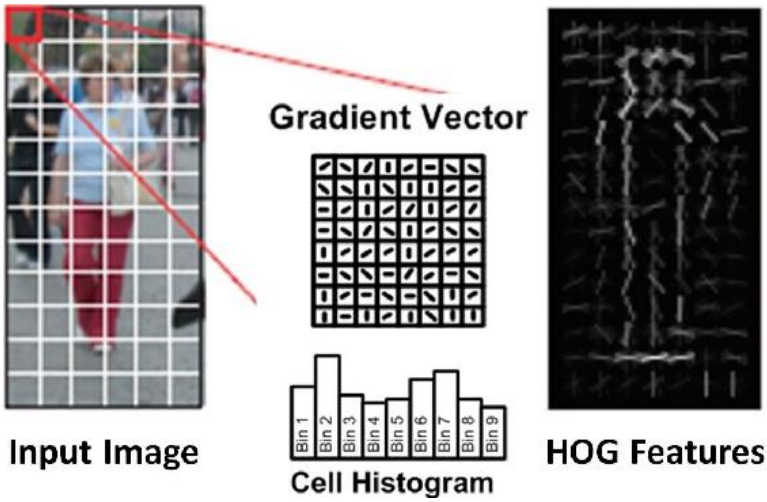
$$b_i = \frac{b_i}{\sqrt{\|b_i\|^2 + \varepsilon}} \tag{2. 11}$$

Ket: $b_i = nilai\ normalisasi\ pada\ 1\ blok$

$\varepsilon = konstanta\ bernilai\ 1$

2.5.6 Fitur HOG

Setelah mendapatkan nlai blok tahap selanjutnya mencari fitur HOG dimana blok yang dinormalisasikan digabungkan menjadi satu buah fitur HOG.
Berikut adalah contoh hasil fitur HOG yang di ilustrasikan pada Gambar 2.6.



Gambar 2.6 Ilustrasi fitur HOG

Nilai normalisasi kemudian digabungkan menjadi satu vektor menjadi fitur HOG yang mana langkah selanjutnya dilakukan normalisasi dengan persamaan berikut[3]:

$$h = \frac{h}{\sqrt{\|h\|^2 + \varepsilon}} \quad (2.\ 12)$$

Ket: $h = nilai\ normalisasi\ block$

$\varepsilon = konstanta\ bernilai\ 1$

2.7 Histogram warna

Histogram citra berwarna dinyatakan dalam histogram intensitas dan histogram masing-masing kanal citra. Ketika mengacu pada histogram dari citra berwarna, biasanya apa yang dimaksud adalah histgram dari intensitas citra (*luminance*) atau dari setiap kanal warna. Kedua hal ini didukung oleh hampir setiap

aplikasi pengolahan citra dan digunakan untuk menilai kualitas citra, terutama langsung setelah mengakuisisi citra[7].

2.6.1 Konversi nilai RGB ke HSV

Untuk mentransformasi dari RGB ke HSV. Diasumsikan koordinat-koordinat R, G, B [0,1] adalah berurutan merah, hijau, biru dalam ruang warna RGB, dengan max adalah nilai maksimum dari nilai red, green, blue, dan min adalah nilai minimum dari nilai red, green, blue. Untuk memperoleh sudut hue [0,360] yang tepat untuk ruang warna HSV, menggunakan rumus seperti berikut [7]:

$$V = \max\{R, G, B\},\ V_m = \min\{R, G, B\} \quad (2.\ 13)$$

Keterangan: $V = nilai\ maksimum\ dari\ R, G, B$

$V_m = nilai\ minimum\ dari\ R, G, B$

$$S = \begin{cases} 0°, jika\ V = 0 \\ \dfrac{V_m}{V}, jika\ V > 0 \end{cases} \quad (2.\ 14)$$

Keterangan: $S = nilai\ saturasi$

$$H = \begin{cases} \dfrac{G - B}{V - Vm} \text{ x } 60, if\ R = V \\ 120 + \dfrac{B - R}{V - Vm} \text{ x } 60, if\ G = V \\ 240 + \dfrac{R - G}{V - Vm} \text{ x } 60, if\ B = V \end{cases} \quad (2.\ 15)$$

