Gruppo 8: Barchetti Dennis, Bortolin Samuel, Grassi Alessandro

PROGETTO ELABORAZIONE E TRASMISSIONE DELLE IMMAGINI:

Computer Vision applicato al Basket
Detection canestri fatti

Introduzione:

Lo scopo del nostro progetto è quello di sviluppare un algoritmo in grado di contare i canestri effettuati durante una partita di basket avendo la registrazione di essa.
Successivamente implementare quest'ultimo attraverso un linguaggio di programmazione, nel nostro caso abbiamo scelto Python.
Per poter elaborare i dati ci siamo basati principalmente su due librerie:
-opencv: per la manipolazione dei file multimediali;
-numpy: per la manipolazione delle matrici;
Lo sviluppo del programma si basa essenzialmente su una partita suddivisa in 4 tempi.
Inizialmente ci siamo concentrati sul quarto tempo, per poi estendere l'algoritmo agli altri tre in modo da avere una certa rilevanza statistica.
Le aspettative erano quelle di avere un True Positive Rate (TPR) attorno all'80% ed un False Positive Rate (FPR) il più basso possibile.
Questi due indici rappresentano rispettivamente l'ammontare di canestri rilevati su quelli che "avrei dovuto rilevare" -> TPR = (# canestri presi)/(# canestri effettivi) e quelli che ho rilevato anche se realmente inesistenti su quelli che avrei dovuto rilevare -> FPR = (# falsi canestri)/(# canestri effettivi).
Come prima cosa abbiamo ricavato una regione di interesse (ROI) al fine di isolare i canestri.
Il nostro approccio in merito all'algoritmo è stato quello di cercare di "isolare" la palla dal resto, in modo da renderne più semplice la rilevazione.
In un primo momento è stata applicata una sogliatura del colore, mentre in un secondo momento una sottrazione dello sfondo attraverso tecnica KNN.
Una volta ottenuta una visione più nitida della palla abbiamo posto tre soglie sui canestri entro le quali essa doveva muoversi al fine di segnare un punto;
(sono state poi ridotte a due poiché il tabellone pubblicitario dava particolari problemi, così abbiamo mantenuto solo quella superiore e quella inferiore).
Essendo diverse le coordinate per canestro sinistro e destro sono stati pensati due metodi appositi per ognuno.
Per migliorare la visibilità della palla abbiamo impiegato alcuni operatori morfologici come open e dilate.

Opencv:

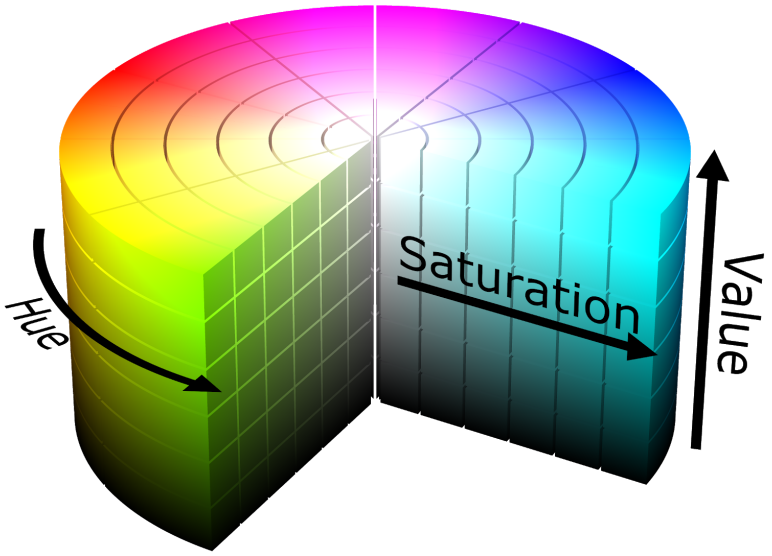
Ci siamo interamente basati sui metodi della libreria opencv, di seguito è mostrato un esempio di apertura di un video attraverso essa.

-Apertura di un video:

Il programma mostrato in seguito “scorre” e mostra frame per frame un video ad intervalli di 1ms.
Nello specifico il video sarà passato come argomento nel metodo **cv.VideoCapture(“video”)** nel quale sarà possibile passarlo anche un percorso oltre che come stringa.
il metodo **capture.isOpened** ci permette di controllare se il video sia stato effettivamente aperto (restituisce un valore booleano che controlliamo attraverso una condizione if)
Appena entrati nel ciclo while mostriamo ogni frame attraverso il metodo **cv.imshow("frame",frame)** ogni 1ms poichè **cv.waitKey(1)** ovvero il tempo di “attesa” dopo aver mostrato ogni frame è di 1ms.
Se il frame fosse = None oppure se viene premuto il tasto ESC (codice ASCII 27) si esce dal ciclo.

```
capture = cv.VideoCapture(“quarto_tempo.asf”)

if not capture.isOpened:
    print('Unable to open')
    exit(0)

while True:
captureStatus,frame= capture.read()
 if frame is None:
        break

 cv.imshow("frame",frame)

 key = cv.waitKey(1)
    if key == 27:
        break
```

-SOGLIATURA DEL COLORE HSV:



Per permettere un corretto isolamento della palla dal resto dell'immagine abbiamo adottato una sogliatura del colore attraverso il modello HSV.
L'hsv è un sistema codificato con il quale è possibile definire dei colori attraverso l'uso di 3 parametri: colore (hue), saturazione e valore da cui ne deriva l'acronimo inglese HSV.

