# BAB 9

# Pengolahan Citra Berwarna

Setelah bab ini berakhir, diharapkan pembaca memahami berbagai bahasan berikut dan mampu mempraktikkan berbagai hal yang terkait dengan materi bersangkutan.

- ✓ Dasar Warna
- ✓ Ruang Warna
- ✓ Ruang Warna RGB
- ✓ Ruang Warna CMY/CMYK
- ✓ Ruang Warna YIQ
- ✓ Ruang Warna YCbCr
- ✓ Ruang Warna HSI, HSV, dan HSL
- ✓ Ruang Warna CIELAB
- ✓ Memperoleh Statistika Warna
- ✓ Mengatur Kecerahan dan Kontras
- ✓ Menghitung Jumlah Warna
- ✓ Aplikasi Pencarian Citra Berdasarkan Warna Dominan

## 9.1 Dasar Warna

Manusia sebenarnya melihat warna adalah karena cahaya yang dipantulkan oleh objek. Dalam hal ini, spektrum cahaya kromatis berkisar antara 400-700 nm (Zhou, dkk., 2010). Istilah kromatis berarti kualitas warna cahaya yang ditentukan oleh panjang gelombang.

Karakteristik persepsi mata manusia dalam yang membedakan antara satu warna dengan warna yang lain berupa *hue*, *saturation*, dan *brightness*.

- *Hue* merujuk ke warna yang dikenal manusia, seperti merah dan hijau. Properti ini mencerminkan warna yang ditangkap oleh mata manusia yang menanggapi berbagai nilai panjang gelombang cahaya. Sebagai contoh, bila mata menangkap panjang gelombang antara 430 dan 480 nanometer, sensasi yang diterima adalah warna biru, sedangkan jika panjang gelombang berkisar antara 570 sampai dengan 600 nm, warna yang terlihat adalah kuning (Crane, 1997), sedang campuran merah dan hijau terlihat kuning.
- *Saturation* menyatakan tingkat kemurnian warna atau seberapa banyak cahaya putih yang tercampur dengan *hue*. Setiap warna murni bersaturasi 100% dan tidak mengandung cahaya putih sama sekali. Dengan kata lain, suatu warna murni yang bercampur dengan cahaya putih memiliki saturasi antara 0 dan 100%.
- *Brightness* atau kadang disebut *lightness* (kecerahan) menyatakan intensitas pantulan objek yang diterima mata. Intensitas dapat dinyatakan sebagai perubahan warna putih menuju abu-abu dan terakhir mencapai ke warna hitam, atau yang dikenal dengan istilah aras keabuan.

Perlu diketahui, istilah kromatik berarti gabungan antara *hue* dan *saturation* dan istilah akromatik merujuk ke kecerahan.

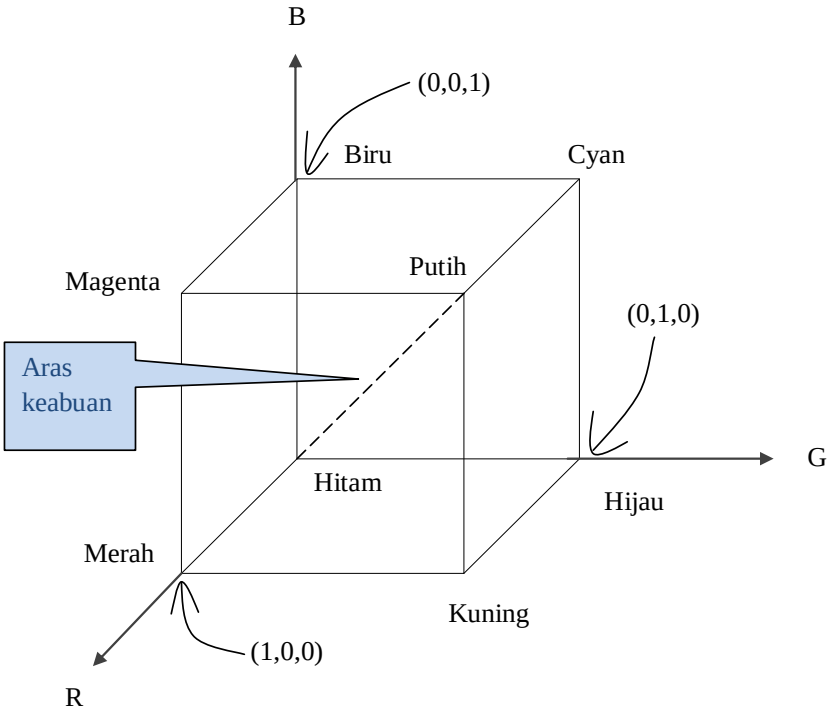
## 9.2 Ruang Warna

Gonzalez & Woods (2002) mendefinisikan ruang warna (atau kadang disebut sistem warna atau model warna) sebagai suatu spesifikasi sistem koordinat dan suatu subruang dalam sistem tersebut dengan setiap warna dinyatakan dengan satu titik di dalamnya. Tujuan dibentuknya ruang warna adalah untuk memfasilitasi spesifikasi warna dalam bentuk suatu standar. Ruang warna yang paling dikenal pada perangkat komputer adalah RGB, yang sesuai dengan watak manusia dalam menangkap warna. Namun, kemudian dibuat banyak ruang warna, antara lain HSI, CMY, LUV, dan YIQ.

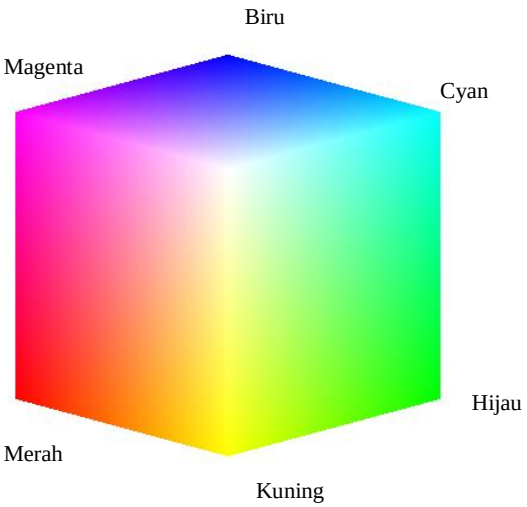
### 9.2.1 Ruang Warna RGB

Ruang warna RGB biasa diterapkan pada monitor CRT dan kebanyakan sistem grafika komputer. Ruang warna ini menggunakan tiga komponen dasar

yaitu merah (R), hijau (G), dan biru (B). Setiap piksel dibentuk oleh ketiga komponen tersebut. Model RGB biasa disajikan dalam bentuk kubus tiga dimensi, dengan warna merah, hijau, dan biru berada pada pojok sumbu (Gambar 9.1). Warna hitam berada pada titik asal dan warna putih berada di ujung kubus yang berseberangan. Gambar 9.2 memperlihatkan kubus warna secara nyata dengan resolusi 24 bit. Perlu diketahui, dengan menggunakan 24 bit, jumlah warna mencapai 16.777.216.




**Gambar 9.1 Skema ruang warna RGB dalam bentuk kubus**




**Gambar 9.2 Kubus warna dengan 24 bit**

RGB biasa digunakan karena kemudahan dalam perancangan *hardware*, tetapi sebenarnya tidak ideal untuk beberapa aplikasi. Mengingat warna merah, hijau, dan biru sesungguhnya terkorelasi erat, sangat sulit untuk beberapa algoritma pemrosesan citra (Crane, 1997). Sebagai contoh, kebutuhan untuk memperoleh warna alamiah seperti merah dengan menggunakan RGB menjadi sangat kompleks mengingat komponen R dapat berpasangan dengan G dan B, dengan nilai berapa saja. Hal ini menjadi mudah jika menggunakan ruang warna HLS ataupun HSV.

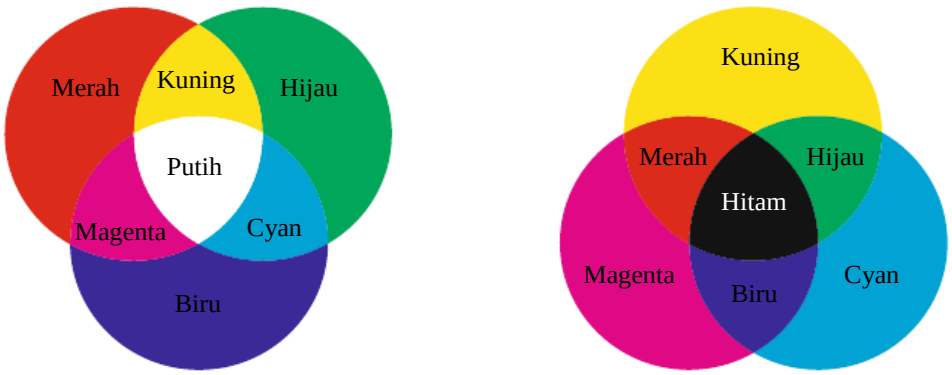
### 9.2.2 Ruang Warna CMY/CMYK

Model warna CMY (*cyan*, *magenta*, *yellow*) mempunyai hubungan dengan RGB sebagai berikut:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{9.1}$$

Dalam hal ini, R, G, dan B berupa nilai warna yang telah dinormalisasi, dengan jangkauan [0, 1].

Pada CMY, warna hitam diperoleh jika C, M, dan Y bernilai sama. Namun, pada aplikasi printer, warna hitam ditambahkan tersendiri sehingga membentuk CMYK, dengan K menyatakan warna hitam. Alasannya, kalau ada warna hitam, warna dapat diambilkan secara langsung dari tinta hitam, tanpa perlu mencampur dengan warna lain. Lagipula, tinta warna hitam lebih murah daripada tinta berwarna dan paling sering digunakan terutama untuk teks.




(a) Penjumlahan warna pada sistem RGB

(b) Pengurangan warna pada sistem CMY

**Gambar 9.3 Warna-warna lain dapat dibentuk melalui kombinasi tiga warna dasar**

Perlu diketahui, konversi dari CMY ke CMYK dapat menggunakan berbagai cara perhitungan. Salah satu rumus yang digunakan sebagai berikut (Crane, 1997):

$$K = min(C, M, Y) \quad (9.2)$$
$$C' = C - K \quad (9.3)$$
$$M' = M - K \quad (9.4)$$
$$Y' = Y - K \quad (9.5)$$

Dengan pendekatan seperti itu, salah satu dari C', M', atau Y' akan bernilai 0. Namun, ada pula yang menggunakan rumus seperti berikut (Dietrich, 2003):

$$K = min(C, M, Y) \quad (9.6)$$
$$C = (C - K)/(1-K) \quad (9.7)$$
$$M = (M - K)\ (1-K) \quad (9.8)$$
$$Y = (Y - K)\ (1-K) \quad (9.9)$$

Dalam hal ini, jika K = 1, C=Y=K=0.

Selain itu, pendekatan yang lain terdapat pada Pratt (2001). Rumus yang digunakan berupa:

$$K_b = min(1-R, 1-G, 1-B) \quad (9.10)$$
$$C = 1 - R - uK_b \quad (9.11)$$
$$M = 1 - G - uK_b \quad (9.12)$$
$$Y = 1 - B - uK_b \quad (9.13)$$
$$K = bK_b \quad (9.14)$$

Dalam hal ini, $0 \leq u \leq 1$ dan $0 \leq b \leq 1{,}0$.

Contoh konversi dari RGB ke CMYK ditunjukkan di bawah ini.

