

Analisi di immagini binarie

17634 | VISIONE ARTIFICIALE

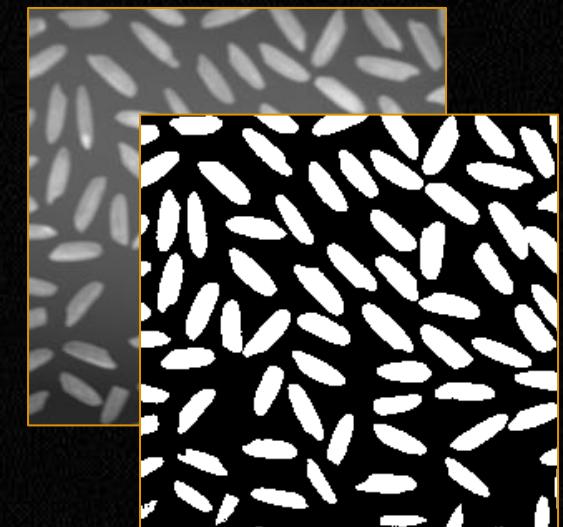


Immagini binarie

- Le immagini binarie sono molto importanti nelle applicazioni di visione artificiale: sono **alla base** di molti processi di **rilevamento** e **riconoscimento** di oggetti
 - Spesso il risultato di tecniche di elaborazione di immagini (es. il rilevamento di bordi) è costituito da immagini binarie
 - Applicazioni come il riconoscimento del testo, conteggio e misura di oggetti, etc. necessitano di lavorare su immagini binarie



Cantami o Diva del pelide
Achille l'ira funesta



Topologia digitale

- ▶ Disciplina che studia proprietà e caratteristiche topologiche delle immagini (es. componenti connesse, bordi di un oggetto, ...)
- ▶ Considera immagini binarie, ossia con due soli colori:
 - background (nel seguito assumeremo che abbia valore 0)
 - foreground (in genere 255, nel seguito assumeremo che sia diverso da 0)
- ▶ Sia F l'insieme di tutti i pixel di foreground e F^* l'insieme di quelli di background di un'immagine Img :
 - $F = \{\mathbf{p} \mid \mathbf{p} = [x, y]^\top, Img[y, x] \neq 0\}$
 - $F^* = \{\mathbf{p} \mid \mathbf{p} = [x, y]^\top, Img[y, x] = 0\}$



Metriche e distanze

- Le metriche discrete più comuni sono basate su:
 - distanza d_4 (chiamata anche City-block, Manhattan, L_1)
 - distanza d_8 (chiamata anche Chessboard, Chebyshev, L_∞)
- I vicini di un pixel \mathbf{p} sono quelli aventi distanza unitaria da \mathbf{p}
- Un percorso di lunghezza n da \mathbf{p} a \mathbf{q} è una sequenza di pixel $\{\mathbf{p}_0 = \mathbf{p}, \mathbf{p}_1, \dots, \mathbf{p}_n = \mathbf{q}\}$ tale che, in accordo con la metrica scelta, \mathbf{p}_i è un vicino di \mathbf{p}_{i+1} , $0 \leq i < n$
- Una componente连通 è un sottoinsieme di F (o di F^*) tale che, presi due qualsiasi dei suoi pixel, esiste tra questi un percorso appartenente a F (o di F^*). A seconda della metrica adottata si parla di 4-connessione o di 8-connessione

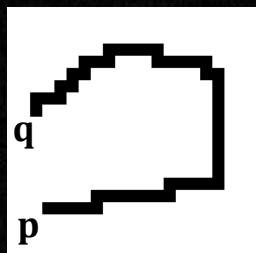
$$\mathbf{p} = [x_p, y_p]^\top, \mathbf{q} = [x_q, y_q]^\top$$

$$d_4(\mathbf{p}, \mathbf{q}) = |x_q - x_p| + |y_q - y_p|$$

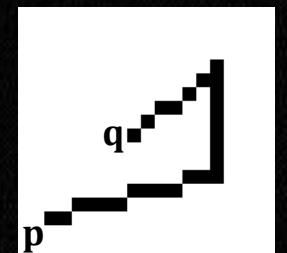
$$d_8(\mathbf{p}, \mathbf{q}) = \max\{|x_q - x_p|, |y_q - y_p|\}$$

d_4	<table border="1"> <tr><td>2</td><td>1</td><td>2</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>2</td><td>1</td><td>2</td></tr> </table>	2	1	2	1	0	1	2	1	2	d_8	<table border="1"> <tr><td>1</td><td>1</td><td>1</td></tr> <tr><td>1</td><td>0</td><td>1</td></tr> <tr><td>1</td><td>1</td><td>1</td></tr> </table>	1	1	1	1	0	1	1	1	1
2	1	2																			
1	0	1																			
2	1	2																			
1	1	1																			
1	0	1																			
1	1	1																			

Vicini di un pixel \mathbf{p} secondo le due metriche



Percorso (d_4)



Percorso (d_8)



Due componenti connesse secondo d_4

Un'unica componente连通 secondo d_8



Trasformata distanza

5

- La trasformata distanza di F rispetto a F^* è una replica di F in cui i pixel sono etichettati con il valore della loro distanza da F^*

Metrica d_4

Metrica d_8

Trasformata distanza: un semplice algoritmo sequenziale

- ▶ Si considera il caso della metrica d_4 .
- ▶ La trasformata può essere calcolata con due semplici scansioni dell'immagine:
 - una **diretta** (dall'alto verso il basso da sinistra verso destra),
 - una **inversa** (dal basso verso l'alto da destra verso sinistra).
 - Durante le scansioni i pixel di F vengono trasformati; i pixel di F^* sono posti a zero (distanza nulla).
- ▶ $\forall \mathbf{p} \in F$, il valore trasformato $Img'[\mathbf{p}]$ è calcolato come segue:
 - **Scansione diretta:** la distanza viene propagata dai vicini in alto a sinistra (che sono già stati considerati).

