

Gruppo 8: Barchetti Dennis, Bortolin Samuel, Grassi Alessandro



UNIVERSITÀ
DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Progetto di Elaborazione e Trasmissione delle Immagini:

Computer Vision applicata al Basket:

Detection canestri fatti

Anno accademico 2019/2020

INDICE:

1. Introduzione	2
1.1 Prerequisiti	3
1.2 Sogliatura del colore HSV	4
1.3 Operatori morfologici	6
1.4 KNN	8
1.5 Rilevazione dei contorni	9
2. Sviluppo	12
2.1 Documentazione Standard Video Operation	12
2.2 Descrizione funzionamento dell'algoritmo countWhitePixels	15
2.3 Descrizione del codice del main	17
3. Risultati e conclusioni	19
3.1 Applicazione dell'algoritmo	20
3.2 Confronto dell'accuratezza tra sinistra e destra	23
3.3 Lavoro futuro	24

1. 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 quattro tempi. Inizialmente ci siamo concentrati sul quarto quarto di gioco, 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" e quelli che ho rilevato anche se realmente inesistenti su quelli che "avrei dovuto rilevare".

Come prima cosa abbiamo ricavato una regione di interesse (ROI, che sta per Region of Interest) 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, per migliorare la visibilità della palla abbiamo impiegato alcuni operatori morfologici come open e dilate, 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 thread con metodi appositi per ognuno due canestri.

1.1 Prerequisiti:

Per permettere il funzionamento del codice sono necessarie alcune librerie, nonché avere python3 installato sul proprio pc.

Numpy:

Libreria utile alla visualizzazione ed elaborazione di matrici.

Imutils:

Libreria necessaria per il metodo circles.

Opencv:

Libreria utile all'elaborazione video in tempo reale.

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

Caratteristiche tecniche del video di test:

- Numero telecamere: una telecamera fissa con copertura del campo
- Video Input: risoluzione: 4k (3840x2160) formato: video.asf
- Frame per secondo: 25

È poi necessario inserire nel codice il percorso del video che si intende aprire ed impostare i valori della ROI adeguata al quarto di interesse.

Il codice è ottenibile clonando il repository del progetto da GitHub con il seguente comando da terminale: git clone <https://github.com/samuelbortolin/ETI-G8-Basket.git> o eventualmente scaricandolo dal browser se non si dispone di git.

Apertura di un video:

Il programma mostrato in seguito "scorre" e mostra frame per frame un video ad intervalli di 1ms (il minimo possibile).

Nello specifico il nome del video sarà passato come argomento nel metodo **cv.VideoCapture("video")** nel quale sarà possibile passare anche un percorso del video 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 poiché **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 e il programma termina.

```
capture = cv.VideoCapture(" /path/video.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
```

1.2 Sogliatura del colore HSV:

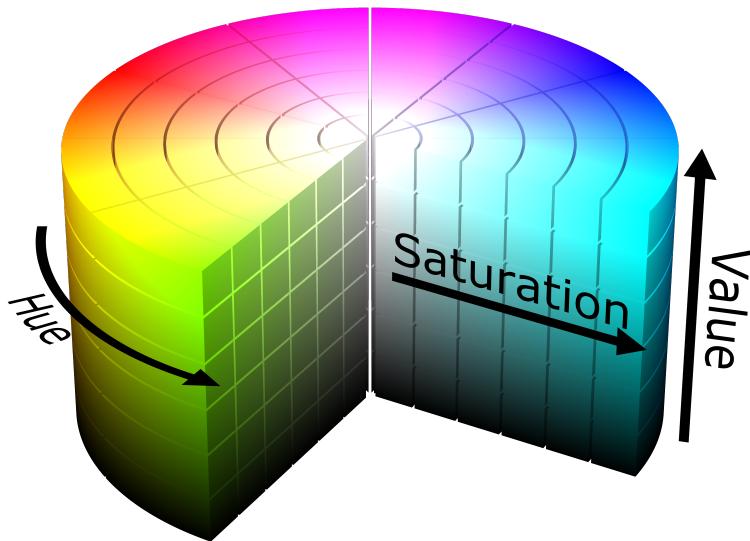


Figura 1.1: Grafico a torta rappresentante la variazione dei parametri Hue, Saturation e Value all'interno del dominio HSV. Fonte: https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png

