

BAB 08

IRANSFORMASI MORFOLOGIS

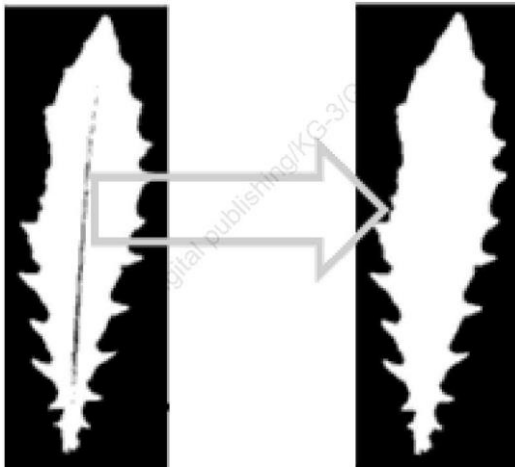
Pokok bahasan pada bab ini mencakup hal-hal berikut:

- pengertian transformasi morfologis;
- elemen pembentuk struktur;
- transformasi dilasi;
- transformasi erosi;
- pemerolehan tepi objek;
- efek ukuran dan bentuk elemen pembentuk struktur;
- transformasi pembukaan;
- transformasi penutupan;
- transformasi gradien morfologis;
- transformasi morfologis pada citra berskala keabu-abuan dan citra berwarna;
- transformasi top-hat;
- transformasi black-hat;
- transformasi hit-or-miss.

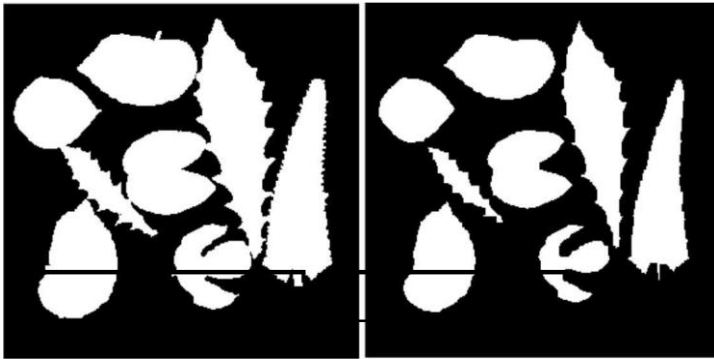
8.1 Pengertian Transformasi Morfologis

Transformasi morfologis merupakan operasi yang umum dikenakan pada citra biner (hitam-putih) untuk mengubah struktur bentuk suatu objek yang terkandung dalam suatu citra. Namun, beberapa jenis transformasi morfologis juga dapat diterapkan pada citra berskala keabu-abuan.

Gambar 8.1 memperlihatkan contoh penerapan transformasi morfologis yang dapat digunakan untuk lubang pada daun. Objek-objek daun yang saling berhimpitan pun dapat dipisahkan melalui transformasi morfologis, sebagaimana ditunjukkan pada Gambar 8.2.

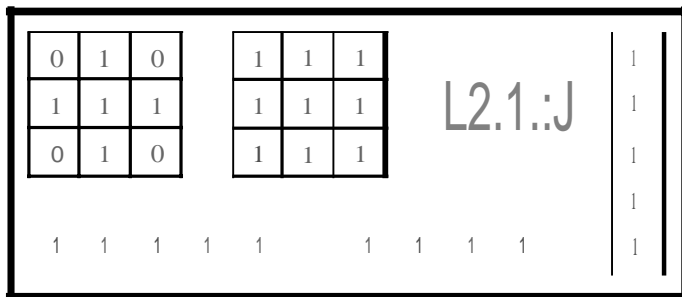


Gambar 8.1! Tulang daun dapat diperlakukan sebagai bagian daun melalui morfologi

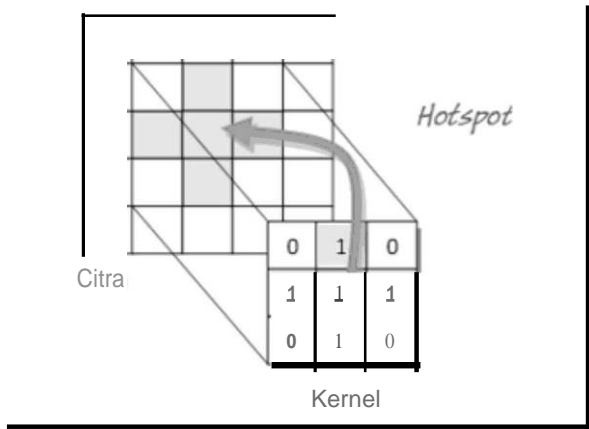


Gambar 8.5 Sekumpulan daun yang bersinggungan dapat dipisahkan melalui transformasi morfologis yang berdampak pada pengecilan ukurandaun

Transformasi morfologis melibatkan dua larik piksel. Larik pertama berupa citra yang akan dikenai transformasi morfologis, sedangkan larik kedua dinamakan kernel atau elemen pembentuk struktur (*structuring element*). Contoh berbagai kernel ditunjukkan pada Gambar 8.4. Pada contoh tersebut, piksel pusat (biasa diberi nama *hotspot*) ditandai dengan warna abu-abu. Piksel pusat ini yang menjadi pusat dalam melakukan operasi terhadap citra, sebagaimana diilustrasikan di Gambar 8.5.



Gambar 8.4 Contoh sejumlah elemen pembentuk struktur



Gambar 8.5 Operasi elemen pembentuk struktur terhadap citra

Dua jenis transformasi morfologis berupa dilasi dan erosi. Dua operasi varian lain yang didasarkan pada kedua operasi tersebut adalah *opening* dan *closing*. Masing-masing akan dibahas secara tersendiri.

8.2 Elemen Pembentuk Struktur

Untuk kepentingan memudahkan dalam membentuk elemen penstruktur, OpenCV menyediakan `getStructuringElement()`. Bentuk pemakaiannya seperti berikut:

```
cv2.getStructuringElement(bentuk, (m,n))
```

Dalam hal ini, argumen pertama menentukan bentuk elemen pembentuk struktur, yang dapat berupa:

- `cv2.MORPH_RECT` (kernel berupa kotak);
- `cv2.MORPH_ELLIPSE` (kernel berupa elips);
- `cv2.MORPH_CROSS` (kernel berbentuk silang).

Adapun argumen kedua berupa tupel, dengan m menyatakan jumlah kolom dan n menyatakan jumlah baris.