$$Img'[\mathbf{p}] = \min\{Img'[\mathbf{p}_O], Img'[\mathbf{p}_N]\} + 1$$
 - **Scansione inversa:** la distanza viene aggiornata tenendo conto anche dei percorsi verso il basso e a destra

$$Img'[\mathbf{p}] = \min\{Img'[\mathbf{p}_E] + 1, Img'[\mathbf{p}_S] + 1, Img'[\mathbf{p}]\}$$

	\mathbf{p}_N	
\mathbf{p}_O	\mathbf{p}	\mathbf{p}_E
	\mathbf{p}_S	



Trasformata distanza: esempio di calcolo sequenziale

7



Trasformata distanza in OpenCV

8

```
img = cv.imread('esempi/mario-bw.png', cv.IMREAD_GRAYSCALE)

dt4 = cv.distanceTransform(img, cv.DIST_L1, 3)

dt8 = cv.distanceTransform(img, cv.DIST_C, 3)
```



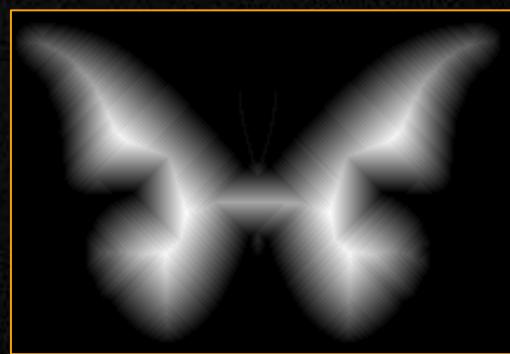
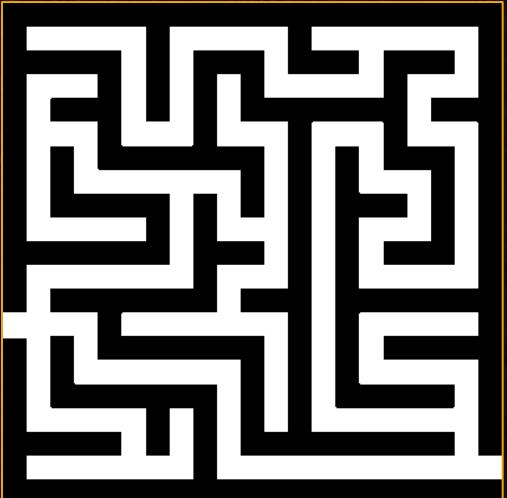
Applicazioni della trasformata distanza

- ▶ Può essere utilizzata per la **misurazione geometrica** di: lunghezze, spessori, ...
- ▶ Produce massimi locali di intensità in corrispondenza degli assi mediani dell'oggetto (**scheletro**).
 - Nel caso di oggetti allungati la trasformata può essere utilizzata per algoritmi di assottigliamento o thinning.
- ▶ Applicazioni più avanzate della trasformata (o di sue varianti) includono:
 - Template matching
 - Robotica (ricerca direzione di movimento ottimale, aggiramento ostacoli, ...)



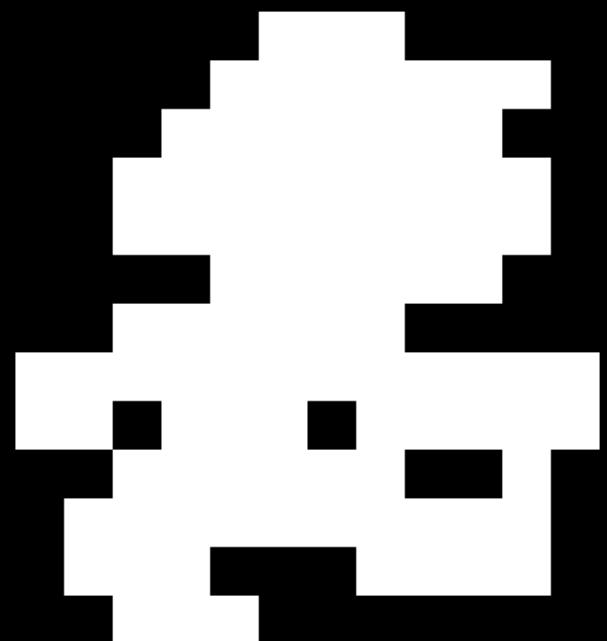
Trasformata distanza: esempi

10



Estrazione del contorno

- ▶ Il contorno di F è costituito dalla sequenza ordinata dei pixel che hanno distanza unitaria da F^* secondo la metrica adottata per F^* .
- ▶ Per l'estrazione del contorno di un oggetto viene normalmente impiegata una tecnica di **inseguimento** che percorre il bordo nella stessa direzione, fino a tornare al pixel di partenza.
- ▶ Il contorno può essere memorizzato in vari modi:
 - Semplice lista ordinata dei pixel
(OpenCV: cv.CHAIN_APPROX_NONE);
 - Lista dei vertici dei **segmenti** che lo compongono
(OpenCV: cv.CHAIN_APPROX_SIMPLE);
 - Approssimandolo con una serie di **curve**
(OpenCV: cv.CHAIN_APPROX_TC89_L1 e cv.CHAIN_APPROX_TC89_KCOS)



Estrazione e disegno del contorno in OpenCV

```
esempi = ('labirinto', 'leaf', 'butterfly', 'bat')

for n in esempi:
    img = cv.imread('esempi/' + n + '.png', cv.IMREAD_GRAYSCALE)

    c, _ = cv.findContours(img, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)

    res = cv.drawContours(cv.cvtColor(img, cv.COLOR_GRAY2BGR), c, -1, (255,0,0), 2)
```

