UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Sviluppo di un algoritmo per il tracking di volti mediante eventi e frame da camere DVS

Development of an algorithm to track faces using events and frames from DVS cameras

Relatore:
Dott. Mancini Adriano

Laureando:
Ali Waqar Badar

Anno Accademico 2020-2021

Indice

Capitolo 1

Introduzione

Il Face Traking è una tecnologia di Computer Vision che rileva e localizza la presenza di un volto umano in un'immagine o video digitale. La tecnologia è anonima e distingue solo un volto umano concentrandosi sui tratti del viso. Il face detection e face tracking sono tecniche per individuare i volti nelle immagini e nelle sequenze video; il riconoscimento facciale è una tecnica per identificare o verificare persone sconosciute utilizzando un database memorizzato di volti noti.
Le tecniche di elaborazione del volto umano per la trasmissione di video, inclusi d face detection, face tracking, e face recognition, hanno suscitato molto interesse nella ricerca a causa del loro valore in varie applicazioni, come la strutturazione video, l'indicizzazione, il recupero e il riepilogo. La ragione principale di ciò è che il volto umano fornisce informazioni complete per individuare l'aspetto di alcune persone di interesse.
Principalmente andremo a studiare mediante eventi e frame da camere Dynamic Vision System (DVS) il tracking dei volti e per fare ciò avremo bisogno di varie strumentazioni, cioè una camera ad eventi, e dei programmi per creale l'algoritmo che consente di effettuare il lavoro.

1.1 Analisi del progetto

L'obiettivo del progetto è quello di sviluppare un algoritmo in grado di rilevare il movimento del volto mandando da un lato gli eventi e dall'altro i frames, infatti si andrà ad analizzare sia le immagini che gli eventi per vedere principalmente come si sta comportando il volto, in base anche ad un input di tipo acustico (la sirena). L'algoritmo deve essere in grado di prendere le riprese fatte dalla videocamera ad eventi ed estrapolare i dati e di analizzare le immagini. L'introduzione delle **Event Cameras** nel campo della Computer Vision consente, a partire dalle sue caratteristiche, di colmare le mancanze delle camere *frame-based*. La novità di questi sensori consiste nell'introduzione di un nuovo tipo di dato, l'evento, il quale rappresenta il cambiamento di luminosità in un determinato punto nella scena il quale rappresenta il cambiamento di luminosità in un determinato punto (x,y) nella scena. Questo cambiamento di luminosità corrisponde al verificarsi di un

movimento nel punto stesso. In questo modo il sensore è in grado di raccogliere i dati in maniera asincrona, cioè non costante, e questo fa sì che, in assenza di movimento, non vengano raccolti dati provenienti dagli elementi statici della scena. Come evidente, tutti i problemi legati allo spreco di memoria e al tempo di elaborazione vengono risolti, difatti il volume di dati è direttamente proporzionale alla quantità di movimenti nella scena. La registrazione del cambiamento di luminosità consente di superare le difficoltà legate all'illuminazione della scena e in più, avendo il sensore un tempo di latenza pressoché nullo, si avrà una reattività molto elevata favorendo così l'acquisizione di oggetti in movimento ad alta velocità.
Il progetto e stato sviluppato principalmente dal linguaggio di programmazione **Python** con le varie libreria di **Mediapipe**.

1.2 Struttura della Tesi

La parte essenziale del progetto consiste nello sviluppo di un algoritmo per il tracking di volti mediante eventi e frame da camere *DVS*. Inizialmente si introduce, nello stato dell'arte, il concetto di *Drowsiness and Attention Detection Driver*, inoltre si specifica il concetto di *Face Detection, Face Tracking e Face Recognition*. In seguito si introducono tutti gli strumenti utilizzati per lo sviluppo del software e i metodi evidenziando gli *step* che hanno portato alla produzione dello stesso. Successivamente viene posto rilievo agli step che hanno portato alla produzione del software stesso consistenti in analisi del video, individuazione di un metodo di *Face Meshing* e quindi progettazione e sviluppo del programma. Infine si mostrano i risultati e i possibili sviluppi futuri del progetto.
Quindi la struttura dell'elaborato è individuabile nei seguenti punti:

- Introduzione ed esposizione dei concetti generali;
- Introduzione degli strumenti e dei metodi;
- Analisi e sviluppo del modulo software;
- Risultati;
- Conclusioni.

Capitolo 2

Stato dell'Arte

Le tecniche di elaborazione del volto umano per la trasmissione di video, inclusi d face detection, face tracking, e face recognition, hanno suscitato molto interesse nella ricerca a causa del loro valore in varie applicazioni, come la strutturazione video, l'indicizzazione, il recupero e il riepilogo. La ragione principale di ciò è che il volto umano fornisce informazioni complete per individuare l'aspetto di alcune persone di interesse.
Le tecniche di rilevamento dei volti presentate sopra sono principalmente per le immagini fisse piuttosto che per i video. Tuttavia, considerando ogni frame del video come un'immagine fissa, queste tecniche possono essere utilizzate per i video. Sebbene le tecniche di tracking dei volti basate su frame siano state dimostrate su immagini reali, la loro capacità di rilevare i volti nei video è ancora primitiva. Le prestazioni del detector possono diminuire per vari motivi, tra cui occlusioni e cambiamenti nelle condizioni di illuminazione e nelle pose del viso. Senza ulteriori informazioni, le risposte del rilevatore possono essere facilmente respinte, anche se indicano la presenza di un volto. Per fornire segmenti video più completi in cui tracciare la persona di interesse, è quindi importante incorporare informazioni temporali in una sequenza video.

2.1 Drowsiness and Attention Detection Driver

Il numero sempre crescente di incidenti stradali nella CE a causa del diminuito livello di vigilanza del conducente è diventato un serio problema per la società. L'affaticamento del conducente derivante dalla privazione del sonno o dai disturbi del sonno è un fattore importante nel crescente numero di incidenti sulle strade di oggi. Le statistiche mostrano che una delle principali cause di incidenti stradali mortali o che causano lesioni è dovuta a conducenti con un livello di vigilanza ridotto. Rilevare automaticamente il livello di attenzione visiva dei conducenti con sufficiente anticipo per scaldarli sulla mancanza di un'adeguata attenzione visiva a causa della fatica può salvare una quantità significativa di vite e sofferenze personali. Pertanto, è importante esplorare l'uso di tecnologie innovative per risolvere

il problema del monitoraggio dell'attenzione visiva del conducente.
Il rilevamento della sonnolenza del conducente è una tecnologia per la sicurezza dell'auto che previene gli incidenti quando il conducente è assonnato. Vari studi hanno suggerito che circa il 20% di tutta la strada gli incidenti sono legati alla fatica, fino al 50% su determinate strade. L'affaticamento del conducente è un fattore significativo in un gran numero di incidenti stradali. Recenti statistiche stimano che ogni anno 1.200 morti e 76.000 feriti possono essere attribuiti alla fatica relativi *crash*. Lo sviluppo di tecnologie per rilevare o prevenire la sonnolenza al volante è un importante sfida nel campo dei sistemi di prevenzione degli incidenti. A causa del pericolo che la sonnolenza presenta sulla strada, devono essere sviluppati metodi per contrastare i suoi effetti. La disattenzione del conducente potrebbe essere il risultato di una mancanza di vigilanza durante la guida a causa della sonnolenza e della distrazione del conducente. La distrazione del conducente si verifica quando un oggetto o un evento attira un l'attenzione della persona lontano dal compito di guida. A differenza del conducente distrazione, la sonnolenza del conducente non comporta alcun evento scatenante ma, invece, è caratterizzato da un progressivo ritiro di attenzione dalle esigenze della strada e del traffico. Entrambi driver sonnolenza e distrazione, tuttavia, potrebbero avere lo stesso effetti, ovvero diminuzione delle prestazioni di guida, reazione più lunga tempo e un aumento del rischio di coinvolgimento in un incidente [1]. Figura 2.1, mostra lo schema a blocchi del sistema complessivo. Basato su Acquisizione video dalla telecamera che si trova di fronte il driver esegue l'elaborazione in tempo reale di un video in arrivo flusso per dedurre il livello di affaticamento del conducente se il la sonnolenza è stimata, quindi l'uscita viene inviata all'allarme sistema e l'allarme è attivato.

Figura 2.1: Schema a blocchi del sistema complessivo [2]

Molti sforzi sono stati riportati in letteratura sullo sviluppo di sistemi di monitoraggio della fatica basati su immagini in tempo reale non intrusivi [3], [4], [5], [6], [7], [8], [9]. Misurare la fatica sul posto di lavoro è un processo complesso. Ci sono quattro tipi di misure che vengono tipicamente utilizzate nella misurazione della fatica: fisiologica, comportamentale, soggettiva e misure di performance [10]. Un'importante misura fisiologica che è stata studiata per rilevare l'affaticamento sono stati i movimenti oculari. Diversi movimenti oculari sono stati utilizzati per misurare l'affaticamento come frequenza di ammiccamento, durata di ammiccamento, frequenza di chiusura lunga, ampiezza di ammiccamento, frequenza di saccade e velocità di picco della saccade.
La presente soluzione si concentra sulla rotazione della testa e sull'ammiccamento

degli occhi, due importanti segnali per determinare l'attenzione visiva del guidatore, per raccogliere statistiche sul livello di attenzione visiva del guidatore. Poiché il movimento della posa della testa di una persona e la direzione dello sguardo sono profondamente correlati alla sua intenzione e attenzione, la capacità di rilevare la presenza di attenzione visiva e/o determinare cosa sta guardando una persona stimando la direzione dello sguardo e l'orientamento del viso è utile per misurare il livello di attenzione del conducente. Uno dei nostri scopi principali è quello di recuperare e tracciare i tre gradi di libertà di rotazione di una testa umana, senza alcuna conoscenza preliminare della forma esatta della testa e del viso osservati. Questo algoritmo di stima dell'orientamento automatico è combinato con un robusto rilevamento e tracciamento delle caratteristiche facciali per fornire vincoli aggiuntivi per stimare la sequenza degli orientamenti della testa osservati.

2.1.1 Studi

Il **Rilevamento della Sonnolenza** (o **Drowsiness Detection**) può essere suddiviso in tre principali categorie: *(1) Basato sul veicolo (2) Basato sul comportamento (3) Basato sulla fisiologia*. La Figura 2.2 mostra i tre diversi approcci per il rilevamento della sonnolenza. Rilevamento della sonnolenza si basa su questi tre parametri. Una recensione dettagliata su queste misure forniranno informazioni sui sistemi attuali, problemi ad essi associati e i miglioramenti necessari da fare per creare un sistema robusto.

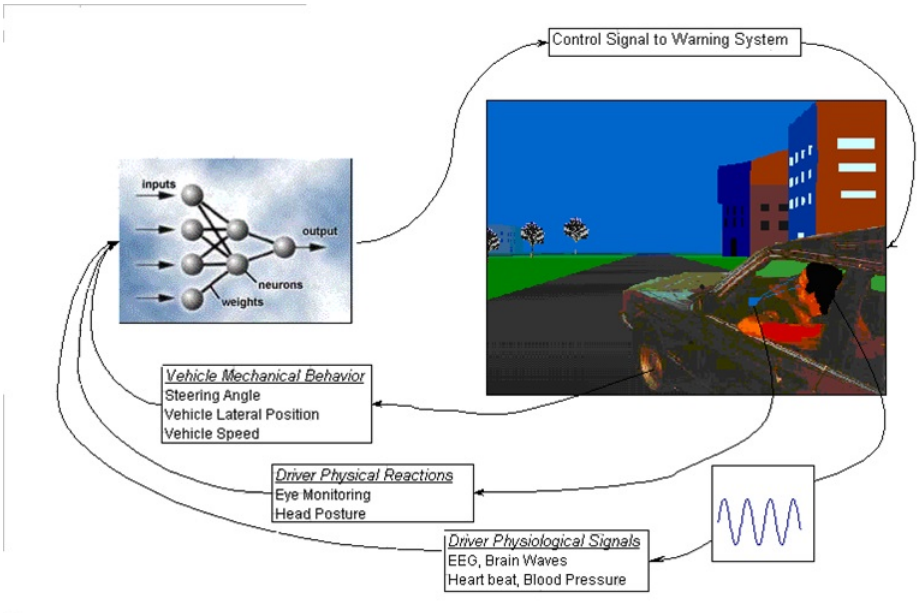


Figura 2.2: Approcci diversi per il rilevamento e gli avvertimenti della sonnolenza [11]

Misure basate sul veicolo: una serie di metriche, tra cui deviazioni dalla posizione della corsia, movimento dello sterzo ruota, pressione sul pedale dell'accele-

ratore, ecc, sono costantemente monitorati e qualsiasi cambiamento in questi che attraversa una soglia specificata indica un aumento significativo probabilità che il conducente sia assonnato.
Misure basate sul comportamento: il comportamento del conducente, inclusi sbadigli, chiusura degli occhi, sbattere le palpebre, posa della testa, ecc. viene monitorato tramite una telecamera e il conducente viene avvisato se viene rilevato uno qualsiasi di questi sintomi di sonnolenza.
Misure basate sulla fisiologia: la correlazione tra segnali fisiologici ECG (elettrocardiogramma) e EOG (elettrooculogramma). La sonnolenza viene rilevata attraverso il polso frequenza, battito cardiaco e informazioni sul cervello.

2.1.2 Scoperte e Tecniche di Drowsiness Detection

Diversi autori hanno proposto diversi approcci per Sistema di rilevamento della sonnolenza, la maggior parte dei quali utilizza l'ECG, Approcci basati sul veicolo.Un robusto sistema integrato in tempo reale piattaforma per monitorare la perdita di attenzione del conducente durante condizioni di guida diurne e notturne. Un sistema di drowsiness detection che utilizza sia l'attività cerebrale che quella visiva è presentato in questo documento [12]. L'attività cerebrale è monitorata utilizzando un singolo canale Elettro-Encefalo-Grafico (EEG). UN metodo per monitorare la sicurezza del conducente analizzando le informazioni correlato alla fatica utilizzando due metodi distinti [13]: monitoraggio del movimento dell'occhio ed elaborazione del segnale biologico. Un sistema di monitoraggio è progettato su base Android Smart-phone. Una macchina vettoriale di supporto (SVM) classifica una sequenza di segmenti video in guida attenta o non attenta eventi [14]. I risultati sperimentali mostrano che lo schema proposto offre un'elevata precisione di classificazione [15]. Lo scopo di questo documento è quello di massimizzare la quantità di sonnolenza correlata informazioni estratte da un insieme di elettroencefalogramma (EEG), Elettro-Oculo-Gramma (EOG) e Elettro-Cardio-Gramma (ECG).

Segnali durante un test di guida simulato. Il sistema è un prototipo software di questo sistema nei veicoli, dove le immagini che vengono catturati verranno elaborati utilizzando l'elaborazione delle immagini tecniche e di conseguenza emettere avvisi [16]. Un occhio robusto algoritmo di rilevamento viene introdotto per affrontare i problemi causati da cambiamenti nell'illuminazione e nella postura del conducente [17]. Sei le misure sono calcolate con percentuale di chiusura palpebrale, durata massima di chiusura e frequenza di lampeggio, media livello di apertura degli occhi, velocità di apertura degli occhi e velocità di chiusura degli occhi [18].

Se le tecnologie automobilistiche prevengono o almeno avvisano di affaticamento del conducente, quali sintomi emette il conducente che si può rilevare? Secondo la ricerca, ci sono più categorie di tecnologie in grado di rilevare il conducente fatica. Il primo è l'uso di telecamere per monitorare la persona comportamento. Ciò include il monitoraggio dei loro alunni, bocca per sbadigli, posizione della testa e

una varietà di altri fattori. Il la prossima di queste tecnologie è il riconoscimento vocale. spesso a la voce di una persona può dare indizi su quanto sia affaticata. La spiegazione dettagliata delle tecniche sottostanti di rilevamento della sonnolenza che vengono utilizzati principalmente per il rilevamento scopo:

- ECG and EEG
- LBP (Local Binary Pattern)
- Steering Wheel Movement (SWM)
- Optical Detection

ECG and EEG

Molti ricercatori hanno considerato quanto segue segnali fisiologici per rilevare la sonnolenza: elettrocardiogramma (ECG), elettroencefalogramma (EEG). Anche la frequenza cardiaca (FC) varia significativamente tra diversi stadi di sonnolenza, come vigilanza e affaticamento. Pertanto, la frequenza cardiaca, che può essere facilmente determinata dal Il segnale ECG può essere utilizzato anche per rilevare la sonnolenza. Altri hanno misurato la sonnolenza utilizzando la variabilità della frequenza cardiaca (HRV), in cui le frequenze basse (LF) e alte (HF) rientrano nell'intervallo 0,04-0,15 Hz e 0,14-0,4 [19], la Figura 2.3 mostra un sistema di rilevamento del segnale fisiologico che può essere integrato nei veicoli per rilevare la sonnolenza del conducente.

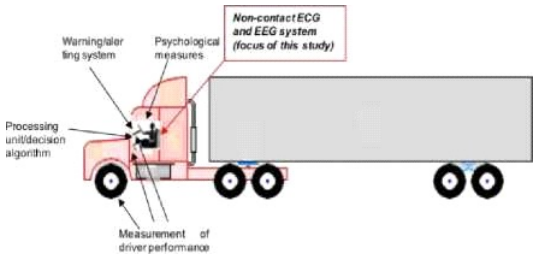


Figura 2.3: Schema dell'integrazione del sistema di rilevamento per la sonnolenza del conducente Rilevazione e assistenza [20]

L'elettroencefalogramma (EEG) è il fisiologico segnale più comunemente usato per misurare la sonnolenza. Il segnale EEG ha varie bande di frequenza, inclusa la banda delta (0,5-4 Hz), che corrisponde all'attività del sonno, la banda theta (4-8 Hz), che è correlata alla sonnolenza, il banda alfa (8-13 Hz), che rappresenta il rilassamento e creatività e la banda beta (13–25 Hz), che corrisponde alla vigilanza. Una diminuzione delle variazioni di potenza nella banda di frequenza alfa e un aumento del theta banda di frequenza indica sonnolenza.
L'elettroencefalogramma (EEG) è il fisiologico segnale più comunemente usato per misurare la sonnolenza. Il segnale EEG ha varie bande di frequenza, inclusa la banda delta (0,5-4 Hz), che corrisponde all'attività del sonno, la banda theta

(4-8 Hz), che è correlata alla sonnolenza, il banda alfa (8-13 Hz), che rappresenta il rilassamento e creatività e la banda beta (13–25 Hz), che corrisponde alla vigilanza. Una diminuzione delle variazioni di potenza nella banda di frequenza alfa e un aumento del theta banda di frequenza indica sonnolenza.

Local binary pattern (LBP)

I modelli binari locali (LBP) hanno suscitato un aumento interesse per l'elaborazione delle immagini e la visione artificiale. Come un metodo non parametrico, LBP riassume le strutture locali di immagini in modo efficiente confrontando ogni pixel con il suo pixel adiacenti. Le proprietà più importanti di LBP sono la sua tolleranza nei confronti dei cambiamenti di illuminazione monotona e la sua semplicità di calcolo. Questa tecnica è per lo più utilizzato per rilevare le emozioni sul viso come, felicità, tristezza, eccitazione ecc. Viene utilizzato LBP (modello binario locale) nel rilevamento della sonnolenza per rilevare il volto del conducente, è divide l'immagine in quattro quadranti poi la parte superiore e parte inferiore vengono rilevati. Figura 2.4, mostra LBP estrarre il l'immagine dal video, quindi l'immagine viene divisa in blocchi, dopo che l'istogramma LBP viene generato da ciascun blocco e si formano gli istogrammi delle caratteristiche [21] La figura mostra il tecnica LBP.

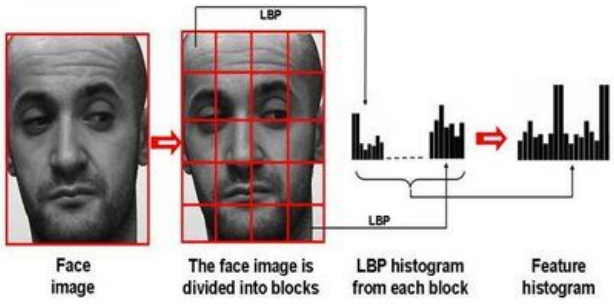


Figura 2.4: Local Binary pattern [21]

Steering Wheel Movement (SWM)

Misurato utilizzando il sensore dell'angolo di sterzata ed è ampiamente misura utilizzata basata sul veicolo per rilevare il livello del conducente sonnolenza. Utilizzo di un sensore angolare montato sullo sterzo piantone, viene misurato il comportamento di sterzata del conducente. quando sonnolenza, il numero di microcorrezioni sullo sterzo ruota si riduce rispetto alla guida normale. Permesso e Graham ha scoperto che i conducenti privati del sonno ne fanno di meno inversioni del volante rispetto ai normali conducenti. Da eliminare l'effetto dei cambi di corsia, i ricercatori hanno considerato solo piccoli movimenti del volante (tra 0,5° e 5°), che sono necessari per regolare la posizione laterale all'interno del la corsia Figura 2.5 mostra il rilevamento basato su SWM. Generalmente, comportamento dello sterzo è influenzato dalle caratteristiche del compito di guida (ad es. velocità,

curvatura e larghezza della corsia), conducente tratti (ad es. esperienza di guida) e stati del conducente (ad es. lassismo, distrazione o affaticamento). I conducenti sono costantemente giudicando la situazione in anticipo e applicando piccole, morbide, regolazioni dello sterzo per correggere piccoli dossi stradali e vento di traverso girando il volante in piccolo incrementi.

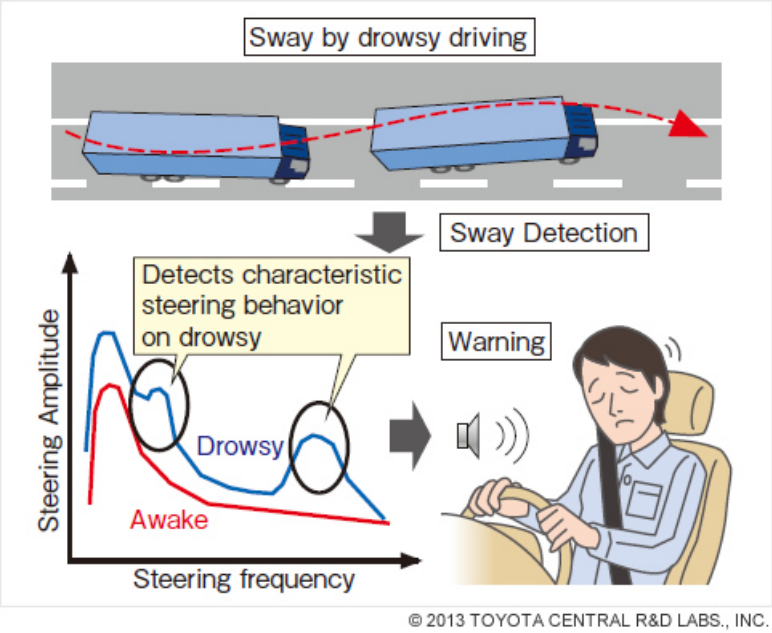


Figura 2.5: Rilevamento basato sul movimento dello sterzo

Quindi, sulla base di piccoli SWM, è possibile determinare lo stato di sonnolenza del conducente e quindi fornire un avviso se necessario. In un ambiente simulato, venti laterali leggeri che spinto l'auto sul lato destro della strada sono stati aggiunti lungo una strada curva per creare variazioni nel posizione laterale e costringere i driver a fare correttivi SWM. Le case automobilistiche, come Nissan e Renault, hanno adottato SWM ma funziona in situazioni molto limitate. Questo è perché possono funzionare in modo affidabile solo in particolari ambienti e sono troppo dipendenti dalla geometria caratteristiche della strada e in misura minore sul caratteristiche cinetiche del veicolo [22].

