

BAB 04

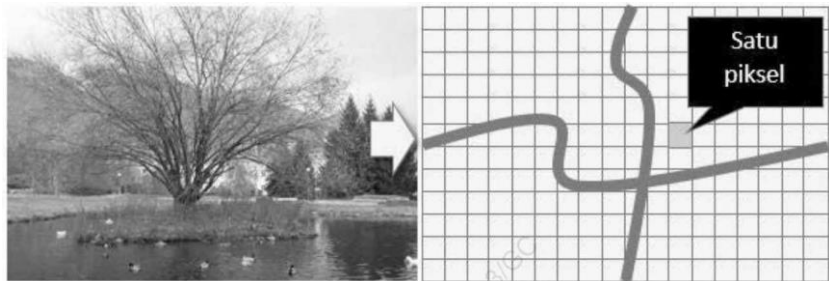
CASAR **PENGOLAHAN CITRA**

Pokok bahasan pada bab ini mencakup hal-hal berikut:

- citra adalah matriks piksel;
- citra untuk percobaan;
- perintah untuk membaca berkas citra;
- pengaksesan piksel pada citra berwarna;
- pengaksesan piksel pada citra berskala keabu-abuan;
- cara mendapatkan informasi mengenai citra;
- pengolahan citra berbasis piksel;
- penggunaan irisan untuk memilih area di dalam citra;
- inversi citra;
- pembuatan citra biner;
- operasi pengolahan citra berbasis matriks;
- cara mengukur lama suatu proses;
- konversi citra berwarna ke citra berskala keabu-abuan dan citra biner;
- dasar pembuatan gambar;
- penyimpanan citra;
- pemisahan dan penggabungan kanal citra.

4.1 Citra adalah Matriks Piksel

Suatu citra (gambar) adalah kumpulan piksel dalam sejumlah baris dan sejumlah kolom. Dengan perkataan lain, citra adalah matriks piksel. Adapun piksel adalah elemen terkecil yang menyusun suatu. Gambar 4.1 memberikan gambaran mengenai suatu citra dan perwakilannya dalam bentuk matriks.

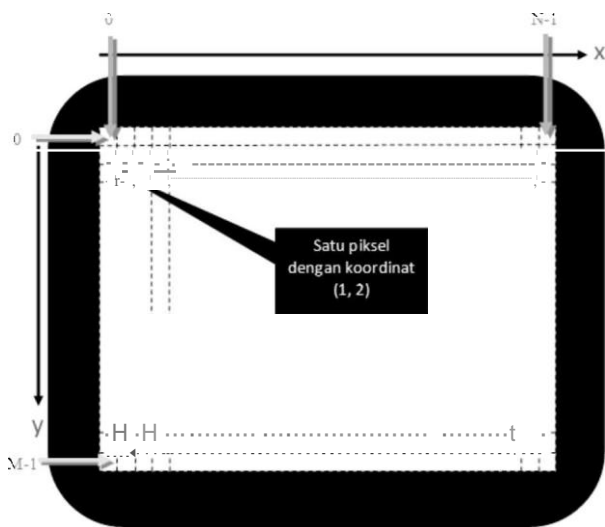


(a) Citra

(b) Matriks piksel

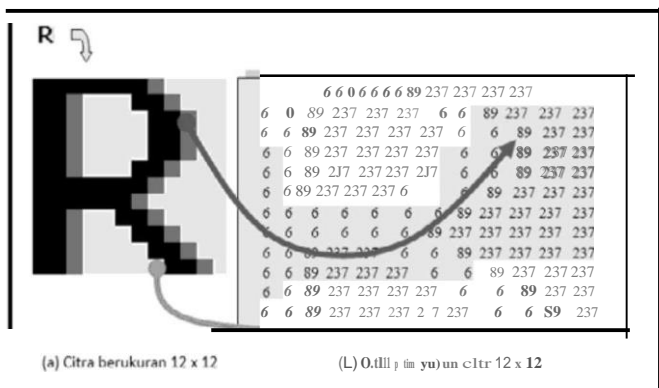
Gambar 4.1 Citra dan matriks piksel

Setiap piksel mempunyai alamat, yang dinyatakan dalam indeks baris dan indeks kolom. Gambar 2.2 memperlihatkan sistem koordinat untuk piksel-piksel yang menyusun suatu citra dengan ukuran $M \times N$ piksel. Indeks baik baris maupun kolom dimulai dari 0. Dengan menggunakan notasi (x, y) untuk mewakili suatu piksel, maka piksel pada pojok kiri-atas mempunyai koordinat $(0, 0)$. Adapun piksel yang ditunjukkan pada Gambar 4.2 mempunyai koordinat $(1, 2)$.



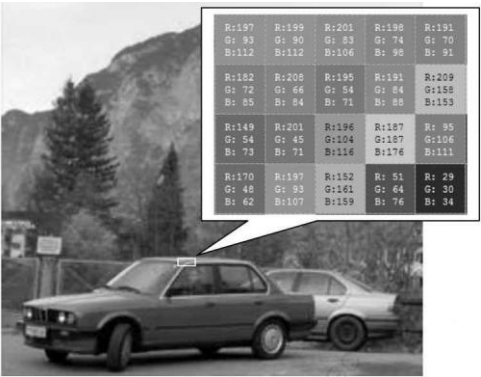
Gambar 4.11 Sistem koordinat piksel dalam citra

Gambar 4.3 menunjukkan contoh citra berskala keabu-abuan berukuran 12 x 12 piksel yang membentuk huruf R. Nilai seperti 6 dan 237 dinamakan intensitas, yang menyatakan seberapa terang suatu piksel. Nilai 0 mewakili gelap (hitam) dan 255 menyatakan terang (putih).



Gambar 4.3 Contoh citra dan nilai-nilai yang menyusunnya

Untuk citra berwarna, setiap piksel diwakili oleh tiga komponen warna, berupa (R), hijau (G), dan biru (B). Gambar 4.4 menunjukkan contoh warna penyusun area kecil pada citra.



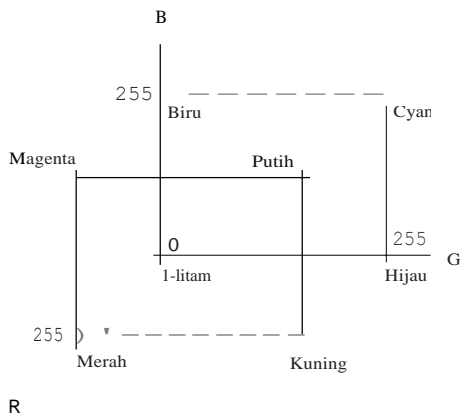
Gambar 4.4 Citra berwarna tersusun atas komponen R, G, dan B

Setiap komponen warna menggunakan delapan bit (nilainya berkisar antara 0 sampai dengan 255). Dengan demikian, kemungkinan warna yang dapat disajikan mencapai 255 x 255 x 255 atau 16.581.375 warna. Tabel 4.1 menunjukkan contoh warna dan nilai R, G, dan B.

Tabel 4.1 Warna dan nilai penyusun warna

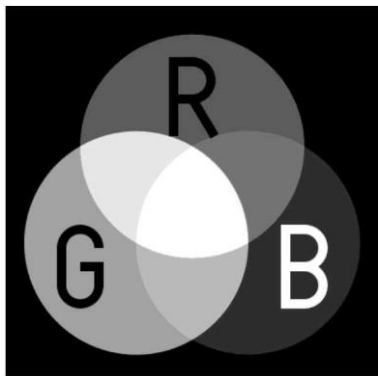
Warna	R	G	B
Merah	255	0	0
Hijau	0	255	0
Biru	0	0	255
Hitam	0	0	0
Putih	255	255	255
Kuning	0	255	255

Gambar 4.5 menunjukkan pemetaan warna dalam ruang tiga dimensi. Tampak bahwa warna hitam menyatakan posisi 0 untuk R, G, dan B. Adapun warna putih menyatakan posisi 255 untuk R, G, dan B.



Gambar 4.5 Warna RGB dalam ruang berdimensi tiga

Penyusunan ketiga komponen dalam citra berwarna secara visual ditunjukkan pada Gambar 4.6. Tampak bahwa perpaduan komponen R, G, dan B menimbulkan efek warna putih.



Gambar 4.6 Efek warna pada perpaduan komponen R, G, dan B
(Sumber: Jacobolus, Wikimedia)

4.2 Citra untuk Percobaan

Untuk keperluan percobaan di bab ini dan bab-bab selanjutnya, sejumlah berkas citra diperlukan. Untuk itu, berkas-berkas gambar berikut perlu diunduh dan diletakkan di folder C: \LatOpenCV.

1. Citra berwarna:

- <https://homepages.cae.wisc.edu/~ece533/images/baboon.png>
- <https://homepages.cae.wisc.edu/~ece533/images/airplane.png>
- <https://homepages.cae.wisc.edu/~ece533/images/peppers.png>
- <https://homepages.cae.wisc.edu/~ece533/images/fruits.png>
- <https://homepages.cae.wisc.edu/~ece533/images/lena.png>

2. Citra berskala keabu-abuan:

- <https://homepages.cae.wisc.edu/~ece533/images/boat.png>
- <https://homepages.cae.wisc.edu/~ece533/images/cameraman.tif>
- <https://homepages.cae.wisc.edu/~ece533/images/goldhill.png>
- <https://homepages.cae.wisc.edu/~ece533/images/barbara.png>

3. Citra berindeks:

<https://homepages.cae.wisc.edu/~ece533/images/lena.bmp>

4.3 Perintah untuk Membaca Berkas Citra

Sebagaimana telah dipraktikkan beberapa kali di depan, berkas citra dibaca melalui `cv2.imread()`. Namun, sejauh ini, argumen yang digunakan hanya satu, yaitu nama berkas citra. Sesungguhnya, argumen kedua juga bisa diberikan. Nilainya dapat berupa salah satu konstanta berikut:

- `cv2.IMREAD_COLOR` atau angka 1: citra dimuat dalam bentuk berwarna tanpa informasi transparansi;

- `cv2.IMREAD_GRAYSCALE` atau angka 0: citra dimuat dalam bentuk berskala keabu-abuan;
- `cv2.IMREAD_UNCHANGED` atau angka -1: citra dimuat seperti apa adanya sehingga informasi transparansi kalau ada akan disertakan.

Catatan

- Secara bawaan, pada saat argumen kedua tidak disertakan pada `cv2.imread()`, `cv2.IMREAD_COLOR` akan digunakan.
 - Format yang didukung oleh `imread()` mencakup Windows bitmaps (*.bmp, *dib), Portable image formats (*.pbm, *.pgm, *.ppm), Sun rasters (*.sr, *.ras), JPEG (*.jpeg, *.jpg, *.jpe), JPEG 2000 (*.jp2), Portable Network Graphics (*.png), TIFF (*.tiff, *.tif), dan **WebP** (*.webp).
-

Contoh berikut menunjukkan cara membaca berkas baboon.png



untuk citra berskala keabu-abuan:

Berkas : grayscale.py

```
# Pembacaan baboon.png dalam bentuk skala keabu-abuan

import cv2

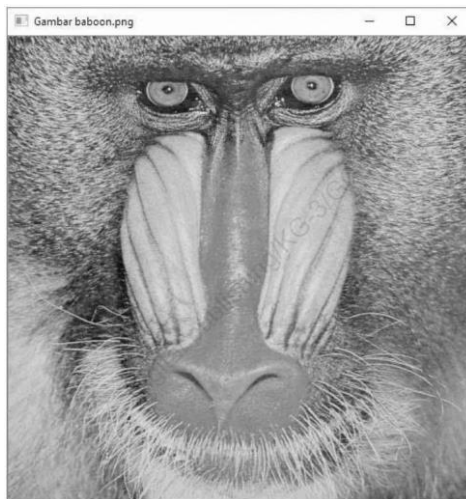
citra = cv2.imread('baboon.png', cv2.IMREAD_GRAYSCALE)
if not citra is None:
    cv2.imshow('Gambar baboon.png', citra)
    cv2.waitKey(0)
```

Akhir berkas

Penyebutan `cv2.IMREAD_GRAYSCALE` pada `imread()` membuat objek citra berisi citra berskala keabu-abuan. Perintah `if` digunakan untuk memastikan bahwa citra `baboon.png` berhasil dibaca. Kalau berhasil dibaca, citra ditampilkan melalui `cv2.imshow()`. Adapun `cv2.waitKey(0)` menunggu pemakai menekan sebarang tombol untuk mengakhiri skrip. Gambar 4.7 memperlihatkan hasil pemanggilan:

```
python grayscale.py
```

di Command Prompt.



Gambar 4.7 Tampilan baboon.png dalam bentuk skala abu-abu walaupun aslinya berwarna

4.4 Pengaksesan Piksel pada Citra Berwarna

Sebelum membahas cara mengakses piksel dalam suatu gambar, marilah untuk memuat gambar `baboon.png` ke dalam memori komputer. Perintah yang digunakan diperlihatkan berikut ini:


```

.....
•>>>import cv2(V
  >>> citra = cv2.imread('baboon.png')
: >>>
.....