In seguito ad una serie di tentativi permessi dal seguente programma è stato possibile ricavare il valore di arancione corrispondente alla palla, al fine di isolarla :

#Creazione di una finestra denominata "tracking" (su cui muovere le trackbar)
cv2.namedWindow("Tracking",cv2.WINDOW_NORMAL)

#Creazione trackbar (una per ogni livello low e high di ogni parametro):
cv2.createTrackbar("LH","Tracking",0,255,nothing)
cv2.createTrackbar("LS","Tracking",0,255,nothing)
cv2.createTrackbar("LV","Tracking",0,255,nothing)
cv2.createTrackbar("UH","Tracking",255,255,nothing)
cv2.createTrackbar("US","Tracking",255,255,nothing)
cv2.createTrackbar("UV","Tracking",255,255,nothing)

```
while True:
    captureStatus, frame = capture.read()

    if frame is None:
    break

    #Conversione del frame da RGB a HSV:
    hsv=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)

    #Associo le variabili alla posizione della trackbar:
    l_h = cv2.getTrackbarPos("LH","Tracking")
    l_s = cv2.getTrackbarPos("LS", "Tracking")
    l_v = cv2.getTrackbarPos("LV", "Tracking")

    u_h = cv2.getTrackbarPos("UH", "Tracking")
    u_s = cv2.getTrackbarPos("US", "Tracking")
    u_v = cv2.getTrackbarPos("UV", "Tracking")

    #Definisco 2 range (lower red, upper red):
    l_r=np.array([l_h,l_s,l_v])
    u_r=np.array([u_h,u_s,u_v])

    #Infine creo una maschera e la applico al frame bit a bit ottenendo res:
    mask = cv2.inRange(hsv,l_r, u_r)
    res= cv2.bitwise_and(frame,frame,mask=mask)
```

Mostrando res attraverso imshow possiamo muovere in tempo reale la trackbar per trovare i valori più adatti:

VALORI HSV RICAVATI:

```
lower_red = np.array([160, 75, 85])
upper_red = np.array([180, 255, 255])
```

OPERATORI MORFOLOGICI:

Filtri Morfologici:PARTE 3 da pg 76

Gli operatori morfologici sono un particolare tipo di filtri non-lineari che operano sulla base della forma della maschera.

I filtri morfologici si basano su:
-Un elemento strutturale, ovvero una forma 2D discreta dotata di un punto di riferimento (non necessariamente il centro della forma, ma potrebbe essere anche esterno).
-Due operatori base definiti: **erosione** e **dilatazione** che determinano l'azione che l'elemento strutturale compie sull'immagine.

Dato un oggetto A (immagine) e un elemento strutturale B, definiamo :

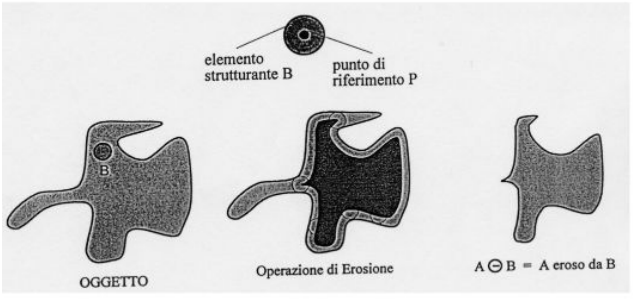
EROSIONE: $A \ominus B = \{x \mid B_x \subset A\}$

DILATAZIONE: $A \oplus B = \{x \mid B_x \cap A \neq 0\}$

dove:
- {x} rappresenta il luogo dei punti x
- B_x rappresenta l'elemento strutturante B con il punto di riferimento centrato in x sul piano immagine

Interpretazione geometrica di erosione e dilatazione:

Erosione->



Dilatazione->

Nel nostro programma l'operatore di dilatazione è stato usato per rendere più facile la rilevazione della palla dilatando l'oggetto.
Il tutto in seguito all'applicazione di open.
#L'operatore è stato applicato alla maschera hsv ricavata precedentemente.
mask = cv.dilate(mask, None, iterations=2)

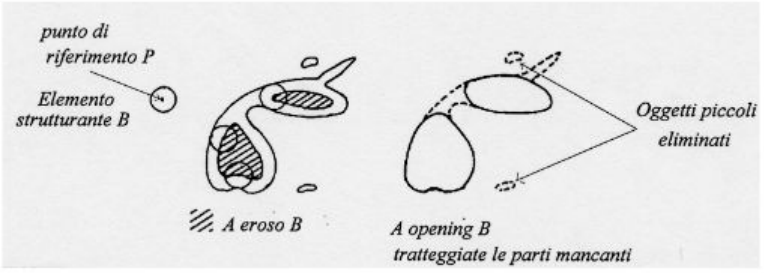
Combinando le due operazioni base si possono ottenere altri 2 operatori morfologici:

APERTURA: $A \circ B = (A \ominus B) \oplus B$

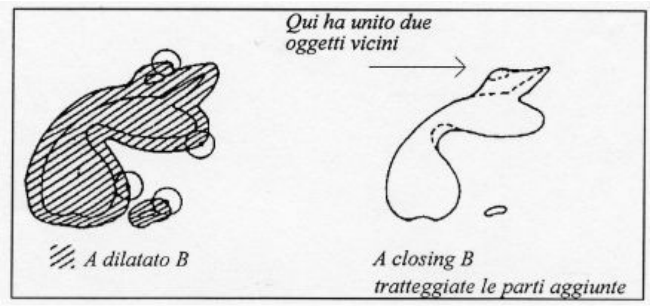
CHIUSURA: $A \bullet B = (A \oplus B) \ominus B$

-l'Apertura ha l'effetto di eliminare i dettagli "piccoli" e di separare oggetti uniti in modo debole;
-la chiusura ha l'effetto di raggruppare oggetti vicini o uniti in modo debole.

