BAB 10

PENGOLAHANWARNA

Bab ini membahas cara melakukan pengolahan yang didasarkan pada warna. Materi yang dikupas mencakup:

- dasar warna;
- dasar ruang warna;
- ruang warna RGB;
- ruang warna CMV/CMYK;
- ruang warna YIQ;
- ruang warna YCrCb;
- ruang warna HSL dan HLS;
- ruang warna CIELAB;
- ruang warna CIELUV;
- pemilihan warna pada citra;
- deteksi lingkaran berwarna merah;
- segmentasi Nemo;
- penerapan logika samar untuk pencarian warna dominan.

10.1 Dasar Warna

Manusia sebenarnya melihat warna adalah karena cahaya yang dipantulkan oleh objek. Dalam hal ini, spektrum cahaya berkisar antara 400-700 nm.

Tiga faktor yang menentukan warna adalah *hue,* saturasi, dan kecerahan. Berikut penjelasan masing-masing.

- *Hue* merujuk ke warna yang dikenal manusia, misal merah, hijau, dan biru. Warna yang ditangkap oleh mata manusia bergantung pada panjang gelombang cahaya. Sebagai contoh, warna biru akan ditangkap oleh mata sekiranya cahaya yang diterima memiliki panjang gelombang antara 430 dan 480 nanometer.
- Saturasi 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 perkataan lain, suatu warna murni yang bercampur dengan cahaya putih memiliki saturasi antara O dan 100%.
- 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.

10.2 Dasar Ruang Warna

Ruang warna atau sistem warna adalah model yang digunakan untuk menyatakan warna setiap piksel pada citra. Secara bawaan, OpenCV menggunakan ruang warna BGR. Adapun ruang warna yang lain antara lain berupa RGB, HSI, HSV, CMV, LUV, dan YIQ.

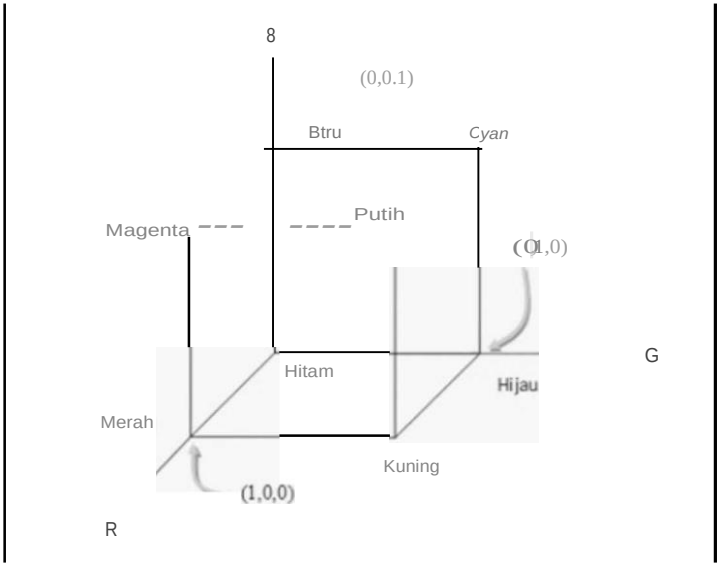
Kadangkala, diperlukan untuk menggunakan ruang warna selain GBR untuk melakukan pengolahan citra. Hal ini disebabkan, pada ruang GBR, informasi warna dan intensitas cahaya tercampur. Sebagai contoh, dengan menggunakan ruang warna HIS ataupun HSV, pencarian citra yang mengandung warna tertentu bisa dilakukan karena informasi warna terdapat pada satu kanal saja, yaitu H.

Hal yang menarik, OpenCV menyediakan sejumlah metode yang dapat digunakan untuk melakukan konversi dari ruang warna BGR ke ruang warna lain. Walaupun demikian, teori yang mendasari konversi antarruang warna dibahas di bab ini.

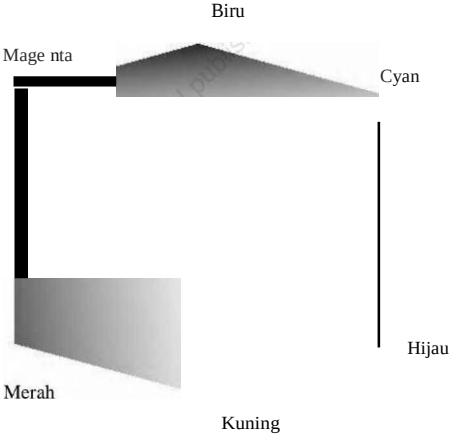
10.3 Ruang Warna RGB

Ruang warna RGB biasa diterapkan pada monitor CRT dan kebanyakan sistem grafika komputer. Tiga komponen dasar yang mendasari ruang warna ini tersirat dalam namanya, yaitu merah (R), hijau (G), dan biru (B). Ketiga komponen itulah yang digunakan untuk mewarnai setiap piksel dalam citra. Informasi warna dan intensitas cahaya tercampur.

Model RGB biasa disajikan dalam bentuk kubus tiga dimensi, dengan warna merah, hijau, dan biru berada pada pojok sumbu (Gambar 10.1). Warna hitam berada pada titik asal dan warna putih berada di ujung kubus yang berseberangan. Gambar 10.2 memperlihatkan kubus warna secara nyata dengan resolusi 24 bit. Dengan menggunakan 24 bit, jumlah warna mencapai 16.777.216.



Gambar 10.1 Skema ruang warna RGB dalam bentuk kubu



*Gambar 10. **Ku**bus warna yang menggunakan 4 bit*

Ruang warna BGR yang digunakan pada OpenCV sebenarnya adalah varian RGB. Perbedaan hanya terletak pada susunan ketiga komponen warna. Konversi dari BGR ke RGB dilakukan dengan cara seperti berikut:

```
rgb = bgr[...,::-1]
```

Adapun konversi dari RGB ke BGR dilaksanakan dengan menggunakan:

```
bgr = rgb[...,::-1]
```

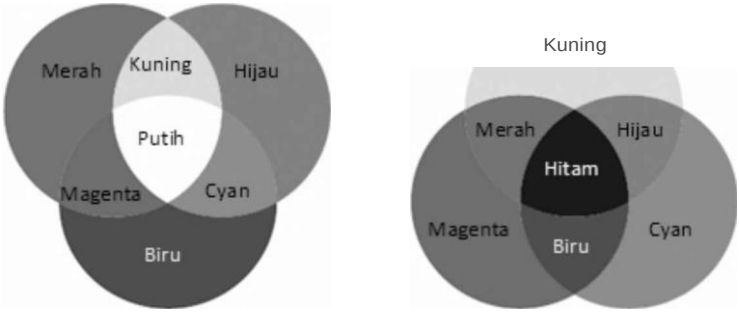
10.4 Ruang Warna CMV/CMYK

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

$$[\ ]\{\}-[\ ]$$

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

Pada CMV, 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.



(a) Penjumlahan wama pada sistem RGB (b) Pengurangan warna pada sistem CMY

Gambar 10.S Komposisi warna pada CMY dan CMYK

Rumus yang digunakan untuk melakukan konversi CMV ke CMYK bermacam-macam. Salah satu caranya adalah seperti berikut:

K = min(C, M, Y)

C'=C-K

M' = M - K

Y' = Y- K

Dengan pendekatan seperti itu, salah satu dari C', M', atau Y' akan bernilai 0.

Berikut adalah contoh skrip yang digunakan untuk mengonversi nilai RGB ke CMYK dan sebaliknya.

Berkas : cmyk.py

```
# Konversi RGB ke CMYK dan sebaliknya

import cv2

SKALA_CMYK = 255

def rgbKeCmyk(r, g, b):
    if (r == 0) and (g == 0) and (b    0):
        # warna hitam
        return 0, 0, 0, SKALA_CMYK

    # RGB [0,255] -> CMY [0,1]
    c = 1   r / 255.0
    m = 1   g / 255.0
    y   1   b / 255.0

    # Peroleh C, M, Y, K
    k   min(c, m, y)
    c = (c    k)
    m = (m    k)
    y    (y    k)

    # Penyekalaan ke jangkauan [0, SKALA_CMYK]
    return int(c * SKALA_CMYK + .5), \
           int(m * SKALA_CMYK + .5), \
```

```
            int(y * SKALA_CMYK + .5), \
            int(k * SKALA_CMYK + .5)

def cmykKeRgb(c, m, y, k):
    r = 255 * (1.0 - (c + k) / float(SKALA_CMYK))
    g = 255 * (1.0 - (m + k) / float(SKALA_CMYK))
    b = 255 * (1.0 - (y + k) / float(SKALA_CMYK))
    return int(r + .5)' int(g + .5)' int(b + .5)

# ---------------------------------
# Program utama

c, m, y, k = rgbKeCmyk(172, 215, 170)
print("C   ", c)
print("M   ", m)
print("Y   ", y)
print("K   ", k)

r, g, b = cmykKeRgb(c,  m, y, k)
print("R   ", r)
print("G      g)
t("B      b)
```

Akhir berkas

Pada skrip ini, fungsi rgbKeCmyk () **digunakan untuk mengonversi nilai R, G, dan B ke C, M, Y, dan K. Adapun fungsi** cmykKeRgb () **digunakan untuk melakukan konversi dari C, M, Y, dan K ke R, G, dan B.**

Hasil skrip ditunjukkan berikut ini:

```
C:\LatOpenCV>python cmyk.py
C     42
M     0
Y     45
K     40
R     172
G     215
B     170

C:\LatOpenCV>
```

10.5 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 luminansi atau intensitas cahaya) dan I serta Q disebut *chroma* (yang berisi informasi warna). Komponen I digunakan untuk menyimpan informasi yang menyatakan perubahan warna oranye ke biru dan komponen Q dipakai untuk menyimpan informasi perubahan warna dari ungu ke hijau.

