UNIVERSITÀ
DI TRENTO

Progetto di Elaborazione e Trasmissione delle Immagini:

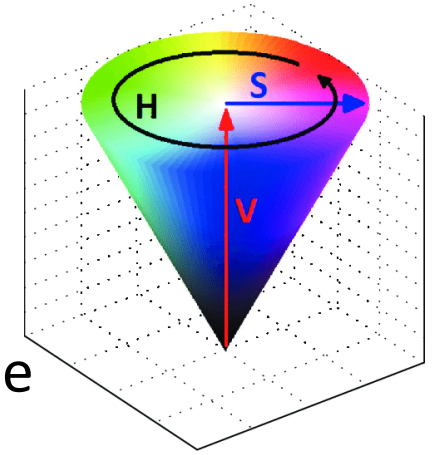
Gruppo 8: Barchetti Dennis, Bortolin Samuel, Grassi Alessandro

Computer Vision applicata al Basket: Rilevazione canestri fatti

Introduzione e prerequisiti:

- Lo scopo del progetto è quello di sviluppare un algoritmo in grado di contare i canestri effettuati ed implementarlo poi attraverso Python.
- Sviluppo basato su una partita suddivisa in quattro tempi.
- Aspettative: TPR : circa 80%, FPR il più basso possibile.
- Prerequisiti: - installare python3 sul PC;
  - installare le librerie: numpy/opencv/imutils.
- Caratteristiche tecniche dei video di test:
  - Numero telecamere: una telecamera fissa con copertura del campo;
  - Video Input: risoluzione: 4k (3840x2160), formato: video.asf;
  - Frame per secondo: 25.

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.
- Di seguito è mostrata una parte del codice utilizzata per trovare i parametri hsv rappresentanti l'arancione della palla da basket:

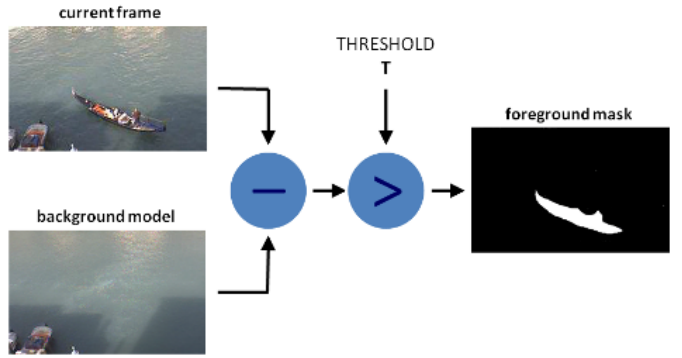
```
# Creazione di una finestra denominata "tracking" (su cui muovere le trackbar)
cv2.namedWindow("Tracking", cv2.WINDOW_NORMAL)

# 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
```

VALORI HSV RICAVATI:

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

```
# 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)
```

Sottrazione dello sfondo KNN



https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html

- KNN (K Nearest Neighbors) e 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.

- Nella figura 1.8 il cerchio verde rappresenta l'input (pixel), i triangoli rossi la prima classe mentre i quadrati blu la seconda.
  Possiamo vedere come il risultato cambi a seconda del valore dei k elementi presi in esame.

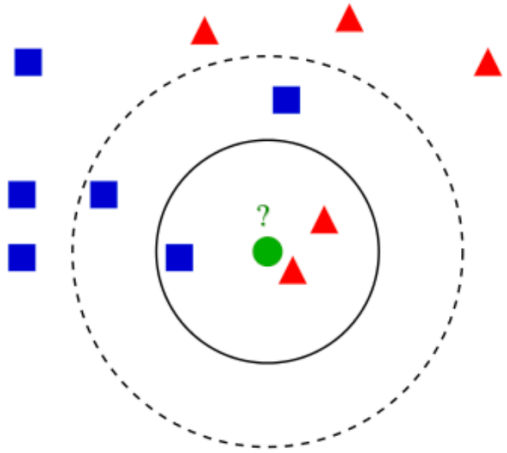


Figura 1.8: rappresentazione di un grafico dove si vede la differenza di avere un key=3 e un key=5, cambia totalmente il risultato.
Fonte: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg

Sviluppo (SVO):

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 e stato fatto per due motivi:

1. la possibilità sperimentare con i metodi proposti
2. la possibilità di aggiungerne altri in rilasci futuri nel caso si scegliesse di espandere le sue funzionalità.

