



UNIVERSITÀ DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Discussione Finale

Computer Vision applicata al Basket:

Detection canestri

Anno accademico 2020/2021

Supervisori:

Nicola Conci
Andrea Rosani

Studente:

Dennis Barchetti

INDICE:

Abstract

1. Introduzione	3
1.1 Motivazioni.....	3
1.2 Definizione del problema.....	3
1.3 Miglioramenti Preliminari.....	4
1.4 Optical Flow.....	5
2. Metodologia	6
2.1 Descrizione dataset.....	6
2.2 Soluzione proposta	7
2.3 Criticità del dato.....	8
3. Risultati Ottenuti	11
3.1 Premessa.....	11
3.2 Confronto tra gli algoritmi.....	12
3.3 Valutazione dei risultati.....	13
4. Conclusioni	

Sitografia & Bibliografia

Abstract:

Un **sistema di visione artificiale** è un apparato elettronico che esegue funzioni di **visione artificiale**. Esso integra una telecamera, un sistema di illuminazione ed un software (esterno) a quest'ultima.¹

Lo scopo della tesi è quello di replicare la vista umana al fine di contare i canestri eseguiti durante una partita di basket.

In particolare ho cercato di migliorare un precedente algoritmo formulato assieme ad altri due compagni di corso nel quale, attraverso delle soglie poste sotto e sopra al canestro, verifichiamo la posizione della palla durante l'azione riuscendo a stabilire se sia passata o meno attraverso esso.

Per prima cosa ho raggruppato alcune operazioni all'interno dello stesso metodo al fine di compiere un numero maggiore di operazioni contemporaneamente.

In secondo luogo ho aggiunto la possibilità di “tarare” il colore che dovrà poi essere filtrato. Per rendere l'immagine meno rumorosa infatti è stata effettuata una sogliatura hsv e una sottrazione dello sfondo (KNN-BackgroundSubtraction²).

Successivamente ho applicato l'optical flow, l'idea è stata quella di cercare di valutare non solo la presenza o meno dell'oggetto all'interno di una porzione di video (ROI) ma anche il moto che essa sta descrivendo ovvero stabilire se la palla stia effettivamente **scendendo**.

1. Introduzione:

1.1. Motivazioni:

Ho scelto di continuare il progetto svolto durante il corso di elaborazione e trasmissione delle immagini perchè sono rimasto affascinato dal mondo della computer vision e avevo voglia di mettermi alla prova cercando di migliorare la soluzione precedentemente trovata.

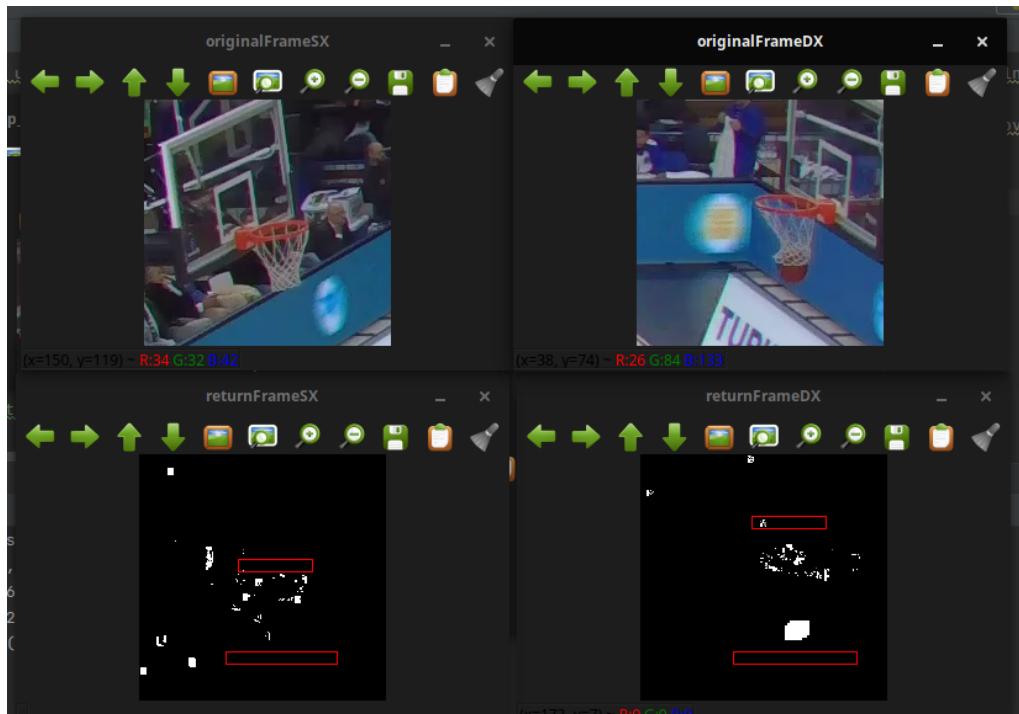
Inoltre, lavorare a questo progetto, mi ha permesso di migliorare le mie competenze in ambito di sviluppo software e soprattutto ha migliorato le mie conoscenze in python, riguardo la libreria numpy per l'elaborazione delle matrici e di opencv per quanto riguarda la computer vision.

