Main del code, fase di testing risultati e cambiamenti RoI

#todo samuel: descrizione main (ne sono stati fatti 2 uno per SX e uno per DX); retest del filtro mediano e quello gaussiano → inutili anzi peggiorano; descrizione della fase di testing (anche breve accenno dei ritagli con ffmpeg) e dei risultati ottenuti (con verità raccolta a mano): differenze tra i vari quarti e i relativi problemi con le RoI nei vari quarti, differenza tra DX e SX a causa della posizione della videocamera; conclusioni sul progetto e possibili miglioramenti futuri

#todo: Cartella condivisa condividendo parti di video dove l'algoritmo presenta errori

Codice del main (diversificazione per SX e DX, vedi 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. (tentativo di fare due thread uno per sx e uno per dx)

```
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)
```

```
top_frame = middle_frame = bottom_frame = frame_counter = last_score_frame = 0
```

indicatori dei frame che saranno utili nell'analisi (vedi in main 5 che vengono differenziati per sx e dx)

#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:", frame_counter)
                        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
                # (vedi in main 5 che cambiano delle cose)

                print("middle FRAME:", frame_counter)
                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_frame) > 25:	if svo.spotBallOnBottom_right(finalFrame) and 3 < (frame_counter - top_frame) < 25 and (frame_counter - last_score_frame) > 25:

```
        # 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 25 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
        (vedi in main 5 che cambiano delle cose)

                print("bottom FRAME:", frame_counter)
                print("score")
                last_score_frame = frame_counter
                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
```

to describe the main code:

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=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

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 ma comunque molto buoni con TRP = 0.87 e FPR = 0.25

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

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

3° e 4° quarto . . .
Analisi per i tempi in che è stata pensata/perfezionata TPR = 0.93 e FPR = 0.25

Accuratezza detection sinistra vs detection di destra:

A DX TPR più alto (0.92 contro il 0.89 di SX) e anche FPR più alto (0.25 contro il 0.24 di SX, questo in generale ma nei primi di quarti il FPR a SX è 0.28 contro i 0.20 di DX, spostamento della videocamera influenza probabilmente ciò, dato che negli ultimi 2 quarti a SX è 0.20 contro il 0.29 di DX), i false positive sono dovuti al problema del monitor che aiuta a far segnare canestro anche quando non c'è
In conclusione l'algoritmo realizzato si presta bene, in complessivo le performance riportate dopo l'analisi complessiva della partita sono TPR = 0.90 e FPR = 0.25

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

Future work:

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 trovando un modo di ridurre il numero dei canestri fasulli, dato che circa il 22% dei canestri rilevati risulta essere fasullo (calcolato come 28 fasulli/(102 corretti + 28 fasulli) e quindi quelli rivelati che risultano essere veri sono circa il 78%). 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 . . . → 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.