Un esempio completo di estrazione contorni in OpenCV

13

```
img = cv.imread('esempi/donuts.png')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
# Binarizza con soglia 253 (lo sfondo è bianco)
_, binarized = cv.threshold(gray, 253, 255, cv.THRESH_BINARY_INV)

# La funzione restituisce anche una struttura dati ad albero che contiene eventuali
# relazioni fra contorni (un contorno "padre" contiene uno o più "figli")
contours, tree = cv.findContours(binarized, cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)

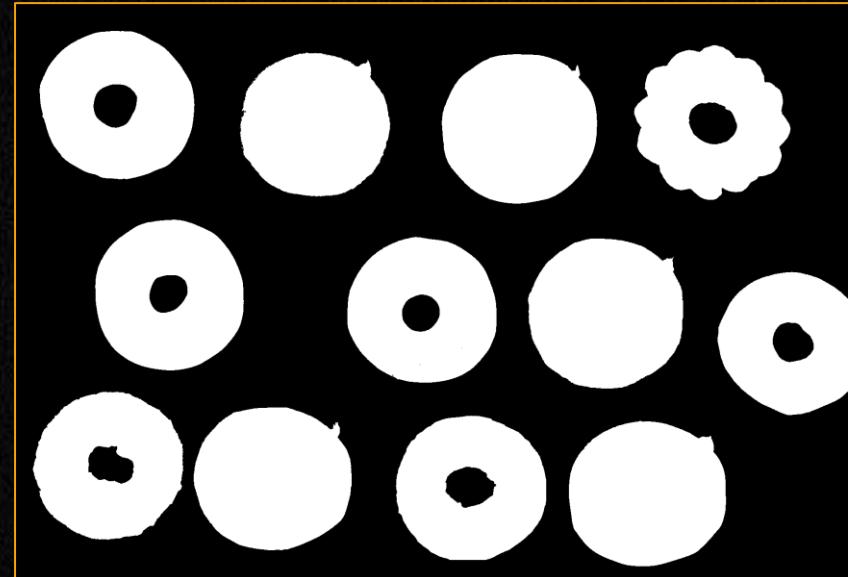
# Disegna tutti i contorni trovati
allc = cv.drawContours(img.copy(), contours, -1, (255,0,0), 5)

# Esclude i contorni con area piccola e abbina due flag booleani per sapere se
# il contorno ha un padre e/o figli
info = [(c, tree[0,i,2]>=0, tree[0,i,3]>=0) for i, c in enumerate(contours)
         if cv.contourArea(c)>100]
# Colora diversamente i contorni selezionati che hanno "padri" o "figli"
res = img.copy()
for c, f, p in info:
    cv.drawContours(res, [c], -1, (0,0,255) if p else (255,0,0) if f else (0,255,0), 5)
```



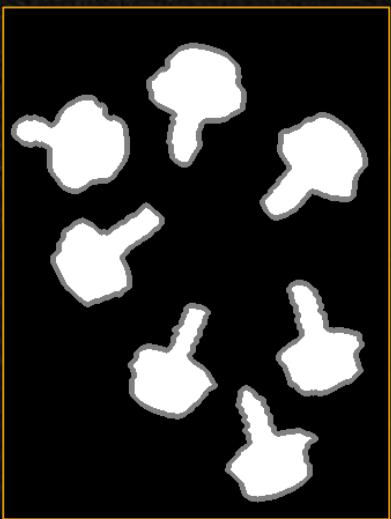
Un esempio completo di estrazione contorni in OpenCV: risultati

