

Rilevare il movimento nei video

17634 | VISIONE ARTIFICIALE



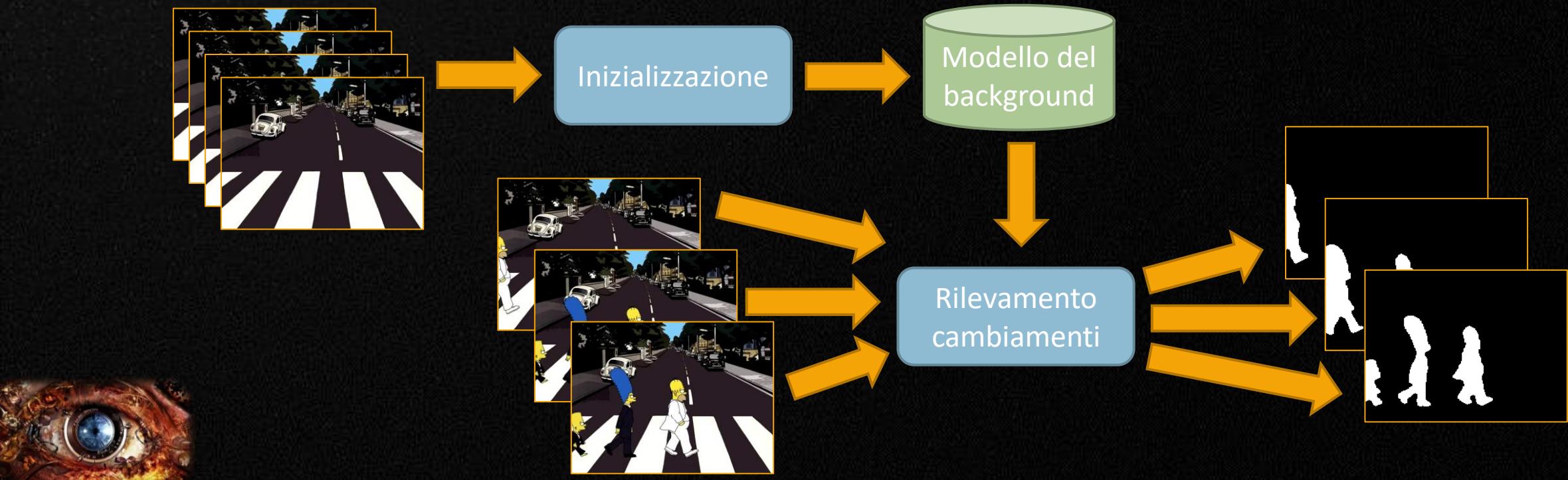
Obiettivo

- ▶ Rilevamento di oggetti in movimento all'interno di sequenze video con telecamera fissa
- ▶ Primo e fondamentale passo di molteplici applicazioni, fra cui:
 - Video sorveglianza e analisi video
 - Navigazione e guida autonoma
 - Osservazione di animali nel loro ambiente
- ▶ Risultato atteso:
 - Per ogni frame una immagine binaria in cui i pixel degli oggetti in movimento sono marcati come foreground, mentre tutto il resto è marcato come background



Approccio generale

1. Viene costruito un **modello del background** (tipicamente a partire da un certo numero di frame iniziali)
2. In ogni nuovo frame, pixel (o gruppi di pixel) che differiscono in modo significativo da quanto atteso in base al modello del background sono considerati foreground (oggetti in movimento)
3. Il modello del background può essere poi periodicamente aggiornato in base ai nuovi frame

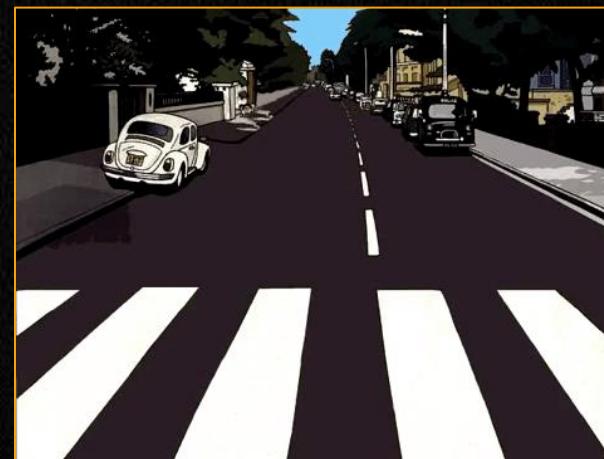


Differenza tra frame successivi

- ▶ Il metodo più semplice:

$$|frame_i - frame_{i-1}| > thr$$

- ▶ Il modello del background è il frame precedente
- ▶ Può essere efficace solo in particolari condizioni di velocità degli oggetti e di frame rate
- ▶ Molto sensibile alla scelta della soglia thr