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])
```

1.3 OPERATORI MORFOLOGICI:

Fonte: [PDF del corso di Elaborazione e trasmissione delle immagini \[140127\] tenuto dal professor DE NATALE](#)

Filtri Morfologici:

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 e dilatazione come la formulazione matematica presente in figura 1.2.

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

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

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

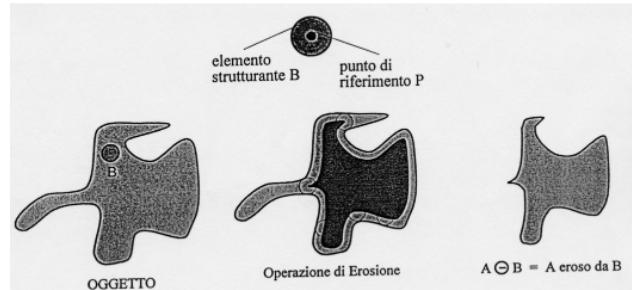
Figura 1.2: Rappresentante la formulazione matematica di erosione e dilatazione

Interpretazione geometrica di erosione e dilatazione:

Erosione:

Figura 1.3:

Rappresentazione grafica di una erosione



Dilatazione:

Figura 1.4:

Rappresentazione grafica di una 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 un'operazione di apertura per ridurre il rumore.

L'operatore è stato applicato alla maschera hsv ricavata precedentemente.

`mask = cv.dilate(mask, None, iterations=2) # due iterazioni`

Combinando le due operazioni base si possono ottenere altri 2 operatori morfologici, come mostrato in figura 1.5.

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

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

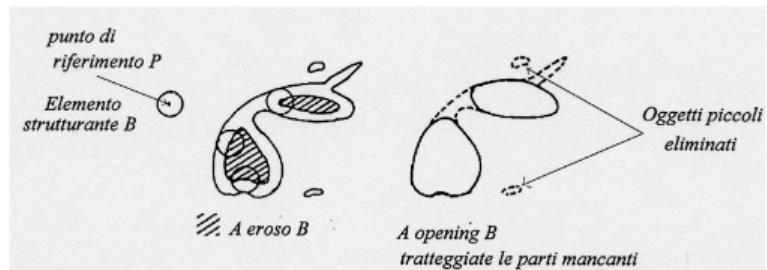
Figura 1.5: Rappresentante la formulazione matematica di apertura e chiusura

- **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:

Figura 1.6:

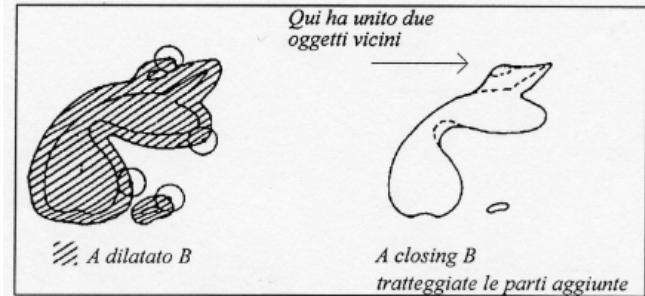
Rappresentazione grafica apertura



Chiusura:

Figura 1.7:

Rappresentazione grafica chiusura



L'operatore di apertura (open) è stato usato per rimuovere il rumore di fondo al fine di riuscire ad isolare meglio la palla.

Mask = cv.morphologyEx(mask, cv.MORPH_OPEN, (5, 5), iterations=1)

1.4 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 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 quattro vicini nell'esempio sottostante ci si troverebbe in una situazione di indecisione.