Konversi YIQ berdasarkan RGB sebagai berikut:

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

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:

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

Skrip berikut menunjukkan contoh untuk menangani konversi dari RGB

Berkas : yiq.py

```
# Konversi RGB ke YIQ dan sebaliknya

import cv2
import numpy as np

SKALA_CMYK = 255
MAT_YIQ = np.array([
    [0.299, 0.587,  0.114],
    [0.596, -0.274, -0.322],
    [0.211, -0.523, 0.312]
])

MAT_RGB = np.array([
    [1, 0.956, 0.621],
    [1, -0.272, -0.647],
    [1, -1.106, 1.703]
])

def rgbKeYiq(r, g, b):
    rgb   np.array([[r], [gl, [bl] )

    yiq   MAT_YIQ.dot(rgb)
    return int(yiq[O, OJ +  .5), \
           int(yiq[l, OJ+   .5), \
           int(yiq[2, OJ +  .5)

def yiqKeRgb(y, i, q):
    yiq   np.array([[y], [i], [q]])
    rgb = MAT_RGB.dot(yiq)

    return int(rgb[O, OJ +  .5), \
           int(rgb[l, OJ+   .5), \
           int(rgb[2, OJ +  .5)

# ---------------------------------
# Program  utama

y, i, q = rgbKeYiq(171, 20, 250)
print("Y      y)
print("!      i)
print("Q      q)
```

```
r,   g, b = yiqKeRgb(y, i,        q)
print("R  =  ",   r)
print("G  =  ",   g)
t("B  =  ",   b)

Akhir berkas
```

Skrip ini melibatkan modul numpy **untuk kepentingan melakukan operasi perkalian matriks. Pernyataan berikut pada definisi fungsi** rgbKeYiq () **digunakan untuk membentuk larik berdimensi dua bernama** matYIQ:

```
MAT_YIQ = np.array([
    [0.299,  0.587,   0.114],
    [0.596,  -0.274,  -0.322],
    [0.211,  -0.523,  0.312]
])
```

Dalam hal **ini,** ma tYIQ **adalah matriks yang berisi koefisien-koefisien untuk melakukan konversi dari RGB ke YIQ. Pernyataan**

```
rgb  = np.array([[r], [g], [bl])
```

digunakan untuk membentuk larik seperti berikut:

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

Adapun pernyataan

```
yiq  =  MAT_YIQ.dot(rgb)
```

digunakan untuk melakukan operasi berikut:

$$yiq = \begin{bmatrix} 0.299 & 0.587 & 0,114 \\ 0,596 & -0,274 & -0,322 \\ 0,211 & -0,523 & 0,312 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

Pernyataan berikut memberikan nilai balik berupa komponen Y, I, dan Q:

```
return int(yiq[O, OJ +  .5), \
       int(yiq[l, OJ + .5), \
       int(yiq[2,  OJ + .5)
```

Penambahan 0,5 dimaksudkan agar pembulatan dilakukan ke atas.

Berikut adalah hasil pengujian skrip `yiq.py`:

```
C:\LatOpenCV>python yiq.py<P
Y     91
I     16
Q     104
R     171
G     19
B     250

C:\LatOpenCV>
```

10.6 Ruang Warna YCrCb

Ruang warna YCrCb biasa digunakan pada video digital. Komponen Y menyatakan luminansi (atau disebut luma) yang dihitung berdasarkan komponen R, G, dan B, yang menyatakan intensitas cahaya. Adapun Cb dan Cr berisi informasi warna. Dalam hal ini, Cr menyatakan seberapa jauh warna merah terhadap luminansi dan Cb menyatakan seberapa jauh warna biru terhadap luminansi.

Proses konversi dari RGB dilakukan dengan beberapa cara. Perhitungan yang digunakan pada OpenCV adalah seperti berikut:

$Y = 0{,}299R + 0{,}587G + 0{,}114B$

$C, = (R-Y)x0{,}713 + delta$

$Cb = (B-Y)x0{,}564 + delta$

$R = Y + 1{,}403x(C, -delta)$

$G = Y-0{,}714x(C, -delta)-0{,}344x(Cb -delta)$

$B = Y + 1.773x(Cb -delta)$

Dalam hal ini,

$$delta = \begin{cases} 128 & untuk\ citra\ 8\text{-}bit \\ 32768 & untuk\ citra\ 16\text{-}bit \\ 0{,}5 & untuk\ citra\ titik\text{-}mengambang \end{cases}$$

Untuk melakukan konversi, konstanta-konstanta berikut bisa digunakan pada pemanggilan fungsi cv2. cvtColor () :

- cv2. COLOR BGR2YCrCb: untuk mengonversi dari BGR ke YCrCb;
- cv2. COLOR RGB2YCrCb: untuk mengonversi dari RGB ke YCrCb;
- cv2 • COLOR_ YCrCb2BGR: untuk mengonversi dari YCrCb ke BGR;
- cv2 • COLOR_ YCrCb2RGB: untuk mengonversi dari YCrCb ke RGB.

Skrip berikut memberikan gambaran tentang cara mengonversi citra R ke YCrCb:

Berkas : **yi'l,ICY**

```
# Contoh konversi dari GBR ke YCrCb

import cv2

citra = cv2.imread('baboon.png')
if not citra is None:
    # Konversi ke YCrCb
    yCrCb = cv2.cvtColor(citra, cv2.COLOR_BGR2YCrCb)

    print("Informasi untuk piksel di pojok kiri-atas:")
    print("Y    ", yCrCb[0, 0, 0])
    print("Cr   ", yCrCb[0, 0, l])
    print("Cb = ", yCrCb[0, 0,  2])
```

Akhir berkas

Pada skrip ini, citra yang dibaca adalah citra berwarna `baboon.png`. Jika citra ini berhasil dibaca, pernyataan berikut digunakan untuk mengonversinya ke ruang warna YCrCb:

```
yCrCb = cv2.cvtColor(citra, cv2.COLOR_BGR2YCrCb)
```

Hasilnya berupa larik `yCrCb` yang berdimensi tiga.

Selanjutnya, pengaksesan data komponen Y, Cr, dan Cb untuk piksel yang terletak di pojok kiri-atas (koordinat 0, O) dilakukan melalui:

- `yCrCb [ 0, 0, 0]` untuk komponen Y;
- `yCrCb [ 0, 0, 1]` untuk komponen Cr;
- `yCrCb [ 0, 0, 2]` untuk komponen Cb,

Hasil skrip dapat dilihat berikut ini:

```
C:\LatOpenCV>python ycrcb.py
Informasi untuk piksel terkiri-atas:
Y     145
Cr    142
Cb    86

C:\LatOpenCV>
```

10.7 Ruang Warna 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", dan V berasal dari "value".

Ruang warna HSV biasa disebut HSB. Modelnya diperlihatkan pada Gambar 10.4. Dalam hal ini,

- H menyatakan komponen warna;

- S menyatakan tingkat kemurnian warna;
- L menyatakan intensitas cahaya.

Gambar 10.4 Silinder warna pada ruang warna HSV (Sumber: Wikipedia)

Terdapat beberapa cara yang bisa digunakan untuk mendapatkan nilai H, S, V berdasarkan R, G, dan B. Cara yang digunakan pada OpenCV adalah seperti berikut:

$$r = \frac{R}{R+G+B}, g = \frac{G}{R+G+B}, b = \frac{B}{R+G+B}$$

$$V = max(r,g,b)$$

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

$$H = \begin{cases} 0, & jika \quad S=0 \\ \frac{60x(g-b)}{SxV}, & jika \quad V=r \\ 60x[2+\frac{b-r}{SxV}], & jika \quad V=g \\ 60x[4+\frac{r-g}{SxV}] & jika \quad V=b \end{cases}$$

$$H = H + 360 \quad jika \quad H < 0$$

Rumus ini terdapat pada Acharya & Ray (2005).

Untuk melakukan konversi, konstanta-konstanta berikut bisa digunakan pada pemanggilan fungsi cv2.cvtColor() :

- cv2::COLOR BGR2HSV: untuk mengonversi dari BGR ke HSV;
- cv2:COLOR_RGB2HSV: untuk mengonversi dari RGB ke HSV;
- cv2 : COLOR_HSV2BGR: untuk mengonversi dari HSV ke BGR;
- cv2:COLOR_HSV2RGB: untuk mengonversi dari HSV ke RGB.

Hasilnya konversi ke HLS, H berada antara 0 dan 179. Adapun S dan V berada antara 0 dan 255.

Skrip berikut memberikan gambaran tentang cara mengonversi citra R ke HSV:

Berkas : **yi'I-ICY**

```
# Contoh konversi dari GBR ke HSV

import cv2

citra = cv2.imread('baboon.png')
if not citra is None:
    # Konversi ke HSV
    hsv = cv2.cvtColor(citra, cv2.COLOR_BGR2HSV)

    print("Informasi untuk piksel di pojok kiri-atas:")
    print("H   ", hsv[0, 0, 0])
    print("S   ", hsv[0, 0, l])
    print("V =",hsv[0, 0, 2])
```

Akhir berkas

Pada skrip ini, citra yang dibaca adalah citra berwarna baboon.png. Jika citra ini berhasil dibaca, pernyataan berikut digunakan untuk mengonversinya ke ruang warna HSV:

```
hsv = cv2.cvtColor(citra, cv2.COLOR_BGR2HSV)
```

Hasilnya berupa larik `hsv` yang berdimensi tiga.

Selanjutnya, pengaksesan data komponen H, S, dan V untuk piksel yang terletak di pojok kiri-atas (koordinat 0, O) dilakukan melalui:

- `hsv[0, 0, 0]` untuk komponen H;
- `hsv[0, 0, 1]` untuk komponen S;
- `hsv[0, 0, 2]` untuk komponen V.