1.2. Definizione del problema:

Il compito dell'algoritmo è quello di contare i canestri durante una partita di basket sia per canestro sinistro che destro. Il tutto deve essere determinato dal video catturato dalla singola telecamera posta sopra agli spalti a livello della linea di mezzeria.

1.3. Miglioramenti Preliminari :

Il punto di partenza è stato il progetto eseguito durante il corso.



Nell'immagine sopra si può vedere l'output delle operazioni che vengono eseguite.

Per prima cosa sono state determinate delle regioni di interesse (ROI) al fine di eseguire tutti i calcoli e le valutazioni solo per la porzione di immagine di nostro interesse.

Viene fatta una sogliatura hsv basata su un range di colori (lower color e upper color) al fine di isolare il colore della palla poi, sul risultato di essa, viene applicato un background subtractor basato su KNN³ il quale si occupa di sottrarre dal frame lo sfondo al fine di non valutare eventi in quella zona.

Inizialmente ho cercato di migliorare l'algoritmo dal punto di vista di velocità computazionale.

Per rendere questo possibile ho cercato di raggruppare in un minor numero possibile di funzioni le varie operazioni eseguite sulle immagini.

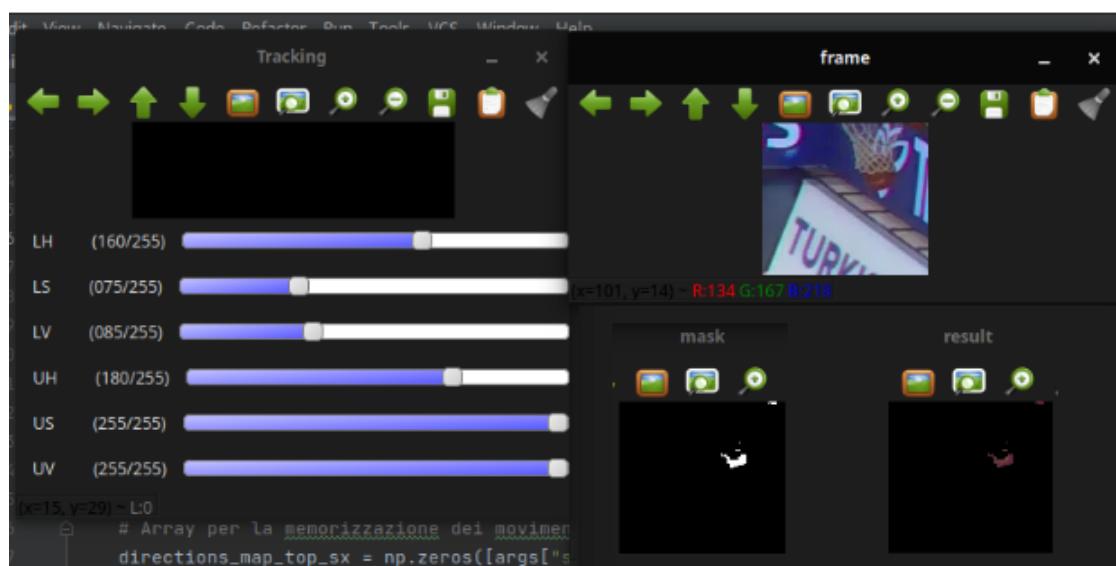
Infatti a differenza di prima, quando calcolavo sia per destra che per sinistra ROI, sogliatura hsv e sottrazione sfondo ora gestisco il tutto grazie a funzioni in grado di lavorare sia a sinistra che a destra contemporaneamente.