<https://www.cronj.com/blog/background-subtraction/>

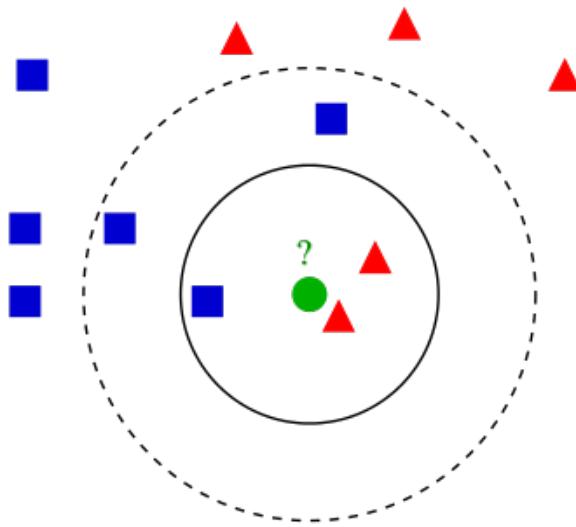


Figura 1.8: rappresentazione di un grafico dove si vede la differenza di avere un key=3 e un key=5, cambia totalmente il risultato.

Fonte: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg

1.5 Rilevazione dei contorni

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 in figura 1.9.

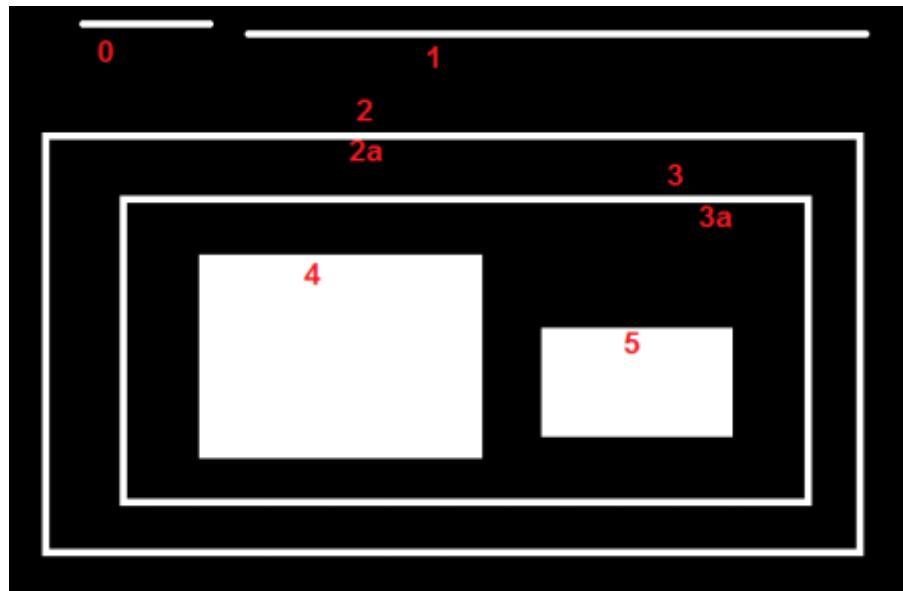


Figura 1.9: esempio di come sono riconosciute le gerarchie, in questo esempio 0 e 1 sono a gerarchia 0 mentre 4 e 5 sono a gerarchia 2.

Fonte: https://docs.opencv.org/3.4/d9/d8b/tutorial_py_contours_hierarchy.html

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.

Per chiamare il metodo cv.findContours() occorre dare 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 che contiene l'oggetto, 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 10 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.

Fonte: https://docs.opencv.org/trunk/d4/d73/tutorial_py_contours_begin.html

I contorni degli oggetti nel video però nel nostro caso sono risultati troppo rumorosi, rendendo così impossibile la rilevazione della palla, la quale veniva confusa con altri oggetti di forma sferica come, ad esempio, le mele mostrate sui tabelloni pubblicitari.

2. Sviluppo

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.

2.1 Documentazione Standard Video Operation

`__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).

`set_left(upper_left, bottom_right)`

parameters:

- `upper_left`: tupla contenente le coordinate dell'angolo in alto a sinistra della ROI
- `bottom_right`: tupla 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)`

parameters:

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

Imposta le variabili interne riguardanti la ROI del canestro destro.

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

parameters:

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

return value: ROI del frame passato per parametro con la dimensione come indicata da upper_left e bottom_right.