**Program : RGBkeCMY.m**

```
function [C,M,Y,K] = RGBkeCMY(R,G,B)
% RGBkeCMY digunakan untuk mengonversi RGB ke CMYK
%    Berdasarkan Pratt (2001)

% Normalisasi RGB ke [0, 1]
R = double(R);
G = double(G);
B = double(B);

if max(max(R)) > 1.0 || max(max(G)) > 1.0 || ...
   max(max(B)) > 1.0
```

```
    R = double(R) / 255;
    G = double(G) / 255;
    B = double(B) / 255;
end

u = 0.5;
b = 1;
[tinggi, lebar] = size(R);
for m=1: tinggi
    for n=1: lebar
        Kb = min([(1-R(m,n)) (1-G(m,n)) (1-B(m,n))]);
        if Kb == 1
            C(m,n) = 0;
            M(m,n) = 0;
            Y(m,n) = 0;
        else
            C(m,n) = (1.0 - R(m,n) - u * Kb);
            M(m,n) = (1.0 - G(m,n) - u * Kb);
            Y(m,n) = (1.0 - B(m,n) - u * Kb);
            K(m,n) = b * Kb;
        end
    end
end

% Konversikan ke jangkauan [0,255]
C = uint8(C * 255);
M = uint8(M * 255);
Y = uint8(Y * 255);
K = uint8(K * 255);
```

**Akhir Program**

Contoh di atas didasarkan pada Persamaan 9.10 hingga 9.14. Masukan R, G, dan B dapat berjangkauan [0, 1] ataupun [0, 255]. Fungsi `RGBkeCMY` dengan sendirinya akan menormalisasi R,G,B sehingga berjangkauan [0, 1]. Hasil C,M,Y, dan K akan diatur berjangkauan [0, 255]. Contoh penggunaan pada satu piksel:

```
>> [C,M,Y,K] = RGBkeCMY(171, 215, 170) ↵
C = 64
M = 20
Y = 65
K = 40
>>
```

Contoh konversi untuk seluruh citra diperlihatkan berikut ini.

```
>> Img = imread('C:\Image\lapangan.png'); ↵
>> [C,M,Y,K] = RGBkeCMY(Img(:,:,1), Img(:,:,2), Img(:,:,3)); ↵
>>
```

Pada contoh di atas, Img(:,:,1),Img(:,:,2),Img(:,:,3) secara berturut-turut menyatakan komponen R, G, dan B. Dengan cara seperti itu, *C*, *M*, *Y*, dan *K* berupa matriks yang masing-masing menyatakan komponen C, M, Y, dan K.

Konversi dari CMY ke RGB pada dasarnya dapat dilakukan dengan mudah, dengan mengacu pada Persamaan 9.11 hingga 9.14. Implementasinya dapat dilihat pada contoh berikut.

**Program : CMYkeRGB.m**

```
function [R,G,B] = CMYkeRGB(C,M,Y,K)
% CMYkeRGB digunakan untuk mengonversi CMYK ke RGB
%    Berdasarkan Pratt (2001)
%    Dasar: b=1 dan u = 0,5

% Normalisasi CMY ke [0, 1]
C = double(C);
M = double(M);
Y = double(Y);
K = double(K);

if max(max(C)) > 1.0 || max(max(M)) > 1.0 || ...
   max(max(Y)) > 1.0 || max(max(K)) > 1.0
    C = double(C) / 255;
    M = double(M) / 255;
    Y = double(Y) / 255;
    K = double(K) / 255;
end

u = 0.5;
b = 1;
[tinggi, lebar] = size(C);
for m=1: tinggi
    for n=1: lebar

        Kb = K(m,n) / b;
        if Kb == 1
            R(m,n)=0;
            G(m,n)=0;
            B(m,n)=0;
        else
            R(m,n) = 1 - (C(m, n) + u * Kb);
            G(m,n) = 1 - (M(m, n) + u * Kb);
            B(m,n) = 1 - (Y(m, n) + u * Kb);
        end
    end
end

% Konversikan ke jangkauan [0,255]
R = uint8(R * 255);
G = uint8(G * 255);
B = uint8(B * 255);
```

**Akhir Program**

Contoh:

```
>> [R,G,B] = CMYkeRGB(64,20,65,40)
R = 171
G = 215
B = 170
>>
```

### 9.2.3 Ruang Warna YIQ

Ruang warna YIQ, yang juga dikenal dengan nama ruang warna NTSC, dirumuskan oleh NTSC ketika mengembangkan sistem televisi berwarna di Amerika Serikat. Pada model ini, Y disebut *luma* (yang menyatakan luminans) dan I serta Q disebut *chroma*. Konversi YIQ berdasarkan RGB sebagai berikut:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ 0{,}596 & -0{,}274 & -0{,}322 \\ 0{,}211 & -0{,}523 & 0{,}312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad (9.2)$$

Komposisi RGB untuk Y secara statistis optimal untuk ditampilkan pada penerima TV hitam-putih.

Matriks yang berisi koefisien konversi mempunyai ciri-ciri sebagai berikut:

1) jumlah elemen dalam baris pertama bernilai 1;
2) jumlah koefisien elemen dalam baris kedua maupun baris ketiga bernilai nol.

Adapun konversi RGB dari YIQ sebagai berikut:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1{,}000 & 0{,}956 & 0{,}621 \\ 1{,}000 & -0{,}272 & -0{,}647 \\ 1{,}000 & -1{,}106 & 1{,}703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix} \quad (9.3)$$

Contoh berikut menunjukkan suatu fungsi yang ditujukan untuk menangani konversi dari RGB ke YIQ.

**Program : RGBkeYIQ.m**

```
function [Y, I, Q] = RGBkeYIQ(R,G,B)
% RGBkeYIQ digunakan untuk mengonversi RGB ke YIQ

% Normalisasi RGB ke [0, 1]
R = double(R);
G = double(G);
```

```
B = double(B);

if max(max(R)) > 1.0 || max(max(G)) > 1.0 || ...
   max(max(B)) > 1.0
    R = double(R) / 255;
    G = double(G) / 255;
    B = double(B) / 255;
end

[tinggi, lebar] = size(R);
for m=1: tinggi
    for n=1: lebar
        Y(m,n) = 0.299*R(m,n)+0.587*G(m,n)+0.114*B(m,n);
        I(m,n) = 0.596*R(m,n)-0.274*G(m,n)-0.322*B(m,n);
        Q(m,n) = 0.211*R(m,n)-0.523*G(m,n)+0.312*B(m,n);
    end
end

% Konversikan ke jangkauan [0,255]
Y = uint8(Y * 255);
I = uint8(I * 255);
Q = uint8(Q * 255);
```

**Akhir Program**

Contoh penggunaan fungsi `RGBkeYIQ`:

```
>> [Y,I,Q] = RGBkeYIQ(171, 20, 250) ↵
Y = 91
I = 16
Q = 104
>>
```

Fungsi kebalikannya, yaitu untuk melakukan konversi dari YIQ ke RGB ditunjukkan berikut ini.

**Program : YIQkeRGB.m**

```
function [R, G, B] = YIQkeRGB(Y,I,Q)
% YIQkeRGB digunakan untuk mengonversi YIQ ke RGB

% Normalisasi YIQ ke [0, 1]
Y = double(Y);
I = double(I);
Q = double(Q);

if max(max(Y)) > 1.0 || max(max(I)) > 1.0 || ...
   max(max(Q)) > 1.0
```

```
    Y = double(Y) / 255;
    I = double(I) / 255;
    Q = double(Q) / 255;
end

[tinggi, lebar] = size(Y);
for m=1: tinggi
    for n=1: lebar
        R(m,n) = Y(m,n)+0.956 * I(m,n) + 0.621 * Q(m,n);
        G(m,n) = Y(m,n)-0.272 * I(m,n) - 0.647 * Q(m,n);
        B(m,n) = Y(m,n)-1.106 * I(m,n) + 1.703 * Q(m,n);
    end
end

% Konversikan ke jangkauan [0,255]
R = uint8(R * 255);
G = uint8(G * 255);
B = uint8(B * 255);
```

**Akhir Program**

Contoh penggunaan fungsi `YIQkeRGB`:

```
>> [Y,I,Q] = YIQkeRGB(48, 16, 43)
Y = 90
I = 16
Q = 104
>>
```

### 9.2.4 Ruang Warna $YC_bC_r$

Ruang warna $YC_bC_r$ biasa digunakan pada video digital. Pada ruang warna ini, komponen Y menyatakan intensitas, sedangkan $C_b$ dan $C_r$ menyatakan informasi warna. Proses konversi dari RGB dilakukan dengan beberapa cara. Contoh berikut didasarkan pada rekomendasi CCIR 601-1 (Crane, 1997):

$$Y = 0{,}29900R + 0{,}58700G + 0{,}11400B \quad (9.4)$$
$$C_b = -0{,}16874R - 0{,}33126G + 0{,}5000B \quad (9.5)$$
$$C_r = +0{,}5000R - 0{,}41869G - 0{,}08131B \quad (9.6)$$

Adapun pengonversian dari $YC_bC_r$ ke RGB sebagai berikut:

$$R = Y + 1.40200C_r \quad (9.7)$$
$$G = Y - 0{,}34414C_b - 0{,}71414C_r \quad (9.8)$$
$$B = Y + 1{,}77200C_b \quad (9.9)$$

Contoh fungsi yang digunakan untuk melakukan konversi dari RGB ke $YC_bC_r$ dapat dilihat berikut ini.

**Program : RGBkeYCB.m**

```
function [Y, Cb, Cr] = RGBkeYCB(R,G,B)
% RGBkeYCB digunakan untuk mengonversi RGB ke YCbCr

% Normalisasi RGB ke [0, 1]
R = double(R);
G = double(G);
B = double(B);

if max(max(R)) > 1.0 || max(max(G)) > 1.0 || ...
   max(max(B)) > 1.0
    R = double(R) / 255;
    G = double(G) / 255;
    B = double(B) / 255;
end

[tinggi, lebar] = size(R);
for m=1: tinggi
    for n=1: lebar
        Y(m,n)=0.299*R(m,n)+0.587*G(m,n)+ 0.114*B(m,n);
        Cb(m,n)=-0.1687*R(m,n)-0.33126*G(m,n)+0.5*B(m,n);
        Cr(m,n)=0.5*R(m,n)-0.41869*G(m,n)-0.08131*B(m,n);
    end
end


Y = Y * 255;
Cb = Cb * 255;
Cr = Cr * 255;
```

**Akhir Program**

Contoh penggunaan fungsi `RGBkeYCB`:

```
>> [Y,Cb,Cr] = RGBkeYCB(9, 16, 250)
Y = 40.5830
Cb = 118.1815
Cr = -22.5265
>>
```

Adapun fungsi yang digunakan untuk mengonversi dari $YC_bC_r$ ke RGB dapat dilihat di bawah ini.

**Program : YCBkeRGB.m**

```
function [R, G, B] = YCBkeRGB(Y,Cb,Cr)
% YCBkeRGB digunakan untuk mengonversi YCbCr ke RGB

% Normalisasi Y, Cb, Cr ke [0, 1]
Y = double(Y);
Cr = double(Cr);
Cb = double(Cb);

if max(max(Y)) > 1.0 || max(max(Cb)) > 1.0 || ...
   max(max(Cr)) > 1.0
    Y = double(Y) / 255;
    Cr = double(Cr) / 255;
    Cb = double(Cb) / 255;
end

[tinggi, lebar] = size(Y);
for m=1: tinggi
    for n=1: lebar
        R(m,n) = Y(m,n) + 1.402 * Cr(m,n);
        G(m,n)= Y(m,n)-0.34414*Cb(m,n)-0.71414*Cr(m,n);
        B(m,n)= Y(m,n) + 1.7720 * Cb(m,n);
    end
end

R = uint8(R * 255);
G = uint8(G * 255);
B = uint8(B * 255);
```

**Akhir Program**

Contoh penggunaan:

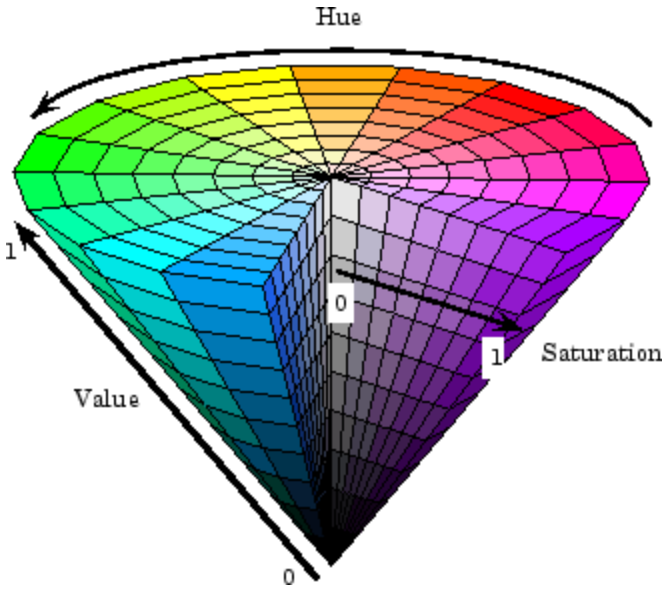
```
>> [R,G,B] = YCBkeRGB(40.5830, 118.1815, -22.5265)
R = 9
G = 16
B = 250
>>
```

### 9.2.5 Ruang Warna HSI, HSV, dan HSL

HSV dan HSL merupakan contoh ruang warna yang merepresentasikan warna seperti yang dilihat oleh mata manusia. H berasal dari kata "hue", S berasal dari "saturation", L berasal dari kata "luminance", I berasal dari kata "intensity", dan V berasal dari "value".