Per fare ciò ho riscritto i metodi contenute nella classe SrtandardVideoOperation in modo che essi ritornassero 2 o più valori. I valori di ritorno sono stati memorizzati in una tupla e poi trattati singolarmente in caso di necessità oppure passate interamente come argomento.

Per misurare effettivamente quanto più veloce riusciva ad eseguire il codice il mio pc ho fatto partire un timer calcolando il tempo necessario per l'esecuzione in millisecondi.

Prendendo come riferimento il quarto tempo sono riuscito a passare da un tempo complessivo di **2223.68 ms a 1177.27 ms**.

Ho aggiunto infine la possibilità di fare una calibrazione per quanto riguarda il range di colore che dovrà essere filtrato.



1.4 Optical Flow

Successivamente ho sfruttato il **flusso ottico (optical flow)** ovvero una modalità di percezione visiva del movimento degli oggetti in relazione al soggetto, che deve venire elaborata a livello corticale comparando diversi parametri come la *velocità, l'intensità di luce, la posizione del corpo, della testa, dei movimenti oculari.*⁴

Ciò è implementato in openCV attraverso una funzione chiamata `cv.calcOpticalFlowFarneback`⁵

Essa restituisce il **dense optical flow** del frame corrente (ovvero per ogni pixel) ricevendo in ingresso due frame (`gray_previous` e `gray`) rappresentanti istante attuale e passato (frame precedente).

$$I(x(t), y(t), t) = \text{Constant}$$

Take derivative of both sides wrt time:

$$\frac{d I(x(t), y(t), t)}{dt} = 0$$

(using chain rule)

$$\frac{\partial I}{\partial x} \frac{dx}{dt} + \frac{\partial I}{\partial y} \frac{dy}{dt} + \frac{\partial I}{\partial t} = 0$$

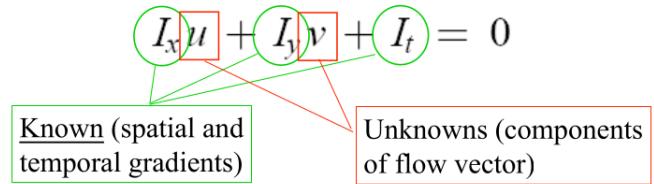
$\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}$ (spatial gradient; we can compute this!)

$\frac{dx}{dt}, \frac{dy}{dt} = (u, v)$ (optical flow, what we want to find)

$\frac{\partial I}{\partial t}$ (derivative across frames. Also known, e.g. frame difference)

Vincoli Optical Flow:⁶

Al fine di rendere possibile la misura del flusso ottico
dobbiamo assumere **costante** la luminosità del punto nella scena in differenti istanti temporali.



Optical Flow is **CONSTRAINED** to be on a line !
(equation has for a $u + b v + c = 0$)

Attraverso manipolazioni matematiche possiamo dimostrare che l'optical flow che noi vorremmo calcolare sarà vincolato ad una equazione lineare con **u** e **v** incognite rappresentanti proprio l'**optical flow lungo le sue componenti x e y**.

Per risolvere il nostro problema il calcolo dell'optical flow è gestito dal metodo `cv.calcOpticalFlowFarneback` che attraverso l'algoritmo di Farneback calcola il **dense optical flow** ovvero il flusso ottico per ogni pixel del frame.

Questi movimenti verranno poi sogliati nelle possibili direzioni (up, **down**, right, left). Ovviamente per lo scopo prefissato di isolare i movimenti dall'alto verso il basso porremmo particolare attenzione alla direzione down.