Apertura->



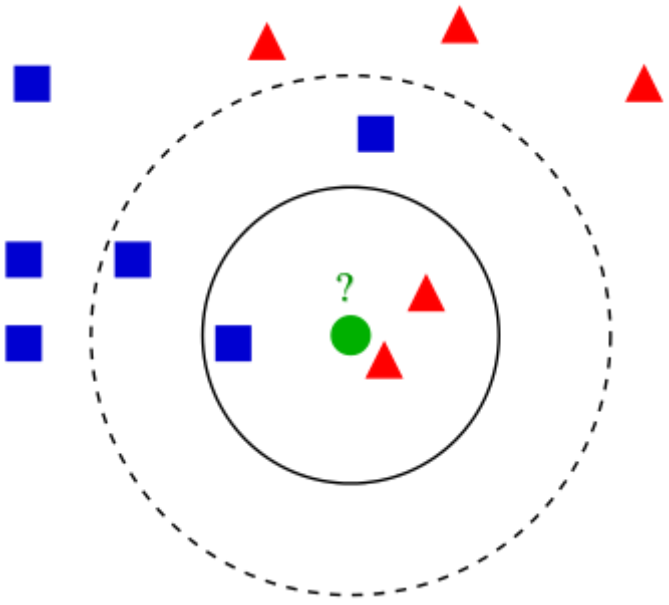
Chiusura->



L'operatore di apertura (open) è stato usato per rimuovere il rumore di fondo al fine di isolare la palla.
mask = cv.morphologyEx(mask, cv.MORPH_OPEN,(5, 5), iterations=1)

KNN

KNN (K Nearest Neighbors) è un metodo usato per la classificazione e per la regressione. Nel caso del background subtraction si parla di classificazione, ovvero decidere la classe di appartenenza del pixel preso in esame. Nello specifico le classi sono background e foreground. Per capire il funzionamento dell'algoritmo si può prendere in considerazione una rappresentazione grafica, nell'immagine sottostante è rappresentato un grafico dove sono riportati i valori negli assi cartesiani di due classi rappresentate da quadrati blu e triangoli rossi, potrebbero rappresentare background e foreground. Il cerchio verde rappresenta l'input dell'algoritmo, ovvero il campione che si vuole classificare (pixel). Per eseguire la classificazione si calcola la distanza la distanza euclidea tra le feature (campioni) attorno al campione e si prendono in considerazione solo le k più vicine dove k è un valore intero arbitrario solitamente dispari o primo. La classe di appartenenza del valore di input che verrà assegnata sarà quella della classe presente in maggioranza tra i K valori vicini, da qui si può capire l'importanza di avere numeri dispari nel caso di sole due classi, nel caso si scegliessero 4 vicini nell'esempio sottostante ci si troverebbe in una situazione di indecisione.



Contours

Un contorno è una curva che si unisce includendo tutti i punti continui che hanno lo stesso colore o intensità. Il loro utilizzo è consigliato su immagini che hanno uno

sfondo nero e l'oggetto da contornare in bianco. Il metodo da noi utilizzato prende in input tre parametri: il frame sorgente, hierarchy (gerarchia) e il metodo di approssimazione del contorno. Il frame sorgente può essere in qualsiasi formato, ma è raccomandabile usarne uno in bianco e nero in quanto si massimizza così l'accuratezza del risultato. Per gerarchia si intende il numero di "sottocontorno" associato al contorno preso in esame. Un contorno sarà di gerarchia 0 se non è contenuto da nessun altro oggetto. Se invece è gerarchia 1 allora l'oggetto preso in esame è contenuto da un oggetto. Per capire meglio si può guardare la rappresentazione sottostante. Il terzo parametro, approssimazione di contorno, specifica il formato in cui verrà restituita l'informazione riguardante il contorno. Se non si usa nessuna approssimazione verrà ritornato un array con tutte le coordinate dei pixel facenti parte dei contorni.

Questo metodo è stato preso in considerazione con l'intento di identificare il contorno della palla e tracciarne il suo movimento (il movimento del punto centrale) per capire se entra o no nel canestro.

A riga 1 viene chiamato il metodo cv.findContours() al quale viene dato in input una copia del frame in bianco e nero, nel nostro caso la maschera del frame originale al quale vengono tenuti solo i valori da noi ritenuti appartenenti alla palla. Come secondo input si specifica cv.RETR_EXTERNAL, si indica qui di prendere solo in considerazione contorni con gerarchia 0. Infine si imposta cv.CHAIN_APPROX_SIMPLE che specifica che il contorno è rappresentato solo dai punti estremi del rettangolo, in questo modo si risparmia memoria senza avere dati ridondanti. Successivamente per ogni contorno si ricava il cerchio che meglio li contiene e si prendono solo quelli con raggio tra 15 e 25 pixel esclusi, questo per isolare oggetti troppo grandi o troppo piccoli che non possono essere la palla. Ognuno di questi cerchi viene disegnato in un frame che sarà poi il valore di ritorno del metodo.

Questo metodo non viene utilizzato nella soluzione finale in quanto il rumore è troppo elevato e vengono rappresentate figure inesistenti, specialmente in prossimità degli schermi vicino ai canestri. Date le buone potenzialità di questo algoritmo si è deciso di tenerlo in quanto con una riduzione migliore del rumore potrebbe rivelarsi molto efficace.