14

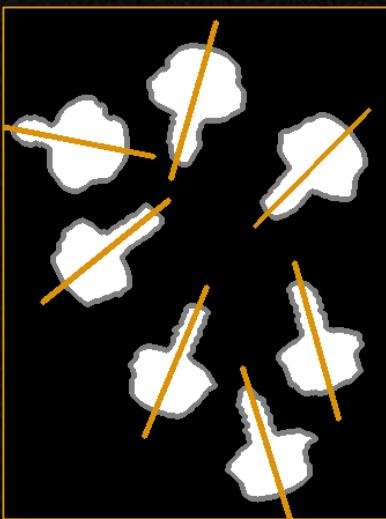


OpenCV: alcune utili funzioni che si possono applicare ai contorni

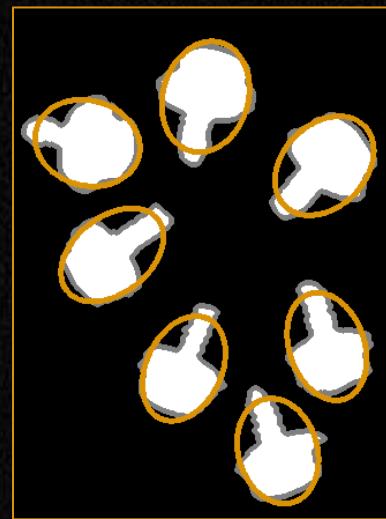
15



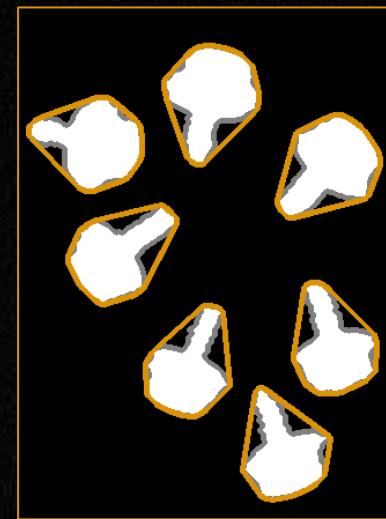
Contorni



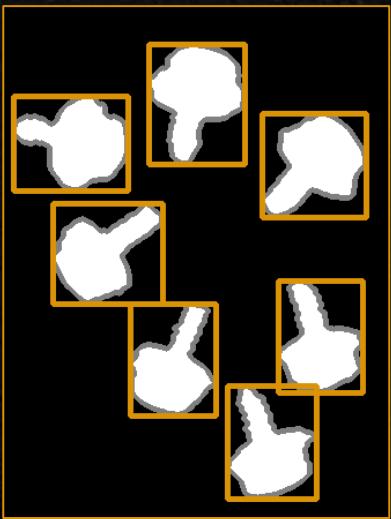
`cv.fitLine()`



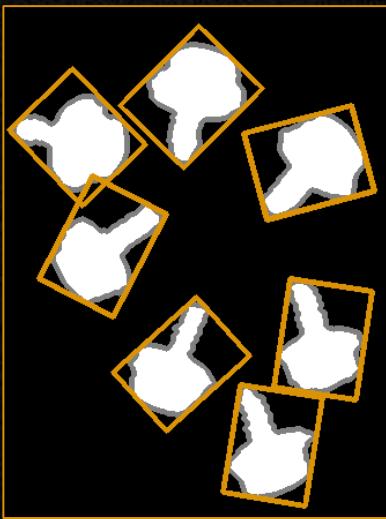
`cv.fitEllipse()`



`cv.convexHull()`



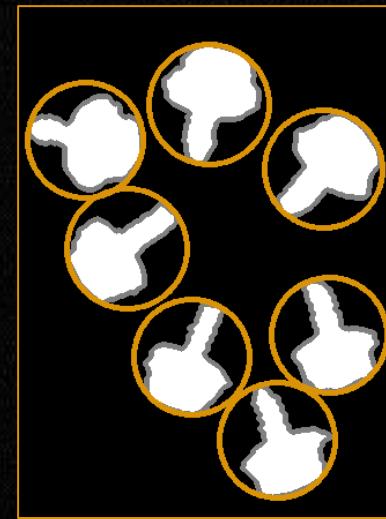
`cv.boundingRect()`



`cv.minAreaRect()`



`cv.minEnclosingTriangle()`



`cv.minEnclosingCircle()`



- ▶ Procedura molto importante nell'analisi di immagini binarie:
 - Individuare automaticamente le diverse componenti connesse in un'immagine, assegnando loro etichette (tipicamente numeriche)
- ▶ Algoritmo comunemente utilizzato:
 1. Si scorre l'immagine e, per ogni pixel di foreground **p**, si considerano i pixel di foreground vicini già visitati:
 - se nessuno è etichettato, si assegna una nuova etichetta a **p**;
 - se uno è etichettato, si assegna a **p** la stessa etichetta;
 - se più di uno è etichettato, si assegna a **p** una delle etichette e si annotano le equivalenze.
 2. Si definisce un'unica etichetta per ogni insieme di etichette marcate come equivalenti e si effettua una seconda scansione assegnando le etichette finali



Algoritmo di etichettatura: fase 1

17

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	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	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	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	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	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	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	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



Algoritmo di etichettatura: fase 2

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	1	0	0	2	2	0	0	0	3	3	3	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	2	2	0	0	3	3	0	3	3	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	2	2	0	3	3	0	0	0	0	3	3	0	0	0	0	0	0	0	0	0
0	1	1	0	0	2	2	0	3	3	0	0	0	0	3	3	0	0	0	0	0	0	0	0	0
0	1	1	0	0	2	2	0	3	3	3	3	3	3	3	3	3	0	0	0	0	0	0	0	0
0	1	1	0	0	2	2	0	3	3	3	3	3	3	3	3	3	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0	3	3	3	3	3	3	3	3	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	3	3	0	0	0	0	3	3	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	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	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	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	0	0	0	0	0	0	0	0	0	0	0	0	0

1=2

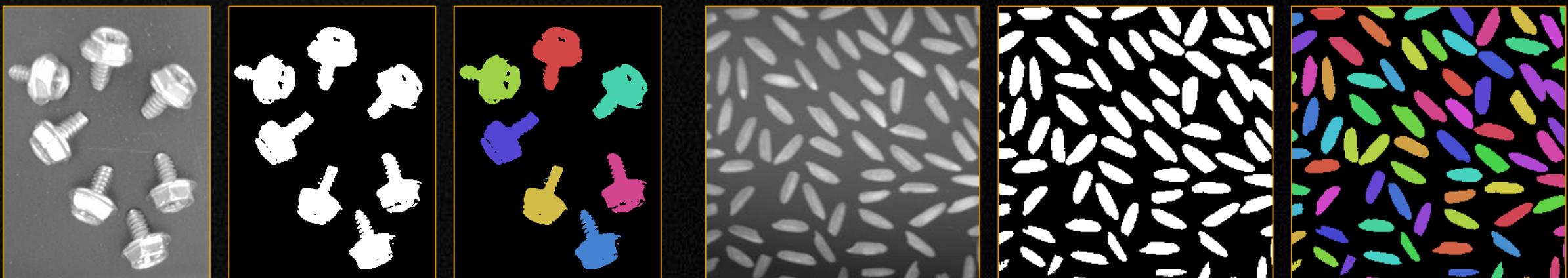


Etichettatura delle componenti connesse in OpenCV

19

```
img = cv.imread(name, cv.IMREAD_GRAYSCALE)
_, bw = cv.threshold(img, thr, 255, cv.THRESH_BINARY)

n, cc = cv.connectedComponents(bw)
# n è il numero di componenti connesse
# cc è un'immagine di interi in cui il valore di ogni pixel
# è l'indice della componente connessa a cui appartiene
print(f'Componenti connesse: {n}')
```