2. Metodologia:

2.1 Descrizione dataset

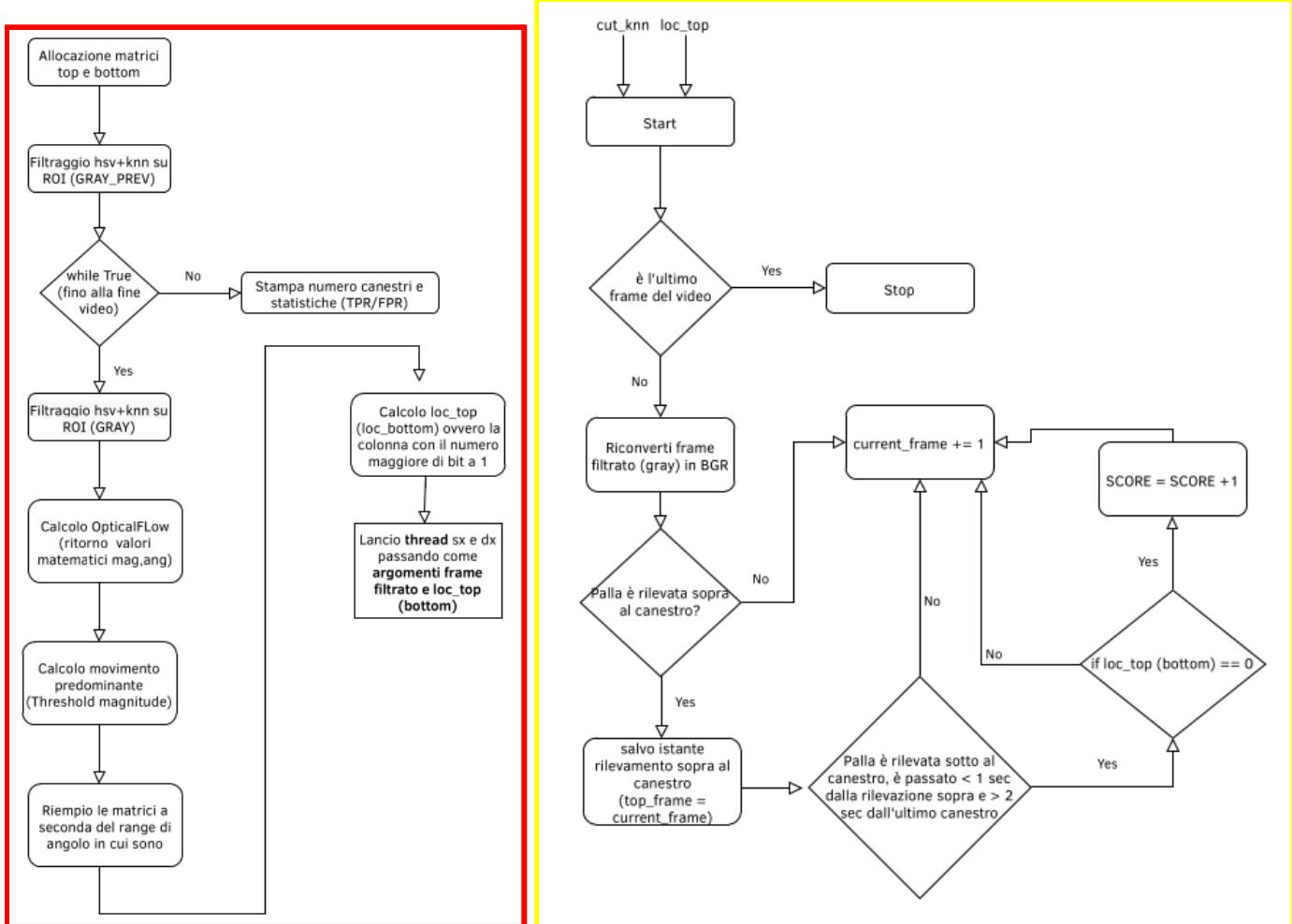
Lo sviluppo dell'algoritmo si è basato quasi interamente sul quarto tempo della partita. Esso, e gli altri 3 tempi della partita costituiscono un video con le seguenti caratteristiche:

- Numero telecamere : **1 telecamera fissa** con copertura del campo
- Video Input: definizione : **4k** (3840x2160)
- formato : nome_video.**ASF**
- Frame per secondo : **25 frame/sec**

Nota: sebbene la telecamera fosse sempre la stessa per tutti i tempi essa è soggetta a **piccole vibrazioni e movimenti**. Di conseguenza sarà necessario qualche piccolo aggiustamento riguardo alle ROI nei vari tempi.



