BAB 06

TRANFORMASI GEOMETRIK

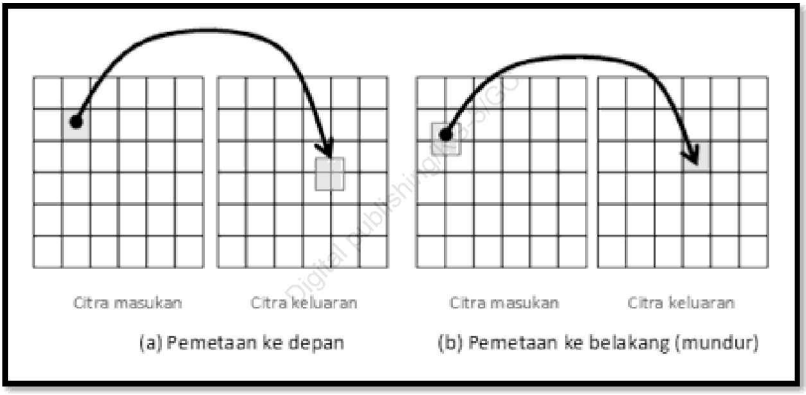
Pokok bahasan pada bab ini mencakup hal-hal berikut:

- pengantar transformasi geometrik;
- penyekalaan citra dengan `resize();`
- metode `wrapAffine();`
- penggeseran citra;
- pemutaran citra;
- penyekalaan dengan `wrapAffine();`
- pembengkokan citra;
- pencerminan citra;
- transformasi Affine.

6.1 Pengantar Transformasi Geometrik

Transformasi geometrik adalah suatu operasi pada citra yang dilakukan secara geometris yang mencakup translasi, rotasi, dan penyekalaan. Transformasi yang dilakukan tidak mengubah konten pada citra.

Transformasi geometrik melibatkan pemetaan geometrik, yang menyatakan hubungan pemetaan antara piksel pada citra masukan dan piksel pada citra keluaran. Dua cara yang digunakan dapat berupa pemetaan ke depan dan kedua berupa pemetaan ke belakang. Perbedaan kedua cara ini ditunjukkan pada Gambar 6.1.



Gambar 6.1 Pemetaan geometrik

Tampak bahwa pada pemetaan ke depan, posisi pada citra keluaran ditentukan oleh posisi piksel pada citra masukan. Seandainya hasil perhitungan menghasilkan posisi yang tidak tepat, yakni berada pada empat posisi piksel keluaran karena hasil yang tidak berupa bilangan bulat, penempatannya dapat berada pada salah satu dari empat kemungkinan. Oleh karena itu, ada kemungkinan suatu piksel pada citra keluaran tidak pernah mendapat jatah sama sekali, yang menimbulkan efek kosong sehingga timbul bintik-bintik hitam (Gambar 6.2). Hal ini

berbeda dengan pada pemetaan ke belakang karena semua piksel pada citra keluaran akan diberi nilai sekali saja berdasarkan piksel masukan.

(a) Citra gedung asli

(b) Hasil pemutaran yang menimbulkan lubang-lubang (bintik-bintik gelap) pada citra

Gambar 6.2 Efek pemetaan ke depan (Sumber: Kadir dan Susanto, 2012)

Pada Gambar 6.2(a), piksel yang digunakan untuk menentukan piksel keluaran dapat ditentukan oleh salah satu piksel yang tercakup dalam kotak yang menggantung pada keempat piksel. Hal itu merupakan cara paling sederhana yang dapat dilakukan dan biasa dinamakan sebagai pemilihan berdasarkan tetangga terdekat. Di OpenCV, cara ini dinyatakan dengan konstanta `INTER_NEAREST`. Cara lain dilakukan dengan memperhitungkan empat piksel yang dapat mewakilinya. Cara ini dikenal dengan sebutan interpolasi bilinear, yaitu linear di arah vertikal dan mendatar. Di OpenCV, cara ini dinyatakan dengan konstanta `INTER_LINEAR`.