Durante lo sviluppo della soluzione abbiamo deciso di mantenere il codice leggibile, applicando alcune regole basilari del clean code, e di raggruppare tutti i metodi utilizzati in una classe. Questa scelta è dovuta al fatto di voler rendere i metodi applicabili con facilità su altri problemi, o meglio su altre partite. Abbiamo scelto di non inserire un metodo che svolge tutte le operazioni che portano alla soluzione, ma abbiamo inserito solo i metodi che svolgono le singole operazioni. Questo è stato fatto per due motivi: la possibilità sperimentare con i metodi proposti e la possibilità di aggiungerne altri in rilasci futuri nel caso si scegliesse di espandere le sue funzionalità.

DOCUMENTAZIONE METODI

`__init__(self)`
Costruttore della classe, ritorna un oggetto StandardVideoOperations con le proprietà upper_left_LEFT, upper_right_LEFT, upper_left_RIGHT e upper_right_RIGHT impostate a (0, 0).

`video_cutter(frame, upper_left, bottom_right)`
params:

- frame: matrice almeno bidimensionale rappresentante un frame hsv
- upper_left: lista contenente due interi rappresentanti l'angolo in alto a sinistra della ROI
- bottom_right: ista contenente due interi rappresentanti l'angolo in basso a destra della ROI

set_left(upper_left, bottom_right)
params:

- upper_left: lista contenente le coordinate dell'angolo in alto a sinistra della ROI
- bottom_right: lista contenente le coordinate dell'angolo in basso a destra della ROI

Imposta le variabili interne riguardanti la roi del canestro sinistro

set_right(upper_left, bottom_right)
params:

- upper_left: lista contenente le coordinate dell'angolo in alto a sinistra della ROI
- bottom_right: lista contenente le coordinate dell'angolo in basso a destra della ROI

Imposta le variabili interne riguardanti la roi del canestro destro

`cut_left(startingFrame)`
params:

- startingFrame: matrice almeno bidimensionale rappresentante un frame

return value: frame contenente solo la ROI ottenuta mediante il metodo video_cutter con i valori impostati tramite set_left

`cut_right(startingFrame)`
params:

- startingFrame: matrice almeno bidimensionale rappresentante un frame

return value: frame contenente solo la ROI ottenuta mediante il metodo video_cutter con i valori impostati tramite set_right

`draw_rectangle(frameBGR, upperLeft, bottomRight, color_string)`
params:

- frameBGR: matrice rappresentante un frame in formato BGR
- upperLeft: lista rappresentante coordinata dell'angolo in alto a sinistra del rettangolo da disegnare
- bottomRight: lista rappresentante coordinate dell'angolo in basso a destra del rettangolo da disegnare
- color_string: stringa che accetta due valori: "green" e "red"

return value: frame in formato BGR con disegnato un rettangolo con bordo di spessore un pixel nelle coordinate specificate da upperLeft e bottomRight del colore specificato da color_string.

L'utilità di avere due colori sta nella possibilità di disegnare un rettangolo rosso quando non viene rilevata nessuna palla e un rettangolo verde quando viene rilevata, facilita notevolmente il debugging.

`get_hsvmask_on_ball(frame_hsv)`
params:

- frame_hsv: matrice rappresentante un frame in formato HSV

return value: frame hsv con "attivi" solo i pixel che hanno un colore compreso tra i parametri Hue Saturation e Value che corrispondono a quelli di una palla da basket, gli altri pixel saranno neri.

`get_knn_on_left_frame(frame)`
params:

- frame: matrice bidimensionale che rappresenta un frame in scala di grigi

return value: frame in scala di grigi al quale viene applicato il metodo di opencv *cv.createBackgroundSubtractorKNN().apply(frame)*

`get_knn_on_right_frame(frame)`
params:

- frame: matrice bidimensionale che rappresenta un frame in scala di grigi

return value: frame in scala di grigi al quale viene applicato il metodo di opencv *cv.createBackgroundSubtractorKNN().apply(frame)*

```
@staticmethod
    def find_circles(frame_to_scan, frame_to_design)
```
params:

- frame_to_scan:
- frame_to_design:

`countWhitePixels(rows, colRange, greyScaleFrame)`
params:

- rows:array che rappresenta le coordinate y rispetto al greyScaleFrame.
- colRange: array che rappresenta le coordinate x rispetto al greyScaleFrame.
- greyScaleFrame: frame che verrà analizzato nelle intersezioni dei due parametri precedentemente descritti

return value: booleano rappresentante l'avvenuta rilevazione del canestro, true se è rilevato false altrimenti.

`spotBallOnTop_right(greyScaleFrame)`
params:

- greyScaleFrame: matrice bidimensionale rappresente un frame in scala di grigi

return value: boolean che indica la presenza o meno della palla nel rettangolo in alto nella roi di destra

`spotBallOnBottom_right(greyScaleFrame):`
params:

- greyScaleFrame: matrice bidimensionale rappresente un frame in scala di grigi

return value: boolean che indica la presenza o meno della palla nel rettangolo in basso nella roi di destra

`spotBallOnTop_left(greyScaleFrame):`
params:

- greyScaleFrame: matrice bidimensionale rappresente un frame in scala di grigi

return value: boolean che indica la presenza o meno della palla nel rettangolo in alto nella roi di sinistra

`spotBallOnBottom_left(greyScaleFrame):`
params:

- greyScaleFrame: matrice bidimensionale rappresente un frame in scala di grigi

return value: boolean che indica la presenza o meno della palla nel rettangolo in basso nella roi di sinistra

DESCRIZIONE FUNZIONAMENTO ALGORITMO countWhitePixels

L'algoritmo che abbiamo implementato identifica la presenza della palla se vede almeno 16 pixel bianchi di fila con al massimo un solo pixel nero in mezzo. Il numero di 16 pixel è stato deciso sperimentalmente, di fatti si è visto che è il numero di pixel minimo che può occupare la palla e che esclude buona parte del rumore di sfondo.
L'algoritmo è implementato nel metodo countWhitePixels documentato nella sezione precedente e descritto qui di seguito.