Estrazione dei frame da un video e calcolo differenza in OpenCV

5

```
# Costruisce un oggetto VideoCapture che permette di estrarre i frame da un file video
cap = cv.VideoCapture('esempi/simpsons_abbey_road.mp4')

thr = 50 # La soglia che si intende utilizzare
# Per ogni frame, convertito in grayscale, crea una maschera
# dei pixel la cui differenza dal precedente è maggiore di thr
fm = [] # Lista di tuple (frame, mask)
prev = None
while True:
    ret, frame = cap.read()
    if not ret: # ret è False quando non ci sono più frame
        break
    f = cv.cvtColor(frame, cv.COLOR_BGR2GRAY).astype(np.int16)
    if prev is not None:
        mask = np.zeros_like(f)
        mask[np.abs(f - prev) > thr] = 255 # A 255 i pixel la cui differenza supera thr
        fm.append( (frame, mask) ) # Lista di tuple (frame, mask)
    prev = f

cap.release() # Rilascia le risorse (in questo caso il file)
```



Variazioni del background

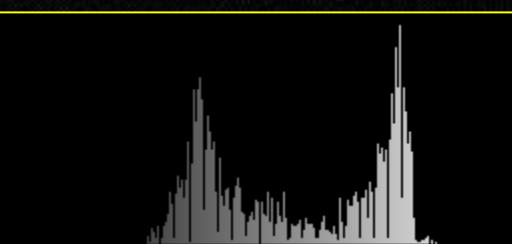
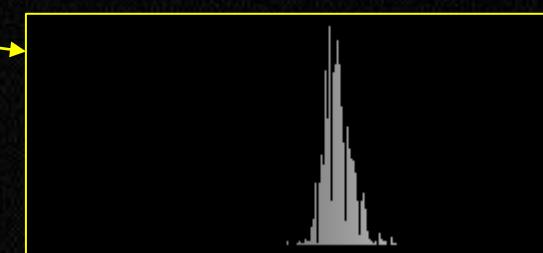
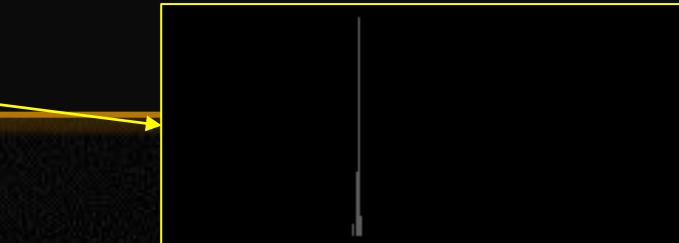
- ▶ Ombre
- ▶ Riflessi
- ▶ Cambi di illuminazione
- ▶ Movimento della superficie dell'acqua
- ▶ Nuvole, foglie o altri elementi mossi dal vento
- ▶ Pioggia, neve
- ▶ Rumore digitale nelle immagini
- ▶ Etc.



Esaminiamo la distribuzione dei valori di alcuni pixel di background

7

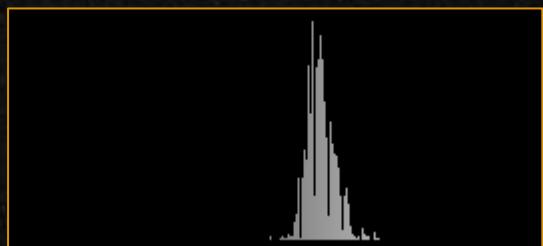
```
# Calcolo dell'istogramma della luminosità di un pixel (x,y) lungo un video
gray_frames = [cv.cvtColor(frame, cv.COLOR_BGR2GRAY) for frame in frames]
frames3d = np.dstack(gray_frames) # Tensore 3D con tutti i frame del video
x, y = 486, 305
hist, _ = np.histogram(frames3d[y,x],256,(0,255)) # Slicing per prendere i valori
```



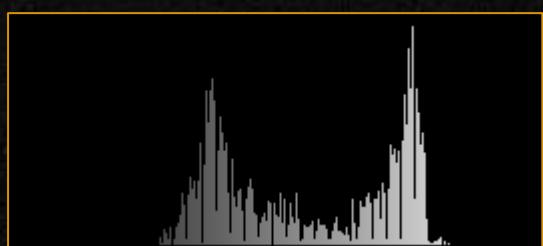
Quale modello per i pixel di background?