OpenCV: componenti connesse con statistiche

20

```
img = cv.imread('esempi/bolts.png', cv.IMREAD_GRAYSCALE)
_, bw = cv.threshold(img, 152, 255, cv.THRESH_BINARY)

# La funzione cv.connectedComponentsWithStats, oltre all'immagine
# con le etichette, restituisce, per ogni componente connessa,
# le coordinate del baricentro, l'area e il bounding box.
n, cc, stats, centroids = cv.connectedComponentsWithStats(bw)

@interact(i=(0,n-1))
def show_stats(i=0):
    res = cv.cvtColor(img, cv.COLOR_GRAY2BGR)
    res[cc==i,1] = res[cc==i,1]//4 + 255*3//4

    x, y = stats[i,cv.CC_STAT_LEFT], stats[i,cv.CC_STAT_TOP]
    w, h = stats[i,cv.CC_STAT_WIDTH], stats[i,cv.CC_STAT_HEIGHT]
    area = stats[i,cv.CC_STAT_AREA]

    cv.rectangle(res, (x,y), (x+w,y+h), (255,0,0), 3)

    va.center_text(res, f'A[{i}]={area}', centroids[i].astype(int), (0,0,255))
```



- ▶ Tecnica di analisi ed elaborazione di immagini binarie derivata dalla teoria degli insiemi
- ▶ Fornisce strumenti utili a:
 - estrarre informazioni per rappresentare e descrivere la forma (contorno, scheletro, ...);
 - rimuovere particolari irrilevanti mantenendo le informazioni importanti sulla forma degli oggetti.
- ▶ Elemento strutturante
 - Piccola immagine binaria (es. 3x3 o 5x5) che viene utilizzata come parametro nelle operazioni morfologiche (anch'essa considerata un insieme di pixel di foreground)
 - Tipicamente quadrata (lato dispari) e centrata rispetto all'origine



Morfologia matematica: notazione

- ▶ Sia F l'insieme di tutti i pixel di foreground e F^* l'insieme di quelli di background di un'immagine Img :
 - $F = \{\mathbf{p} \mid \mathbf{p} = [x, y]^\top, Img[y, x] \neq 0\}$
 - $F^* = \{\mathbf{p} \mid \mathbf{p} = [x, y]^\top, Img[y, x] = 0\}$
- ▶ Operazioni derivate dalla teoria degli insiemi
 - Intersezione e unione: $A \cap B = \{\mathbf{p} \mid \mathbf{p} \in A \wedge \mathbf{p} \in B\}$, $A \cup B = \{\mathbf{p} \mid \mathbf{p} \in A \vee \mathbf{p} \in B\}$
 - Complemento: $A^c = \{\mathbf{p} \mid \mathbf{p} \notin A\}$
 - Differenza: $A - B = \{\mathbf{p} \mid \mathbf{p} \in A \wedge \mathbf{p} \notin B\} = A \cap B^c$
 - Traslazione rispetto a un punto \mathbf{q} : $(A)_{\mathbf{q}} = \{\mathbf{p} \mid \mathbf{p} = \mathbf{a} + \mathbf{q}, \mathbf{a} \in A\}$
 - Riflessione rispetto all'origine: $A^r = \{\mathbf{p} \mid \mathbf{p} = -\mathbf{q}, \mathbf{q} \in A\}$



Dilatazione ed erosione

23

- ▶ La maggior parte delle operazioni morfologiche si basano su due soli operatori:

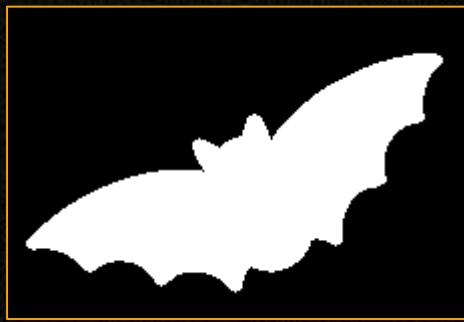
- Dilatazione: $F \oplus S$
- Erosione: $F \ominus S$