Contoh berikut memperlihatkan pembentukan elemen pembentuk struktur berbentuk kotak berukuran 5 x 5:

```
>>> cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]], dtype=uint8)
>>>
```

Hasil tersebut identik dengan perintah berikut:

```
import numpy as np
np.ones((5,5), np.uint8)
```

Contoh berikut memperlihatkan pembentukan elemen pembentuk struktur berbentuk kotak berukuran 1 x 5:

```
>>> cv2.getStructuringElement(cv2.MORPH_RECT, (5, 1))
: array([[1, 1, 1, 1, 1]], dtype=uint8)
: >>
```

Conteh berikut memperlihatkan pembentukan elemen pembentuk struktur berbentuk elips berukuran 5 x 5:

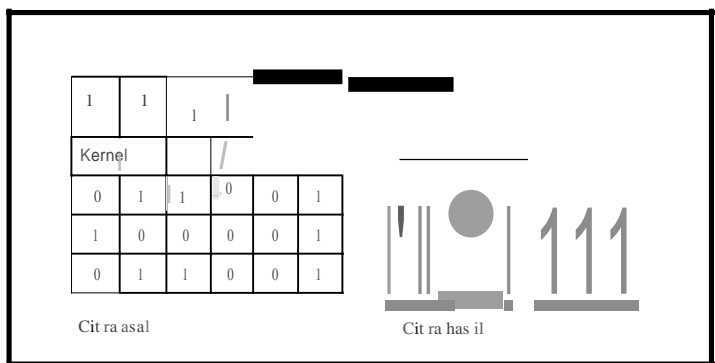
```
>>> cv2.getStructuringElement(cv2.CV_MORPH_ELLIPSE,
(5,5))
array([[0, 0, 1, 0, 0],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1],
       [0, 0, 1, 0, 0]], dtype=uint8)
>>>
```

Conteh berikut memperlihatkan pembentukan elemen pembentuk struktur berbentuk silang berukuran 5 x 5:

```
>>> cv2.getStructuringElement(cv2.MORPH_CROSS, (5, 5))
array([ [0, 0, 1, 0, 0],
        [0, 0, 1, 0, 0],
        [1, 1, 1, 1, 1],
        [0, 0, 1, 0, 0],
        [0, 0, 1, 0, 0], dtype=uint8)
>>>
```

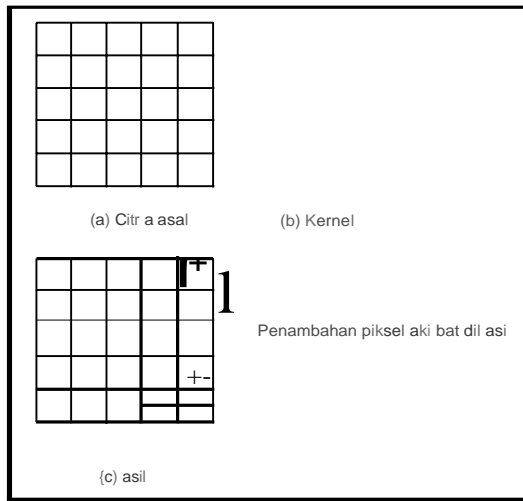
8.3 Transformasi Dilasi

Transformasi dilasi biasa dipakai untuk mendapatkan efek pelebaran terhadap piksel yang bernilai 1. Prinsipnya, setiap piksel pada citra yang berada di bawah kernel (elemen pembentuk struktur) yang bernilai 1 akan berubah menjadi 1. Ilustrasi dapat dilihat pada Gambar 8.6. Pada contoh ini, piksel yang bernilai 1 dikenai operasi dengan kernel. Karena semua elemen di kernel bernilai 1, maka piksel pada citra hasil pada posisi kernel menumpang akan bernilai 1. Hal ini tidak berlaku untuk piksel yang bernilai 0. Hal inilah yang membuat operasi dilasi memperlebar objek.



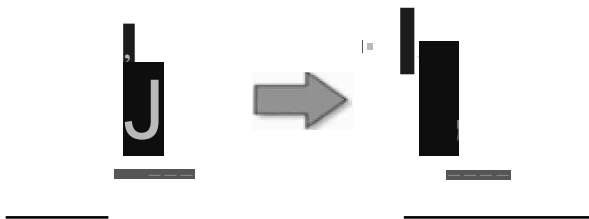
Gambar 8.6 Proses pelebaran piksel melalui dilasi

Gambar 8.7 menunjukkan hasil lengkap setelah operasi dilasi menggunakan suatu kernel diterapkan pada citra. Karena kernel bersifat tegak, pelebaran dilakukan pada arah atas dan bawah.

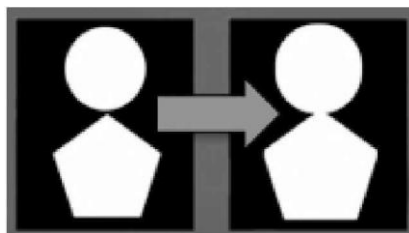


Gambar 8.7 Contoh pelebaran objek melalui dilasi

Efek transformasi dilasi, sebagai contoh, suatu tulisan dapat diperbesar (Gambar 8.7) dan dua bagian yang terputus dapat disambung (Gambar 8.8).



Gambar 8.8 Efek pelebaran tulisan melalui dilasi



Gambar 8.8 Contoh penyambungan dua bagian terputus melalui dilasi

Transformasi dilasi dilaksanakan dengan menggunakan `cv2.dilate()`. Bentuk pemakaiannya seperti berikut:

`cv2.dilate(citra, kernel)`

Argumen pertama berupa citra asal sebagai sumber untuk diproses. Argumen kedua berupa elemen pembentuk struktur. Nilai balik berupa citra yang mengalami dilasi.

Berikut memberikan gambaran penggunaan `dilate()`:



Berkas :dilasi.py

```
# Contoh penggunaan dilate()

import cv2
import numpy as np

citra = cv2.imread('tulisan.png', 0)

kernel = cv2.getStructuringElement(
    cv2.MORPH_RECT, (3, 3))
dilasi = cv2.dilate(citra, kernel)

hasil = np.hstack((citra, dilasi))

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

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



Akhir berkas

Hasilnya diperlihatkan pada Gambar 8.9.



Gambar 8.9 Hasil pembesaran tulisan dengan dilasi

Jumlah pengenalan dilasi dapat dilakukan beberapa kali. Hal ini dilaksanakan dengan menyertakan argumen `iterations = n` pada `dilate()` dengan `n` menyatakan jumlah pengulangan. Contoh kan pada skrip berikut:

m

Berkas : dilasi2.py

```
#Contoh penggunaan dilate()
#    dengan melibatkan argumen iteration

import cv2
import numpy as np    .Z>

citra = cv2.imread('tulisan.png', 0)

kernel    cv2.getStructuringElement(
            cv2.MORPH_RECT, (3, 3))
dilasi    cv2.dilate(citra, kernel, iterations    3)

hasil = np.hstack((citra, dilasi))