Quando la variabilità è contenuta, un semplice valore (il primo frame, il frame precedente, oppure la media degli ultimi n frame) può essere sufficiente.



Con una distribuzione di questo tipo, può essere utile modellare ciascun pixel con una **gaussiana**, stimando media e varianza dai precedenti n frame.



In caso di variazioni "periodiche" (nel video del lucido precedente il pixel era caratterizzato da un'alternanza di colori chiari e scuri dovuti all'ombra di alberi mossi dal vento) la **distribuzione** può essere **multimodale**: in tal caso può essere utile modellare ciascun pixel con una *Mixture of Gaussians (MoG)*.



L'algoritmo BackgroundSubtractorMOG2 in OpenCV

```
mog = cv.createBackgroundSubtractorMOG2(detectShadows=False)

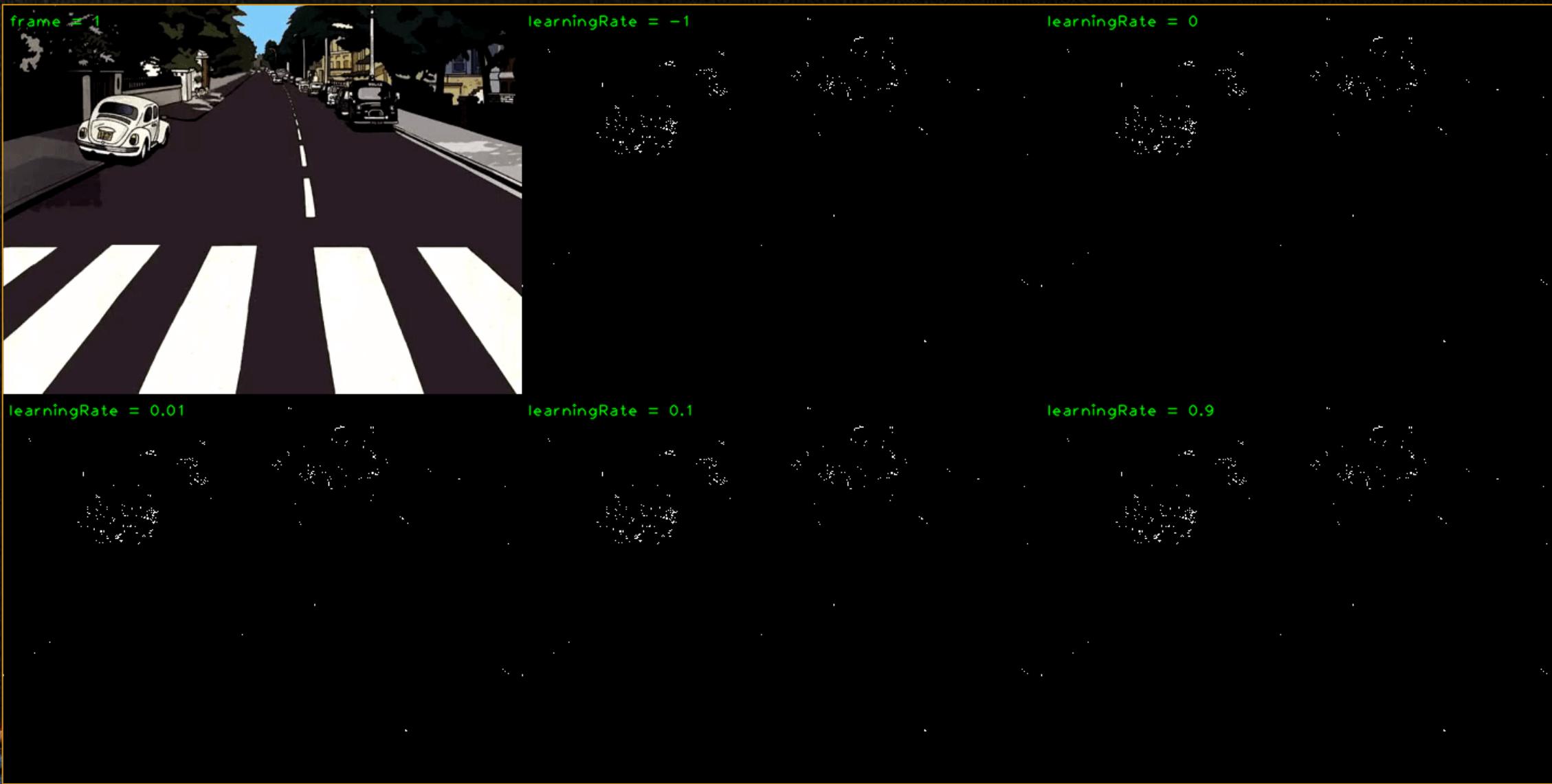
cap = cv.VideoCapture('esempi/simpsons_abby_road.mp4')
fm = [] # Lista di tuple (frame, mask)
while True:
    ret, frame = cap.read()
    if not ret:
        break
    f = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    mask = mog.apply(f) # Utilizza l'algoritmo MOG2 sui frame grayscale
    fm.append( (frame, mask) )
cap.release()
```