```

Pada contoh ini, `import cv2` disertakan dengan asumsi bahwa modul `cv2` belum diimpor.

Setelah objek `citra` berisi data gambar pada `baboon.png`, pengaksesan terhadap piksel bisa dimulai. Pertama-tama, perlu diketahui bahwa objek citra sebenarnya adalah suatu matriks. Untuk citra berwarna, setiap piksel yang dikandungnya dapat diakses dengan notasi seperti berikut:

citra[indeks_baris, indeks_kolom]

Contoh berikut menampilkan nilai piksel pada baris dengan indeks 0 dan kolom dengan indeks 1:

```

.....
: >>> citra[0, 1]
: array([31, 57, 63], dtype=uint8)
: >>>
.....

```

Pada hasil yang diberikan, senarai `[31, 57, 63]` menyatakan data piksel yang secara berturutan menyatakan komponen B, G, dan R. Informasi `uint8` menyatakan bahwa data berupa bilangan bulat berukuran 8 bit (atau disebut 1 byte).

Pengaksesan data per komponen warna juga bisa dilakukan. Contoh:

```

>>> citra[0, 1][0]
31
>>> citra[0, 1][1]
57
>>> citra[0, 1][2]
63
>>>

```

Pada contoh ini, [0] menyatakan komponen B, [1] menyatakan komponen G, dan [2] menyatakan komponen R.

Dengan menggunakan notasi-notasi seperti itu dimungkinkan untuk mengubah nilai piksel. Contoh berikut memperlihatkan cara mengubah piksel pada baris dengan indeks 0 dan kolom dengan indeks 1 menjadi berwarna hitam:

```

: >>> citra[0, 1][0] = 0<P
: >>> citra[0, 1][1] = 0<P
: >>> citra[0, 1][2] = 0<P
: >>>

```

Ketiga pernyataan tersebut dapat ditulis menjadi:

```

: >>> citra[0, 1][0] = [0, 0, 0]<P
: >>>

```

Catatan

Notasi seperti `citra[0, 1][2]` boleh ditulis menjadi:
`citra[0, 1, 2]`

4.5 Pengaksesan Piksel pada Citra Berskala Keabu-abuan

Untuk citra berskala keabu-abuan, setiap piksel yang dikandungnya dapat diakses dengan notasi seperti berikut:

citra[indeks_baris, indeks_kolom]

Untuk mempraktikkan citra berskala keabu-abuan terhadap baboon.png, perlu dilakukan pemberian perintah seperti berikut terlebih dahulu:

```

' .....
: >>> citra = cv2.imread('baboon.png')
: >>> citra = cv2.cvtColor(citra, cv2.COLOR_BGR2GRAY)
: >>>
' .....

```

Pada contoh ini, `cv2.cvtColor(citra, cv2.COLOR_BGR2GRAY)` digunakan untuk mengonversi `citra` menjadi citra berskala keabu-abuan. Mengingat nilai balik ini diberikan kembali ke `citra`, `citra` berisi gambar monyet yang berskala keabu-abuan.

Contoh berikut menampilkan nilai piksel pada baris dengan indeks 0 dan kolom dengan indeks 1:

```

' .....
: >>> citra[0, 1]
: 56
: >>>
' .....

```

Berbeda dengan pada citra berwarna, hasilnya berupa nilai tunggal pada citra berskala keabu-abuan. Nilai 56 tersebut menyatakan intensitas piksel pada baris dengan indeks 0 dan kolom dengan indeks 1.

Dengan menggunakan notasi seperti itu dimungkinkan untuk mengubah nilai piksel. Contoh berikut memperlihatkan cara mengubah piksel pada baris dengan indeks 0 dan kolom dengan indeks 1 menjadi berwarna hitam:

```

' .....
: >>> citra[0, 1] = 04)
: >>>
' .....

```

4.6 Cara Mendapatkan Informasi Mengenai Citra

Informasi mengenai citra yang antara lain berupa ukuran gambar yang berupa jumlah baris dan jumlah kolom, serta jumlah kanal gambar kadang diperlukan untuk kepentingan pengolahan citra. Data ketiga informasi ini terdapat pada properti `shape` milik objek yang dihasilkan

oleh `cv2.imread()` dengan argumen kedua berupa `cv2.IMREAD_UNCHANGED`. Contoh diperlihatkan berikut ini:

```
.....>>> citra = cv2.imread('baboon.png',  
cv2.IMREAD_UNCHANGED) 4)  
>>> citra.shape  
(512, 512, 3)  
>>> citra = cv2.imread('goldhill.png',  
cv2.IMREAD_UNCHANGED) 4)  
>>> citra.shape  
(512, 512)  
>>>
```

Perhatikan bahwa hasil `citra.shape` berupa tupel dengan tiga elemen untuk citra `baboon.png` yang berupa citra berwarna. Elemen pertama menyatakan jumlah baris pada citra, elemen kedua menyatakan jumlah kolom pada citra, dan elemen ketiga menyatakan jumlah kanal (G, B, dan R). Namun, pada citra `goldhill.png` yang berupa citra berskala keabu-abuan, jumlah elemen tupel hanya dua, yang secara berturut-turut berisi informasi jumlah baris dan jumlah kolom pada citra.

Skrip berikut menunjukkan pemakaian properti `shape` untuk menentukan citra yang dibaca berupa citra berwarna atau citra berskala



buan:

Berkas : `jenisgbr.py`

```
# Penentuan gambar sebagai gambar berwarna atau  
# berskala keabu-abuan
```

```
import cv2  
import sys
```

```
if len(sys.argv) == 1:  
    print('Masukkan nama berkas gambar')  
else:  
    berkas = sys.argv[1]
```

```

citra = cv2.imread(berkas, cv2.IMREAD_UNCHANGED)
if citra is None:
    print('Tidak dapat membaca berkas', berkas)
else:
    info= citra.shape
    if len(info) ==2:
        print('Citra berskala keabu-abuan')
    else:
        print('Citra berwarna')

```

Akhir berkas

Skrip ini digunakan untuk menentukan jenis citra dengan nama berkas gambar dimasukkan sebagai argumen baris perintah sewaktu skrip dipanggil. Nama berkas diperoleh melalui:

```
berkas = sys.argv[1]
```

Penentuan citra sebagai citra berskala keabu-abuan atau citra berwarna dilakukan melalui kondisi `len (info) == 2` dengan `info` adalah hasil penugasan dari `citra.shape`. Artinya, kalau jumlah elemen dalam `info` sebesar 2, citra tersebut adalah citra berskala keabu-abuan. Untuk keadaan sebaliknya, dapat dipastikan bahwa citra tersebut adalah citra berwarna.

Berikut adalah hasil pengujian skrip `jenisgbr.py`:

```

.....
C:\LatOpenCV>python jenisgbr.py baboon.png<P
Citra berwarna

C:\LatOpenCV>python jenisgbr.py goldhill.png<P
Citra berskala keabu-abuan

C:\LatOpenCV>

```

4.7 Pengolahan Citra Berbasis Piksel

Dengan melakukan pengubahan nilai pada piksel, berbagai aplikasi terhadap citra bisa dilakukan. Contoh yang sederhana adalah untuk

membuat tepi gambar diubah menjadi hitam sehingga seperti diberi



Contoh berikut berlaku untuk citra berskala keabu-abuan:

Berkas : jenis1:llr.11y

```
# Contoh pemberian bingkai pada
# citra berskala keabu-abuan

import cv2

citra = cv2.imread('goldhill.png')
hasil = citra.copy() # Salin isi citra

# Pengolahan citra
TEBAL = 20
HITAM = 0

jumBaris = hasil.shape[0]
jumKolom = hasil.shape[1]

#--- Bingkai di atas
for baris in range(TEBAL):
    for kolom in range(jumKolom):
        hasil[baris, kolom] = HITAM

#--- Bingkai di bawah
for baris in range(jumBaris - TEBAL - 1, jumBaris):
    for kolom in range(jumKolom):
        hasil[baris, kolom] = HITAM

#--- Bingkai di kiri
for baris in range(TEBAL, jumBaris - TEBAL - 1):
    for kolom in range(TEBAL):
        hasil[baris, kolom] = HITAM

#--- Bingkai di kanan
for baris in range(TEBAL, jumBaris - TEBAL - 1):
    for kolom in range(jumKolom - TEBAL - 1, jumKolom):
        hasil[baris, kolom] = HITAM

cv2.imshow('Gambar asal', citra)
cv2.imshow('Gambar hasil', hasil)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Akhir berkas

Pada skrip ini, pernyataan berikut digunakan untuk menyalin isi citra ke hasil:

```
hasil= citra.copy()
```

Metode `copy()` diperlukan agar nilai pada hasil bersifat independen terhadap citra. Jika perintah berikut diberikan sebagai pengganti perintah di atas, baik `hasil` dan `citra` merujuk ke data yang sama:

```
hasil= citra
```

Dengan demikian, perubahan pada `hasil` akan berimplikasi pada `citra`. Oleh karena itu, `copy()` diperlukan.

Konstanta `TEBAL` menyatakan tebal bingkai dalam satuan piksel. Konstanta ini digunakan pada empat proses untuk membuat kotak yang diwarnai hitam. Keempat kotak ini terletak pada bagian atas, bagian bawah, bagian kiri, dan bagian kanan gambar.

Pernyataan berikut digunakan untuk memperoleh jumlah baris pada citra:

```
jumBaris = hasil.shape[0]
```

Adapun pernyataan berikut digunakan untuk memperoleh jumlah kolom pada citra:

```
jumKolom = hasil.shape[1]
```

Prinsip pembuatan keempat bingkai sama. Sebagai contoh, berikut adalah perintah yang digunakan untuk membentuk bingkai pada bagian atas:

```
for baris in range(TEBAL):  
    for kolom in range(jumKolom):  
        hasil[baris, kolom] = HITAM
```

Pada perintah ini, `range (TEBAL)` digunakan untuk mendapatkan senarai dengan elemen berupa 0, 1, 2, ..., `TEBAL -1`. Artinya, nilai baris untuk satu saat akan bernilai secara berturut-turut sesuai dengan nilai pada senarai tersebut. Adapun nilai pada kolom akan bervariasi dari 0 hingga `jumBaris - 1`. Nah, untuk setiap piksel pada posisi nilai baris dan kolom dibuat menjadi berwarna hitam melalui:

```
hasil[baris, kolom] = HITAM
```

Kedua perintah berikut digunakan untuk menampilkan citra pada `ci tra` dan `hasil` pada jendela yang berbeda:

```
cv2.imshow('Gambar asal', citra)
cv2.imshow('Gambar hasil', hasil)
```

Setelah `cv2.waitKey(0)` yang digunakan untuk menunggu pemakai menekan sebarang tombol terdapat perintah:

```
cv2.destroyAllWindows()
```

Pernyataan ini digunakan untuk membebaskan semua jendela gambar. Namun, karena pernyataan ini diletakkan sebelum skrip berakhir, pernyataan ini bisa tidak ditulis. Hal ini disebabkan, program python dengan sendirinya akan membebaskan semua memori yang digunakan semua objek ketika program berakhir.

Gambar 4.8 menunjukkan dua jendela yang dihasilkan oleh skrip `bingkai.py`.



Gambar 4.8 Pemberian bingkai pada gambar untuk citra berskala keabu-abuan

Pada citra berwarna, pemberian nilai 0 untuk membentuk bingkai hitam perlu dilakukan pada ketiga komponen G, B, dan R. Contoh berikut



```
# Contoh pemberian bingkai pada
# citra berwarna

import cv2

citra = cv2.imread('peppers.png')
hasil = citra.copy() # Salin isi citra

# Pengolahan citra
TEBAL = 20
HITAM = [0, 0, 0]

jumBaris = hasil.shape[0]
jumKolom = hasil.shape[1]

# --- Bingkai di atas
for baris in range(TEBAL):
    for kolom in range(jumKolom):
        hasil[baris, kolom] = HITAM

# --- Bingkai di bawah
for baris in range(jumBaris - TEBAL - 1, jumBaris):
```

```

for kolom in range(jumKolom):
    hasil[baris, kolom] = HITAM

#--- Bingkai di kiri
for baris in range(TEBAL, jumBaris - TEBAL - 1):
    for kolom in range(TEBAL):
        hasil[baris, kolom] = HITAM

#--- Bingkai di kanan
for baris in range(TEBAL, jumBaris - TEBAL - 1):
    for kolom in range(jumKolom - TEBAL - 1, jumKolom):
        hasil[baris, kolom] = HITAM

cv2.imshow('Gambar asal', citra)
cv2.imshow('Gambar hasil', hasil)
cv2.waitKey(0)
cv2.destroyAllWindows()