Hasil skrip dapat dilihat berikut ini:

```
C:\LatOpenCV>python hsv.py
Informasi untuk piksel di pojok kiri-atas:
H     25
S     145
V     164

C:\LatOpenCV>
```

Model ruang HLS (atau kadang disebut HSL) diperlihatkan pada Gambar 10.5. Besaran kecerahan (dinamakan *lightness)* disimbolkan dengan L. Perhitungan komponen H, L, dan S berdasarkan komponen R, G, dan B adalah seperti berikut (Agoston, 2005):

$Vmm = \mathbf{min(R,G,B)}$

$Vmax = \mathbf{max(R,G,B)}$

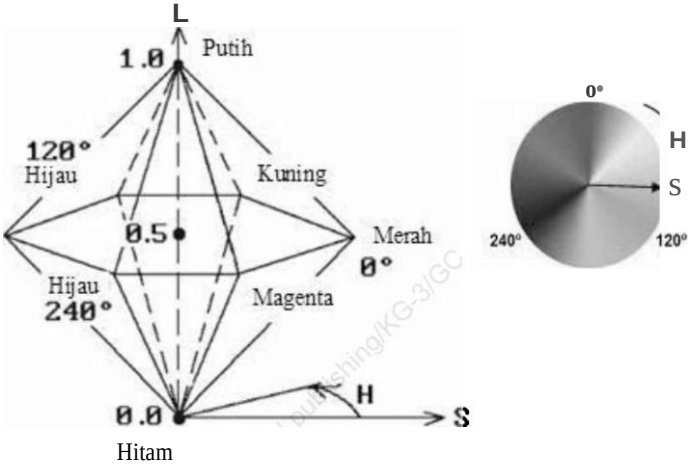
$$L = \frac{V_{\min} + V_{\max}}{2}$$

$$S = \begin{cases} \dfrac{V_{\max} - V_{\min}}{V_{\max} - V_{\min}} & jika \quad L < 0{,}5 \\ \dfrac{V_{\max} - V_{\min}}{2 - (V_{\max} - V_{\min})} & jika \quad L ?: 0{,}5 \end{cases}$$

$$H = \begin{cases} 60(G - B)/(Vmax - Vmin), & jika \quad Vmax = R \\ 120(B=R)/(Vmax = Vmin), & jika \quad Vmax = G \\ 120(R \quad G)/(Vmax \quad Vmin), & jika \quad Vmax = B \end{cases}$$

$$H = H + 360 jika \qquad H < 0$$

Hasilnya L dan S berada antara O dan 1, sedangkan H antara O dan 360.



Gambar 10.5 Model HLS

Untuk melakukan konversi, konstanta-konstanta berikut bisa digunakan pada pemanggilan fungsi cv2.cvtColor ():

- cv2: : COLOR BGR2HLS: untuk mengonversi dari BGR ke HLS;
- cv2: COLOR_RGB2HLS: untuk mengonversi dari RGB ke HLS;
- cv2: COLOR_HLS2BGR: untuk mengonversi dari HLS ke BGR;
- cv2: COLOR_HLS2RGB: untuk mengonversi dari HLS ke RGB.

Hasilnya konversi ke HLS, H berada antara O dan 179. Adapun L dan S berada antara O dan 255.

Skrip berikut memberikan gambaran tentang cara mengonversi citra ke HLS:

Berkas : **yi11,'1Y**

```
# Contoh konversi dari GBR ke HLS

import cv2

citra = cv2.imread('baboon.png')
if not citra is None:
    # Konversi ke HLS
    hls = cv2.cvtColor(citra, cv2.COLOR_BGR2HLS)

    print("Informasi untuk piksel di pojok kiri-atas:")
    print("H    ", hls[0, 0, OJ)
    print("S =",      n.LSLU,0,    .LJJ
    print("V =",   hls[0, 0, 2])
```

Akhir berkas

Pada skrip ini, citra yang dibaca adalah citra berwarna baboon. png. Jika citra ini berhasil dibaca, pernyataan berikut digunakan untuk mengonversinya ke ruang warna HLS:

```
    hls = cv2.cvtColor(citra, cv2.COLOR_BGR2HLS)
```

Hasilnya berupa larik hls yang berdimensi tiga.

Selanjutnya, pengaksesan data komponen H, S, dan V untuk piksel yang terletak di pojok kiri-atas (koordinat 0, O) dilakukan melalui:

- hls [0, 0, 0 J untuk komponen H;
- hl s [0, 0, 1 J untuk komponen L;
- hl s [0, 0, 2 J untuk komponen S.

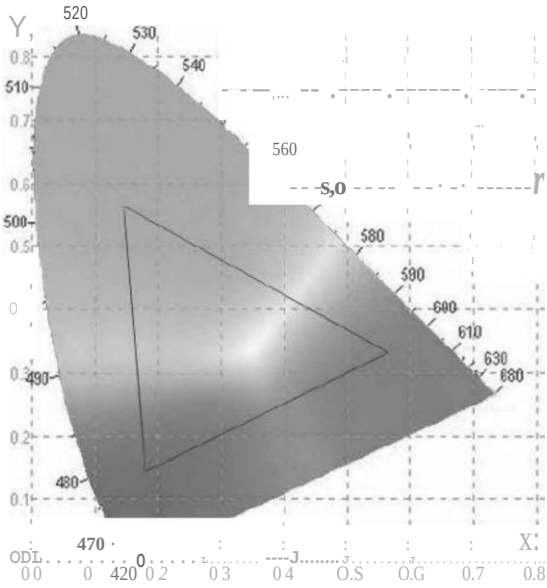
Hasil skrip dapat dilihat berikut ini:

```
C:\LatOpenCV>python hs1.py<P
Informasi untuk piksel di pojok kiri-atas:
H     25
S     118
V     101

C:\LatOpenCV>
```

10.8 Ruang Warna CIELAB

CIELAB adalah nama lain dari CIE L*a*b*. Diagram kromasitas CIE *(Commission Internatiole de L'Eclairage)* ditunjukkan pada Gambar 10.6. Pada diagram tersebut, setiap perpaduan x dan y menyatakan suatu warna. Namun, hanya warna yang berada dalam area ladam (tapal kuda) yang dapat 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.

Pada ruang warna CIELAB, informasi warna terletak pada komponen a dan b, sedangkan intensitas cahaya terekam pada komponen L. Dalam hal ini, komponen a merupakan perubahan dari warna hijau ke magenta dan komponen b merupakan perubahan dari warna biru ke kuning.



Gambar 10.6 Model CIELAB (Sumber: Russ, f!Ol1}

Transformasi RGB ke CIELAB dimulai dengan melakukan perhitungan sebagai berikut:

$$\begin{bmatrix} \\ Z \end{bmatrix} = \begin{bmatrix} \\ 0{,}019334 & 0{,}119193 & 0{,}950227 \end{bmatrix} \begin{bmatrix} \\ B \end{bmatrix}$$

$$X = XI \quad Xn, Xn = 0{,}950456$$

$$Z = Z\ I\ Zn.Zn = 1{,}088754$$

$$L = \begin{cases} 116Y^{112} - 16, & jika \quad Y > 0{,}008856 \\ 903{,}3Y, & jika \quad Y \ 0{,}008856 \end{cases}$$

$$a = 500f(f(X) - f(Y)) + delta$$

$$b = 200f(f(Y) - f(Z)) + delta$$

$$f(t)=\begin{cases} q^{'''} & jika \quad q > 0{,}008856 \\ 7.787q+16/116, & jika \quad q \ 0{,}008856 \end{cases}$$

$$delta=\begin{cases} 128, & untuk \quad citra \quad 8\text{-}bit \\ 0' & untuk \quad citra \quad pecahan \end{cases}$$

Hasilnya, L berada antara O dan 100 dan a serta b berada antara -127 dan 127. Oleh karena itu, agar semua nilai berada antara O dan 255, terdapat perlakukan seperti berikut:

$$L = \text{Lx}255/100,\text{a} = a + 28,\text{b} = b + 128$$

Untuk melakukan konversi, konstanta-konstanta berikut bisa digunakan pada pemanggilan fungsi `cv2.cvtColor()`:

- `cv2::COLOR BGR2Lab`: untuk mengonversi dari BGR ke CIELAB;
- `cv2:COLOR RGB2Lab`: untuk mengonversi dari RGB ke CIELAB;
- `cv2:COLOR Lab2BGR`: untuk mengonversi dari CIELAB ke BGR;
- `cv2:COLOR Lab2RGB`: untuk mengonversi dari CIELAB ke RGB.

Skrip berikut memberikan gambaran tentang cara mengonversi citra R ke CIELAB:


```
# Contoh konversi dari GBR ke CIELAB
import cv2
citra = cv2.imread('baboon.png')
```

```
if not citra is None:
    # Konversi ke CIELAB
    lab= cv2.cvtColor(citra, cv2.COLOR_BGR2Lab)

    print("Informasi untuk piksel di pojok kiri-atas:")
    print("L   ", lab[0, 0, 0])
    print("a   ", lab[0, 0, 1])
    print("b = ", lab[0, 0, 2])

Akhir berkas
```

Pada skrip ini, citra yang dibaca adalah citra berwarna `baboon.png`. Jika citra ini berhasil dibaca, pernyataan berikut digunakan untuk mengonversinya ke ruang warna CIELAB:

```
lab= cv2.cvtColor(citra, cv2.COLOR_BGR2Lab)
```

Hasilnya berupa larik `lab` yang berdimensi tiga.

Selanjutnya, pengaksesan data komponen L, a, dan b untuk piksel yang terletak di pojok kiri-atas (koordinat 0, O) dilakukan melalui:

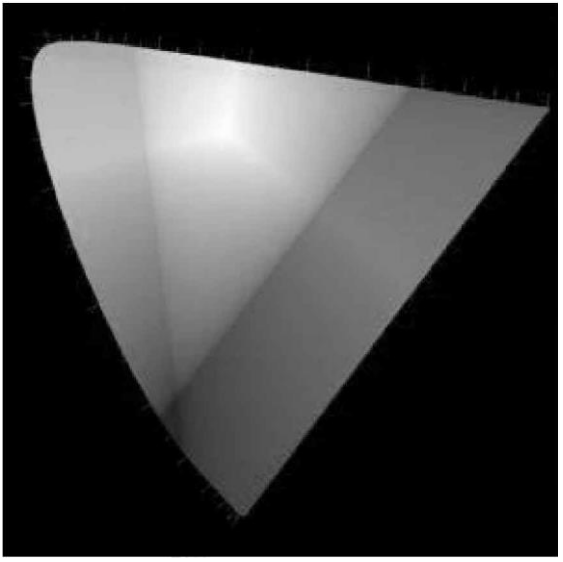
- `lab[0, 0, 0]` untuk komponen L;
- `lab[0, 0, 1]` untuk komponen a;
- `lab[0, 0, 2]` untuk komponen b.

Hasil skrip dapat dilihat berikut ini:

```
C:\LatOpenCV>python cie1ab.py<P
Informasi untuk piksel di pojok kiri-atas:
L    158
a=   123
b    171

C:\LatOpenCV>
```

10.9 Ruang Warna CIELUV

Nama lengkap ruang warna ini adalah CIE 1976 L*, u*, v*, yang diadopsi dari CIE pada tahun 1976. Modelnya ditunjukkan pada Gambar 10.7. Tampak bahwa modelnya menyerupai pada CIELAB.