- ▶ OpenCV integra un algoritmo basato su **Mixture of Gaussians**, descritto in:
 - Z. Zivkovic and F. van der Heijden, Efficient adaptive density estimation per image pixel for the task of background subtraction, Pattern recognition letters, 2006.
 - Z. Zivkovic, Improved adaptive gaussian mixture model for background subtraction, ICPR 2004.
- ▶ Al metodo **apply()** è possibile passare un parametro **learningRate** fra 0 e 1 che controlla in che misura il modello del background è aggiornato dal nuovo frame: 0 significa che il modello non viene aggiornato, 1 che il modello è reinizializzato dal nuovo frame. Se non specificato o negativo, è scelto in automatico.



BackgroundSubtractorMOG2: esempi di funzionamento

10



Miglioramento immagine binaria con le tecniche viste in precedenza (1)

11

```
# Per la maschera binaria è normalmente sufficiente una risoluzione più bassa.  
# Ridurre la risoluzione del frame prima di passarlo all'algoritmo di estrazione del  
# foreground riduce la complessità computazionale dell'algoritmo stesso e di  
# eventuali successive altre operazioni sulla maschera.  
MASK_SF = 2.0  
#...  
mask = mog.apply(cv.resize(frame, None, fx = 1/MASK_SF, fy = 1/MASK_SF))  
# ... eventuali modifiche alla maschera (a risoluzione ridotta) ...  
# Riscala la maschera alle dimensioni originali  
mask = cv.resize(mask, None, fx = MASK_SF, fy = MASK_SF, interpolation = cv.INTER_NEAREST)  
fm.append( (frame, mask) )  
# ...
```



Frame



Maschera a risoluzione originale

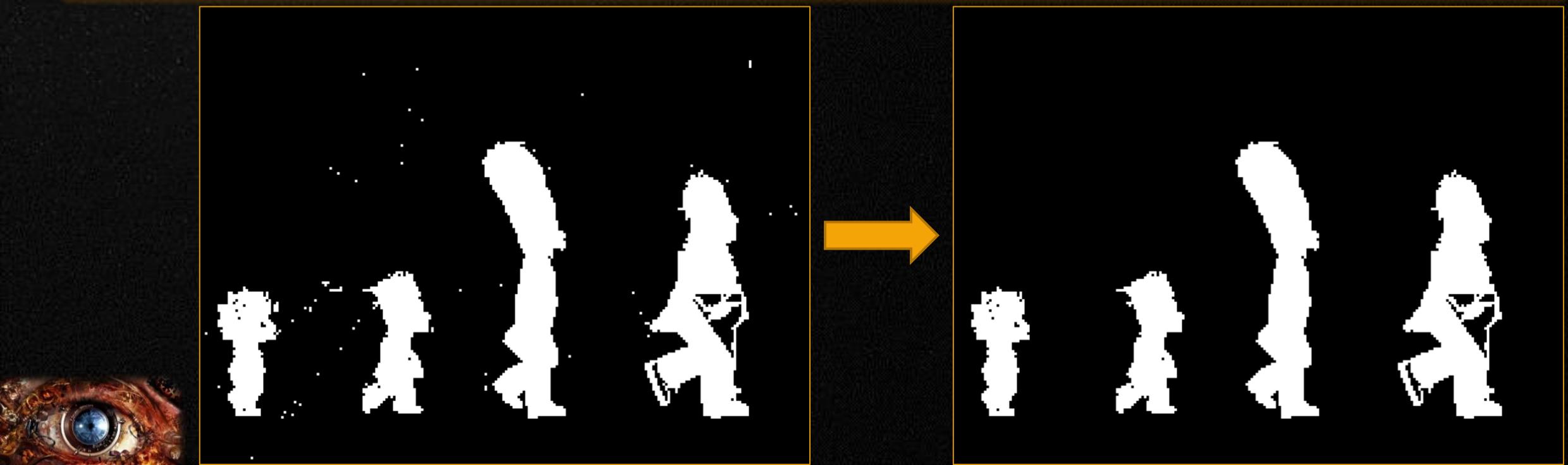


Maschera a risoluzione dimezzata

Miglioramento immagine binaria con le tecniche viste in precedenza (2)