**Catatan**

Ruang warna HLS terkadang disebut HSL, sedangkan HSV terkadang dinamakan HSB, dengan B berasal dari kata "brightness".




**Gambar 9.4 Ruang warna HSV**
*(Sumber: MATLAB)*

Model HSV, yang pertama kali diperkenalkan A. R. Smith pada tahun 1978, ditunjukkan pada Gambar 9.4. Untuk mendapatkan nilai H, S, V berdasarkan R, G, dan B, terdapat beberapa cara. Cara yang tersederhana (Acharya & Ray, 2005) adalah seperti berikut.

$$H = \tan\left(\frac{3(G-B)}{(R-G)+(R-B)}\right) \quad (9.10)$$

$$S = 1 - \frac{\min(R,G,B)}{V} \quad (9.11)$$

$$V = \frac{R+G+B}{3} \quad (9.12)$$

Namun, cara ini membuat *hue* tidak terdefinisikan kalau S bernilai nol. Cara kedua terdapat pada Acharya & Ray (2005). Rumus-rumus yang digunakan sebagai berikut:

$$r = \frac{R}{(R+G+B)}, g = \frac{G}{(R+G+B)}, b = \frac{B}{(R+G+B)} \quad (9.13)$$

$$V = \max(r, g, b) \tag{9.14}$$

$$S = \begin{cases} 0, & \text{jika V} = 0 \\ 1 - \frac{\min(r,g,b)}{V}, & \text{V} > 0 \end{cases} \tag{9.15}$$

$$H = \begin{cases} 0, & jika\ S = 0 \\ \frac{60*(g-b)}{S*V}, & jika\ V = r \\ 60 * \left[2 + \frac{b-r}{S*V}\right], & jika\ V = g \\ 60 * \left[4 + \frac{r-g}{S*V}\right], & jika\ V = b \end{cases} \tag{9.16}$$

$$H = \text{H} + 360 \text{ jika H} < 0 \tag{9.17}$$

Implementasi berikut didasarkan pada rumus-rumus di atas.

**Program : RGBkeHSV.m**

```
function [H,S,V] = RGBkeHSV(R,G,B)
% RGBkeHSV digunakan untuk mengonversi RGB ke HSV.
%    Algoritma berdasarkan Acharya & Ray (2005)

% Normalisasi RGB ke [0, 1]
R = double(R);
G = double(G);
B = double(B);

if max(max(R)) > 1.0 || max(max(G)) > 1.0 || ...
   max(max(B)) > 1.0
    R = double(R) / 255;
    G = double(G) / 255;
    B = double(B) / 255;
end

[tinggi, lebar] = size(R);
for m=1: tinggi
    for n=1: lebar
        minrgb = min([R(m,n) G(m,n) B(m,n)]);
        maxrgb = max([R(m,n) G(m,n) B(m,n)]);
        V(m,n) = maxrgb;
        delta = maxrgb - minrgb;

        if maxrgb == 0
            S(m,n) = 0;
        else
            S(m,n) = 1 - minrgb / maxrgb;
        end

        if S(m,n) == 0
            H(m,n) = 0;
        else
```

```
            SV = S(m,n) * V(m,n);

            if R(m,n) == maxrgb
                % Di antara kuning dan magenta
                H(m,n) = (G(m,n)-B(m,n)) / SV;
            elseif G(m,n) == maxrgb
                % Di antara cyan dan kuning
                H(m,n) = 2 + (B(m,n)-R(m,n)) / SV;
            else
                % Di antara magenta dan cyan
                H(m,n) = 4 + (R(m,n)-G(m,n)) / SV;
            end

            H(m,n) = H(m,n) * 60;
            if H(m,n) < 0
                H(m,n) = H(m,n)+360;
            end
        end
    end
end

% Konversikan ke jangkauan [0, 255] atau [0, 360]
H = uint8(H * 255/360);
S = uint8(S * 255);
V = uint8(V * 255);
```

**Akhir Program**

Proses untuk mengonversi HSV ke RGB dapat dilihat di bawah ini.

**Program : HSVkeRGB.m**

```
function [R,G,B] = HSVkeRGB(H,S,V)
% HSVkeRGB digunakan untuk mengonversi HSV ke RGB

% Normalisasi SV ke [0, 1] dan H ke [0, 360]
H = double(H);
S = double(S);
V = double(V);

if max(max(H)) > 1.0 || max(max(S)) > 1.0  || ...
   max(max(V)) > 1.0
    H = double(H) / 255 * 360;
    S = double(S) / 255;
    V = double(V) / 255;
end

[tinggi, lebar] = size(H);
for m=1: tinggi
    for n=1: lebar
        if S(m,n) == 0
            R(m,n) = V(m,n);
```

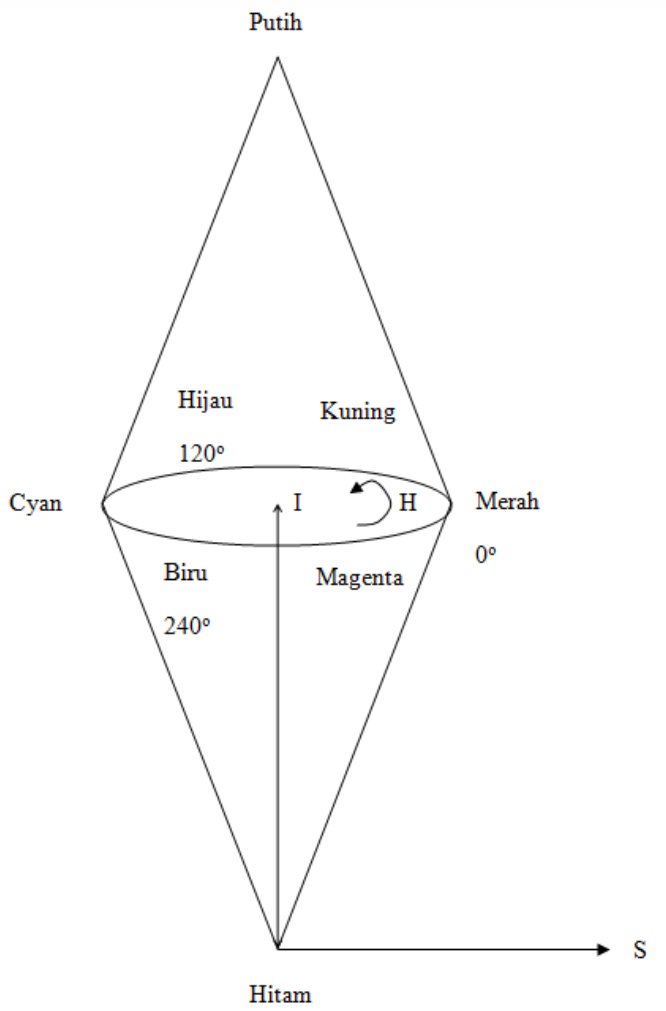
```
            G(m,n) = V(m,n);
            B(m,n) = V(m,n);
        else
            % S != 0

            % Menghitung posisi sektor (0 s/d 5)
            H(m,n) = H(m,n) / 60;
            sektor = floor(H(m, n));
            faktor = H(m,n) - sektor;
            p = V(m,n) * (1 - S(m,n));
            q = V(m,n) * (1 - S(m,n) * faktor);
            t = V(m,n) * (1 - S(m,n) * (1 - faktor));

            switch sektor
                case 0
                    R(m,n) = V(m,n);
                    G(m,n) = t;
                    B(m,n) = p;
                case 1
                    R(m,n) = q;
                    G(m,n) = V(m,n);
                    B(m,n) = p;
                case 2
                    R(m,n) = p;
                    G(m,n) = V(m,n);
                    B(m,n) = t;
                case 3
                    R(m,n) = p;
                    G(m,n) = q;
                    B(m,n) = V(m,n);
                case 4
                    R(m,n) = t;
                    G(m,n) = p;
                    B(m,n) = V(m,n);
                otherwise % case 5
                    R(m,n) = V(m,n);
                    G(m,n) = p;
                    B(m,n) = q;
            end
        end
    end
end

R = uint8(R * 255);
G = uint8(G * 255);
B = uint8(B * 255);
```

**Akhir Program**

Berikut adalah contoh pemanggilan fungsi RGBkeHSV dan HSVkeRGB :

```
>> [H,S,V] = RGBkeHSV(100, 120, 80) ↵
H = 64
S = 85
```

```
V = 120
>> [R,G,B] = HSVkeRGB(64, 85, 120)
R = 100
G = 120
B = 80
>>
```




**Gambar 9.5 Model HSI**

Gambar 9.5 memperlihatkan ruang warna HSI. Konversi dari RGB ke HSI dilakukan melalui rumus berikut (Gonzalez & Woods, 2002):

$$H = \begin{cases} 0, \ jika B \leq G \\ 360 - \theta, \ jika \ B > G \end{cases} \tag{9.18}$$

Pada rumus di atas, H menyatakan *hue*. Adapun θ diperoleh melalui rumus berikut:

$$\theta = cos^{-1}\left\{\frac{1/2[(R-G)+(R-B)]}{[(R-G)^2(R-B)(G-B)]^{1/2}}\right\} \quad (9.19)$$

Selanjutnya, komponen *saturation* dihitung dengan menggunakan rumus:

$$S = 1 - \frac{3}{(R+G+B)}[\min(R,G,B)] \quad (9.20)$$

dan komponen intensitas diperoleh melalui:

$$I = \frac{1}{3}(R+G+B) \quad (9.21)$$

Untuk memperoleh RGB berdasarkan HSI, diperlukan beberapa aturan. Apabila H berada dalam sektor RG ($0^o \leq H < 120^o$), komponen R, G, dan B dihitung dengan menggunakan rumus-rumus berikut:

$$B = I(1-S) \quad (9.22)$$
$$R = I\left(1 + \frac{S\cos H}{\cos(60^o - H)}\right) \quad (9.23)$$
$$G = 3I - (R+B)) \quad (9.24)$$

Apabila H berada di dalam sektor GB ($120^o \leq H < 240^o$), komponen R, G, dan B dihitung dengan menggunakan rumus-rumus berikut:

$$H = H - 120 \quad (9.25)$$
$$R = I(1-S) \quad (9.26)$$
$$G = I\left(1 + \frac{S\cos H}{\cos(60^o - H)}\right) \quad (9.27)$$
$$B = 3I - (R+G)) \quad (9.28)$$

Apabila H berada di dalam sektor GB ($240^o \leq H < 360^o$), komponen R, G, dan B dihitung dengan menggunakan rumus-rumus berikut:

$$H = H - 240 \quad (9.29)$$
$$G = I(1-S) \quad (9.30)$$
$$B = I\left(1 + \frac{S\cos H}{\cos(60^o - H)}\right) \quad (9.31)$$
$$R = 3I - (R+G)) \quad (9.32)$$

Perlu diketahui, mengingat nilai pada HSI berada di dalam jangkauan [0, 1], maka untuk mendapatkan nilai H yang berkisar antara $0^o$-$360^o$, H perlu dikalikan terlebih dulu dengan 360. Dengan demikian, jangkauan H berada dalam [0, 360].

Contoh berikut merupakan perwujudan fungsi yang ditujukan untuk melakukan konversi dari RGB ke HSI.

**Program : RGBkeHSI.m**

```
function [H,S,I] = RGBkeHSI(R,G,B)
% RGBkeHSI digunakan untuk mengonversi RGB ke HSI.

% Normalisasi RGB ke [0, 1]
R = double(R);
G = double(G);
B = double(B);

if max(max(R)) > 1.0 || max(max(G)) > 1.0 || ...
   max(max(B)) > 1.0
    R = double(R) / 255;
    G = double(G) / 255;
    B = double(B) / 255;
end

[tinggi, lebar] = size(R);
for m=1: tinggi
    for n=1: lebar
        minrgb = min([R(m,n) G(m,n) B(m,n)]);
        I(m,n) = (R(m,n) + G(m,n) + B(m,n)) / 3.0;

        if R(m,n) == G(m,n) && G(m,n) == B(m,n)
            S(m,n) = 0;
            H(m,n) = 0;
        else
            S(m,n) = 1 - 3 * minrgb / ...
                     (R(m,n)+G(m,n)+B(m,n));

            y = (R(m,n)-G(m,n)+R(m,n)-B(m,n))/2;
            x = (R(m,n)-G(m,n))*(R(m,n)-G(m,n)) + ...
                (R(m,n)-B(m,n)) * (G(m,n)-B(m,n));
            x = sqrt(x);
            sudut = acos(y/x) * 180/pi;
            if B(m,n) > G(m,n)
                H(m, n) = 360 - sudut;
            else
                H(m,n) = sudut;
            end
        end
    end
end

% Konversikan ke jangkauan [0, 255] dan [0, 360]
H = uint8(H * 255/360);
S = uint8(S * 255);
I = uint8(I * 255);
```

**Akhir Program**

Adapun fungsi `HSIkeRGB` ditujukan untuk mengonversi data pada ruang HSI ke RGB. Implementasinya seperti berikut.