Gambar 10.7 Model CIELUV (Sumber: Wikipedia)

Konversi dari RGB ke CIELUV dilakukan seperti berikut:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0,212671 & 0,715160 & 0,072169 \\ 0,019334 & 0,119193 & 0,950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$L = \begin{cases} 116Y^{1/3} - 16, & jika \quad Y > 0,008856 \\ 903,3Y, & jika \quad Y \leq 0,008856 \end{cases}$$

$u' = 4X /(X + 15Y + 3Z)$

$v' = 9Y /(X + 15Y + 3Z)$

$u = BL(u'-0,19793943)$

$v = 13L(v'-0,46831096)$

Hasilnya, L berada antara O dan 100, u berada antara -134 dan 220, dan v berada antara -140 dan 122. Oleh karena itu, terdapat langkah untuk membuat L, u, dan v berada antara O dan 255 melalui:

$$L = 255/100L, u = 255/344 + (u + 134), v = 255/262(v+140)$$

Untuk melakukan konversi, konstanta-konstanta berikut bisa digunakan pada pemanggilan fungsi cv2. cvtColor ():

- cv2:COLOR_BGR2Luv: untuk mengonversi dari BGR ke CIELUV;
- cv2:COLOR_RGB2Luv: untuk mengonversi dari BGR ke CIELUV;
- cv2 : COLOR_Luv2BGR: untuk mengonversi dari CIELUV ke BGR;
- cv2 : COLOR_LuvRGB: untuk mengonversi dari CIELUV ke RGB.

Skrip berikut memberikan gambaran tentang cara mengonversi citra dari BGR ke CIELUV:

Berkas : cieluv.py

```
# Contoh konversi dari GBR ke CIELUV

import cv2

citra = cv2.imread('baboon.png')
if not citra is None:
    # Konversi ke CIELUV
```

```
luv = cv2.cvtColor(citra, cv2.COLOR_BGR2Luv)

print("Informasi untuk piksel di pojok kiri-atas:")
print("L   ", luv[O,  0, OJ)
print("a   ", luv[O,  0, l])
print("b  =", luv[O,    0, 2])

Akhir berkas
```

Pada skrip ini, citra yang dibaca adalah citra berwarna `baboon.png`. Jika citra ini berhasil dibaca, pernyataan berikut digunakan untuk mengonversinya ke ruang warna CIELUV:

```
luv = cv2.cvtColor(citra, cv2.COLOR_BGR2Luv)
```

Hasilnya berupa larik 1uv yang berdimensi tiga.

Selanjutnya, pengaksesan data komponen L, u, dan v untuk piksel yang terletak di pojok kiri-atas (koordinat 0, O) dilakukan melalui:

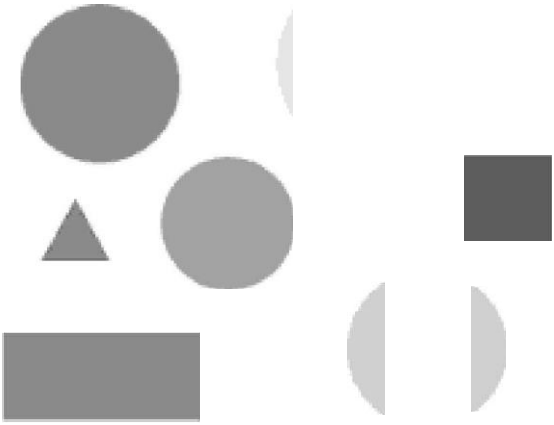
- 1uv [0, 0, 0] untuk komponen L;
- 1 uv [0, 0, 1] untuk komponen u;
- 1uv [0, 0, 2] untuk komponen v.

Hasil skrip dapat dilihat berikut ini:

```
C:\LatOpenCV>python cie1uv.py<P
Informasi untuk piksel di pojok kiri-atas:
L    157
a    106
b    186

C:\LatOpenCV>
```

10.10 Pemilihan Warna pada Citra

Gambar 10.8 menunjukkan enam bentuk yang berupa lingkaran, segitiga, dan kotak dengan aneka warna. Nah, sekarang akan ditentukan untuk menampilkan semua bentuk yang berwarna biru.



Gambar 10.8 Citra dengan beberapa bentuk berwarna (bentuiwara.png)

Persoalan ini dapat diselesaikan dengan melakukan konversi citra ke ruang warna HSV atau HLS.

Bagaimana cara menentukan warna biru pada komponen H? Pertanyaan seperti ini seringkali muncul. Pertama-tama, kita ketahui bahwa pada model BGR, warna biru dinyatakan dengan [255, 0, 0]. Dengan demikian, warna biru dapat didefinisikan seperti berikut:

```
biru = np.uint8([[[255, 0, 0]]])
```

Perintah membuat larik `biru` berdimensi tiga yang berisi satu baris, satu kolom, dan tiga kedalaman. Format yang digunakan adalah BGR. Jadi, nilai B adalah 255, sedangkan G dan R bernilai 0.

Nah, perintah selengkapnya untuk mendapatkan H pada ruang warna HSV dengan menggunakan Python adalah seperti berikut:

```
>>> import cv2<P
>>> import numpy as np<P
>>> biru = np.uint8([[[255, O, 0]]]) <P
>>> hsvBiru = cv2.cvtCo1or(biru,
cv2.COLOR BGR2BSV) <P
>>> print(hsvBiru) <P
[[[120 255 255]]]
>>>
```

Hasil [120 255 255] menyatakan bahwa H=120, 5=255, dan V= 255. Nah berdasarkan nilai H, jangkauan warna yang digunakan di H5V yang digunakan berupa [H-20, 20, 20] dan [H+l5, 255, 255]. Untuk H = 120, maka jangkauan berupa [100, 100, 100] dan [140, 255, 255]. Penggunaan 5=20 dan V=20 pada batas rendah hanya bersifat "coba-coba" dengan mempertimbangkan tingkat kecerahan dan saturasi warna. Nilai 5 yang rendah berkecenderungan putih dan warna 5 yang tinggi berkecenderungan hitam. Nilai V yang rendah mengarah ke gelap dan nilai V yang tinggi mengarah ke cerah (lihat Gambar 10.4) 5ebagai contoh, jika nilai 5 pada batas rendah ditinggikan, warna biru muda tidak terdeteksi.

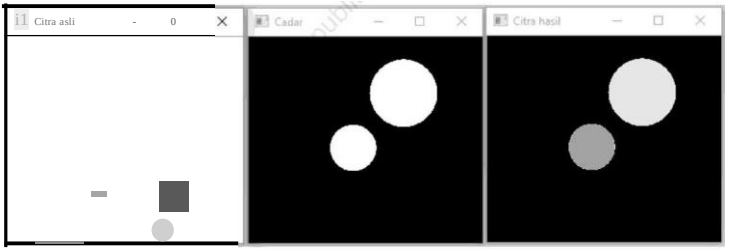
Berikut adalah skrip untuk mendapatkan warna biru dengan menggunakan jangkauan di atas:

11!1 Berkas : **deluv.11y**

```
# Pencarian warna biru

import cv2
import numpy as np

citra = cv2.imread('bentukwarna.png')
if not citra is None:
    # Konversi ke HSV
    hsv = cv2.cvtColor(citra, cv2.COLOR_BGR2HSV)

    batasRendah = np.array([100,20,20])
    batas Atas = np.array([140,255,255])
```

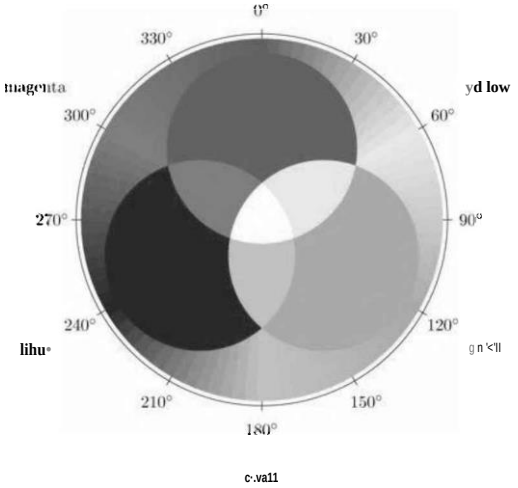
```
# Peroleh bagian yang berwarna biru
cadar = cv2.inRange{hsv, batasRendah, batas Atas)

# Kenakan operasi "dan" terhadap citra asli
hasil = cv2.bitwise_and{citra, citra, mask= cadar)

cv2.imshow{'Citra asli', citra)
cv2.imshow{'Cadar', cadar)
cv2.imshow{'Citra hasil', hasil)

Akhir berkas
y{O)
```

Hasilnya diperlihatkan pada Gambar 10.9. Gambar paling kiri adalah citra yang mengandung berbagai bentuk. Gambar di tengah berupa gambar hitam putih yang mengandung warna yang dicari. Dalam hal ini, warna yang dicari akan dinyatakan dengan warna putih. Citra ini jika dikenakan operasi "dan" terhadap citra asal akan menghasilkan citra yang mengandung warna biru (gambar paling kanan).



Gambar 10.9 Hasil pencarian warna biru

Selain menggunakan cara yang telah diterangkan di depan untuk mendapatkan nilai *hue* untuk suatu warna, Gambar 10.10 dapat dijadikan sebagai pedoman. Nilai yang sesungguhnya perlu dibagi dengan 2. Sebagai contoh, warna *cyan* terletak pada 180°. Maka, nilai *hue* untuk Python berupa 90 (yakni 180/2).



Gambar JO.JO Warna pada komponen hue (Sumber: http://www.texample.net/tikz/examples/rgb-color-mixing/)

Warna merah mempunyai titik tengah *hue* berupa 0. Nah, berdasarkan nilai tersebut, jangkauan warna yang digunakan berupa [H-20, 20, 20] dan [H+20, 255, 255]. Untuk H = 0, maka jangkauan berupa [-20, 100, 100] dan [20, 255, 255].

Karena terdapat nilai yang negatif, jangkauan perlu diubah seperti berikut.