12

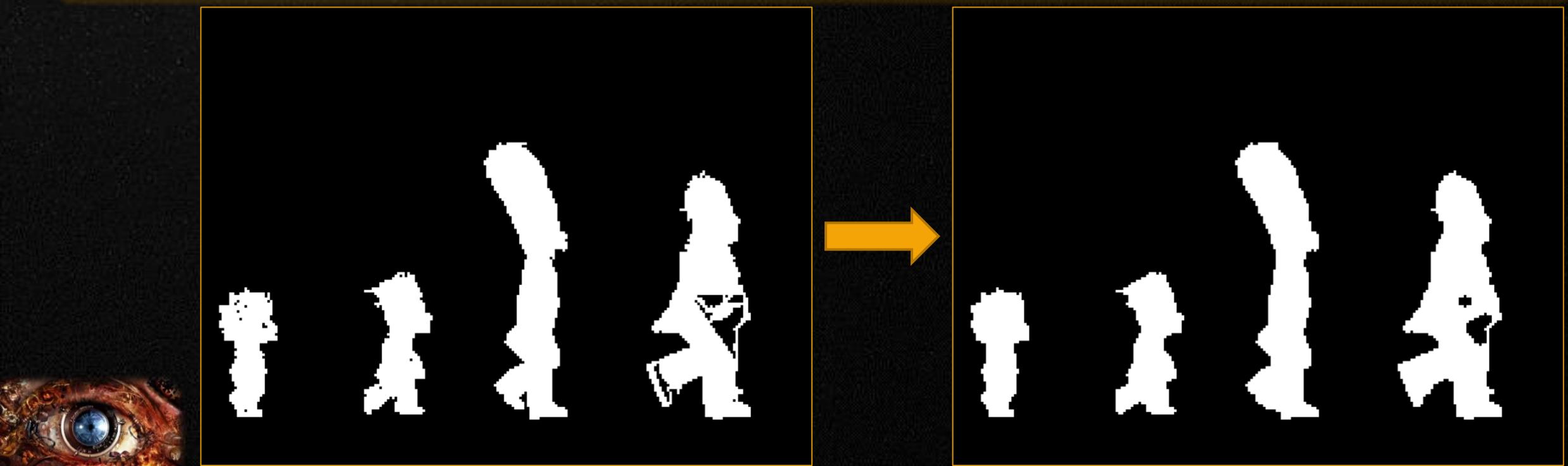
```
# Rimuove piccole componenti connesse: se è possibile fare un'ipotesi sulla dimensione  
# minima degli oggetti in movimento, si può supporre che tutte le componenti con  
# area minore di una certa soglia siano dovute al rumore e possano essere eliminate.  
  
n, cc, stats, _ = cv.connectedComponentsWithStats(mask, connectivity=4)  
small = [i for i in range(1,n) if stats[i, cv.CC_STAT_AREA]<70]  
# oppure: small = np.nonzero(stats[1:,cv.CC_STAT_AREA]<70)[0] + 1  
mask[np.isin(cc, small)] = 0
```



Miglioramento immagine binaria con le tecniche viste in precedenza (3)

13

```
# Chiude piccoli buchi e concavità utilizzando l'operazione di chiusura della morfologia  
# matematica.  
# La dimensione dell'elemento strutturante va scelta con attenzione.  
# Vedremo nel prossimo lucido come chiudere eventuali buchi di dimensione maggiore.  
  
se = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5))  
mask = cv.morphologyEx(mask, cv.MORPH_CLOSE, se)
```



Miglioramento immagine binaria con le tecniche viste in precedenza (4)