**Program : HSIkeRGB.m**

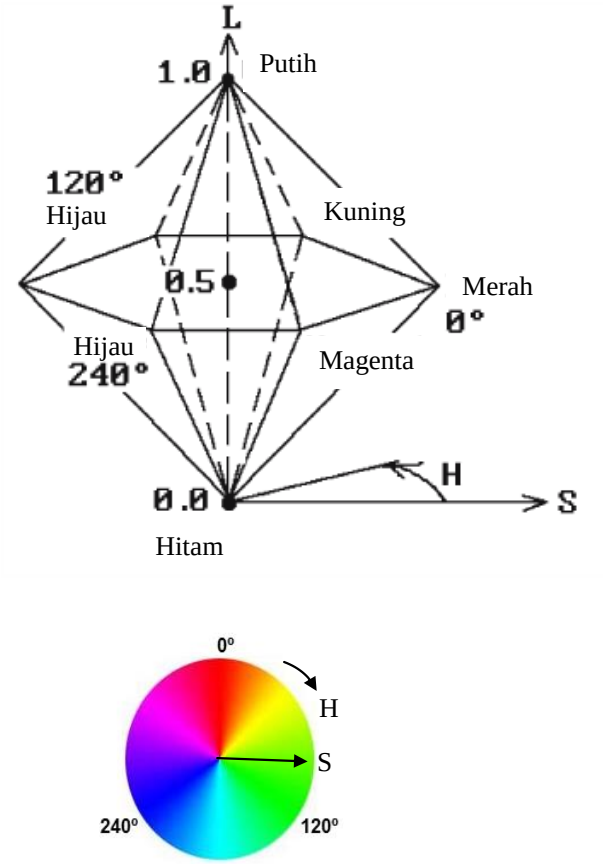
```
function [R,G,B] = HSIkeRGB(H,S,I)
% HSIkeRGB digunakan untuk mengonversi HSI ke RGB.

% Normalisasi HSI ke [0, 1]
H = double(H);
S = double(S);
I = double(I);

if max(max(H)) > 1.0 || max(max(S)) > 1.0 || ...
   max(max(I)) > 1.0
    H = double(H) / 255 * 360;
    S = double(S) / 255;
    I = double(I) / 255;
end

[tinggi, lebar] = size(H);
for m=1: tinggi
    for n=1: lebar
        if I(m,n) == 0
            R(m,n) = 0;
            G(m,n) = 0;
            B(m,n) = 0;
        elseif S(m,n) == 0
            R(m,n) = I(m,n);
            G(m,n) = I(m,n);
            B(m,n) = I(m,n);
        else
            if H(m,n) < 0
                H(m,n) = H(m,n) + 360.0;
            end

            skala = 3 * I(m,n);
            if H(m,n) <= 120
                sudut1 = H(m,n) * 0.017453292;
                sudut2 = (60 - H(m,n)) * 0.017453292;
                B(m,n) = (1 - S(m,n)) / 3;
                R(m,n) = (1 + (S(m,n) * cos(sudut1)/...
                         cos(sudut2))) / 3;
                G(m,n) = 1 - R(m,n) - B(m,n);

                B(m,n) = B(m,n) * skala;
                R(m,n) = R(m,n) * skala;
                G(m,n) = G(m,n) * skala;
            elseif H(m,n) <= 240
```

```
                H(m,n) = H(m,n) - 120;

                sudut1 = H(m,n) * 0.017453292;
                sudut2 = (60 - H(m,n)) * 0.017453292;
                R(m,n) = (1 - S(m,n)) / 3;
                G(m,n) = (1 + (S(m,n) * cos(sudut1)/...
                          cos(sudut2))) / 3;
                B(m,n) = 1 - R(m,n) - G(m,n);

                R(m,n) = R(m,n) * skala;
                G(m,n) = G(m,n) * skala;
                B(m,n) = B(m,n) * skala;
            else
                H(m,n) = H(m,n) - 240;

                sudut1 = H(m,n) * 0.017453292;
                sudut2 = (60 - H(m,n)) * 0.017453292;
                G(m,n) = (1 - S(m,n)) / 3;
                B(m,n) = (1 + (S(m,n) * cos(sudut1)/...
                          cos(sudut2))) / 3;
                R(m,n) = 1 - G(m,n) - B(m,n);

                G(m,n) = G(m,n) * skala;
                B(m,n) = B(m,n) * skala;
                R(m,n) = R(m,n) * skala;

            end
        end
    end
end

% Konversikan ke jangkauan [0, 255]
R = uint8(R * 255);
G = uint8(G * 255);
B = uint8(B * 255);
```

**Akhir Program**

Contoh penggunaan `HSIkeRGB` dan `RGBkeHSI` ditunjukkan di bawah ini.

```
>> [H,S,I] = HSIkeRGB(156, 146, 140) ↵
H = 0.2347
S = 0.4759
I = 0.9365
>>
>> [R,G,B] = HSIkeRGB(156, 146, 140) ↵
R = 60
G = 121
B = 239
>>
```




**Gambar 9.6 Model HSL**

Model ruang HSL diperlihatkan pada Gambar 9.6. Besaran kecerahan (dinamakan *lightness*) disimbolkan dengan L. Perhitungan komponen H, S, dan L berdasarkan komponen R,G, dan B adalah seperti berikut (Agoston, 2005).

$$C_{min} = \min(R, G, B) \tag{9.33}$$

$$C_{max} = \max(R, G, B) \tag{9.34}$$

$$L = \frac{C_{max} + C_{min}}{2} \tag{9.35}$$

$$S = \begin{cases} 0, & C_{min} = C_{max} \\ \frac{C_{max} - C_{min}}{C_{max} + C_{min}}, & L \leq 0{,}5 \\ \frac{C_{max} - C_{min}}{2 - (C_{max} + C_{\min})}, & L > 0{,}5 \end{cases} \tag{9.36}$$

$$H = \begin{cases} UNDEFINED, & C_{min} = C_{max} \\ \frac{G - B}{C_{max} - C_{min}}, & R = C_{max} \\ 2 + \frac{B - R}{C_{max} - C_{min}}, & G = C_{max} \\ 4 + \frac{R - G}{C_{max} - C_{min}}, & untuk\ lainnya \end{cases} \tag{9.37}$$

$$H = H\ x\ 60 \quad (9.38)$$

$$Jika\ H < 0\ maka\ H = H + 360 \quad (9.39)$$

Fungsi yang ditujukan untuk mengonversi dari RGB ke HSL dapat dilihat di bawah ini.

**Program : RGBkeHSL.m**

```
function [H,S,L] = RGBkeHSL(R,G,B)
% RGBkeHSL digunakan untuk mengonversi RGB ke HSL.
%    Berdasarkan algoritma Max K. Agoston (2005)

% Normalisasi RGB
R = double(R);
G = double(G);
B = double(B);

if max(max(R)) > 1.0 || max(max(G)) > 1.0 || ...
   max(max(B)) > 1.0
    R = double(R) / 255;
    G = double(G) / 255;
    B = double(B) / 255;
end

[tinggi, lebar] = size(R);
for m=1: tinggi
    for n=1: lebar
        minrgb = min([R(m,n) G(m,n) B(m,n)]);
        maxrgb = max([R(m,n) G(m,n) B(m,n)]);

        if maxrgb == minrgb
            S(m,n) = 0;
            H(m,n) = 0; % Cek microsoft
        else
            L(m,n) = (minrgb + maxrgb) / 2;
            d = (maxrgb - minrgb);

            if L(m,n) <= 0.5
                S(m,n) = d / (maxrgb + minrgb);
            else
                S(m,n) =  d / (2 - minrgb - maxrgb);
            end

            % Tentukan hue
            if R(m,n) == maxrgb
                % Warna antara kuning dan magenta
                H(m,n) = (G(m,n)-B(m,n))/d;
            elseif G(m,n) == maxrgb
                % Warna antara cyan dan kuning
                H(m,n) = 2+(B(m,n)-R(m,n))/d;
            else
                % warna antara magenta dan cyan
```

```
                H(m,n) = 4+(R(m,n)-G(m,n))/d;
            end

            H(m,n) = H(m,n) * 60;
            if H(m,n) < 0
                H(m,n) = H(m,n) + 360;
            end
        end
    end
end

% Konversikan ke jangkauan [0, 255] atau [0, 360]
H = uint8(H * 255/360);
S = uint8(S * 255);
L = uint8(L * 255);
```

**Akhir Program**

Adapun fungsi yang ditujukan untuk mengonversi dari HSL ke RGB dapat dilihat di bawah ini.

**Program : HSLkeRGB.m**

```
function [R,G,B] = HSLkeRGB(H,S,L)
% HSLkeRGB digunakan untuk mengonversi HSL ke RGB.
%    Berdasarkan algoritma Max K. Agoston (2005)

% Normalisasi HSL
H = double(H);
S = double(S);
L = double(L);

if max(max(H)) > 1.0 || max(max(S)) > 1.0 || ...
   max(max(L)) > 1.0
    H = double(H) / 255 * 360;
    S = double(S) / 255;
    L = double(L) / 255;
end

[tinggi, lebar] = size(H);
for m=1: tinggi
    for n=1: lebar
        if L(m,n) <= 0.5
            v = L(m,n) * (1 + S(m,n));
        else
            v = L(m,n) + S(m,n) - L(m,n) * S(m,n);
        end

        if v == 0
            R(m,n) = 0;
            G(m,n) = 0;
```

```
            B(m,n) = 0;
        else
            terkecil = 2 * L(m,n) - v;
            sv = (v - terkecil) / v;
            if H(m,n) == 360
                H(m,n) = 0;
            else
                H(m,n) = H(m,n) / 60;
            end

            sektor = floor(H(m,n)); % 0-5
            frak = H(m,n) - sektor;
            vsf = v * sv * frak;
            mid1 =  terkecil + vsf;
            mid2 = v - vsf;
            switch sektor
                case 0
                    R(m,n) = v;
                    G(m,n) = mid1;
                    B(m,n) = terkecil;
                case 1
                    R(m,n) = mid2;
                    G(m,n) = v;
                    B(m,n) = terkecil;
                case 2
                    R(m,n) = terkecil;
                    G(m,n) = v;
                    B(m,n) = mid1;
                case 3
                    R(m,n) = terkecil;
                    G(m,n) = mid2;
                    B(m,n) = v;
                case 4
                    R(m,n) = mid1;
                    G(m,n) = terkecil;
                    B(m,n) = v;
                case 5
                    R(m,n) = v;
                    G(m,n) = terkecil;
                    B(m,n) = mid2;
            end
        end
    end
end

% Konversikan ke jangkauan [0, 255]
R = uint8(R * 255);
G = uint8(G * 255);
B = uint8(B * 255);
```