F



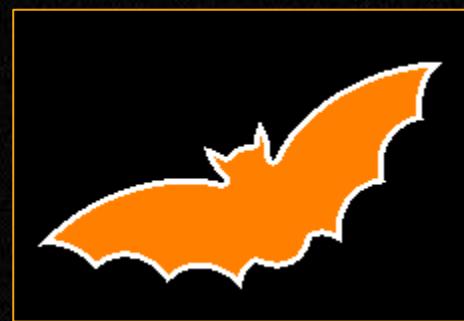
S



$F \oplus S$



$F \ominus S$

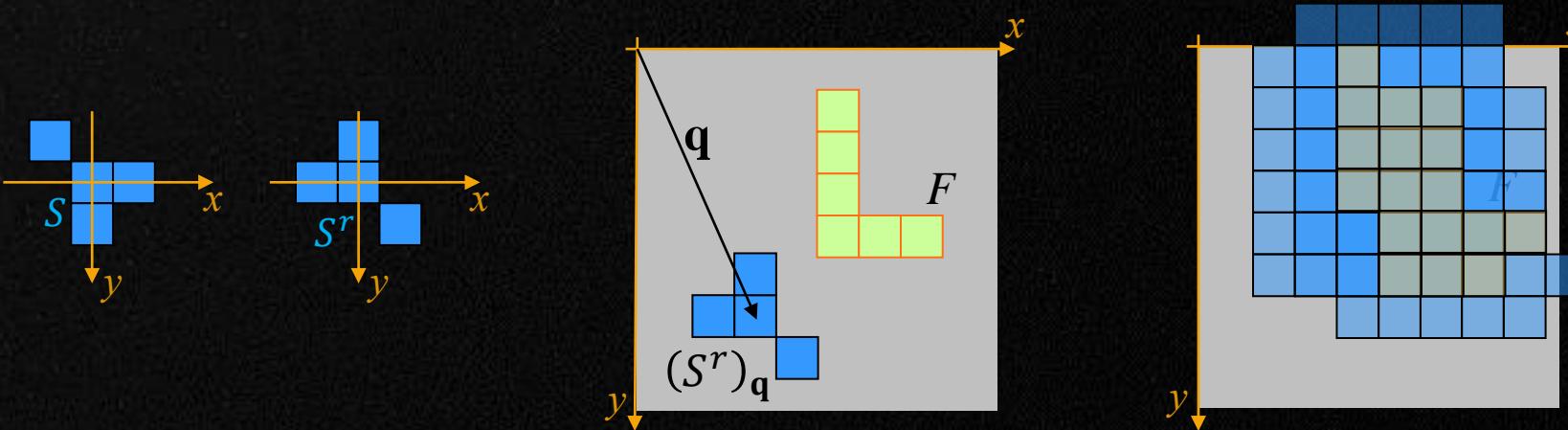


Dilatazione

24

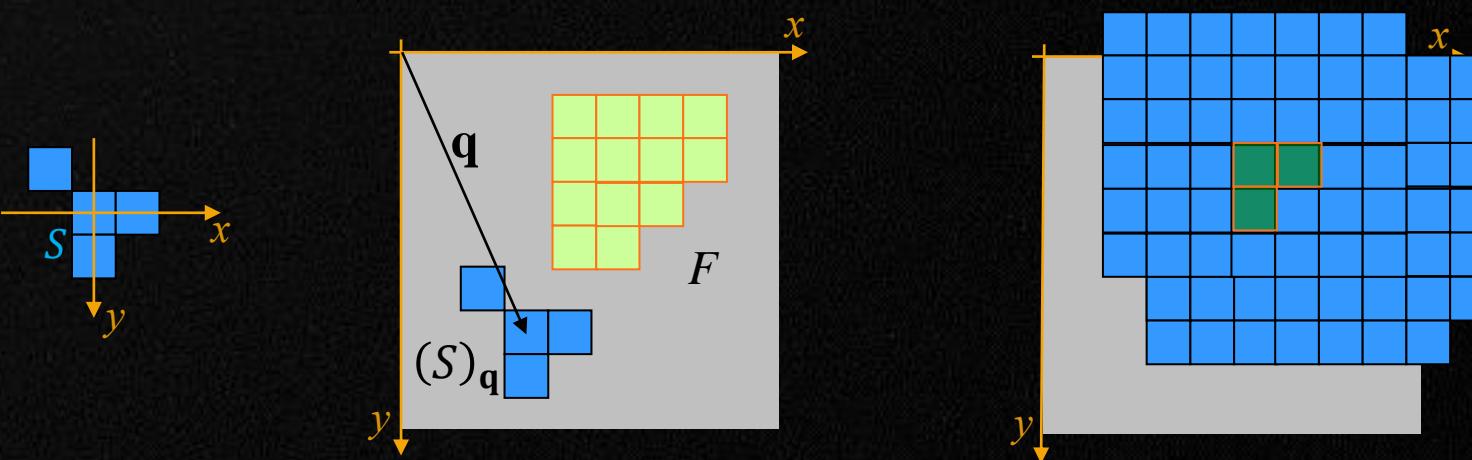
- ▶ La nuova immagine è l'insieme dei pixel tali che, traslando in essi S^r , almeno uno dei suoi elementi è sovrapposto a F

$$F \oplus S = \{\mathbf{q} | (S^r)_{\mathbf{q}} \cap F \neq \emptyset\}$$



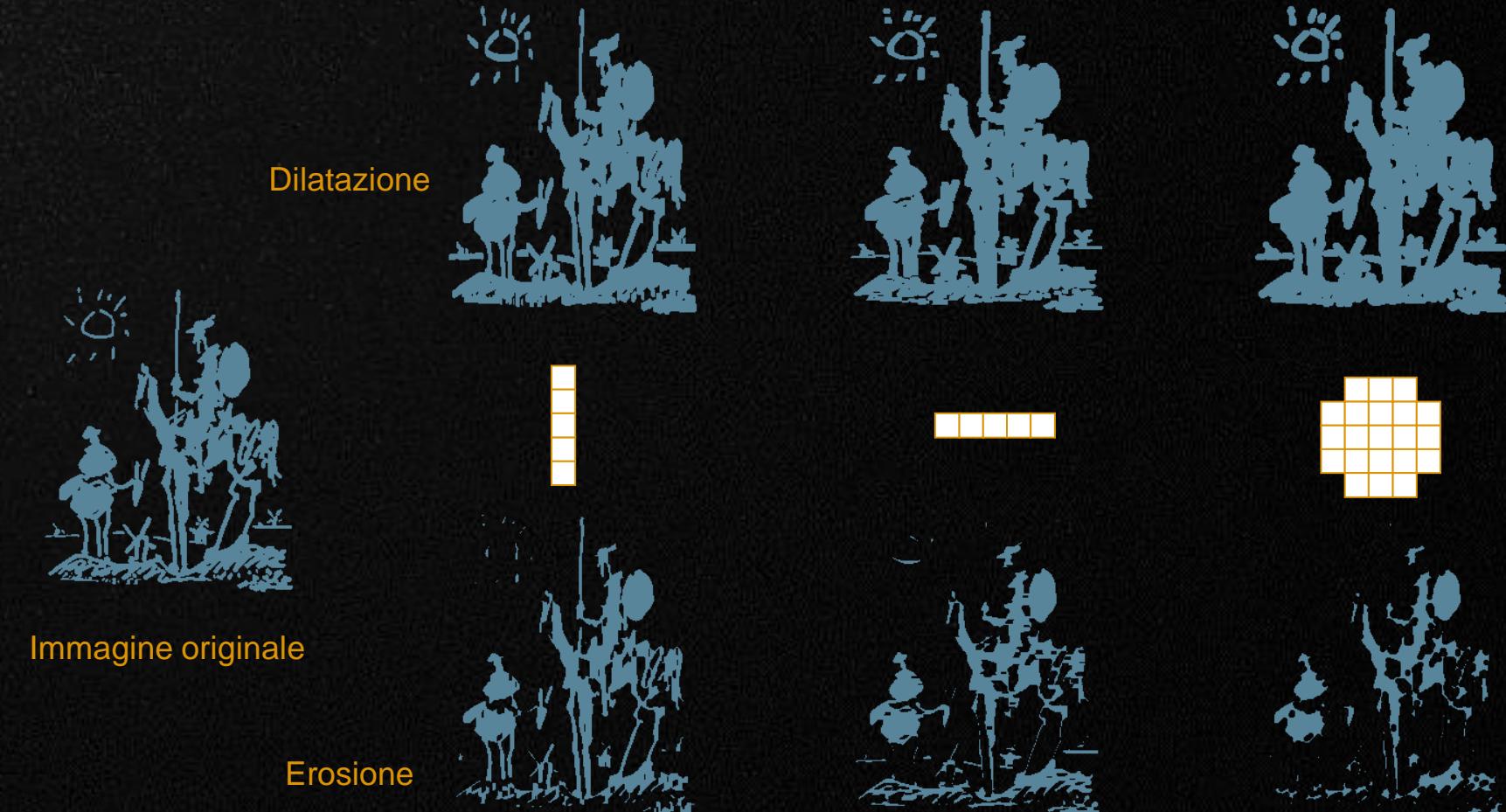
- ▶ La nuova immagine è l'insieme dei pixel tali che, traslando in essi S , l'intero elemento strutturante è contenuto in F