La complessità dell'algoritmo nel caso pessimo è θ(n*m) dove n è il numero di righe da controllare e m è il numero di colonne da controllare.

Riga 1, 2 e 3 si inizializza un ciclo for che scorre ogni elemento contenuto in rows e lo salva in row e vengono inizializzate due variabili a 0: conecutiveWhitePixels e consecutiveBlackPixels che rappresentano il numero di pixel consecutivi del rispettivo colore rilevati finora.
Riga 4 vengono cilcati i valori in colRange.
A riga 5 è presente l'if che controlla se il pixel correntemente in analisi è una scala di grigi. Se questo avviene viene incrementata la variabile dei pixel neri consecutivi, nel momento che vengono rilevati 2 pixel neri consecutivi viene resettata la variabile dei pixel bianchi consecutivi, questo viene fatto dopo due pixel neri per evitare che il rumore influisca il risultato.
Nel else viene resettata la variabile che conta i pixel neri consecutivi e incrementata quella dei pixel bianchi. Dal momento che si arriva a 15 pixel bianchi viene considerato canestro effettuato.
L'ultima riga ritorna false come valore di default.

```
1 def countWhitePixels(rows, colRange, greyScaleFrame):
2       for row in rows:
3               consecutiveWhitePixels = 0
4               consecutiveBlackPixels = 0
5               for col in colRange:
6                       if greyScaleFrame[row, col] < 255:
7                               consecutiveBlackPixels += 1
8                               if consecutiveBlackPixels == 2:
9                                       consecutiveWhitePixels = 0
10                      else:
11                              consecutiveBlackPixels = 0
12                              consecutiveWhitePixels += 1
13                              if 15 < consecutiveWhitePixels:
14                                      return True
15      return False
```

Codice del main (diversificazione per SX e DX nei Thread):

In questa sezioni sono riportati i passi più importanti dei codici riportati come allegati, spiegando le funzionalità e come funziona il main del nostro codice, che va ad invocare principalmente i metodi della classe Standard Video Operation presentata nel capitolo precedente.

```
svo = StandardVideoOperations()          # istanza della nostra classe per eseguire le varie operazioni
capture = cv.VideoCapture("/path/tempo.asf")     # rileva dal percorso fornito il video da analizzare
```

#SX:	#DX:
svo.set_left((540, 940), (740, 1140))	svo.set_right((3225, 900), (3425, 1100))

```
# vedi sotto perchè le coordinate cambiano il base al quarto di gioco preso in analisi

if not capture.isOpened: # verifica la corretta apertura del video
        print('Unable to open')
        exit(0)


frame_counter = 0
top_frameSX = middle_frameSX = last_score_frameSX = 0
top_frameDX = middle_frameDX = last_score_frameDX = 0
# indicatori dei frame che saranno utili nell'analisi
```

#SX:	#DX:
upper_left1 = (80, 85) bottom_right1 = (140, 95)	upper_left1 = (90, 50) bottom_right1 = (150, 60)
upper_left2 = (80, 125) bottom_right2 = (140, 135)	upper_left2 = (90, 100) bottom_right2 = (150, 110)
upper_left3 = (70, 160) bottom_right3 = (160, 170)	upper_left3 = (75, 160) bottom_right3 = (175, 170)

```
# posizioni dei vari rettangoli che vengono utilizzati nell'analisi

while True:
        frame_counter += 1
        captureStatus, startingFrame = capture.read()          # lettura di un frame del video
        if startingFrame is None:                  # interruzione del ciclo alla conclusione del video
                break
```

#SX:	#DX:

leftCut = svo.cut_left(startingFrame)
# blurred = cv.GaussianBlur(leftCut, (5, 5), 0) 	
# blurred = cv.medianBlur(blurred, 5) 	
hsvFrame = cv.cvtColor(leftCut, cv.COLOR_BGR2HSV)
	rightCut = svo.cut_right(startingFrame)
	# blurred = cv.GaussianBlur(rightCut, (5, 5), 0)
	# blurred = cv.medianBlur(blurred, 5)
	hsvFrame = cv.cvtColor(rightCut, cv.COLOR_BGR2HSV)

ritaglio della RoI e conversione in HSV, filtri mediani non utilizzati perché portavano ad errori

```
res = svo.get_hsvmask_on_ball(hsvFrame)        # esecuzione della sogliatura del colore
finalFrame = svo.get_knn_on_frame(res)         # utilizzo della background subtraction KNN
returnFrame = cv.cvtColor(finalFrame, cv.COLOR_GRAY2BGR)
# conversione in BGR per disegnare i rettangoli colorati in base al rilevamento della palla


if frame_counter > 10:    # evitiamo problemi dovuti al background subtraction nei primi frame
```

#SX:	#DX:
if svo.spotBallOnTop_left(finalFrame):	if svo.spotBallOnTop_right(finalFrame):

```
    # ricerchiamo la presenza della palla nel rettangolo sopra il canestro e se è presente
    salviamo il numero del frame, lo segnaliamo in output e coloriamo il rettangolo di
    verde, altrimenti lasciamo il rettangolo colorato di rosso (vedi in main 5 che cambiano
    delle cose)

        top_frame = frame_counter
        print("top FRAME:", top_frame)
        returnFrame = svo.draw_rectangle(returnFrame, upper_left1, bottom_right1, "green")
else:
        returnFrame = svo.draw_rectangle(returnFrame, upper_left1, bottom_right1, "red")

    """
    # il rettangolo centrale a livello della retina è stato provato ma con risultati peggiori in
    quanto non veniva sempre segnalata la presenza della palla e l'attivazione era dovuta
    principalmente al movimento retina o a causa del  monitor dietro il canestro (che
    spesso influisce e causa problemi anche sul rilevamento della palla nel rettangolo
    sotto il canestro)
```