Selain dua cara interpolasi yang telah dibahas di depan, terdapat tiga metode lain berupa:

- `INTER_AREA`: (metode ini menerapkan *resampling* menggunakan hubungan daerah piksel dan dianjurkan dipakai hanya pada pengecilan citra);
- `INTER_CUBIC` (menggunakan metode interpolasi *bicubic* yang melibatkan ketetanggaan 4 x 4 piksel);
- `INTER_LANCZOS4`: (menggunakan metode interpolasi Lanczos yang melibatkan ketetanggaan 8 x 8 piksel).

Dalam beberapa metode yang disediakan oleh OpenCV untuk transformasi geometrik, pengaturan metode untuk interpolasi bisa dipilih. Namun, secara bawaan, metode `INTER_LINEAR` digunakan.

6.2 Penyekalaan Citra dengan resize()

Penyekalaan citra dapat berarti pembesaran citra ataupun pengecilan citra. Hal ini dapat dilakukan dengan menggunakan `cv2.resize()`. Bentuk pemakaiannya seperti berikut:

resize(*citraAsal*, *skala*, interpolation = INTER_LINEAR)

Argumen pertama berupa citra sumber yang menjadi bahan penyekalaan. Argumen ketiga menyatakan skala yang digunakan. Dalam hal ini, argumen ini berupa tupel dengan dua elemen, dengan elemen pertama berupa skala pada arah mendatar dan elemen kedua skala pada arah vertikal. Argumen ketiga menentukan jenis interpolasi yang digunakan dalam penyekalaan. Nilai bawaannya adalah `INTER_LINEAR`. Dengan demikian, argumen ini bisa tidak disertakan ketika memanggil `resize()`. Adapun nilai balik `resize()` berupa citra penyekalaan.

Contoh berikut menunjukkan cara memperkecil citra setengah kali ukuran pada citra asli:

Berkas : perkecil.py

```
# Cara mengecilkan citra

import cv2

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

hasil = cv2.resize(citra,
            (int(0.5 * jumBaris), int(0.5 * jumKolom)))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Asal', citra)
cv2.imshow('Hasil', hasil)

cv2.waitKey(0)
```

Akhir berkas

Setelah berkas citra `taipei101.png` dibaca, dimensi citra diperoleh melalui:

```
jumBaris, jumKolom = citra.shape[:2]
```

Notasi `:2` berarti dari 0 hingga 1. Dengan demikian, `jumBaris` berisi `shape[0]` dan `jumKolom` berisi `shape[1]`.

Perintah berikut digunakan memperkecil citra setengah kali ukuran citra sumber:

```
hasil = cv2.resize(citra,
        (int(0.5 * jumKolom), int(0.5 * jumBaris)))
```

Hal ini diatur melalui argumen kedua `resize()`. Penggunaan `int()` dimaksudkan agar nilai ukuran citra hasil berupa bilangan bulat. Perhatikan bahwa tupel pada argumen kedua `resize()` berupa jumlah baris dan jumlah kolom citra baris. Gambar 6.3 menunjukkan hasil skrip `perkecil.py`.

Gambar 6.3 Hasil pengecilan citra

Adapun contoh berikut menunjukkan cara mengatur citra agar tingginya berupa 100 piksel:

Berkas : perkecil.py

```
# Cara mengatur gambar dengan tinggi 100 piksel

import cv2

citra = cv2.imread('taipei101.png')

jumBaris, jumKolom = citra.shape[:2]
rasio = jumKolom / jumBaris
tinggiBaris = 100

hasil = cv2.resize(citra,
            (int(rasio * tinggiBaris), tinggiBaris))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Asal', citra)
```

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

cv2.waitKey(0)
```

Akhir berkas

Untuk mempertahankan rasio citra hasil seperti citra sumber, rasio citra asal dihitung terlebih dahulu melalui:

```
rasio = jumKolom / jumBaris
```

Selanjutnya, rasio ini digunakan untuk menentukan jumlah kolom melalui:

```
int(rasio * tinggiBaris)
```

6.3 Metode wrapAffine()

Untuk kepentingan menangani berbagai jenis transformasi, metode `cv2.wrapAffine()` disediakan. Fungsi ini dapat digunakan untuk rotasi citra, translasi citra, penyekalaan citra, maupun transformasi *affine*. Bentuk penggunaannya yang paling dasar adalah seperti berikut:

cv2.warpAffine(*citra*, *M*, *ukuran*)

Argumen pertama berupa citra yang menjadi sumber pengolahan. Argumen kedua berupa matriks berukuran 2 x 3 yang menyatakan transformasi yang hendak dilakukan. Argumen ketiga menentukan ukuran matriks hasil. Argumen ini berupa tupel dengan elemen pertama menyatakan jumlah kolom citra hasil dan elemen kedua berupa jumlah baris pada citra hasil.

6.4 Penggeseran Citra

Penggeseran atau translasi citra adalah transformasi yang menggeser setiap piksel pada citra ke arah mendatar dan vertikal sebesar nilai penggeseran yang ditetapkan.

Matriks yang diperlukan berupa:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

Dalam hal ini, t_x menyatakan nilai penggeseran pada arah mendatar dan t_y pada arah vertikal.

Contoh penggeseran citra dapat dilihat pada skrip berikut:

Berkas : perkecil.py

```
# Contoh translasi citra

import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

matriks = np.float32([[1, 0, 50], [0, 1, 100]])
hasil = cv2.warpAffine(citra, matriks,
                       (jumKolom, jumBaris))
cv2.imshow('Hasil translasi', hasil)
cv2.waitKey()
```

Akhir berkas

Setelah berkas citra `taipei101.png` dibaca, dimensi citra diperoleh melalui:

```
jumBaris, jumKolom = citra.shape[:2]
```

Notasi `:2` berarti dari 0 hingga 1. Dengan demikian, `jumBaris` berisi `shape[0]` dan `jumKolom` berisi `shape[1]`.

Perintah berikut digunakan untuk membuat matriks translasi:

```
matriks = np.float32([[1, 0, 50], [0, 1, 100]])
```

Nilai 50 menyatakan nilai translasi pada arah mendatar dan 100 menyatakan nilai translasi pada arah vertikal. Perlu diketahui,

`np.float32()` memungkinkan nilai dalam matriks berupa nilai pecahan.

Gambar 6.4 menunjukkan hasil skrip `translasi.py`.

Gambar 6.4 Hasil translasi citra

6.5 Pemutaran Citra

Pemutaran citra adalah transformasi yang memutar setiap piksel pada citra asal berdasarkan suatu titik pusat pemutaran. Matriks untuk pemutaran berupa:

$$M = \begin{bmatrix} \alpha & \beta & (1-\alpha).x_{pusat} - \beta.y_{pusat} \\ -\beta & \alpha & \beta.x_{pusat} + (1-\alpha).y_{pusat} \end{bmatrix}$$

Dalam hal ini,

$$\alpha = skala.\cos\phi$$
$$\beta = sklala.\sin\phi$$

Matriks ini memungkinkan pemutaran tidak harus berdasarkan koordinat (0, 0) melainkan pada (x, y) dan sekaligus memungkinkan penyekalaan.

Matriks M tersebut dapat diperoleh dengan mudah menggunakan metode `cv2.getRotationMatrix2D()`. Bentuk pemakaiannya seperti berikut:

M = getRotationMatrix2D(*pusat*, *sudut*, *skala*)

Argumen pertama berupa tupel yang berisi dua elemen dengan elemen pertama menyatakan x titik pusat pemutaran citra dan elemen kedua menyatakan y titik pusat pemutaran citra. Argumen kedua menyatakan sudut dalam derajat. Nilai positif membuat citra diputar ke kiri dan nilai negatif membuat citra diputar ke kanan. Argumen ketiga menyatakan skala untuk memperbesar atau memperkecil citra hasil.

Contoh berikut menunjukkan cara untuk memutar citra:

Berkas : rotasi.py