**Akhir Program**

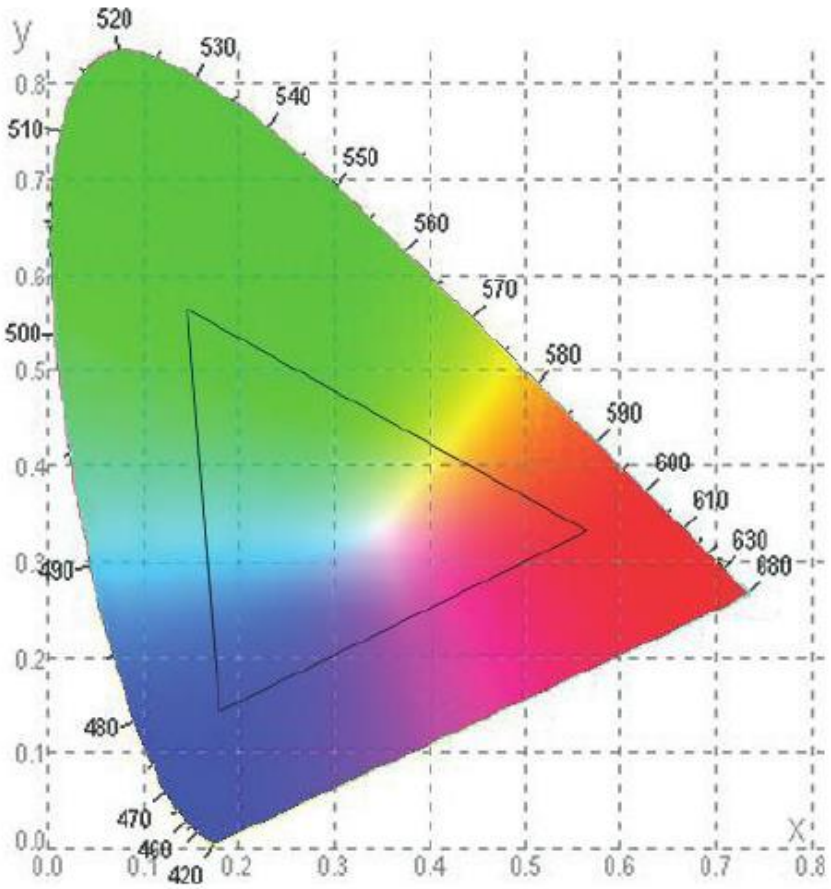
Contoh penggunaan `RGBkeHSL` dan `HSLkeRGB`:

```
>> [H,S,L] = RGBkeHSL(60, 120, 240) ↵
```

```
H = 156
S = 219
L = 150
>>
>> [R,G,B] = HSLkeRGB(156, 219, 150)
R = 60
G = 119
B = 240
>>
```

### 9.2.6 Ruang Warna CIELAB

CIELAB adalah nama lain dari CIE L*a*b*. Diagram kromasitas CIE (Commission Internatiole de L'Eclairage) ditunjukkan pada Gambar 9.7. Pada diagram tersebut, setiap perpaduan x dan y menyatakan suatu warna. Namun, hanya warna yang berada dalam area ladam (tapal kuda) yang bisa terlihat. Angka yang berada di tepi menyatakan panjang gelombang cahaya. Warna yang terletak di dalam segitiga menyatakan warna-warna umum di monitor CRT, yang dapat dihasilkan oleh komponen warna merah, hijau, dan biru.




**Gambar 9.7 Diagram kromasitas CIE**
**(Sumber: Russ, 2011)**

Transformasi RGB ke CIELAB dimulai dengan melakukan perhitungan sebagai berikut:

$$X = 0{,}412453R + 0{,}357580G + 0{,}180423B \quad (9.40)$$
$$Y = 0{,}212671R + 0{,}715160G + 0{,}072169B \quad (9.41)$$
$$Z = 0{,}019334R + 0{,}119193G + 0{,}950227B \quad (9.42)$$

Selanjutnya, L*a*b* didefinisikan sebagai berikut:

$$L^* = 116f\left(\frac{Y}{Y_n}\right) - 16 \quad (9.43)$$
$$a^* = 500\left[f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right)\right] \quad (9.44)$$
$$b^* = 200\left[f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right)\right] \quad (9.45)$$

Dalam hal ini, f(q) dihitung seperti berikut:

$$f(q) = \begin{cases} q^{\frac{1}{3}}, \ jika\ q > 0{,}008856 \\ 7{,}787q + 16/116, \ untuk\ yang\ lain \end{cases} \quad (9.46)$$

$X_n$, $Y_n$, $Z_n$ diperoleh melalui R=G=B=1 dengan jangkauan R,G, B berupa [0, 1].

Contoh untuk melakukan transformasi dari RGB ke CIELAB ditunjukkan di bawah ini.

**Program : RGBkeLab.m**

```
function [L,a,b] = RGBkeLab(R,G,B)
% RGBkeLab digunakan untuk mengonversi RGB ke CIELAB
% Nilai balik:
%    L -> [1, 100]
%    a dan b -> [-110,110]

R = double(R);
G = double(G);
B = double(B);

if max(max(R)) > 1.0 || max(max(G)) > 1.0 || ...
   max(max(B)) > 1.0
  R = double(R) / 255;
  G = double(G) / 255;
  B = double(B) / 255;
end

% RGB to XYZ
Koef = [0.412453 0.357580 0.180423;
```

```
        0.212671 0.715160 0.072169;
        0.019334 0.119193 0.950227];

RGB =  [R, G, B]';

XYZ = Koef * RGB;

% Peroleh Xq=X/Xn, Yq=Y/Yn, Zq=Z/Zn
%    dengan R=G=B=1 untuk menghitung Xn, Yn, Zn
Xq = XYZ(1,:) / 0.950456;
Yq = XYZ(2,:);
Zq = XYZ(3,:) / 1.088754;

[tinggi, lebar] = size(B);
for m=1 : tinggi
    for n=1 : lebar
        fqx = fq(Xq(m, n));
        fqy = fq(Yq(m, n));
        fqz = fq(Zq(m, n));

        L(m, n) = 116.0 * fqy - 16;
        a(m, n) = 500.0 * (fqx - fqy);
        b(m, n) = 200.0 * (fqy - fqz);
    end
end

return

function hasil = fq(q)
% Untuk menghitung f(q)

if q > 0.008856
    hasil = q ^(1/3);
else
    hasil = 7.787*q + 16/116.0;
end
```

**Akhir Program**

Fungsi di atas dapat digunakan untuk menguji per piksel ataupun beberapa piksel. Contoh berikut menunjukkan pengujian satu piksel dengan R=0,5, G=0,3, dan B=0,1:

```
>> [L,A,B] = RGB2Lab(0.5,0.3,0.1)
L = 64.0068
A = 7.1133
B = 36.8877
>>
```

Contoh lain:

```
>> [L,a,b] = RGBkeLab(127, 76, 25) ↵
L = 63.8471
a = 7.1409
b = 37.1270
>>
```

Contoh berikut menunjukkan fungsi yang digunakan untuk mengonversikan CIELAB ke RGB.

**Program : LabkeRGB.m**

```
function [R, G, B] = LabkeRGB(L, a, b)
% LabkeRGB Mengonversi CIELAB ke RGB

[tinggi, lebar] = size(L);

% Peroleh Xq=X/Xn, Yq=Y/Yn, Zq=Z/Zn
for m=1 : tinggi
    for n=1 : lebar
        fqy = (L(m, n) + 16) / 116.0;
        fqx = a(m, n) / 500.00 + fqy;
        fqz = fqy - b(m, n) / 200.0;

        Xq(m, n) = peroleh_q(fqx);
        Yq(m, n) = peroleh_q(fqy);
        Zq(m, n) = peroleh_q(fqz);

    end
end

% Hitung X, Y, dan Z
XYZ(1,:) = Xq * 0.950456;
XYZ(2,:) = Yq;
XYZ(3,:) = Zq * 1.088754;

% XYZ to RGB
Koef = [ 3.240479 -1.537150 -0.498535;
        -0.969256  1.875992  0.041556;
         0.055648 -0.204043  1.057311];
RGB = Koef * XYZ;
R = uint8(RGB(1, :) * 255);
G = uint8(RGB(2, :) * 255);
B = uint8(RGB(3, :) * 255);

return

function q = peroleh_q(fq)
% Peroleh nilai q

q = fq ^ 3;
if q > 0.008856
```

```
    hasil = q;
else
    q = (fq - 16 / 116.0) / 7.787;
end
```

**Akhir Program**

Contoh penggunaan `LabkeRGB`:

```
>> [R,G,B] = LabkeRGB(63.8471, 7.1409, 37.1270)
R = 127
G = 76
B = 25
>>
```

### 9.3 Memperoleh Statistika Warna

Fitur warna dapat diperoleh melalui perhitungan statistis seperti rerata, deviasi standar, *skewness*, dan kurtosis (Martinez & Martinez, 2002). Sebagai contoh, fitur-fitur tersebut dapat digunakan untuk kepentingan identifikasi tanaman hias (Kadir, dkk., 2011b dan Kadir, dkk., 2011c). Perhitungan dikenakan pada setiap komponen R, G. dan B.

Rerata memberikan ukuran mengenai distribusi dan dihitung dengan menggunakan rumus:

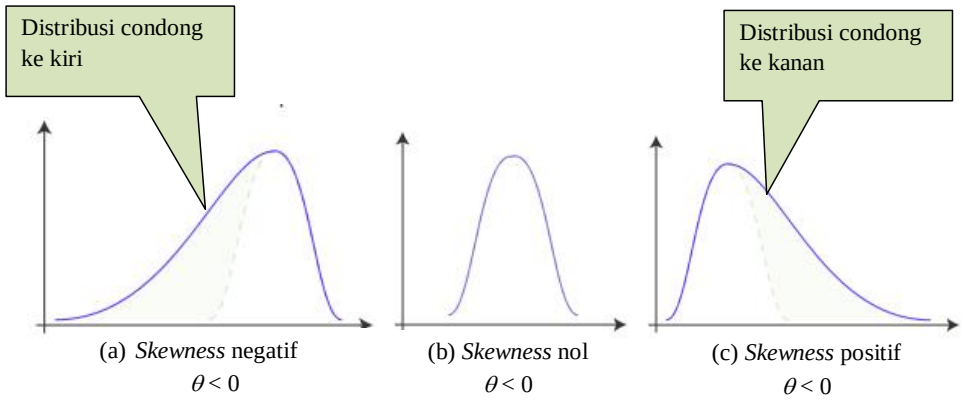
$$\mu = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N} P_{ij} \tag{9.47}$$

Varians menyatakan luas sebaran distribusi. Akar kuadrat varians dinamakan sebagai deviasi standar. Adapun rumus yang digunakan untuk menghitungnya sebagai berikut:

$$\sigma = \sqrt{\frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(P_{ij}-\mu)^2} \tag{9.48}$$

*Skewness* atau kecondongan menyatakan ukuran mengenai ketidaksimetrisan. Distribusi dikatakan condong ke kiri apabila memiliki nilai *skewness* berupa bilangan negatif. Sebaliknya, distribusi dikatakan condong ke kanan apabila memiliki nilai *skewness* berupa bilangan positif. Jika distribusi simetris, koefisien *skewness* bernilai nol. Ilustrasi *skewness* dapat dilihat pada Gambar 9.8. *Skewness* dihitung dengan cara seperti berikut:

$$\theta = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}(P_{ij}-\mu)^3}{MN\sigma^3} \tag{9.49}$$

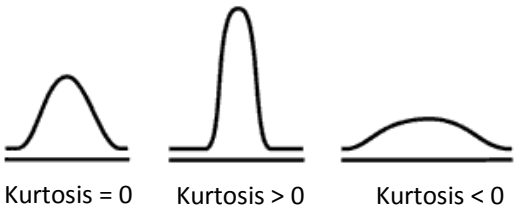



**Gambar 9.8 Skewness menggambarkan kecondongan distribusi data**

Kurtosis merupakan ukuran yang menunjukkan sebaran data bersifat meruncing atau menumpul. Perhitungannya seperti berikut:

$$\gamma = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}(P_{ij}-\mu)^4}{MN\sigma^4} - 3 \tag{9.50}$$

Definisi di atas membuat distribusi normal standar memiliki kurtosis nol. Nilai positif mengindikasikan bahwa distribusi bersifat lancip dan nilai negatif menyatakan distribusi yang datar (lihat Gambar 9.9). Perlu diketahui, pada Persamaan 9.47 hingga 9.50, M adalah tinggi citra, N menyatakan lebar citra, dan $P_{ij}$ adalah nilai warna pada baris $i$ dan kolom $j$.




**Gambar 9.9 Kurtosis menggambarkan keruncingan distribusi normal**

Contoh statwarna.m berikut mewujudkan fungsi yang dapat digunakan untuk memperoleh statistika warna.