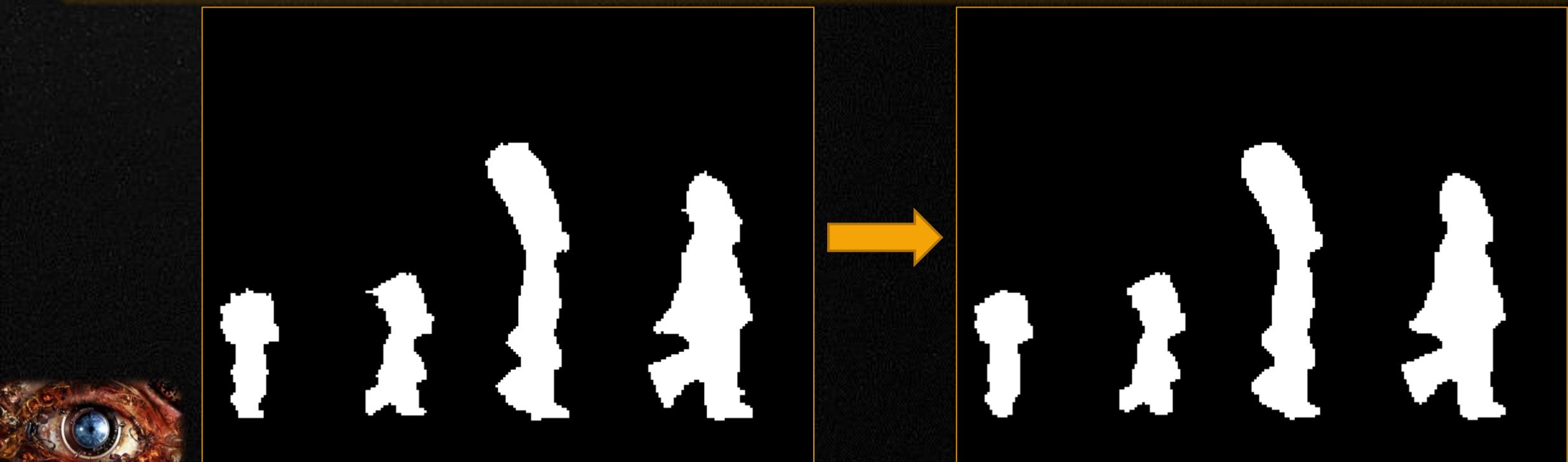
14

```
# Riempie eventuali altri buchi eseguendo l'estrazione delle componenti connesse  
# sul "negativo" della maschera (ossia sul background). Componenti connesse con area  
# inferiore a una soglia ragionevole sono normalmente "buchi" che possono essere "chiusi".  
  
n, cc, stats, _ = cv.connectedComponentsWithStats(255-mask, connectivity=4)  
holes = [i for i in range(1,n) if stats[i, cv.CC_STAT_AREA]<2000]  
# oppure: holes = np.nonzero(stats[1:,cv.CC_STAT_AREA]<10000)[0] + 1  
mask[np.isin(cc, holes)] = 255
```

Miglioramento immagine binaria con le tecniche viste in precedenza (5)

15

```
# Infine rimuove eventuali piccole protuberanze  
# mediante l'operazione di apertura della morfologia matematica.  
# Questo può aiutare a rimuovere eventuali artefatti dovuti al rumore che si sono "fusi"  
# con le componenti connesse principali.  
  
se = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5,5))  
mask = cv.morphologyEx(mask, cv.MORPH_OPEN, se)
```



Risultato finale

16



Tracking di oggetti

17

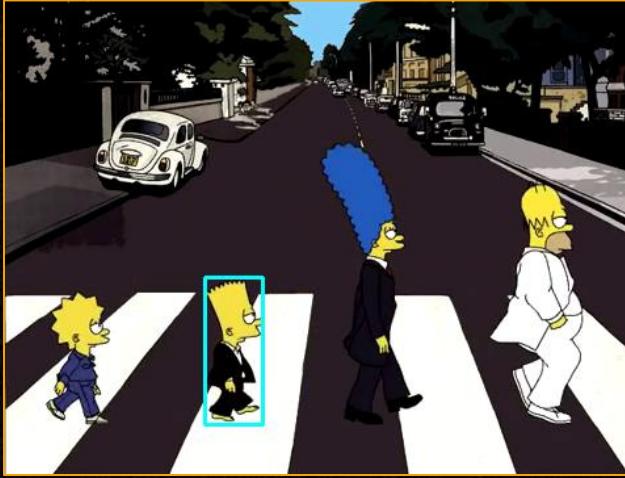
- Obiettivo: associare oggetti in movimento in frame consecutivi del video, ovvero "inseguire" uno o più oggetti "bersaglio" lungo la sequenza di frame in cui sono presenti.