$$F \ominus S = \{\mathbf{q} | (S)_{\mathbf{q}} \subseteq F\}$$



Dilatazione ed erosione: esempi

26



Dilatazione ed erosione in OpenCV

27

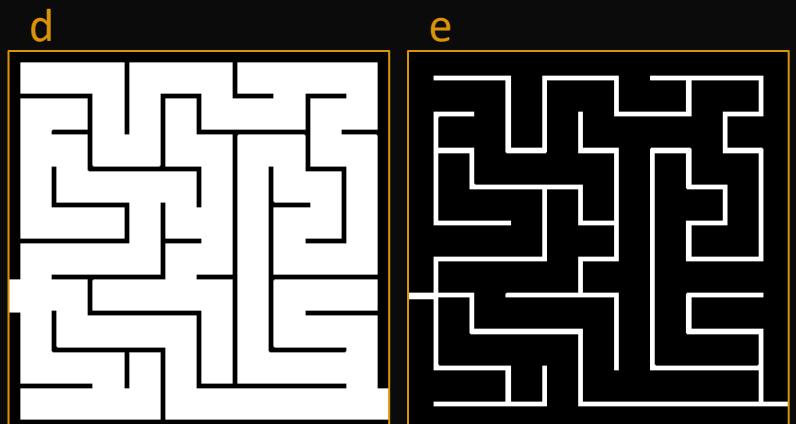
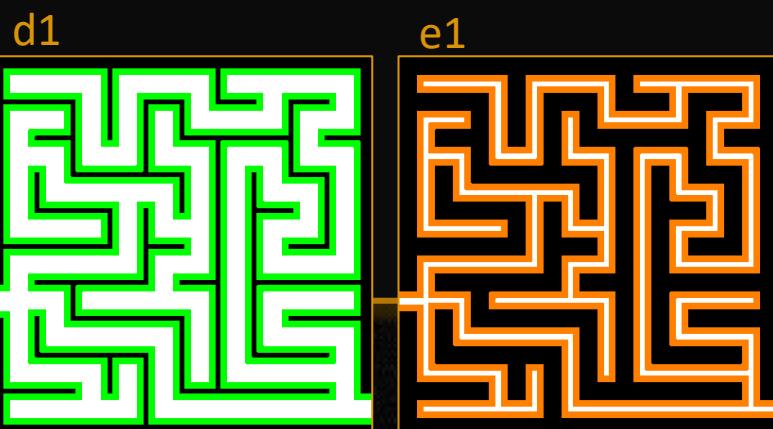
```
img = cv.imread('esempi/labirinto.png', cv.IMREAD_GRAYSCALE)

# Elemento strutturante: quadrato 15x15
se = cv.getStructuringElement(cv.MORPH_RECT, (15,15))

d = cv.morphologyEx(img, cv.MORPH_DILATE, se)
e = cv.morphologyEx(img, cv.MORPH_ERODE, se)

# Colora i pixel aggiunti dalla dilatazione
d1 = cv.cvtColor(d, cv.COLOR_GRAY2BGR)
d1[d!=img] = (0,255,0)

# Colora i pixel rimossi dall'erosione
e1 = cv.cvtColor(e, cv.COLOR_GRAY2BGR)
e1[e!=img] = (0,128,255)
```



► Apertura

$$F \circ S = (F \ominus S) \oplus S$$

- Erosione seguita da dilatazione
- Separa oggetti debolmente connessi e rimuove regioni piccole