```
cut_left(startingFrame)
```

parameters:

- 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)
```

parameters:

- 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)
```

parameters:

- frameBGR: matrice rappresentante un frame in formato BGR
- upperLeft: tupla rappresentante coordinata dell'angolo in alto a sinistra del rettangolo da disegnare
- bottomRight: tupla 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)
```

parameters:

- 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)
```

parameters:

- 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)
```

parameters:

- 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)`.

```
find_circles(frame_to_scan, frame_to_design)
```

parameters:

- frame_to_scan: frame che sarà analizzato

- frame_to_design: frame nel quale verranno disegnati i cerchi

return value: frame_to_design con dei cerchi disegnati intorno ai contorni riconosciuti.

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

parameters:

- 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: Boolean rappresentante l'avvenuta rilevazione del canestro, True se è rilevato False altrimenti.

```
spotBallOnTop_left(greyScaleFrame) :
```

parameters:

- greyScaleFrame: matrice bidimensionale rappresentante 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.

spotBallOnMiddle_left(greyScaleFrame) :

parameters:

- greyScaleFrame: matrice bidimensionale rappresentante un frame in scala di grigi
return value: Boolean che indica la presenza o meno della palla nel rettangolo al centro nella ROI di sinistra.

spotBallOnBottom_left(greyScaleFrame) :

parameters:

- greyScaleFrame: matrice bidimensionale rappresentante 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.

spotBallOnTop_right(greyScaleFrame)

parameters:

- greyScaleFrame: matrice bidimensionale rappresentante 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.

spotBallOnMiddle_right(greyScaleFrame) :

parameters:

- greyScaleFrame: matrice bidimensionale rappresentante un frame in scala di grigi
return value: Boolean che indica la presenza o meno della palla nel rettangolo al centro nella ROI di destra.

spotBallOnBottom_right(greyScaleFrame) :

parameters:

- greyScaleFrame: matrice bidimensionale rappresentante 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.

2.2 Descrizione funzionamento dell'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 $\theta(n*m)$ dove n è il numero di righe da controllare e m è il numero di colonne da controllare.

Con le righe 2, 3 e 4 si inizializza un ciclo for che scorre ogni elemento contenuto in rows e lo salva in row e vengono inizializzate due variabili a 0: consecutiveWhitePixels e consecutiveBlackPixels che rappresentano il numero di pixel consecutivi del rispettivo colore rilevati finora.

A riga 5 vengono cilcati i valori in colRange.

A riga 6 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.

Nell'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 segnalato che la palla si trova all'interno della zona e viene ritornato True.

L'ultima riga ritorna False come valore di default nel caso non venisse rilevata la palla.

```
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
```

2.3 Descrizione del codice del main:

Il main è stato implementato per leggere il video fornito e suddividere il processing in due thread, uno per la parte riguardante il canestro di sinistra e l'altra per il canestro di destra. In figura 2.1 è riportato un flowchart che spiega la logica di svolgimento dei passi più importanti del codice, spiegando le funzionalità e come funziona su ognuna delle due parti, il quale va ad invocare principalmente i metodi della classe Standard Video Operation presentata nella sezione precedente.