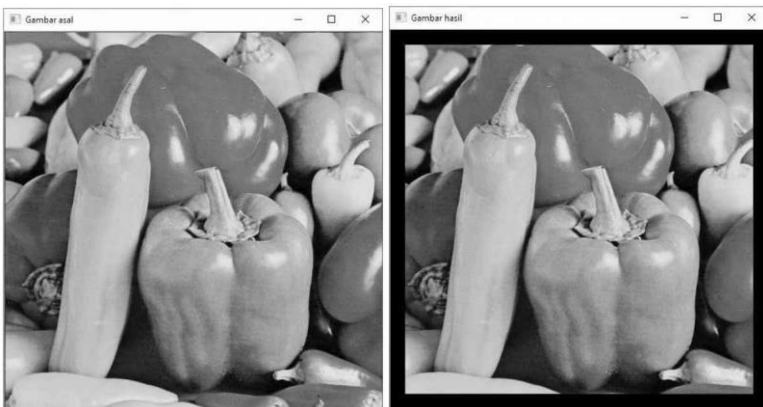
```

Akhir berkas

Perbedaan utama dengan pada gambar beraras keabu-abuan hanya terletak pada kode berikut:

```
HITAM = [0, 0, 0]
```

Pada citra berskala keabu-abuan, HITAM berupa besaran skalar, sedangkan pada citra berwarna berupa senarai yang mengandung tiga elemen, yang menyatakan nilai ketiga komponen penyusun warna, yaitu B, G, dan R.



Gambar 4.9 Pemberian bingkai pada gambar untuk citra berwarna

4.8 Penggunaan Irisan untuk Memilih Area dalam Citra

Selain menggunakan indeks untuk mengakses satu piksel, dimungkinkan untuk menggunakan irisan untuk mengakses sekumpulan piksel. Pola irisan yang umum berupa:

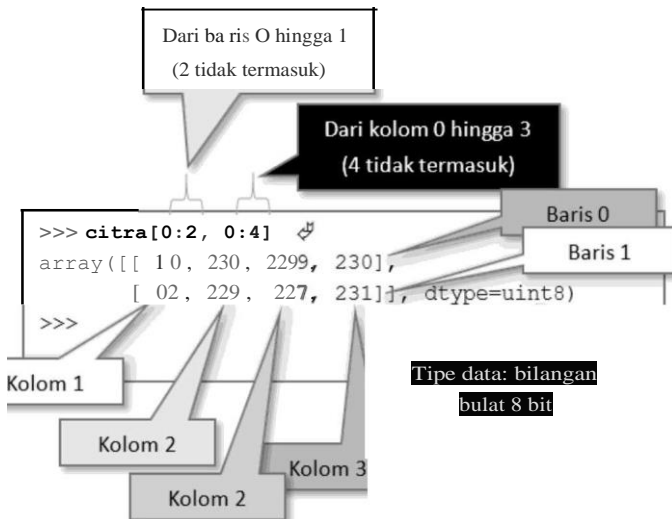
m:n

Irisan ini berarti dari m hingga n-1.

Contoh berikut menunjukkan contoh penggunaan irisan untuk memperoleh sejumlah piksel:

```
>>> citra = cv2.imread('goldhill.png', -1)
>>> citra[0:2, 0:4]
array([[ 10, 230, 229, 230],
       [202, 229, 227, 231]], dtype=uint8)
>>>
```

Gambar 4.10 menjelaskan tentang hasil irisan di atas.



Gambar 4.10 Penjelasan irisan untuk mengakses area pada citra

Skrip berikut menunjukkan contoh penggunaan irisan untuk memperoleh bagian yang pojok kiri-atasnya adalah (180, 170) dan pojok kanan-bawah berurutan (480, 424).

11!1 Berkas : jenis:1tr.11y

```
# Contoh penggunaan irisan untuk memperoleh  
#      suatu bagian pada citra
```

```
import cv2
```

```
citra = cv2.imread('peppers.png')  
bagian = citra[180:490, 170:425]  
cv2.imshow('Irisan', bagian)  
waitKey(0)
```

Akhir berkas

Hasilnya diperlihatkan pada Gambar 4.11.



Gambar 4.11 Penjelasan irisan untuk mengakses area pada citra



ahan pada citra juga dapat dilakukan melalui irisan. Contoh:

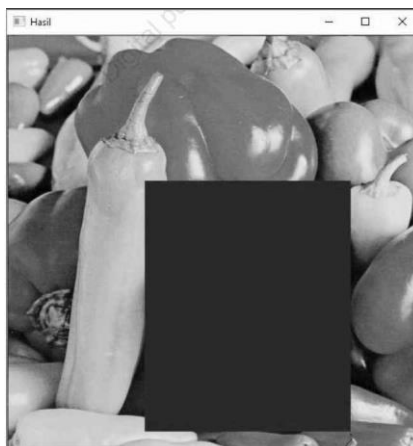
Berkas : jenis1:llr.11y

```
# Contoh penggunaan irisan untuk mengubah  
#      suatu bagian pada citra  
  
import cv2
```

```
citra = cv2.imread('peppers.png')  
citra[180:490, 170:425] = [255, 0, 0]  
cv2.imshow('Hasil', citra)  
waitKey(0)
```

Akhir berkas

Pada contoh ini, perintah berikut digunakan untuk membuat area yang dinyatakan dengan `citra[180:490, 170:425]` diisi dengan warna `[255, 0, 0]`. Karena komponen G diisi dengan 255 dan B serta R bernilai 0, maka warna yang digunakan adalah biru. Hasilnya diperlihatkan pada Gambar 4.12.



Gambar 4.1!2 Hasil pengubahan area pada citra melalui irisan

Selain menggunakan pola `m:n`, irisan dapat berupa seperti berikut:

- 1) `m`: (berarti dari indeks m hingga terakhir);

- 2) ... (berarti semua, yang apabila diletakkan pada bagian baris berarti semua baris dan jika diletakkan pada bagian kolom berarti semua kolom).

Skrip berikut memberikan gambaran penggunaan irisan berbentuk m:



```
# Contoh penggunaan irisan untuk memperoleh
#      suatu bagian pada citra

import cv2

citra = cv2.imread('goldhill.png', cv2.IMREAD_UNCHANGED)
bagianA = citra[0:100, ...]
bagianB = citra[..., 250:]

cv2.imshow('Bagian A', bagianA)
cv2.imshow('Bagian B', bagianB)
```

```
cv2.waitKey(0)
```

Akhir berkas

Pada skrip ini, perintah berikut membuat bagianA berisi baris 0 hingga 99 untuk semua kolom:

```
bagianA = citra[0:100, ...]
```

Tanda ... bisa juga ditulis menjadi ..

Hasilnya seperti terlihat pada Gambar 4.13.



Gambar 4.19 Contoh hasil melalui irisan ... pada kolom

Adapun perintah berikut membuat bagianB berisi kolom 250 hingga kolom terakhir untuk semua baris:

```
bagianB = citra[..., 250:]
```

Tanda ... boleh ditulis menjadi :. Hasilnya seperti terlihat pada Gambar 4.14.



Gambar 4.14 Contoh hasil melalui irisan ... pada baris

4.9 Inversi Citra

Inversi citra berarti membalik citra. Di bidang fotografi dengan film, inversi menghasilkan gambar negatif. Pada citra berskala keabu-abuan, hal ini dapat dilakukan dengan menerapkan rumus berikut:

$$\text{Intensitas baru} = 255 - \text{intensitas lama}$$



rikut memberikan contoh penerapan rumus ini:

Berkas : inversi.py

```
# Inversi citra

import cv2
import sys
import numpy as np

if len(sys.argv) == 1:
    print('Masukkan nama berkas gambar')
else:
    berkas = sys.argv[1]
    citra = cv2.imread(berkas, cv2.IMREAD_GRAYSCALE)
    if citra is None:
        print('Tidak dapat membaca berkas', berkas)
    else:
        jumBaris = citra.shape[0]
        jumKolom = citra.shape[1]
        hasil = np.zeros((jumBaris, jumKolom), np.uint8)

        for brs in range(jumBaris) :
            for kol in range(jumKolom):
                hasil[ brs, kol] = 255 - citra[brs, kol]

        cv2.imshow('Asli', citra)
        cv2.imshow('Hasil', hasil)
        cv2.waitKey(0)
```

Akhir berkas

Skrrip ini memproses berkas gambar yang dimasukkan melalui baris perintah. Modul numpy dilibatkan untuk pembentukan larik citra. Itulah sebabnya, terdapat perintah:

```
import numpy as np
```

Penggunaan as np dimaksudkan agar modul numpy dapat diakses melalui kata np.

Jumlah baris dan jumlah kolom diperoleh melalui:

```
jumBaris    citra.shape[0]
jumKolom    citra.shape[1]
```


Baik `jumBaris` maupun `jumKolom` akan digunakan pada `for`.

Perintah berikut digunakan untuk membentuk matriks hasil dengan ukuran `jumBaris x jumKolom` dengan data bertipe `np. uint8` (8 bit):

```
hasil = np.zeros({jumBaris, jumKolom}, np.uint8)
```

Dengan demikian, `hasil` sebenarnya adalah citra yang seluruh nilai elemennya berupa 0. Dengan perkataan lain, `hasil` adalah citra berwarna hitam.

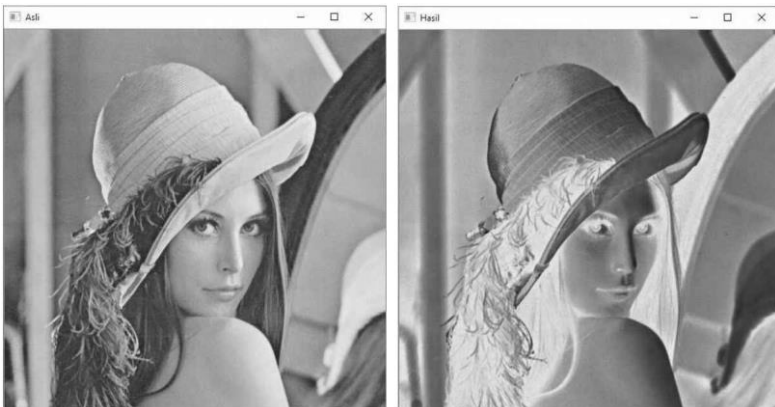
Pernyataan berikut digunakan melakukan perhitungan rumus inversi citra untuk setiap piksel:

```
for brs in range(jumBaris):
    for kol in range(jumKolom):
        hasil[brs, kol] = 255 - citra[brs, kol]
```

Setelah pernyataan ini berakhir, `hasil` berisi inversi citra pada `citra`.

Gambar 4.15 menunjukkan hasil setelah perintah berikut diberikan di Command Prompt:

```
python invesi.py lena.bmp
```




Gambar 4.15 Inversi citra pada lena.bmp

4.10 Pembuatan Citra Biner

Citra biner adalah citra yang hanya mengandung dua nilai, yang biasanya berupa 0 dan 1. Dalam hal ini, 0 menyatakan hitam dan 1 menyatakan putih. Jika menggunakan citra berskala keabu-abuan, nilai 1 dapat dikonversi ke 255. Citra ini menjadi dasar dalam mengenali objek.

Secara prinsip, pengubahan citra berskala keabu-abuan menjadi citra biner dilakukan dengan menentukan nilai ambang yang dijadikan sebagai pedoman untuk mengubah suatu intensitas menjadi nilai 0 atau 255. Algoritmanya seperti berikut:

```
JIKA intensitas >= AMBANG
    Intensitas <- 255
SEBALIKNYA
    Intensitas <- 0
```

dannya adalah seperti berikut:

Berkas :inversi.11y

```
#Konversi ke biner

import cv2
import sys
import numpy as np

if len(sys.argv) != 3:
    print('Masukkan nama berkas gambar dan nilai ambang')
else:
    berkas = sys.argv[1]
    ambang = int(sys.argv[2])
    citra = cv2.imread(berkas, cv2.IMREAD_GRAYSCALE)
    if citra is None:
        print('Tidak dapat membaca berkas', berkas)
    else:
        jumBaris = citra.shape[0]
        jumKolom = citra.shape[1]
        hasil = np.zeros((jumBaris, jumKolom), np.uint8)
```

```

for brs in range(jumBaris):
    for kol in range(jumKolom):
        if citra[brs, kol] >= ambang:
            hasil[brs, kol] = 255

cv2.imshow('Asli', citra)
cv2.imshow('Hasil', hasil)
cv2.waitKey(0)

```

Akhir berkas

Skrip ini memproses berkas gambar yang dimasukkan melalui baris perintah. Modul `numpy` dilibatkan untuk pembentukan larik citra. Itulah sebabnya, terdapat perintah:

```
import numpy as np
```

Penggunaan `as np` dimaksudkan agar modul `numpy` dapat diakses melalui kata `np`.

Skrip ini membaca dua argumen baris perintah berupa nama berkas gambar dan nilai ambang yang digunakan sebagai nilai untuk menentukan setiap piksel menjadi 0 atau 255. Perintahnya berupa:

```

berkas = sys.argv[1]
ambang = int(sys.argv[2])

```

Jumlah baris dan jumlah kolom diperoleh melalui:

```

jumBaris = citra.shape[0]
jumKolom = citra.shape[1]

```

Baik `jumBaris` maupun `jumKolom` akan digunakan pada `for`.

Perintah berikut digunakan untuk membentuk matriks hasil dengan ukuran `jumBaris x jumKolom` dengan data bertipe `np.uint8` (8 bit):