```
# Contoh rotasi citra

import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

matriks = cv2.getRotationMatrix2D(
   (jumKolom // 2, jumBaris // 2), -15, 1)
hasil = cv2.warpAffine(citra, matriks,
                       (jumKolom, jumBaris))
cv2.imshow('Hasil translasi', hasil)
cv2.waitKey()
```

Akhir berkas

Setelah berkas citra `taipei101.png` dibaca, dimensi citra diperoleh melalui:

```
jumBaris, jumKolom = citra.shape[:2]
```

Notasi `:2` berarti dari 0 hingga 1. Dengan demikian, `jumBaris` berisi `shape[0]` dan `jumKolom` berisi `shape[1]`.

Perintah berikut digunakan untuk membuat matriks rotasi:

```
matriks = cv2.getRotationMatrix2D(
   (jumKolom // 2, jumBaris // 2), -15, 1)
```

Dalam hal ini, `jumKolom // 2` menyatakan nilai x titik tengah citra dan `jumBaris // 2` menyatakan nilai y titik tengah citra. Penggunaan // digunakan untuk memastikan bahwa hasilnya adalah bilangan bulat. Nilai -15 menyatakan sudut 15° searah jarum jam. Nilai 1 menyatakan bahwa citra hasil berukuran sama dengan citra asal.

Gambar 6.5 menunjukkan hasil skrip `rotasi.py`.

Gambar 6.5 Hasil pemutaran citra dengan titik pusat di tengah-tengah citra

Jika matriks rotasi diubah menjadi seperti berikut, akan diperoleh hasil seperti terlihat pada Gambar 6.6:

```
matriks = cv2.getRotationMatrix2D(
    (0, 0), -15, 1)
```

Pada contoh ini, pemutaran dilakukan dengan titik pusat berupa (0, 0).

Gambar 6.6 Hasil pemutaran citra dengan titik pusat (0, 0)

6.6 Penyekalaan Citra dengan wrapAffine()

Metode wrapAffine dapat digunakan untuk melakukan translasi. Matriks yang diperlukan berupa:

$$M = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \end{bmatrix}$$

Skrip berikut digunakan untuk membuat citra berukuran setengah citra asal:

Berkas : perkecil.py

```
# Contoh penyekalaan citra
#    menggunakan wrapAffine()

import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

matriks = np.float32([[0.5, 0, 0], [0, 0.5, 0]])

hasil = cv2.warpAffine(citra, matriks,
                       (jumKolom, jumBaris))
cv2.imshow('Hasil penyekalaan', hasil)
cv2.waitKey()
```

Akhir berkas

Setelah berkas citra `taipei101.png` dibaca, dimensi citra diperoleh melalui:

```
jumBaris, jumKolom = citra.shape[:2]
```

Notasi `:2` berarti dari 0 hingga 1. Dengan demikian, `jumBaris` berisi `shape[0]` dan `jumKolom` berisi `shape[1]`.

Perintah berikut digunakan untuk membuat matriks translasi untuk memperoleh citra berukuran setengah citra asal:

```
matriks = np.float32([[0.5, 0, 0], [0, 0.5, 0]])
```

Matriks inilah yang dikenakan pada `wrapAffine()`. Hasilnya diperlihatkan pada Gambar 6.7.

Gambar 6.7 Hasil pengecilan citra

Gambar 6.7 memperlihatkan bahwa dimensi citra hasil tetap berukuran seperti citra asal. Untuk mendapatkan citra yang berukuran setengahnya, ukuran setengah citra juga perlu disebutkan pada argumen ketiga pada `wrapAffine()`. Jadi, perintahnya berupa:

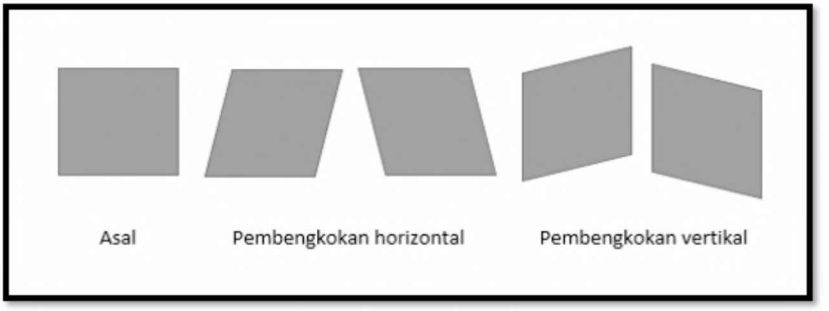
```
hasil = cv2.warpAffine(citra, matriks,
                       (jumKolom // 2, jumBaris // 2))
```

Hasilnya dperlihatkan pada Gambar 6.8.

Gambar 6.8 Hasil pengecilan citra termasuk ukurannya

6.7 Pembengkokan Citra

Pembengkokan citra adalah suatu transformasi yang melakukan penggeseran piksel ke kiri, ke kanan, ke atas, atau ke bawah secara bertahap mengikuti posisi x dan y. Gambar 6.9 memberikan efek pembengkokan citra.



Gambar 6.9 Pembengkokan citra

Pembengkokan citra secara horizontal dilakukan dengan menggunakan matriks berikut:

$$M = \begin{bmatrix} 1 & s & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Pembengkokan citra secara vertikal dilakukan dengan menggunakan matriks berikut:

$$M = \begin{bmatrix} 1 & 0 & 0 \\ s & 1 & 0 \end{bmatrix}$$

Pada kedua jenis pembengkokan tersebut, nilai *s* dapat berupa nilai positif maupun negatif yang nilai absolutnya kurang dari 1.

Contoh berikut menunjukkan cara membengkokkan citra secara horizontal:

Berkas : bengkokh.py

```
# Contoh pembengkokan citra
#    secara horizontal

import cv2
import numpy as np
import math

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

matriks = np.float32([[1, 0.25, 0],
                      [0, 1, 0]])

hasil = cv2.warpAffine(citra, matriks,
                (int(1.4 * jumKolom), jumBaris))

gab = np.hstack((citra, hasil))
cv2.imshow('Hasil pembengkokan', gab)
cv2.waitKey(0)
```

Akhir berkas

Matriks yang digunakan untuk pembengkokan horizontal menggunakan nilai 0,25. Adapun pada pemanggilan `warpAffine()`, jumlah kolom hasil diatur sebesar:

```
int(1.4 * jumKolom)
```

Hal ini dimaksudkan agar bagian citra bagian bawah tidak terpenggal. Hasilnya dapat dilihat pada Gambar 6.10.

Gambar 6.10 Pembengkokan mendatar

Apabila matriks diubah dengan nilai pembengkokan mendatar menjadi negatif seperti berikut, pembengkokan yang terjadi berlawanan dengan contoh sebelum ini:

```
matriks = np.float32([[1, -0.25, 0],
                      [0, 1, 0]])
```

Hasilnya diperlihatkan pada Gambar 6.11.

Gambar 6.11 Pembengkokan mendatar ke arah kanan

Agar bagian kiri tidak terpotong, translasi pada arah mendatar perlu diatur, misalnya sebesar 180 seperti berikut:

```
matriks = np.float32([[1, -0.25, 180],
                      [0, 1, 0]])
```

Hasilnya ditunjukkan pada Gambar 6.12.

Gambar 6.12 Pembengkokan ke kanan disertai dengan translasi ke kanan

Contoh berikut menunjukkan cara membengkokkan citra secara vertikal:

Berkas : bengkokv.py

```
# Contoh pembengkokan citra
#   secara vertikal

import cv2
import numpy as np
import math

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

matriks = np.float32([[1, 0, 0],
                      [0.25, 1, 0]])

hasil = cv2.warpAffine(citra, matriks,
                (jumKolom, jumBaris))

gab = np.hstack((citra, hasil))
cv2.imshow('Hasil pembengkokan', gab)
cv2.waitKey(0)
```

Akhir berkas

Matriks yang digunakan untuk pembengkokan vertikal menggunakan nilai 0,25. Hasilnya dapat dilihat pada Gambar 6.13.

Gambar 6.13 Hasil pembengkokan ke bawah

Apabila matriks diubah dengan nilai pembengkokan horizontal menjadi negatif seperti berikut, pembengkokan yang terjadi berlawanan dengan contoh sebelum ini:

```
matriks = np.float32([[1, 0, 0],
                      [-0.25, 1, 0]])
```

Hasilnya diperlihatkan pada Gambar 6.14.

Gambar 6.14 Hasil pembengkokan ke atas

Pembengkokan juga dapat dilakukan secara vertikal dan horizontal. Contoh:

```
matriks = np.float32([[1, 0.1, 0],
                      [0.25, 1, 0]])
```

Hasilnya diperlihatkan pada Gambar 6.15.

Gambar 6.15 Hasil pembengkokan secara vertikal dan horizontal

6.8 Pencerminan Citra

Metode `warpAffine()` dapat digunakan untuk melakukan pencerminan. Matriks yang digunakan berupa seperti berikut.

- Untuk pencerminan ke arah horizontal:

$$M = \begin{bmatrix} -1 & 0 & t_x \\ 0 & 1 & 0 \end{bmatrix}$$

- Untuk pencerminan ke arah vertikal:

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & t_y \end{bmatrix}$$

- Untuk pencerminan ke kedua arah:

$$M = \begin{bmatrix} -1 & 0 & t_x \\ 0 & -1 & t_y \end{bmatrix}$$

Dalam hal ini, t_x diisi dengan lebar citra dan t_y diisi dengan tinggi citra.

Skrip berikut menunjukkan contoh pencerminan ke kedua arah:

Berkas : bengkokv.py

```
# Pencerminan citra secara vertikal dan horizontal

import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

matriks = np.float32([[-1, 0, jumKolom],
                      [0, -1, jumBaris]])

hasil = cv2.warpAffine(citra, matriks,
                       (jumKolom, jumBaris))
gab = np.hstack((citra, hasil))

cv2.imshow('Hasil pencerminan', gab)
cv2.waitKey()
```

Akhir berkas

Setelah berkas citra `taipei101.png` dibaca, dimensi citra diperoleh melalui:

```
jumBaris, jumKolom = citra.shape[:2]
```

Notasi `:2` berarti dari 0 hingga 1. Dengan demikian, `jumBaris` berisi `shape[0]` dan `jumKolom` berisi `shape[1]`.

Perintah berikut digunakan membentuk matriks yang digunakan untuk mencerminkan citra ke kedua arah:

```
matriks = np.float32([[-1, 0, jumKolom],
                      [0, -1, jumBaris]])
```

Hasil skrip diperlihatkan pada Gambar 6.16.

Gambar 6.16 Hasil pencerminan dengan sumbu tegak dan sumbu mendatar

Adapun skrip berikut menunjukkan cara mencerminkan gambar menggunakan `cv2.flip()`:

Berkas : cerminb.py

```
# Pencerminan citra secara vertikal dan horizontal
#    Versi 2

import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
hasil = cv2.flip(citra, -1)
gab = np.hstack((citra, hasil))

cv2.imshow('Hasil pencerminan', gab)
cv2.waitKey()
```

Akhir berkas

Argumen -1 pada `flip()` menyatakan bahwa pencerminan dilakukan secara vertikal dan horizontal.

Skrip berikut memberikan contoh pencerminan citra dengan sumbu tegak menggunakan `wrapAffine()`:

Berkas : bengkokv.py

```
# Pencerminan citra dengan sumbu tegak

import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

matriks = np.float32([[-1, 0, jumKolom], [0, 1, 0]])

hasil = cv2.warpAffine(citra, matriks,
                       (jumKolom, jumBaris))
gab = np.hstack((citra, hasil))

cv2.imshow('Hasil pencerminan', gab)
cv2.waitKey()
```

Akhir berkas

Hasilnya ditunjukkan pada Gambar 6.17.

Gambar 6.17 Hasil pencerminan dengan sumbu tegak

Adapun skrip berikut menunjukkan perwujudan dengan `flip()`:

Berkas : bengkok2.py