Optical Detection

L'implementazione più comune di un sensore ottico sistema utilizza LED a infrarossi o nel vicino infrarosso per illuminare gli alunni del conducente, che vengono poi monitorati da una telecamera sistema. Gli algoritmi del computer analizzano la frequenza di lampeggiamento e durata per determinare la sonnolenza. Il sistema della fotocamera potrebbe monitorare anche le caratteristiche del viso e la posizione della testa per segni di sonnolenza, come sbadigli e improvvisi cenni di testa. Raffigura l'uso di un sistema di rilevamento ottico [23].

Eye Blinking Based Technique

In questo battito di ciglia e la durata della chiusura degli occhi è misurato per rilevare la sonnolenza del conducente. Perché quando l'autista si sentiva assonnato in quel momento i suoi occhi ammiccavano e guardavano tra le palpebre sono diverse dalle situazioni normali, quindi loro rilevare facilmente la sonnolenza. La Figura 2.6 mostra l'occhio lampeggiante rilevamento basato sulla sonnolenza In questo sistema la posizione di le iridi e gli stati degli occhi sono monitorati nel tempo per stimare frequenza di ammiccamento e durata della chiusura degli occhi [24] e in questo tipo di sistema utilizza una telecamera posizionata in remoto per vengono poi applicati metodi di acquisizione video e di visione artificiale per localizzare in sequenza le posizioni di viso, occhi e palpebre per misurare il rapporto di chiusura. Usando questi occhi più vicino e razione lampeggiante si può rilevare la sonnolenza del conducente [25]. Come un sistema, montato in un angolo discreto dell'auto, potrebbe monitorare eventuali segni di inclinazione della testa, occhi cadenti o la bocca che sbadiglia simultaneamente. La seguente figura mostra il rilevamento del battito di ciglia.

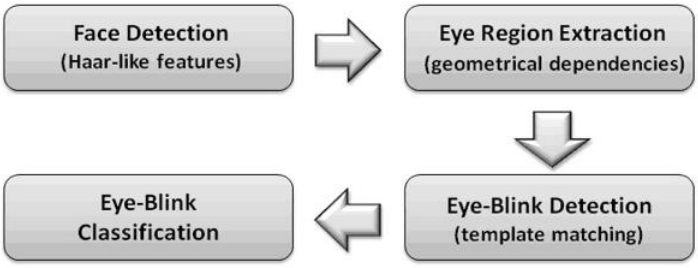


Figura 2.6: Schema dell'algoritmo proposto per il rilevamento del battito di ciglia [26]

Yawning Based Technique

Rilevamento della sonnolenza degli azionamenti in base allo sbadiglio misurazione. Ciò comporta diversi passaggi tra cui il reale rilevamento del tempo e rilevamento del volto del conducente, rilevamento e tracciamento del contorno della bocca e rilevamento, spettacoli di sbadigli basati sulla misurazione sia della velocità che del quantità di cambiamenti nell'area del contorno della bocca. APEX piattaforma per smart camera automobilistica sviluppata da Connive Corp. Nel nostro approccio, il volto del conducente è continuamente catturato utilizzando una videocamera installata sotto il specchio anteriore all'interno dell'auto, rilevare la sonnolenza comporta due passaggi principali per correttamente misurare i cambiamenti nei gesti facciali che implicano sonnolenza. Innanzitutto, il volto del conducente viene rilevato e tracciato nella serie di fotogrammi scattati dalla fotocamera. Dopo aver individuato il faccia del conducente, il passo successivo è rilevare e tracciare la posizione della bocca. Abbiamo scelto di rilevare e tracciare il faccia prima di seguire la bocca poiché questo fa la bocca procedura di tracciamento più robusta

contro i falsi rilevamenti. Dopo il rilevamento della bocca, viene rilevato lo stato di sbadiglio basato sulla misurazione del tasso di variazione nell'area del contorno della bocca e proporzioni dell'area della bocca.

Head Nodding Detection

Un altro metodo attualmente in uso è la posizione della testa Rilevamento. Questa tecnologia determina semplicemente l'inclinazione della testa angolo. Quando l'angolo della testa va oltre un certo angolo, l'allarme acustico viene trasmesso nell'orecchio del conducente.

2.1.3 Anthropometric Face Model

La forma del viso è dinamica, a causa dei molti gradi di libertà articolare della testa umana e delle deformazioni del viso e delle sue parti indotte dall'azione muscolare. Anche la variabilità della forma del viso è fortemente limitata da vincoli genetici e biologici ed è caratterizzata da un alto grado di simmetria (approssimativa) e da invarianti (approssimativi) delle scale e dei rapporti della lunghezza del viso. L'antropometria è una scienza biologica che si occupa delle misurazioni del corpo umano e delle sue diverse parti [27]. Riguarda la tabulazione e la modellazione delle distribuzioni di queste scale e rapporti e può fungere da utile fonte di vincoli di posizione di forma e parti per l'analisi di sequenze di immagini di volti umani. Dopo aver eseguito la misurazione antropometrica su diverse immagini del viso frontale prese da diversi soggetti umani, viene costruito un modello antropometrico del volto umano che può essere utilizzato per individuare le aree più importanti delle caratteristiche facciali dalle immagini del viso [28]. I punti dei volti che sono stati misurati per costruire il modello antropometrico del volto sono rappresentati in Figura 2.7. Alcune statistiche di proporzione sono state ottenute da questi punti e le statistiche della bocca servono come parametro principale per misurare la posizione centrale e le dimensioni delle altre regioni dei tratti del viso. La tabella 2.1 mostra la proporzione delle distanze $\frac{D_i}{D_1}$ prendendo la distanza degli estremi della bocca come parametro principale di misurazione. Viene utilizzata la statistica della bocca, invece delle statistiche inter-occhi, principalmente perché i centri degli occhi (allievi) non possono essere rilevati con gli occhi chiusi.

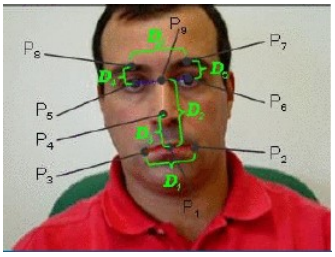


Figura 2.7: Modello antropometrico del viso utilizzato per la localizzazione dell'area dei tratti del viso. *Vengono visualizzati i segni del viso (P_i) e le misurazioni antropometriche (D_i) del nostro modello di viso antropometrico.*

Tabella 2.1: Rapporti di proporzione ottenuti dalle misurazioni antropometriche D_i.

Ratios	*Description*	*Value*
$D_{4,5}/D_1$	Proportion of the eye-brow distance to the mouth width	≈ 0.35
D_2/D_1	Proportion of the nose tip to mouth center distance to the mouth width	≈ 0.65
D_3/D_1	Proportion of the distance between the midpoint of the eyes center and the mouth center to the mouth width	≈ 1.40
D_6/D_1	Proportion of the inter-eyes distance to the mouth width	≈ 1.20

2.1.4 Image Face Detection

Sebbene le persone differiscano per colore e lunghezza dei capelli, è ragionevole presumere che la quantità di pelle che può essere vista e la posizione dei pixel della pelle all'interno della cornice sia un segnale relativamente invariante per il rilevamento del volto di una persona in un'immagine statica. Molti ricercatori hanno sfruttato la relativa unicità del colore della pelle per rilevare e tracciare i volti [29], [30]. Per selezionare automaticamente i pixel della pelle del viso sull'immagine, viene definito un istogramma della pelle del colore prelevando campioni da diversi scenari [31]. Una regione della pelle del viso selezionata manualmente viene eseguita su una sequenza di frame di apprendimento per calcolare un modello di istogramma del colore della pelle (normalizzato) nello spazio RGB. Quindi calcoliamo la probabilità che ogni pixel nelle immagini del viso sia tratto dal modello di istogramma della pelle predefinito. Utilizzando l'istogramma della pelle, e adottando l'approccio proposto in [31], ogni pixel in ciascuna immagine del viso viene prelevato da uno specifico contenitore RGB e quindi viene assegnato il relativo peso che può essere interpretato come una probabilità che il pixel provenga dal modello della pelle . La Figura 2.8 mostra i risultati del rilevamento del colore della pelle su un'immagine del volto umano. Utilizzando la soluzione di Birchfield [29], che combina gradienti di intensità e istogrammi di colore, la proiezione del viso sul piano dell'immagine è modellata come un'ellisse. La Figura 2.8 mostra l'area rilevata della pelle filtrata e l'ellisse adattata a quell'area. Questa ellisse è il punto di partenza per il nostro algoritmo di rilevamento automatico delle caratteristiche del viso. Il centro dell'ellisse e l'asse dell'ellisse verranno utilizzati come nuovo sistema di coordinate della faccia dell'immagine.

Figura 2.8: Il rilevamento del colore della pelle del viso dell'immagine con l'area rilevata del viso e l'ellisse montata.

2.1.5 Rilevamento dei punti caratteristici del viso

Poiché il modello del viso antropometrico proposto è stato ottenuto per le immagini del viso frontale, è necessaria una posa frontale della testa di partenza per garantire un rilevamento robusto dei punti delle caratteristiche facciali. Per rilevare in modo affidabile le caratteristiche del viso su una sequenza di immagini del volto umano prese da diverse pose della testa, un rilevamento delle caratteristiche del filtro di kalman è combinato con il rilevamento delle caratteristiche del modello antropometrico del viso proposto.

A. Identificazione delle aree dei tratti del viso

L'angolo di rotazione (θ) tra il sistema di coordinate del fotogramma e il sistema di coordinate dell'ellisse (vedi Figura 2.8) codifica (in questa fase) la rotazione di rollio del viso, e per adattare il modello antropometrico del viso al viso dell'immagine l'intera immagine è ruotato di questo importo.

Poiché la larghezza della bocca D_1 serve come parametro principale per misurare le posizioni centrali delle altre regioni delle caratteristiche facciali, l'implementazione del nostro rilevamento automatico dei punti delle caratteristiche facciali inizia con la rilevazione dell'area della bocca. Una volta rilevata correttamente la zona della bocca, è possibile rilevare con precisione i punti P_2 e P_3 e utilizzare la distanza D_1 per individuare i punti P_5, P_6, P_7 e P_8 calcolando le distanze D_2, D_4, D_5 e D_6 utilizzando i rapporti di proporzionalità proposti dal viso antropometrico modello. Le aree rettangolari per confinare le regioni delle caratteristiche facciali vengono quindi approssimate utilizzando la larghezza della bocca come criterio di misurazione.

B. Rilevamento delle caratteristiche della bocca

Una volta che il viso è allineato per adattarsi al modello antropometrico, il rilevamento della bocca inizia cercando nell'area inferiore dell'ellisse la regione non segmentata come pelle più vicina all'asse dell'ellisse maggiore. I punti finali di quest'area e il valore di spessore massimo vengono utilizzati per definire con precisione i punti d'angolo delle labbra e la linea tra le labbra.

Prendendo le coordinate dei punti finali (P_l,P_r) dell'area rilevata come gli angoli iniziali delle labbra, la posizione degli angoli delle labbra corretti si ottiene utilizzando il seguente approccio:

1. Converte dal colore alla scala di grigi e applica un allungamento del contrasto per aumentare il contrasto;

2. Siano P_l e P_r le coordinate dei vertici delle labbra iniziali, e sia T_{c_k} lo spessore massimo dell'area rilevata;

3. Per ogni colonna che si estende oltre entrambi gli angoli del labbro, considera una linea verticale $\mathfrak{J}_c$ (di altezza T_{c_k} pixel e centrata sull'angolo del labbro rilevato in precedenza) e trova il pixel più scuro su questa linea verticale [4]. Il pixel più scuro sarà generalmente un pixel nello spazio tra le labbra.

4. Per determinare dove sono gli angoli del labbro il sistema ottiene

$$f(x,y) = \frac{1}{D(x,y)} + \frac{1}{I(x,y)}, \quad \& \quad D(x,y) < 3(1)$$

   dove D(x,y) è la distanza di un pixel (x,y) dall'angolo più vicino della bocca e I(x,y) è l'intensità del livello di grigio a (x,y). Questo darà un pixel che è vicino all'angolo del labbro precedente e che non è troppo luminoso. La funzione massima è il nuovo angolo del labbro.

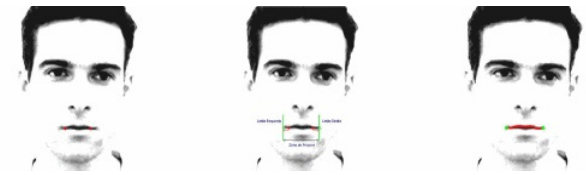


Figura 2.9: Rilevamento degli angoli delle labbra. Da sinistra a destra: immagine a livello di grigio dopo l'allungamento del contrasto con gli angoli iniziali del labbro sovrapposti; L'area di ricerca; Gli angoli delle labbra correttamente posizionati e la linea interlabiale estratta.

5. Questo processo di ricerca viene interrotto quando il gradiente lungo la linea verticale è al di sotto di una certa soglia.

$$|\nabla \mathfrak{J}_c| < 0.1$$

la Figura 2.9 mostra il risultato dell'algoritmo proposto per la posizione dell'angolo del labbro e il rilevamento della linea tra le labbra.

C. Rilevamento delle caratteristiche delle sopracciglia

Il rilevamento dei punti caratteristici del sopracciglio viene effettuato prima del rilevamento della pupilla dell'occhio e dell'angolo dell'occhio per due motivi fondamentali: i) per tenere conto delle situazioni di occhi chiusi e ii) perché il rilevamento corretto degli angoli dell'occhio è difficile da ottenere per grandi angoli di rotazione della testa di imbardata dovuti alle occlusioni.
Poiché le razioni di proporzione antropometrica falliscono per le immagini del viso non frontale, per far fronte a grandi rotazioni di imbardata e beccheggio della testa devono essere adottati vincoli aggiuntivi. La seguente strategia viene utilizzata per rilevare in modo affidabile entrambe le sopracciglia:

1. Utilizzando il modello del viso, stimare la posizione e la dimensione di entrambe le regioni delle caratteristiche del sopracciglio. La dimensione della feature region è correlata alla misura antropometrica D_1, essendo definita come: *Larghezza=125×*D_1*, Altezza=0.8×*D_1.

2. Eseguire il rilevamento del sopracciglio all'interno di ciascuna delle aree caratteristiche utilizzando

   (a) Passa dal colore alla scala di grigi e applica un allungamento del contrasto per aumentare il contrasto;

   (b) Rilevare le caratteristiche orizzontali utilizzando una maschera di gradiente verticale (es. Sobel) e soglia il risultato;

   (c) Eseguire un filtraggio del rumore;

   (d) Seleziona i punti estremi più bassi dell'area segmentata come punti d'angolo del sopracciglio;

3. Per ottenere il corretto rilevamento dei punti finali, in particolare per le grandi rotazioni della testa di imbardata, regolare la dimensione della regione caratteristica (passi del 10%) e ripetere dal passaggio 2^1;

4. Interrompere il processo quando la posizione dei punti finali rimane invariata.

La Figura 2.10 mostra le regioni definite del sopracciglio per due rotazioni dello sguardo della testa di imbardata con il risultato della segmentazione del sopracciglio sovrapposto.

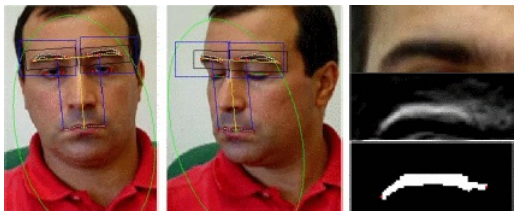


Figura 2.10: Rilevamento sopracciglia. Regioni del sopracciglio definite per diverse rotazioni della testa con il sopracciglio segmentato sovrapposto; colonna di destra: la segmentazione del sopracciglio e il rilevamento dell'angolo del sopracciglio.

D. Rilevamento delle caratteristiche dell'occhio

Il rilevamento delle caratteristiche dell'occhio è il compito più impegnativo a causa della variabilità delle forme. La regione dell'occhio è composta da una palpebra superiore con ciglia, palpebra inferiore, pupilla, sclera brillante e la regione della pelle che circonda l'occhio. Poiché sia la pupilla che la sclera cambiano la loro forma con vari possibili aspetti degli occhi, specialmente quando l'occhio è chiuso o parzialmente chiuso, il rilevamento robusto del centro della pupilla e degli angoli dell'occhio non è un compito facile. La maggior parte degli approcci trovati in letteratura modellano la palpebra e rileva le caratteristiche dell'occhio (centro della pupilla e angoli dell'occhio) principalmente per le immagini del viso frontale [32], [33], [34]. La Figura 2.11 mostra la variabilità delle forme degli occhi in diverse pose della testa.

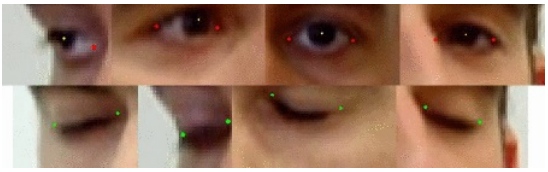


Figura 2.11: Gli occhi presentano variabilità

Il nostro scopo è essere in grado di rilevare con precisione le caratteristiche più importanti dell'occhio (centro della pupilla e angoli degli occhi) per l'ampia serie di aspetti diversi di un occhio che possono verificarsi all'interno del nostro scenario lavorativo. Gli angoli di ciascun sopracciglio vengono utilizzati come punto di partenza per un'accurata selezione della regione dell'occhio. Prendendo il modello del viso antropometrico, la posizione della regione dell'occhio è posizionata immediatamente sopra la regione del sopracciglio. Per gestire le immagini del viso non frontale, la dimensione di ciascuna regione dell'occhio è vincolata dalla dimensione del sopracciglio corrispondente. Rappresentando $D_{eyebrow}$ la distanza tra entrambi gli angoli del sopracciglio, la larghezza (M) e l'altezza (N) di ciascuna regione dell'occhio è definita come:

$$M = D_{eyebrow} + 0.4 \times \max(D_1, D_{eyebrow})$$
$$N = 0.7 \times \max(D_1, D_{eyebrow}).$$

Il rilevamento delle caratteristiche degli occhi viene effettuato in due fasi. Nella prima fase viene verificata la presenza dell'iride su ciascuna regione dell'occhio e se l'occhio è etichettato come occhio aperto, viene applicata una seconda fase di elaborazione per trovare il centro della pupilla e adattare un'ellisse alla forma dell'iride.

1) Occhi aperti contro occhi chiusi. La rilevazione dell'occhio aperto rispetto all'occhio chiuso si ottiene calcolando la funzione di proiezione della varianza [35]

su ciascuna regione delle caratteristiche dell'occhio. La funzione di proiezione della varianza (VPF) è definita come

$$\sigma_H^2(y) = \frac{1}{M} \sum_{i=1.M} [I(x_i, y) - H(y)]^2$$

dove H(y) è il valore dell'intensità media per la riga y e I(x,y) è l'intensità del pixel (x,y). Questa funzione di proiezione della varianza viene applicata in entrambe le direzioni e viene utilizzata per delimitare l'area dell'iride (Figura 2.12). Per aumentare la robustezza del processo di controllo dell'iride, l'iride rilevata viene sottoposta a un controllo incrociato in base a vincoli di dimensione e proporzione.

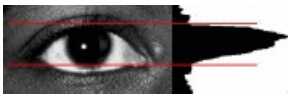


Figura 2.12: La funzione di proiezione della varianza utilizzata per delimitare l'area dell'iride.

2) Rilevamento del centro pupillare e modellazione della forma dell'iride. Indipendentemente dal colore dell'iride, la pupilla sarà sempre la sua regione più scura. L'uso di questo segnale per localizzare il centro della pupilla è vincolato dal livello di dettaglio della regione dell'occhio e dalla quantizzazione del colore o del livello di grigio dell'immagine. Per il nostro scopo, la posizione precisa del centro pupillare non è un grosso problema e il centro pupillare è stato considerato coincidente con il centro dell'iride.

Il seguente approccio è stato utilizzato per rilevare il centro della pupilla e per modellare la forma dell'iride.

1. Passa dal colore alla scala di grigi e applica un allungamento del contrasto per aumentare il contrasto;

2. Sogliare l'area dell'iride delimitata e considerare il centro di massa della regione segmentata come centro dell'iride;

3. Ottenere i punti del contorno dell'iride misurando i valori massimi del gradiente lungo le linee radiali a partire dal centro dell'iride;

4. è un'ellisse per i punti di contorno rilevati.

La Figura 2.13 mostra i risultati ottenuti con l'approccio proposto.

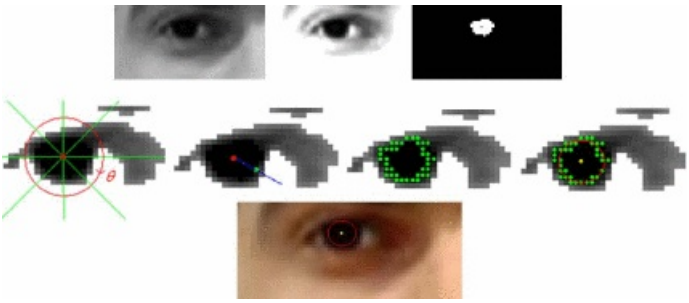


Figura 2.13: Rilevamento del centro pupillare e modello di forma ellittica dell'iride.

3) Rilevamento degli angoli degli occhi. Il rilevamento degli angoli dell'occhio è un compito impegnativo in particolare per le grandi rotazioni della testa di imbardata. In questi casi, la corretta localizzazione degli angoli oculari è estremamente difficile principalmente a causa delle occlusioni. La soluzione che proponiamo cerca di superare questo problema, rilevando gli angoli degli occhi sia in presenza di occhi aperti che di occhi chiusi.

a) Occhi chiusi. Per rilevare gli angoli di un occhio chiuso sfrutteremo la regione scura che si crea con l'unione di entrambe le ciglia. In questi casi viene utilizzato il seguente algoritmo:

1. Passa dal colore alla scala di grigi e applica un allungamento del contrasto per aumentare il contrasto;

2. Applicare una maschera di sfumatura verticale per migliorare i bordi e la soglia orizzontali;

3. Ottenere lo scheletro della regione segmentata seguita da un'operazione di potatura;

4. Seleziona i punti finali dello scheletro come angoli dell'occhio.

La Figura 2.14 presenta l'evoluzione dell'algoritmo e alcuni risultati.

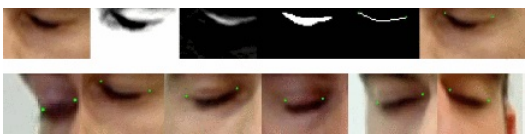


Figura 2.14: Rilevamento degli angoli chiusi. Riga in alto: Evoluzione dell'algoritmo proposto; Riga inferiore: angoli rilevati su diverse immagini Eyes-Shut.

b) Eyes-Open Per Open-eyes, definiamo l'angolo come il punto di transizione più lontano tra la sclera (pixel più luminosi) e la pelle (pixel più scuri). Questo

segnale è facilmente visibile per le immagini del viso frontale, ma manca di visibilità quando gli occhi sono puntati sulle dimensioni e quando l'immagine del viso non è frontale.

Per superare questa mancanza di visibilità è stato aggiunto un ulteriore spunto. La palpebra superiore e le ciglia sono normalmente associate alla regione più scura appena sopra l'iride, e la soluzione proposta sfrutterà questo fatto per estrarre la forma della palpebra superiore. La forma della palpebra viene utilizzata per vincolare l'area di posizione per gli angoli. La seguente strategia viene utilizzata per individuare gli angoli degli occhi:

1. Passa dal colore alla scala di grigi e applica un allungamento del contrasto per aumentare il contrasto;

2. Prendendo l'iride precedentemente segmentata, ricavare le valli VPF in prossimità orizzontale dell'iride; Questi valori codificheranno la variabilità del livello di grigio che esiste tra le regioni scure delle palpebre e l'area chiara della sclera.