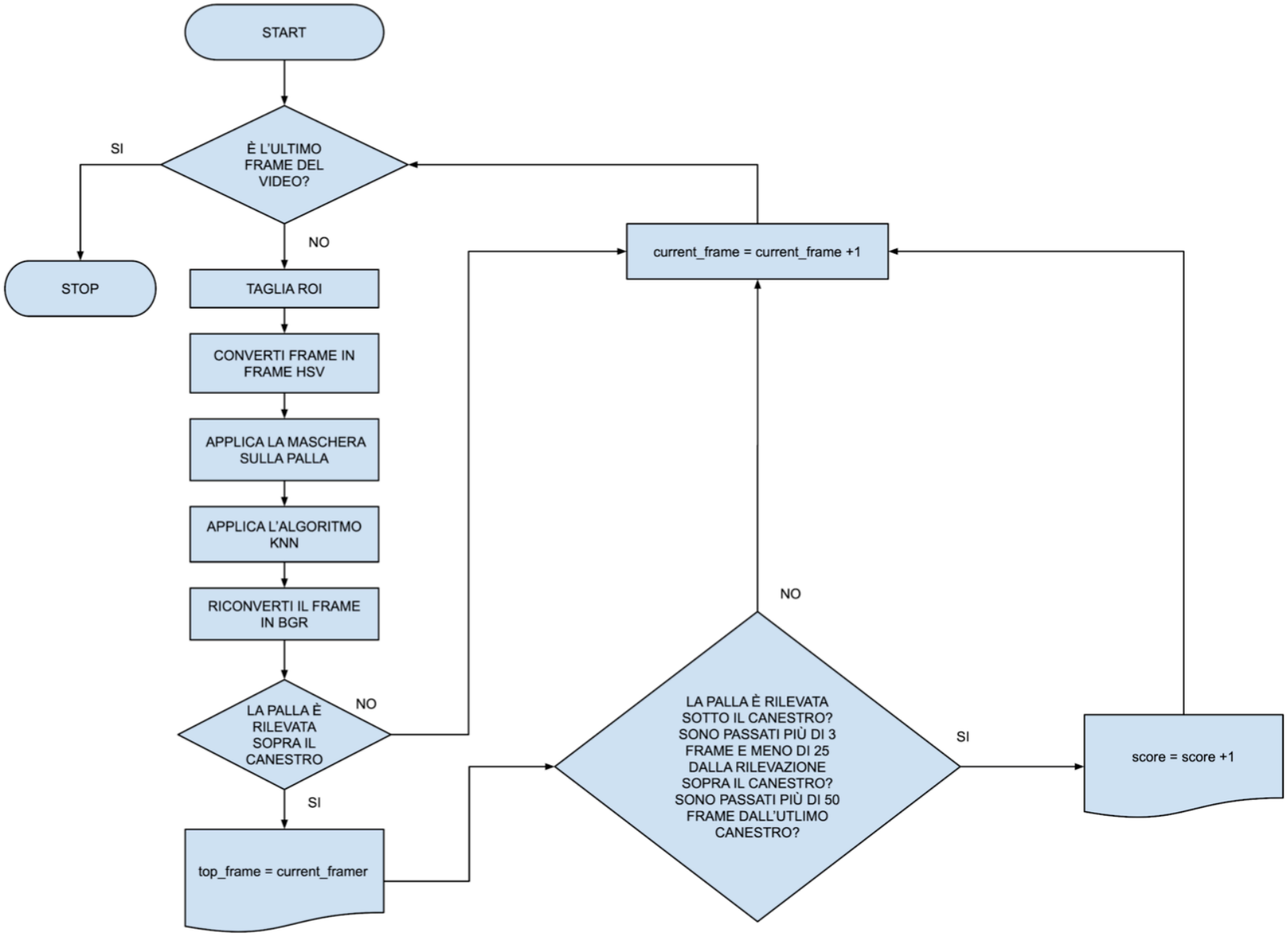
- __init__(self);
- set_left(upper_left, bottom_right);
- set_right(upper_left, bottom_right);
- video_cutter(frame, upper_left, bottom_right);
- cut_left(startingFrame);
- cut_right(startingFrame);
- draw_rectangle(frameBGR, upperLeft, bottomRight, color_string);
- get_hsvmask_on_ball(frame_hsv);
- get_knn_on_left_frame(frame);
- get_knn_on_right_frame(frame);
- find_circles(frame_to_scan, frame_to_design);
- spotBallOnTop_left(greyScaleFrame);
- spotBallOnMiddle_left(greyScaleFrame);
- spotBallOnBottom_left(greyScaleFrame);
- spotBallOnTop_right(greyScaleFrame);
- spotBallOnMiddle_right(greyScaleFrame);
- spotBallOnBottom_right(greyScaleFrame);
- countWhitePixels(rows, colRange, greyScaleFrame).