Keterangan: $H = nilai\ hue$

2.6.2 Mean

Pada sebuah histogram dapat diketahui intensitas citra secara keseluruhan sehingga bisa disimpulkan citra tersebut gelap atau terang. *Luminosity* (terangnya cahaya) dari sebuah citra dapat diukur dengan menghitung nilai rata-rata (*mean*) dengan persamaan 2.16 berikut[7].

$$mean = \sum_i \frac{H_{norm}}{N} \quad (2.\ 16)$$

Keterangan: N = Jumlah banyaknya histogram

H_{norm} = Nilai Normalisasi Hue

i = piksel pada kolom i

2.8 *Support Vector Machine*

Support Vector Machine (SVM) adalah salah satu metode yang dapat digunakan dalam ekstraksi informasi. *Support Vector Machine* (SVM) dikembangkan oleh Boser, Guyon, dan Vapnik, pertama kali diperkenalkan pada tahun 1992 di *Annual Workshop on Computational Learning Theory*. Pemahan dalam cara kerja SVM adalah memisahkan dua buah kelas yang terpisah secara linier dengan membuat sebuah garis pemisah yang disebut *hyperplane*[12]. Dalam SVM dikenal istilah margin, yaitu jarak antara garis hyperplane dengan data yang paling dekat yang disebut dengan support vector. Usaha untuk mencari lokasi *hyperplane* ini merupakan inti dari proses pelatihan pada SVM.

Support Vector Machine (SVM) memiliki beberapa tahap dalam pengerjaannya, pada tahap awal yaitu pendefinisan persamaan suatu *hyperplane* pemisah. *Hyperplane* adalah sebuah garis lurus atau bidang mendatar yang memisahkan kelas-kelas.

Persamaan (2.17) *Hyperplane*:

$$f(x) = \vec{w} . \vec{x} + b \quad (2.17)$$

Dimana w merupakan suatu bobot vektor, yaitu $\{w_1, w_2, \dots, w_n\}$ n adalah jumlah atribut dan b merupakan suatu skalar yang disebut dengan bias. Jika berdasarkan pada atribut A1, A2 dengan permisalan tupel pelatihan X = (x_1, x_2) , x1 dan x2 merupakan nilai dari atribut A1 dan A2.

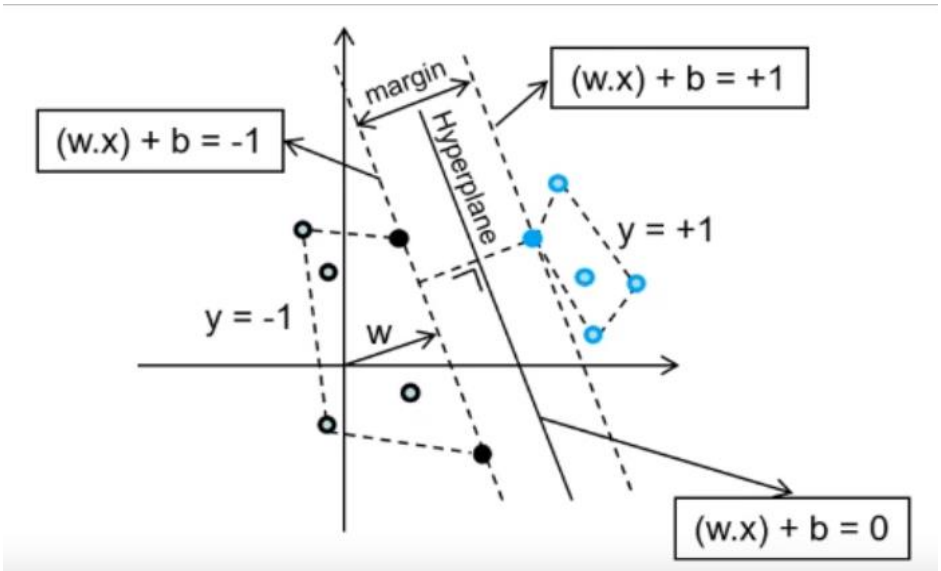
Sehingga diperoleh persamaan -1 (sample negatif) memenuhi pertidaksamaan (2.5)

$$\vec{w} . \vec{x} + b \leq -1 \quad (2.18)$$

Dan kelas +1 pattern yang memenuhi pertidaksamaan (2.19):

$$\vec{w} . \vec{x} + b \geq +1 \quad (2.19)$$

Berikut ilustrasi hyperplane pada Gambar 2.8:



Gambar 2.7 Ilustrasi *Hyperplane*

Margin terbesar dapat dicari dengan cara memaksimalkan jarak antar bidang pembatas kedua kelas dan titik terdekatnya, yaitu $2/|w|$. Hal ini dirumuskan sebagai permasalahan *Quadratic Programming* (QP) problem yaitu mencari titik minimal persamaan (2.20) dengan memperhatikan persamaan (2.21) berikut:

$$\min\ \tau(w) = \frac{1}{2}||w||^2 \quad (2.\ 20)$$

$$y_i\ (w * x_i + b) - 1 \geq 0, (i = 1, .., n) \quad (2.\ 21)$$

Permasalahan ini ini dapat dipecahkan dengan berbagi teknik komputasi. Lebih mudah diselesaikan dengan mengubah persamaan (2.20) ke dalam fungsi *Lagrangian* pada persamaan (2.22) berikut:

$$L\,p = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i y_i (w^\tau x_i + b) - 1 \quad (2.\ 22)$$

α_i adalah *Lagrange Multiplier* yang berkorespondensi dengan x_i. Nilai α_i adalah nol atau positif:

Untuk meminimalkan *Lagrangian*, Persamaan (2.22) harus diturunkan terhadap w dan b, dan diset dengan nilai nol untuk syarat optimasi di atas:

Syarat 1:

$$\frac{\partial Lp}{\partial w} = 0\ \Rightarrow w = \sum_{i=1}^{N} \alpha_i y_i x_i \quad (2.\ 23)$$

Syarat 2:

$$\frac{\partial Lp}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^{N} \alpha_i y_i = 0 \quad (2.\ 24)$$

N adalah jumlah data yang menjadi *support vector*.

Karena *Lagrange Multiplier* (α) tidak diketahui nilainya, persamaan di atas tidak dapat diselesaikan secara langsung untuk mendapatkan w dan b. Untuk menyelesaikan masalah tersebut, modifikasi Persamaan (2.22) diatas menjadi kasus memaksimalkan dengan syarat optimal untuk dualitasnya menggunakan konstrain Karush-Kuhn-Tucker (KKT) sebagai berikut:

Syarat 1:

$$\alpha_i[y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) - 1] = 0 \quad (2.\ 25)$$

Syarat 2:

$$\alpha_i > 0, i = 1,2, \dots, N \quad (2.\ 26)$$

Dengan menerapkan konstrain pada Persamaan (2.25) dan (2.26) maka dipastikan bahwa nilai *Lagrange Multiplier* sama banyaknya dengan data latih, meskipun sebenarnya banyak dari data latih yang *Lagrange Multiplier* sama dengan nol (karena hanya beberpa saja yang akan menjadi *support vector*) ketika menerapkan syarat pertama. Konstrain diatas menyatakan bahwa *Lagrange Multiplier* α_i harus nol kecuali untuk data latih x_i yang memenuhi persamaan:

$$y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) = 0 \quad (2.\ 27)$$

Data latih tersebut, dengan $\alpha_i > 0$, terletak pada *hyperplane* b_{i1} atau b_{i2}, dan disebut *support vector*. Data latih yang tidak terletak di *hyperplane* tersebut mempunyai $\alpha_i = 0$,. Persamaan (2.25) dan (2.26) juga menyarankan parameter w dan b yang mendefinisikan *hyperplane* hanya tergantung *support vector*.

Masalah optimasi di atas masih sulit diselesaikan karena banyaknya parameter (w, b dan α_i). Untuk menyederhanakannya, persamaan optimasi (2.22) diatas harus ditransformasi ke dalam fungsi *Lagrange Multiplier* itu sendiri (disebut dualitas masalah).