3. Ottenere una stima per la posizione degli angoli degli occhi facendo la soglia dei valori VPF;

4. Soglia l'immagine per migliorare le aree dell'iride e della palpebra superiore e rimuovere i pixel che appartengono all'area dell'iride precedentemente segmentata.

5. Ottenere lo scheletro della regione rimanente e adattare allo scheletro una funzione polinomiale;

6. I punti finali della funzione polinomiale scheletro vengono utilizzati per individuare gli angoli dell'occhio.

7. Combina le informazioni fornite da entrambi gli approcci.

La Figura 2.15 mostra i risultati di rilevamento dell'algoritmo proposto.

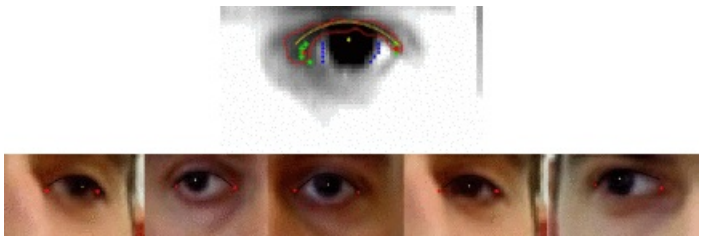


Figura 2.15: Rilevamento angoli occhi aperti. Riga in alto: Evoluzione dell'algoritmo proposto; Riga inferiore: angoli rilevati su diverse immagini Eyes-open.

E. Risultati sperimentali

Le prestazioni delle soluzioni proposte per la localizzazione delle caratteristiche facciali sono state valutate utilizzando cinque sequenze video di un volto umano che guarda in diversi punti 3D e sbatte le palpebre. L'algoritmo è stato testato con diversi soggetti umani e con diverse scale di immagine. La Tabella 2.2 presenta la precisione di rilevamento del rilevatore automatico di punti di caratteristiche facciali descritto sulla carta e la Figura 2.16 mostra la rilevazione di caratteristiche facciali su diverse facce di immagini umane estratte dal database di sequenze di immagini.

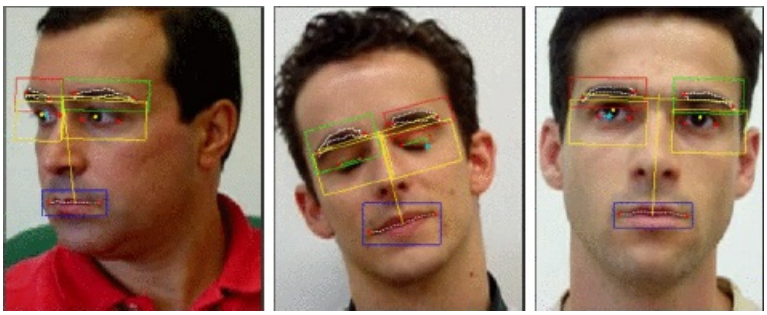


Figura 2.16: Rilevamento delle caratteristiche del viso per soggetti diversi e con diverso orientamento dello sguardo del viso. Le feature region ottenute utilizzando il modello antropometrico del volto e le feature rilevate vengono sovrapposte alle immagini.

2.1.6 Statistiche dell'attenzione visiva del conducente

Tra le misure di rilevamento della sonnolenza, la misura denominata PERCLOS [36] è risultata essere la determinazione più affidabile e valida del livello di vigilanza dei conducenti. PERCLOS è la percentuale di chiusura della palpebra della pupilla nel tempo e riflette le chiusure lente delle palpebre (cadute) piuttosto che le palpebre. Per misurare la chiusura della palpebra della pupilla, la dimensione della pupilla è stata presa come la dimensione media di entrambe le pupille e il tasso di chiusura è definito come $rate_{closure} = 1 - (pupilsize)/max(pupilsize)$, definendo un occhio chiuso se $rate_{closure} > 0.8$.

AECS è la velocità media di chiusura degli occhi [6], che significa la quantità di tempo necessaria per chiudere completamente gli occhi e per aprirli completamente. Una velocità di chiusura dell'occhio individuale è definita come il periodo di tempo durante il quale $0.2 \leq rate_{closure} \leq 0.8$. La Figura 11 mostra PERCLOS e AECS per un periodo di 80 secondi.

Tabella 2.2: Precisione di rilevamento (in percentuale) del rilevatore automatico di punti caratteristici del viso

		Facial Feature Points Detection Rate(%)		
Seq. no.	# frames	% Mouth	% eyebrows left-right	% eyes corners left-right
1	360	99%	100%-99%	96.6%-98.3%
2	360	100%	98.2%-98%	97.7%-96.6%
3	360	100%	100%-96.6%	98.0%-96.6%
4	360	97%	99.4%-98.7%	95.5%-88.3%
5	346	99%	98.2%-100%	96.5%-98.3%

	Eyes-Shut Detection Rate		
Seq. no.	# Eyes-Shut	False Positives left-right	% left-right
1	95	3-1	96.8%-99.0%
2	10	1-0	99.0%-100%
3	90	4-4	95.5%-95.5%
4	47	13-3	72.3%-93.6%
5	105	3-1	97.1%-99.0%

	Eyes-Shut Detection Rate		
Seq. no.	# Eyes-Open	False Negatives left-right	% left-right
1	265	1-1	96.6%-96.6%
2	350	1-3	99.7%-99.1%
3	270	2-1	99.2%-99.6%
4	313	1-7	99.6%-97.7%
5	241	2-5	99.1%-97.9%

2.1.7 Orientamento della testa umana

L'approccio presentato modella la forma del volto umano con un'ellisse, poiché i volti umani possono essere accuratamente modellati con un'ellisse ed è meno sensibile ai cambiamenti dell'espressione facciale. Per recuperare la posa del viso 3D da una singola immagine, si presume che il rapporto tra gli assi maggiore e

minore dell'ellisse del viso 3D sia noto. Questo rapporto è ottenuto attraverso la statistica del volto antropometrico. Il nostro scopo è recuperare i tre angoli di rotazione: imbardata (attorno all'asse verticale), beccheggio (attorno all'asse orizzontale) e rollio (attorno all'asse ottico).

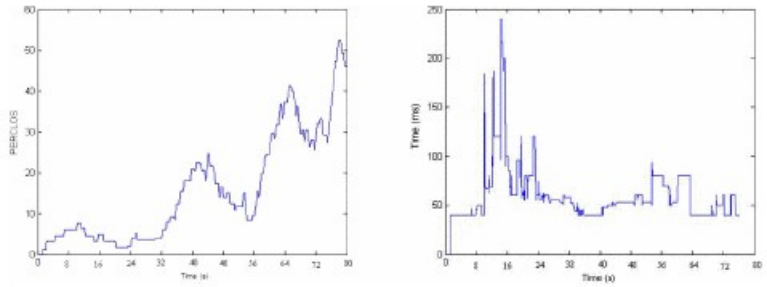


Figura 2.17: Misure PERCLOS (sinistra) e AECS (destra) su un periodo di 80 secondi.

A. Rilevamento e tracciamento dell'ellisse del volto dell'immagine

Al fine di rilevare correttamente l'ellisse del viso, devono essere considerati alcuni vincoli, in dimensioni, posizione e orientamento speciali. La distanza tra le pupille rilevate e la loro posizione viene utilizzata per vincolare la dimensione e la posizione dell'ellisse del viso dell'immagine. L'orientamento della linea che passa attraverso entrambe le pupille è direttamente correlato alla rotazione del viso 3D. Per le pose senza rollio questa linea rimane orizzontale, il che significa che è invariante alle rotazioni di imbardata e beccheggio. Sotto questi vincoli, l'angolo di rollio (γ) è definito da $\gamma = atan[(y_{pl} - y_{pr})/(x_{pl} - x_{pr})]$, dove $P_l = (x_{pl}, y_{pl}) e P_r = (x_{pr}, y_{pr})$ sono la posizione dell'immagine rispettivamente della pupilla rilevata sinistra e destra.

Per l'orientamento frontale, si può assumere una proiezione prospettica debole e la simmetria del viso per la posizione degli occhi all'interno dell'ellisse del viso 3D vale per l'ellisse del viso dell'immagine. Ciò significa che l'asse maggiore dell'ellisse del viso è normale alla linea che collega i due occhi e passa per il centro della linea. Infatti, questi vincoli non valgono per l'orientamento non frontale e l'orientamento della linea maggiore non è normale alla linea di collegamento. Tuttavia, la soluzione adottata ha mantenuto il vincolo che l'asse maggiore dell'ellisse passi per il centro della linea, considerando l'esistenza di un angolo α tra l'asse maggiore e la normale alla linea che collega i due occhi.

Supponendo l'esistenza di una cornice di coordinate ellittiche situata nel punto medio della linea di collegamento degli occhi, con gli assi X e Y allineati rispettivamente con gli assi minore e maggiore dell'ellisse, l'ellisse della faccia dell'immagine è caratterizzata da una 4-tupla $e = (m_i, n_i, d, \alpha)$, dove m_i e n_i sono rispettivamente le lunghezze del semiasse maggiore e minore dell'ellisse, d è la distanza dal centro

dell'ellisse dell'immagine e α è l'angolo di rotazione.

Seguendo l'approccio proposto da Birchfield [29], l'ellisse della faccia dell'immagine può essere rilevata come quella che minimizza la somma normalizzata dell'ampiezza del gradiente proiettata lungo le direzioni ortogonali all'ellisse attorno al perimetro dell'ellisse e all'intersezione dell'istogramma di colore del interno del viso. Questo può essere formulato ha $\varepsilon_g(e) = \frac{1}{N}\sigma_{i=1}^{N}|n(i) \cdot g(i)|$ dove n(i) è il vettore unitario normale all'ellisse al pixel i,g(i) è il gradiente di intensità del pixel e $(\cdot)$ indica il prodotto scalare e $\varepsilon_c(e) = \frac{\sum_{i=1}^{N} min(I_e(i), M_e(i))}{\sum_{j=1}^{N} I_s(i)}$ dove $I_e(i)$ e M(i) sono i numeri di pixel nell'i-esimo bin degli istogrammi, e N è il numero di bin.

La migliore ellisse della faccia è $\chi = argmax_e \in B(\bar{\varepsilon}_g(e) + \bar{\varepsilon}_c(e))$ dove lo spazio di ricerca E è l'insieme delle possibili ellissi prodotte variando i parametri a 4 tuple dell'ellisse e $\bar{\varepsilon}_g$ e $\bar{\varepsilon}_c$ sono valori normalizzati. I parametri a 4 tuple dell'ellisse vengono filtrati tramite un filtro kalman e una tupla stimata *a posteriori* viene utilizzata per definire una stima iniziale per la migliore ricerca dell'ellisse di faccia. La Figura 2.18 mostra diversi risultati dell'adattamento dell'ellisse del viso dell'immagine.

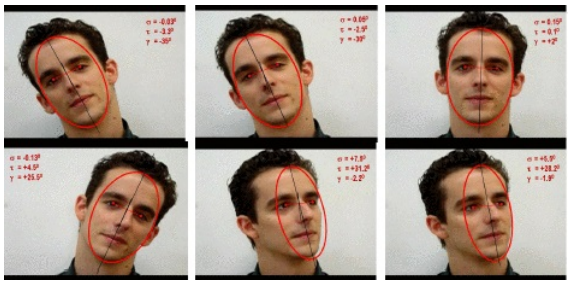


Figura 2.18: Il modello ellittico del viso e la stima 3D dello sguardo del viso. I risultati relativi alla stima dell'orientamento del viso 3D sono sovrapposti alle immagini (τ: yaw; σ: pitch; γ: roll; detti anche: (imbardata;beccheggio;rollio))

B. Orientamento del viso

Si consideri un riquadro di coordinate oggetto attaccato all'ellisse della faccia 3D, con la sua origine situata al centro dell'ellisse e i suoi assi X e Y allineati con gli assi maggiore e minore dell'ellisse. L'asse Z si trova normale al piano dell'ellisse 3D. La cornice delle coordinate della telecamera si trova al centro ottico della telecamera con X_c e Y_c allineati con le direzioni dell'immagine con Z_c lungo l'asse ottico. Poiché l'ellisse della faccia 3D si trova sul piano Z=0, l'equazione di proiezione che caratterizza la relazione tra un punto dell'ellisse della faccia dell'immagine $p_i = (x, y, 1)^T$ e il punto dell'ellisse della faccia 3D corrispondente $P_i = (X, Y, 1)^T$ è dato da $p_i = \beta K[R|t]P_i$ dove K rappresenta la matrice dei parametri intrinseci della camera, $M = [R|t] = [r_1 \quad r_2|t]$ è la matrice dei parametri estrinseci e $\beta = \delta/f$ è uno scalare sconosciuto.

Che rappresentano: $\begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a & c/2 & d/2 \\ c/2 & b & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{bmatrix} y \\ x \\ 1 \end{bmatrix} = 0$

la formula matriciale generica di un'ellisse, l'ellisse della faccia 3D e l'ellisse della faccia dell'immagine possono essere definite, rispettivamente, come

$$[X \quad Y \quad 1]\mathbf{Q}[X \quad Y \quad 1]^T = 0$$
$$\bar{[}x \quad y \quad 1]\mathbf{A}[x \quad y \quad 1]T = 0$$

Sostituendo $p_i = \beta KMP_i$ alla seconda equazione portano a:

$$[X \quad Y \quad 1]\beta M^T K^T AKM[X \quad Y \quad 1]^T = 0.$$

Denotando $B = KTAK$, la matrice ellittica 3D Q produce $Q = \beta M^T BM$. Lascia che la lunghezza dell'asse maggiore e minore dell'ellisse della faccia 3D sia m e n, rispettivamente, e poiché la cornice dell'oggetto si trova al centro dell'ellisse, la matrice dell'ellisse Q è parametrizzata come

$$Q = \begin{bmatrix} 1/m^2 & 0 & 0 \\ 0 & 1/n^2 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

l'equazione risultante

$$Q = \begin{bmatrix} 1/m^2 & 0 & 0 \\ 0 & 1/n^2 & 0 \\ 0 & 0 & -1 \end{bmatrix} = \beta \begin{bmatrix} r_1^T Br_1 & r_1^T Br_2 & r_1^T Bt \\ r_2^T Br_1 & r_2^T Br_2 & r_2^T Bt \\ t^T Br_1 & t^T Br_2 & t^T Bt \end{bmatrix}$$

A causa della simmetria della matrice, esistono solo sei equazioni (vincoli) per un totale di nove incognite.

Poiché l'angolo di rollio è già stato ottenuto, l'orientamento della faccia può essere definito solo dalla rotazione di imbardata e beccheggio. Assumendo un vettore di traslazione nullo, la matrice di rotazione ottenuta dalla rotazione di imbardata e beccheggio è

$$R = R_\sigma R_v = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix} = \begin{bmatrix} cos(\sigma) & sin(\sigma)sin(v) & -sin(\sigma)cos(v) \\ 0 & cos(v) & sin(v) \\ sin(\sigma) & -cos(\sigma)sin(v) & cos(\sigma)cos(v) \end{bmatrix}$$

Supponendo che il rapporto tra l'asse maggiore e minore dell'ellisse 3D della faccia sia conosciuto dall'analisi antropometrica della faccia [27], e lasciando che $c =$

m^2/n^2 rappresenti questo rapporto, la sotto-matrice 2×2 produce

$$\begin{bmatrix} r_1^T B r_1 & r_1^T B r_2 \\ r_2^T B r_1 & r_2^T B r_2 \end{bmatrix} = \begin{bmatrix} 1/m^2 & 0 \\ 0 & 1/n^2 \end{bmatrix}$$

risultando le seguenti equazioni di vincolo

$$r_1^T B r_2 = 0$$
$$\frac{\beta r_1^T B r_1}{1/m^2} = \frac{\beta r_2^T B r_2}{1/2^2} \Leftrightarrow r_1^T B r_1 = \frac{n^2}{m^2} r_2^T B r_2 \Leftrightarrow r_2^T B r_2 - c r_1^T B r_1 = 0$$

Usando queste due equazioni è possibile risolvere il beccheggio e l'imbardata in modo iterativo. Le stime iniziali di 0° per entrambi gli angoli sono state utilizzate per il primo fotogramma della sequenza.

Dati r_1 e r_2 la traslazione T può essere calcolata fino a un fattore di scala utilizzando

$$\beta \begin{bmatrix} r_2^T B T \\ r_1^T B T \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Sia $T = (t_x, t_y, t_z), t_x/t_z$ e t_x/t_z possono essere risolti utilizzando analiticamente l'equazione precedente. Questo approccio è stato testato con diverse immagini reali con buoni risultati. Tuttavia, l'accuratezza ottenuta con questo approccio dipende fortemente dall'ellisse del viso dell'immagine ottenuta. Un rilevamento robusto e accurato della pupilla degli occhi è fondamentale per le prestazioni complessive di questo approccio. La Figura 2.18 mostra i risultati ottenuti con l'approccio di stima dell'orientamento del viso.

C. Monitoraggio del punto di attenzione del conducente

Oltre alla sonnolenza, l'attenzione visiva è un altro aspetto della vigilanza del conducente. Un modo comune per caratterizzare tale livello di disattenzione è la durata e il punto di attenzione del conducente. Il punto di attenzione del conducente si ottiene utilizzando la soluzione di orientamento del viso descritta nella sottosezione precedente, misurando il tasso di disattenzione visiva su un intervallo di tempo fisso (60 s) e il periodo più grande di disattenzione visiva. La soluzione proposta è in grado di misurare in modo robusto la rotazione della testa di imbardata sull'intervallo [-30°.+30°] e la rotazione della testa di beccheggio sull'intervallo [-20°.+20°]. L'intervallo dello sguardo della testa è suddiviso in diverse regioni dello sguardo (R_i) secondo la seguente mappatura

La Figura 2.19 mostra il rilevamento della regione del punto di attenzione utilizzando l'orientamento dello sguardo della testa proposto.

	Angolo di imbardata	Angolo di inclinazione
$R{-}_3$	-30°.-20°	N.D.
$R-2$	-20°.-10°	-20°.-10°
$R-1$	-10°.0°	-10°.0°
$R0$	-5°.+5°	-5°.+5°
R_1	0°.+10°	0°.+10°
R_2	+10°.+20°	+10°.+20°
R_3	+20°.+30°	N.D.

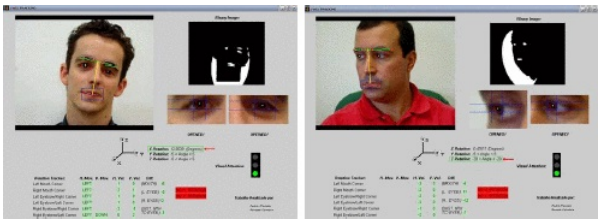


Figura 2.19: L'applicazione di interfaccia grafica utilizzata per il monitoraggio dell'attenzione visiva del conducente. Il livello di sonnolenza e il monitoraggio del punto di attenzione sono visualizzati dal semaforo mostrato sul GI.

2.2 Face Detection

Face Detection, che è il compito di localizzare i volti in un'immagine di input, è una parte fondamentale di qualsiasi sistema di elaborazione dei volti. I volti estratti possono quindi essere utilizzati per l'inizializzazione del rilevamento dei volti o per il riconoscimento automatico dei volti. Un rilevatore facciale ideale dovrebbe possedere le seguenti caratteristiche:

- *Robustezza*: dovrebbe essere in grado di gestire variazioni di aspetto in posa, dimensioni, illuminazione, occlusione, sfondi complessi, espressioni facciali e risoluzione.

- *Rapidità*: dovrebbe essere abbastanza veloce per eseguire l'elaborazione in tempo reale, che è un fattore importante nell'elaborazione di archivi video di grandi dimensioni.

- *Semplicità*: il processo di formazione dovrebbe essere semplice. Ad esempio, il tempo di formazione dovrebbe essere breve, il numero di parametri dovrebbe essere piccolo e i campioni di formazione dovrebbero essere raccolti a basso costo

2.2.1 Real-Time Face Detection mediante classificatori a cascata

Ci sono molti approcci per costruire rivelatori di volti veloci e robusti [37]. Tra questi, quelli che utilizzano metodi di apprendimento avanzati, come reti neurali, macchine vettoriali di supporto e potenziamento, sono i migliori. Come mostrato in Figura 2.20, il rilevamento dei volti in un'immagine richiede in genere i seguenti passaggi:

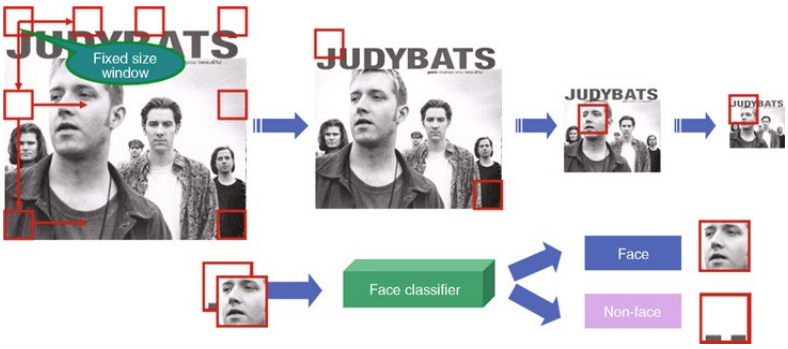


Figura 2.20: Face Detection, Tracking, and Recognition. *Un tipico sistema di rilevamento del volto in cui viene utilizzata una finestra di dimensioni fisse per eseguire la scansione in ogni posizione e scala per estrarre modelli di immagini che vengono poi passati attraverso un classificatore per verificare l'esistenza di un volto.*

1. *Window scanning*: per rilevare i volti in più posizioni e dimensioni, viene utilizzata una finestra di dimensioni fisse (ad es. 24×24 pixel) per estrarre modelli di immagine in ogni posizione e scala. Il numero di motivi estratti da un'immagine di cornice 320×240 è grande, circa 160.000, e solo un piccolo numero di questi motivi contiene un volto.

2. *Estrazione delle feature*: le caratteristiche vengono estratte dal modello di immagine dato. Il tipo di caratteristica più popolare è la Haar wavelet perché è molto veloce da calcolare usando l'immagine integrale [38]. Altri tipi di caratteristiche includono l'intensità dei pixel [39], i modelli binari locali [40] e l'istogramma dell'orientamento dei bordi [41].

3. *Classificazione*: le caratteristiche estratte vengono passate attraverso un classificatore che è stato precedentemente addestrato per classificare il modello di input associato a queste caratteristiche come una faccia o una non faccia.

4. *Merging overlapping detections*:poiché il classificatore è insensibile a piccoli cambiamenti nella traduzione e nella scala, potrebbero esserci più rilevamenti attorno a ciascuna faccia. Per restituire un unico rilevamento finale per volto, è necessario combinare i rilevamenti sovrapposti in un unico rilevamento. A tal fine, l'insieme dei rilevamenti è suddiviso in sottoinsiemi disgiunti in

modo che ciascun sottoinsieme sia costituito dai rilevamenti vicini per una posizione e una scala specifiche. La media degli angoli di tutti i rilevamenti in ogni sottoinsieme è considerata come gli angoli di una regione della faccia restituita da questo insieme.