- Jangkauan negatif diganti menjadi:

  [-20, 20, 20] -> [179-20, 20, 20] dan [179,255, 255]

- Jangkauan positif diubah mejadi:

  [20, 255, 255] -> [O, 20, 20] dan [20, 255, 255]

Perlu diketahui, Gambar 10.10 menunjukkan bahwa nilai merah terletak pada dua bagian, yaitu antara sudut 345° dan sudut 15°. Oleh karena itu, warna merah perlu ditangani dengan dua jangkauan.

adalah perwujudannya:

Berkas : cariwarna2.py

```
# Pencarian warna merah

import cv2
import numpy as np

citra = cv2.imread('bentukwarna.png')
if not citra is None:
    # Konversi ke HSV
    hsv = cv2.cvtColor(citra, cv2.COLOR_BGR2HSV)

    batasRendahA = np.array([159, 20, 20])
    batasAtasA = np.array([179, 255, 255])

    batasRendahB = np.array([0, 20, 20])
    batasAtasB = np.array([20, 255, 255])

    # Peroleh bagian yang berwarna merah
    cadarA = cv2.inRange(hsv, batasRendahA, batasAtasA)
    cadarB = cv2.inRange(hsv, batasRendahB, batasAtasB)
    cadar = cadarA | cadarB

    # Kenakan operasi and terhadap citra asli
    hasil = cv2.bitwise_and(citra, citra, mask=cadar)

    cv2.imshow('Citra asli', citra)
    cv2.imshow('Cadar', cadar)
    cv2.imshow('Citra hasil', hasil)

    cv2.waitKey(0)
```

Akhir berkas

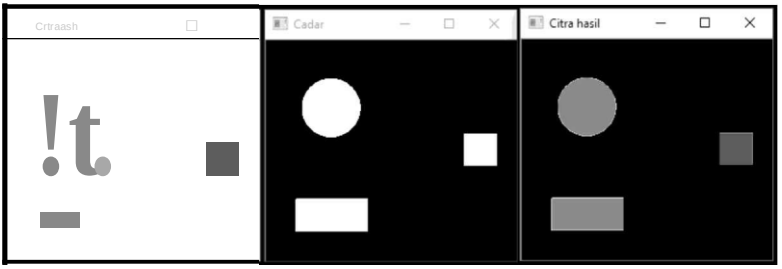
Skrip ini menggunakan tiga cadar. Cadar pertama (cadarA) **digunakan untuk menangani *hue* pada jangkauan [159, 20, 20] dan [179, 255, 255]. Cadar kedua** (cadarB) **digunakan untuk menangani *hue* pada jangkauan [00, 20, 20] dan [20, 255, 255]. Adapun cadar ketiga** (cadar) **adalah hasil gabungan** cadarA **dan** cadarB, **yang dibentuk melalui operasi "atau" berikut:**

```
cadar = cadarA | cadarB
```

Selanjutnya, cadar ini digunakan pada:

```
hasil = cv2.bitwise_and(citra, citra, mask    cadar)
```

Hasilnya diperlihatkan pada Gambar 10.9.



Gambar 10.11 Hasil pencarian warna merah

10.11 Deteksi Lingkaran Berwarna Merah

Aplikasi berikutnya yang diwujudkan di sini berupa cara untuk mendeteksi lingkaran khusus yang berwarna merah. Untuk keperluan ini, transformasi Hough untuk lingkaran bisa digunakan. Di OpenCV, hal ini bisa dilaksanakan dengan menggunakan cv2.HoughCircles (). **Berikut implementasinya:**

11!1 Berkas : ci?lliv.iry

```
# Pencarian lingkaran berwarna merah

import cv2
import numpy as np

citra = cv2.imread('bentukwarna.png')
if not citra is None:
    # Konversi ke HSV
    hsv = cv2.cvtColor(citra, cv2.COLOR_BGR2HSV)

    batasRendahA = np.array([159, 20, 20])
    batasAtasA = np.array([179, 255, 255])

    batasRendahB = np.array([0, 20, 20])
    batasAtasB = np.array([20, 255, 255])
```

```
# Peroleh cadar
cadarA = cv2.inRange(hsv, batasRendahA, batasAtasA)
cadarB = cv2.inRange(hsv, batasRendahB, batasAtasB)
cadar = cadarA | cadarB

# Pengaburan pada cadar
cadar = cv2.GaussianBlur(cadar,(5, 5), 0)

# Deteksi dengan transformasi Hough
dafLingkaran = cv2.HoughCircles(
    cadar, cv2.HOUGH_GRADIENT,
    1, 20, param1 = 50, param2 = 30,
    minRadius = 0, maxRadius = 0)
if dafLingkaran is None:
    print("Tidak ada lingkaran terdeteksi")
    exit()

# Lakukan pembulatan
dafLingkaran = np.uint16(np.around(dafLingkaran))

# Gambar lingkaran
for (x, y, r) in dafLingkaran[0, :]:
    cv2.circle(citra, (x, y), r,
               (255, 255, 0), 2)
    cv2.circle(citra, (x, y), 3,
               (255, 255, 255), -1)

cv2.imshow('Lingkaran merah', citra)

cv2.waitKey(0)
```

Akhir berkas

Skrip ini adalah hasil pengembangan skrip cariwarna2. py. **Pada tahap awal, citra yang digunakan sebagai cadar untuk mendapatkan objek yang berwarna merah didapatkan. Namanya** cadar. **Kemudian, citra ini dikaburkan melalui:**

```
cadar = cv2.GaussianBlur (cadar, (5, 5), 0)
```

Tujuannya adalah untuk mengurangi derau dan menghindari positif palsu *(false positive).* Positif palsu menyatakan keberadaan suatu objek yang sesungguhnya tidak ada. Gambar 10.12 menunjukkan hasil setelah

pengaburan dilakukan. Tampak bahwa terdapat satu lingkaran dan dua kotak.

Gambar 10.n Cadar yang telah dikaburkan

Perintah yang digunakan untuk mendapatkan lingkaran-lingkaran pada citra yang terlihat pada Gambar 10.12 berupa:

```
dafLingkaran = cv2.HoughCircles(
    cadar, cv2.HOUGH_GRADIENT,
    1, 20, paraml = 50, param2   30,
    minRadius = 0, maxRadius = 0)
```

Pernyataan ini membuat dafLingkaran berisi data lingkaran-lingkaran yang diperoleh. Argumen masing-masing adalah seperti berikut.

- Argumen pertama pada HoughCircles () berupa citra yang diproses.
- Argumen kedua berupa metode yang digunakan untuk mencari lingkaran, yaitu metode yang menggunakan informasi gradien pada bagian tepi untuk mendapatkan lingkaran. Saat ini, hanya metode tersebut yang didukung oleh OpenCV.
- Argumen ketiga menentukan resolusi akumulator yang digunakan pada metode untuk memperoleh lingkaran. Nilai 1

menyatakan bahwa akumulator mempunyai resolusi yang sama dengan citra masukan.

- Argumen keempat menentukan jarak terkecil antara pusat dua lingkaran yang terdeteksi.
- Argumen kelima yang melibatkan paraml menyatakan nilai ambang tertinggi yang digunakan pada metode untuk mendapatkan lingkaran.
- Argumen keenam yang melibatkan param2 menyatakan nilai ambang akumulator yang digunakan pada metode untuk mendapatkan lingkaran.
- Argumen minRadius menentukan ukuran jari-jari terkecil untuk lingkaran dan argumen maxRadius menentukan ukuran terbesar untuk lingkaran. Kedua argumen ini diisi dengan nol jika tidak da penentuan jari-jari lingkaran atau cukup dibuang.

Pernyataan berikut digunakan untuk mengantisipasi kalau tidak ada lingkaran yang terdeteksi:

```
if dafLingkaran is None:
    print("Tidak ada lingkaran terdeteksi")
    exit()
```

Pernyataan ini membuat informasi yang menyatakan lingkaran tidak terdeteksi dan eksekusi skrip dihentikan.

Pernyataan

```
dafLingkaran = np.uint16(np.around(dafLingkaran))
```

digunakan untuk membulatkan nilai pada dafLingkaran dan kemudian mengonversi ke tipe bilangan bulat 16 bit tidak bertanda.

Selanjutnya, data pada dafLingkaran diproses melalui:

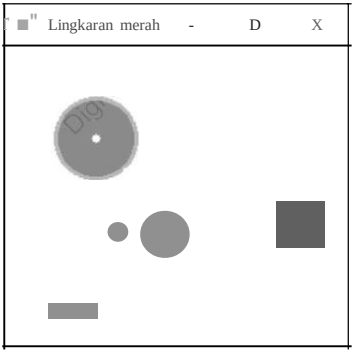
```
for (x, y, r) in dafLingkaran[0, :]:
    cv2.circle(citra, (x, y), r,
               (255, 255, 0), 2)
    cv2.circle(citra, (x, y),  3,
               (255, 255, 255), -1)
```

Pada contoh ini, (x, y) menyatakan pusat lingkaran dan r menyatakan jari-jari lingkaran. Dua lingkaran digambar. Lingkaran pertama digambar pada tepi bagian tepi lingkaran yang didapat. Warna yang digunakan berupa *cyan.* Ketebalan 2 piksel. Lingkaran kedua digambar di tengah titik pusat dengan jari-jari 3 piksel dan lingkaran diarsir dengan warna putih.

Hasil akhirnya digambar melalui:

```
cv2.imshow('Lingkaran merah', citra)
```

Gambar 10.13 menunjukkan hasil perintah ini. Tampak hanya satu lingkaran (terkiri dan teratas) yang terpilih.



Gambar 10.19 Hasil pencarian lingkaran berwarna merah

10.12 Segmentasi Nemo

Contoh berikut memberikan gambaran untuk mendapatkan objek ikan yang dikenal dengan nama Nemo dengan memisahkan terhadap latar belakangnya. Prosesnya biasa disebut segmentasi. Gambar 10.14 menunjukkan tampilan ikan ini.

Gambar I0.14Nemo (Sumber: https://commons.wikimedia.org/wiki/File:Clown_lish_in_the_Andaman_Coral_ReefJpg)

Untuk keperluan latihan, berkasnya bisa diunduh melalui:

https://commons.**wiki**media.org/wiki/File:Clown_fish_in_the_Andama n_Coral_Reef.jpg

Nama berkasnya 800px-Clown_fish_in_the_Andaman_Coral_Reef.