Persamaan *Lagrange Multiplier* (2.28) dapat dijabarkan menjadi:

$$Lp = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i y_i (\boldsymbol{w}.\boldsymbol{x_i}) - b\sum_{i=1}^{N} \alpha_i y_i + \sum_{i=1}^{N} \alpha_i \quad (2.28)$$

Syarat optimal (2.24) ada dalam suku ketiga di ruas kanan dalam persamaan (2.28), dan memaksa suku ini menjadi sama dengan nol. Dengan mengganti *w* dari syarat (2.23), dan suku $\|w\|^2 = \boldsymbol{w_i}.\boldsymbol{w_j}$, maka persamaan di atas akan berubah menjadi dualitas *Lagrange Multiplier* berupa Ld dan didapatkan:

Maksimalkan:

$$Ld = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i,j}^{N} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i}.\boldsymbol{x_j} \quad (2.29)$$

$\boldsymbol{x_i}.\boldsymbol{x_j}$ merupakan *dot-product* dua data dalam data latih.

Syarat 1:

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \quad (2.30)$$

Syarat 2:

$$\alpha_i > 0, i = 1,2, \dots, N \quad (2.31)$$

Untuk set data yang besar, masalah dualitas optimasi tersebut (2.29, 2.30, 2.31) dapat diselesaikan dengan metode numerik seperti *Quadratic Programming*. Sekali α_i didapatkan, persamaan (2.23) dan (2.24) bisa digunakan untuk mendapatkan solusi layak untuk *w* dan *b*.

Hyperplane (batas keputusan) didapatkan dengan formula:

$$\left(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x_i}.\boldsymbol{z}\right) + b = 0 \quad (2.32)$$

N adalah jumlah data yang menjadi *support vector*, x_i merupakan *support vector*, *z* merupakan data uji yang akan diprediksi kelasnya, dan $\boldsymbol{x_i}.\boldsymbol{z}$ merupakan *inner-product* antara x_i dan *z*. Untuk nilai *b* didapatkan dari persamaan (2.25) pada *support vector*. Karena α_i dihitung dengan teknik metode numerik dan mempunyai *error* numerik, nilai yang dihitung untuk *b* bisa jadi tidak sama. Hal ini disebabkan oleh *support vector* yang digunakan dalam persamaan (2.25), biasanya diambil nilai rata-rata dari *b* yang didapat untuk menjadi parameter *hyperplane*. Untuk persamaan (2.25) dalam mendapat dapat *b* dapat disederhanakan menjadi:

$$b_i = 1 - y_i(\boldsymbol{w}.\boldsymbol{x_i}) \quad (2.33)$$

Penjelasan di atas berdasarkan asumsi bahwa kedua kelas dapat terpisah secara sempurna oleh *hyperplane*. Akan tetapi, pada umumnya kedua kelas tersebut tidak dapat terpisah secara sempurna. Hal ini menyebabkan proses optimalisasi tidak dapat diselesaikan karena tidak ada *w* dan *b* yang memenuhi pertidaksamaan (2.21). Untuk itu pertidaksamaan tersebut dimodifikasi dengan memasukan variabel slack ξ_i ($\xi_i \geq 0$), Menjadi:

$$y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) \geq 1 - \xi_i \tag{2. 34}$$

Demikian juga untuk masalah persamaan (2.20):

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \tag{2. 35}$$

Parameter *C* berguna untuk mengontrol *trade-off* antara margin dan *error* klasifikasi. Semakin besar nilai *C* maka semakin besar pula pelanggaran yang dikenakan untuk tiap klasifikasi.

Metode untuk mengoptimalisasi *hyperplane* SVM umumnya dipakai untuk menyelesaikan *Quadratic Programming* dengan konstrain yang ditetapkan. Beberapa pilihan metode yang bisa digunakan adalah *chunking* (Vapnik, 1982), metode dekomposisi (Osuna *et al*, 1997), dan *Sequential Minimal Optimization* (SMO)(Plat, 1999).

SVM sebenarnya adalah *hyperplane* linear yang hanya bekerja pada data yang dapat dipisahkan secara linear. Untuk data yang distribusi kelasnya tidak linear biasanya digunakan pendekatan kernel pada fitur data dari awal set data. Kernel dapat di definisikan sebagai suatu fungsi yang memetakan fitur data dari dimensi awal (rendah) ke fitur lain yang berdimensi lain yang lebih tinggi (bahkan jauh lebih tinggi).