2.2 Soluzione Proposta:



Nel diagramma a sinistra (in rosso) è esposto il funzionamento del main all'interno del quale vengono eseguite tutte le operazioni di filtraggio, di calcolo dell'optical flow e della direzione predominante.

Attraverso delle variabili globali vengono salvati numero di canestri e statistiche, stampate a video alla fine del ciclo while. Per calcolare quest'ultime ho utilizzato due array precedentemente riempiti di frame corrispondenti al canestro realmente esistente.

Al momento dell'incremento dello score eseguo un controllo attraverso un altro ciclo for al fine di valutare se il frame trovato dall'algoritmo corrisponda ad uno dei frame presenti nell'array.

Così facendo marchio questi valori come "canestro rilevato correttamente" per poi calcolare gli indici TPR ed FPR in automatico.

Nel secondo diagramma (in giallo) è invece mostrato l'algoritmo responsabile della determinazione di un canestro (sinistro o destro).

Per le due porzioni di campo esaminate passiamo ai relativi thread dei valori come argomento, cut_knn è una tupla contenente nella prima posizione (cut_knn[0]) la ROI di sinistra (cut_knn[1]: ROI destra) a cui è stato applicato nel main un threshold hsv ed una sottrazione dello sfondo.

loc_top (loc_bottom) sono invece gli indici della colonna della matrice accumulatrice istanziata ad inizio programma rappresentanti un movimento per colonna. Se essi sono uguali a 0 il movimento sarà effettivamente considerato verso il basso.

2.3 Criticità del dato

Inizialmente ho cercato di basarmi interamente sull'optical flow tuttavia ho riscontrato troppo rumore video causato da vari fattori.

L'algoritmo così strutturato riusciva con i dovuti valori di "threshold" del magnitude, "size" della matrice accumulatrice a raggiungere ottimi valori di TPR (circa 90%) ma allo stesso tempo con quelle configurazioni il False Positive Ratio aumentava considerevolmente (200-300%).

Cercando di rendere il sistema meno reattivo d'altro canto il TPR non riusciva ad andare oltre al 50% mentre l'FPR era pari a 80-90%.

Per fare delle prove attraverso dei cicli for e salvando tutto direttamente su file, ho eseguito varie simulazioni per valori di "threshold" e "size" prima e muovendo le roi successivamente.

Questa instabilità può essere dovuta dalla sensibilità dell'optical flow nei confronti di cambi di illuminazione (molto frequenti a causa degli annunci pubblicitari) e di persone presenti per alcuni istanti nella ROI corrispondente al canestro, esse infatti muovendosi causano spesso movimenti involontariamente interpretati erroneamente dal sistema.

Di seguito alcuni esempi:



Nelle due immagini a sinistra (due frame consecutivi) possiamo notare come il cartellone mostri la pubblicità attraverso uno scorrimento di essa subito sotto alla fine della rete del canestro.

In questa zona noi calcoliamo **loc_bottom** la quale potrebbe essere un falso positivo.



In queste due invece avviene un'altra situazione "critica".

La palla colpisce il ferro e quindi è rilevata la sua presenza nell'area sovrastante al canestro.

Tuttavia essa non entra in rete, non scende mai all'interno della ROI bottom, dove tuttavia viene individuato l'arbitro che muovendosi fa scattare l'incremento dello score.



Nella sequenza sopra è raffigurata una sequenza in cui a causa del particolare affollamento dell'area sottostante al canestro il sistema fallisce.

3. Risultati Ottenuti:

Premessa:

L'algoritmo sviluppato si basa su varie soglie, determinate in modo euristico.

In linea teorica abbiamo:⁷

-"threshold" rappresenta un valore minimo che dovrà avere il magnitude in output dall'optical flow.
Nella variabile move_sense andremo a considerare solo gli angoli tali per cui il magnitude ad essi corrispondente sia abbastanza grande.
Attraverso un'operazione di moda andiamo poi a ricavare l'angolo maggiormente ripetuto stabilendo una direzione predominante per quel frame.

```
move_sense_top = ang_top_sx[mag_top_sx > args["threshold"]]
move_mode_top = mode(move_sense_top)[0]

move_sense_bottom = ang_bottom_sx[mag_bottom_sx > args["threshold"]]
move_mode_bottom = mode(move_sense_bottom)[0]

return move_mode_top, move_mode_bottom
```