Segmentasi untuk mendapatkan Nemo secara prinsip menggunakan dasar-dasar yang telah dibahas sebelum ini. Tahap pertama adalah mengonversi citra ke dalam ruang warna HSV. Melalui komponen *hue,* pengaturan warna yang mendominasi Nemo bisa diatur.

m ikut adalah skrip untuk melakukan segmentasi Nemo:

Berkas : ci lu·1.py

```
# Segmentasi Nemo

import cv2
import numpy as np

citra = cv2.imread(
    '800px-Clown_fish_in the_Andaman Coral Reef.jpg')
if not citra is None:
    # Konversi ke HSV
    hsv = cv2.cvtColor(citra, cv2.COLOR_BGR2HSV)

    oranyeMuda = np.array([2, 128, 128])
    oranyeGelap = np.array([20, 255, 255])

    putihMuda = np.array([0, 0, 160])
    putihGelap = np.array([179, 255, 255])
```

```
# Peroleh bagian yang berwarna merah
cadarA = cv2.inRange(hsv, oranyeMuda, oranyeGelap)
cadarB = cv2.inRange(hsv, putihMuda, putihGelap)
cadar = cadarA | cadarB

# Kenakan operasi and terhadap citra asli
hasil = cv2.bitwise_and(citra, citra, mask    cadar)

cv2.imshow('Citra asli', citra)

cv2.imshow('Citra hasil', hasil)

cv2.waitKey(O)
```

Akhir berkas

Dua warna yang diproses pada Nemo adalah oranye dan putih. Warna oranye ditentukan oleh:

```
oranyeMuda = np.array([2, 128, 128])
oranyeGelap = np.array([20, 255, 255])
```

Warna putih ditentukan oleh:

```
putihMuda = np.array([O, 0, 160])
putihGelap = np.array([l79, 255, 255])
```

Kedua pasang batas warna inilah yang digunakan untuk membentuk cadar bernama cadar. **Dengan menggunakan**

```
hasil = cv2.bitwise_and(citra, citra, mask= cadar)
```

maka hasil berisi Nemo, sebagaimana diperlihatkan pada Gambar 10.15. Hasilnya memang belum sempurna. Namun, objek-objek kecil di luar Nemo sebenarnya dapat disingkirkan, misalnya menggunakan operasi morfologis atau pemotongan terhadap objek yang lebih kecil dari ukuran tertentu.

Gambar 10.1d Hasil segmentasi yang menghasilkan Nemo

Untuk menghaluskan hasil segmentasi, pengaburan dengan filter Gaussian bisa diterapkan. Sebagai contoh, dua pernyataan berikut bisa ditambahkan sebelum baris yang berisi cv2 . wai tKey (0) ; :

```
hasil = cv2 .GaussianBlur (hasil, (5, 5), 0)
cv2.imshow('Citra dikaburkan', hasil)
```

Hasilnya diperlihatkan pada Gambar 10.16.

Gambar 10.16 Hasil pengaburan terhadap Nemo

10.13 Pelabelan Titik-Titik Terhubung dengan Warna

Kadangkala, diperlukan untuk memberikan label pada setiap objek yang berada pada citra biner. Dalam hal ini, objek pertama yang dijumpai akan diberi label 1. Objek berikutnya diberi label 2, dan seterusnya. Hal seperti ini dapat diperoleh dengan menggunakan cv2. connectedComponents (). **Selanjutnya, jika dikehendaki untuk memvisualkan setiap objek dengan warna yang berbeda, nilai-nilai label objek dapat diganti dengan warna secara otomatis dengan menggunakan nilai *hue.* Perwujudannya seperti berikut:**

Berkas : pelabelan.py

```
# Visualisasi pelabelan dengan warna

import cv2
import numpy as np

def visualisasiWarna(citraBerlabel):
    # Penggunaan label hue
    citraWarnaHue = np.uint8(
        179 * citraBerlabel / np.max(citraBerlabel))
    kanalLain = 255 * np.ones like(citraWarnaHue)
    citraWarna = cv2.merge(
        [citraWarnaHue, kanalLain, kanalLain])

    # Konversi ke format BGR
    citraWarna   cv2.cvtColor(citraWarna,
                              cv2.COLOR_HSV2BGR)

    # Latar belakang dinolkan kembali
    citraWarna[citraWarnaHue == OJ = 0

    cv2.imshow("Citra berlabel", citraWarna)
    cv2.waitKey(0)

# Program utama

citra = cv2.imread("struktur.png", 0)
ambang, citra = cv2.threshold(citra, 127, 255,
                        cv2.THRESH_BINARY)
```

```
jumObjek, citraBerlabel = cv2.connectedComponents(
                                    citra)
[redacted]alisasiWarna(citraBerlabel)

[redacted]Akhir berkas
```

Fungsi visualisasiWarna () **digunakan untuk memvisualkan citra yang telah dilengkapi dengan label dengan warna-warna yang ditentukan melalui *hue.* Warna ditentukan secara otomatis melalui:**

```
citraWarnaHue = np.uint8(
    179 * citraBerlabel / np.max(citraBerlabel))
```

Angka 179 adalah angka terbesar pada komponen *hue.* Dengan cara seperti ini, ci traWarna **berukuran sama dengan** ci traBer label **dan nilainya telah diisi dengan nilai *hue* yang disesuaikan nilai label semula. Citra inilah yang akan mewakili komponen H pada citra dalam format HSV.**

Adapun kanalLain **merupakan komponen citra yang digunakan untuk mewakili komponen S dan V. Perintah**

```
kanalLain = 255 * np.ones_like(citraWarnaHue)
```

digunakan untuk membuat matriks dengan ukuran sama dengan pada ci traWarnaHue **dan seluruh elemennya diisi dengan 255.**

Selanjutnya, penyusunan citra HSV dilakukan melalui:

```
citraWarna = cv2.merge(
    [citraWarnaHue, kanalLain, kanalLain])
```

Agar dapat ditampilkan dengan imshow (), ci traWarna **ini perlu dikonversi ke format BGR. Perintahnya berupa:**

```
citraWarna = cv2.cvtColor(citraWarna,
                  cv2.COLOR_HSV2BGR)
```

Untuk membuat warna latar belakang hitam, perintah berikut perlu diberikan:

```
citraWarna[citraWarnaHue == OJ = 0
```

Perintah ini membuat semua elemen pada ci traWarna **pada posisi seperti pada** ci traWarnaHue **yang bernilai nol dinolkan.**

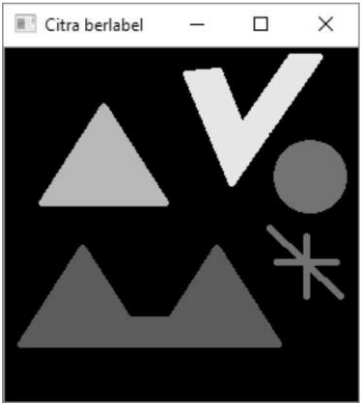
Di akhir, ci traWarna **ditampilkan melalui** imshow () . **Adapun** wai tKey (0) **digunakan untuk menunggu pemakai menekan sebarang tombol agar eksekusi fungsi** visualisasiWarna () **diakhiri.**

Pengujian fungsi visualisasiWarna () **dilakukan melalui sederetan pernyataan berikut:**

```
citra = cv2.imread("struktur.png", 0)
ambang, citra = cv2.threshold(citra, 127, 255,
                        cv2.THRESH_BINARY)

jumObjek, citraBerlabel = cv2.connectedComponents(
                                  citra)
visualisasiWarna(citraBerlabel)
```

Mula-mula, citra struktur. png**dibaca dengan mode skala keabu-abuan. Lalu, hasilnya dikonversi ke citra biner. Citra biner inilah yang dikirimkan ke** connectedComponents () **untuk mendapatkan citra berlabel. Lalu, citra berlabel dikirim ke** visualisasiWarna ().

Hasil skrip ditunjukkan pada Gambar 10.17.

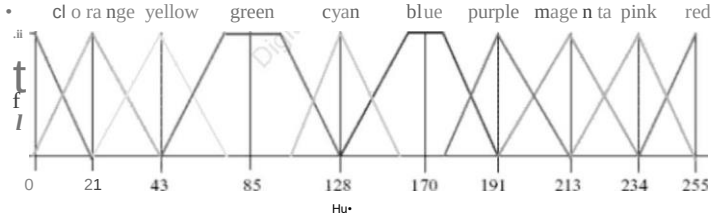


Gambar 10.17 Pengaturan warna objek melalui waI71a hue

10.14 Penerapan Logika Samar untuk Pencarian Warna Dominan

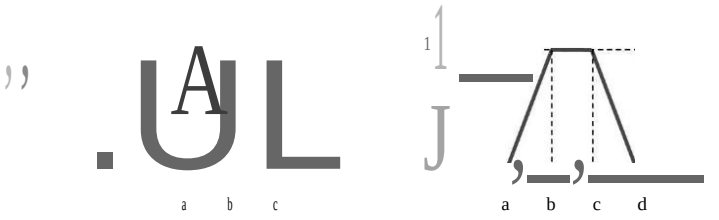
Conteh aplikasi penentuan warna dominan pada daun dibahas oleh Kadir, dkk. (2011). Cara yang dibahas pada artikel tersebut diterapkan pada contoh berikut untuk mencari berkas-berkas citra yang memiliki warna dominan tertentu.

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. Ruang warna HSV dapat digunakan untuk keperluan ini. Pada sistem HSV, komponen *hue* sebenarnya menyatakan warna seperti yang biasa dipahami oleh manusia. Younes, dkk. (2007) membuat model logika samar untuk menyatakan warna seperti terlihat pada Gambar 10.18. Dengan cara seperti ini, derajat keanggotaan suatu warna bisa dihitung.



Gamhar 10.18 Penerapan logika samar pada komponen hue (Sumher: Younes, dkk., f.!007)

Dua jenis fungsi keanggotaan *fuzzy* yang digunakan pada Gambar 10.18 berbentuk segitiga dan trapesium. Gambar 10.19 mencantumkan kedua bentuk tersebut.



(a) Bentuk segitiga (b) Bentuk trapesium

Gambar 10.19 Fungsi keanggotaan pada bentuk segitiga dan trapesium

Fungsi keanggotaan berbentuk segitiga didefinisikan sebagai berikut:

$$A(x,a,b,c) = \begin{cases} 0, & untuk \quad x \le a \\ (x-a)/(b-a), & untuk \quad a < x \le b \\ (c-x)/(c-b), & untuk \quad b < x \le c \\ 0, & untuk \quad x > c \end{cases}$$

Adapun fungsi keanggotaan berbentuk trapesium didefinisikan sebagai berikut:

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

Khusus untuk warna hitam, putih, dan abu-abu dilakukan penanganan secara khusus, yaitu dengan memperhatikan komponen S dan V. Warna hitam diperoleh manakala V bernilai O 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.