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

waitKey(0)
```

Akhir berkas

Hasilnya dapat dilihat pada Gambar 8.10.

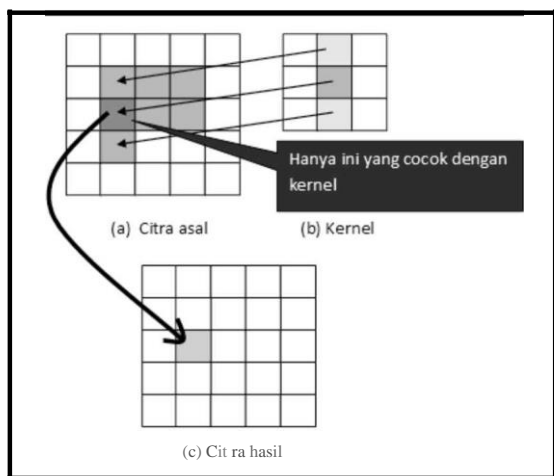


Gambar 8.10 Pembesaran sebanyak tiga kali melalui iteratons = S

8.4 Transformasi Erosi

Transformasi erosi merupakan kebalikan transformasi dilasi dan biasa dipakai untuk melakukan penyempitan terhadap suatu struktur dalam citra. Penyempitan terjadi bagian tepi suatu struktur.

Prinsipnya, mula-mula semua elemen pada citra hasil akan bernilai 0. Kemudian, setiap kali elemen pembentuk struktur menumpang pada citra. Citra hasil pada posisi *hotspot* (tengah kernel) tersebut diisi 1 hanya kalau semua piksel yang bersesuaian dengan elemen kernel yang bernilai 1 juga bernilai 1. Gambar 8.11 memperlihatkan efek erosi.



Gambar 8.11 Efek erosi

Transformasi erosi dilaksanakan dengan menggunakan `cv2.erode()`. Bentuk pemakaiannya seperti berikut:

```
cv2.erode(citra, kernel)
```

Argumen pertama berupa citra asal sebagai sumber untuk diproses. Argumen kedua berupa elemen pembentuk struktur. Nilai balik berupa citra yang mengalami erosi.

Skrin berikut memberikan gambaran penggunaan `erode()`:



```
# Contoh penggunaan erode{}

import cv2
import numpy as np

citra = cv2.imread('tulisan.png', 0)

kernel = cv2.getStructuringElement(
    cv2.MORPH_RECT, (3, 3))
dilasi = cv2.dilate(citra, kernel, iterations = 3)
erosi = cv2.erode(dilasi, kernel)

hasil = np.hstack((dilasi, erosi))

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

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

Akhir berkas

Skrin ini mula-mula memperbesar citra di `tulisan.png` dengan hasil terlihat pada Gambar 8.12 bagian kiri. Kemudian, `erode()` yang dikenakan pada citra ini membuat tulisan "Pelangi" menjadi lebih ramping (Gambar 8.12 bagian kanan).



Gambar 8.1f1 Erosi terhadap citra yang berada di kiri dengan hasil di kanan

Seperti halnya pada dilate (), jumlah pengenaaan erosi dapat dilakukan beberapa kali. Hal ini dilaksanakan dengan menyertakan argumen `iterations = n` pada `erode ()` dengan *n* menyatakan jumlah pengulangan. Contoh ditunjukkan pada skrip berikut:

11!1 Berkas : erosi2.py

```
# Contoh penggunaan erode()
#     dengan melibatkan iterations

import cv2
import numpy as np

citra = cv2.imread('tulisan.png', 0)

kernel    cv2.getStructuringElement(
            cv2.MORPH_RECT, (3, 3))
dilasi    cv2.dilate(citra, kernel, iterations= 3)
erosi     cv2.erode(dilasi, kernel, iterations= 3)

hasil     np.hstack((dilasi, erosi))

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

waitKey(0)
```

Akhir berkas

Pada skrip ini, `erode()` melibatkan `iterations` diperlihatkan pada Gambar 8.13.

3. Hasilnya



Gambar 8.13 Hasil pembesaran tiga kali dan pengecilan tiga kali melalui transformasi morfologis

Contoh berikut memperlihatkan efek erosi yang dapat dimanfaatkan untuk memisahkan sejumlah dedaunan yang bersinggungan:

11!1 Berkas : pisahdaun.py

```
# Contoh untuk menunjukkan efek erode()
#   untuk memisahkan sejumlah daun
#   yang bersinggungan

import cv2
import numpy as np

citra = cv2.imread('dedaunan.png', 0)

# Konversi ke biner
ambang, biner = cv2.threshold(citra,
                              30, 255, cv2.THRESH_BINARY)

kernelA = cv2.getStructuringElement(
    cv2.MORPH_RECT, (4, 4))
kernelB = cv2.getStructuringElement(
    cv2.MORPH_RECT, (6, 6))

erosiA = cv2.erode(biner, kernelA)
erosiB = cv2.erode(biner, kernelB)

baris1 = np.hstack((citra, biner))
baris2 = np.hstack((erosiA, erosiB))
```

```
hasil = np.vstack((baris1, baris2))

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

Akhir berkas

Hasil skrip ini diperlihatkan pada Gambar 8.14. Tampak bahwa semakin besar ukuran kernel, antardaun terpisah agak jauh. Namun, sebagai konsekuensinya, ukuran daun menjadi lebih kecil.



Gambar 8.14 Pemisahan dedaunan melalui transformasi morfologis

8.5 Pemerolehan Tepi Objek

Transformasi erosi dapat dimanfaatkan untuk mendapatkan tepi suatu objek. Hal ini dilakukan dengan melakukan pengurangan citra biner asal dengan citra hasil erosi.

Skrip berikut memberikan gambaran untuk mendapatkan tepi objek melalui transformasi morfologis:

```
#Contoh penggunaan erode()
#    untuk mendapatkan tepi objek

import cv2
import numpy as np

citra = cv2.imread('bentuk.png', 0)

kernel= cv2.getStructuringElement(
            cv2.MORPH_RECT, (3, 3))
erosi = cv2.erode(citra, kernel)
tepi = citra - erosi

hasil = np.hstack((citra, tepi))

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

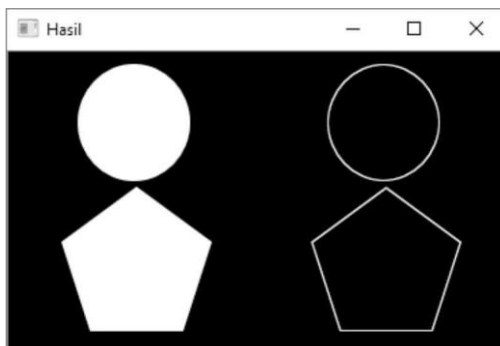
waitKey(0)
```

Akhir berkas

Kunci dalam mendapatkan tepi objek adalah pada pernyataan:

```
tepi = citra - erosi
```

Hasilnya diperlihatkan pada Gambar 8.15. Bagian kiri adalah objek asal dan bagian kanan berupa tepi objek.



Gambar 8.15 *Tepi objek*

Skrip berikut memberikan gambaran untuk mendapatkan tepi objek melalui transformasi morfologis berdasarkan citra lena.png:

11!1 | Berkas : III\isi.11v