```
hasil = np.zeros((jumBaris, jumKolom), np.uint8)
```

Dengan demikian, **hasil** sebenarnya adalah citra yang seluruh nilai elemennya berupa 0. Dengan perkataan lain, **hasil** adalah citra berwarna hitam.

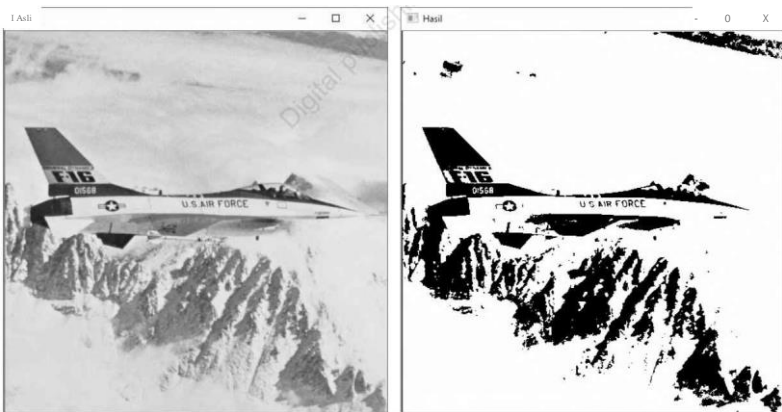
Pernyataan berikut digunakan melakukan perhitungan rumus inversi citra untuk setiap piksel:

```
for brs in range(jumBaris):
    for kol in range(jumKolom):
        if citra[brs, kol] >= ambang:
            hasil[brs, kol] = 255
```

Setelah pernyataan ini berakhir, **hasil** berisi citra biner dengan nilai 0 dan 255.

Gambar 4.16 menunjukkan hasil setelah perintah berikut diberikan di Command Prompt:

```
python biner.py airplane.png 128
```



Gambar 4.16 Inversi citra dengan nilai ambang sebesar 128

Gambar 4.17 menunjukkan hasil setelah perintah berikut diberikan di Command Prompt:

```
python biner.py airplane.png 80
```



Gambar 4.17 Inversi citra dengan nilai ambang sebesar 80

4.11 Operasi Pengolahan Citra Berbasis Matriks

Apabila A dan B adalah matriks, operasi seperti $A + B$ atau $A + 5$ dimungkinkan. Pada $A + B$, setiap elemen di matriks A dan B pada posisi yang sama dijumlahkan. Adapun $A + 5$ menghasilkan matriks dengan setiap elemennya adalah hasil penjumlahan elemen di A pada posisi yang sama dengan angka 5.

Contoh berikut memberikan gambaran operasi $255 - A$ dengan A adalah matriks yang dapat digunakan untuk melakukan inversi citra:

11!11

Berkas : negatif.py

```
# Pengaturan kecerahan
```

```
import cv2
```

```
citra = cv2.imread('goldhill.png', 0)
cv2.imshow('Citra asli', citra)
```

```
hasil = 255 - citra
cv2.imshow('Citra hasil', hasil)
```

```
waitKey(0)
```

Akhir brkas

Pada skrip ini, pernyataan

```
hasil = 255 - citra
```

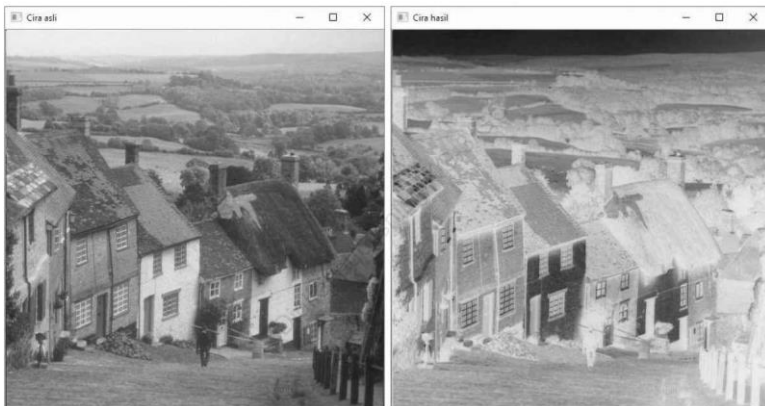
identik dengan pernyataan

```
for brs in range(jumBaris):  
    for kol in range(jumKolom):  
        hasil[brs, kol] = 255 - citra[brs, kol]
```

yang terdapat pada skrip `inversi. py`.

Gambar 4.18 menunjukkan contoh hasil pemanggilan

```
python goldhill.py
```



Gambar 4.1 B Operasi 55 - citra untuk mendapatkan Inversi citra

Kelebihan operasi citra berbasis matriks secara langsung terletak pada kecepatan eksekusi. Namun, kelemahannya terletak pada ketiadaan cara untuk mengontrol piksel untuk keadaan tertentu. Sebagai contoh, pada operasi $A + b$ dengan b adalah skalar, ada kemungkinan hasil penjumlahan suatu elemen dengan b melebihi 255. Pada keadaan seperti ini, nilai yang dihasilkan justru berubah menjadi kecil. Efeknya malah menimbulkan warna hitam. Hal ini dapat dipraktikkan dengan menggunakan skrip berikut:

```
# Pengaturan kecerahan
```

```
import cv2
```

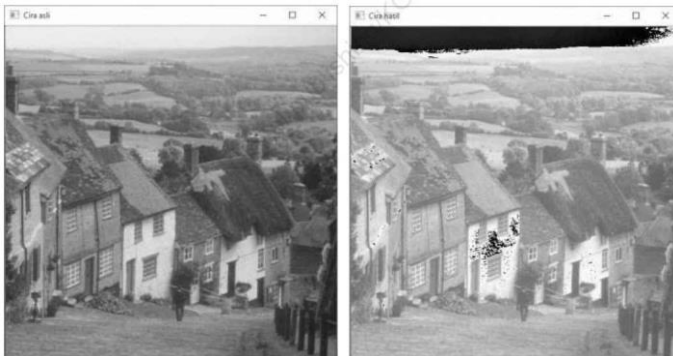
```
citra = cv2.imread('goldhill.png', 0)
cv2.imshow('Cira asli', citra)
```

```
hasil = citra + 50
cv2.imshow('Cira hasil', hasil)
```

```
cv2.waitKey(0)
```

Akhir berkas

Pada contoh ini, citra + 50 dapat menimbulkan suatu elemen bernilai melebihi 255. Akibatnya dapat dilihat terutama pada bagian awan, yang berubah menjadi hitam.



Gambar 4.19 Efek pada operasi citra + 50

4.12 Cara Mengukur Lama Suatu Proses

Kadangkala, diperlukan untuk mendapatkan lama waktu yang diperlukan oleh suatu proses. Untuk keperluan ini, `cv2.getTickCount()` dan `cv2.getTickFrequency()` bisa digunakan. Adapun pemakaiannya seperti berikut:

```

awal = cv2.getTickCount()
#proses yang diukur dieksekusi di sini
akhir = cv2.getTickCount()
selangWaktu = {akhir - awal} / cv2.getTickCount()

```

Skrip berikut menunjukkan perbedaan kinerja antara penggunaan proses berbasis piksel dan berbasis matriks untuk melakukan inversi citra dengan cara melakukan pengukuran waktu yang diperlukan untuk proses masing-masing:

m

Berkas :ukurkinerja.py

```

#Pengukuran kinerja dua proses untuk
# melakukan inversi citra

import cv2
import numpy as np

citra = cv2.imread('lena.bmp', cv2.IMREAD_GRAYSCALE)

#Proses 1
awal = cv2.getTickCount()

jumBaris = citra.shape[0]
jumKolom = citra.shape[1]
hasil = np.zeros((jumBaris, jumKolom), np.uint8)

for brs in range(jumBaris):
    for kol in range(jumKolom):
        hasil[brs, kol] = 255 - citra[brs, kol]

akhir = cv2.getTickCount()
selangWaktu = {akhir - awal} / cv2.getTickFrequency()
print('Waktu untuk Proses 1:', selangWaktu, 'detik')

#Proses 2
awal = cv2.getTickCount()

hasil = 255 - citra

akhir = cv2.getTickCount()
selangWaktu = {akhir - awal} / cv2.getTickFrequency()
print('Waktu untuk Proses 2:', selangWaktu, 'detik')

```

Akhir berkas

Berikut adalah hasil pemanggilan skrip ini:

```
C:\LatOpenCV>python ukurkinerja.py
Waktu untuk Proses 1: 2.6151944533333333 detik
Waktu untuk Proses 2: 0.00081408 detik
```

```
C:\LatOpenCV>
```

.....
Tampak bahwa operasi dengan matriks jauh lebih cepat dibandingkan yang menggunakan pemrosesan berbasis piksel.

4.13 Konversi Citra Berwarna ke Citra Berskala Keabu-abuan dan Citra Biner

Kadangkala, setelah suatu citra berwarna diperoleh, diperlukan untuk mengonversi citra tersebut ke citra berskala keabu-abuan. Hal ini dapat dilakukan dengan mudah dengan menggunakan `cv2.cvtColor()`. Contoh penggunaannya telah dibahas di subbab 4.5.

Adapun konversi citra berwarna ke citra biner dilakukan dengan cara seperti berikut:

- 1) citra berwarna dikonversi ke citra berskala keabu-abuan;
- 2) citra berskala keabu-abuan dikonversi ke citra biner dengan menggunakan `cv2.threshold()`.

Bentuk pemakaian metode `threshold()` adalah seperti berikut:

```
cv2.threshold(citra, ambang, x, nilaiMaks, mode)
```

Argumen pertama berupa citra yang diproses. Argumen kedua menyatakan nilai ambang yang digunakan untuk menjadikan nilai 0 atau 1 terhadap piksel. Argumen keempat menyatakan nilai terbesar yang akan diberikan ke citra hasil untuk menyatakan nilai biner 1. Argumen

kelima menyatakan mode pembentukan citra biner, yang dapat berupa `cv.THRESH_BINARY` atau `cv.THRESH_BINARY_INV`. Perbedaan kedua mode ini hanya pada cara penentuan 0 dan 1 sewaktu membentuk citra biner.

Nilai balik metode ini berupa dua objek, dengan objek pertama berisi nilai ambang yang digunakan dan objek kedua berupa citra biner.

ikut memberikan contoh penggunaan `threshold()`:

Berkas : `ukurklnerji.i.11y`

```
#Konversi ke citra biner

import cv2

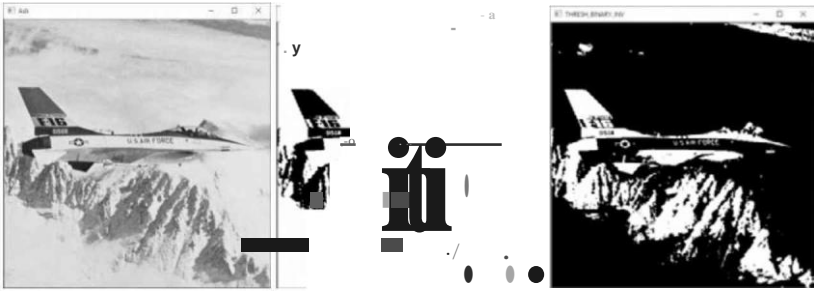
citra = cv2.imread('airplane.png', cv2.IMREAD_GRAYSCALE)
ambang, binerA = cv2.threshold(citra, 128, 255,
cv2.THRESH_BINARY)
ambang, binerB = cv2.threshold(citra, 128, 255,
cv2.THRESH_BINARY_INV)

cv2.imshow('Asli', citra)
cv2.imshow('THRESH_BINARY', binerA)
cv2.imshow('THRESH_BINARY_INV', binerB)
```

`waitKey(0)`

Akhir berkas

Pada skrip ini, `binerA` berisi citra biner yang menerapkan `cv.THRESH_BINARY`, sedangkan `binerB` berisi citra biner yang menerapkan `cv.THRESH_BINARY_INV`. Perbedaan hasil kedua citra ini diperlihatkan pada Gambar 4.12.



Gambar 4. Hasil dua cara untuk memperoleh citra biner

Metode `threshold()` dapat diatur agar algoritma yang digunakan untuk membentuk citra biner menggunakan metode Otsu. Metode ini mencari nilai ambang yang optimal. Oleh karena itu, pada penggunaan metode ini, nilai ambang pada `threshold()` cukup diisi dengan nol. Adapun pemilihan metode Otsu dilakukan pada argumen kelima dengan menambahkan:

```
+ cv2.THRESH_OTSU
```

Contoh penggunaan metode Otsu ditunjukkan berikut ini:

11!1 Berkas : ukurkinerja.11y

```
#Konversi ke citra biner menggunakan metode Otsu
```

```
import cv2
```

```
citra = cv2.imread('airplane.png', cv2.IMREAD_GRAYSCALE)
ambang, biner = cv2.threshold(citra, 128, 255,
                             cv2.THRESH_BINARY+ cv2.THRESH_OTSU)
```

```
cv2.imshow('Asli', citra)
cv2.imshow('Otsu - nilai ambang =' + str(int(ambang)),
biner)
```