Berikut ini adalah beberapa fungsi kernel yang umum digunakan yaitu:

a. Kernel linier

$$K\ (xi, x)\ =\ xi^T x \tag{2. 36}$$

b. Polynomial

$$K\ (xi, x)\ =\ (\gamma . xi^T x + r)p\ , \gamma\ >\ 0 \tag{2. 37}$$

c. Radial basis *function* (RBF)

$$K(xi, x) = exp\ (-\gamma|\mathrm{x}_i - \mathrm{x}|^2)\ , \gamma > 0 \quad (2.\ 38)$$

d. Sigmoid kernel

$$K(xi, x) = \ tanh\ (\gamma xi^T + r). \quad (2.\ 39)$$

Keterangan:

x adalah pasangan dua data dari semua bagian data latih. Parameter $\gamma > 0$, merupakan konstanta. $\|x_i - x\|^2$ merupakan kuadrat jarak antara vektor x_i dan x.

2.9 *Confusion matrix*

Confusion matrix merupakan alat pengukuran yang dapat digunakan untuk menghitung kinerja atau tingkat kebenaran proses klasifikasi. Dengan *confusion matrix* dapat dianalisa seberapa baik *classifier* dapat mengenali *record* dari kelas-kelas yang berbeda[13]. Tabel *confusion matrix* ditunjukkan pada tabel 2.1 berikut:

Table 2.1 *Confusion matrix*

		Prediksi	
		Positif	Negatif
Aktual	Positif	TP (True Positive)	FN (False Negative)
	Negatif	FP (False Positif)	TN (True Negatif)

Keterangan:

a. TP (True Positive) adalah citra angkot yang diklasifikasikan sebagai angkot.
b. FP (False Positive) adalah citra angkot yang diklasifikasikan sebagai bukan angkot.
c. FN (False Negative) adalah citra bukan angkot yang diklasifikasikan sebagai angkot.
d. TN (True Negative) adalah citra bukan angkot yang diklasifikasikan sebagai bukan angkot.

Setelah didapat *true positive, false positive, true negative* dan *false negative*, selanjutnya hitung untuk menghitung nilai akurasinya. Berikut persamaan untuk menghitung akurasi:

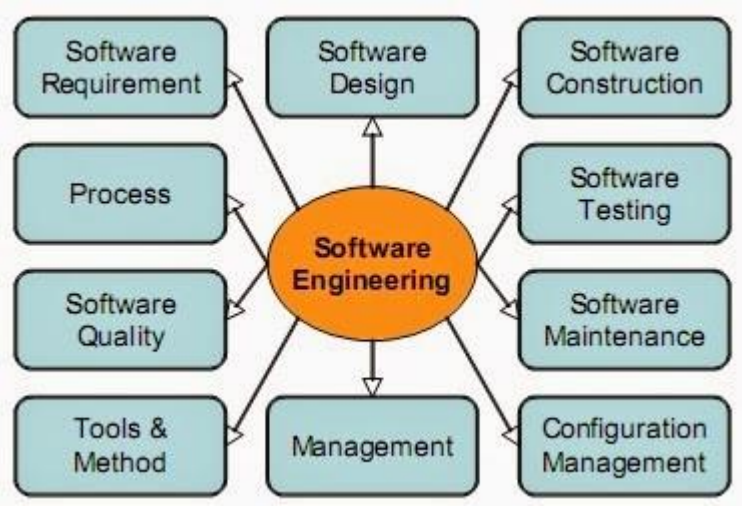
$$Akurasi = \frac{TP + TN}{TP + TN + FP + FN} \text{ x } 100\% \qquad (2.\ 40)$$

2.10 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[14]:

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 2.9.



Gambar 2.8 Ruang Lingkup RPL

Berikut ini merupakan penjelasan dari ruang lingkup RPL:

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.

2.11 Python

Python adalah bahasa pemrograman interpretatif multiguna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. Python mendukung multi paradigma pemrograman utamanya, namun tidak dibatasi pada pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Salah satu fitur yang tersedia pada python adalah sebagai bahasa pemrograman dinamis yang dilengkapi dengan manajemen memori otomatis. Seperti halnya pada bahasa pemrograman dinamis lainnya,

python umumnya digunakan sebagai bahasa skrip meskin pada praktiknya penggunaan bahasa ini lebih luas mencakup konteks pemanfaatan yang umumnya tidak dilakukan dengan menggunakan bahasa script. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi[15]. Saat ini kode python dapat dijalankan di berbagai platform sistem operasi.

2.12 Open CV *(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[16]:

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. *Fiturs2d* – 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*.