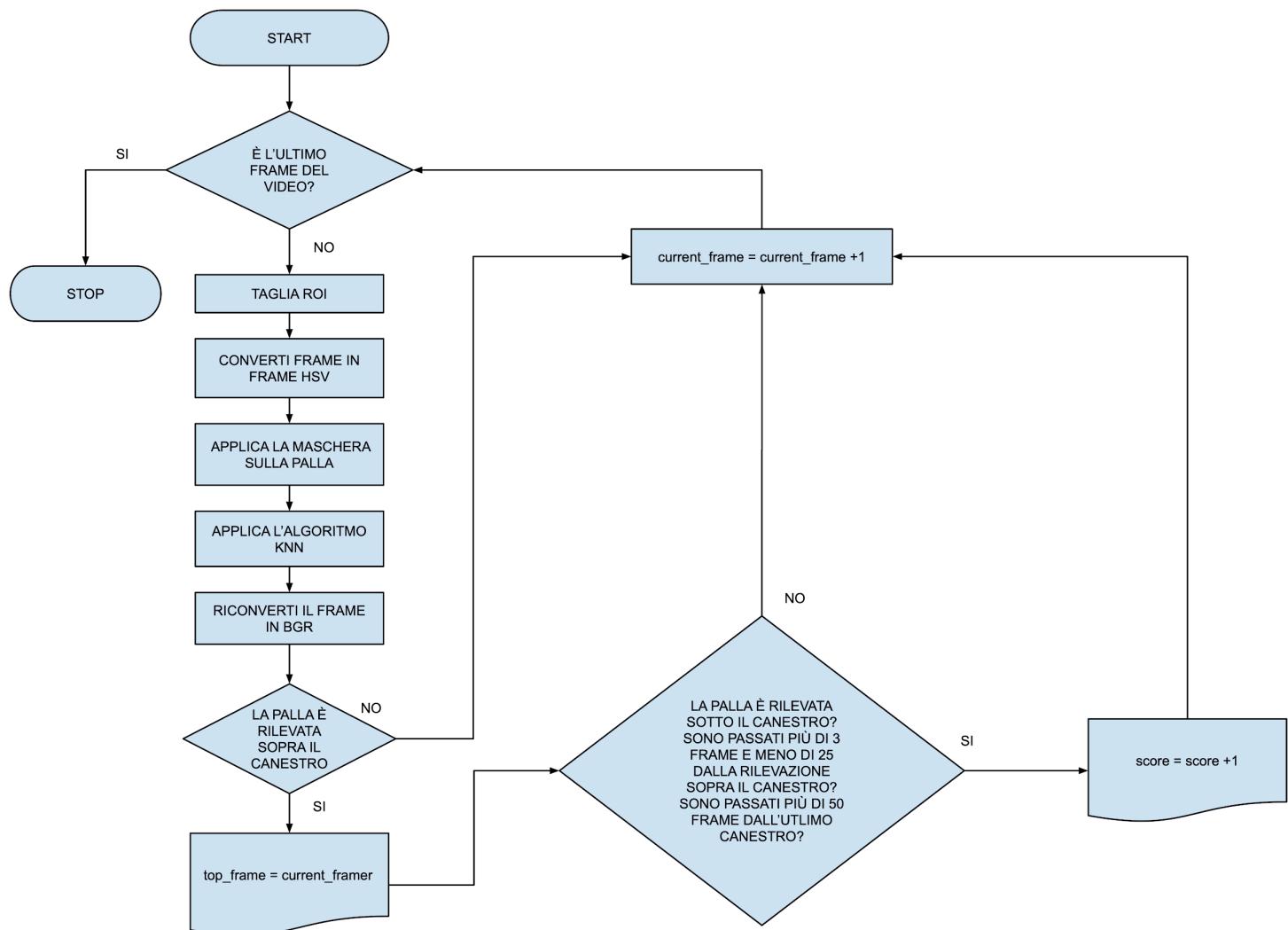


Figura 2.1: Flowchart rappresentante la logica di funzionamento del processing svolto su ognuno dei due thread

La parte iniziale del codice occorre per aprire il file contenente il video e preparare i frame per l'analisi della presenza della palla nelle varie zone. Per entrambi i canestri viene isolata una ROI, effettuata una sogliatura sulla base di un range di colori HSV ed applicato un algoritmo di Background Subtraction con tecnica KNN.

La parte sotto del codice serve per sfruttare le due zone e dedurre se c'è un canestro effettuando un'analisi su due livelli, inizialmente è stata pensata per essere su tre livelli sperando in risultati migliori, ma in realtà quella al centro si è rivelata poco utile a causa del rumore sulla retina e del monitor dietro il canestro. Questa analisi è stata effettuata per vedere se la palla compie l'azione di scendere nei frame successivi al rilevamento sopra il canestro, per cui è stata fissata una soglia minima di 3 frame e una massima di 25 frame.

Il posizionamento delle zone di rilevamento è stato effettuando usando l'euristica, capendo come la palla dovrebbe scendere dopo un canestro e valutando se la palla passa in quella zona.

Se la palla viene rilevata nella zona sopra il canestro e nel range di tempo definito viene rilevata nella zona sotto il canestro e l'ultimo canestro è stato almeno 50 frame fa, allora questa azione viene segnata in output come score e il contatore aumenta. Altrimenti si prosegue semplicemente nell'analisi del video fino alla fine dello stesso.