```
# Pencerminan citra dengan sumbu tegak
#    Versi 2

import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
hasil = cv2.flip(citra, 1)
gab = np.hstack((citra, hasil))

cv2.imshow('Hasil pencerminan', gab)
cv2.waitKey()
```

Akhir berkas

Argumen kedua yang bernilai 1 menyatakan bahwa pencerminan dilakukan dengan sumbu tegak.

Adapun skrip berikut menunjukkan pencerminan citra dengan sumbu mendatar menggunakan `wrapAffine()`:

Berkas : bengkok3.py

```
# Pencerminan citra dengan sumbu mendatar

import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

matriks = np.float32([[1, 0, 0], [0, -1, jumBaris]])

hasil = cv2.warpAffine(citra, matriks,
                       (jumKolom, jumBaris))
gab = np.hstack((citra, hasil))

cv2.imshow('Hasil pencerminan', gab)
cv2.waitKey()
```

Akhir berkas

Hasilnya ditunjukkan pada Gambar 6.18.

Gambar 6.18 Hasil pencerminan dengan sumbu tegak

Adapun skrip berikut menunjukkan perwujudan dengan `flip()`:

Berkas : cermin3b.py

```
# Pencerminan citra dengan sumbu mendatar
#    Versi 2
import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]
hasil = cv2.flip(citra, 0)
gab = np.hstack((citra, hasil))

cv2.imshow('Hasil pencerminan', gab)
cv2.waitKey()
```

Akhir berkas

Argumen kedua yang bernilai 1 menyatakan bahwa pencerminan dilakukan dengan sumbu mendatar.

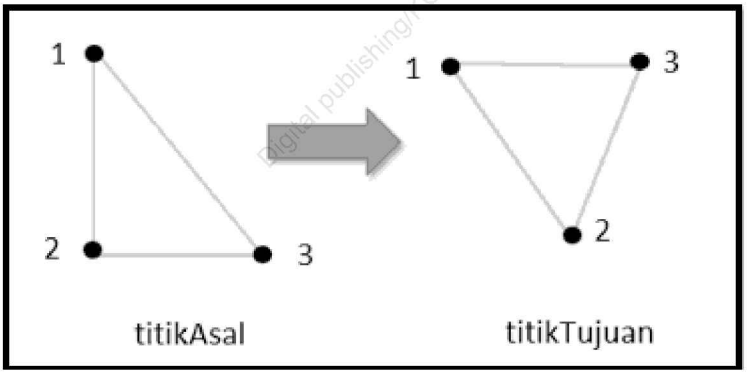
6.9 Transformasi Affine

Transformasi *affine* adalah transformasi yang menggabungkan penyekalaan, translasi, dan rotasi. Hal ini dapat menimbulkan efek pembengkokan.

Matriks untuk transformasi *affine* dapat diperoleh melalui `cv2.getAffineTransfrom()`. Bentuk pemakaiannya seperti berikut:

nilaiBalik = cv.getAffineTransform(*titikAsal*, *titikTujuan*)

Dalam hal ini, *titikAsal* dan *titikTujuan* berisi titik segitiga, yang menyatakan perubahan dari posisi di pada *titikAsal* di citra asal menjadi posisi di *titikTujuan* pada citra hasil. Gambar 6.19 memberikan ilustrasi mengenai hal ini.



Gambar 6.19 Transformasi citra melalui titik kontrol segitiga

Skrip berikut memberikan contoh penggunaan `cv2.getAffineTransfrom()`.

Berkas : bengkokv.py

```
# Contoh transformasi affine

import cv2
import numpy as np

citra = cv2.imread('taipei101.png')
jumBaris, jumKolom = citra.shape[:2]

titik1 = np.float32([[10, 10], [10, 50], [50, 50]])
titik2 = np.float32([[10, 0], [20, 50],[50, 60]])
matriks = cv2.getAffineTransform(titik1, titik2)
hasil = cv2.warpAffine(citra, matriks,
                       (jumKolom, jumBaris))

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

Akhir berkas

Hasil yang didapat diperlihatkan pada Gambar 6.20.

Gambar 6.20 Hasil transformasi affine