**Program : statwarna.m**

```
function [stat] = statwarna(berkas)
% STATWARNA Menghitung statistik warna pada citra RGB
%     Masukan:
%        berkas berupa citra warna
%     Nilai balik berupa statistik warna yang mencakup
%        rerata, deviasi standar, kecondongan, dan
%        kurtosis

RGB = double(imread(berkas));
[m,n,d] = size(RGB);
if (d ~= 3)
    disp('Citra harus berupa citra berwarna');
    return;
end

% --- Peroleh warna
% Hitung warna rata-rata R, G, dan B
jum_r=0;
jum_g=0;
jum_b=0;
jum_piksel = m * n;
for baris = 1:m
    for kolom = 1:n
          jum_r = jum_r + RGB(baris, kolom, 1);
          jum_g = jum_g + RGB(baris, kolom, 2);
          jum_b = jum_b + RGB(baris, kolom, 3);
     end
end

% Hitung rerata
mean_r = jum_r / jum_piksel;
mean_g = jum_g / jum_piksel;
mean_b = jum_b / jum_piksel;

% Inisialisasi perhitungan deviasi standar,
%    skewness, dan kurtosis
jum_dev_r = 0;
jum_dev_g = 0;
jum_dev_b = 0;

jum_skew_r = 0;
jum_skew_g = 0;
jum_skew_b = 0;

jum_cur_r = 0;
jum_cur_g = 0;
jum_cur_b = 0;

for baris = 1:m
    for kolom = 1:n
        jum_dev_r = jum_dev_r + ...
            (RGB(baris, kolom,1) - mean_r)^2;
        jum_dev_g = jum_dev_g + ...
```

```
            (RGB(baris, kolom,2) - mean_g)^2;
        jum_dev_b = jum_dev_b + ...
            (RGB(baris, kolom,3) - mean_b)^2;

        jum_skew_r = jum_skew_r + ...
            (RGB(baris, kolom,1) - mean_r)^3;
        jum_skew_g = jum_skew_g + ...
            (RGB(baris, kolom,2) - mean_g)^3;
        jum_skew_b = jum_skew_b + ...
            (RGB(baris, kolom,3) - mean_b)^3;

        jum_cur_r = jum_cur_r + ...
            (RGB(baris, kolom,1) - mean_r)^4;
        jum_cur_g = jum_cur_g + ...
            (RGB(baris, kolom,2) - mean_g)^4;
        jum_cur_b = jum_cur_b + ...
            (RGB(baris, kolom,3) - mean_b)^4;
    end
end

% Hitung deviasi standar
dev_r = sqrt(jum_dev_r/jum_piksel);
dev_g = sqrt(jum_dev_g/jum_piksel);
dev_b = sqrt(jum_dev_b/jum_piksel);

% Hitung skewness
skew_r = jum_skew_r/ (jum_piksel * (dev_r^3));
skew_g = jum_skew_g/ (jum_piksel * (dev_g^3));
skew_b = jum_skew_b/ (jum_piksel * (dev_b^3));

% Hitung kurtosis
cur_r = jum_cur_r / (jum_piksel * (dev_r^4)) - 3;
cur_g = jum_cur_g / (jum_piksel * (dev_g^4)) - 3;
cur_b = jum_cur_b / (jum_piksel * (dev_b^4)) - 3;

% Tentukan keluaran
stat.mean_r = mean_r;
stat.mean_g = mean_g;
stat.mean_b = mean_b;

stat.dev_r = dev_r;
stat.dev_g = dev_g;
stat.dev_b = dev_b;

stat.skew_r = skew_r;
stat.skew_g = skew_g;
stat.skew_b = skew_b;

stat.cur_r = cur_r;
stat.cur_g = cur_g;
stat.cur_b = cur_b;
```

**Akhir Program**

Contoh pemakaian `statwarna`:

```
>> S = statwarna('C:\Image\lapangan.png') ↵
S =

  scalar structure containing the fields:

    mean_r =  124.19
    mean_g =  116.05
    mean_b =  115.10
    dev_r =  61.079
    dev_g =  64.549
    dev_b =  66.176
    skew_r =  0.86789
    skew_g =  1.0295
    skew_b =  1.1270
    cur_r = -0.39905
    cur_g = -0.31038
    cur_b = -0.026880

>>
>> S = statwarna('C:\Image\innsbruckcity.png') ↵
S =

  scalar structure containing the fields:

    mean_r =  114.17
    mean_g =  116.22
    mean_b =  117.40
    dev_r =  79.905
    dev_g =  83.627
    dev_b =  88.439
    skew_r =  0.36087
    skew_g =  0.44561
    skew_b =  0.46497
    cur_r = -1.4307
    cur_g = -1.4784
    cur_b = -1.5103

>>
```

Perhatikan bahwa berdasarkan contoh di atas, kedua citra (lapangan.png dan innsbruckcity.png) mempunyai statistika warna yang jauh berbeda. Perbedaan seperti itulah yang dapat digunakan untuk membedakan antara satu citra dengan citra yang lain.

## 9.4 Mengatur Kecerahan dan Kontras

Pada Bab 3 telah dijelaskan cara mengatur kontras dan kecerahan pada citra berskala keabuan. Cara seperti itu, secara prinsip dapat diterapkan pada citra

berwarna. Untuk melihat efek kecerahan dan kontras, cobalah beberapa perintah berikut.

```
>> Img = imread('C:\Image\inns.png'); ↵
>> imshow(Img) ↵
>>
```

Kode di atas digunakan untuk melihat citra inns.png (Gambar 9.10.(a)). Selanjutnya,

```
>> C = Img + 30; ↵
>> imshow(C) ↵
>>
```

membuat citra dicerahkan sejauh 30. Hasilnya ditunjukkan pada Gambar 9.10(b). Lalu, cobalah kode berikut:

```
>> K = 2 * C; ↵
>> imshow(K) ↵
>>
```

Hasilnya dapat dilihat pada Gambar 9.10(c). Hal yang menarik dapat diperoleh dengan hanya memberikan kontras pada komponen R. Caranya:

```
>> K = C; ↵
>> K(:,:,1) = 2 * K(:,:,1); ↵
>>
```

Kode di atas mula-mula membuat K bernilai sama dengan C (efek pencerahan). Selanjutnya,

```
K(:,:,1) = 2 * K(:,:,1);
```

membuat hanya komponen R saja yang dinaikkan dua kali. hasilnya ditunjukkan pada Gambar 9.10(d).

(a) Citra inns.png

(b) Efek pencerahan melalui C = Img + 30;

(c) Efek kontras dan pencerahan melalui C = 2 x (Img + 30)

(d) Efek pencerahan melalui C = Img + 30 pada komponen R, G, dan B dan kontras sebesar 2 kali hanya pada komponen R

**Gambar 9.10 Pencerahan dan peningkatan kontras pada citra berwarna**

## 9.5 Menghitung Jumlah Warna

Berapa jumlah warna yang menyusun suatu citra? Bila terdapat kebutuhan seperti itu, jumlah warna dapat dihitung dengan memanfaatkan fungsi jumwarna berikut.

**Program : jumwarna.m**

```
function [jumlah] = jumwarna(berkas)
% JUMWARNA Menghitung jumlah warna pada citra RGB
%     Masukan:
%        berkas berupa citra warna
%     Nilai balik berupa jumlah warna

RGB = double(imread(berkas));
```

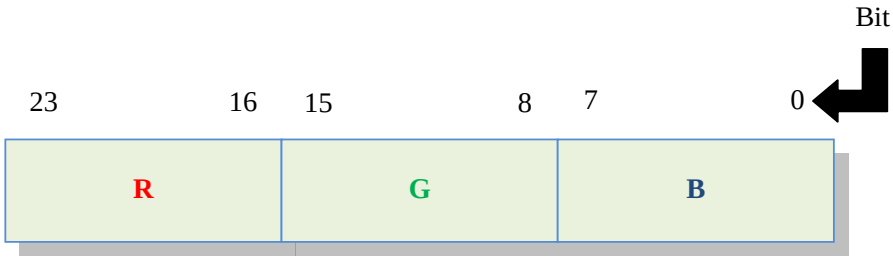
```
[m,n,d] = size(RGB);

if (d ~= 3)
    disp('Citra harus berupa citra berwarna');
    return;
end

RGB = double(RGB);
Data = zeros(1, m * n); % Array kosong

jum = 0;
for i=1:m
    for j=1:n
        jum = jum + 1;

        r = RGB(i,j,1);
        g = RGB(i,j,2);
        b = RGB(i,j,3);

        Data(jum) = bitshift(r,16) + bitshift(g, 8) + b;
    end
end

% Urutkan data pada array Data
Data = sort(Data);

% Hitung jumlah warna
jwarna = 1;
for i = 1 : jum - 1
    if Data(i) ~= Data(i+1)
        jwarna = jwarna + 1;
    end
end

jumlah = jwarna;
```

## Akhir Program

Penghitungan warna dilakukan dengan mula-mula menyusun komponen R, G, dan B untuk setiap piksel menjadi sebuah nilai dengan komposisi seperti terlihat pada Gambar 9.11. Untuk keperluan seperti itu, maka:

- G perlu digeser ke kiri sebanyak 8 bit dan
- R perlu digeser ke kiri sebanyak 16 bit.

Pada *Octave* dan *MATLAB*, penggeseran bit dilakukan melalui fungsi **bitshift**.




**Gambar 9.11 Komposisi R, G, dan B dalam sebuah nilai**

Setelah nilai gabungan R, G, dan B terbentuk dan diletakkan ke larik `Data`, isi larik tersebut diurutkan. Pengurutan tersebut dimaksudkan untuk mempermudah penghitungan jumlah warna. Implementasi penghitungan pada data yang telah urut seperti berikut:

```
jwarna = 1;
for i = 1 : jum - 1
    if Data(i) ~= Data(i+1)
        jwarna = jwarna + 1;
    end
end
```

Berdasarkan kode di atas, nilai `jwarna` dinaikkan sebesar satu sekiranya suatu nilai dan nilai berikutnya tidak sama.

Contoh penggunaan fungsi `jumwarna`:

```
>> C = jumwarna('C:\Image\lapangan.png') ↵
C = 92475
>>
```

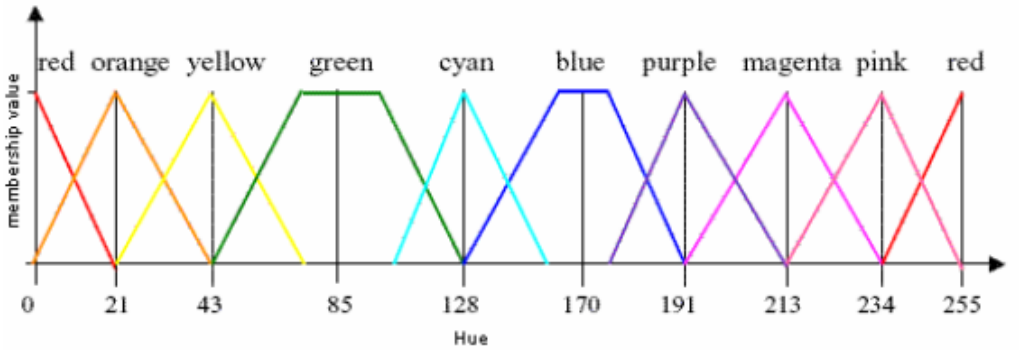
Hasil di atas menyatakan bahwa jumlah warna yang terkandung pada lapangan.png adalah 92.475.

## 9.6 Aplikasi Pencarian Citra Berdasarkan Warna Dominan

Contoh aplikasi penentuan warna dominan pada daun dibahas oleh Kadir (2011d). Dengan mengacu tulisan dalam artikel tersebut, contoh aplikasi pada pencarian citra berdasarkan warna dominan akan dijelaskan pada subbab ini. Model yang sederhana akan diimplementasikan dalam bentuk program.

Agar pencarian menurut warna dominan seperti merah atau hijau dapat dilakukan, setiap piksel yang menyusun citra harus dapat dipetakan ke dalam warna alamiah semacam merah atau hijau. Hal ini dapat dilakukan kalau ruang warna yang digunakan berupa HSV. Pada sistem HSV, komponen *hue* sebenarnya

menyatakan warna seperti yang biasa dipahami oleh manusia. Younes, dkk. (2007) membuat model *fuzzy* untuk menyatakan warna seperti terlihat pada Gambar 9.12.

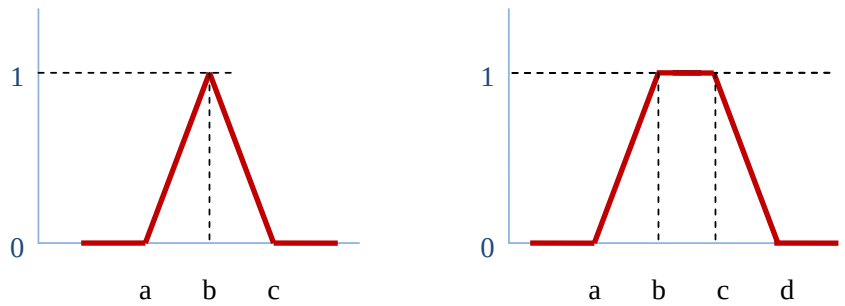



**Gambar 9.12 Dimensi H**
**(Sumber: Younes, dkk., 2007)**

Berdasarkan Gambar 9.12, dimungkinkan untuk menerapkan *fuzzy logic* untuk menentukan suatu area warna beserta derajat keanggotaannya. Model tersebut didasarkan pada sumbu melingkar pada komponen *Hue* (H). Mengingat warna merah berada pada nilai H sama dengan nol, maka jangkauan warna merah berada di sekitar angka 0 dan 255.