► Chiusura

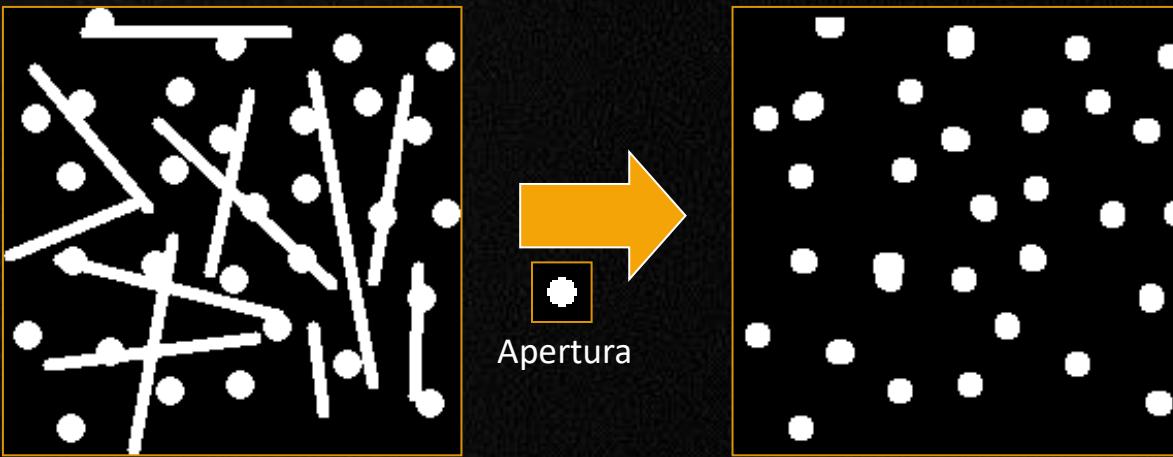
$$F \bullet S = (F \oplus S) \ominus S$$

- Dilatazione seguita da erosione
- Riempie buchi e piccole concavità e rafforza la connessione di regioni unite debolmente

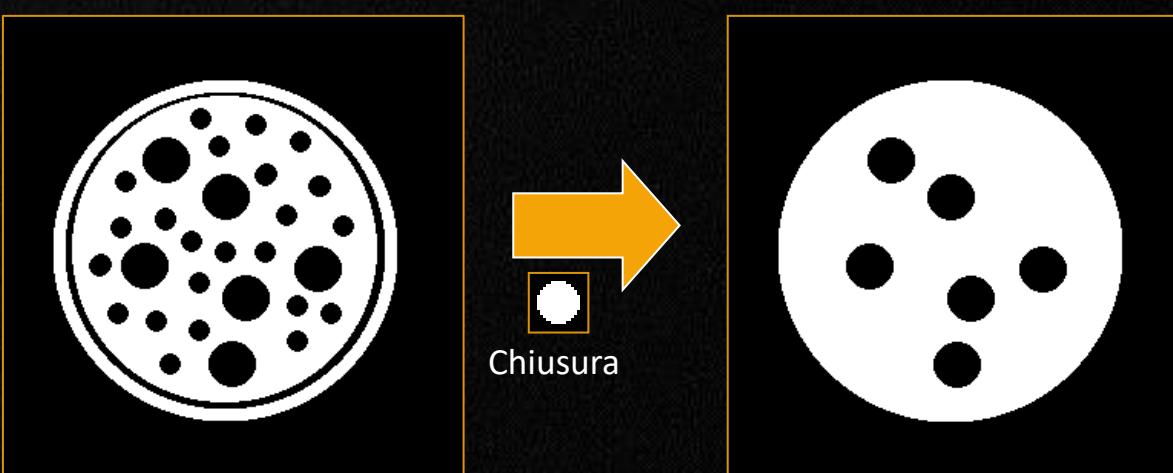


Apertura e chiusura: esempi

29



Apertura



Chiusura



Apertura e chiusura: altri esempi

Apertura



Chiusura



Apertura e chiusura in OpenCV

```

img = cv.imread('esempi/bolts.png', cv.IMREAD_GRAYSCALE)
_, bw = cv.threshold(img, 140, 255, cv.THRESH_BINARY)

# L'immagine appena binarizzata presenta due problemi:
# - Alcune piccole componenti connesse dovute al rumore
# - Alcuni "buchi" all'interno degli oggetti

s = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5))

# L'apertura permette di risolvere il primo problema
res1 = cv.morphologyEx(bw, cv.MORPH_OPEN, s)

# La chiusura permette di risolvere il secondo
res2 = cv.morphologyEx(res1, cv.MORPH_CLOSE, s)

res = img.copy()
res[res2==0]=0

```



► Immagini binarie

- Foreground e background, immagini come insiemi
- Metriche e distanze, percorsi e componenti connesse

► Trasformata distanza

- Algoritmo, funzione OpenCV, applicazioni

► Estrazione dei contorni

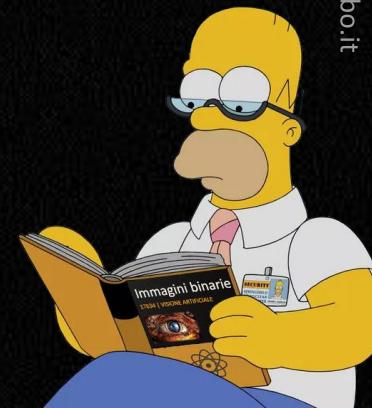
- Funzione OpenCV, gerarchia dei contorni: contorni esterni e interni

► Etichettatura delle componenti connesse

- Algoritmo, funzione OpenCV, statistiche sulle componenti connesse

► Morfologia matematica

- Dilatazione, erosione, apertura e chiusura
- Funzioni OpenCV e applicazioni



- ▶ Nel libro [Szeliski, Computer Vision: Algorithms and Applications, 2022]:
 - Sezione 3.3.3: Binary image processing
- ▶ Documentazione OpenCV del modulo «Structural Analysis and Shape Descriptors»:
 - https://docs.opencv.org/master/d3/dc0/group__imgproc__shape.html
- ▶ Documentazione OpenCV del modulo «Image processing - Image Filtering»:
 - https://docs.opencv.org/master/d4/d86/group__imgproc__filter.html