Selanjutnya, nilai H, S, V digunakan untuk menentukan kelompok warna piksel. Sebagai contoh, fungsi untuk menentukan warna hijau didefinisikan sebagai berikut:

```
def fHijau(h, s, v):
    if (h == 0) and (s     0):
        derajat   0.0
    else:
        derajat   fTrapesium(43, 65, 105, 128, h)

    return derajat
```

Dalam hal ini, fTrapesium () adalah fungsi untuk mengimplementasikan Gambar 10.18 bagian kanan. Definisinya seperti berikut:

```
def fTrapesium(a, b, c, d, h):
    if (h > a) and (h < b):
        derajat = (h - a) / (b - a)
    elif (h > c) and (h < d):
        derajat = (d - h) / (d - c)
    elif (h >= b) and (h <= c):
        derajat   1.0
    else:
        derajat   0.0
```

Pemrosesan untuk mendapatkan derajat keanggotaan warna masing-masing dilakukan dengan menggunakan rumus berikut:

$$hijau = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} LLfHijau(Hii,Sii,Vii)$$

$$merah = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} LLfMerah(Hii,Sii,Vii)$$

$$kuning = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} LLfKuning(Hii,Sii,Vii)$$

dst.

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

Selanjutnya, pencarian citra dengan warna dominan dilakukan melalui:

warna dominan = max(hijau, merah, kuning, ...)

pencarian(warna): warna dominan = warna

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

Implementasi fungsi cariWarna () **yang ditujukan untuk mencari dominan pada citra dapat dilihat berikut ini.**

Berkas : warnadominan.py

```
# Pencarian warna merah

import cv2
import numpy as np
import os
from matplotlib import pyplot as plt
from operator import itemgetter

# ----------------------------------------------
# Bagian untuk menghitung keanggotaan logika samar
# ----------------------------------------------
def fSegitiga(a, b, c, h):
    if h ==b:
        derajat = 1.0
    elif (h > a) and (h < b):
        derajat = (h - a) / (b - a)
    elif (h > b) and (h < c):
        derajat   (c   h) / (c    b)
    else:
        derajat   0.0

    return derajat

def fSegitigaKiri(a, b, h):
    if h ==b:
        derajat = 1.0
    elif (h > a) and (h < b):
        derajat   (h - a) / (b - a)
    else:
        derajat   0.0
```

```
    return derajat

def fSegitigaKanan(a, b, h):
    if h == a:
        derajat = 1.0
    elif h > a and h < b:
        derajat   (b - h) / (b - a)
    else:
        derajat   0.0

    return derajat

def fTrapesium(a, b, c, d, h):
    if (h > a) and (h < b):
        derajat = (h - a) / (b - a)
    elif (h > c) and (h < d):
        derajat = (d - h) / (d - c)
    elif (h >= b) and (h <= c):
        derajat   1.0
    else:
        derajat   0.0

    return derajat

def fMerah(h, s, v):
    if (h == 0) and (s     0):
        derajat   0.0
    else:
        derajat   fSegitigaKanan(0 21, h) + \
                  fSegitigaKiri(24, 255, h)

    return derajat

def fHijau(h, s, v):
    if (h == 0) and (s    0):
        derajat   0.0
    else:
        derajat   fTrapesium(43, 65, 105, 128, h)

    return derajat

def fKuning(h, s, v):
    if (h == 0) and (s    0):
        derajat   0.0
    else:
        derajat   fSegitiga(21, 43, 65, h)

    return derajat
```

```
def fBiru(h, s, v):
    if (h == 0) and (s    0):
        derajat   0.0
    else:
        derajat   fTrapesium(128, 155, 180, 191, h)

    return derajat

def fUngu(h, s, v):
    if (h == 0) and (s    0):
        derajat   0.0
    else:
        derajat   fSegitiga(180, 191, 213, h)

    return derajat

def fCyan(h, s, v):
    if (h == 0) and (s    0):
        derajat   0.0
    else:
        derajat   fSegitiga(105, 128, 155, h)

    return derajat

def fOrange(h, s, v):
    if (h == 0) and (s    0):
        derajat   0.0
    else:
        derajat   fSegitiga(0, 21, 43, h)

    return derajat

def fMagenta(h, s, v):
    if (h == 0) and (s    0):
        derajat   0.0
    else:
        derajat   fSegitiga(l91, 213, 234, h)

    return derajat

def fPink(h, s, v):
    if (h == 0) and (s    0):
       derajat   0.0
    else:
       derajat   fSegitiga(213, 234, 255, h)

    return derajat

def fPutih(h, s, v):
```

```
    if (s <= 10) and (v >= 250)
        derajat   1.0
    else:
        derajat   0.0

    return derajat

def fAbuAbu(h, s, v)
    if (h == 0) and (s    0) and\
       (v >= 15) and (v < 250):
         derajat = 1.0
    else:
        derajat = 0.0

    return derajat

def fHitam(h, s, v):
    if (h == 0) and (s    0) and (v < 15)
       derajat   1.0
    else:
       derajat   0.0

    return derajat

def cariWarna(warna, lokdir):
# Digunakan untuk mencari gambar yang berada
#     pada folder lokdir
#     dan memiliki warn dominan sesuai
#     dengan argumen warna

  if warna -  "merah" or \
      warna    "biru" or \
      warna    "cyan" or \
      warna    "hijau" or \
      warna    "magenta" or \
      warna    "jingga" or \
      warna    "merah muda" or \
      warna    "ungu" or  \
      warna    "putih" or \
      warna    "hitam" or \
      warna    "abu-abu" or \
      warna    "kuning":
      print("Tunggu. Sedang memroses...")
  else:
      print("Untuk sementara warna yang dapat dipakai: "
+ \
            "merah, biru, cyan, hijau, magenta, jingga, "
+ \
            "merah muda, ungu, " +\
```

```
            "putih, hitam, abu-abu, dan kuning")

    return""

senarai = [] # Kosongkan senarai

# Proses untuk mendapatkan nama-nama berkas
for root, direktori, daftarBerkas in os.walk(lokdir):
    for namaBerkas in daftarBerkas:
        print(namaBerkas)

        citra = cv2.imread(lokdir + "/" + namaBerkas)
        [tinggi, lebar] = citra.shape[:2]

        # Konversi ke HSV
        hsv = cv2.cvtColor(citra, cv2.COLOR_BGR2HSV)

        nilaiAnggota   0.0

        anggotaMerah   0.0
        anggotaBiru = 0.0
        anggotaCyan = 0.0
        anggotaHijau = 0.0
        anggotaMagenta = 0.0
        anggotaOranye = 0.0
        anggotaPink = 0.0
        anggo taUngu = 0.0
        anggotaPutih = 0.0
        anggotaHitam = 0.0
        anggotaAbuAbu   0.0
        anggotaKuning = 0.0

        # Proses untuk semua piksel dalam sebuah  citra
        for yin  range(0, tinggi):
            for x in range(0, lebar):
                h   hsv[x, y, OJ * 255 / 179.0
                s =  hsv[x, y, 1]
                v   hsv[x, y, 2]

                nilaiAnggota = fMerah(h, s, v)
                if nilaiAnggota > 0:
                    anggotaMerah += nilaiAnggota

                nilaiAnggota = fBiru(h, s , v)
                if nilaiAnggota > 0:
                    anggotaBiru += nilaiAnggota

                nilaiAnggota = fCyan(h, s, v)
                if nilaiAnggota > 0:
```

```
                    anggotaCyan += nilaiAnggota

                nilaiAnggota = fHijau(h, s, v)
                if nilaiAnggota > 0:
                    anggotaHijau += nilaiAnggota

                nilaiAnggota = fMagenta(h, s, v)
                if nilaiAnggota > 0:
                    anggotaMagenta += nilaiAnggota

                nilaiAnggota = fOrange(h, s, v)
                if nilaiAnggota > 0:
                    anggotaOranye += nilaiAnggota

                nilaiAnggota = fKuning(h, s, v)
                if nilaiAnggota > 0:
                    anggotaKuning += nilaiAnggota

                nilaiAnggota = fPink(h, s, v)
                if nilaiAnggota > 0:
                    anggotaPink += nilaiAnggota

                nilaiAnggota = fUngu(h, s, v)
                if nilaiAnggota > 0:
                    anggotaUngu += nilaiAnggota

                nilaiAnggota = fPutih (h, s, v)
                if nilaiAnggota > 0:
                    anggotaPutih += nilaiAnggota

                nilaiAnggota = fHitam(h, s, v)
                if nilaiAnggota > 0:
                    anggotaHitam += nilaiAnggota

                nilaiAnggota = fAbuAbu(h, s, v)
                if nilaiAnggota > 0:
                    anggotaAbuAbu += nilaiAnggota

        # Akhir  kedua for

        maks = max(
            anggotaMerah, anggotaBiru, anggotaCyan,
            anggotaHijau, anggotaMagenta,
            anggotaOranye, anggotaPink,
            anggotaUngu, anggotaPutih, anggotaAbuAbu,
            anggotaHitam, anggotaKuning)

        # Mendapatkan hasil yang memenuhi warna
permintaan
```

```
bobot = 0
if warna == "merah":
    if maks == anggotaMerah:
        bobot = maks

elif warna == "biru":
    if maks == anggotaBiru:
        bobot = maks

elif warna == "cyan":
    if maks == anggotaCyan:
        bobot = maks

elif warna == "hijau":
    if maks == anggotaHijau:
          bobot = maks

elif warna == "magenta":
    if maks == anggotaMagenta:
        bobot = maks

elif warna == "jingga":
    if maks == a nggotaOranye:
        bobot =maks

elif warna == "pink":
    if maks == anggotaPink:
        bobot =maks

elif warna == "ungu":
    if maks == anggotaUngu:
        bobot =maks

elif warna == "putih":
    if maks == anggotaPutih:
        bobot =maks

elif warna == "hitam":
    if maks == anggotaHitam:
        bobot =maks

elif warna == "abu-abu":
    if maks == anggotaAbuAbu:
        bobot =maks

elif warna == "kuning":
    if maks == anggotaKuning:
        bobot =maks
```

```
            # Sisipkan ke senarai
            if bobot > 0:
                senarai.append((namaBerkas, bobot))

    # Pengurutan senarai menurut bobot
    #      secara urut turun
    sorted(senarai, key= itemgetter(1), reverse    True)

    # Menampilkan maksimum 24 berkas
    jum = len(senarai)
    print("Jumlah citra =", jum)

    if jum > 24:
        jum = 24

    n = 4
    m    int((jum + 3) / n)

    print("m =", m, ", n =", n)

    # Susun citra
    indeks = 0

    for baris in range(O, m):
      for kolom in range(O, n):
          if (indeks < jum):
              namaBerkas = senarai[indeksJ[OJ
              citra = cv2.imread(lokdir + "/"   + namaBerkas)
              rgb = citra[...,    : --lJ
              pl t.subplot(m, n, indeks + 1)
              plt.imshow(rgb)
              plt.xticks([J), plt.yticks([J)
              plt.title(senarai[indeksJ[OJ, size    8)
              indeks = indeks + 1

    # Tampilkan citra
    plt.show()

    return

# ------------------------------------------------------------

# Program utama

    Warna("merah", "warna")
```

Akhir berkas

Catatan

Untuk mempraktikkan program warnadominan. py, **subfolder** Warna **yang berada di bawah folder** Image **di berkas yang dapat diunduh dengan mengikuti petunjuk di Prakata perlu disalin ke dalam folder** C: \LatOpenCV. **Dengan demikian, pada** C: \La tOpenCV **terdapat subfolder** Warna.