-"size" che rappresenta la dimensione intesa come numero di righe della nostra matrice accumulatrice.

```

# Array per la memorizzazione dei movimenti
directions_map_top_sx = np.zeros([args["size"], 5])
directions_map_bottom_sx = np.zeros([args["size"], 5])

directions_map_top_dx = np.zeros([args["size"], 5])
directions_map_bottom_dx = np.zeros([args["size"], 5])

"""
0      0      0      0      0      *
0      0      0      0      0      |
0      0      0      0      0      |
0      0      0      0      0      |
0      0      0      0      0      |
0      0      0      0      0      |
0      0      0      0      0      |
0      0      0      0      0      |
0      0      0      0      0      *
down    right   up     left    wait   size
"""

```

```

move_mode_top = move_mode[0]
move_mode_bottom = move_mode[1]

if 5 < move_mode_top <= 140:
    directions_map_top[-1, 0] = 1
    directions_map_top[-1, 1:] = 0
    directions_map_top = np.roll(directions_map_top, -1, axis=0)
elif 140 < move_mode_top <= 190:
    directions_map_top[-1, 1] = 1
    directions_map_top[-1, :1] = 0
    directions_map_top[-1, 2:] = 0
    directions_map_top = np.roll(directions_map_top, -1, axis=0)

```

Per un certo intervallo di angoli viene infatti riempita la colonna corrispondente al movimento e poi vengono srotolati tutti i valori di ogni colonna lungo axis=0, in numpy l'asse delle righe.

Il numero di righe su cui viene fatta questa operazione determina quanto il sistema sia reattivo poiché per ogni iterazione verrà eseguita una media (lungo le righe) e viene preso l'indice della colonna con valore massimo.

```

# CALCOLO CANESTRI SX:
loc_top_sx = directions_map_top_sx.mean(axis=0).argmax()
loc_bottom_sx = directions_map_bottom_sx.mean(axis=0).argmax()
#lancio thread

# CALCOLO CANESTRI DX:
loc_top_dx = directions_map_top_dx.mean(axis=0).argmax()
loc_bottom_dx = directions_map_bottom_dx.mean(axis=0).argmax()
#lancio thread

```

3.1. Confronto tra algoritmi:

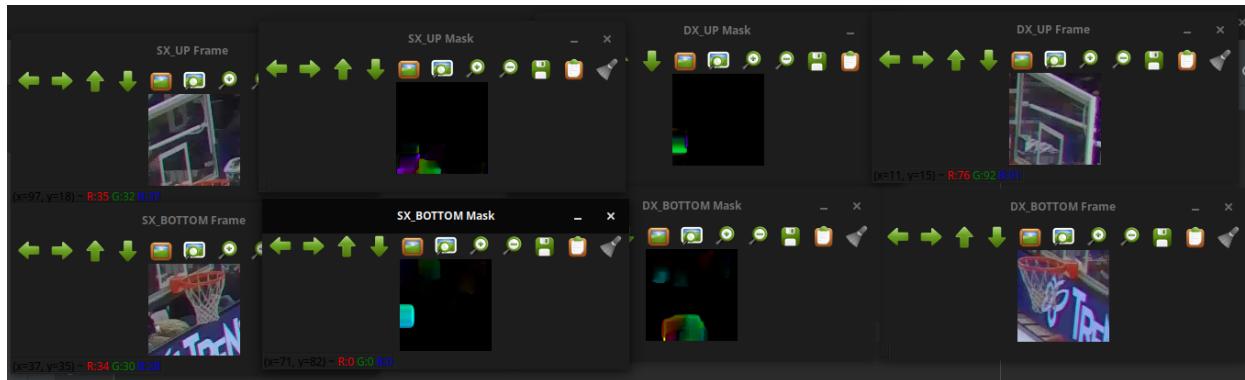
Per valutare la bontà della soluzione ci siamo basati sui seguenti indici:

$$TPR = \frac{\text{numero di canestri rilevati corretti}}{\text{numero di canestri da rilevare}} \quad FPR = \frac{\text{numero di canestri rivelati fasulli}}{\text{numero di canestri da rilevare}}$$

Algoritmo solo Optical Flow:

Per prima cosa ho provato a realizzare un algoritmo interamente basato sull' optical flow.