#SX:	#DX:
if svo.spotBallOnMedium_left(finalFrame) and 2 < (frame_counter - top_frame) < 8:	if svo.spotBallOnMedium_right(finalFrame) and 2 < (frame_counter - top_frame) < 8:

```
        middle_frame = frame_counter
        print("middle FRAME:", middle_frame)
        returnFrame = svo.draw_rectangle(returnFrame, upper_left2, bottom_right2, "green")
else:
        returnFrame = svo.draw_rectangle(returnFrame, upper_left2, bottom_right2, "red")
    """
```

#SX:	#DX:
if svo.spotBallOnBottom_left(finalFrame) and 3 < (frame_counter - top_frame) < 25 and (frame_counter - last_score_frameSX) > 50 and (frame_counter - last_score_frameDX) > 50:	if svo.spotBallOnBottom_right(finalFrame) and 3 < (frame_counter - top_frame) < 25 and (frame_counter - last_score_frameDX) > 50 and (frame_counter - last_score_frameSX) > 50:

ricerchiamo la presenza della palla nel rettangolo sotto il canestro e se è presente dopo essere stata rilevata nel rettangolo sopra tra 3 e 25 frame prima e l'ultimo score è stato segnato almeno 50 frame fa salviamo il numero del frame, segnaliamo in output il canestro rilevato e coloriamo il rettangolo di verde, altrimenti lasciamo il rettangolo colorato di rosso

```
        last_score_frame = frame_counter
        print("score:", last_score_frame)
        bottom_frame = frame_counter
        returnFrame = svo.draw_rectangle(returnFrame, upper_left3, bottom_right3, "green")
else:
        returnFrame = svo.draw_rectangle(returnFrame, upper_left3, bottom_right3, "red")
```

if top_frame - last_score_frame < 0 and frame_counter - last_score_frame == 5:
# questa parte è stata utilizzata per valutare che non venisse segnalata nuovamente la presenza della palla nella parte in alto e effettuiamo un controllo 5 frame dopo che è stato segnalato il canestro e avvisiamo in output che il canestro è stato segnato con una ulteriore precauzione

```
        print("score con precauzione top")
```

#SX:	#DX:
cv.imshow("returnFrame", returnFrame) cv.imshow("originalFrame", leftCut)	cv.imshow("returnFrame", returnFrame) cv.imshow("originalFrame", rightCut)

presentazione a monitor della RoI dopo l'analisi e anche del video originale per visualizzare come l'algoritmo sta svolgendo il proprio lavoro

key = cv.waitKey(1)
# attesa minima tra un frame e il successivo, ma può essere prolungata per visualizzare più lentamente l'esecuzione dell'algoritmo e capire la cause di eventuali problematiche
if key == 27: # per ogni evenienza se viene premuto esc si interrompe l'esecuzione
    break

cv.destroyAllWindows() # alla fine dell'esecuzione rimuove tutte le finestre dei frame dallo schermo

Descrizione del codice del main:

Parte sopra del codice per aprire il file contenente il video e preparare i frame per l'analisi della presenza della palla nelle varie zone.
Parte sotto del codice per sfruttare le 2 zone e dedurre se c'è un canestro → Analisi su 2 livelli (o anche 3 per migliorare i risultati, ma in realtà quella al centro si è rivelata poco utile), per vedere se la palla scende nei frame successivi (da fissare una soglia max, impostata a 25 frame, ossia 1 sec)
→ Ragionamento sul posizionamento delle zone di rilevamento, euristica, . . .

Testing sui vari quarti + problemino con la RoI:

Per effettuare i primi test abbiamo provveduto a tagliare delle parti del filmato del quarto quarto (quarto_tempo.asf) di una partita di EuroCup della squadra di basket trentina Aquila Basket Trento, estraendo varie sequenze nelle quali comparivano canestri, schiacciate, errori. Usando l'euristica abbiamo perfezionato i parametri del nostro algoritmo e ottenuto buoni risultati. (verità raccolta/acquisita manualmente)
Il ritaglio delle sequenze è stato eseguito con ffmpeg utilizzando il seguente comando: ffmpeg -i input.asf -ss 00:01:00 -to 00:01:05 -q 0 output.asf
Dove -i input.asf rappresenta il file in ingresso, output.asf il file di uscita, -ss 00:01:00 l'istante temporale da cui inizierà il nuovo filmato in uscita, -to 00:01:05 l'istante temporale a cui finirà il nuovo filmato di uscita (oppure se mettiamo -t 00:00:05 il filmato in uscita avrà durata pari a quella indicata) e infine il parametro più importante -q 0 che ci permette di tagliare il video senza perdere qualità (lossless).
Poi abbiamo deciso di testarlo su l'intero quarto periodo e fare statistica su quanti canestri presi, sia reali che farlocchi e estratto TPR (True Positive Ratio) e FPR (False Positive Ratio credo, vedi qui). Poi per completare la nostra sperimentazione ed avere un'idea della performance complessiva del nostro algoritmo, lo abbiamo mandato in esecuzione su tutta la partita riscontrando nei primi due quarti dei problemi dovuti al fatto che la videocamera abbia una posizione diversa rispetto agli ultimi due quarti. Per questo abbiamo deciso di adattare la RoI (Region of Interest) per definire l'area del canestro da analizzare differentemente per questi quarti:

SX: 1° quarto	SX: 2° quarto	SX: 3° quarto e 4° quarto
upper_left = (455, 955) bottom_right = (655, 1155)	upper_left = (485, 950) bottom_right = (685, 1150)	upper_left = (540, 940) bottom_right = (740, 1140)

DX: 1° quarto	DX: 2° quarto	DX: 3° quarto e 4° quarto
upper_left = (3145, 895) bottom_right = (3345, 1095)	upper_left = (3185, 900) bottom_right = (3385, 1100)	upper_left = (3225, 900) bottom_right = (3425, 1100)

Risultati e conclusioni:

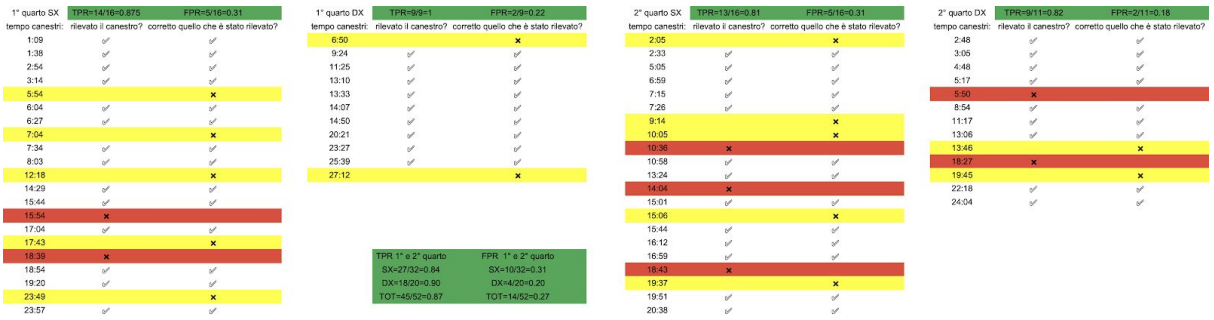
In questa sezione verranno presentati e discussi risultati ottenuti dal nostro algoritmo per riconoscere l'atto del canestro.

Analizzando i risultati ottenuti si può notare che ci sono delle differenze sia tra canestro destro e sinistro ma anche tra primo e secondo tempo. Prenderemo in analisi 2 parametri che descrivono a fondo la capacità di rilevare un canestro e la capacità di non incorrere in errori dovuti ad azioni che non si sono concluse con dei canestri.

$$\text{TPR (true positive rate)} = \frac{\text{numero di canestri rilevati corretti}}{\text{numero di canestri da rilevare}}$$

$$\text{FPR (false positive rate)} = \frac{\text{numero di canestri rivelati fasulli}}{\text{numero di canestri da rilevare}}$$

Riportiamo i risultati applicandolo al 1° e 2° quarto:



1° quarto SX	TPR=14/16=0.875	FPR=5/16=0.31
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
1:09	✓	✓
1:38	✓	✓
2:54	✓	✓
3:14	✓	✓
5:54		×
6:04	✓	✓
6:27	✓	✓
7:04		×
7:34	✓	✓
8:03	✓	✓
12:18		×
14:29	✓	✓
15:44	✓	✓
15:54	×	
17:04	✓	✓
17:43		×
18:39	×	
18:54	✓	✓
19:20	✓	✓
23:49		×
23:57	✓	✓

1° quarto DX	TPR=9/9=1	FPR=2/9=0.22
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
6:50		×
9:24	✓	✓
11:25	✓	✓
13:10	✓	✓
13:33	✓	✓
14:07	✓	✓
14:50	✓	✓
20:21	✓	✓
23:27	✓	✓
25:39	✓	✓
27:12		×

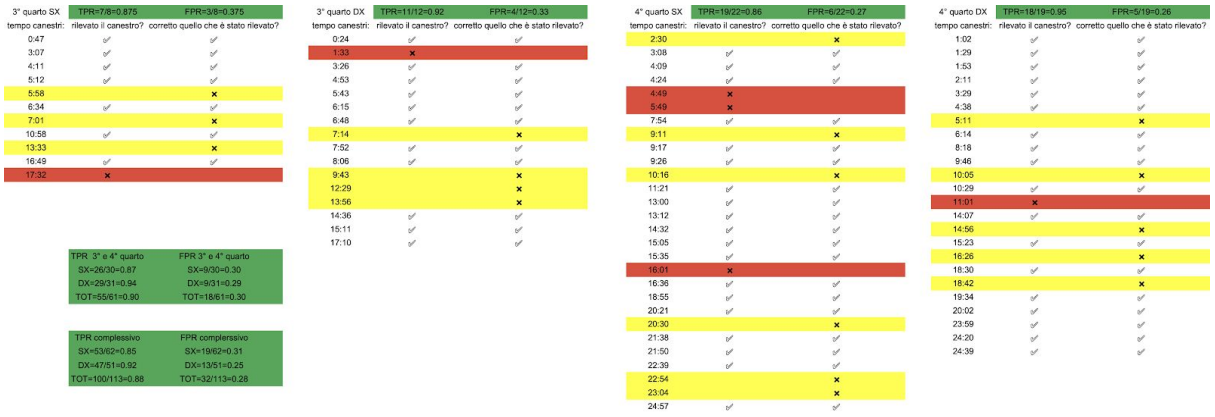
TPR 1° e 2° quarto	FPR 1° e 2° quarto
SX=27/32=0.84	SX=10/32=0.31
DX=18/20=0.90	DX=4/20=0.20
TOT=45/52=0.87	TOT=14/52=0.27