```
cv2.waitKey(0)
```

```
Akhir berkas
```

Setelah perintah

```
ambang, biner = cv2.threshold(citra, 128, 255,  
cv2.THRESH_BINARY + cv2.THRESH_OTSU)
```

dikerjakan, ambang berisi nilai ambang yang digunakan untuk membentuk citra biner.

Hasil skrip `otsu.py` ditunjukkan pada Gambar 4.21.



Gambar 4.21 Hasil penerapan metode Otsu pada citra airplane.png

Catatan

Metode Otsu dijelaskan pada buku penulis yang berjudul "Teori dan Aplikasi Pengolahan Citra" (Penerbit Andi); termasuk cara mewujudkannya.

4.14 Dasar Pembuatan Gambar

OpenCV menyediakan sejumlah metode yang digunakan untuk membuat gambar berupa garis, lingkaran, persegi panjang, dan elips. Selain itu, terdapat pula metode yang ditujukan untuk meletakkan tulisan.

4.14.1 Pemhuatan Garis

Garis dibuat menggunakan `cv2.Line()`. Dasar penggunaannya seperti berikut:

```
cv2.line(citra, titik1, titik2, warna [, ketebalan])
```

Argumen pertama menyatakan citra yang hendak digambari. Argumen kedua dan ketiga berupa tupel yang berisi data *x* dan *y* yang menyatakan koordinat titik. Argumen kedua menyatakan titik awal garis dan argumen ketiga menyatakan titik akhir garis. Argumen keempat berisi tupel yang mengandung tiga elemen yang secara berturutan menyatakan warna G, R, dan B]. Argumen kelima bersifat opsional dan menentukan ketebalan garis. Dalam hal ini, ketebalan bawaan adalah 1 piksel.

Contoh berikut memberikan gambaran tentang penggunaan `line()` untuk membuat garis tegak dan garis mendatar yang melewati titik tengah gambar:



Berkas : ukurl.inerj.1.11y

```
# Pembuatan garis tegak dan mendatar
# melalui titik tengah citra

import cv2
import sys

if len(sys.argv) == 1:
    print('Masukkan nama berkas gambar')
else:
    berkas = sys.argv[1]
    citra = cv2.imread(berkas, cv2.IMREAD_UNCHANGED)
    if citra is None:
        print('Tidak dapat membaca berkas', berkas)
    else:
        jumBaris = citra.shape[0]
        jumKolom = citra.shape[1]
        xTengah = jumKolom // 2
        yTengah = jumBaris // 2
```

```
#Buat garis tegak
cv2.line(citra, (xTengah, 0),
        (yTengah, jumBaris - 1),
        [ 128, 128, 128], 5)

#Buat garis mendatar
cv2.line(citra, (0, yTengah),
        (jumKolom - 1, yTengah),
        [ 128, 128, 128], 5)

#Tampilkan citra
cv2.imshow('Hasil', citra)

cv2.waitKey(0)
```

Akhir berkas

Pada skrip ini, jumlah baris dan jumlah kolom diperoleh melalui:

```
jumBaris    citra.shape[0]
jumKolom    citra.shape[1]
```

Selanjutnya, titik tengah gambar dihitung melalui:

```
xTengah    jumKolom // 2
yTengah    jumBaris // 2
```

Penggunaan // dimaksudkan agar hasil pembagian berupa bilangan bulat.

Garis tegak dibentuk melalui:

```
cv2.line(citra, (xTengah, 0),
        (yTengah, jumBaris - 1),
        [ 128, 128, 128], 5)
```

Garis dibuat dari baris 0 hingga baris terakhir dengan warna abu-abu (yang ditentukan oleh [128, 128, 128]) dan dengan tebal 5 piksel.

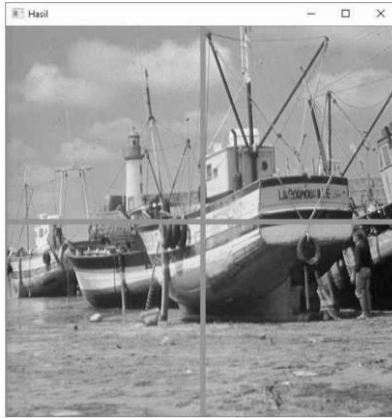
Garis mendatar dibentuk melalui:

```
cv2.line(citra, (0, yTengah),
        (jumKolom - 1, yTengah),
        [ 128, 128, 128], 5)
```

Garis dibuat dari kolom 0 hingga kolom terakhir dengan warna abu-abu (yang ditentukan oleh [128, 128, 128]) dan dengan tebal 5 piksel.

Gambar 4.22 menunjukkan hasil pemanggilan:

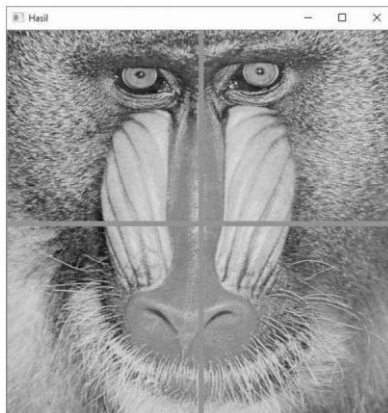
```
python garispusat.py boat.png
```



Gambar 4.22:J Garis tegak danmendatar yang melalui titik tengah gambar pada citra boat.png

Gambar 4.22 menunjukkan hasil pemanggilan:

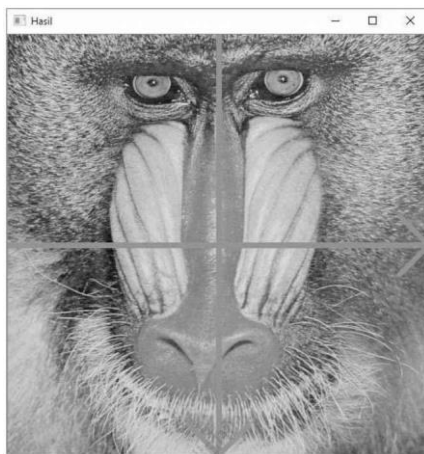
```
python garispusat.py baboon.png
```



Gambar 4.23:J Garis tegak danmendatar yang melalui titik tengah gambar pada citra baboon.png

Selain `line ()`, terdapat `arrowedLine ()`. Fungsi ini digunakan untuk membuat garis dengan ujung akhir dilengkapi mata anak panah. Gambar 4.24 menunjukkan hasil kalau `arrowedLine ()` digunakan untuk menggantikan `line ()` pada pemanggilan:

```
python garispusat.py baboon.png
```



Gambar 4.14 Garis dengan mata anak panah

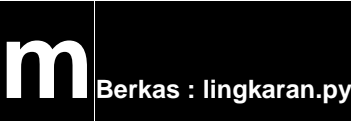
4.14.2 Pemhuatan Lingkaran

Lingkaran dibuat dengan menggunakan `cv2.Circle ()`. Dasar penggunaannya seperti berikut:

```
cv2.circle(citra, titikl, radius, warna[, ketebalan])
```

Argumen pertama menyatakan citra yang hendak digambari. Argumen kedua menyatakan koordinat titik pusat lingkaran. Argumen ketiga berupa jari-jari lingkaran. Argumen keempat berisi tupel yang mengandung tiga elemen yang secara berturutan menyatakan warna G, R, dan B]. Argumen kelima bersifat opsional dan menentukan ketebalan garis. Dalam hal ini, ketebalan bawaan adalah 1 piksel. Apabila argumen ini diisi dengan `cv2.FILLED`, bagian dalam lingkaran akan diwarnai.

Contoh berikut memberikan gambaran tentang penggunaan `circle()` untuk membuat lingkaran dengan titik pusat di tengah



ar 25:

```
# Pembuatan lingkaran di tengah gambar

import cv2
import sys

if len(sys.argv) == 1:
    print('Masukkan nama berkas gambar')
else:
    berkas = sys.argv[1]
    citra = cv2.imread(berkas, cv2.IMREAD_UNCHANGED)
    if citra is None:
        print('Tidak dapat membaca berkas', berkas)
    else:
        jumBaris = citra.shape[0]
        jumKolom = citra.shape[1]
        xTengah = jumKolom // 2
        yTengah = jumBaris // 2

        # Buat lingkaran
        cv2.circle(citra, (xTengah, yTengah), 100,
                   [255, 255, 255], 10)

        # Tampilkan citra
        cv2.imshow('Hasil', citra)

        cv2.waitKey(0)
```

Akhir berkas

Pada skrip ini, jumlah baris dan jumlah kolom diperoleh melalui:

```
jumBaris = citra.shape[0]
jumKolom = citra.shape[1]
```

Selanjutnya, titik tengah gambar dihitung melalui:

```
xTengah = jumKolom // 2
yTengah = jumBaris // 2
```

Penggunaan // dimaksudkan agar hasil pembagian berupa bilangan bulat.

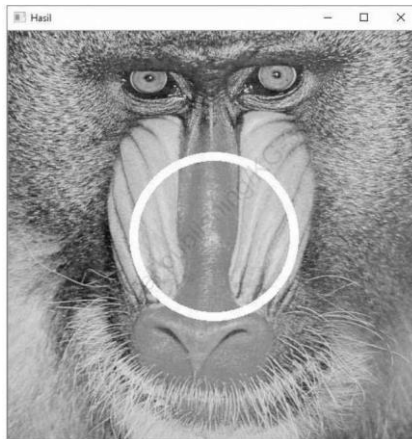
Lingkaran dibentuk melalui:

```
cv2.circle (citra, (xTengah, yTengah), 100,  
            [255, 255, 255], 10)
```

Lingkaran dibuat pada titik di tengah-tengah dengan warna abu-abu (yang ditentukan oleh [128, 128, 128]) dan dengan tebal 10 piksel.

Gambar 4.25 menunjukkan hasil pemanggilan:

```
python lingkaran.py baboon.png
```



Gambar 4d15 Lingkaran di tengah-tengah gambar

4.14.3 Pemhuatan Elips

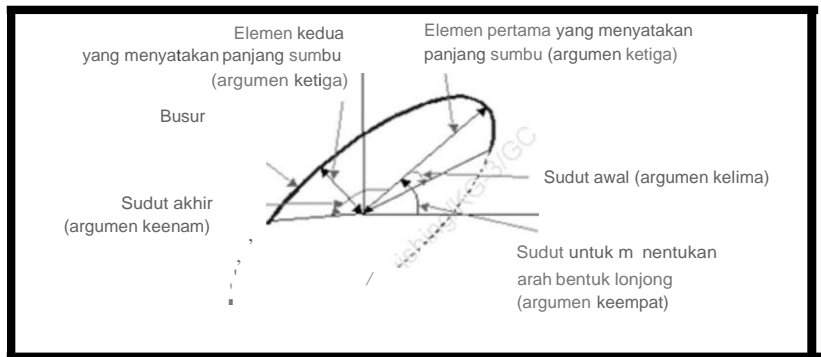
Bentuk lonjong dibuat dengan menggunakan `cv2.ellipse()`. Dasar penggunaannya seperti berikut:

```
cv2.ellipse(citra, titikPusat, sumbu, sudut,  
            sudutAwal, sudutAkhir, warna [, ketebalan])
```

Argumen pertama menyatakan citra yang hendak digambari. Argumen kedua berupa tupel yang menyatakan titik pusat. Argumen ketiga

berupa tupel yang berisi data separuh panjang sumbu x dan y. Argumen keempat menyatakan sudut untuk memutar elips dalam derajat. Argumen kelima dan keenam menyatakan sudut awal dan sudut akhir untuk membuat busur. Argumen ketujuh berisi tupel yang mengandung tiga elemen yang secara berturut-turut menyatakan warna G, R, dan B. Argumen kedelapan bersifat opsional dan menentukan ketebalan garis. Dalam hal ini, ketebalan bawaan adalah 1 piksel.

Gambar 4.26 menunjukkan hubungan antara bentuk lonjong yang dibentuk dan argumen-argumen pada `ellipse()`.



Gambar 4.26 Penentuan bentuk lonjong (Diadaptasi dari https://docs.opencv.org/4.4/modules/core/doc/drawing_functions.html)

Contoh berikut memberikan gambaran tentang penggunaan `ellipse()`, `line()`, dan `circle()` untuk membuat suatu bentuk



Berkas : `linek- .m.11y`

```
# Pembuatan elips

import cv2
import numpy as np

# Buat citra berwarna hitam
```

```

citra = np.zeros({256, 256, 3}, np.uint8)

# Buat elips
cv2.ellipse {citra, (128, 128), (100, 50), 0, 0, 360,
              (255, 255, 255), 5)

# Buat busur
cv2.ellipse {citra, (128, 230), (80, 50), 0, -180, 0,
              (255, 255, 255), 5)

# Buat mata
cv2.circle {citra, (128,128), 10, (0, 0, 128), 3)

# Buat belalai
cv2.line {citra, (128, 80), (128, 40), (255, 255, 255), 3)
cv2. circle {citra, (128, 45), 5, (255, 255, 255), 3)

# Tampilkan citra
cv2.imshow {'Hasil', citra)

waitKey(0)

```

Akhir berkas

Scrip ini melibatkan modul `numpy` untuk membentuk larik citra. Itulah sebabnya, terdapat perintah:

```
import numpy as np
```

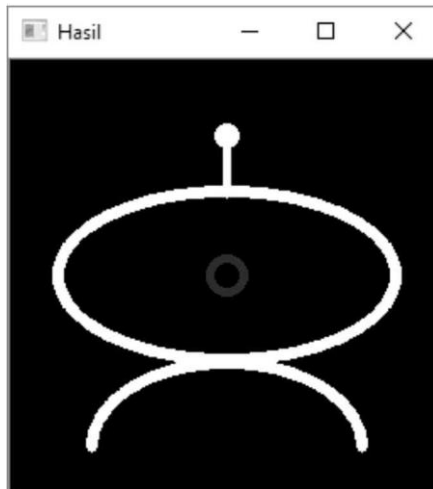
Penggunaan `as np` dimaksudkan agar modul `numpy` dapat diakses melalui kata `np`.

Pembentukan citra dilakukan melalui:

```
citra = np.zeros({256, 256, 3}, np.uint8)
```

Perintah ini membuat citra berwarna berukuran 256 x 256 dengan warna hitam mengingat semua elemen bernilai 0. Di citra yang dibentuk oleh perintah inilah `line ()`, `circle ()`, dan `ellipse ()` diterapkan.

Hasil skrip ini diperlihatkan pada Gambar 4.27.



Gambar 4.f/7 Hasil pembuatan bentuk lonjong

4.14.4 Pemhuatan Persegipanjang

Persegipanjang dibuat dengan menggunakan `cv2.rectangle()`. Dasar penggunaannya seperti berikut:

```
cv2.rectangle(citra, titik1, titik2, warna[, ketebalan])
```

Argumen pertama menyatakan citra yang hendak digambari. Argumen kedua dan ketiga berupa tupel yang berisi data x dan y yang menyatakan koordinat titik. Argumen kedua menyatakan titik pojok kiri-atas persegipanjang dan argumen ketiga menyatakan titik pojok kanan-bawah persegipanjang. Argumen keempat berisi tupel yang mengandung tiga elemen yang secara berturutan menyatakan warna G, R, dan B]. Argumen kelima bersifat opsional dan menentukan ketebalan garis. Dalam hal ini, ketebalan bawaan adalah 1 piksel. Apabila argumen ini diisi dengan `cv2.FILLED`, bagian dalam persegipanjang akan diwarnai.

Contoh berikut memberikan gambaran tentang penggunaan `rectangle()` untuk membuat dua persegi panjang:



Berkas :lin1:ki1r,m.11y

```
# Pembuatan persegipanjang

import cv2
import numpy as np

# Buat citra berwarna putih
citra = 255 * np.ones((256, 256, 3), np.uint8)

# Buat dua persegipanjang
cv2.rectangle (citra, (10, 10), (250,128), (0, 0, 0),
               cv2.FILLED)
cv2.rectangle(citra, (100, 80), (236,246),
               (0, 0, 255), 10)

# Tampilkan citra
cv2.imshow('Hasil', citra)

waitKey(0)
```

Akhir berkas

Skrrip ini melibatkan modul numpy untuk membentuk larik citra. Itulah sebabnya, terdapat perintah:

```
import numpy as np
```

Penggunaan `as np` dimaksudkan agar modul numpy dapat diakses melalui kata `np`.

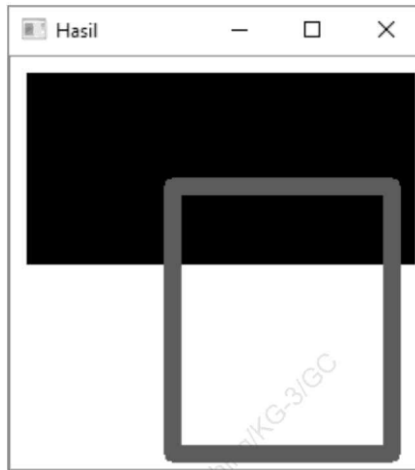
Pembentukan citra dilakukan melalui:

```
citra = 255 * np.ones((256, 256, 3), np.uint8)
```

Perintah ini membuat citra berwarna berukuran 256 x 256 warna hitam mengingat semua elemen bernilai 255. Di citra yang dibentuk oleh perintah inilah `rectangle()` diterapkan. Pada perintah yang pertama, `cv2.FILLED` disertakan dengan maksud agar bagian dalam

persegi panjang diwarnai. Pada perintah yang kedua, argumen terakhir bernilai 10, yang dimaksudkan agar ketebalan garis pada persegi panjang sebesar 10 piksel.

Hasil skrip ini diperlihatkan pada Gambar 4.28.



Gambar 4.28 Hasil pembuatan dua persegi panjang

4.14.5 Pemhuatan Poligon

Poligon dibuat dengan menggunakan `cv2.Line()`. Dasar penggunaannya seperti berikut:

```
cv2.polylines(citra, larikTitik, tertutup, warna [, ketebalan])
```

Argumen pertama menyatakan citra yang hendak digambari. Argumen kedua berupa larik `numpy`. Argumen ketiga berupa `True` atau `False` dengan nilai `True` menyatakan poligon tertutup dengan titik pertama dan titik terakhir dihubungkan dan `False` menyatakan bahwa titik pertama dan titik terakhir tidak dihubungkan. Argumen keempat berisi tupel yang mengandung tiga elemen yang secara berturut-turut menyatakan warna G, R, dan B]. Argumen kelima bersifat opsional dan

menentukan ketebalan garis. Dalam hal ini, ketebalan bawaan adalah 1 piksel.

Contoh berikut memberikan gambaran tentang penggunaan



ones() untuk membuat suatu poligon:

Berkas : lin1:h:ran.11y

```
# Pembuatan poligon

import cv2
import numpy as np

# Buat citra berwarna putih
citra = 255 * np.ones((256, 256, 3), np.uint8)

# Buat poligon
titik = np.array([[10, 128], [200, 10],
                  [180, 120], [240, 240]], np.int32)
cv2.polylines(citra, [titik], True, (0, 0, 0), 3)

# Tampilkan citra
cv2.imshow('Hasil', citra)
```

```
waitKey(0) 1>
'''
```

Akhir berkas

Skip ini melibatkan modul `numpy` untuk membentuk larik citra. Itulah sebabnya, terdapat perintah:

```
import numpy as np
```

Penggunaan `as np` dimaksudkan agar modul `numpy` dapat diakses melalui kata `np`.

Pembentukan citra dilakukan melalui:

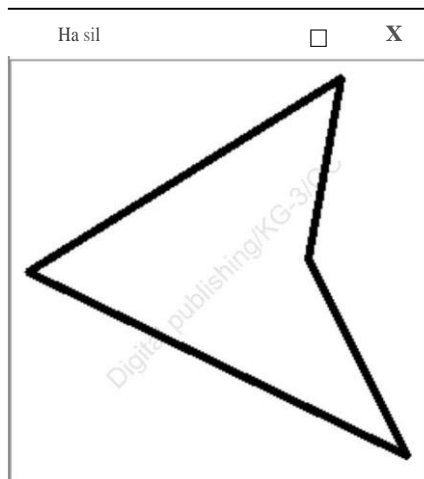
```
citra = 255 * np.ones((256, 256, 3), np.uint8)
```


Perintah ini membuat citra berwarna berukuran 256 x 256 warna hitam mengingat semua elemen bernilai 255. Di citra yang dibentuk oleh perintah inilah `polylines ()` diterapkan. Perintahnya berupa:

```
cv2.polylines (citra, [titik], True, (0, 0, 0), 3)
```

Nilai `True` menyatakan bahwa titik pertama dan titik terakhir akan dihubungkan untuk membuat poligon. Nilai `(0, 0, 0)` menyatakan warna hitam dan `3` menyatakan ketebalan garis.

Hasil skrip ini diperlihatkan pada Gambar 4.29.



Gambar 4.29 Poligon

Untuk membuat poligon yang bagian dalamnya diwarnai, `fillPoly ()` perlu digunakan. Contoh penggunaannya dapat dilihat pada skrip berikut:



```
#Pernbuatan poligon dengan bagian dalamnya diwarnai
```

```
import cv2
import numpy as np
```

```
# Buat citra berwarna putih
citra = 255 * np.ones((256, 256, 3), np.uint8)

# Buat poligon
titik = np.array([ [10, 128], [200, 10],
                   [180, 120], [240, 240]], np.int32)
cv2.fillPoly(citra, [titik], (0, 0, 0))

# Tampilkan citra
cv2.imshow('Hasil', citra)

cv2.waitKey(0)
```

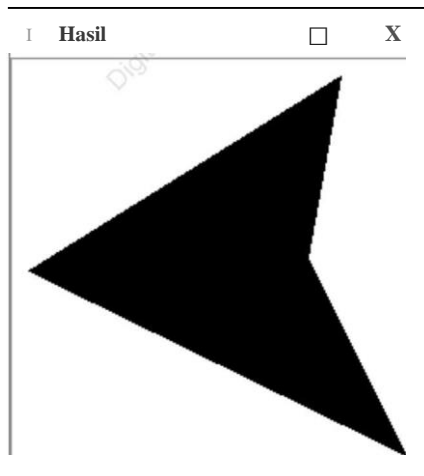
Akhir berkas

Perhatikan bahwa pemanggilan `fillPoly()` adalah seperti berikut:

```
cv2.fillPoly(citra, [titik], (0, 0, 0))
```

Jadi, yang perlu disertakan hanya citra yang diproses, data titik penyusun poligon dan warna yang digunakan.

Hasilnya diperlihatkan pada Gambar 4.30.



Gambar 4.90 Bagian dalam poligon diwarnai

4.14.6 Penulisan Teks

Teks dapat dituliskan ke citra dengan menggunakan `put Text ()`.

Bentuk dasar pemakaiannya seperti berikut:

```
cv2.putText(citra, teks, posisi, fontFace, fontScale, warna [, ketebalan])
```

Argumen pertama menyatakan citra yang hendak digambari. Argumen kedua berupa teks yang akan dituliskan. Argumen ketiga berupa posisi pokok kiri-bawah teks yang dinyatakan dalam tupel. Argumen keempat jenis font. Argumen kelima menyatakan ukuran font. Argumen keenam berisi tupel yang mengandung tiga elemen yang secara berturutan menyatakan warna G, R, dan B]. Argumen ketujuh bersifat opsional dan menentukan ketebalan garis. Dalam hal ini, ketebalan bawaan adalah 1 piksel.

Contoh berikut memberikan gambaran tentang penggunaan



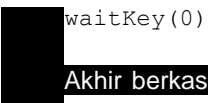
```
# Contoh putText()

import cv2

citra = cv2.imread('baboon.png')

# Buat tulisan
font= cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(citra, 'Baboon', (10, 500),
            font, 4, (0, 0, 0), 10)

# Tampilkan citra
cv2.imshow('Hasil', citra)
```



Hasil skrip ini diperlihatkan pada Gambar 4.31.



Gambar 4.31 Garis dengan mata anak panah

Jenis *font* yang digunakan dapat berupa salah satu dari berikut:

- FONT_HERSHEY_COMPLEX
- FONT_HERSHEY_COMPLEX_SMALL
- FONT_HERSHEY_DUPLEX
- FONT_HERSHEY_PLAIN
- FONT_HERSHEY_SCRIPT_COMPLEX
- FONT_HERSHEY_SCRIPT_SIMPLEX
- FONT_HERSHEY_SIMPLEX
- FONT_HERSHEY_TRIPLEX

Untuk membentuk tulisan miring tambahkan jenis font tersebut dengan konstanta berikut:

FONT_ITALIC

Skrip berikut memberikan contoh tulisan yang dimiringkan:

```
# Contoh putText()

import cv2

citra = cv2.imread('baboon.png')

# Buat tulisan
font= cv2.FONT_HERSHEY_COMPLEX+ \
      cv2.FONT_ITALIC
cv2.putText(citra, 'Baboon', (10, 500),
           font, 4, (0, 0, 0), 5)

# Tampilkan citra
cv2.imshow('Hasil', citra)
```

```
waitKey(0)
```

Akhir berkas

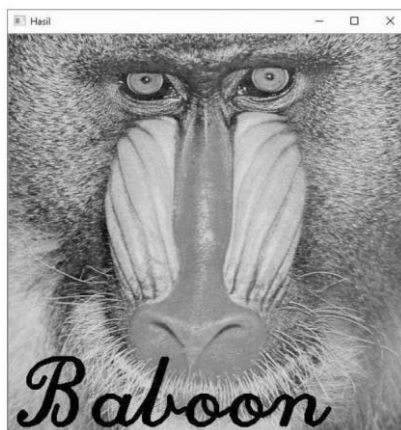
Perhatikan penentuan font melalui:

```
font= cv2.FONT_HERSHEY_COMPLEX+ cv2.FONT_ITALIC
```

Jenis font yang dipilih adalah `cv2.FONT_HERSHEY_COMPLEX`.

Adapun `cv2.FONT_ITALIC` **digunakan untuk memiringkan tulisan.**

Gambar 4.32 memperlihatkan pemanggilan skrip tulisan2.py.



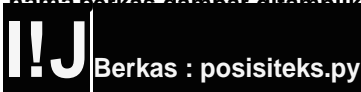
Gambar 4.32 Hasil Tulisan miring

Kadangkala, sebelum menampilkan suatu teks pada citra diperlukan informasi mengenai ukuran tempat yang digunakan oleh suatu teks, Untuk itu, fungsi yang digunakan adalah `getTextSize()`. Bentuk pemakaiannya seperti berikut:

`cv2.getTextSize(teks, jenisFont, ukuranFont, ketebalan)`

Jadi, yang perlu disebutkan adalah teks yang akan ditampilkan, jenis font, ukuran font, dan ketebalan teks. Nilai baliknya ada dua, dengan yang pertama berupa tupel yang berisi lebar dan tinggi teks dan yang kedua berupa nilai tinggi garis dasar teks terhadap titik terendah teks.

Contoh berikut memberikan gambaran penggunaan `getTextSize()` untuk kepentingan mengatur tulisan di tengah gambar. Dalam hal ini, nama berkas gambar ditampilkan di tengah-tengah gambar.