Poiché la stragrande maggioranza dei modelli elaborati sono non facciali, i sistemi basati su classificatori singoli, come la rete neurale [39] e le macchine vettoriali di supporto [40], sono generalmente lenti. Per superare questo problema, è stata proposta una combinazione di classificatori semplici e complessi che ha portato [38] al primo rilevatore di volti robusto in tempo reale. In questa struttura, i classificatori veloci e semplici vengono utilizzati come filtri nelle prime fasi di rilevamento per rifiutare rapidamente un gran numero di modelli non facciali, mentre classificatori più lenti ma più accurati vengono utilizzati nelle fasi successive per classificare modelli simili a volti. In questo modo, la complessità dei classificatori può essere adattata per corrispondere alla crescente difficoltà dei modelli di input. La Figura 2.21 mostra un esempio di questa struttura.

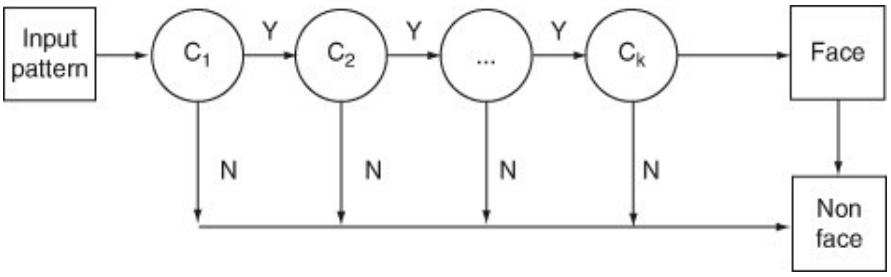


Figura 2.21: Face Detection, Tracking, and Recognition. *Una struttura a cascata per il rilevamento rapido dei volti in cui i modelli facili vengono rifiutati da classificatori semplici nelle fasi precedenti mentre i modelli più difficili vengono elaborati da classificatori più complicati nelle fasi successive.*

2.3 Face Traking

Face Traking è il processo di localizzazione di uno o più volti in movimento in un periodo di tempo utilizzando una telecamera (come illustrato in Figura 2.22). Un determinato volto viene prima inizializzato manualmente o da un rilevatore di volti. Il tracker del viso analizza quindi i successivi fotogrammi video ed emette la posizione del volto inizializzato all'interno di questi fotogrammi stimando i parametri di movimento del volto in movimento. Questo è diverso dal rilevamento dei volti, il cui risultato è la posizione e la scala di un singolo volto in un singolo fotogramma. Il rilevamento dei volti acquisisce informazioni su più volti consecutivi all'interno di fotogrammi video consecutivi. Ancora più importante, questi volti hanno la stessa identità.

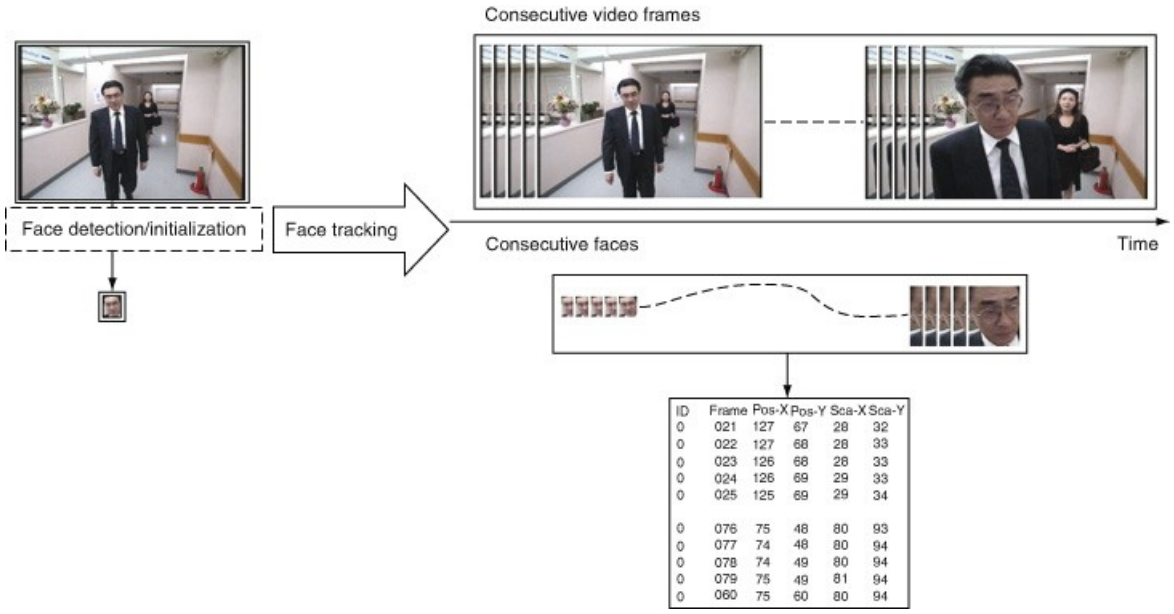


Figura 2.22: Face Detection, Tracking, and Recognition. *Overview of face tracking.*

2.3.1 Vantaggi del Face Tracking

Una delle principali applicazioni del rilevamento dei volti è il recupero della persona dal video trasmesso, ad esempio: "avanzamento rapido intelligente", in cui il video passa alla scena successiva contenente una determinata persona/attore; o il recupero di diversi segmenti TV, interviste, spettacoli, ecc., che presentano una determinata persona in un video o in una vasta raccolta di video. Sivic et al. [42] propone un modo semplice di tracciare i volti per il recupero delle persone da video di lungometraggi. In fase di esecuzione, l'utente delinea un volto in un fotogramma video e le tracce dei volti all'interno del filmato vengono quindi classificate in base alla loro somiglianza con il volto della query delineato allo stesso modo di Google. Poiché una traccia del volto corrisponde a un'identità, a differenza del rilevamento del volto basato su frame, il carico di lavoro della corrispondenza del volto intra-scatto è notevolmente ridotto. Inoltre, il rilevamento del volto fornisce più esempi dell'aspetto dello stesso personaggio per aiutare con la corrispondenza dei volti tra gli scatti.

Il tracciamento dei volti viene utilizzato anche per l'associazione del nome del volto, il cui obiettivo è etichettare le riprese televisive o cinematografiche con l'identità della persona presente in ogni fotogramma del video. Everingham et al. [43] ha proposto un sistema di associazione automatica dei nomi dei volti. Questo sistema utilizza un tracker simile a quello [42] in che può estrarre alcune centinaia di tracce di ogni personaggio particolare in un singolo scatto. Sulla base delle informazioni temporali ottenute dal tracker del volto, vengono utilizzate informazioni testuali per filmati TV e film, inclusi sottotitoli e trascrizioni, per assegnare il nome del personaggio a ciascuna traccia del viso. Ad esempio, gli scatti che

contengono una persona in particolare possono essere recuperati inserendo una parola chiave come "Bush" o "Julia Roberts" invece di inserire un volto di query delineato [42].

Oltre ai video broadcast, il face tracking ha anche importanti applicazioni nella robotica umanoide, nella sorveglianza visiva, nell'interazione uomo-computer (HCI), nelle videoconferenze, nell'autenticazione biometrica della persona basata sul volto, ecc.

2.3.2 Workflow del Face Tracking

Il Face tracking può essere considerato come una sorta di algoritmo che analizza i fotogrammi video e restituisce la posizione dei volti in movimento all'interno di ciascun fotogramma. Per ogni faccia tracciata, sono coinvolti tre passaggi, ovvero inizializzazione, tracciamento e arresto (come illustrato in Figura 2.23).

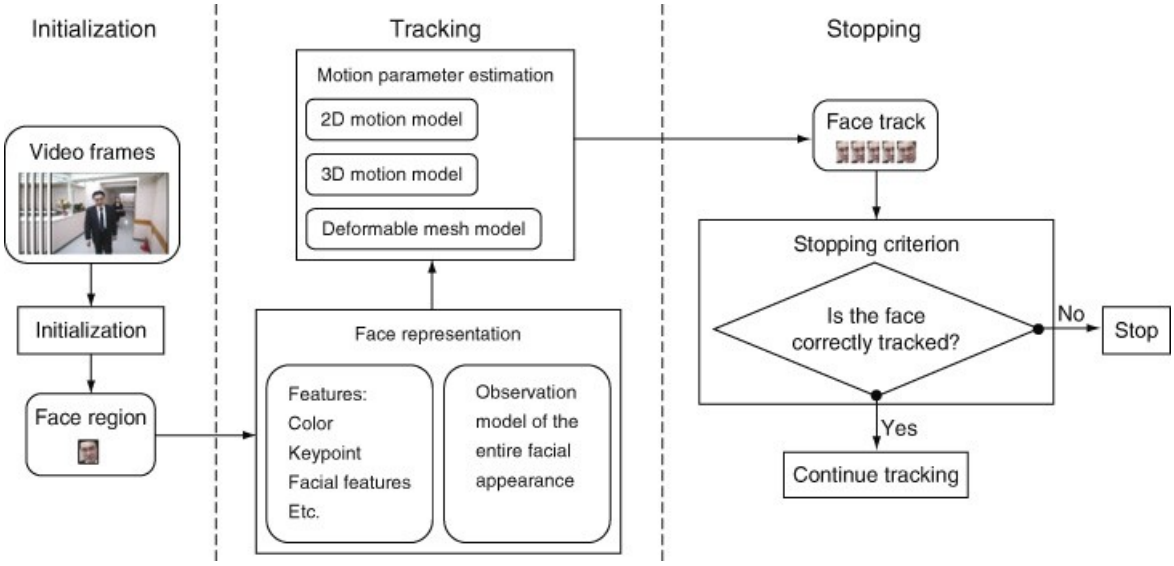


Figura 2.23: Face Detection, Tracking, and Recognition. *Face tracking flowchart*

La maggior parte dei metodi utilizza un rilevatore di volti per l'inizializzazione dei processi di tracciamento. Una difficoltà sempre ignorata in questo passaggio è come controllare i rilevamenti di falsi volti come descritto sopra. Un altro problema è nella gestione di nuovi volti non frontali. Sebbene ci siano stati studi sui rilevatori di volti di profilo o di posa intermedia, tutti soffrono del problema del falso rilevamento molto più di un rilevatore di volti frontali. Per alleviare questi problemi, Chaudhury et al. [44] ha utilizzato due mappe di probabilità del volto invece di una soglia fissa per inizializzare il tracker, una per le viste frontali e una per i profili. Tutti i massimi locali in queste mappe sono scelti come candidati per le facce, le cui probabilità per le facce sono propagate lungo la sequenza temporale.

I candidati le cui probabilità vanno a zero o rimangono basse nel tempo vengono determinati come non-face e vengono eliminati. Le informazioni delle mappe di probabilità delle due facce sono combinate per rappresentare una posa della testa intermedia. I loro esperimenti hanno mostrato che il rilevatore probabilistico proposto era più accurato di un rilevatore facciale tradizionale e poteva gestire i movimenti della testa che coprono una rotazione fuori piano di $\pm 90^{\circ}$ (imbardata).

Dopo l'inizializzazione, si dovrebbero scegliere le caratteristiche da tracciare prima di tracciare un volto. Lo sfruttamento del colore è una delle scelte più comuni perché è invariante alle espressioni facciali, alla scala e ai cambiamenti di posa [45][46]. Tuttavia, i face tracker basati sul colore spesso dipendono da un set di apprendimento dedicato a un determinato tipo di video elaborato e potrebbero non funzionare su video sconosciuti con condizioni di illuminazione variabili o su volti di persone di razze diverse. Inoltre, l'immagine a colori è suscettibile di occlusione da parte di altri oggetti simili a teste. Altre due scelte che sono più robuste a diverse illuminazioni e occlusioni sono il punto chiave [43][44] e le caratteristiche del viso [47][48][49], ad esempio occhi, naso, bocca, ecc. Sebbene la generalità dei punti chiave consenta il tracciamento di diversi tipi di oggetti, senza alcuna conoscenza specifica del viso, il potere di questo metodo di discriminare tra l'obiettivo e il disordine potrebbe non essere sufficiente per gestire il rumore di fondo o altre condizioni avverse. Le funzioni facciali consentono il tracciamento di informazioni facciali di alto livello, ma sono di scarsa utilità quando il video è di bassa qualità. La maggior parte dei face tracker basati su caratteristiche facciali [48][49] sono stati testati utilizzando solo video non trasmessi, ad esempio video webcam, e la loro applicabilità al video trasmesso è discutibile. Notare che i diversi segnali descritti sopra possono essere combinati.

Un tracker basato sull'aspetto o senza caratteristiche abbina un modello di osservazione dell'intero aspetto del viso con l'immagine di input, invece di scegliere solo alcune caratteristiche da tracciare. Un esempio, in [44], è il tracker facciale basato sull'aspetto menzionato sopra. Un altro esempio, in [46], utilizza un rilevatore di volti multi-vista per rilevare e tracciare volti da pose diverse. Oltre al modello di osservazione basato sul viso, è incluso anche un modello di testa per rappresentare la parte posteriore della testa. Questo modello si basa sull'idea che una testa può essere un oggetto di interesse perché il viso non è sempre tracciabile Un filtro viene utilizzato per fondere questi due insiemi di informazioni per gestire le occlusioni dovute a rotazioni della testa fuori dal piano (imbardata) superiore a $\pm 90^{\circ}$.

Durante la procedura di tracciamento, i sistemi di tracciamento del volto di solito utilizzano un modello di movimento che descrive come l'immagine del bersaglio potrebbe cambiare per diversi possibili movimenti del volto. Esempi di modelli di movimento semplice sono i seguenti. Supponendo che il viso sia un oggetto planare, il modello di movimento corrispondente può essere una trasformazione 2D,

ad esempio trasformazione affine o omografia, di un'immagine facciale, ad esempio il fotogramma iniziale [47][48]. Alcune ricerche trattano il viso come un oggetto 3D rigido; il modello di movimento risultante definisce gli aspetti che dipendono dalla posizione 3D e dall'orientamento [49]. Tuttavia, una faccia è in realtà sia 3D che deformabile. Alcuni sistemi cercano di modellare le facce in questo senso, e l'immagine della faccia può essere coperta con una mesh, cioè un modello sofisticato di geometria e texture del viso [50][51]. Il movimento della faccia è definito dalla posizione dei nodi della mesh. Se la qualità del video è alta, un modello di movimento più sofisticato darà risultati più accurati. Ad esempio, un sofisticato modello di geometria e trama potrebbe essere più insensibile ai rilevamenti di falsi volti e alla deriva rispetto a un semplice modello di trasformazione 2D. Tuttavia, la maggior parte dei face tracker basati su 3D e mesh richiedono un aspetto relativamente chiaro, un'alta risoluzione e una variazione di posa limitata, ad esempio rotazioni della testa fuori dal piano (rollio e imbardata) che sono molto inferiori a $\pm 90°$. Questi requisiti non possono essere soddisfatti nel caso di video broadcast. Pertanto, la maggior parte dei face tracker basati su 3D e mesh vengono testati solo su video non trasmessi, ad esempio video webcam [49][50][51].

Infine, la procedura di arresto è discussa raramente. Ciò costituisce una grave carenza per gli algoritmi di tracciamento del volto che generalmente non sono in grado di arrestare un tracciamento del volto in caso di errori di tracciamento, ovvero drifting. Arnaud et al. [47] hanno proposto un approccio che utilizza un tracker di oggetti generico per il tracking del viso e un criterio di arresto basato sull'aggiunta di un eye tracker per alleviare la deriva. Le due posizioni degli occhi tracciati vengono confrontate con la posizione del viso tracciata. Se nessuno degli occhi si trova nella regione del viso, si determina che si sta verificando una deriva e il processo di tracciamento si interrompe. Inoltre, si presume che la maggior parte dei tracker basati su mesh o top-down siano in grado di evitare la deriva.

2.4 Face Recognition

Il riconoscimento facciale (o *Face Recognition*) è il processo di identificazione o verifica di una o più persone che compaiono in una scena utilizzando un database di volti archiviato. Le applicazioni del riconoscimento facciale nei video sono le seguenti.

1. *Face retrieval*: Recupero del volto: recupera gli scatti contenenti le sembianze di una persona utilizzando una o più immagini del volto come query [42][52].

2. *Face matching*: confronta le sequenze di volti di persone sconosciute con le sequenze di volti annotate nel database per l'annotazione o l'identificazione [53].

3. *Face grouping*: organizzare le sequenze di volti rilevati in cluster per l'auto-cast listing [54]

4. *Name-face association*: associate names and faces in video by multi-modal analysis for annotation and retrieval [55][43][56].

Simile al face tracking e al face detection, il face recognition incontra difficoltà nella gestione delle variazioni di risoluzione, dimensione del viso, posa, illuminazione, occlusione ed espressione facciale. Inoltre, è fondamentale gestire le inter-variazioni, ovvero le variazioni tra individui, e le intra-variazioni, ovvero le variazioni che interessano ogni individuo per un robusto sistema di face recognition.

2.4.1 Tecniche di pre-elaborazione

I volti rilevati di solito variano notevolmente e non sono affidabili per la corrispondenza. Pertanto è necessario un passaggio di normalizzazione per eliminare gli effetti di sfondi complessi e diverse illuminazioni, pose e dimensioni. Una tecnica semplice per gestire diverse illuminazioni (Figura 2.24) consiste nel sottrarre il piano di luminosità più adatto ed eseguire l'equalizzazione dell'istogramma. Per gestire i cambiamenti di posa e le diverse dimensioni, le caratteristiche facciali come occhi, naso e bocca vengono rilevate e utilizzate per rettificare tutte le facce in una posa canonica e ridimensionarle alla stessa dimensione. Le maschere ellittiche o altre tecniche di sottrazione dello sfondo possono essere utilizzate per rimuovere il disordine sullo sfondo. Arandjelovic e Zisserman [52] propongono una sofisticata tecnica di normalizzazione del viso che prevede una serie di trasformazioni, ciascuna volta a rimuovere l'effetto di una particolare variazione.

Figura 2.24: Face Detection, Tracking, and Recognition. *Facce prima e dopo il processo di normalizzazione*

2.4.2 Tecniche di Face Recognition

In generale, un'immagine di un singolo volto può essere vista come un punto in un'immagine euclidea (*image space*). La dimensionalità, D, di questo spazio è uguale al numero di pixel dell'immagine della faccia in ingresso. Di solito D è grande, portando alla maledizione del problema della dimensionalità. Tuttavia, le superfici delle facce sono per lo più lisce e hanno una trama regolare, il che rende il loro aspetto piuttosto limitato. Di conseguenza, ci si può aspettare che le

immagini del viso possano essere confinate in uno spazio del viso, una varietà di dimensione inferiore d ¡ D incorporata nello spazio dell'immagine[57].

Il metodo eigen-face è una tecnica popolare per la riduzione della dimensionalità quando viene fornita una serie di immagini di volti di addestramento [58]. Una faccia è rappresentata come un punto in uno spazio auto-faccia ad alta dimensione, che viene calcolato da un insieme di facce di addestramento utilizzando l'analisi delle componenti principali (PCA). la Figura 2.25 mostra un esempio di un insieme di auto-facce calcolate da 3.816 facce [59].

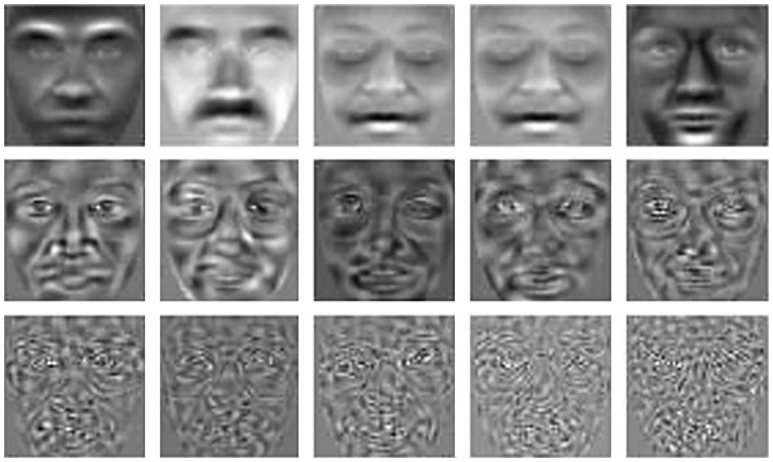


Figura 2.25

Capitolo 3

Strumenti e Metodi

3.1 MediaPipe Face Mesh

MediaPipe Face Mesh è una soluzione per la geometria del viso che stima 468 Landmarks del viso 3D in tempo reale anche su dispositivi mobili. Mediante l'uso del Machine Learning (ML) deduce la geometria della superficie 3D, richiedendo solo un singolo input della telecamera senza la necessità di un sensore di profondità dedicato. Utilizzando architetture modello leggere insieme all'accelerazione GPU in tutta la pipeline, la soluzione offre prestazioni in tempo reale fondamentali per le esperienze live.
Inoltre, la soluzione è in bundle con il modulo Face Geometry che colma il divario tra la stima del landmark del viso e le utili applicazioni di realtà aumentata (AR) in tempo reale. Stabilisce uno spazio 3D metrico e utilizza le posizioni dello schermo del landmark del viso per stimare la geometria del viso all'interno di quello spazio. I dati della geometria della faccia sono costituiti da primitive di geometria 3D comuni, tra cui una matrice di trasformazione della posa della faccia e una face mesh triangolare. Infine viene impiegato un metodo di analisi statistica leggero chiamato Procrustes Analysis per guidare una logica robusta, performante e portatile. L'analisi viene eseguita sulla CPU e ha un ingombro minimo di velocità/memoria oltre all'inferenza del modello ML.

3.1.1 ML Pipeline

La nostra pipeline ML è composta da due modelli di rete neurale profonda in tempo reale che funzionano insieme: un rilevatore che opera sull'immagine completa e calcola le posizioni dei volti e un modello 3D di landmark del volto che opera su tali posizioni e prevede la geometria approssimativa della superficie tramite regressione. Avere la faccia accuratamente ritagliata riduce drasticamente la necessità di comuni aumenti di dati come trasformazioni affini costituite da rotazioni, traslazioni e modifiche di scala. Invece consente alla rete di dedicare la maggior parte della sua capacità all'accuratezza della previsione delle coordinate. Inoltre, nella nostra pipeline i ritagli possono essere generati anche sulla base dei landmark del

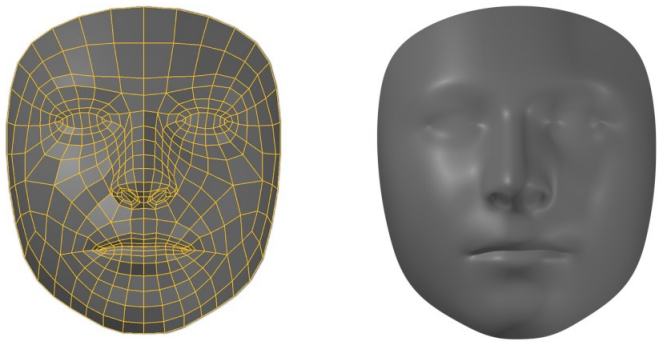
volto identificati nel frame precedente, e solo quando il modello del landmark non è più in grado di identificare la presenza del volto viene invocato il face detector per ri-localizzare il volto. Questa strategia è simile a quella impiegata nella nostra soluzione *MediaPipe Hands*, che utilizza un rilevatore di palmi insieme a un modello di riferimento della mano.
La pipeline è implementata come un grafo MediaPipe che usa un *face landmark subgraph* dal *face landmark module* ed esegue il rendering usando un sottografo del *face renderer subgraph*. Il *face landmark subgraph* utilizza internamente un *face-detection-subgraph* del volto dal *face detection module*.