Come spiegato prima infatti viene calcolato il movimento predominante nelle due roi una posta sopra e una sotto al canestro;



una volta rilevato il movimento predominante, se esso è verso il basso e quindi la colonna riempita di 1 è quella con indice 0, incremento dei contatori. In particolare se rilevo un movimento dall'alto al basso nella roi superiore aumento una variabile **contagiù_top** di 1, se viene rilevato nella roi sottostante al canestro invece incremento **contagiù_bottom** sempre di 1. Per il secondo contatore ho inserito dei controlli al fine di incrementarlo solo qualora fosse passato un certo intervallo di tempo dalla rilevazione nella roi sopra.

Una volta raggiunta una certa soglia (e se è passato un certo tempo dall'ultimo canestro) viene effettivamente stampato il frame e segnato canestro.

```
#CALCOLO CANESTRI SX:
loc_top_sx = directions_map_top_sx.mean(axis=0).argmax() # SX e DX si sono girati da qualche parte!!!
loc_bottom_sx = directions_map_bottom_sx.mean(axis=0).argmax()

if (loc_top_sx == 0 and contagiu_top_sx == 0): # situazione base
    topFrame_sx = frame_counter_sx # rilevo la palla e segno il frame corrispondente

if (loc_top_sx == 0): # situazione base
    contagiu_top_sx += 1

if (loc_bottom_sx == 0 and (frame_counter_sx - topFrame_sx > 6)): # è passato almeno 1/4 secondo da rilevazione sotto e sopra
    contagiu_bottom_sx += 1

if (frame_counter_sx - topFrame_sx > 50):
    contagiu_top_sx = 0
    contagiu_bottom_sx = 0

if ((contagiu_top_sx > 2 and contagiu_bottom_sx > 2) and (frame_counter_sx - last_score_frame_sx > 75)): # sono passati 3 secondi dall'ultimo canestro
    last_score_frame_sx = frame_counter_sx
    for i in np.nditer(Frame_canestri_sx):
        if ((last_score_frame_sx in range(i, i + 45)) or (last_score_frame_sx in range(i - 45, i))):
            canestri_corretti_sx += 1
            print("CANESTRO SX CORRETTAMENTE RILEVATO al frame = ", last_score_frame_sx)
    last_score_frame_dx = frame_counter_sx
    print("sx: ", frame_counter_sx)
    canestri_sx += 1
```

Algoritmo “unione” di soglie e optical flow come controllo ulteriore:

Dopo aver constatato che l'approccio con il solo utilizzo dell'optical flow era troppo rumoroso ed instabile, (a seconda della roi considerata infatti ottengo risultati molto diversi), ho deciso di unire il calcolo dell'optical flow al precedente algoritmo basato su due soglie rettangolari poste una sopra e una sotto al canestro.

Quando la palla passa attraverso la prima e la seconda (con vari controlli al fine di evitare falsi positivi) viene valutato anche la direzione del movimento nella roi_bottom (quella subito sotto il canestro).

Lo scopo è di essere certi che la palla stia effettivamente scendendo quando attraversa la seconda soglia, essendo quest'ultima la responsabile di vari errori (tabellone pubblicità- giocatori che passano in mezzo).

Nel seguente algoritmo avremo quindi 4 roi dove verrà calcolato il dense optical flow e le 2 “vecchie” roi nel quale si trovano le soglie.