3. Risultati e conclusioni:

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

Per effettuare i primi test abbiamo provveduto a tagliare delle parti del filmato del quarto quarto di gioco (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. La verità su cui ci siamo basati per valutare l'accuratezza del sistema proposto è stata acquisita manualmente dai vari video.

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 fasulli e estratto TPR (True Positive Ratio) e FPR (False Positive Ratio). 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 per definire l'area del canestro da analizzare differentemente per questi quarti, come rappresentato in Figura 3.1.

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)

Figura 3.1: Tabella rappresentante i diversi valori delle ROI per i vari quarti di gioco

3.1 Applicazione dell'algoritmo:

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 in figura 3.2 i risultati dell'algoritmo applicandolo al primo quarto di gioco. Il TPR del canestro di sinistra è pari a 0.875 con 14 canestri rilevati sui 16 presenti ed un FPR di 0.31 con 5 canestri fasulli sui 16 presenti. Il TPR del canestro di destra è pari a 1 con 9 canestri rilevati sui 9 presenti ed un FPR di 0.33 con 3 canestri fasulli sui 9 presenti.

1° quarto SX		TPR=14/16=0.875	FPR=5/16=0.31
tempo canestri:	rilevato il canestro?	correto 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=3/9=0.33
tempo canestri:	rilevato il canestro?	correto 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	✗		
27:57	✗		

Figura 3.2: Tabella rappresentante i risultati del primo quarto di gioco

Riportiamo in figura 3.3 i risultati dell'algoritmo applicandolo al secondo quarto di gioco. Il TPR del canestro di sinistra è pari a 0.81 con 13 canestri rilevati sui 16 presenti ed un FPR di 0.25 con 4 canestri fasulli sui 16 presenti. Il TPR del canestro di destra è pari a 0.82 con 9 canestri rilevati sugli 11 presenti ed un FPR di 0.09 con 1 canestri fasulli sugli 11 presenti.

2° quarto SX	TPR=13/16=0.81	FPR=4/16=0.25	2° quarto DX	TPR=9/11=0.82	FPR=1/11=0.09
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?	tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
2:05		x	2:48	✓	✓
2:33	✓	✓	3:05	✓	✓
5:05	✓	✓	4:48	✓	✓
6:59	✓	✓	5:17	✓	✓
7:15	✓	✓	5:50	x	
7:26	✓	✓	8:54	✓	✓
9:14		x	11:17	✓	✓
10:05		x	13:06	✓	✓
10:36	x		13:46		x
10:58	✓	✓	18:27	x	
13:24	✓	✓	22:18	✓	✓
14:04	x		24:04	✓	✓
15:01	✓	✓			
15:44	✓	✓			
16:12	✓	✓			
16:59	✓	✓			
18:43	x				
19:37		x			
19:51	✓	✓			
20:38	✓	✓			

TPR 1° e 2° quarto	FPR 1° e 2° quarto
SX=27/32=0.84	SX=9/32=0.28
DX=18/20=0.90	DX=4/20=0.20
TOT=45/52=0.87	TOT=13/52=0.25

Figura 3.3: Tabella rappresentante i risultati del secondo quarto di gioco

Nel primo e secondo quarto la videocamera è mossa rispetto al quarto quarto di gioco e quindi come già detto siamo stati costretti ad un cambio di posizione della ROI e i risultati sono leggermente peggiori rispetto agli altri due quarti ma comunque molto buoni con un TPR complessivo pari a 0.87 con 45 canestri rilevati sui 52 presenti ed un FPR complessivo di 0.25 con 13 canestri fasulli sui 52 presenti.

Più nello specifico per la parte sinistra il TPR è leggermente più basso della parte di destra con un 0.84 dato da 27 canestri rilevati sui 32 presenti contro un 0.90 della parte di destra dato da 18 canestri rilevati sui 20 presenti. Il FPR della parte di sinistra si è anche confermato più alto con un 0.28 dato da 9 canestri fasulli sui 32 presenti contro un 0.20 della parte di destra dato da 4 canestri fasulli sui 20 presenti.