Modelli

- **FACE DETECTION MODEL** E' stato studiato dalla *MediaPipe Face Detection*, la quale è una soluzione di face detection ultraveloce che viene fornita con 6 landmark e un supporto multi-faccia. Si basa su *BlazeFace*, un rilevatore di volti leggero e dalle buone prestazioni su misura per l'inferenza della GPU mobile. Le prestazioni super-in tempo reale del rilevatore consentono di applicarlo a qualsiasi esperienza di mirino dal vivo che richiede un'accurata regione facciale di interesse come input per altri modelli specifici per attività, come 3D facial keypoints o la stima della geometria (ad esempio, *MediaPipe Face Mesh*), caratteristiche del viso o classificazione delle espressioni e segmentazione della regione del viso. BlazeFace utilizza una rete di estrazione delle funzionalità leggera ispirata, ma distinta da *MobileNetV1/V2*, uno schema di ancoraggio compatibile con GPU modificato da *Single Shot MultiBox Detector (SSD)* e una strategia di risoluzione dei legami migliorata alternativa alla soppressione non massima.

- **FACE LANDMARK MODEL** Per i 3D face landmark abbiamo impiegato l'apprendimento del trasferimento e addestrato una rete con diversi obiettivi: la rete prevede simultaneamente le coordinate del punto di riferimento 3D su dati sintetici renderizzati e contorni semantici 2D su dati annotati del mondo reale. La rete risultante ci ha fornito ragionevoli previsioni 3D sui landmarks non solo su dati sintetici ma anche su dati reali.
  La rete di landmark in 3D riceve come input un fotogramma video ritagliato senza ulteriore input di profondità. Il modello restituisce le posizioni dei punti 3D, nonché la probabilità che una faccia sia presente e ragionevolmente allineata nell'input. Un approccio alternativo comune consiste nel prevedere una mappa termica 2D per ogni punto di riferimento, ma non è suscettibile di previsione della profondità e ha costi di calcolo elevati per così tanti punti. Miglioriamo ulteriormente l'accuratezza e la robustezza del nostro modello avviando iterativamente e perfezionando le previsioni. In questo modo possiamo far crescere il nostro set di dati in casi sempre più difficili, come smorfie, angoli obliqui e occlusioni.
  Di seguito c'è la spiegazione più approfondita del face landmark model:

– **Introduzione.** Il problema di prevedere la geometria del viso allineando un template di face mesh, chiamato anche *face alignment* o *face registration*, è stata per lungo tempo una pietra angolare di visione computerizzata. È comunemente posto in termini di localizzazione relativamente pochi (tipicamente 68) landmark o keypoints. Queste punti hanno una semantica distinta o partecipano a contorni facciali significativi.[60]
Un approccio alternativo è quello di stimare la posa, la scala, e i parametri di un modello 3D morphable (3DMM) [61]. Un 3DMM, come BFM2017, l'edizione 2017 del Basel Face Model [62], è solitamente ottenuto attraverso l'analisi delle componenti principali. La mesh risultante presenta in genere molti più punti (circa 50K nel caso di BFM), ma il range delle possibili previsioni è limitato dalla varietà lineare attraversato dalla base PCA, che a sua volta è determinata da la diversità del set di volti catturati per il modello. Come un esempio concreto, il BFM è apparentemente incapace di rappresentare in modo affidabile un volto con esattamente un occhio chiuso.
Abbiamo posto un problema di stima delle posizioni del 3D mesh vertici con una rete neurale, trattando ogni vertice come un landmark indipendente. La topologia mesh è composta da 468 punti disposti in quad fissi (Figura 3.1a). I punti sono stati selezionati manualmente in conformità



(a) (b)

Figura 3.1: La topologia mesh prevista (a) e la Suddivisione a 3 livelli di Catmull-Clark (b)

con le presunte applicazioni, come effetti AR espressivi, accessori virtuali e prova di abbigliamento e trucco. Le aree che dovrebbero avere maggiore variabilità e maggiore importanza nella percezione umana sono stati assegnati con densità di punti maggiore. Permette di costruire un plausibile liscio rappresentazione della superficie con l'applicazione della suddivisione di Catmull-Clark [63] (Figura 3.1b).
L'input al modello è un frame (o, più in generale, a flusso di frame) di una singola telecamera RGB: non sono necessarie informazioni sul

sensore di profondità. Un esempio dell'output del modello è presentato nella Figura 3.2. La nostra configurazione mira in tempo reale inferenza GPU mobile, ma abbiamo anche progettato più leggero versioni del modello per affrontare l'inferenza della CPU sui dispositivi mobili privi di un adeguato supporto GPU. chiameremo il modello di targeting per GPU è un modello "completo", contrastandolo con il modello "più leggero" su misura per la CPU negli esperimenti.

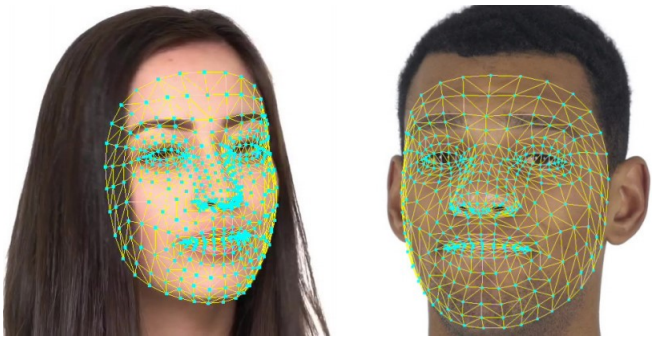


Figura 3.2: Esempio di Face Mesh Prediction

– **Image processing pipeline.** Si organizza l'elaborazione di un'immagine come segue:

  1. L'intero frame dall'input della fotocamera viene elaborato da un face detector molto leggero [64] che produce volto rettangoli di delimitazione e diversi landmark (ad esempio centri oculari, regioni dell'orecchio e punta del naso). I landmark sono utilizzati per ruotare un rettangolo facciale per allineare la linea che collega i centri oculari con l'asse orizzontale del rettangolo.
  2. Un rettangolo ottenuto nel passaggio precedente viene ritagliato da l'immagine originale e ridimensionata in modo da formare l'input alla rete neurale di previsione della mesh (di dimensioni variabili da 256×256 pixel nel modello completo a 128×128 pollici il più piccolo). Questo modello produce un vettore di coordinate 3D del landmark, che successivamente vengono mappate di nuovo nel sistema di coordinate dell'immagine originale. Un output di rete scalare distinto (face flag) produce la probabilità dell'evento che una faccia ragionevolmente allineata sia effettivamente presente nel ritaglio fornito.

  Abbiamo adottato la politica che le coordinate x e y dei vertici corrispondono alle posizioni dei punti nella 2D piano come dato dalle coordinate dei pixel dell'immagine. Le coordinate z vengono interpretate come la profondità relativa a un piano di riferimento passante per il centro di massa della mesh. Vengono ridimensionati in modo da mantenere un rapporto di aspetto fisso tra l'intervallo delle coordinate x e

l'intervallo di coordinate z, cioè una faccia che è ridimensionata a metà della sua dimensione ha il suo intervallo di profondità (dal più vicino al più lontano) ridotto dello stesso moltiplicatore.
Se utilizzato sull'ingresso video in modalità di face tracking, a un buon ritaglio del viso è disponibile dalla precedente previsione del fotogramma e l'uso del face detector è ridondante. In questo scenario, viene utilizzato solo nel primo frame e nel rari eventi di ri-acquisizione (dopo la probabilità prevista dalla bandiera faccia scende al di sotto della soglia appropriata).
Va notato che con questa configurazione, la seconda rete riceve gli ingressi con facce ragionevolmente centrate e allineato. Riteniamo che ciò consenta di risparmiare una parte della capacità di rappresentazione del modello che potrebbe altrimenti essere spesa per gestire i casi con rotazione e traslazione sostanziali. In particolare, potremmo ridurre la quantità di potenziamenti correlati ottenendo al tempo stesso la qualità della previsione.

– **Dataset, annotation, and training.** Nel nostro training, ci affidiamo a un dataset di provenienza globale di circa 30K di foto di fotocamere mobili in natura scattate da un'ampia varietà di sensori in condizioni di illuminazione variabili. Durante il training, aumentiamo ulteriormente il dataset con primitive standard di ritaglio ed elaborazione delle immagini e anche a pochi specializzati: modellazione del rumore del sensore della fotocamera [65] e applicando una trasformazione parametrica non lineare randomizzata all'istogramma dell'intensità dell'immagine (quest'ultimo aiuta simulando condizioni di luce marginale).
Ottenere la verità al suolo per i 468 3D mesh-points è un compito laborioso e altamente ambiguo. Invece di annotando manualmente i punti uno per uno, utilizziamo il seguendo una procedura iterativa.

1. Addestrare un modello iniziale utilizzando le seguenti due fonti di supervisione:
   * Rendering sintetici di un 3DMM sui rettangoli facciali delle foto del mondo reale (al contrario di, ad esempio, solidi sfondi, per evitare un eccessivo adattamento su di essi). Il le coordinate del vertice di verità al suolo sono quindi immediatamente disponibile da una corrispondenza predefinita tra i 468 mesh points e un sottoinsieme di vertici 3DMM.
   * Landmark 2D corrispondenti a un piccolo sottoinsieme dei vertici della mesh che partecipano a un insieme di contorni semantici (vedi Figura 3.3) annotati sul dataset "in-the-wild" effettivo. I landmark sono previsti come output separato alla fine di un ramo di rete dedicato, introdotto con l'intento di condividere le rappresentazioni delle facce intermedie tra i percorsi 2D e 3D.

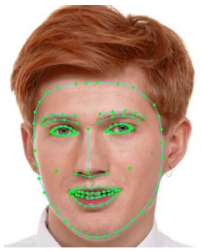


Figura 3.3: I contorni semantici 2D utilizzati durante l'iniziale processo di bootstrap

Dopo il training di questo primo modello, fino al 30 percento delle immagini nel nostro dataset avevano previsioni adatte per il perfezionamento nella fase successiva.

2. Perfeziona iterativamente le coordinate x e y bootstrap applicando alle immagini il modello più aggiornato, filtrando quelle adatte a tale affinamento (ovvero dove l'errore di previsione è tollerabile). Il perfezionamento rapido delle annotazioni è reso possibile da uno strumento "brush" con regolazione raggio che consente di spostare un'intera gamma di punti a una volta. La quantità di movimento diminuisce esponenzialmente con la distanza lungo i bordi della mesh da il vertice del pivot sotto il cursore del mouse. Questo permette annotatori per regolare spostamenti sostanziali dell'area con grandi "strockes" (o tratti) prima dei perfezionamenti locali, preservando l'uniformità della superficie della mesh. Notiamo che le coordinate z vengono lasciate intatte; l'unica fonte di supervisione per loro è il rendering sintetico 3D delineato sopra. Nonostante le previsioni di profondità non siano quindi metricamente accurate, nella nostra esperienza le maglie risultanti sono visivamente sufficientemente plausibili per es. guida 3D realistici rendering di texture sul viso o allineare oggetti 3D resi come parte dell'esperienza di prova degli accessori virtuali.

– **Model architecture.** Per il modello di previsione della mesh, utilizziamo un'architettura di rete neurale residua personalizzata ma abbastanza semplice. Usiamo un sotto-campionamento più aggressivo nei primi strati della rete e dedicare la maggior parte del calcolo alla sua parte superficiale.
Pertanto, i campi recettivi dei neuroni iniziano a coprire ampie aree dell'immagine in ingresso relativamente presto. Quando un tale campo ricettivo raggiunge il confine dell'immagine, la sua posizione relativa nell'immagine di input diventa implicitamente disponibile per il modello su cui fare affidamento (a causa dell'imbottitura di convoluzione). Di conseguenza, è probabile che i neuroni per gli strati più profondi distinguano tra ad es. caratteristiche rilevanti per la bocca e per gli occhi.

Il modello è in grado di completare un volto leggermente occluso o che attraversa il confine dell'immagine. Questo ci porta ad a conclusione che una rappresentazione mesh di alto livello e bassa dimensionalità è costruita dal modello che viene trasformato in coordinate solo negli ultimi strati della rete.

3.1.2 Face Geometry Module

Il Face Landmark Module esegue un rilevamento del landmark facciale a telecamera singola nello spazio delle coordinate dello schermo: le coordinate X e Y sono coordinate dello schermo normalizzate, mentre la coordinata Z è relativa e viene ridimensionata come la coordinata X sotto il modello della telecamera di proiezione prospettica debole . Questo formato è adatto per alcune applicazioni, tuttavia non abilita direttamente l'intero spettro delle funzionalità di realtà aumentata (AR) come l'allineamento di un oggetto 3D virtuale con un volto rilevato.
Il Face Geometry Module [66] si allontana dallo spazio delle coordinate dello schermo verso uno spazio 3D metrico e fornisce le primitive necessarie per gestire un volto rilevato come un normale oggetto 3D. In base alla progettazione, sarai in grado di utilizzare una telecamera prospettica per proiettare la scena 3D finale nello spazio delle coordinate dello schermo con la garanzia che le posizioni dei landmark del viso non vengano modificate.

Concetti chiave

- **METRIC 3D SPACE**, stabilito all'interno del Face Geometry Module è uno spazio di coordinate 3D metrico ortonormale destrorso. All'interno dello spazio, c'è una telecamera prospettica virtuale situata all'origine dello spazio e puntata nella direzione negativa dell'asse Z. Nella pipeline attuale, si presume che i frame della telecamera di input siano osservati esattamente da questa telecamera virtuale e quindi i suoi parametri vengano successivamente utilizzati per convertire le coordinate del punto di riferimento dello schermo nello spazio metrico 3D. I parametri della telecamera virtuale possono essere impostati liberamente, tuttavia per risultati migliori si consiglia di impostarli il più vicino possibile ai parametri fisici della telecamera reale.

- **CANONICAL FACE MODEL** è un modello 3D statico di un volto umano, che segue la topologia del 468 landmark del volto in 3D del *Face Landmark Model*. Il modello ha due importanti funzioni:

  - Definisce le unità metriche: la scala del modello canonico della faccia definisce le unità metriche dello spazio Metrico 3D. Un'unità metrica utilizzata dal modello di faccia canonica predefinito [67] è un centimetro;
  - Collega spazi statici e di runtime: la matrice di trasformazione della posa del viso è - in effetti - una mappa lineare dal modello canonico del viso nel set di landmark del viso runtime stimato su ciascun frame. In

questo modo, le risorse 3D virtuali modellate attorno al modello di faccia canonica possono essere allineate con una faccia tracciata applicando loro la matrice di trasformazione della posa della faccia.

Componenti

- La **Geometry Pipeline** è un componente chiave, responsabile della stima degli oggetti della geometria della faccia all'interno dello spazio metrico 3D. Su ogni frame, vengono eseguiti i seguenti passaggi nell'ordine indicato:

  - Le coordinate dello schermo del face landmark vengono convertite nelle coordinate dello spazio metrico 3D;
  - La matrice di trasformazione della posa del viso è stimata come una mappatura lineare rigida dal landmark della metrica del volto canonico impostato nel landamrk della metrica del volto di runtime impostato in modo da ridurre al minimo la differenza tra i due;
  - Una face mesh viene creata utilizzando i landmark metrici della faccia di runtime come posizioni dei vertici (XYZ), mentre sia le coordinate della trama dei vertici (UV) che la topologia triangolare sono ereditate dal modello canonico della faccia.

  La Geometry pipepile è implementata come un *calcolatore* [68] MediaPipe. Per tua comodità, il calcolatore della Geometry pipeline del viso è raggruppato insieme ai metadati corrispondenti in un sottografo [69] MediaPipe unificato. Il formato della geometria della faccia è definito come un messaggio [70] Buffer di protocollo.

- L'**Effect Renderer** è un componente che funge da esempio funzionante di un renderer di effetti facciali. Si rivolge all'API OpenGL ES 2.0 per consentire prestazioni in tempo reale su dispositivi mobili e supporta le seguenti modalità di rendering:

  - 3D object rendering mode: un oggetto virtuale viene allineato con un viso rilevato per emulare un oggetto attaccato al viso (esempio: occhiali);
  - Face mesh rendering mode: una texture viene allungata sulla superficie della mesh del viso per emulare una tecnica di pittura del viso.

  In entrambe le modalità di rendering, la face mesh viene prima renderizzata come un occlusore direttamente nel buffer di profondità. Questo passaggio aiuta a creare un effetto più credibile nascondendo elementi invisibili dietro la superficie del viso.

3.1.3 Solution APIs

Opzioni di Configurazione

Lo stile di denominazione e la disponibilità possono differire leggermente tra piattaforme/linguaggi.

- **STATIC_IMAGE_MODE**
  Se impostato su `false`, la soluzione considera le immagini di input come un flusso video. Cercherà di rilevare i volti nelle prime immagini di input e, in caso di rilevamento riuscito, localizzerà ulteriormente i punti di riferimento dei volti. Nelle immagini successive, una volta rilevati tutti i volti `"max_num_faces"` e localizzati i punti di riferimento corrispondenti, traccia semplicemente quei punti di riferimento senza invocare un altro rilevamento finché non perde traccia di nessuno dei volti. Ciò riduce la latenza ed è ideale per l'elaborazione di fotogrammi video. Se impostato su `true`, il rilevamento dei volti viene eseguito su ogni immagine di input, ideale per elaborare un batch di immagini statiche, possibilmente non correlate. Predefinito su `false`.

- **MAX_NUM_FACES**
  Numero massimo di volti da rilevare. Ha valore `1` di default

- **MIN_DETECTION_CONFIDENCE**
  Valore minimo di confidenza (`[0.0, 1.0]`) dal modello di rilevamento del volto affinché il rilevamento sia considerato riuscito. Il valore di default è `0.5`.

- **MIN_TRACKING_CONFIDENCE**
  Valore minimo di confidenza (`[0.0, 1.0]`) dal modello di tracciamento dei landmark affinché i landmark del volto vengano considerati tracciati con successo, altrimenti il face detection verrà invocato automaticamente nell'immagine di input successiva. Impostarlo su un valore più alto può aumentare la robustezza della soluzione, a scapito di una latenza più elevata. Ignorato se `"static_image_mode"` è `true`, dove il rilevamento dei volti viene eseguito semplicemente su ogni immagine. Il valore di default è `0.5`.

3.1.4 Output

Lo stile di denominazione e la disponibilità possono differire leggermente tra piattaforme/linguaggi.

- **MULTI_FACE_LANDMARKS**
  Raccolta di volti rilevati/tracciati, in cui ogni volto è rappresentato come un elenco di 468 punti di riferimento del volto e ciascun punto di riferimento è composto da `x`, `y` e `z`. `x` e `y` sono normalizzati a `[0.0, 1.0]` rispettivamente dalla larghezza e dall'altezza dell'immagine. `z` rappresenta la profondità del

punto di riferimento con la profondità al centro della testa come origine, e minore è il valore più il punto di riferimento è vicino alla fotocamera. La grandezza di `z` usa all'incirca la stessa scala di `x`.

- **Python Solution API**
  Principalmente andremo ad usare **Python** per il nostro progetto, e il codice iniziale da cui si partirà è il seguente, il quale successivamente verrà implementato e migliorato. Per fare ciò andremo ad installare il *package di Mediapipe in Python* e andando a capire i concetti usando anche **Google Colab** [71].
  Di seguito c'è il codice che verrà implementato nel progetto:

```
import cv2
import mediapipe as mp
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_face_mesh = mp.solutions.face_mesh

# For static images:
IMAGE_FILES = []
drawing_spec = mp_drawing.DrawingSpec(thickness=1, circle_radius=1)
with mp_face_mesh.FaceMesh(
    static_image_mode=True,
    max_num_faces=1,
    min_detection_confidence=0.5) as face_mesh:
  for idx, file in enumerate(IMAGE_FILES):
    image = cv2.imread(file)
    # Convert the BGR image to RGB before processing.
    results = face_mesh.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

    # Print and draw face mesh landmarks on the image.
    if not results.multi_face_landmarks:
      continue
    annotated_image = image.copy()
    for face_landmarks in results.multi_face_landmarks:
      print('face_landmarks:', face_landmarks)
      mp_drawing.draw_landmarks(
          image=annotated_image,
          landmark_list=face_landmarks,
          connections=mp_face_mesh.FACEMESH_TESSELATION,
          landmark_drawing_spec=None,
          connection_drawing_spec=mp_drawing_styles
          .get_default_face_mesh_tesselation_style())
      mp_drawing.draw_landmarks(
          image=annotated_image,
          landmark_list=face_landmarks,
          connections=mp_face_mesh.FACEMESH_CONTOURS,
          landmark_drawing_spec=None,
          connection_drawing_spec=mp_drawing_styles
```

```
38                .get_default_face_mesh_contours_style())
39          cv2.imwrite('/tmp/annotated_image' + str(idx) + '.png
    ', annotated_image)
40
41      # For webcam input:
42      drawing_spec = mp_drawing.DrawingSpec(thickness=1,
    circle_radius=1)
43      cap = cv2.VideoCapture(0)
44      with mp_face_mesh.FaceMesh(
45          min_detection_confidence=0.5,
46          min_tracking_confidence=0.5) as face_mesh:
47        while cap.isOpened():
48          success, image = cap.read()
49          if not success:
50            print("Ignoring empty camera frame.")
51            # If loading a video, use 'break' instead of '
    continue'.
52            continue
53
54          # Flip the image horizontally for a later selfie-view
     display, and convert
55          # the BGR image to RGB.
56          image = cv2.cvtColor(cv2.flip(image, 1), cv2.
    COLOR_BGR2RGB)
57          # To improve performance, optionally mark the image
    as not writeable to
58          # pass by reference.
59          image.flags.writeable = False
60          results = face_mesh.process(image)
61
62          # Draw the face mesh annotations on the image.
63          image.flags.writeable = True
64          image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)
65          if results.multi_face_landmarks:
66            for face_landmarks in results.multi_face_landmarks:
67              mp_drawing.draw_landmarks(
68                  image=image,
69                  landmark_list=face_landmarks,
70                  connections=mp_face_mesh.FACEMESH_TESSELATION
    ,
71                  landmark_drawing_spec=None,
72                  connection_drawing_spec=mp_drawing_styles
73                  .get_default_face_mesh_tesselation_style())
74              mp_drawing.draw_landmarks(
75                  image=image,
76                  landmark_list=face_landmarks,
77                  connections=mp_face_mesh.FACEMESH_CONTOURS,
78                  landmark_drawing_spec=None,
79                  connection_drawing_spec=mp_drawing_styles
80                  .get_default_face_mesh_contours_style())
81          cv2.imshow('MediaPipe FaceMesh', image)
82          if cv2.waitKey(5) & 0xFF == 27:
83            break
```

```
84      cap.release()
85
```

Listing 3.1: Codice Iniziale

3.2 Event Camera

Una **Event Camera** [72] nota anche come una neuromorphic camera, silicon retina o dynamic vision sensor, è un sensore di immagine che risponde ai cambiamenti locali di luminosità. Le camere ad eventi non acquisiscono immagini utilizzando un Shutter[1] come fanno le fotocamere convenzionali. Invece, ogni pixel all'interno di una camera ad eventi funziona in modo indipendente e asincrono, segnalando i cambiamenti di luminosità non appena si verificano o rimanendo in silenzio altrimenti. Le moderne camere ad eventi hanno una risoluzione temporale di microsecondi, una gamma dinamica di 120 dB e meno sotto/sovraesposizione e sfocatura del movimento rispetto alle frame camera.

3.2.1 Introduzione

Le Event camera contengono pixel che rispondono in modo indipendente ai cambiamenti di luminosità nel momento in cui si verificano. Ciascun pixel memorizza un livello di luminosità di riferimento e lo confronta continuamente con il livello di luminosità corrente. Se la differenza di luminosità supera una soglia preimpostata, quel pixel azzera il suo livello di riferimento e genera un evento: un pacchetto discreto di informazioni contenente l'indirizzo del pixel e il timestamp. Gli eventi possono anche contenere la polarità (aumento o diminuzione) di un cambiamento di luminosità, o una misurazione istantanea del livello di illuminazione attuale. Pertanto, le telecamere di eventi emettono un flusso asincrono di eventi innescati da cambiamenti nell'illuminazione della scena.