Skrip ini ditujukan untuk mencari warna merah pada semua berkas citra yang berada di folder Warna:

```
cariWarna("merah", "warna")
```

Nama folder harus nama yang berada pada folder yang sama dengan ternpat skrip berada.

Bagian awal pada fungsi cariWarna () **berisi kode untuk memeriksa nama warna pada argumen valid atau tidak. Jika tidak valid, pesan akan ditampilkan dan eksekusi fungsi diakhiri.**

Senarai bernama senarai **digunakan untuk mencatat nama berkas yang warna dominannya sesuai dengan argumen** warna. **Elemennya berupa tupel yang mengandung nama berkas dan derajat keanggotaan warna. Pada keadaan awal,** senarai **dalam keadaan kosong. Hal ini ditentukan melalui:**

```
senarai = []
```

Selanjutnya, pernyataan

```
for root, direktori, daftarBerkas in os.walk(lokdir):
    for namaBerkas in daftarBerkas:
```

digunakan untuk mendapatkan berkas yang berada pada folder lokdir. **Penggunaan** `os.walk()` **mengharuskan keberadaan baris berikut di awal skrip:**

```
import os
```

Pada perintah for di atas, namaBerkas akan berisi nama berkas yang terdapat pada lokdir untuk setiap iterasi.

Perintah

```
citra = cv2.imread(lokdir + "/" + namaBerkas)
```

digunakan untuk membaca data citra pada berkas `lokdir` + "/" + `namaBerkas` dan dicatat pada `citra`. Selanjutnya, ukuran citra diperoleh melalui:

```
tinggi, lebar = citra.shape[:2]
```

Dalam hal ini, tinggi menyatakan jumlah baris pada citra dan lebar berisi jumlah kolom pada citra.

Mengingat komponen *hue* untuk pengolahan warna diperlukan, konversi ke ruang warna HSV dilakukan. Perintah yang digunakan berupa:

```
hsv = cv2.cvtColor(citra, cv2.COLOR_BGR2HSV)
```

Selanjutnya, derajat keanggotaan untuk setiap warna diatur bernilai 0 pada keadaan awal. Itulah sebabnya, terdapat pernyataan seperti:

```
anggotaMerah = 0.0
```

Tahap berikutnya adalah memroses setiap piksel dalam citra. Hal ini dilaksanakan melalui:

```
for yin  range(0, tinggi):
    for x in range(0, lebar):
```

Pertama-tama, perintah

```
h = hsv[x, y, OJ * 255 / 179.0
```

digunakan untuk memperoleh nilai *hue* (yang dicatat pada h). Mengingat nilai *hue* yang diperlukan berkisar antara O dan 255 (sesuai

dengan Gambar 10.14), nilai *hue* pada hsv [x, y, 0] yang bernilai antara O dan 179 perlu dikalikan dengan 255 / 179. Konversi seperti ini tidak diperlukan untuk komponen S dan V.

Deretan pernyataan seperti berikut digunakan untuk menghitung:

```
nilaiAnggota = fMerah(h, s, v)
if nilaiAnggota > 0:
    anggotaMerah = anggotaMerah + nilaiAnggota
```

Pada contoh ini, derajat keanggotaan untuk warna merah diperoleh. Apabila nilainya tidak nol, nilainya ditambahkan ke anggotaMerah. Perlu diketahui, berdasarkan Gambar 10.14, satu nilai *hue* dapat berada pada dua warna. Sebagai contoh, *hue* 130 dapat digolongkan ke warna *cyan* dan biru dengan derajat keanggotaan yang berbeda.

Setelah semua piksel diproses, nilai maksimum derajat keanggotaan semua warna didapatkan dan dicatat di maks. Lalu, dilakukan pemeriksaan terhadap nilai maksimum ini untuk memastikan milik warna yang sesuai dengan argumen warna atau tidak. Jika ya, bobot yang diberi nilai awal sama dengan nol diubah menjadi sama dengan maks.

Lalu, penyisipan ke senarai dilakukan sekiranya bobot bernilai lebih dari 0. Perintahnya berupa:

```
senarai.append((namaBerkas, bobot))
```

Tupel yang disisipkan berisi data nama berkas dan nilai bobot (yang menyatakan nilai derajat keanggotaan warna merah).

Setelah semua berkas diproses, pernyataan berikut digunakan untuk mengurutkan data pada senarai berdasarkan data derajat keanggotaan:

```
sorted(senarai, key= itemgetter(1),reverse= True)
```

Pada perintah ini, `key itemgetter(1)` digunakan untuk mengurutkan data pada senarai berdasarkan data derajat keanggotaan

warna. Sebagaimana diketahui, setiap tupel pada senarai berisi data nama berkas dan derajat keanggotaan warna. Nama berkas mempunyai indeks O dan derajat keanggotaan warna mempunyai indeks 1. Dengan demikian, `itemgetter(1)` merujuk ke derajat keanggotaan warna. Perlu diketahui, jika `itemgetter()` dilibatkan, skrip harus menyertakan

```
from operator import itemgetter
```

Pernyataan

```
jum = len(senarai)
```

digunakan untuk mendapatkan jumlah elemen dalam senarai. Selanjutnya,

```
if jum > 24:
    jum = 24
```

digunakan untuk membuat jum bernilai 24 sekiranya nilainya melebihi 24. Hal ini dimaksudkan untuk membatasi jumlah citra yang ditampilkan.

Pernyataan

```
n = 4
m = int((jum + 3) / n)
```

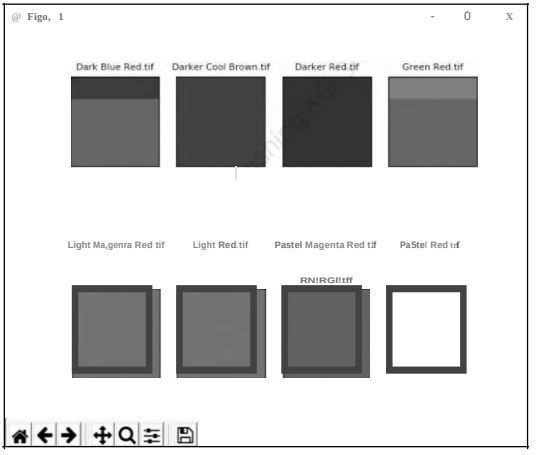
digunakan untuk menentukan jumlah kolom (n) dan jumlah baris (m) untuk menampilkan citra.

Pernyataan-pernyataan selanjutnya digunakan untuk mengatur tampilan citra-citra yang tercatat di senarai.

Tampilan ketika skrip dijalankan semacam berikut:

```
C:\LatOpenCV> python warnadominan.py
Tunggu. Sedang memroses...
Black White.tif
Black.tif
Blue CMYK.tif
Cyan CMYK.tif
Cyan RGB.tif
Dark Blue Red.tif
Dark Blue Violet.tif
Dark Blue.tif
Dark Cool Brown.tif
Dark Cyan Blue.tif
Dark Cyan.tif
```

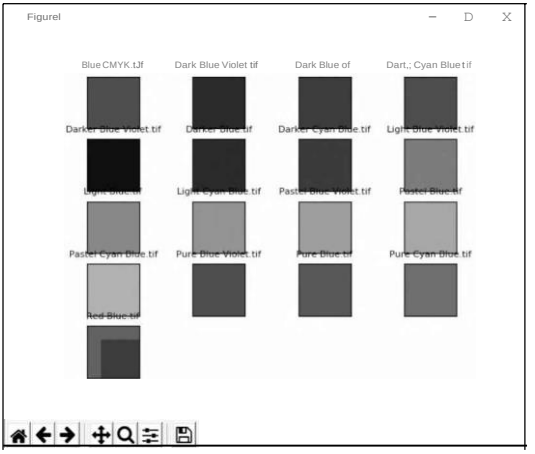
Gambar 10.20 menunjukkan hasil skrip `warnadominan.py`.



Gambar 10. 0 Hasil pencarian warna dominan merah

Gambar 10.21 menunjukkan hasil ketika pernyataan berikut diberikan:

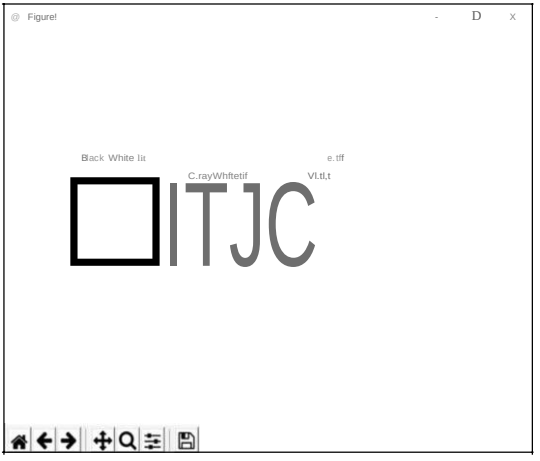
```
cariWarna("biru", "warna")
```



Gambar 10.f!l Hasil pencarian warna dominan biru

Gambar 10.22 menunjukkan hasil ketika pernyataan berikut diberikan:

```
cariWarna("putih", "warna")
```



Gambar 10.f!f! Hasil pencarian warna dominan putih