Riportiamo in figura 3.4 i risultati dell'algoritmo applicandolo al terzo quarto di gioco. Il TPR del canestro di sinistra è pari a 1 con 8 canestri rilevati sugli 8 presenti ed un FPR di 0.375 con 3 canestri fasulli sugli 8 presenti. Il TPR del canestro di destra è pari a 0.92 con 11 canestri rilevati sui 12 presenti ed un FPR di 0.33 con 4 canestri fasulli sui 12 presenti.

3° quarto SX	TPR=8/8=1	FPR=3/8=0.375	3° quarto DX	TPR=11/12=0.92	FPR=4/12=0.33
tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?	tempo canestri:	rilevato il canestro?	corretto quello che è stato rilevato?
0:47	✓	✓	0:24	✓	✓
3:07	✓	✓	1:33	✗	
4:11	✓	✓	3:26	✓	✓
5:12	✓	✓	4:53	✓	✓
5:58		✗	5:43	✓	✓
6:34	✓	✓	6:15	✓	✓
7:01		✗	6:48	✓	✓
10:58	✓	✓	7:14		✗
13:33		✗	7:52	✓	✓
16:49	✓	✓	8:06	✓	✓
17:32	✓	✓	9:43		✗
			12:29		✗
			13:56		✗
			14:36	✓	✓
			15:11	✓	✓
			17:10	✓	✓

Figura 3.4: Tabella rappresentante i risultati del terzo quarto di gioco

Riportiamo in figura 3.5 i risultati dell'algoritmo applicandolo al quarto quarto di gioco. Il TPR del canestro di sinistra è pari a 0.91 con 20 canestri rilevati sui 22 presenti ed un FPR di 0.14 con 3 canestri fasulli sui 22 presenti. Il TPR del canestro di destra è pari a 0.95 con 18 canestri rilevati sui 19 presenti ed un FPR di 0.26 con 5 canestri fasulli sui 19 presenti.

Il terzo e il quarto quarto di gioco sono i quarti per cui che è stata pensata/perfezionata l'analisi, infatti si nota che i risultati sono leggermente migliori rispetto agli altri due quarti con un TPR complessivo pari a 0.93 con 57 canestri rilevati sui 61 presenti ed un FPR complessivo di 0.25 con 15 canestri fasulli sui 61 presenti.

Più nello specifico per la parte sinistra il TPR è leggermente più basso della parte di destra con un 0.93 dato da 28 canestri rilevati sui 30 presenti contro un 0.94 della parte di destra dato da 29 canestri rilevati sui 31 presenti. Il FPR della parte di sinistra si è confermato più basso con un 0.20 dato da 6 canestri fasulli sui 30 presenti contro un 0.29 della parte di destra dato da 9 canestri fasulli sui 31 presenti.

Gruppo 8: Barchetti Dennis, Bortolin Samuel, Grassi Alessandro

4° quarto SX	TPR=20/22=0.91	FPR=3/22=0.14	4° quarto DX	TPR=18/19=0.95	FPR=5/19=0.26
tempo canestri: rilevato il canestro? corretto quello che è stato rilevato?					
2:30		x	1:02	✓	✓
3:08	✓	✓	1:29	✓	✓
4:09	✓	✓	1:53	✓	✓
4:24	✓	✓	2:11	✓	✓
4:49	x		3:29	✓	✓
5:49	✓	✓	4:38	✓	✓
7:54	✓	✓	5:11	x	
9:17	✓	✓	6:14	✓	✓
9:26	✓	✓	8:18	✓	✓
10:16	x		9:46	✓	✓
11:21	✓	✓	10:05	x	
13:00	✓	✓	10:29	✓	✓
13:12	✓	✓	11:01	x	
14:32	✓	✓	14:07	✓	✓
15:05	✓	✓	14:56	x	
15:35	✓	✓	15:23	✓	✓
16:01	x		16:26	x	
16:36	✓	✓	18:30	✓	✓
18:55	✓	✓	18:42	x	
20:21	✓	✓	19:34	✓	✓
20:30	x		20:02	✓	✓
21:38	✓	✓	23:59	✓	✓
21:50	✓	✓	24:20	✓	✓
22:39	✓	✓	24:39	✓	✓
24:57	✓	✓			
TPR 3° e 4° quarto			TPR complessivo		
SX=28/30=0.93		SX=6/30=0.20	SX=55/62=0.89		SX=15/62=0.24
DX=29/31=0.94		DX=9/31=0.29	DX=47/51=0.92		DX=13/51=0.25
TOT=57/61=0.93		TOT=15/61=0.25	TOT=102/113=0.90		TOT=28/113=0.25