Le Event Camera inviano informazioni sui pixel indipendenti non appena la loro variazione di intensità supera una soglia superiore o inferiore,generando rispettivamente eventi "ON" o "OFF" (vedi Figura 3.4). A differenza delle fotocamere convenzionali, in cui le immagini sono complete dato a un frame rate fisso–, in caso di telecamere, i messaggi di variazione di intensità arrivano in modo asincrono per pixel, questo accade alla risoluzione del microsecondo. Inoltre, le telecamere per eventi presentano un'elevata gamma dinamica di luminosità (ad es. 120dB per il Modello Davis 240C [73] utilizzato in questo lavoro). Questi due asset li rendono adatti per applicazioni ad alta velocità e/o con condizioni di illuminazione difficili (livelli di illuminazione bassi o sovraesposizione). Esempi emergenti dell'uso di queste telecamere nella robotica mobile sono: flusso ottico basato sugli eventi per la robotica micro-aerea [74], prevenzione degli ostacoli [75] [76], localizzazione e mappatura simultanee (SLAM) [77] [78] e riconoscimento di oggetti [79], tra gli

[1]Consente alla luce di passare per un determinato periodo, esponendo alla luce una pellicola fotografica o sensore digitale fotosensibile per catturare un'immagine permanente di una scena.

altri.

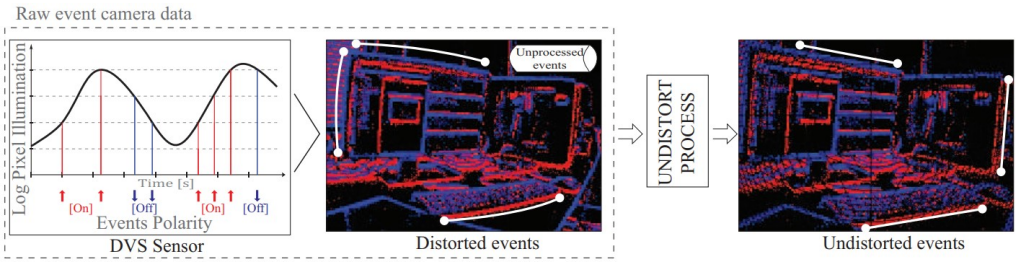


Figura 3.4: Principio di funzionamento delle Event Camera (a sinistra) con uscita distorta (al centro) e non distorta (a destra)

Le camere ad eventi sono sensori asincroni che pongono a cambiamento di paradigma nel modo in cui le informazioni visive vengono acquisite. Questo è perché campionano la luce in base alla dinamica della scena, piuttosto che su un orologio che non ha alcuna relazione con il visto scena. I loro vantaggi sono: risoluzione temporale molto elevata e bassa latenza (entrambi nell'ordine dei microsecondi), molto alta gamma dinamica (140 dB contro 60 dB delle telecamere standard), e basso consumo energetico. Quindi, le telecamere per eventi hanno un grande potenziale per la robotica e le applicazioni indossabili in scenari impegnativi per le fotocamere standard, come alta velocità e alta gamma dinamica. Sebbene le telecamere per eventi sono diventati disponibili in commercio solo dal 2008 [80], il recente corpo di letteratura su questi nuovi sensori [81] così come i recenti piani per la produzione di massa rivendicati da aziende come Samsung [82] e Prophesee [83], evidenziare che ci è un grande interesse commerciale nello sfruttare queste nuove visioni sensori per robotica mobile, realtà aumentata e virtuale (AR/VR) e applicazioni per videogiochi. Tuttavia, perché le telecamere per eventi funzionano in modo fondamentalmente diverso da fotocamere standard, misurando le variazioni di luminosità per pixel (chiamati "eventi") in modo asincrono anziché misurare la luminosità "assoluta" a velocità costante, sono necessari nuovi metodi per elaborare il loro output e sbloccare il loro potenziale.

3.2.2 Principio di Funzionamento

A differenza delle fotocamere standard, che acquisiscono immagini complete a una velocità specificata da un clock esterno (ad es. 30 fps), le event camera, come il Dynamic Vision Sensor (DVS) [80], rispondono ai cambiamenti di luminosità nella scena in modo asincrono e indipendente per ogni pixel (Figura 3.5b). Pertanto, l'uscita di una telecamera per eventi è una sequenza di velocità dati variabile di "eventi" o "picchi" digitali, con ciascuno evento che rappresenta un cambiamento di luminosità (intensità del registro)4 di magnitudine predefinita in corrispondenza di un pixel in un determinato momento5 (Figura 3.5b). Questa codifica è ispirata al natura spiking delle vie visive biologiche. Ogni pixel memorizza l'intensità del registro ogni volta che viene invia un evento e monitora continuamente per

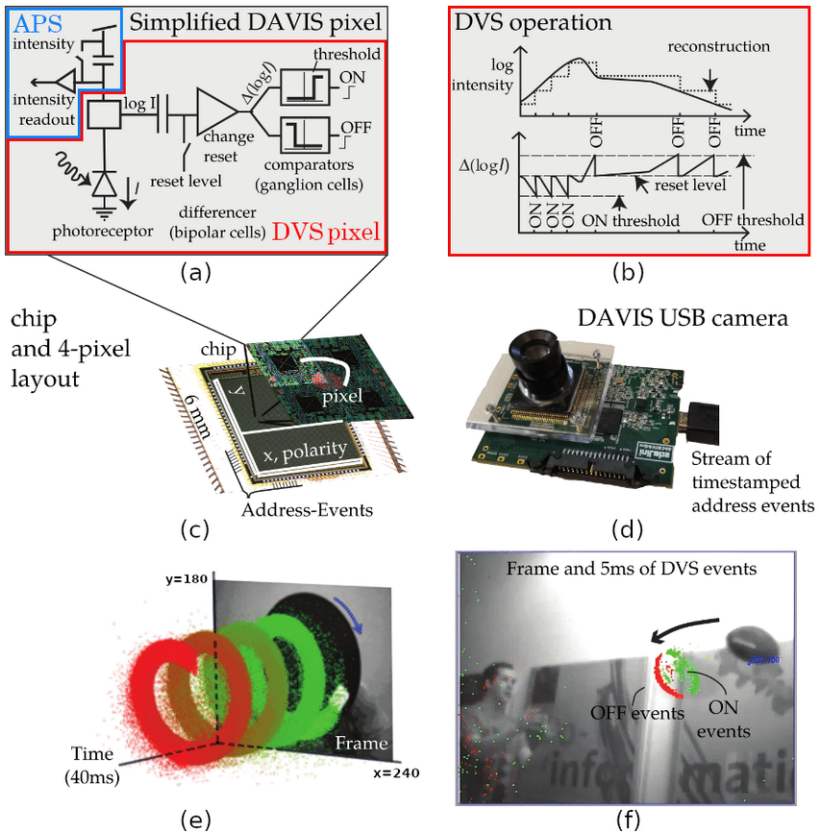


Figura 3.5: Riepilogo della telecamera DAVIS [84], comprendente un evento basato su sensore di visione dinamica (DVS [80]) e un sensore pixel attivo basato su frame **(APS)** nello stesso array di pixel, condividendo lo stesso fotodiodo in ciascuno pixel. (a) Schema circuitale semplificato del pixel DAVIS (pixel DVS in rosso, pixel APS in blu). **(b)** Schema del funzionamento di un pixel DVS, trasformare la luce in eventi. **(c)-(d)** Immagini del chip DAVIS e USB telecamera. **(e)** Un quadrato bianco su un disco nero rotante visto dalla DAVIS produce fotogrammi in scala di grigi e una spirale di eventi nello spazio-tempo. Eventi nello spazio-tempo sono codificati a colori, dal verde (passato) al rosso (presente). **(f)** Inquadratura e eventi sovrapposti di una scena naturale; le cornici sono in ritardo gli eventi a bassa latenza (colorati in base alla polarità). Immagini adattate da [84]. Un confronto più approfondito tra DVS, DAVIS e ATIS i design dei pixel dove possono essere trovati.

un cambiamento di grandezza sufficiente da questo valore memorizzato (Figura 3.5a). Quando il cambiamento supera una soglia, la fotocamera invia un evento, che viene trasmesso dal chip con x, y posizione, il tempo t e la polarità a 1 bit p del cambiamento (vale a dire, aumento della luminosità ("ON") o diminuzione ("OFF")). Questo l'uscita dell'evento è illustrata nelle Figure 3.5b, 3.5e e 3.5f.
Gli eventi vengono trasmessi dall'array di pixel alla periferia e quindi fuori dalla fotocamera utilizzando un digitale condiviso bus di uscita, in genere utilizzando la

rappresentazione dell'indirizzo-evento (AER) lettura. Questo bus può saturarsi, che perturba i tempi di invio degli eventi. Telecamere per eventi hanno velocità di lettura che vanno da 2MHz [80] a 1200MHz, a seconda del chip e del tipo di interfaccia hardware.
Le telecamere per eventi sono sensori basati sui dati: la loro uscita dipende dalla quantità di movimento o dalla variazione di luminosità nel scena. Più veloce è il movimento, più eventi al secondo vengono generati, poiché ogni pixel adatta il suo modulatore delta frequenza di campionamento alla velocità di variazione del segnale di intensità del registro che monitora. Gli eventi hanno un timestamp con microsecondi risoluzione e vengono trasmessi con una latenza inferiore al millisecondo, che fanno reagire questi sensori rapidamente agli stimoli visivi.
La luce incidente su un pixel è un prodotto dell'illuminazione della scena e della riflessione della superficie. Se l'illuminazione è approssimativamente costante, una variazione di intensità del registro segnala una riflettanza modificare. Questi cambiamenti nella riflettanza sono principalmente il risultato del movimento degli oggetti nel campo visivo. È per questo gli eventi di variazione della luminosità DVS hanno un'invarianza incorporata all'illuminazione della scena.
Confronto delle larghezze di banda dei pixel DVS e della fotocamera basata su frame Sebbene i pixel DVS siano veloci, come tutti trasduttori fisici, hanno una larghezza di banda finita: se l'intensità della luce in ingresso varia troppo rapidamente, il front-end i circuiti fotorecettori filtrano le variazioni [40]. L'aumento e tempo di caduta che è analogo al tempo di esposizione in sensori di immagine standard è il reciproco di questa larghezza di banda. La Figura 3.6 mostra un esempio di frequenza pixel DVS misurata risposta (DVS128). La configurazione della misurazione (Figura 3.6a) utilizza un segnale generato sinusoidalmente variabile per misurare la risposta. La Figura 3.6b mostra che, alle basse frequenze, il DVS pixel produce un certo numero di eventi per ciclo. Al di sopra una certa frequenza di taglio, le variazioni vengono filtrate dal dinamica dei fotorecettori, e quindi il numero di eventi per ciclo scende. Questa frequenza di taglio è monotonicamente funzione crescente di intensità luminosa. Alla luce più brillante intensità, la larghezza di banda del pixel DVS è di circa 3 kHz, equivalente a un tempo di esposizione di circa 300 µs. A 1000× inferiore intensità, la larghezza di banda DVS è ridotta a circa 300 Hz. Anche quando la luminosità del LED è ridotta di un fattore 1000, la risposta in frequenza dei pixel DVS è dieci volte superiore alla frequenza Nyquist di 30 Hz da un'immagine a 60 fps sensore. Inoltre, la fotocamera basata su frame esegue l'aliasing delle frequenze sopra la frequenza di Nyquist torna alla banda base, mentre il pixel DVS non è dovuto alla risposta temporale continua.

Event Camera Designs

La prima retina in silicio è stata sviluppata da Mahowald e Mead al Caltech durante il periodo 1986-1992, in Ph.D. tesi opera [85] che si è aggiudicata il prestigioso

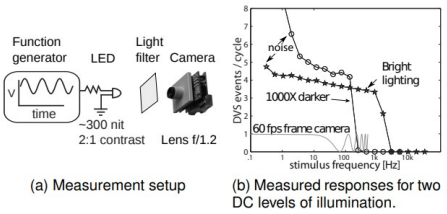


Figura 3.6: "Funzione di trasferimento degli eventi" da un singolo pixel DVS in risposta alla stimolazione LED sinusoidale. Gli eventi in background causano ulteriori Eventi ON a frequenze molto basse. La curva della fotocamera a 60 fps mostra il funzione di trasferimento incluso l'aliasing dalle frequenze sopra il Nyquist frequenza.

premio Clauser. Il sensore di Mahowald e Mead aveva pixel logaritmici, era modellato sulla retina di Kufler a tre strati e prodotto come eventi di picco di output utilizzando il protocollo AER. Però, soffriva di diversi difetti: ogni filo avvolto la scheda retina richiedeva una regolazione precisa dei potenziometri di polarizzazione; c'era una notevole discrepanza tra i risposte di diversi pixel; e i pixel erano troppo grandi per essere un dispositivo di uso pratico. Nel decennio successivo la comunità neuromorfa sviluppò una serie di retine di silicio.

La **DVS** *event camera* ha avuto la sua genesi in un frame-based design della retina al silicio in cui il fotorecettore a tempo continuo era accoppiato capacitivamente a un circuito di lettura che era resettato ogni volta che il pixel è stato campionato [86]. Più recente la tecnologia della fotocamera per eventi è stata recensita nella letteratura elettronica e neuroscientifica. Sebbene sorprendentemente molte applicazioni possano essere risolte elaborando solo gli eventi DVS (ovvero i cambiamenti di luminosità), è diventato chiaro che alcuni richiedono anche una qualche forma di staticità uscita (cioè, luminosità "assoluta"). Per ovviare a questa mancanza, ci sono stati diversi sviluppi delle fotocamere che emettono contemporaneamente informazioni dinamiche e statiche.

Il *Asynchronous Time Based Image Sensor* **(ATIS)** [87], [88] ha pixel che contengono un subpixel DVS (chiamato cambiamento CD di rilevamento) che attiva un altro subpixel per leggere il intensità assoluta (misurazione dell'esposizione EM). Il grilletto ripristina un condensatore ad alta tensione. La carica è dissanguata da questo condensatore da un altro fotodiodo. Il più luminoso è luce, più velocemente si scarica il condensatore. L'intensità dell'ATIS la lettura trasmette altri due eventi che codificano il tempo tra attraversando due tensioni di soglia, come in. In questo modo, solo i pixel che cambiano forniscono i loro nuovi valori di intensità. Il più brillante è l'illuminazione, più breve è il tempo tra questi due eventi. L'ATIS raggiunge un'ampia gamma dinamica statica (> 120dB). Tuttavia, l'ATIS ha lo svantaggio che i pixel sono almeno il doppio dell'area dei pixel DVS. Anche in scene scure il tempo tra i due eventi di intensità può essere lungo e la lettura dell'intensità può essere interrotta da nuovi eventi.

Il diffuso sensore *Dynamic and Active Pixel Vision Sensor* **(DAVIS)** [84],

illustrato in Figura 3.5 combina un sensore di pixel attivo convenzionale (APS) nello stesso pixel con DVS. Il vantaggio rispetto ad ATIS è una dimensione dei pixel molto più piccola poiché il fotodiodo è condiviso e solo il circuito di lettura aggiunge circa il 5% all'area dei pixel DVS. Frame di intensità (APS) può essere attivato a frame rate costante o su richiesta, dall'analisi degli eventi DVS, sebbene quest'ultimo sia raro sfruttato. Tuttavia, la lettura APS ha una dinamica limitata gamma (55 dB) e come una telecamera standard, è ridondante se i pixel non cambiano.

Vantaggi delle Event Camera

Le camere ad eventi offrono numerosi potenziali vantaggi rispetto a fotocamere standard:

- *Alta risoluzione temporale o (High Temporal Resolution)*: monitoraggio della luminosità i cambiamenti sono veloci, in circuiti analogici, e la lettura degli eventi è digitale, con un clock di 1 MHz, cioè gli eventi vengono rilevati e timestamp con risoluzione al microsecondo. Perciò, le camere ad eventi possono catturare movimenti molto veloci, senza soffrire del motion blur tipico delle telecamere basate su frame.

- *Bassa Latenza o (Low Latency)*: ogni pixel funziona in modo indipendente e lì non è necessario attendere un tempo di esposizione globale del fotogramma: non appena viene rilevata la variazione, viene trasmessa. Quindi, le telecamere degli eventi hanno una latenza minima: circa 10 µs in laboratorio bench e sub-millisecondi nel mondo reale.

- *Bassa Potenza o (Low Power)*: Poiché le telecamere degli eventi trasmettono solo i cambiamenti di luminosità e quindi rimuovono i dati ridondanti, l'alimentazione è utilizzato solo per elaborare i pixel che cambiano. A livello di dado, la maggior parte le fotocamere utilizzano circa 10 mW e ci sono prototipi che raggiungere meno di 10 µW. Sistemi di telecamere per eventi incorporati dove il sensore è direttamente interfacciato a un processore hanno mostrato il consumo energetico a livello di sistema (ovvero, rilevamento plus elaborazione) di 100 mW o meno.

- *High Dynamic Range HDR o (Ampia gamma dinamica)*: L'altissima dinamica la gamma di telecamere per eventi (>120 dB) supera notevolmente i 60 dB di fotocamere frame-based di alta qualità, rendendole in grado di acquisire informazioni dal chiaro di luna alla luce del giorno. è dovuto ai fatti che operano i fotorecettori dei pixel in scala logaritmica e ogni pixel funziona in modo indipendente, non aspettando un otturatore globale. Come le retine biologiche, DVS i pixel possono adattarsi a stimoli molto scuri e molto luminosi.

3.2.3 Event Processing

Una delle domande chiave del cambio di paradigma posto dalle camere ad eventi sono come estrarre informazioni significative dai dati dell'evento per svolgere un determinato compito. Questo è veramente domanda ampia, poiché la risposta dipende dall'applicazione, e guida la progettazione algoritmica del risolutore di attività.

Le event camera acquisiscono le informazioni in modo asincrono e sparse, con alta risoluzione temporale e bassa latenza. Quindi, l'aspetto temporale, specialmente la latenza, gioca un ruolo essenziale nel modo in cui gli eventi vengono elaborati. Dipendente su quanti eventi vengono elaborati contemporaneamente si possono distinguere due categorie di algoritmi: (i) metodi che operano evento per evento, in cui lo stato di il sistema (le incognite stimate) può cambiare al arrivo di un singolo evento, ottenendo così una latenza minima, e (ii) metodi che operano su gruppi o pacchetti di eventi, che introducono una certa latenza.

Ortogonalmente, a seconda di come vengono elaborati gli eventi, possiamo distinguere tra approcci basati su modelli e approcci privi di modelli (cioè basati sui dati, apprendimento automatico). Supponendo che gli eventi vengano elaborati in un framework di ottimizzazione, un'altra classificazione riguarda il tipo di funzione obiettivo o di perdita utilizzata: geometrica vs temporale vs. su base fotometrica (ad esempio, una funzione della polarità dell'evento o l'attività dell'evento). Ogni categoria presenta metodi con vantaggi e svantaggi e attuali focus di ricerca sull'esplorazione delle possibilità che ogni metodo può offrire.

A. Rappresentazione degli Eventi

Gli eventi vengono elaborati e spesso trasformati in alternative rappresentazioni (Figura 3.7) che facilitano l'estrazione di informazioni significative ("caratteristiche") per risolvere un determinato compito. Qui esaminiamo le rappresentazioni popolari dei dati degli eventi. Molti di nascono dalla necessità di aggregare le poche informazioni veicolati dai singoli eventi in assenza di ulteriori conoscenza. Alcune rappresentazioni sono semplici, fatte a mano trasformazioni dei dati mentre altri sono più elaborati.

Eventi Singoli $e_k \doteq (x_k, t_k, p_k)$ sono utilizzati da metodi di elaborazione evento per evento, come i filtri probabilistici e Spiking Neural Networks **(SNNs)**. Il filtro o SNN ha informazioni aggiuntive, costruite dal passato eventi o dati da conoscenze aggiuntive, che si fondono con l'evento in ingresso in modo asincrono per produrre un output.

Pacchetto dell'Evento: Eventi in uno spazio-temporale quartiere vengono elaborati insieme per produrre un output. Questa rappresentazione conserva informazioni precise su timestamp e polarità. Scelta della dimensione del pacchetto appropriata Ne è fondamentale per soddisfare le ipotesi dell'algoritmo (ad es. velocità di movimento costante durante l'arco del pacchetto), che varia con il compito.

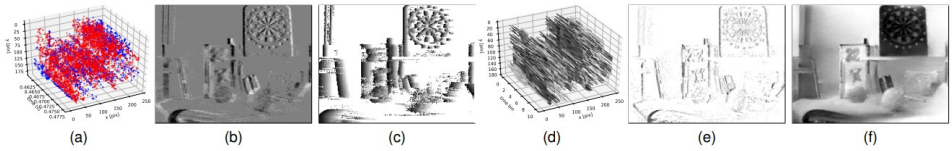


Figura 3.7: Diverse rappresentazioni di eventi della sequenza di profondità del cursore [89]. **(a)** Eventi nello spazio-tempo, colorati secondo la polarità (positivo in blu, negativo in rosso). **(b)** Frame dell'evento (immagine di incremento della luminosità $\Delta L(x)$). **(c)** Superficie temporale con l'ultimo timestamp per pixel (pixel più scuri indicare l'ora recente), solo per gli eventi negativi. **(d)** Griglia voxel interpolata ($240 \times 180 \times 10$ voxel), colorata in base alla polarità, da scura (negativa) a brillante (positivo). **(e)** Immagine dell'evento con compensazione del movimento [90] (i bordi nitidi ottenuti dall'accumulo di eventi sono più scuri dei pixel senza eventi, in bianco). **(f)** Immagine di intensità ricostruita da [91]. Le rappresentazioni a griglia sono compatibili con i metodi di visione artificiale convenzionali [92],

Event frame/image o istogrammi 2D: Gli eventi in un quartiere spazio-temporale sono convertiti in un semplice modo (ad esempio, contando eventi o accumulando polarità pixel-wise) in un'immagine (griglia 2D) che può essere alimentata ad algoritmi di visione artificiale basati su immagini. Alcuni algoritmi possono funzionano nonostante le diverse statistiche dei frame degli eventi e immagini naturali. Tali istogrammi possono fornire una frequenza di campionamento naturale guidata dall'attività; vedere [93] per i metodi per accumulare tali frame per il flusso di calcolo. Tuttavia, questo la pratica non è ideale nel paradigma basato sugli eventi perché quantizza i timestamp degli eventi, può scartare la scarsità (ma vedere [94]), e le immagini risultanti sono altamente sensibili a il numero di eventi utilizzati. Tuttavia l'alto impatto di frame di eventi in letteratura [93], [95] è chiaro perché (i) sono un modo semplice per convertire al flusso di eventi non familiari in una rappresentazione 2D familiare contenente informazioni spaziali sui bordi della scena, che sono le regioni più informative nelle immagini naturali, (ii) informano non solo sulla presenza di eventi ma anche sulla loro assenza (che è informativa), (iii) hanno un'interpretazione intuitiva (ad esempio, una mappa dei bordi, una luminosità incremento immagine) e (iv) sono la struttura dati compatibile con la visione artificiale convenzionale.

Time Surface (TS): Una TS è una mappa 2D in cui ogni pixel memorizza un singolo valore temporale (ad esempio, il timestamp dell'ultimo evento in quel pixel). Così gli eventi vengono convertiti in un'immagine la cui "intensità" è funzione del movimento storia in quella posizione, con valori maggiori corrispondenti a una mozione più recente. I TS sono chiamati Motion History Images nella computer vision classica. Espongono esplicitamente il ricche informazioni temporali degli eventi e possono essere aggiornate asincrono. Utilizzando un kernel esponenziale, i TS enfatizzano gli eventi recenti rispetto agli eventi passati. Per ottenere l'invarianza a velocità di movimento, viene proposta la normalizzazione. Rispetto

ad altre rappresentazioni di eventi simili a una griglia, TS altamente comprimere le informazioni in quanto mantengono solo un timestamp per pixel, quindi la loro efficacia si riduce su scene strutturate, in cui i pixel aumentano frequentemente. Per rendere i TS meno sensibili al rumore, ogni valore di pixel può essere calcolato filtrando gli eventi in una finestra spazio-temporale.