```
# Contoh penggunaan erode()
#     untuk mendapatkan tepi
#     pada citra lena.png

import cv2
import numpy as np

citra = cv2.imread('lena.png', 0)

# Konversi ke biner
ambang, citra = cv2.threshold(citra,
                               128, 255, cv2.THRESH_BINARY)

kernel= cv2.getStructuringElement(
        cv2.MORPH_RECT, (3, 3))
erosi = cv2.erode(citra, kernel)
tepi = citra - erosi

hasil = np.hstack((citra, tepi))

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

waitKey(0)
```

Akhir berkas

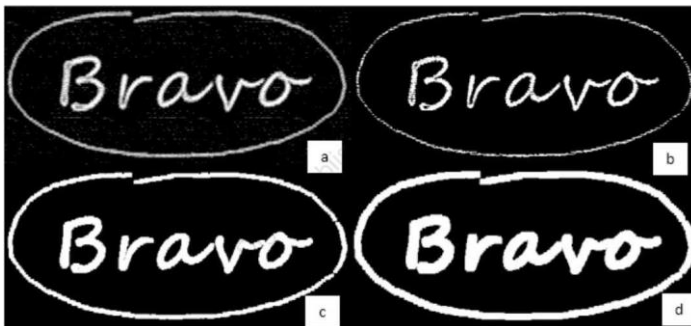
Hasilnya dapat dilihat pada Gambar 8.16.



Gambar 8.16 Hasil tepi pada citra lena.png

8.6 Efek Ukuran dan Bentuk Elemen Pembentuk Struktur

Pemilihan elemen pembentuk struktur baik dari segi ukuran maupun bentuk menentukan efek transformasi morfologis terhadap objek-objek pada citra. Gambar 8.17 menunjukkan tiga contoh citra yang dihasilkan oleh transformasi morfologis dilasi yang menggunakan dua ukuran kernel yang berbeda. Gambar 8.17(a) menyatakan citra dalam skala keabu-abuan. Gambar 8.17(b) adalah hasil konversi ke biner. Gambar 8.17(c) adalah hasil dilasi dengan kernel berukuran 4×4 . Gambar 8.17(d) adalah hasil dilasi dengan kernel berukuran 7×7 . Hal serupa sebenarnya telah disinggung pada pemisahan dedaunan (Gambar 8.14).



Gambar 8.17 Dilasi dengan ukuran berbeda pada citra bravo.png



sil di atas diproses melalui skrip berikut:

Berkas : `llil.isi.11y`

```
# Contoh untuk menunjukkan efek dilate()  
# dengan dua ukuran kernel yang berbeda  
  
import cv2  
import numpy as np  
  
citra = cv2.imread('bravo.png', 0)
```

```
#Konversi ke biner
ambang, biner      cv2.threshold(citra,
                                128, 255, cv2.THRESH_BINARY)

kernelA      cv2.getStructuringElement(
                cv2.MORPH_RECT, (4, 4))
kernelB      cv2.getStructuringElement(
                cv2.MORPH_RECT, (7, 7))

dilasiA      cv2.dilate(biner, kernelA)
dilasiB      cv2.dilate(biner, kernelB)

baris1      np.hstack((citra, biner))
baris2      np.hstack((dilasiA, dilasiB))

hasil = np.vstack((baris1, baris2))

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

waitKey(0)
```

Akhir berkas

Adapun skrip berikut memperlihatkan efek pemilihan ukuran kernel

erasi erosi:

m

Berkas : lililisi.11y

```
#Contoh untuk menunjukkan efek ukuran kernel
# pada transformasi erosi

import cv2
import numpy as np

citra = cv2.imread('struktur.png', 0)

kernelA      cv2.getStructuringElement(
                cv2.MORPH_RECT, (5, 5))
kernelB      cv2.getStructuringElement(
                cv2.MORPH_RECT, (7, 7))
kernelC      cv2.getStructuringElement(
                cv2.MORPH_RECT, (10, 10))

erosiA      cv2.erode(citra, kernelA)
erosiB      cv2.erode(citra, kernelB)
erosiC      cv2.erode(citra, kernelC)
```

```

baris1    np.hstack((citra, erosiA))
baris2    np.hstack((erosiB, erosiC))

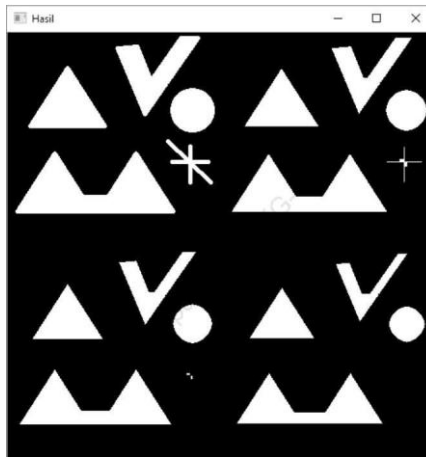
hasil = np.vstack((baris1, baris2))

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

```

Akhir berkas

Hasilnya dapat dilihat pada Gambar 8.18.



Gambar 8.18 Dilasi dengan ukuran berbeda pada citra bravo.png

Adapun skrip berikut digunakan untuk memperlihatkan efek pemilihan



kernel pada citra struktur.png:

Berkas : bentukkernel.py

```

# Contoh untuk menunjukkan efek bentuk kernel
# pada transformasi erosi

import cv2
import numpy as np

citra = cv2.imread('struktur.png', 0)

```

```

kernelA    cv2.getStructuringElement(
              cv2.MORPH_RECT, (10, 10))
kernelB    cv2.getStructuringElement(
              cv2.MORPH_ELLIPSE, (10, 10))
kernelC    cv2.getStructuringElement(
              cv2.MORPH_CROSS, (10, 10))

erosiA     cv2.erode(citra, kernelA)
erosiB     cv2.erode(citra, kernelB)
erosiC     cv2.erode(citra, kernelC)

baris1     np.hstack((citra, erosiA))
baris2     np.hstack((erosiB, erosiC))

hasil = np.vstack((baris1, baris2))