Figura 3.5: Tabella rappresentante i risultati del quarto quarto di gioco

3.2 Confronto dell'accuratezza tra sinistra e destra:

A destra generalmente si conferma un TPR più alto pari a 0.92 dato da 47 canestri rilevati sui 51 presenti contro il 0.89 di sinistra dato da 55 canestri rilevati sui 62 presenti e un FPR più alto pari a 0.25 dato da 13 canestri fasulli sui 51 presenti contro il 0.24 di sinistra dato da 15 canestri fasulli sui 62 presenti. Questo in generale ma nei primi due quarti il FPR a sinistra è pari a 0.28 contro il 0.20 di destra, lo spostamento della videocamera influenza probabilmente ciò, dato che negli ultimi 2 quarti a sinistra è pari a 0.20 contro il 0.29 di destra. I false positive sono dovuti al problema del monitor che aiuta a far segnare canestro anche quando non c'è realmente e questo problema è più evidente a destra negli ultimi 2 quarti e più evidente a sinistra nei primi 2 quarti dato che la videocamera come già detto è stata spostata e la ROI è stata modificata.

In conclusione, l'algoritmo realizzato si presta abbastanza bene per lo scopo per cui è stato pensato, infatti, in complessivo le performance riportate dopo l'analisi complessiva su tutta la partita riportano un **TPR pari a 0.90 dato da 102 canestri rilevati su un totale di 113 presenti e FPR = 0.25 dato da 28 canestri fasulli su un totale di 113 presenti.**

In questa cartella su Google Drive sono riportati i 28 canestri fasulli che sono stati rilevati (False Positive) e i gli 11 canestri esistenti che non sono stati rilevati dall'algoritmo (False Negative).

https://drive.google.com/drive/folders/1ZTNDAyA7iDHI_lyrfJNFrYTRRHFF4dLm?usp=sharing

N.B.: Sono video tagliati in scarsa qualità solo per fare vedere l'azione che causa l'errore e non per farci delle analisi sopra, tagliati con **ffmpeg -i input.asf -ss 00:01:00 -to 00:01:05 output.asf** senza preservare la qualità con -q 0 (lossy).

3.3 Lavoro futuro:

Con un opportuno lavoro futuro su questo progetto, l'obiettivo sarebbe quello di migliorare, non il fatto di cercare di rilevare più canestri dato che ne prende all'incirca il 90% di quelli realmente presenti/effettivi, ma piuttosto di trovare un modo di ridurre il numero dei canestri fasulli, dato che circa il 22% dei canestri rilevati risulta essere fasullo, calcolato come $28 \text{ fasulli} / (102 \text{ corretti} + 28 \text{ fasulli})$, e quindi quelli rivelati che risultano essere realmente esistenti sono circa il 78%, valore buono ma comunque con margini di miglioramento.

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 oggetti che compaiono nel monitor) ed eseguire il tracciamento della palla e riuscire a migliorare i risultati.

Un altro spunto per un lavoro futuro potrebbe essere quello di utilizzare Dense Optical Flow per riuscire a studiare la direzione di moto della palla.

Detector e Optical Flow però rallentano il codice e complicherebbe notevolmente la nostra pipeline di processing. Infatti, il nostro algoritmo riesce a tenere una complessità generalmente bassa, riuscendo a processare circa 20/25 frame per secondo dei video di riferimento e potrebbe essere utilizzato per un'analisi praticamente in real-time senza eccessivi ritardi avendo a disposizione dell'hardware leggermente più potente di quello che dispone un semplice pc.