```
# Penulisan teks di tengah citra

import cv2
import sys

if len(sys.argv) == 1:
    print('Masukkan nama berkas gambar')
else:
    berkas = sys.argv[1]
    citra = cv2.imread(berkas, cv2.IMREAD_UNCHANGED)
    if citra is None:
        print('Tidak dapat membaca berkas', berkas)
    else:
        jumBaris = citra.shape[0]
        jumKolom = citra.shape[1]

        FONT = cv2.FONT_HERSHEY_DUPLEX
        UKURAN = 2
        KETEBALAN = 1

        info1, info2 = cv2.getTextSize(berkas,
                                       FONT, UKURAN, KETEBALAN)

        posisiX = (jumBaris - info1[0]) // 2
```

```

posisiY = (jumBaris - infol[1]) // 2 + infol[1]

print(jumBaris, jumKolom, posisiX, posisiY)
print(infol, info2)

cv2.putText(citra, berkas, (posisiX, posisiY),
            FONT, UKURAN, (255, 255, 255), KETEBALAN)

# Tampilkan citra
cv2.imshow('Hasil', citra)

cv2.waitKey(0)

```

Akhir berkas

Informasi mengenai tinggi dan lebar teks diperoleh melalui:

```

infol, info2 = cv2.getTextSize(berkas,
                               FONT, UKURAN, KETEBALAN)

```

Nilai balik yang dihasilkan berupa infol dan info2. Dalam hal ini, infol berupa tupel dengan dua elemen. Elemen pertama menyatakan panjang teks dalam piksel dan elemen kedua berupa tinggi teks dalam piksel. Adapun info2 berisi jarak antara garis dasar teks dan piksel terendah yang menyusun teks. Nilai balik kedua ini tidak dipakai.

Selanjutnya, pojok kiri-bawah teks diperoleh melalui:

```

posisiX = (jumBaris - infol[0]) // 2
posisiY = (jumBaris - infol[1]) // 2 + infol[1]

```

Kedua nilai inilah yang digunakan pada:

```

cv2.putText(citra, berkas, (posisiX, posisiY),
            FONT, UKURAN, (255, 255, 255), KETEBALAN)

```

Gambar 4.33 menunjukkan hasil pemanggilan:

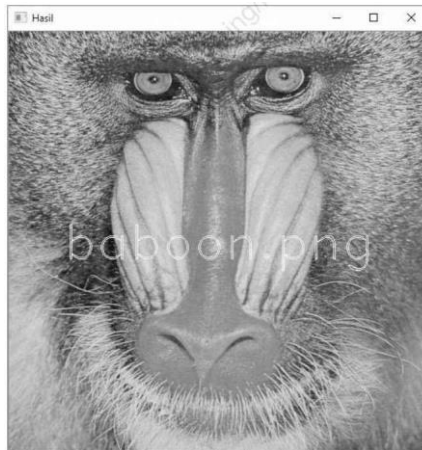
```
python posisiteks.py goldhill.png
```



Gambar 4.33 Nama berkas goldhill.png ditampilkan di tengah gambar

Adapun Gambar 4.34 menunjukkan hasil pemanggilan:

```
python posisiteks.py baboon.png
```



Gambar 4.35 Nama berkas baboon.png ditampilkan di tengah gambar

4.15 Penyimpanan Citra

Citra hasil pemrosesan dapat disimpan dengan menggunakan `cv2.imwrite()`. Bentuk pemakaiannya seperti berikut:

`cv2.imwrite(namaBerkasCitra, citra)`

Argumen pertama berupa nama berkas gambar dan argumen kedua adalah objek citra yang hendak disimpan ke berkas.

Contoh berikut menunjukkan cara menyimpan gambar hasil pengolahan



```
# Penyimpanan citra ke berkas hasil.png
```

```
import cv2
import sys
```

```
citra = cv2.imread('goldhill.png', 0)
```

```
hasil = 255 - citra
cv2.imwrite('hasil.png', hasil)
```

```
print('Citra telah disimpan di hasil.png')
```

Akhir berkas

Citra pada `hasil`, yang merupakan inversi citra pada citra disimpan ke `hasil.png` melalui:

```
cv2.imwrite('hasil.png', hasil)
```

Berikut adalah hasil pemanggilan skrip ini:

```
C:\LatOpenCV>python simpan.py
Citra telah disimpan di hasil.png
```

```
C:\LatOpenCV>
```

Adapun Gambar 4.36 menunjukkan citra hasil yang bisa dibuka dengan editor gambar apa saja.



Gambar 4.36 Gambar yang disimpan melalui skrip `simpan.py`

4.16 Pemisahan dan Penggabungan Kanal Citra

Citra berwarna memiliki tiga kanal yang masing-masing berupa matriks. Matriks pertama berisi semua komponen B. Matriks kedua berisi semua komponen G. Matriks ketiga berisi semua komponen R.

Adakalanya, diperlukan untuk memperoleh matriks satu kanal saja. Nah, untuk kepentingan seperti ini, `cv2.split()` bisa digunakan. Contoh berikut akan pada skrip berikut:



Berkas : `!im11.m.11y`

```
#Pemisahan kanal citra berwarna

import cv2

citra = cv2.imread('peppers.png')

biru, hijau, merah = cv2.split(citra)

cv2.imshow('Kanal biru', biru)
cv2.imshow('Kanal hijau', hijau)
```

```
cv2.imshow('Kanal merah', merah)
```

Akhir berkas

Perintah berikut digunakan untuk memisah tiga kanal pada citra `peppers.png`:

```
biru, hijau, merah = cv2.split(citra)
```

Nah, `biru`, `hijau`, dan `merah` berisi kanal B, G, dan R secara berturutan. Selanjutnya, kanal masing-masing ditampilkan melalui:

```
cv2.imshow('Kanal biru', biru)
cv2.imshow('Kanal hijau', hijau)
cv2.imshow('Kanal merah', merah)
```

Hasilnya diperlihatkan pada Gambar 4.37.



Gambar 4.37 Kanal B, G, dan R dalam bentuk citra berskala keahuan

Selain menggunakan `cv2.split()`, notasi seperti `citra[:, :, 1]` dapat digunakan untuk memperoleh kanal 1 (kanal G). Skrip berikut merupakan alternatif yang menggunakan notasi ini untuk memisahkan

ada citra berwarna:

m

Berkas: `im11.m.11y`

```
#Pemisahan kanal citra berwarna
# Versi 2
```

```
import cv2

citra = cv2.imread('peppers.png')

biru = citra[:, :, 0]
hijau = citra[:, :, 1]
merah = citra[:, :, 2]

cv2.imshow('Kanal biru', biru)
cv2.imshow('Kanal hijau', hijau)
cv2.imshow('Kanal merah', merah)
```

Akhir berkas

Untuk menggabungkan tiga kanal, `cv2.merge()` bisa digunakan.

Contoh berikut menunjukkan cara menampilkan kanal warna biru, hijau,

dan merah, sebenarnya.

11!1 Berkas : gabkanal.py

```
# Pemisahan kanal citra berwarna

import cv2
import numpy as np

citra = cv2.imread('peppers.png')

# Buat citra berwarna hitam
jumBaris = citra.shape[0]
jumKolom = citra.shape[1]

# Matriks berisi nol
nol = np.zeros((jumBaris, jumKolom), np.uint8)

biru, hijau, merah = cv2.split(citra)
citraB = cv2.merge((biru, nol, nol))
citraG = cv2.merge((nol, hijau, nol))
citraR = cv2.merge((nol, nol, merah))

cv2.imshow('Kanal biru', citraB)
cv2.imshow('Kanal hijau', citraG)
cv2.imshow('Kanal merah', citraR)
```

Akhir berkas

Pada skrip ini, perintah berikut digunakan membentuk matriks berukuran `jumlahBaris x jumlahKolom` yang semua elemennya bernilai 0:

```
nol = np.zeros((jumlahBaris, jumlahKolom), np.uint8)
```

Perintah berikut digunakan untuk membentuk citra berwarna dengan menggabungkan biru dan dua nol:

```
citraB = cv2.merge((biru, nol, nol))
```

Dengan cara seperti ini, hanya komponen warna biru yang muncul pada `citraB`. Perhatikan pula untuk membentuk citra berwarna yang hanya mengandung komponen hijau atau merah.

4.17 Penggabungan Citra

Penggabungan dua citra dapat dilakukan dengan menggunakan `cv2.add()`. Syaratnya, citra-citra yang hendak digabungkan harus mempunyai ukuran yang sama.

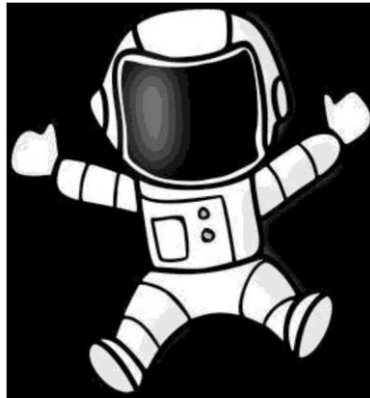
Untuk mempraktikkan ini, berkas `Spaceman-colour.png` dapat diunduh melalui:

<https://openclipart.org/detail/296491/spaceman-colour>

Setelah itu, unduh berkas yang berukuran paling kecil dengan mengeklik

SMALL IMAGE (PNG)

. Kemudian, taruhlah berkas terunduh ke `C:\LatOpenCV`. Gambar 4.38 menunjukkan gambarnya.



Gambar4.38 Spaceman-colour

Sebelum mempraktikkan, penerapan `cv2.add()`, marilah sejenak melihat berkas citra ini dengan memberikan perintah berikut:

```
.....
: >>> citra = cv2.imread( 'Spaceman-colour.png', -1) 4)
  >>> citra.shape
  (300, 281, 4)
  >>>
```

Perhatikan bahwa nilai terakhir dalam tupel yang dihasilkan oleh `shape` berupa 4, bukan 3 seperti yang biasa diberikan oleh citra berwarna. Angka 4 menyatakan bahwa terdapat 4 kanal. Tentu saja, 3 kanal menyatakan komponen B, G, dan R. Lalu, satu komponen lagi digunakan untuk apa? Komponen tersebut dinamakan komponen alfa dan digunakan untuk menyatakan sifat transparansi. Nilai 0 menyatakan transparan sepenuhnya.

Nah, untuk sementara kita abaikan kanal alfa ini. Sekarang, fokus kita adalah pada cara menambahkan citra `Spaceman-colour.png` pada citra `goldhill.png`. Perlu diketahui, ukuran citra `goldhill.png` adalah 512x512, sedangkan hasil di depan menunjukkan bahwa ukuran `Spaceman-colour.png` adalah 300 x 281. Oleh karena itu,

diperlukan cara untuk membuat matriks berukuran 512x512 yang berisi citra tersebut. Skrip berikut menunjukkan perwujudannya:

11!1

Berkas : im11an.11y

```
# Pembuatan citra berukuran 512 x 512 yang mengandung
# citra Spaceman-colour

import cv2
import numpy as np

# Buat citra berwarna hitam
citra = np.zeros((512, 512, 3), np.uint8)

spaceman    cv2.imread('Spaceman-colour.png', -1)
jumBaris    spaceman.shape[0]
jumKolom    spaceman.shape[1]

citra[0:jumBaris, 0:jumKolom, :] = spaceman[:, :, 0:3]
cv2.imshow('Hasil', citra)
```

waitKey(0)

Akhir berkas

Pada skrip ini, perintah berikut digunakan untuk membuat citra berwarna hitam berukuran 512 x 512:

```
citra = np.zeros((512, 512, 3), np.uint8)
```

Adapun perintah berikut digunakan untuk membaca berkas Spaceman-colour.png apabila ada:

```
spaceman= cv2.imread('Spaceman-colour.png', -1)
```

Dengan demikian, informasi kanal alfa tersedia.

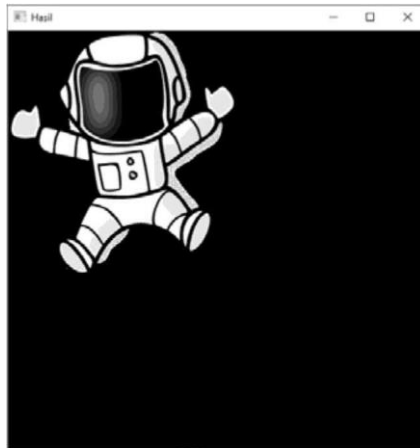
Jumlah baris dan jumlah kolom pada citra tersebut diperoleh melalui:

```
jumBaris    spaceman.shape[0]
jumKolom    spaceman.shape[1]
```

Selanjutnya, pernyataan berikut dipakai untuk menyalin citra Spaceman-colour tanpa kanal alfa ke posisi pojok kiri-atas objek citra:

```
citra[0:jumBaris, 0:jumKolom, :] =spaceman[:, :, 0:3]
```

Gambar 4.28 menunjukkan hasil di ci tra.



Gambar 4.98 Spaceman-colour berukuran 511!! x 511!!

Nah, sekarang dengan bekal tersebut, kita bisa mengembangkan untuk menggabungkan citra pada Gambar 4.38 dengan citra goldhill.png. Perwujudannya adalah seperti berikut:

11!1 Berkas : sim11.m.py

```
# Penggabungan citra Spaceman-colour dengan goldhill.png
```

```
import cv2
import numpy as np
```

```
#Buat citra berwarna hitam
citra = np.zeros((512, 512, 3), np.uint8)
```

```
spaceman    cv2.imread('Spaceman-colour.png', -1)
jumBaris    spaceman.shape[0]
jumKolom    spaceman.shape[1]
```



```

citra[0:jumBaris, 0:jumKolom, :] =spaceman[:, :, 0:3]

goldhill = cv2.imread('goldhill.png')
hasil = cv2.add(goldhill, citra)

cv2.imshow('Hasil', hasil

```

Akhir berkas

Tambahan utama pada skrip ini terhadap skrip sebelumnya adalah pada:

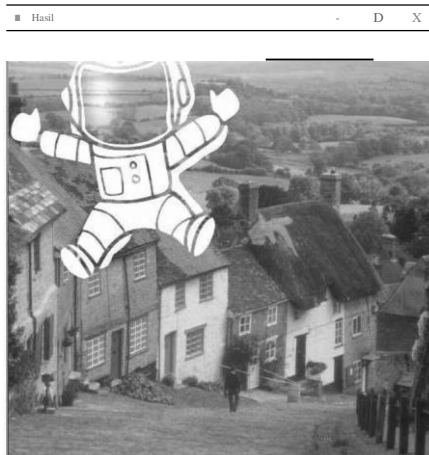
```

goldhill = cv2.imread('goldhill.png')
hasil = cv2.add(goldhill, citra)

```

Pernyataan pertama digunakan untuk membaca goldhill.png dan dicatat di goldhill. Lalu, citra goldhill dan citra digabungkan melalui add () dan hasilnya diletakkan di hasil.

Gambar 4.39 menunjukkan hasilnya.



Gambar 4.99 Spaceman-colour digabungkan ke goldhill

Hal yang perlu diperhatikan pada hasil di atas adalah bahwa helm bersifat transparan. Untuk mengatasi kekurangan ini, cara yang lebih



gunakan pada skrip berikut.

Berkas : simr,.ir..11y

```
# Penggabungan citra Spaceman-colour dengan goldhill.png
# menggunakan komponen alfa
import cv2
import numpy as np

# Buat citra berwarna hitam
citra = np.zeros({512, 512, 3}, np.uint8)
alfa = np.zeros({512, 512}, np.uint8)

spaceman = cv2.imread('Spaceman-colour.png', -1)
jumBaris = spaceman.shape[0]
jumKolom = spaceman.shape[1]

citra[0:jumBaris, 0:jumKolom, :] = spaceman[:, :, 0:3]
alfa[0:jumBaris, 0:jumKolom] = spaceman[:, :, 3]
cadar = cv2.merge({alfa, alfa, alfa})
kebalikan = cv2.bitwise_not(cadar)

cv2.imshow('Cadar', cadar)
cv2.imshow('Kebalikan', kebalikan)

goldhill = cv2.imread('goldhill.png')

hasil and= cv2.bitwise_and(goldhill, kebalikan)
cv2.imshow('Hasil and', hasil and)
print(hasil and.shape)

hasil add= cv2.add(hasil and, citra)
cv2.imshow('Hasil add', hasil add)

waitKey(0)
```

Akhir berkas

Cara yang digunakan pada skrip ini adalah dengan melibatkan komponen alfa. Pertama-tama, objek alfa dibuat melalui:

```
alfa = np.zeros({512, 512}, np.uint8)
```

Dalam hal ini, `alfa` adalah citra beraras keabu-abuan berwarna hitam berukuran 512 x 512. Selanjutnya, komponen alfa milik `Spaceman-colour` disalin ke `alfa` melalui:

```
alfa[0:jumBaris, 0:jumKolom] = spaceman[:, :, 3]
```

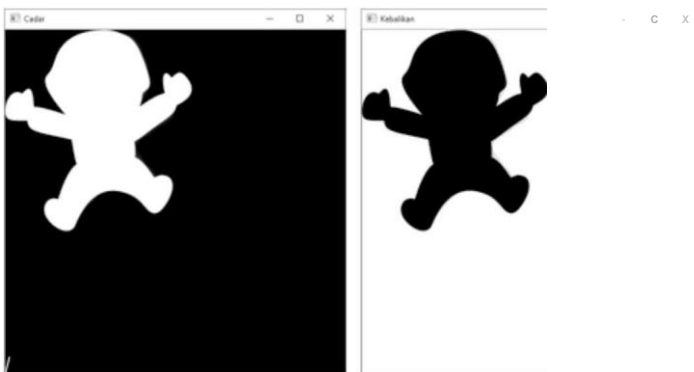
Lalu, `cadar` dengan tiga komponen B, G, R dibentuk melalui:

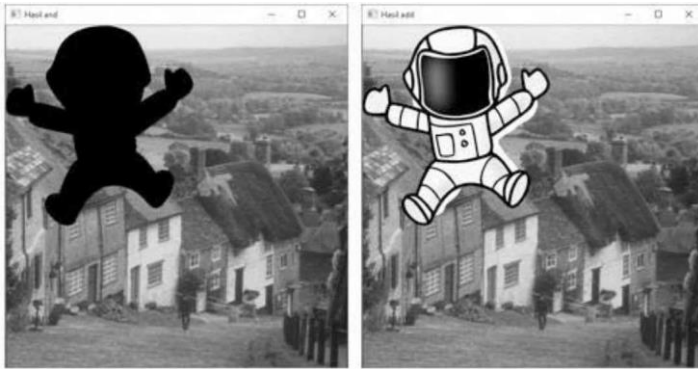
```
cadar = cv2.merge((alfa, alfa, alfa))
```

`Cadar` ini kemudian dibalik supaya warna hitam diubah menjadi putih dan warna putih diubah menjadi hitam. Perintahnya berupa:

```
kebalikan = cv2.bitwise_not(cadar)
```

Dengan begitu, warna yang berisi bagian si angkasawan berwarna hitam dan yang lain berwarna putih. Lalu, citra `kebalikan` ini dikenakan ke citra `goldhill` melalui `bitwise and()`. Dengan cara seperti ini diperoleh citra yang bagian si angkasawan akan berwarna hitam, sedangkan bagian lain tidak berubah. Terakhir, citra yang berisi si angkasawan ditambahkan ke `goldhill` melalui `add()`. Hasil secara sekuensial yang menyatakan urutan pembentukan citra di setiap proses ditunjukkan pada Gambar 4.40.





Gambar 4.40 Proses penggabungan gambar dengan cadar

Contoh lain penggabungan dua warna ditunjukkan pada skrip berikut:

11!1

Berkas : gabung2.py

```
# Penggabungan citra lena dan baboon
```

```
import cv2
```

```
lena = cv2.imread('lena.png')
baboon= cv2.imread('baboon.png')
hasil = cv2.add(baboon, lena)
```

```
cv2.imshow('Hasil', hasil)
```

```
waitKey(0)
```

Akhir berkas

Contoh ini menggabungkan dua citra berwarna, baboon dan lena. Hasilnya diperlihatkan pada Gambar 4.41. Namun, hasilnya lebih cenderung memutih karena penjumlahan kedua piksel cenderung menghasilkan nilai yang besar mendekati 255. Akibatnya, warna putih mendominasi.



Gambar 4.41 Spaceman-colour digabungkan ke goldhill

Proses pada contoh di atas hanya menggunakan penjumlahan biasa. Alternatif lain untuk menggabungkan dua gambar adalah dengan menggunakan rumus berikut yang dapat mengatasi timbulnya warna memutih pada contoh di depan:

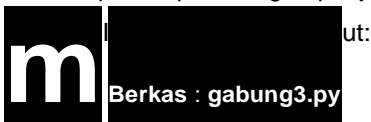
$$\text{keluaran} = \text{alfa} \times \text{citra1} + \text{beta} \times \text{citra2} + \text{gamma}$$

$$\text{beta} = 1 - \text{alfa}$$

Hal ini dapat dilaksanakan dengan menggunakan `cv2.addWeighted()`. Formatnya seperti berikut:

`cv2.addWeighted(citra1, a/fa, citra2, beta, gamma)`

Contoh penerapan dengan penjumlahan citra yang menggunakan bobot



```
#Penggabungan citra lena dan baboon
```

```
import cv2
```

```
lena = cv2.imread('lena.png')
```

```
baboon= cv2.imread('baboon.png')
```

```

hasilA = cv2.addWeighted(baboon, 0.5, lena, 0.5, 0)
cv2.imshow('Alfa: 0.5, beta: 0.5, gamma: 0', hasilA)

hasilB = cv2.addWeighted(baboon, 0.7, lena, 0.3, 0)
cv2.imshow('Alfa: 0.7, beta: 0.3, gamma: 0', hasilB)

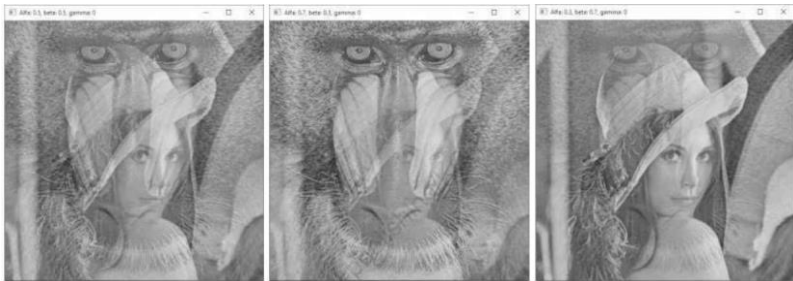
hasilC = cv2.addWeighted(baboon, 0.3, lena, 0.7, 0)
cv2.imshow('Alfa: 0.3, beta: 0.7, gamma: 0', hasilC)

cv2.waitKey(0)

```

Akhir berkas

Gambar 4.42 menunjukkan hasil ketiga pemaduan kedua gambar dengan nilai alfa dan beta yang berbeda.



Gambar 4.4/2 Hasil penggabungan citra dengan addWeighted()

Cara inilah yang digunakan pada *morphing*, yang digunakan untuk mengubah suatu gambar menjadi gambar lain secara gradual.

Penggabungan citra berikutnya yang dibahas adalah cara meletakkan dua gambar berukuran sama secara bersebelahan atau bertumpukan. Untuk meletakkan dua gambar bersebelahan, `hstack()` milik Numpy bisa digunakan. Contoh diperlihatkan pada skrip berikut:

 **Berkas : `gabung3.py`**

```

#Penggabungan dua gambar secara bersebelahan

import cv2
import numpy as np

```

```

citraA    cv2.imread('baboon.png')
citraB    cv2.imread('lena.png')

hasil = np.hstack((citraA, citraB))

# Tampilkan hasilnya
cv2.imshow('Hasil', hasil)

```

```
waitKey(0)
```

Akhir berkas

Penggabungan dua citra secara bersebelahan dilakukan oleh:

```
hasil = np.hstack((citraA, citraB))
```

Hasilnya diperlihatkan pada Gambar 4.43.



Gambar 4.43: Penggabungan dua citra bersebelahan

Adapun skrip berikut memberikan contoh penggabungan dua citra menggunakan `vs tack ()`:



Berkas : `gabung3.py`

```

# Penggabungan dua gambar secara bertumpukan

import cv2
import numpy as np

citraA    cv2.imread('baboon.png')
citraB    cv2.imread('lena.png')

hasil = np.vstack((citraA, citraB))

# Tampilkan hasilnya

```

```
cv2.imshow('Hasil', hasil)
```

```
waitKey(0)
```

Akhir berkas

Penggabungan dua citra secara bersebelahan dilakukan oleh:

```
hasil = np.vstack({citraA, citraB})
```

Hasilnya diperlihatkan pada Gambar 4.44.



Gambar 4.44 Penggabungan dua citra bertumpukan

Catatan

Pada penggabungan dengan `hstack ()` / `tinggi` semua citra yang digabungkan harus sama. Adapun pada penggabungan dengan `vs tack ()`, `lebar` semua citra yang digabungkan harus sama.