Dua jenis fungsi keanggotaan *fuzzy* yang digunakan pada Gambar 9.12 berbentuk segitiga dan trapesium dan digambarkan kembali pada Gambar 9.13. Berdasarkan Gambar 9.13 tersebut, fungsi keangotaan berbentuk segitiga didefinisikan sebagai berikut:

$$\Lambda(x;a,b,c)=\begin{cases}0 & x\le a,\\ (x-a)/(b-a) & a<x\le b,\\ (c-x)/(c-b) & b<x\le c,\\ 0 & x>c\end{cases} \tag{9.51}$$



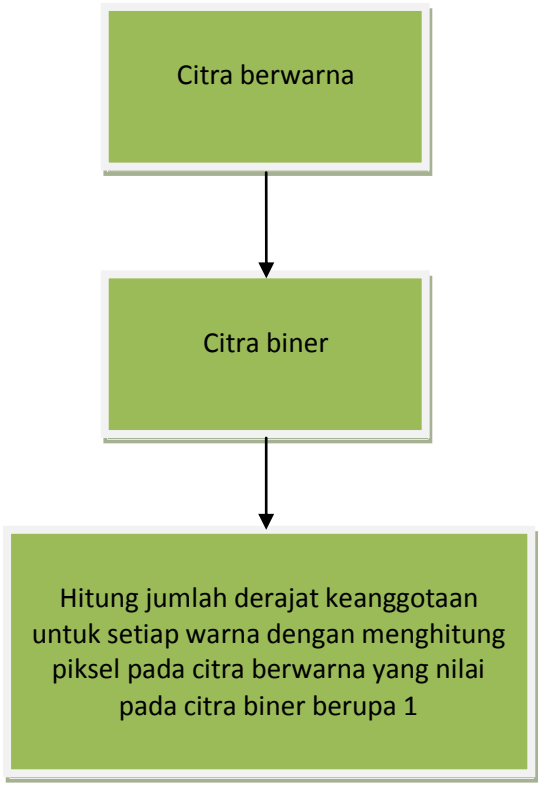

(a) Bentuk segitiga (b) Bentuk trapesium

**Gambar 9.13 Fungsi keanggotan berbentuk segitiga dan trapesium**

Adapun fungsi keanggotaan berbentuk trapesium didefnisikan sebagai berikut:

$$\Pi(x;a,b,c)=\begin{cases}0 & x\le a,\\ (x-a)/(b-a) & a<x\le b,\\ 1 & b<x\le c,\\ (c-x)/(c-b) & c<x\le d,\\ 0 & x>d\end{cases} \quad (9.52)$$

Khusus untuk warna hitam, putih, dan abu-abu dilakukan penanganan secara khusus, yaitu dengan memperhatikan komponen S dan V. Sebagaimana tersirat pada Gambar 9.4, warna hitam diperoleh manakala V bernilai 0 dan warna putih diperoleh ketika S bernilai 0. Warna abu-abu diperoleh ketika S bernilai rendah. Semakin rendah nilai S maka warna abu-abu semakin terlihat tua. Secara garis besar, proses untuk memasukkan setiap piksel ke dalam suatu kategori warna (merah, hijau, dsb.) dilaksanakan melalui mekanisme seperti terlihat pada Gambar 9.14.




**Gambar 9.14 Skema pemrosesan keanggotaan warna**

Citra biner digunakan untuk menentukan area pada citra berwarna yang diproses khusus yang merupakan bagian daun. Dalam hal ini bagian yang berisi daun akan

bernilai 1 pada citra biner dan 0 untuk latarbelakang. Selanjutnya, nilai H, S, V digunakan untuk menentukan kelompok warna piksel. Sebagai contoh, fungsi untuk menentukan warna hijau didefinisikan sebagai berikut:

```
function derajat=f_green(h, s, v)
   if (h==0) && (s==0)
       derajat = 0.0;
   else
      derajat = f_trapesium(43,65,105,128, h);
   end;
```

Dalam hal ini, f_trapesium adalah fungsi untuk mengimplementasikan Gambar 9.13(b). Definisinya seperti berikut:

```
function derajat=f_trapesium(a,b,c,d,h)

if (h>a) && (h<b)
        derajat=(h-a)/(b-a);
    else
        if (h>c) && (h<d)
            derajat=(d-h)/(d-c);
        else
            if (h>=b) && (h<=c)
                derajat=1.0;
            else
                derajat = 0.0;
            end
        end
    end
end
```

Pemrosesan yang dilakukan pada Gambar 9.14 kotak terbawah yaitu menghitung nilai untuk setiap komponen warna:

$$\begin{aligned} hijau &= \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} f\_green(H_{ij},S_{ij},V_{ij}) \\ merah &= \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} f\_red(H_{ij},S_{ij},V_{ij}) \\ kuning &= \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} f\_yellow(H_{ij},S_{ij},V_{ij}) \end{aligned} \quad (9.53)$$

dst.

Setelah seluruh piksel pada daun diproses, setiap warna akan menyimpan komponen warna pada daun. Dengan demikian, warna yang memiliki nilai tertinggi menyatakan warna dominan pada daun.

Selanjutnya, *query* warna dominan dilakukan melalui

$$warna\ domninan = \max(\text{hijau}, \text{merah}, \text{kuning})$$
$$query(warna):\ warna\ domninan = \text{warna}$$

Jadi, untuk semua citra yang warna dominannya sama dengan warna yang diminta dalam *query* akan ditampilkan.

Implementasi fungsi `cariwarna` yang ditujukan untuk mencari warna dominan dapat dilihat berikut ini.

**Program : cariwarna.m**

```
function [Hasil] = cariwarna(warna, lokdir)
% CARIWARNA Digunakan untuk mencari gambar yang berada
%     pada folder lokdir
%     dan memiliki warna dominan sesuai
%     dengan argumen warna
%     Keluaran:
%         Hasil = berisi nama-nama warna yang dicari

if (strcmp(warna,'merah')) || ...
   (strcmp(warna,'biru')) || ...
   (strcmp(warna,'cyan')) || ...
   (strcmp(warna,'hijau')) || ...
   (strcmp(warna,'magenta')) || ...
   (strcmp(warna,'jingga')) || ...
   (strcmp(warna,'merah muda')) || ...
   (strcmp(warna,'ungu')) || ...
   (strcmp(warna,'putih')) || ...
   (strcmp(warna,'hitam')) || ...
   (strcmp(warna,'abu-abu')) || ...
   (strcmp(warna,'kuning'))
  disp('Tunggu...');
else
    disp(['Untuk sementara warna yang bisa dipakai: '...
          'merah, biru, cyan, hijau, magenta, jingga, ' ...
          'merah muda, ungu, ' ...
          'putih hitam abu-abu kuning']);
    return;
end

berkas = dir(lokdir);

jum=0;
indeks=0;
for i=3:length(berkas)
```

```
nama_file = sprintf('%s/%s',lokdir, berkas(i).name);
disp(berkas(i).name);
Img = imread(nama_file);
[tinggi, lebar, dim] = size(Img);

% Mengantisipasi warna hitam, putih, dan abu-abu
%     yang homogen dan selalu dianggap
%     berdimensi satu pada Octave
if dim == 1
   Img(:,:,2) = Img(:,:,1);
   Img(:,:,3) = Img(:,:,1);
end

% Konversi ke HVS
[H,S,V] = RGBkeHSV(Img(:,:,1),Img(:,:,2),Img(:,:,3));

H = double(H);
S = double(S);
V = double(V);

mem_val = 0.0;

anggota_merah = 0.0;
anggota_biru = 0.0;
anggota_cyan = 0.0;
anggota_hijau = 0.0;
anggota_magenta = 0.0;
anggota_oranye = 0.0;
anggota_pink = 0.0;
anggota_ungu = 0.0;
anggota_putih = 0.0;
anggota_hitam = 0.0;
anggota_abu_abu = 0.0;
anggota_kuning = 0.0;

for y=1: tinggi
    for x=1: lebar
        h = H(y,x);
        s = S(y,x);
        v = V(y,x);

        mem_val = f_red(h,s,v);
        if mem_val > 0
            anggota_merah = anggota_merah + mem_val;
        end

        mem_val = f_blue(h,s,v);
        if mem_val > 0
            anggota_biru = anggota_biru + mem_val;
        end

        mem_val = f_cyan(h,s,v);
        if mem_val > 0
            anggota_cyan = anggota_cyan + mem_val;
        end

        mem_val = f_green(h,s,v);
        if mem_val > 0
            anggota_hijau = anggota_hijau + mem_val;
```

```
            end

            mem_val = f_magenta(h,s,v);
            if mem_val > 0
                anggota_magenta = anggota_magenta + mem_val;
            end

            mem_val = f_orange(h,s,v);
            if mem_val > 0
                anggota_oranye = anggota_oranye + mem_val;
            end

            mem_val = f_yellow(h,s,v);
            if mem_val > 0
                anggota_kuning = anggota_kuning + mem_val;
            end

            mem_val = f_pink(h,s,v);
            if mem_val > 0
                anggota_pink = anggota_pink + mem_val;
            end

            mem_val = f_purple(h,s,v);
            if mem_val > 0
                anggota_ungu = anggota_ungu + mem_val;
            end

            mem_val = f_white(h,s,v);
            if mem_val > 0
                anggota_putih = anggota_putih + mem_val;
            end

            mem_val = f_black(h,s,v);
            if mem_val > 0
                anggota_hitam = anggota_hitam + mem_val;
            end

            mem_val = f_gray(h,s,v);
            if mem_val > 0
                anggota_abu_abu = anggota_abu_abu + mem_val;
            end
        end
    end

    maks = max( ...
        [anggota_merah anggota_biru anggota_cyan anggota_hijau ...
         anggota_magenta anggota_oranye anggota_pink ...
         anggota_ungu anggota_putih anggota_abu_abu ...
         anggota_hitam anggota_kuning]);

    % Memperoleh hasil yang memenuhi warna permintaan

    if strcmp(warna,'merah')
        if maks == anggota_merah
            jum = jum +1 ;
            Hasil{jum}.nama = nama_file;
            Hasil{jum}.bobot = anggota_merah;
        end
    elseif strcmp(warna,'biru')
```

```
    if maks == anggota_biru
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_biru;
    end
elseif strcmp(warna,'cyan')
    if maks == anggota_cyan
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_cyan;
    end
elseif strcmp(warna,'hijau')
    if maks == anggota_hijau
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_hijau;
    end
elseif strcmp(warna,'magenta')
    if maks == anggota_magenta
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_magenta;
    end
elseif strcmp(warna,'jingga')
    if maks == anggota_oranye
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_oranye;
    end
elseif strcmp(warna,'pink')
    if maks == anggota_pink
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_pink;
    end
elseif strcmp(warna,'ungu')
    if maks == anggota_ungu
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_ungu;
    end
elseif strcmp(warna,'putih')
    if maks == anggota_putih
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_putih;
    end
elseif strcmp(warna,'hitam')
    if maks == anggota_hitam
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_hitam;
    end
elseif strcmp(warna,'abu-abu')
    if maks == anggota_abu_abu
        jum = jum +1 ;
        Hasil{jum}.nama = nama_file;
        Hasil{jum}.bobot = anggota_abu_abu;
    end
```

```
    elseif strcmp(warna,'yellow')
        if maks == anggota_kuning
            jum = jum + 1 ;
            Hasil{jum}.nama = nama_file;
            Hasil{jum}.bobot = anggota_kuning;
        end
    end
end

% Lakukan pengurutan secara descending
for p = 2: jum
    x = Hasil{p};

    % Sisipkan x ke dalam data[1..p-1]
    q = p - 1;
    ketemu = 0;

    while ((q >= 1) && (~ketemu))
        if (x.bobot > Hasil{q}.bobot)
            Hasil{q+1} = Hasil{q};
            q = q - 1;
        else
            ketemu = 1;
        end

        Hasil{q+1} = x;
    end
end

% Menampilkan maksimum 24 warna

if jum>24
    jum = 24;
end

if jum >= 20
    m=5; n=5;
else
   if jum>=16
      m=5; n=4;
   else
      m=4; n=4;
   end
end

if jum>0
    close;
    figure(1);
    for i=1:jum
       % Tampilkan citra dan nama depan file
       nama = Hasil{i}.nama;
       subplot(m,n,i);
       Citra = imread(nama);
       imshow(Citra);
       [pathstr, name, ext] = fileparts(nama);
       title([name ext]);
    end
end
```

```
return

% ----------------------------------------
% Bagian untuk menghitung keanggotaan fuzzy
% ----------------------------------------

function derajat=f_red(h, s, v)
   if (h==0) && (s==0)
       derajat = 0.0;
   else
       derajat = f_segitiga_kanan(0,21, h) + ...
                 f_segitiga_kiri(234,255, h);
   end

function derajat=f_green(h, s, v)
   if (h==0) && (s==0)
       derajat = 0.0;
   else
      derajat = f_trapesium(43,65,105,128, h);
   end;

function derajat=f_yellow(h, s,v)
   if (h==0) && (s==0)
       derajat = 0.0;
   else
       derajat = f_segitiga(21,43, 65, h);
   end

function derajat=f_blue(h, s,v )
   if (h==0) && (s==0)
       derajat = 0.0;
   else
       derajat = f_trapesium(128,155,180,191, h);
   end;

function derajat=f_purple(h,s,v)
   if (h==0) && (s==0)
       derajat = 0.0;
   else
       derajat = f_segitiga(180,191,213, h);
   end

function derajat=f_cyan(h,s,v)
   if (h==0) && (s==0)
       derajat = 0.0;
   else
       derajat = f_segitiga(105,128,155, h);
   end;

function derajat=f_orange(h,s,v)
   if (h==0) && (s==0)
       derajat = 0.0;
   else
       derajat = f_segitiga(0,21,43, h);
   end;

function derajat=f_magenta(h,s,v)
   if (h==0) && (s==0)
       derajat = 0.0;
```

```
    else
        derajat = f_segitiga(191,213,234, h);
    end

function derajat=f_pink(h,s,v)
    if (h==0) && (s==0)
        derajat = 0.0;
    else
        derajat = f_segitiga(213,234,255, h);
    end;

function derajat=f_white(h, s,v )
    if (s <= 10) && (v>=250)
        derajat = 1.0;
    else
        derajat = 0.0;
    end;

function derajat=f_gray(h, s,v )
    if (h==0) && (s==0) && (v>=15) && (v<250)
        derajat = 1.0;
    else
        derajat = 0.0;
    end;

function derajat=f_black(h, s,v )
    if (h==0) && (s==0) && (v<15)
        derajat = 1.0;
    else
        derajat = 0.0;
    end;

function derajat=f_trapesium(a,b,c,d,h)

     if (h>a) && (h<b)
         derajat = (h-a)/(b-a);
     else
         if (h>c) && (h<d)
             derajat = (d-h)/(d-c);
         else
             if (h>=b) && (h<=c)
                 derajat = 1.0;
             else
                 derajat = 0.0;
             end
         end
     end

function derajat=f_segitiga(a,b,c,h)

if h==b
     derajat = 1.0;
else
     if (h>a) && (h<b)
         derajat = (h-a)/(b-a);
     else
         if (h>b) && (h<c)
             derajat = (c-h)/(c-b);
         else
```

```
            derajat = 0.0;
        end
    end
end

function derajat=f_segitiga_kiri(a,b,h)
if h==b
    derajat=1.0;
else
    if (h>a) && (h<b)
        derajat = (h-a)/(b-a);
    else
        derajat = 0.0;
    end
end

function derajat=f_segitiga_kanan(a,b,h)
if h==a
    derajat=1.0;
else
    if (h>a) && (h<b)
        derajat = (b-h)/(b-a);
    else
        derajat = 0.0;
    end
end
```