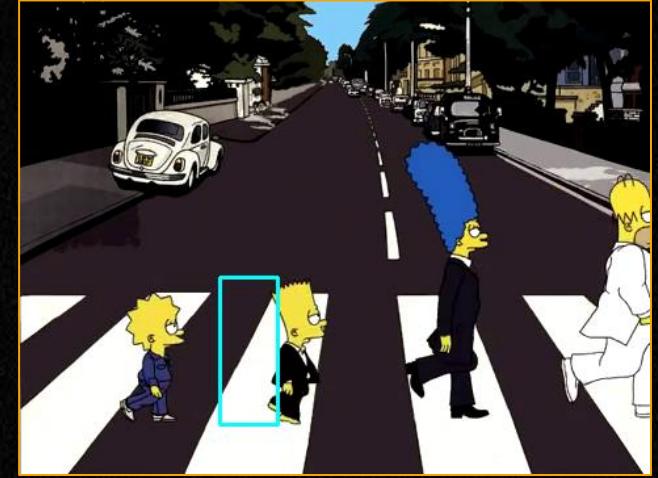
Tracking di un oggetto con Mean-shift

18

- ▶ Mean-shift è un algoritmo iterativo per determinare i massimi locali di una funzione di densità di probabilità a partire da un insieme di campioni.
- ▶ Può essere facilmente applicato al tracking di un oggetto:
 - Sia W la **regione** che nel **frame precedente** conteneva l'oggetto;
 - Sia C una **mappa di confidenza** che indica, per ogni pixel del nuovo frame, la probabilità che tale **pixel appartenga all'oggetto**;
 - Ad ogni iterazione si calcola la media pesata di C in W e si sposta W di conseguenza;
 - Ci si ferma quando la media converge o si raggiunge un massimo numero di iterazioni.



Frame precedente e regione W



Frame successivo e regione W



Mappa di confidenza C



Aggiornamento iterativo di W

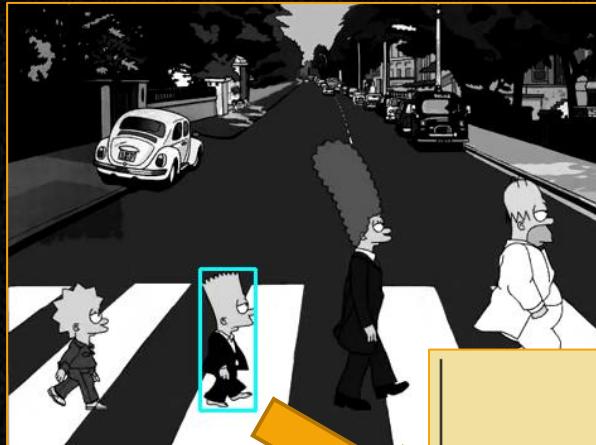
Mappa di confidenza per Mean-shift

- ▶ La mappa di confidenza C può essere ottenuta analizzando il colore o altre caratteristiche dell'oggetto.
- ▶ Per semplicità, negli esempi che seguono, come caratteristica sarà utilizzata la luminosità (valore dei pixel convertiti in grayscale $I_G[y, x]$):

1. Si calcola l'istogramma $h[v]$, $v = 0..255$ considerando solo i pixel che appartengono all'oggetto nel frame precedente;

2. Il valore di confidenza di ciascun pixel (x, y) è $C[y, x] = h[I_G[y, x]]$

- ▶ In altre parole la confidenza è data da quanto il valore di luminosità del pixel è frequente all'interno dell'oggetto.

 h  I_G  C

Mean-shift in OpenCV

20

```
def get_wnd_and_hist(i, frame, stats, mask):
    """Date le statistiche delle componenti connesse stats, calcola l'istogramma
    della luminosità del bounding box della componente di indice i sui soli pixel
    indicati da mask. Restituisce bounding box e istogramma normalizzato in [0,255]."""
    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]
    roi = cv.cvtColor(frame[y:y+h, x:x+w], cv.COLOR_BGR2GRAY)
    hist = cv.calcHist([roi],[0],mask[y:y+h, x:x+w],[256],[0,255])
    return (x, y, w, h), hist

def get_new_wnd(frame, mask, hist, wnd):
    """Esegue meanShift sulla luminosità di frame ponendo a zero i pixel di background
    secondo mask, utilizzando hist come istogramma e wnd come regione di partenza."""
    gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    # La funzione cv.calcBackProject calcola la mappa di confidenza a partire
    # dall'istogramma e dai valori dei pixel
    conf_map = cv.calcBackProject([gray],[0],hist,[0,255],1)
    conf_map[mask==0] = 0 # Confidenza a zero fuori dalla maschera
    _, new_wnd = cv.meanShift(conf_map, wnd, term_crit)
    return new_wnd
```