Voxel Grid: è un istogramma spazio-temporale (3D) di eventi, dove ogni voxel rappresenta un particolare pixel e tempo intervallo. Questa rappresentazione conserva meglio l'informazione temporale degli eventi evitando di farli crollare su una griglia 2D (Figura 3.7). Se viene utilizzata la polarità, la griglia voxel è una discretizzazione intuitiva di un campo scalare (polarità $p(x, y, t)$ o variazione di luminosità $\partial L(x, y, t)/\partial t$) definita sull'immagine piano, con assenza di eventi contrassegnati da polarità zero. Ogni la polarità dell'evento può essere accumulata su un voxel o diffondersi tra i suoi voxel più vicini usando un kernel. Entrambi gli schemi quantizzano i timestamp degli eventi, ma il secondo (griglia voxel interpolata) fornisce una precisione sub-voxel.

Set di punti 3D: Eventi in un quartiere spazio-temporale sono trattati come punti nello spazio 3D, $(x_k, y_k, t_k) \in \mathbb{R}^3$. Così il la dimensione temporale diventa geometrica. È scarso rappresentazione e viene utilizzato su metodi di elaborazione geometrica basati su punti, come l'adattamento del piano [96] o PointNet [97].

Set di punti sul piano dell'immagine: Gli eventi sono trattati come insieme in evoluzione di punti 2D sul piano dell'immagine. È un popolare rappresentazione tra i primi metodi di tracciamento della forma basata su turno medio o ICP [98], [99], [100], [101], [102] dove gli eventi forniscono gli unici dati necessari per tenere traccia dei pattern edge.

Immagine dell'evento con compensazione del movimento: è una rappresentazione che dipende non solo dagli eventi ma anche dall'ipotesi del moto. L'idea della compensazione del movimento è che, come un bordo si muove sul piano dell'immagine, innesca eventi sul pixel che attraversa; il movimento del bordo può essere stimato deformando gli eventi a un tempo di riferimento e massimizzando il loro allineamento, producendo un'immagine nitida (cioè istogramma) di eventi deformati (IWE). Quindi, questa rappresentazione (IWE) suggerisce un criterio per misurare quanto bene gli eventi si adattano un movimento candidato: più nitidi sono gli spigoli prodotti da eventi di deformazione, migliore è l'adattamento. Inoltre, le immagini risultanti con compensazione del movimento hanno un significato intuitivo (vale a dire, i modelli di bordo che causano gli eventi) e forniscono a rappresentazione più familiare delle informazioni visive rispetto al eventi. In un certo senso, la compensazione del movimento rivela un nascosto mappa ("invariante al movimento") degli archi nel flusso di eventi. Le immagini possono essere utili per ulteriori elaborazioni, come funzionalità tracciamento. Esistono versioni con compensazione del movimento di insiemi di punti e superfici temporali.

Immagini ricostruite: Immagini di luminosità ottenute da ricostruzione dell'immagine può essere interpretata come a rappresentazione più invariante rispetto al movimento rispetto ai frame di evento o TS, ed essere utilizzato per l'inferenza producendo risultati di prim'ordine.

Polarità degli eventi: può essere considerata in due modi: elaborare separatamente gli eventi positivi e negativi e fonderli risultati (ad esempio, utilizzando TS), o elaborandoli insieme in una rappresentazione comune (ad esempio, incrementi di luminosità immagini), dove la polarità è spesso aggregata tra eventi vicini. La polarità dell'evento dipende dalla direzione del movimento, quindi è una seccatura per le attività che dovrebbero essere indipendenti dal movimento, come il riconoscimento di oggetti (per mitigare questo, i dati di allenamento da più direzioni di movimento dovrebbero essere disponibile). Per le attività di stima del movimento, la polarità può essere utile, soprattutto per rilevare bruschi cambi di direzione.

Un quadro generale per convertire i dati degli eventi in alcune delle rappresentazioni basate su griglia di cui sopra sono presentate in. Studia anche come la scelta della rappresentazione passato a una rete neurale artificiale (**ANN**) influisce sul compito prestazioni e di conseguenza propone di automaticamente imparare la rappresentazione che massimizza tale performance

3.3 Dynamic Vision System DV

DV è il software per i sensori di visione dinamica iniVation (DVS/DAVIS). DV si collega alla videocamera e ne mostra l'uscita. È anche il kit di sviluppo software ufficiale per scrivere software applicativo per telecamere basate su eventi. DV è una piattaforma per lo sviluppo di applicazioni C++ integrate, implementabili e ad alte prestazioni. Tuttavia, per la prototipazione rapida e la sperimentazione, molti sviluppatori preferiscono utilizzare Python; infatti si utilizza **dv-python** [103] una libreria che collega uno script python agli output del sistema di visione dinamico DV. Viene utilizzato insieme al Dynamic Vision Sensor (DVS). Questa libreria aiuta a:

- ottenere dati in tempo reale (grezzi o elaborati) da DV nella applicazione Python;
- aprire i file registrati con DV;
- aprire file più vecchi, registrati con software ormai obsoleto (come jaer)

3.4 Python

Python[104] è un linguaggio di programmazione di più "alto livello" rispetto alla maggior parte degli altri linguaggi, orientato a oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing.
È un linguaggio multi-paradigma che ha tra i principali obiettivi: dinamicità, semplicità e flessibilità. Supporta il paradigma object oriented, la programmazione strutturata e molte caratteristiche di programmazione funzionale e riflessione.

Le caratteristiche più immediatamente riconoscibili di Python sono le variabili non tipizzate e l'uso dell'indentazione per la sintassi delle specifiche, al posto delle più comuni parentesi.
Altre caratteristiche distintive sono l'overloading di operatori e funzioni tramite delegati, la presenza di un ricco assortimento di tipi e funzioni di base e librerie standard, sintassi avanzate quali slicing e list comprehension.
Il controllo dei tipi è forte (strong typing) e viene eseguito in runtime (dynamic typing): una variabile è un contenitore a cui viene associata un'etichetta (il nome) che può essere associata a diversi contenitori anche di tipo diverso durante il suo tempo di vita. Fa parte di Python un sistema garbage collector per liberazione e recupero automatico della memoria di lavoro.
Python ha qualche somiglianza con Perl, ma i suoi progettisti hanno scelto una sintassi più essenziale e uniforme con l'obiettivo di migliorare la leggibilità del codice. Analogamente a Perl è classificato spesso come linguaggio di scripting, ma pur essendo utile per scrivere script di sistema, in alternativa per esempio a bash, la grande quantità di librerie disponibili e la facilità con cui il linguaggio permette di scrivere software modulare favoriscono anche lo sviluppo di applicazioni molto complesse.

3.4.1 Contenitori in Python

Python considera in generale come contenitori gli oggetti che prevedono la possibilità di iterare su un insieme di elementi, perciò utilizzabili all'interno di contesti quali il ciclo `for` e funzioni quali somma, ricerca e ordinamento. I contenitori in genere permettono di contenere dati di tipo eterogeneo.
Per quanto riguarda i contenitori standard propriamente detti, sono classificabili come sequenze, *insiemi* e *dizionari*. I contenitori seguono una filosofia comune e condividono gran parte dei metodi.
Le *sequenze* sono contenitori ordinati, che condividono metodi basati sull'ordinamento, l'indicizzazione intera e la creazione di sottosequenze tramite *slicing*.
Le *liste* (`list`) sono sequenze estendibili, mentre le tuple (`tuple`) sono sequenze immutabili. Anche le stringhe alfanumeriche (`str` e `unicode`) sono considerate sequenze. A partire da Python 3.0, i tipi `str` e `unicode` sono unificati e compare il tipo `byte`, equivalente grosso modo a una stringa binaria.
Sono previste tutte le operazioni classiche sulle stringhe come concatenamento, formattazione, ricerca, sostituzione e così via. Le stringhe in Python sono sequenze immutabili, cosicché qualsiasi operazione che in qualche modo potrebbe alterare una stringa, per esempio la sostituzione di un carattere, restituisce in effetti una nuova stringa, come avviene in Java e in C#.
Altri contenitori sono i *dizionari* (`dict`), conosciuti in altri contesti con il nome di hash table oppure array associativi. Esiste una sintassi per la creazione di dizionari, i cui elementi sono specificati da una coppia di dati separati da due punti ':'. Il primo elemento della coppia rappresenta l'indice, detto "chiave", e il secondo è il suo valore corrispondente. Infatti ogni elemento di un dizionario è detto anche

"coppia chiave-valore".
Per esempio l'istruzione seguente crea un dizionario identificato come `diz` composto da due elementi le cui chiavi sono `wikipedia` e `wikiquote`, rispettivamente e con associati i valori interi `40` e `60`:

```
1 diz = {'wikipedia': 40, 'wikiquote': 60}
```

Listing 3.2: esempio1

Le chiavi in un dizionario sono immutabili, mentre il valore corrispondente a ciascuna chiave è alterabile tramite un'assegnazione. La seguente istruzione modifica il valore corrispondente a "wikipedia", portandolo a 4500:

```
1 diz['wikipedia'] = 4500
```

Listing 3.3: esempio2

3.4.2 Moduli

I **moduli** costituiscono il meccanismo usato in Python per l'utilizzo di codice definito esternamente. Ogni modulo definisce un proprio namespace globale in cui definire variabili, funzioni, ecc., isolato dagli altri moduli (diversi moduli possono usare gli stessi nomi senza interferenze). All'avvio dell'interprete Python, viene creato un modulo principale in cui sono inseriti gli oggetti dichiarati:

- in modalità interattiva, questo contiene gli oggetti creati nelle istruzioni eseguite man mano dall'utente
- eseguendo un file, contiene gli oggetti dichiarati nel file stesso

Tramite l'*importazione*, è possibile da un modulo richiamarne altri ed utilizzarne le funzionalità

3.4.3 Librerie

Libreria Standard

La libreria standard [105] fornisce un insieme di moduli con funzionalità di uso comune. Questi moduli possono essere importati con l'istruzione `import` e usati allo stesso modo di quelli creati dall'utente.

Librerie Esterne

Le funzionalità di Python possono essere estese oltre quelle della libreria standard tramite librerie di terze parti [105]. Ogni libreria fornisce nuovi moduli e package utilizzabili all'interno dei propri programmi, ogni libreria può a sua volta dipendere da altre librerie. Il *Python Package Index (PyPI)* è un database online di oltre 100.000 librerie disponibili per Python https://pypi.python.org/

pip

L'utility `pip` inclusa in Python può essere usata da linea di comando per installare package da PyPI. Ad es., per installare la libreria NumPy usare dal terminale (non da Python) il comando:

```
$ pip install numpy
```

Listing 3.4: pip

- si può indicare una versione specifica (es. "`numpy==1.14`")
- è possibile specificare più package insieme

Se una libreria dipende da altre, queste sono installate automaticamente. Una libreria può essere installata

- a livello di sistema (servono diritti di amministratore)
- solo per l'utente corrente (con l'opzione "`--user`")

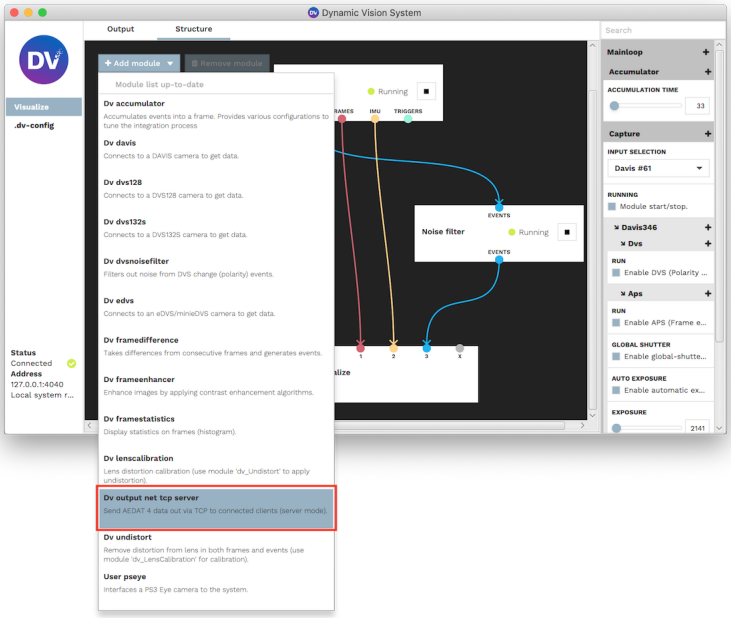
Capitolo 4

Sviluppo Del Progetto

4.1 Introduzione

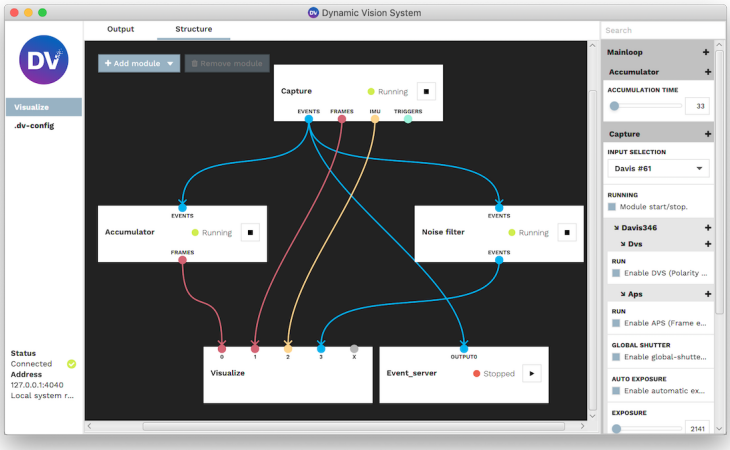
Nella parte iniziale del progetto andremo ad installare i vari strumenti che utilizzeremo per il suo sviluppo, quindi scarichiamo i software **DV** [106], **Python** [107] utilizzando utilizzando il code editor: *Visual Studio Code* [108].
Andiamo ad analizzare il file `dvSave.aedat4` mediante l'ausilio di DV:

1. lanciare DV;
2. andare sulla tab *Structure*;
3. aggiungere `Dv net output tcp server` cliccando su *Add module*;



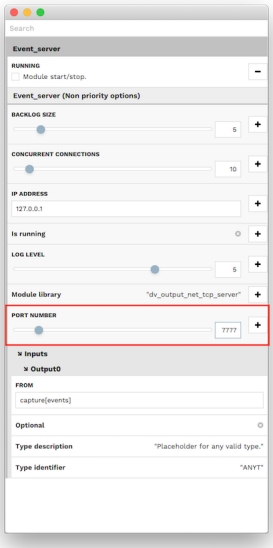
4. assegna un nome al modulo come desideri, ad esempio "event_server";

5. collega l'uscita `events` della tua telecamera con l'ingresso del modulo aggiunto;



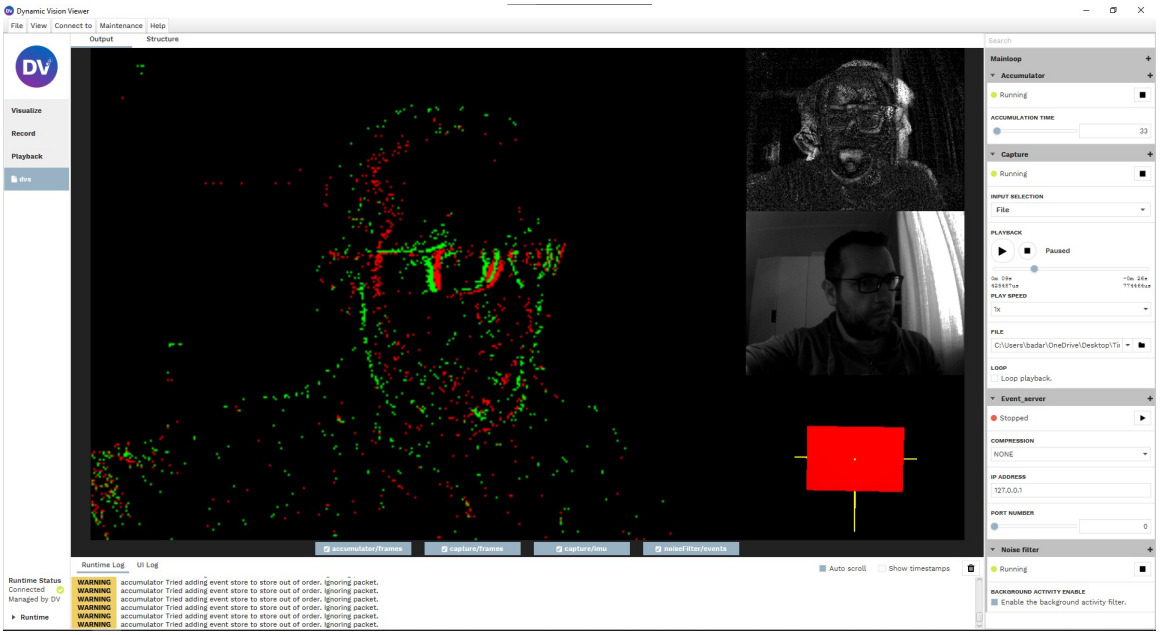
6. il modulo assume una porta casuale per impostazione predefinita. Se sei soddisfatto di ciò, puoi avviare il modulo premendo il pulsante di riproduzione e quindi osservare come il numero di porta cambia da 0 alla porta reale effettiva, utilizzerai quel numero in seguito per connetterti al modulo. In caso contrario, fai clic sul pulsante più accanto al nome dei moduli nell'elenco di configurazione nella barra laterale destra;

7. per cambiare la porta, cambia l'impostazione nella finestra che è apparsa;



8. nella stessa finestra, se necessario, puoi modificare la configurazione del "Max Data Backlog". Per impostazione predefinita, il modulo server TCP elimina i dati se il client Python non riesce a tenere il passo. Per evitare ciò, modificare il valore in '-1';

Il file viene visto nel seguente modo nel software DV:



4.2 Face Meshing e Face Processing

Andare su Python e installare la varie librerie per creare l'algoritmo, quindi installare in visual studio code le librerie dv-python, mediapipe, cv2 attraverso l'istruzione `pip install`.

Creare il file in python dove estrapola il video in cui fa vedere il Facemesh dove c'è la maschera con i 468 landmarks, in cui i sono evidenziati i landmarks degli occhi e delle labbra con diversi colori. Questa istruzione viene fatta con il file `"faceMeshing.py"`.
Il seguente programma si occupa dell'apertura del file con estensione '.aedat4', vengono iterati i frame e contemporaneamente viene applicato l'algoritmo faceMesh per la rilevazione dei keypoints appartenenti al volto. I dati relativi ai keypoints identificati dall'algoritmo vengono salvati in un file `"dvSaveFrames.json"`:

```
1 import cv2
2 import utilities.landmarks as ld
3 import utilities.functions as ft
4 import mediapipe as mp
5 from mediapipe.framework.formats import landmark_pb2
6 from dv import AedatFile
```

```
7
8  mp_drawing = mp.solutions.drawing_utils
9  mp_face_mesh = mp.solutions.face_mesh
10
11 list, dict, count = [], {}, 0
12 with AedatFile(ft.genPath+"dvSave.aedat4") as f:
13     with mp_face_mesh.FaceMesh(min_detection_confidence=0.5,
       min_tracking_confidence=0.5) as face_mesh:
14         for frame in f["frames"]:
15             if count == 0:
16                 baseDate = frame.timestamp/1e6
17                 count = 1
18             timestamp = (frame.timestamp/1e6)-baseDate
19
20             image = cv2.cvtColor(frame.image, cv2.COLOR_GRAY2RGB)
21             annotated_image = image.copy()
22             results = face_mesh.process(image)
23
24             if results.multi_face_landmarks:
25                 for face_landmarks in results.multi_face_landmarks
       :
26
27                     dict, landmark_list = ft.calculateKeypoints(
       face_landmarks)
28
29                     mp_drawing.draw_landmarks(image=
       annotated_image,landmark_list=landmark_pb2.
       NormalizedLandmarkList(landmark = landmark_list[0]),
       landmark_drawing_spec=mp_drawing.DrawingSpec(color=ld.RED_COLOR
       , thickness=1, circle_radius=1))
30                     mp_drawing.draw_landmarks(image=
       annotated_image,landmark_list=landmark_pb2.
       NormalizedLandmarkList(landmark = landmark_list[1]),
       landmark_drawing_spec=mp_drawing.DrawingSpec(color=ld.
       YELLOW_COLOR, thickness=1, circle_radius=1))
31                     mp_drawing.draw_landmarks(image=
       annotated_image,landmark_list=landmark_pb2.
       NormalizedLandmarkList(landmark = landmark_list[2]),
       landmark_drawing_spec=mp_drawing.DrawingSpec(color=ld.
       BLUE_COLOR, thickness=1, circle_radius=1))
32                     mp_drawing.draw_landmarks(image=
       annotated_image,landmark_list=landmark_pb2.
       NormalizedLandmarkList(landmark = landmark_list[3]),
       landmark_drawing_spec = mp_drawing.DrawingSpec(color=ld.
       GREEN_COLOR, thickness=1, circle_radius=1))
33
34                     cv2.putText(annotated_image, 't: '+str(
       timestamp), (0,20), cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255))
35                     cv2.imshow("faceMeshing", annotated_image)
36
37                 dict["timestamp"] = timestamp
38                 list.append(dict)
39
```

```
                cv2.waitKey(1)
                """
                #scansione manuale dei timestamp
                if cv2.waitKey() == ord('q'):
                    break
                """

ft.writeJson(list,"dvSaveFrames")
```

Listing 4.1: faceMeshing.py

Il video viene visualizzato nel seguente modo avviando `"faceMeshing.py"`:

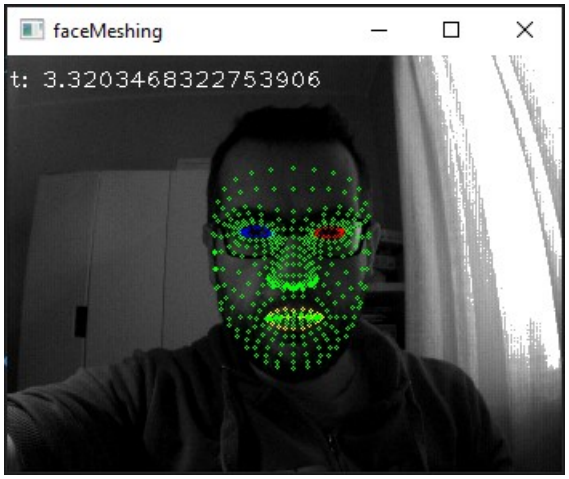


Figura 4.1

I keypoints vengono presi da un altro file denominato `"landmarks.py"`, il quale contiene tutti i keypoints e la loro suddivisione in base al landmark di appartenenza.

```
#lips
lipsUpperOuter = [61, 185, 40, 39, 37, 0, 267, 269, 270, 409, 291]
lipsLowerOuter = [146, 91, 181, 84, 17, 314, 405, 321, 375, 291]
lipsUpperInner = [78, 191, 80, 81, 82, 13, 312, 311, 310, 415,
    308]
lipsLowerInner = [78, 95, 88, 178, 87, 14, 317, 402, 318, 324,
    308]
LIPS = lipsUpperOuter + lipsLowerOuter + lipsUpperInner +
    lipsLowerInner

#right eye
rightEyeUpper0 = [246, 161, 160, 159, 158, 157, 173]
rightEyeLower0 = [33, 7, 163, 144, 145, 153, 154, 155, 133]
R_EYE = rightEyeUpper0 + rightEyeLower0

#left eye
leftEyeUpper0 = [466, 388, 387, 386, 385, 384, 398]
leftEyeLower0 = [263, 249, 390, 373, 374, 380, 381, 382, 362]
```

```
16 L_EYE = leftEyeUpper0 + leftEyeLower0
17
18 #all landmarks
19 TOT_LANDMARKS = []
20 for i in range(468):
21         TOT_LANDMARKS.append(i)
22
23 KEYPOINTS = L_EYE + R_EYE + LIPS
24 SILHOUETTE = [x for x in TOT_LANDMARKS if x not in KEYPOINTS]
25
26 LANDMARKS = L_EYE+R_EYE+LIPS+SILHOUETTE
27
28 #colours
29 RED_COLOR = (0, 0, 255)
30 GREEN_COLOR = (0, 255, 0)
31 YELLOW_COLOR = (0, 255, 255)
32 BLUE_COLOR = (255, 0, 0)
33
34 def recognizeLandmark(values):
35     for x in L_EYE:
36         if(x == values):
37             return "leftEye"
38     for x in R_EYE:
39         if(x == values):
40             return "rightEye"
41     for x in LIPS:
42         if(x == values):
43             return "lips"
44     for x in SILHOUETTE:
45         if(x == values):
46             return "silhouette"
47
48 def getCoupleEyes():
49     return len(L_EYE)
50
51 def getCoupleLips():
52     return len(LIPS)
53
54 def getCoupleSilhouette():
55     return len(SILHOUETTE)
```

Listing 4.2: landmarks.py

Inoltre in un file separato vengono inserite varier funzioni e algoritmi di supporto denominato `"functions.py"`.