Algoritmo countWhitePixels

- Identifica la presenza della palla se vede almeno 16 pixel bianchi di fila con al massimo un solo pixel nero in mezzo.
- La complessità dell'algoritmo nel caso pessimo e θ(n*m) dove n e il numero di righe da controllare e m e il numero di colonne da controllare.

```
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
```

Main:


START
È L'ULTIMO FRAME DEL VIDEO?
SI
STOP
NO
TAGLIA ROI
CONVERTI FRAME IN FRAME HSV
APPLICA LA MASCHERA SULLA PALLA
APPLICA L'ALGORITMO KNN
RICONVERTI IL FRAME IN BGR
LA PALLA È RILEVATA SOPRA IL CANESTRO
NO
SI
top_frame = current_framer
current_frame = current_frame +1
NO
LA PALLA È RILEVATA SOTTO IL CANESTRO? SONO PASSATI PIÙ DI 3 FRAME E MENO DI 25 DALLA RILEVAZIONE SOPRA IL CANESTRO? SONO PASSATI PIÙ DI 50 FRAME DALL'UTLIMO CANESTRO?
SI
score = score +1

Risultati e conclusioni:

- Inizialmente l'algoritmo è stato sviluppato e testato su delle sequenze.
- Per tagliare le sequenze abbiamo usato il comando:
  ffmpeg -i input.asf -ss 00:01:00 -to 00:01:05 -q 0 output.asf
- Per completare la nostra sperimentazione ed avere un'idea della **performance complessiva** del nostro algoritmo, lo abbiamo mandato in esecuzione su tutta la partita, adattando la ROI per definire l'area del canestro da analizzare differentemente nei vari quarti (a causa del movimento della videocamera):

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)

Analisi dell’efficacia:

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.

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

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

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=3/9=0.33
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		✖
27:57		✖

2° quarto SX	TPR=13/16=0.81	FPR=4/16=0.25
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		×
10:36	×	
10:58	✓	✓
13:24	✓	✓
14:04	×	
15:01	✓	✓
15:44	✓	✓
16:12	✓	✓
16:59	✓	✓
18:43	×	
19:37		×
19:51	✓	✓
20:38	✓	✓

2° quarto DX	TPR=9/11=0.82	FPR=1/11=0.09
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		×
18:27	×	
22:18	✓	✓
24:04	✓	✓

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

3° quarto SX	TPR=8/8=1	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	✔	✔

4° quarto SX	TPR=20/22=0.91	FPR=3/22=0.14
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	✓	✓
9:17	✓	✓
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	✓	✓
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	✓	✓

TPR 3° e 4° quarto	FPR 3° e 4° quarto
SX=28/30=0.93	SX=6/30=0.20
DX=29/31=0.94	DX=9/31=0.29
TOT=57/61=0.93	TOT=15/61=0.25

TPR complessivo	FPR complerssivo
SX=55/62=0.89	SX=15/62=0.24
DX=47/51=0.92	DX=13/51=0.25
TOT=102/113=0.90	TOT=28/113=0.25

Lavoro futuro:

- L'obbiettivo di un eventuale implementazione futura è quello di migliorare il parametro FPR, il quale si è rivelato decisamente più critico rispetto al TPR.
- Calcolando il false discovery rate, $fdr = \frac{28\ fasulli}{(102\ corretti + 28\ fasulli)}$ si evince che il 78,5% dei canestri rilevati risultano realmente esistenti mentre **il 21,5% sono fasulli.**
- YOLO/MaskRCNN: per fare in modo che l'algoritmo riconosca proprio la palla consentendoci di togliere rumore e oggetti dello stesso colore;
- Dense Optical Flow: per riuscire a studiare la direzione di moto della palla.