**Akhir Program**

**Catatan**

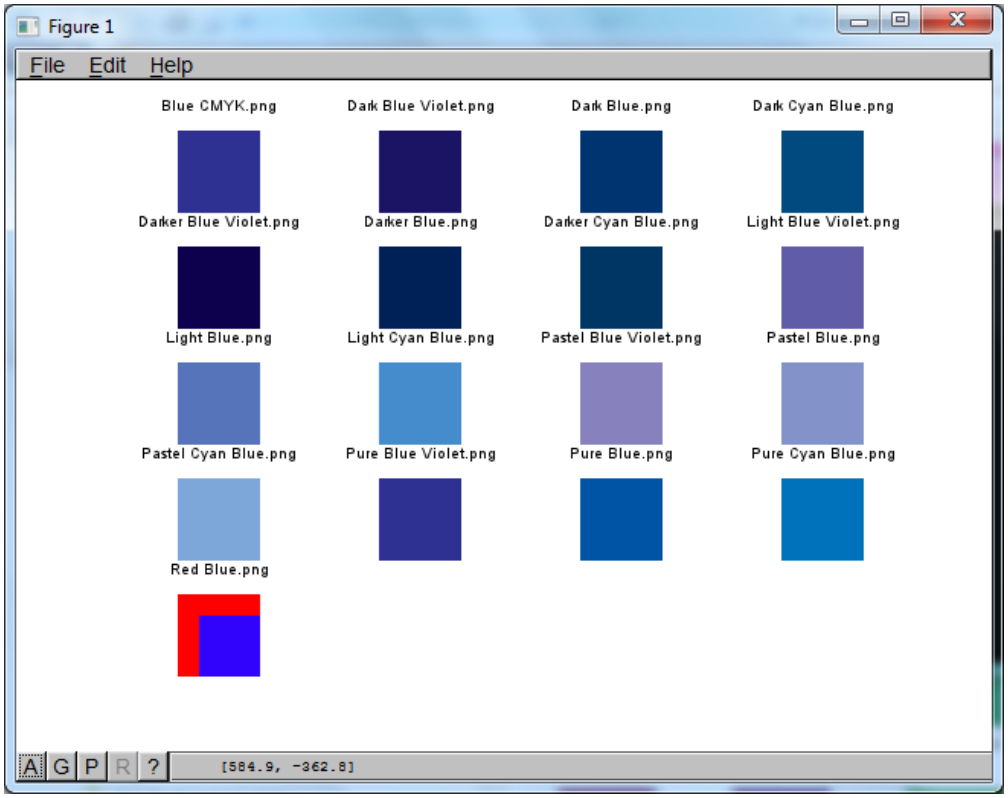
Untuk keperluan mempraktikkan program cariwarna.m, diperlukan untuk menyalin isi subfolder `Warna` yang terdapat pada *file* unduhan (di bawah folder Image) ke dalam folder C:\Image. Dengan demikian, pada C:\Image terdapat subfolder `Warna`.

Contoh penggunaan `cariwarna`:

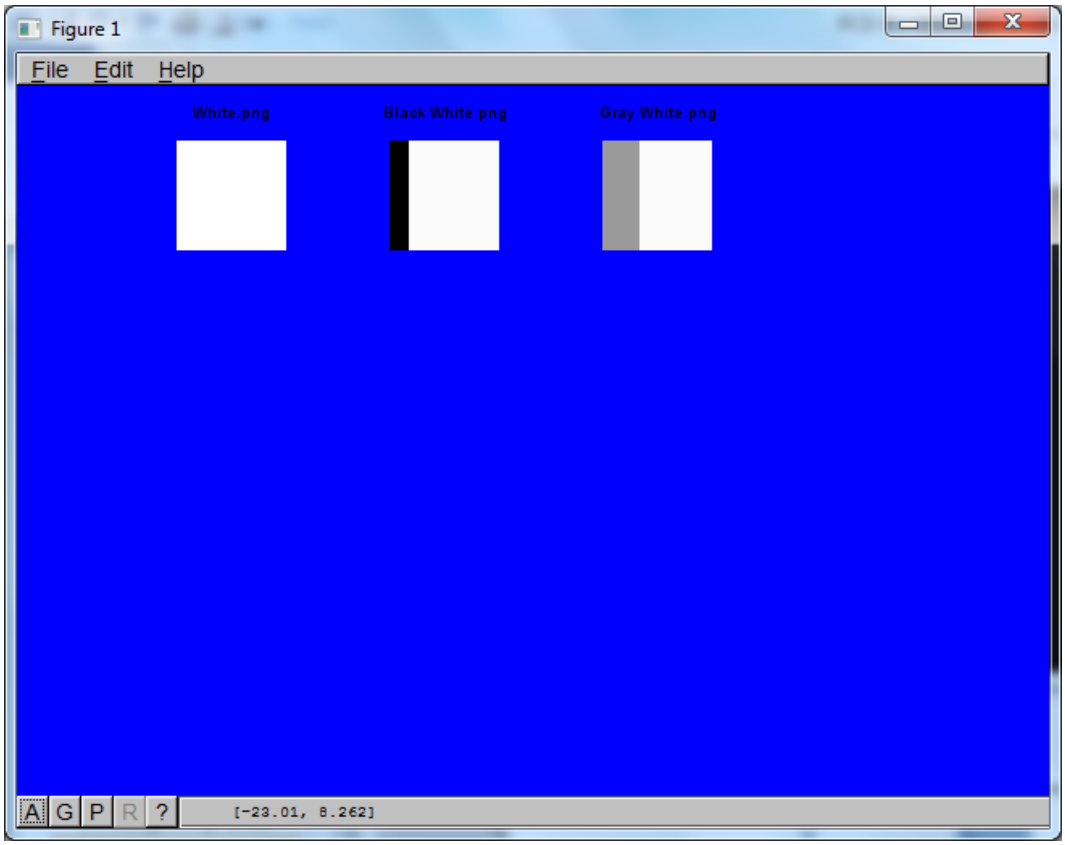
```
>> cariwarna('biru', 'C:\Image\Warna') ↵
```

Perintah di atas digunakan untuk mencari semua gambar yang berada dalam folder C:\Image\Warna yang memiliki warna dominan berupa biru. Hasilnya ditunjukkan pada Gambar 9.15, sedangkan Gambar 9.16 memberikan contoh hasil kalau warna yang dicari adalah putih. Latarbelakang biru pada gambar tersebut diperoleh melalui perintah:

```
set(gcf, 'Color', 'b')
```




**Gambar 9.15 Hasil pencarian warna dominan biru**




**Gambar 9.16 Hasil pencarian warna dominan putih**

## ✍ Latihan

1. Jelaskan istilah-istilah berikut:
   *(a) hue*
   *(b) saturation*
   *(c) brightness*

2. Apa perbedaan CMY dan CMYK?

3. Kapan ruang warna seperti HSV bermanfaat?

4. Apakah ruang warna HIS, HSV, dan HSL itu sama? Kalau berbeda, di mana perbedaannya?

5. Jelaskan penggunaan statistik *mean*, *standard deviation*, *skewness*, dan *kurtosis* pada warna.

6. Mengapa *fuzzy logic* cocok diterapkan pada komponen *hue* (H)?

7. Jelaskan kode berikut:

```
function derajat=f_trapesium(a,b,c,d,h)
  if (h>a) && (h<b)
    derajat = (h-a)/(b-a);
  else
    if (h>c) && (h<d)
      derajat = (d-h)/(d-c);
    else
      if (h>=b) && (h<=c)
        derajat = 1.0;
      else
        derajat = 0.0;
      end
    end
  end
```

8. Jelaskan pula kode berikut:

```
function derajat=f_green(h, s, v)
```

```
if (h==0) && (s==0)
   derajat = 0.0;
else
  derajat = f_trapesium(43,65,105,128,  h);
end;
```

9. Kelemahan dari cariwarna.m terletak pada langkah yang selalu menghitung komposisi warna untuk setiap *file* citra setiap kali terdapat permintaan suatu warna. Langkah yang lebih baik adalah melakukan perhitungan komposisi warna sekali saja dan kemudian hasilnya diletakkan dalam suatu *file*. Dalam hal ini, perintah **save** bisa dipakai. Selanjutnya, *query* terhadap warna dapat dlakukan secara langsung melalui *file* tersebut. Cobalah untuk mengimplementasikannya.

10. Ada kemungkinan warna yang mendominasi suatu citra lebih dari satu warna. Sebagai contoh, sebuah citra berisi warna merah dan putih dengan komposisi yang sama. Kembangkan program pencari warna dominan yang bisa mengantisipasi hal seperti itu.