TPR_SX:	0,91	TPR_SX:	0,78
TPR_DX:	0,95	TPR_DX:	0,79
FPR_SX:	0,14	FPR_SX:	0,09
FPR_DX:	0,26	FPR_DX:	0,15

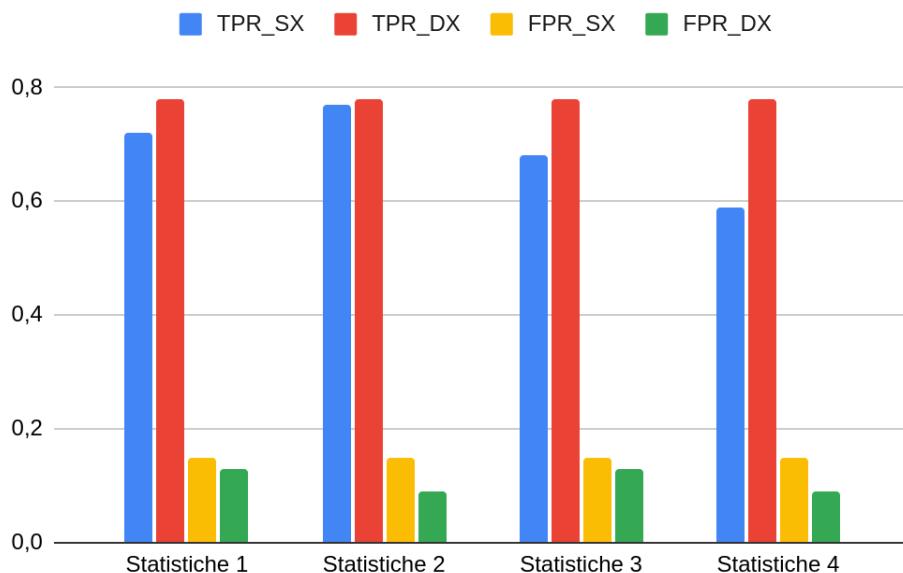
Come risultato abbiamo un sensibile calo dell'FPR sia a destra che a sinistra ma tuttavia anche un calo del TPR, questo perché abbiamo reso più stringente il controllo.

Conclusioni:

	ROI_LEFT_TOP	ROI_LEFT_BOTTOM	ROI_RIGHT_TOP	ROI_RIGHT_BOTTOM
Statistiche 1	(570,1000)x(670,1100)	(570,1010)x(670,1110)	(3320,950)x(3420,1050)	(3320,980)x(3420,1080)
Statistiche 2	(590,1000)x(690,1100)	(590,1010)x(690,1110)	(3320,950)x(3420,1050)	(3320,980)x(3420,1080)
Statistiche 3	(600,1000)x(700,1100)	(600,1010)x(700,1110)	(3320,950)x(3420,1050)	(3320,980)x(3420,1080)
Statistiche 4	(570,980)x(670,1080)	(570,990)x(670,1090)	(3320,950)x(3420,1050)	(3320,980)x(3420,1080)

	TPR_SX	TPR_DX	FPR_SX	FPR_DX
Statistiche 1	0,72	0,78	0,15	0,13
Statistiche 2	0,77	0,78	0,15	0,09
Statistiche 3	0,68	0,78	0,15	0,13
Statistiche 4	0,59	0,78	0,15	0,09

Statistiche al cambio della ROI



L'algoritmo basato su soglie con l'aggiunta del controllo optical flow è stato eseguito su varie simulazioni.

Come si può vedere dal grafico a seconda della roi considerata.

Questo a causa del fatto che l'optical flow utilizzato per determinare la presenza di un movimento di un oggetto nel frame è un **dense optical flow**.

Infatti per definizione esso opera su tutti i pixel dei due frame presi in esame, ovviamente quindi una piccola variazione nella regione di interesse può portare a ritenere predominante un movimento anziché un altro.

Attraverso le simulazioni è possibile vedere tuttavia come si riescano a trovare dei valori accettabili sia per destra che per sinistra.

Dalla tabella possiamo notare inoltre come la situazione sul canestro destro sia più stabile.

Questo è probabilmente causato dal fatto che subito sotto alla rete del canestro sinistro vi è il tabellone, esso viene considerato come elemento in primo piano poiché non è coperto dalla rete (cosa che avviene a destra).

A causa di ciò a seconda della porzione di tabellone presente nella roi e del movimento che viene visualizzato sullo schermo abbiamo risultati differenti.



Sitografia & Bibliografia:

1. **Sistema di visione artificiale:** <https://www.britannica.com/technology/computer-vision>
2. **KNN_python:** https://docs.opencv.org/3.4/db/d88/classcv_1_1BackgroundSubtractorKNN.html
3. **KNN:** https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
4. **Optical Flow:** https://en.wikipedia.org/wiki/Optical_flow
5. **Optical Flow_pyphon:** https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html
6. **Vincoli Optical flow:** Optical Flow Estimation, Robert Collins CSE486, Penn State
7. **Programma di riferimento:**
<https://medium.com/@ggaighernt/optical-flow-and-motion-detection-5154c6ba4419>