2° quarto SX	TPR=13/16=0.81	FPR=5/16=0.31
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
2:05		×
2:33	✓	✓
5:05	✓	✓
6:59	✓	✓
7:15	✓	✓
7:26	✓	✓
9:14		×
10:05		×
10:38	×	
10:58	✓	✓
13:24	✓	✓
14:04	×	
15:01	✓	✓
15:06		×
15:44	✓	✓
16:12	✓	✓
16:59	✓	✓
18:43	×	
19:37		×
19:51	✓	✓
20:38	✓	✓

2° quarto DX	TPR=9/11=0.82	FPR=2/11=0.18
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
2:48	✓	✓
3:05	✓	✓
4:48	✓	✓
5:17	✓	✓
5:50	×	
8:54	✓	✓
11:17	✓	✓
13:06	✓	✓
13:46		×
16:27	×	
19:45		×
22:18	✓	✓
24:04	✓	✓

1° e 2° quarto . . .
La videocamera è mossa rispetto al 4° quarto, costretti a cambio di posizione e risultati leggermente peggiori rispetto agli altri due quarti TRP = 0.87 e FPR = 0.27

Riportiamo i risultati applicandolo al 3° e 4° quarto:



3° quarto SX	TPR=7/8=0.875	FPR=3/8=0.375
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
0:47	✓	✓
3:07	✓	✓
4:11	✓	✓
5:12	✓	✓
5:58		×
6:34	✓	✓
7:01		×
10:58	✓	✓
13:33		×
16:49	✓	✓
17:32	×	

3° quarto DX	TPR=11/12=0.92	FPR=4/12=0.33
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
0:24	✓	✓
1:33	×	
3:26	✓	✓
4:53	✓	✓
5:43	✓	✓
6:15	✓	✓
6:48	✓	✓
7:14		×
7:52	✓	✓
8:06	✓	✓
9:43		×
12:29		×
13:56		×
14:36	✓	✓
15:11	✓	✓
17:10	✓	✓

TPR 3° e 4° quarto	FPR 3° e 4° quarto
SX=26/30=0.87	SX=9/30=0.30
DX=29/31=0.94	DX=9/31=0.29
TOT=55/61=0.90	TOT=18/61=0.30

TPR complessivo	FPR complerssivo
SX=53/62=0.85	SX=19/62=0.31
DX=47/51=0.92	DX=13/51=0.25
TOT=100/113=0.88	TOT=32/113=0.28

4° quarto SX	TPR=19/22=0.86	FPR=6/22=0.27
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
2:30		×
3:08	✓	✓
4:09	✓	✓
4:24	✓	✓
4:49	×	
5:49	×	
7:54	✓	✓
9:11		×
9:17	✓	✓
9:26	✓	✓
10:16		×
11:21	✓	✓
13:00	✓	✓
13:12	✓	✓
14:32	✓	✓
15:05	✓	✓
15:35	✓	✓
16:01	×	
16:36	✓	✓
18:55	✓	✓
20:21	✓	✓
20:30		×
21:38	✓	✓
21:50	✓	✓
22:39	✓	✓
22:54		×
23:04		×
24:57	✓	✓

4° quarto DX	TPR=18/19=0.95	FPR=5/19=0.26
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
1:02	✓	✓
1:29	✓	✓
1:53	✓	✓
2:11	✓	✓
3:29	✓	✓
4:38	✓	✓
5:11		×
6:14	✓	✓
8:18	✓	✓
9:46	✓	✓
10:05		×
10:29	✓	✓
11:01	×	
14:07	✓	✓
14:56		×
15:23	✓	✓
16:26		×
18:30	✓	✓
18:42		×
19:34	✓	✓
20:02	✓	✓
23:59	✓	✓
24:20	✓	✓
24:39	✓	✓

3° e 4° quarto . . .
Analisi per ciò che è stata pensata TPR = 0.90 e FPR = 0.30

Accuratezza detection sinistra vs detection di destra:

A DX TPR più alto (0.92 contro il 0.85 di SX) e FPR più basso (0.25 contro il 0.31 di SX), problema del monitor che aiuta a far segnare canestro anche quando non c'è
In conclusione l'algoritmo realizzato si presta bene
In complessivo analisi performance . . . TPR = 0.88 e FPR = 0.28

Da fare dei ragionamenti sulla problematicità del monitor e della posizione fotocamera ad esempio ed altro . . .

Lavoro futuro:

L'obiettivo sarebbe quello di migliorare, non il fatto di cercare di rilevare più canestri dato che ne prende circa il 90% di quelli realmente presenti/effettivi, ma piuttosto trovando un modo di ridurre il numero dei canestri fasulli, dato che circa il 24% dei canestri rilevati risulta essere fasullo (calcolato come 32 fasulli/(100 corretti + 32 fasulli) e quindi quelli rivelati che risultano essere veri sono circa il 76%). Uno spunto per un lavoro futuro è quello di usare un detector come ad esempio YOLO o MaskRCNN per fare in modo che l'algoritmo riconosca proprio la palla (consentendoci di togliere rumore e oggetti dello stesso colore che non riusciamo ad ignorare dalla nostra analisi e potrebbero essere causa di problemi, come ad esempio i monitor) ed eseguire il tracciamento della palla e riuscire a migliorare i risultati.
Detector però rallenta codice e complica le cose . . . → nostro algoritmo complessità molto bassa, la velocità di esecuzione è di circa 20/25 frame per secondo del video di riferimento e potrebbe essere utilizzato per un'analisi in real-time.