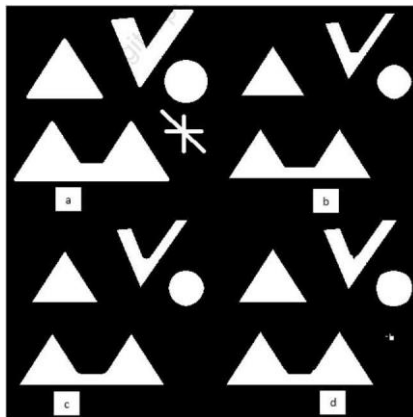
# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)

waitKey(0)

```

Akhir berkas

Hasilnya ditunjukkan pada Gambar 8.19.

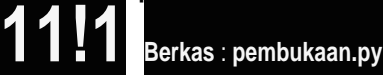


Gambar 8.19 Erosi dengan ukuran berbeda pada citra struktur.png

8.7 Transformasi Pembukaan

Transformasi pembukaan (*opening*) adalah operasi erosi yang diikuti dengan dilasi dengan menggunakan elemen pembentuk struktur yang sama. Operasi ini berguna untuk menghaluskan kontur objek dan menghilangkan seluruh piksel di area yang terlalu kecil untuk ditempati oleh elemen pembentuk struktur. Dengan perkataan lain, semua struktur latar depan yang berukuran lebih kecil daripada elemen pembentuk struktur akan tereliminasi oleh erosi dan kemudian penghalusan dilakukan melalui dilasi.

Contoh skrip yang menerapkan transformasi pembukaan:



```
# Contoh untuk menunjukkan efek ukuran kernel
# pada transformasi erosi dan pembukaan

import cv2
import numpy as np

citra = cv2.imread('struktur.png', 0)

kernelA = cv2.getStructuringElement(
    cv2.MORPH_ELLIPSE, (10, 10))
kernelB = cv2.getStructuringElement(
    cv2.MORPH_ELLIPSE, (12, 12))

erosi = cv2.erode(citra, kernelA)
pembukaanA = cv2.morphologyEx(citra, cv2.MORPH_OPEN,
    kernelA)
pembukaanB = cv2.morphologyEx(citra, cv2.MORPH_OPEN,
    kernelB)

baris1 = np.hstack((citra, erosi))
baris2 = np.hstack((pembukaanA, pembukaanB))

hasil = np.vstack((baris1, baris2))

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

```
waitKey(0)
```

```
Akhir berkas
```

Hasilnya ditunjukkan pada Gambar 8.20.



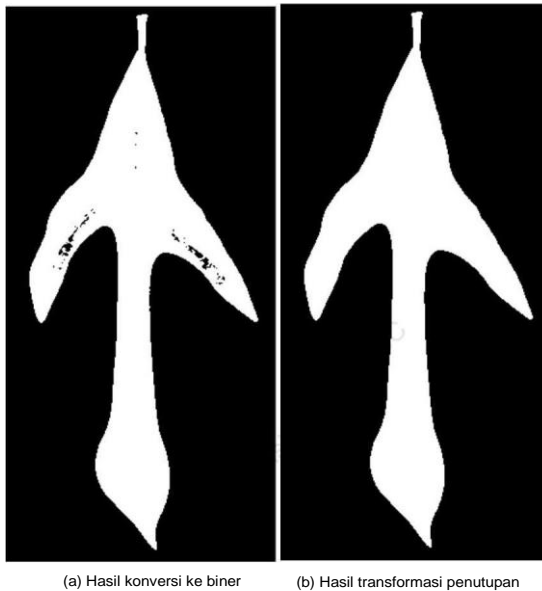
Gambar 8.20 Transformasi erosi dan pembukaan pada citra struktur.png

Gambar 8.20 menunjukkan bahwa operasi erosi membuat objek mengecil dan bahkan ada yang hilang. Adapun operasi pembukaan membuat ukuran objek relatif tetap sama, walaupun juga menghilangkan objek berukuran kecil (kurus). Namun, perlu diketahui, operasi pembukaan membuat penghalusan di bagian tepi. Perhatikan, ujung segitiga tidak tajam setelah dikenai operasi pembukaan.

8.8 Transformasi Penutupan

Transformasi penutupan (*closing*) berupa operasi dilasi dan erosi yang dilakukan secara berturutan dengan menggunakan kernel yang sama. Operasi ini berguna untuk menghaluskan kontur objek dan menghilangkan lubang-lubang kecil pada objek.

Gambar 8.21(a) menunjukkan contoh citra `daun_gray.png` yang mengandung objek satu daun dan di dalamnya terdapat lubang-lubang. Nah, lubang-lubang ini dapat dihilangkan dengan menggunakan transformasi penutupan sehingga diperoleh hasil seperti terlihat pada **Gambar 8.21(b)**.



(a) Hasil konversi ke biner

(b) Hasil transformasi penutupan

Gambar 8.21.11 Efek operasi penutupan



Untuk mewujudkan hal di atas adalah seperti berikut:

Berkas : penutupan.py

```
#Contoh penggunaan transformasi penutupan
#    untuk menutup lubang pada daun

import cv2
import numpy as np

citra = cv2.imread('daun_gray.png', 0)

#Konversi ke biner
ambang, citra = cv2.threshold(citra,
```

```

160, 255, cv2.THRESH_BINARY)

#Balik warna
citra = 255 - citra

kernel= cv2.getStructuringElement(
            cv2.MORPH_RECT, (7, 7))
penutupan = cv2.morphologyEx(citra, cv2.MORPH_CLOSE,
                              kernel)

hasil = np.hstack((citra, penutupan))

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

cv2.waitKey(0)

```

Akhir berkas

8.9 Transformasi Gradien Morfologis

Transformasi gradien morfologis merupakan selisih antara hasil dilasi dan erosi. Hasil yang didapat berupa *outline* objek.

Skrup berikut memberikan contoh penerapan transformasi gradien morfologis:



Berkas : enutL:11.1:1.,y

```

#Contoh transformasi gradien morfologis

import cv2
import numpy as np

citra = cv2.imread('tulisan.png', 0)

kernel = cv2.getStructuringElement(
            cv2.MORPH_RECT, (3, 3))
dilasi = cv2.dilate(citra, kernel, iterations = 4)

#Kenakan transformasi gradien morfologis
kernel= cv2.getStructuringElement(
            cv2.MORPH_RECT, (2, 2))
gradien = cv2.morphologyEx(citra, cv2.MORPH_GRADIENT,

```



```

                                kernel)
hasil = np.hstack((dilasi, gradien))

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

```

```
waitKey(0)
```

Akhir berkas

Hasilnya ditunjukkan pada Gambar 8.22.



Gambar 8. Transformasi gradien morfologis

8.10 Transformasi Morfologis pada Citra Berskala Keabu-abuan dan Citra Berwarna

Transformasi morfologis bisa diterapkan pada citra berskala keabu-abuan maupun citra berwarna. Contoh berikut menunjukkan penerapan



masi erosi pada citra berskala keabu-abuan:

Berkas : 11 nutu11;;i:1.11y

```

# Contoh transformasi erosi pada
# citra berskala keabu-abuan

import cv2
import numpy as np

citra = cv2.imread('simba.png', 0)

kernelA = cv2.getStructuringElement(

```

```

        cv2.MORPH_RECT, (3, 3))
kernelB  cv2.getStructuringElement(
        cv2.MORPH_RECT, (5, 5))
kernelC  cv2.getStructuringElement(
        cv2.MORPH_RECT, (7, 7))

erosiA   cv2.erode(citra, kernelA)
erosiB   cv2.erode(citra, kernelB)
erosiC   cv2.erode(citra, kernelC)

hasil = np.hstack((citra, erosiA,
                    erosiB, erosiC))

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

```

```
waitKey(0)
```

Akhir berkas

Hasilnya ditunjukkan pada Gambar 8.23.