Esempi di tracking di oggetti (1)

```
# Data la sequenza fm di tuple (frame, mask)
term_crit = ( cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 1 )
tracked_objects = [] # (window, hist, label)
next_label = 1
res_tracking = [] # Sequenza di immagini risultato
for frame, mask in fm[1:]:
    n, cc, stats, _ = cv.connectedComponentsWithStats(mask)
    cc_used, next_tracked_objects = set(), []

    if n > 1: # Cerca di continuare a inseguire ciascuno degli oggetti
        for wnd,hist,label in tracked_objects:
            x,y,w,h = get_new_wnd(frame, mask, hist, wnd) # Applica meanShift
            # Quanti pixel per ogni etichetta ci sono dentro la nuova finestra?
            cc_hist, _ = np.histogram(cc[y:y+h, x:x+w], n, (0,n))
            index = np.argmax(cc_hist[1:])+1 # La più rappresentata (escluso back)
            if index not in cc_used and cc_hist[index]>=MIN_AREA_IN_WND:
                wnd, hist = get_wnd_and_hist(index, frame, stats, mask)
                next_tracked_objects.append( (wnd,hist,label) )
                cc_used.add(index)
```

(continua...)



Esempi di tracking di oggetti (2)

22

(...continua)

```
# Controlla se ci sono componenti connesse non abbinate a oggetti
for i in range(1,n):
    if (stats[i, cv.CC_STAT_AREA] >= MIN_AREA) and (i not in cc_used):
        wnd, hist = get_wnd_and_hist(i, frame, stats, mask)
        next_tracked_objects.append( (wnd, hist, next_label) )
        next_label += 1

tracked_objects = next_tracked_objects

# Disegna informazioni sugli oggetti nell'immagine risultato
res = frame.copy()
for (x,y,w,h),hist,label in tracked_objects:
    cv.rectangle(res, (x,y), (x+w,y+h), 255, 2)
    cv.rectangle(res, (x,y-22), (x+w,y), 255, -1)
    cv.putText(res, f'{label}', (x+5, y-5), cv.FONT_HERSHEY_PLAIN, 1, 0, 1)
    # ... altre informazioni, ad es. linea che collega tutti i baricentri di ogni
    # oggetto, contatore che indica da quanti frame è inseguito ciascun oggetto, ...
res_tracking.append(res)
```



Risultato finale (1)

23



Risultato finale (2)

24



► Rilevamento oggetti in movimento

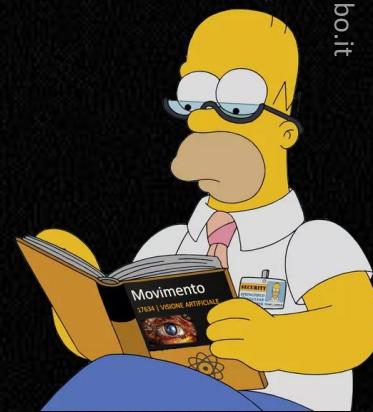
- Un'immagine binaria per ogni frame: il foreground indica gli oggetti in movimento
- Modellazione del background e confronto con il modello
- La semplice differenza fra due frame raramente è sufficiente
- Distribuzione dei valori dei pixel del background

► L'algoritmo MOG2 in OpenCV

- Esempio di utilizzo
- Tecniche di topologia digitale e morfologia matematica possono migliorare il risultato

► Tracking di oggetti

- Mean-shift in OpenCV: esempio con calcolo istogramma e utilizzo di calcBackProject



Per approfondire

26

- ▶ Nel libro [Szeliski, Computer Vision: Algorithms and Applications, 2022]:
 - Sezione 7.5.2 Mean shift
- ▶ Documentazione OpenCV del modulo «Video I/O»:
 - https://docs.opencv.org/master/dd/de7/group__videoio.html
- ▶ Documentazione OpenCV del modulo «Video Analysis - Motion Analysis»:
 - https://docs.opencv.org/master/de/de1/group__video__motion.html
- ▶ Documentazione OpenCV del modulo «Video Analysis - Object Tracking»:
 - https://docs.opencv.org/master/dc/d6b/group__video__track.html