Gambar 8.23 Contoh penerapan transformasi erosi pada citra keabuan

Adapun skrip berikut memperlihatkan penerapan transformasi erosi

mra berwarna:

Berkas : 11 nutu11;;in.t<Jy

```

#Contoh transformasi erosi pada
#    citra berwarna

import cv2
import numpy as np

```

```

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

kernelA    cv2.getStructuringElement(
              cv2.MORPH_RECT, (3, 3))
kernelB    cv2.getStructuringElement(
              cv2.MORPH_RECT, (5, 5))
kernelC    cv2.getStructuringElement(
              cv2.MORPH_RECT, (7, 7))

erosiA     cv2.erode(citra, kernelA)
erosiB     cv2.erode(citra, kernelB)
erosiC     cv2.erode(citra, kernelC)

hasil = np.hstack((citra, erosiA,
                   erosiB, erosiC))

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

waitKey(0)

```

Akhir berkas

Contoh berikut menunjukkan penerapan transformasi dilasi pada citra



keabu-abuan:

Berkas :11nu!Ll1'iln,11y

```

# Contoh transformasi dilasi pada
# citra berskala keabu-abuan

import cv2
import numpy as np

citra = cv2.imread('simba.png', 0)

kernelA    cv2.getStructuringElement(
              cv2.MORPH_RECT, (3, 3))
kernelB    cv2.getStructuringElement(
              cv2.MORPH_RECT, (5, 5))
kernelC    cv2.getStructuringElement(
              cv2.MORPH_RECT, (7, 7))

dilasiA     cv2.dilate(citra, kernelA)
dilasiB     cv2.dilate(citra, kernelB)
dilasiC     cv2.dilate(citra, kernelC)

```

```
hasil    np.hstack((citra, dilasiA,
                    dilasiB, dilasiC))
```

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

```
waitKey(0)
```

Akhir berkas

Hasilnya ditunjukkan pada Gambar 8.24.



Gambar 8. 4 Contoh penerapan transformasi dilasi pada citra keabu-abuan

Adapun skrip berikut memperlihatkan penerapan transformasi dilasi

mra berwarna:

Berkas : 11 nutu11;;in.11y

```
# Contoh transformasi dilasi pada
# citra berwarna
```

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

```
citra = cv2.imread('simba.png')
```

```
kernelA    cv2.getStructuringElement(
            cv2.MORPH_RECT, (3, 3))
kernelB    cv2.getStructuringElement(
            cv2.MORPH_RECT, (5, 5))
kernelC    cv2.getStructuringElement(
            cv2.MORPH_RECT, (7, 7))
```

```

dilasiA cv2.dilate(citra, kernelA)
dilasiB cv2.dilate(citra, kernelB)
dilasiC cv2.dilate(citra, kernelC)

hasil = np.hstack((citra, dilasiA,
                    dilasiB, dilasiC))

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

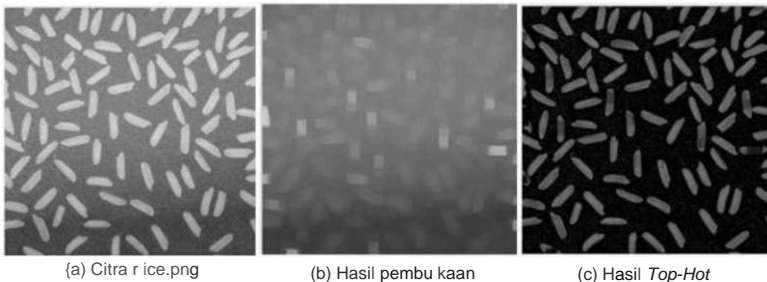
```

```
waitKey(0)
```

Akhir berkas

8.11 Transformasi Top-Hat


Transformasi *top-hat* didefinisikan sebagai perbedaan antara citra setelah mengalami operasi pembukaan dan citra asal. Transformasi ini berguna untuk mendapatkan bentuk global suatu objek yang mempunyai intensitas yang bervariasi. Sebagai contoh, perhatikan Gambar 8.25(a). Pada citra tersebut, butiran-butiran nasi memiliki intensitas yang tidak seragam. Melalui transformasi pembukaan, diperoleh hasil seperti terlihat di Gambar 8.25(b). Hasil transformasi *top-hat* ditunjukkan di Gambar 8.25(c). Perhatikan bahwa hasil butiran nasi di Gambar 8.25(c) terlihat memiliki intensitas lebih seragam dibandingkan pada citra asal.



Gambar 8.25 Hasil operasi pembukaan dan top-hat pada berkas rice.png

Catatan

Berkas `rice.png` dapat diperoleh di Internet dengan memasukkan kata-kunci berikut di mesin pencari:
`rice.png matlab`

 Berikut menunjukkan contoh transformasi *top-hat*:

```
# Contoh transformasi top-hat

import cv2
import numpy as np

citra = cv2.imread('rice.png', 0)

kernel= cv2.getStructuringElement(
    cv2.MORPH_RECT, (9, 9))

tophat = cv2.morphologyEx(citra, cv2.MORPH_TOPHAT,
                           kernel)
pembukaan = cv2.morphologyEx(citra, cv2.MORPH_OPEN,
                              kernel)

hasil = np.hstack((citra, pembukaan, tophat))

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

waitKey(0)
```

 **Akhir berkas**

8.12 Transformasi Black-Hat

Transformasi *black-hat* atau terkadang dinamakan *bottom-hat* didefinisikan sebagai perbedaan antara citra setelah mengalami operasi penutupan dengan citra asal. Dilasi yang diikuti erosi memberikan efek berupa objek-objek yang berdekatan menjadi semakin dekat. Pengurangan oleh citra asal membuat penghubung antarobjek menjadi

hasil tersisa. Dengan perkataan lain, hasil tersisa adalah piksel-piksel yang digunakan untuk mengisi "lubang" atau "penghubung objek".



memahami transformasi *black-hat*, skrip berikut bisa dicoba:

Berkas : penutu11a:1.11y

```
# Contoh untuk memperlihatkan
#      transformasi black-hat

import cv2
import numpy as np

citra = cv2.imread('daun gray.png', 0)

# Konversi ke biner
ambang, citra = cv2.threshold(citra,
                              160, 255, cv2.THRESH_BINARY)

# Balik warna
citra = 255 - citra

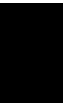
kernel= cv2.getStructuringElement(
        cv2.MORPH_RECT, (7, 7))

penutupan = cv2.morphologyEx(citra,
                             cv2.MORPH_CLOSE,
                             kernel)

blackhat = cv2.morphologyEx(citra,
                             cv2.MORPH_BLACKHAT,
                             kernel)

hasil = np.hstack((citra, penutupan, blackhat))

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

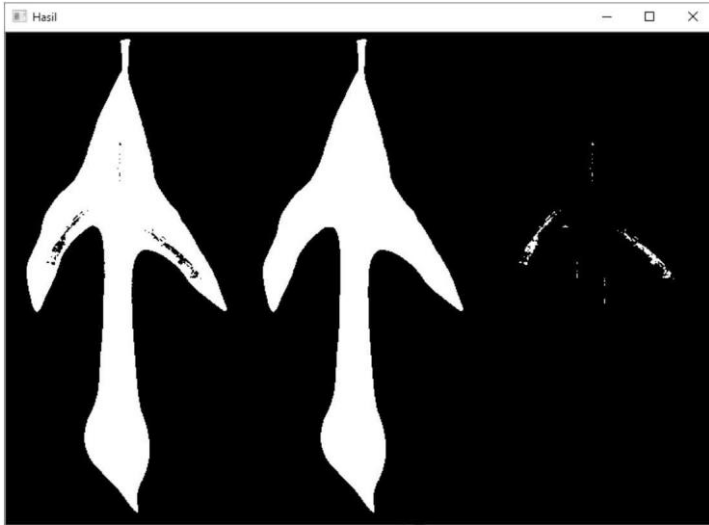


waitKey(0)

Akhir berkas

Hasilnya ditunjukkan pada Gambar 8.26. Gambar paling kiri menunjukkan objek daun yang mengandung lubang-lubang. Gambar tengah menunjukkan keadaan setelah lubang pada daun ditutup.

Gambar paling kanan menyatakan bagian yang digunakan untuk menutup lubang-lubang pada daun.



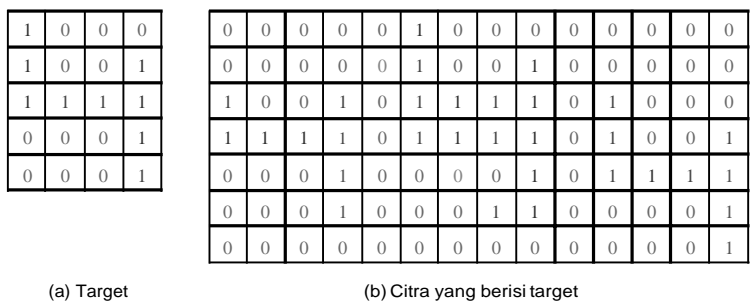
Gambar 8. 6 Efek transformasi black-hat untuk menutup Jubang pada daun

8.13 Transformasi Hit-or-Miss

Transformasi *hit-or-miss* biasa digunakan untuk menemukan pola dalam suatu citra biner. Transformasi ini melibatkan dua elemen pembentuk struktur: B1 dan B2. Dalam hal ini B2 adalah komplemen B1. Adapun operasi yang dilakukan mengandung tiga langkah seperti berikut:

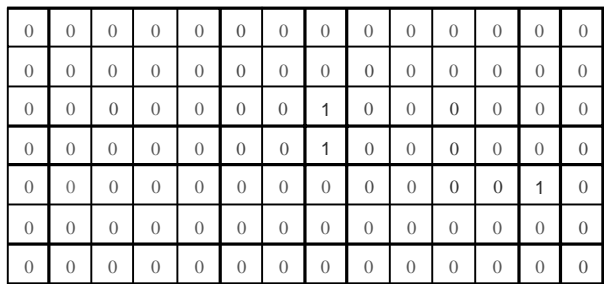
- 1) transformasi erosi pada citra dilakukan dengan menggunakan kernel B1;
- 2) transformasi erosi pada citra dilakukan dengan menggunakan kernel B2;
- 3) Hasil pada langkah 1 dan 2 dikenai operasi "dan".

Sebagai contoh, terdapat pola seperti terlihat pada Gambar 8.27(a). Target yang dikehendaki adalah menemukan pola tersebut pada citra yang terlihat pada Gambar 8.27(b). Arsiran terkiri pada citra menunjukkan target yang mirip, tetapi jelas berbeda.



Gambar 8.27 Contoh target dan citra berisi target

Langkah pertama pada transformasi *hit-or-miss* memberikan hasil pada Gambar 8.28. Nilai 1 menyatakan posisi ditemukannya target. Berdasarkan hasil tersebut, terlihat bahwa terdapat tiga nilai 1. Jadi, terdapat tiga target pada citra. Untuk mempermudah menemukan posisi elemen yang bernilai 1, masing-masing diberi arsiran agak gelap. Adapun target yang didapat diberi arsiran agak terang.



Gambar 8.28 Hasil langkah pertama pada transformasi hit-or-miss

Sebagaimana telah diungkap di depan, kernel kedua tidak lain adalah komplemen kernel pertama. Gambar 8.29 menunjukkan kernel dan target pada citra yang telah dikomplemenkan terhadap citra semula.

| | | | |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |

(a) $B_2 = B_1$

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

(b) Komplemen citra

Gambar 8. 9 Target pada citra dengan menggunakan kernel kedua yang merupakan komplemen kernel pertama

Langkah kedua pada transformasi *hit-or-miss* memberikan hasil pada Gambar 8.30. Nilai 1 menyatakan posisi ditemukannya target. Berdasarkan hasil tersebut, terlihat bahwa terdapat tiga nilai 1. Jadi, terdapat tiga target pada citra. Untuk mempermudah menemukan posisi elemen yang bernilai 1, masing-masing diberi arsiran agak gelap.

| | | | |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |

(a) $B_2 - B_1$

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(b) Kan plemencitra

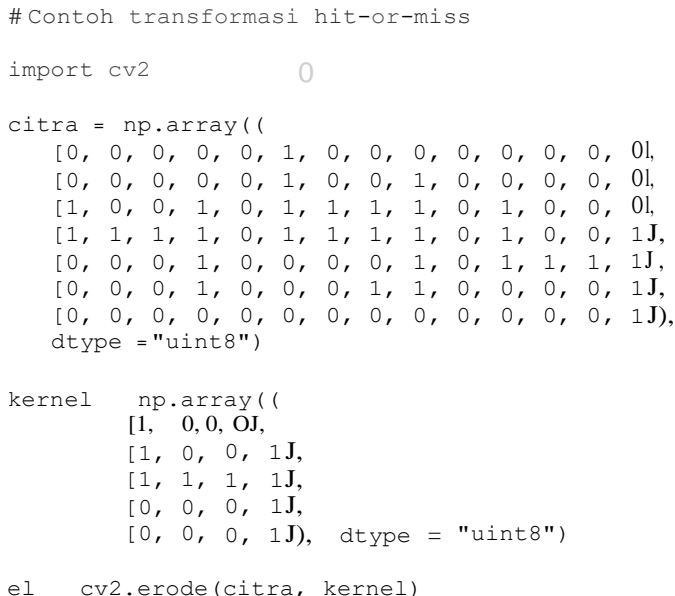
Gambar 8.30 Hasil langkah kedua pada transformasi *hit-or-miss*

Dengan melaksanakan langkah ketiga, yakni melakukan operasi "dan" terhadap hasil langkah 1 dan langkah 2, diperoleh hasil *hit-or-miss*.

discovery

Gambar 8.51 Hasil langkah ketiga pada transformasi hit-or-miss

discovery



```
e2 = cv2.erode{1 - citra, 1 - kernel)

hasil = e1 & e2
```

```
#Tampilkan hasilnya
```

Akhir berkas

Hasilnya seperti berikut:

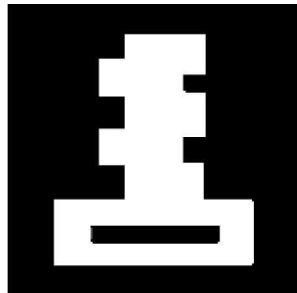
```
.....
C:\LatOpenCV>python hitormiss.py
```

```
[ [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0J
  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0J
  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0J
  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0J
  [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0J
  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0J
  [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0J]
```

```
C:\LatOpenCV>
```

Hit-or-miss dapat dimanfaatkan untuk mencari posisi kunci dalam citra.

Citra yang berisi kunci diperlihatkan pada Gambar 8.32. Adapun dua kernel yang digunakan untuk percobaan diperlihatkan pada Gambar 8.33.



Gambar 8.32 Citra kunci.png

| | | |
|---|---|---|
| 0 | 1 | 1 |
| 0 | 1 | 1 |
| 0 | 1 | 1 |

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

Gambar S.SS Dua kernel untuk percobaan



rikut menunjukkan perwujudannya.

Berkas : carikunci.py

```
# Contoh transformasi hit-or-miss
#   untuk mendapatkan tepi kunci
#   dari sisi kiri dan sisi bawah

import cv2
import numpy as np

kernelA = np.array((
    [0, 1, 1],
    [0, 1, 1],
    [0, 1, 1]), dtype="uint8")

kernelB = np.array((
    [0, 0, 0],
    [1, 1, 1],
    [1, 1, 1]), dtype="uint8")

#Baca citra biner kunci.png
citraAsal = cv2.imread("kunci.png", 0)

#Atur supaya nilai piksel berupa 0 atau 1
citra = citraAsal // 255

#Menggunakan kernelA
e1 = cv2.erode(citra, kernelA)
e2 = cv2.erode(1 - citra, 1 - kernelA)

hasilA = (e1 & e2) * 255

#Menggunakan kernelB
e1 = cv2.erode(citra, kernelB)
e2 = cv2.erode(1 - citra, 1 - kernelB)
```

```

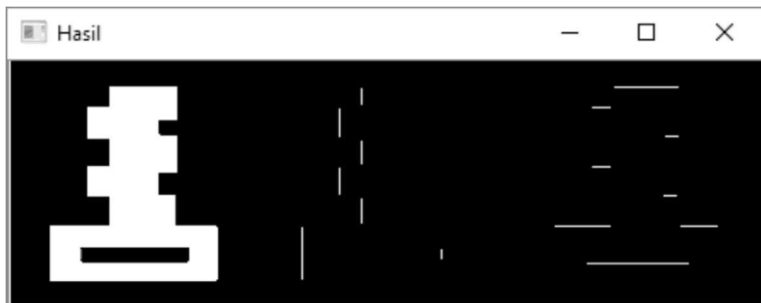
hasilB = (e1 & e2) * 255

#Tampilkan hasilnya
hasil = np.hstack((citraAsal, hasilA, hasilB))
cv2.imshow("Hasil", hasil)

```

Akhir berkas

Hasilnya ditunjukkan pada Gambar 8.34.



Gambar 8.34 Hasil penentuan posisi kunci

Elemen pembentuk struktur yang digunakan untuk melakukan transformasi *hit-or-miss* dapat melibatkan bit-bit yang disebut dengan istilah "don't care" (dampak nilai 1 atau 0 sama saja pada bit ini sama saja). Contoh:

$$B = \begin{bmatrix} 1 & x & x \\ 1 & 0 & x \\ 1 & x & x \end{bmatrix}$$

Pada contoh ini, x menyatakan "don't care" atau bebas (0 atau 1). Nah, untuk menangani kasus seperti itu, dapat dibuat transformasi *hit-or-miss* seperti berikut:

m

Berkas : crikunci.py

```
#Conteh transformasi hit-or-miss
```

```
# melibatkan "don't care"

import cv2
import numpy as np

kernelA = np.array((
    [ 0,  0,  0],
    [ 1,  1,  0],
    [-1,  1,  0]), dtype="int")

kernelB = np.array((
    [ 0,  0, -1],
    [ 1,  1,  0],
    [ 1,  1,  0]), dtype="int")

# Baca citra biner kunci.png
citra = cv2.imread("kunci.png", 0)

# Menggunakan kernelA
hasilA = cv2.morphologyEx(citra,
                           cv2.MORPH_HITMISS, kernelA)

# Menggunakan kernelB
hasilB = cv2.morphologyEx(citra,
                           cv2.MORPH_HITMISS, kernelB)

# Tampilkan hasilnya
hasil = np.hstack((citra, hasilA, hasilB))
cv2.imshow("Hasil", hasil)
waitKey()
```

Akhir berkas

Bagian yang mengandung "don't care" diisi dengan -1. Gambar 8.35 menunjukkan keadaan kedua kernel pada skrip ini secara visual.

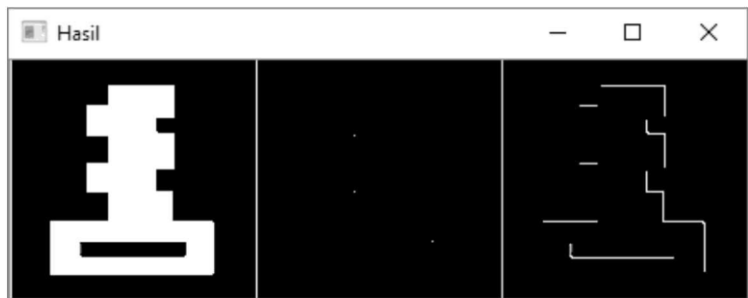
| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| | 1 | 0 | 1 | 1 |

kernelA

kernelB

Gambar 8.35 Hasil penentuan posisi kunci

Hasilnya diperlihatkan pada Gambar 8.36. Silakan untuk mengamati hasil dan hubungannya dengan kernel yang digunakan masing-masing.



Gambar 8.36 Hasil penentuan posisi kunci