Successivamente andiamo a creare un altro file in cui il programma si occupa del processamento dei dati contenute nel file `"dvSaveFrames.json"`, calcolando direzione e velocità dei keypoints relativi ai diversi landmarks, mostrando i loro andamenti mediante dei grafici. Successivamente questi dati vengono salvate in un altro file `"dvSaveDerivatives.json"`.

```
1 import utilities.functions as ft
2
```

```
with open(ft.jsonPath+"dvSaveFrames.json", 'r') as file:
    data = ft.load(file)
    timestamps = ft.getTimestamps(data)
    coupleCoords = ft.getCoupleCoords()

    matLE = ft.getXYYValues(data,coupleCoords[0],'leftEye')
    matLI = ft.getXYYValues(data,coupleCoords[1],'lips')
    matRE = ft.getXYYValues(data,coupleCoords[2],'rightEye')

    #calcolo delle velocita e direzioni
    der = ft.calculateDerivatives(matLE,matLI,matRE,data,
    timestamps,coupleCoords)

    #scrivo i dati nel file
    ft.writeJson(der,"dvSaveDerivatives")

#grafico dei valori delle velocita e della direzione per ogni
    coppia per ogni landmark
with open(ft.jsonPath+"dvSaveDerivatives.json", 'r') as file:
    data = ft.load(file)
    matLE = ft.getDerValues(data,coupleCoords[0],'leftEye')
    matLI = ft.getDerValues(data,coupleCoords[1],'lips')
    matRE = ft.getDerValues(data,coupleCoords[2],'rightEye')
    ft.printGraph(matLE,coupleCoords[0],timestamps[1:-1],'Left Eye
    ')
    ft.printGraph(matLI,coupleCoords[1],timestamps[1:-1],'Lips')
    ft.printGraph(matRE,coupleCoords[2],timestamps[1:-1],'Right
    Eye')
```

Listing 4.3: faceProcessing.py

I grafici che si ottengono sono i seguenti:

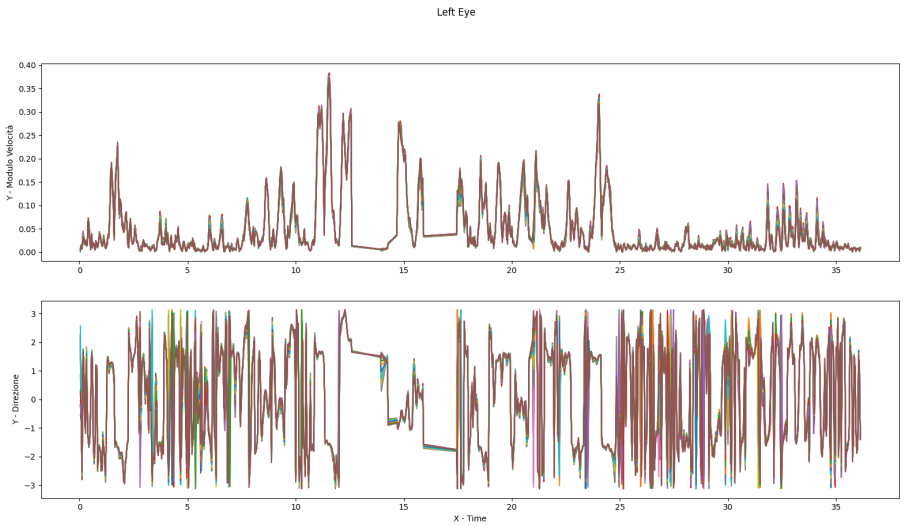


Figura 4.2: Left Eye

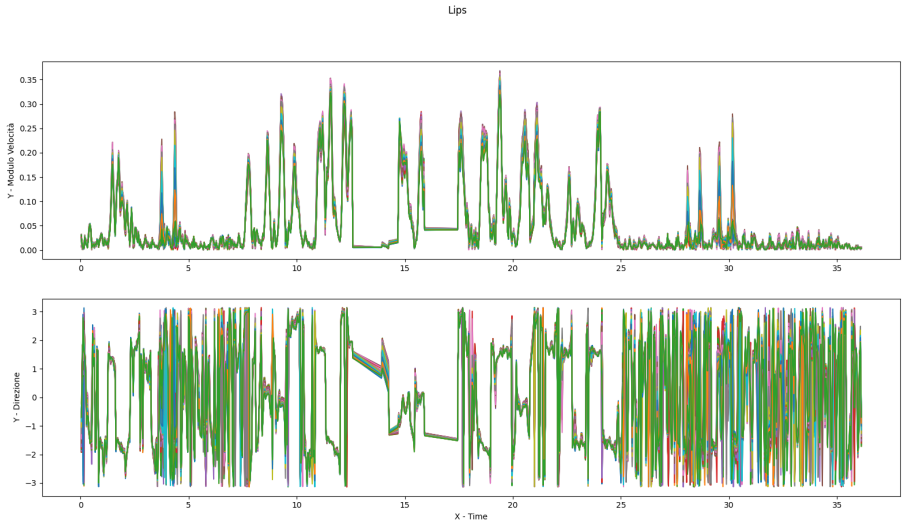


Figura 4.3: Lips

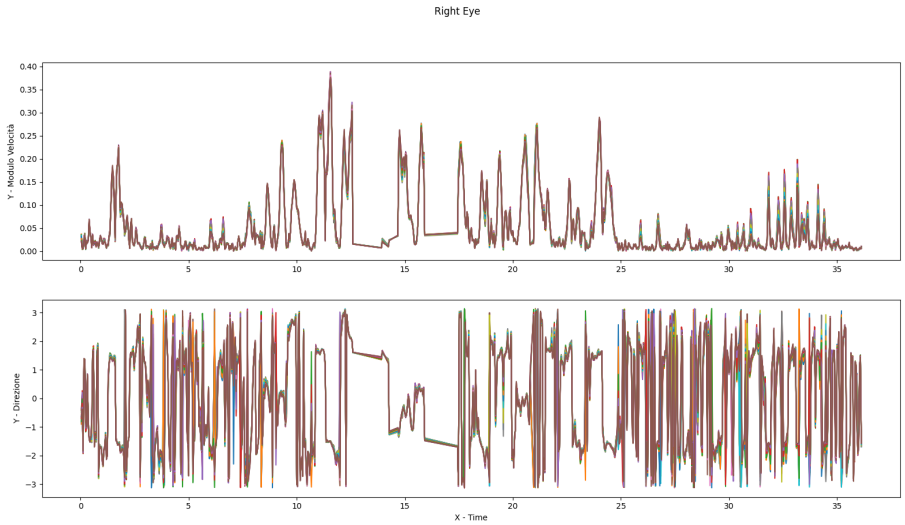


Figura 4.4: Right Eye

Successivamente si va controllare soltanto i timestamp degli occhi che si chiudono, e quindi si è proceduto alla realizzazione di uno script che si occupa del processamento dei dati contenuti nel file `"dvSaveFrames.json"`, limitando l'analisi agl intervalli contenuti nei file `"analyzeEye.json"` e `"analyzeLips.json"`. Le informazioni vengono salvate nel file `"dvSaveReducedDerivatives.json"`

```
import utilities.functions as ft

redTimestampsEye = ft.getReducedTimestamps('analyzeEyes')
```

```
redTimestampsLips = ft.getReducedTimestamps('analyzeLips')

with open(ft.jsonPath+"dvSaveFrames.json", 'r') as file:
    data = ft.load(file)
    timestamps = ft.getTimestamps(data)
    coupleCoords = ft.getCoupleCoords()

    matLE = ft.getXYFilteredValues(data,coupleCoords[0],
    redTimestampsEye,'leftEye')
    matLI = ft.getXYFilteredValues(data,coupleCoords[1],
    redTimestampsLips,'lips')
    matRE = ft.getXYFilteredValues(data,coupleCoords[2],
    redTimestampsEye,'rightEye')

    #calcolo delle velocita e direzioni
    der = ft.calculateRedDerivatives(matLE,matLI,matRE,data,
    timestamps,coupleCoords)

    #scrivo le informazioni nel file
    ft.writeJson(der,"dvSaveReducedDerivatives")


#grafico dei valori delle velocita e della direzione per ogni
    coppia per ogni landmark
with open(ft.jsonPath+"dvSaveReducedDerivatives.json", 'r') as
    file:
    data = ft.load(file)
    matLE = ft.getNanValues(data,coupleCoords[0],'leftEye')
    matLI = ft.getNanValues(data,coupleCoords[1],'lips')
    matRE = ft.getNanValues(data,coupleCoords[2],'rightEye')

    ft.printGraph(matLE,coupleCoords[0],timestamps[1:-1],'Left Eye
    ')
    ft.printGraph(matRE,coupleCoords[2],timestamps[1:-1],'Right
    Eye')
    ft.printGraph(matLI,coupleCoords[1],timestamps[1:-1],'Lips')
```

Listing 4.4: filteredProcesssing.py

I grafici ottenuti sono le Figure 4.5, 4.6, 4.7; tale figure rappresentano una versione ridotta dei grafici di `"faceProcessing.py"`. Mentre il contenuto dei file `"analyzeEye.json"` e `"analyzeLips.json"` è stato trovato attraverso `"faceMeshing.py"` (Listing 4.1) andando ad analizzare frame per frame del video, controllando in quale timestamp si verificava l'apertura e la chiusura degli occhi e delle labbra.

4.3 Eye Meshing e Eye Processing

Andiamo ad analizzare ora le caratteristiche degli occhi, analizzando come si muovono in dei determinati tempi; a tale scopo si usa lo script `"eyeMeshing.py"` che è

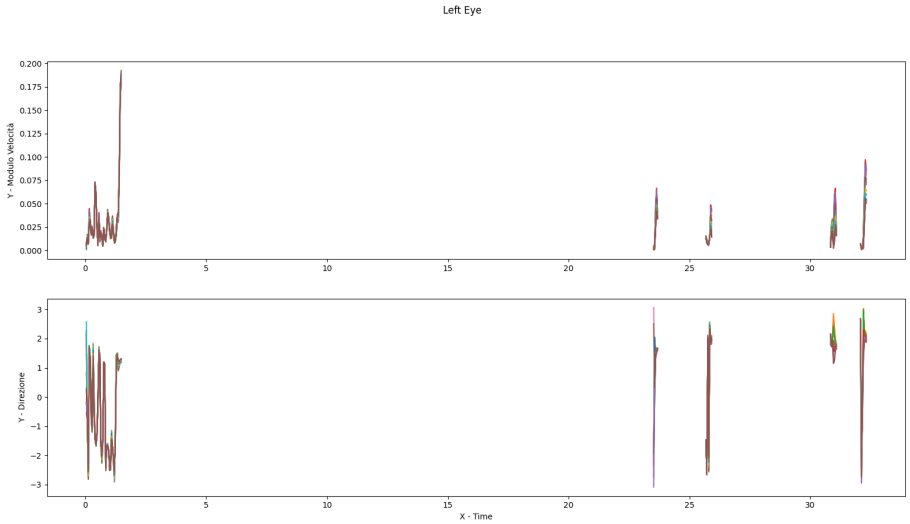


Figura 4.5: Left Eye

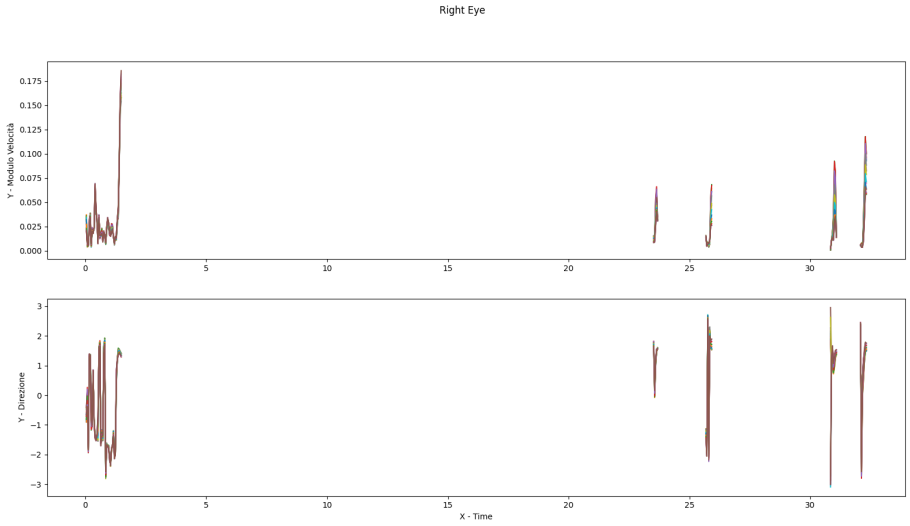


Figura 4.6: Lips

diviso in due parti: la prima è una versione ridotta di `"faceMeshing.py"` (Codice 4.1), che mostra a video il volto circondato da due bounding boxes tempo varianti, che delimiteranno l'area di appartenenza degli eventi successivamente catturati. Le informazioni relative agli occhi vengono salvate nel file `"dvSaveEyesFrames.json"`, mentre quelle relative ai bounding boxes vengono salvate nel file `"eyesBoundingBoxes.json"`. La seconda parte analizza tutti gli eventi appartenenti agli intervalli delineati nel file `"analyzeEyes.json"`; se tali eventi appartengono ai bounding boxes dei due occhi, allora vengono salvati nel file `"dvSaveEyesEvents.json"`.

```
1 import cv2
```

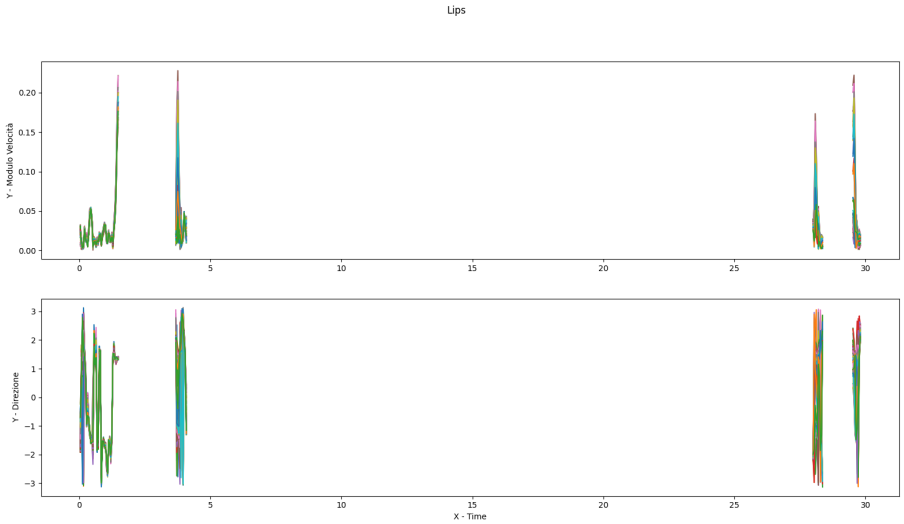


Figura 4.7: Right Eye

```
import utilities.landmarks as ld
import utilities.functions as ft
import mediapipe as mp
import numpy as np
from mediapipe.framework.formats import landmark_pb2
from dv import AedatFile

mp_drawing = mp.solutions.drawing_utils
mp_face_mesh = mp.solutions.face_mesh

image = 0
list,redDdict,count,baseDate = [], {}, 0, 0
reducedTimestamps = ft.getReducedTimestamps("analyzeEyes")

bboxes,eyesDict = [],{}

with AedatFile(ft.genPath+"dvSave.aedat4") as recording:
    with mp_face_mesh.FaceMesh(min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as face_mesh:
        for frame in recording["frames"]:
            if count == 0:
                baseDate = frame.timestamp/1e6
                count = 1
            timestamp = (frame.timestamp/1e6)-baseDate
            image = cv2.cvtColor(frame.image, cv2.COLOR_GRAY2RGB)
            annotated_image = image.copy()
            results = face_mesh.process(image)
            if results.multi_face_landmarks:
                for face_landmarks in results.multi_face_landmarks
    :
                    dict,landmark_list = ft.calculateKeypoints(
    face_landmarks)
```

```
31                      redDict = ft.getEyesValues(face_landmarks,
    reducedTimestamps,timestamp)
32
33                      mp_drawing.draw_landmarks(image=
    annotated_image,landmark_list=landmark_pb2.
    NormalizedLandmarkList(landmark = landmark_list[0]),
    landmark_drawing_spec=mp_drawing.DrawingSpec(color=ld.RED_COLOR
    , thickness=1, circle_radius=1))
34                      mp_drawing.draw_landmarks(image=
    annotated_image,landmark_list=landmark_pb2.
    NormalizedLandmarkList(landmark = landmark_list[2]),
    landmark_drawing_spec=mp_drawing.DrawingSpec(color=ld.
    YELLOW_COLOR, thickness=1, circle_radius=1))
35
36                      startLE,stopLE = ft.getMaxMinXYValues(dict,"
    leftEye",image)
37                      startRE,stopRE = ft.getMaxMinXYValues(dict,"
    rightEye",image)
38
39                      eyesDict = {}
40
41                      cv2.putText(annotated_image, 't: '+str(
    timestamp), (0,20), cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255))
42                      cv2.rectangle(annotated_image,startLE,stopLE,
    ld.RED_COLOR,1)
43                      cv2.rectangle(annotated_image,startRE,stopRE,
    ld.YELLOW_COLOR,1)
44                      cv2.imshow("eyeMeshing", annotated_image)
45
46                      if(bool(redDict)):
47                          eyesDict["timestamp"] = timestamp
48                          eyesDict["leftEye"] = {'xMin':startLE[0],'
    yMin':startLE[1],'xMax':stopLE[0],'yMax':stopLE[1]}
49                          eyesDict["rightEye"] = {'xMin':startRE[0],
    'yMin':startRE[1],'xMax':stopRE[0],'yMax':stopRE[1]}
50                          bboxes.append(eyesDict)
51                          redDict["timestamp"] = timestamp
52                          list.append(redDict)
53                  cv2.waitKey(1)
54
55 ft.writeJson(list,"dvSaveEyesFrames")
56 ft.writeJson(bboxes,"eyesBoundingBoxes")
57
58 list, dict, count, idx = [], {}, 0, 0
59 bool2 = False
60
61 with AedatFile(ft.genPath+"dvSave.aedat4") as f:
62     events = np.hstack([packet for packet in f['events'].numpy()])
63     for event in events:
64         timestamp = (event["timestamp"]/1e6)-baseDate
65         idx, bool1 = ft.filterEventTimestamps(timestamp,
    reducedTimestamps,idx)
66         if bool1:
```

```
67              string,bool2 = ft.filterEventCoords(event["x"],event["
    y"],startLE,stopLE,startRE,stopRE)
68          if bool1 and bool2:
69              dict["timestamp"], dict["x"], dict["y"], dict["
    polarity"],dict["type"]  = timestamp,event["x"],event["y"],
    event["polarity"],string
70              list.append(dict)
71              dict = {}
72
73 ft.writeJson(list,"dvSaveEyesEvents",cls=ft.NumpyEncoder)
```

Listing 4.5: eyeMeshing.py

Quello che viene visualizzato in debug è la Figura 4.8, in cui vengono soltanto evidenziati i landmarks degli occhi nei vari timestamp, e come questi landmarks si muovono coi movimenti del volto:

Figura 4.8

Dopodiché si va a vedere un'animazione dell'andamento nel tempo degli eventi contenuti nel file `"dvSaveEyesEvents.json"`, mentre successivamente viene mostrato l'andamento complessivo delle polarità relative agli stessi eventi. Per fare ciò si crea il programma denominato `"eyeProcessing.py"`.

```
1 import utilities.functions as ft
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 bBoxes = []
6 with open(ft.jsonPath+"eyesBoundingBoxes.json", 'r') as file:
7     bBoxes = ft.load(file)
8 maxLE,maxRE = ft.getMaxMinBoundingBoxes(bBoxes)
9
10 stLE,timestampLE,stRE,timestampRE = [],[],[],[]
11 xDim,yDim = 346, 260
12 mat = np.zeros((yDim,xDim))
13 index, acc, sumLE, sumRE = 0, 0, 0, 0
```

```
step, initStep, valueStep = 0, 1, 0.001
with open(ft.jsonPath+"dvSaveEyesEvents.json", 'r') as file:
    data = ft.load(file)
    while index < len(data):
        acc,idx = 0, 0
        step = valueStep
        action = ""
        if index == 0:
            step = initStep
        while (index+acc < len(data)) and (data[index+acc]["
    timestamp"]<(data[index]["timestamp"]+step)):
            event = data[index+acc]
            if event["polarity"] == 1:
                mat[event["y"],event["x"]] += 1
            else:
                mat[event["y"],event["x"]] -= 1

            string = ft.recognizeEyesBoxes(event["x"],event["y"],
    maxLE,maxRE)
            if string == "leftEye":
                sumLE = int(np.sum(mat[maxLE[0][1]:maxLE[1][1],
    maxLE[0][0]:maxLE[1][0]]))
                stLE.append(sumLE)
                timestampLE.append(event["timestamp"])
            elif string == "rightEye":
                sumRE = int(np.sum(mat[maxRE[0][1]:maxRE[1][1],
    maxRE[0][0]:maxRE[1][0]]))
                stRE.append(sumRE)
                timestampRE.append(event["timestamp"])
            acc += 1
        plt.xlim(125,200)
        plt.ylim(130,100)
        plt.title("Eyes Events")
        plt.xlabel('\nrightEyeSum: '+str(sumRE)+', leftEyeSum: '+
    str(sumLE)+'\nt: '+str(event["timestamp"]))
        plt.matshow(mat,fignum=False,interpolation='nearest',cmap=
    'plasma')
        plt.pause(1)
        index = index+acc+1
plt.close()

ft.printSumEyes(stLE,stRE,timestampLE,timestampRE,initStep,
    valueStep)
```

Listing 4.6: eyeProcessing.py

Quando si avvia il programma viene visualizzata una finestra dove ci sono gli eventi per le diverse finestre di accumulazione nei vari `valueStep` come si vede nella Figura 4.9; mentre andamento complessivo delle polarità relative agli stessi eventi viene mostrato nella Figura 4.10 con finestra d'accumulazione 0.001

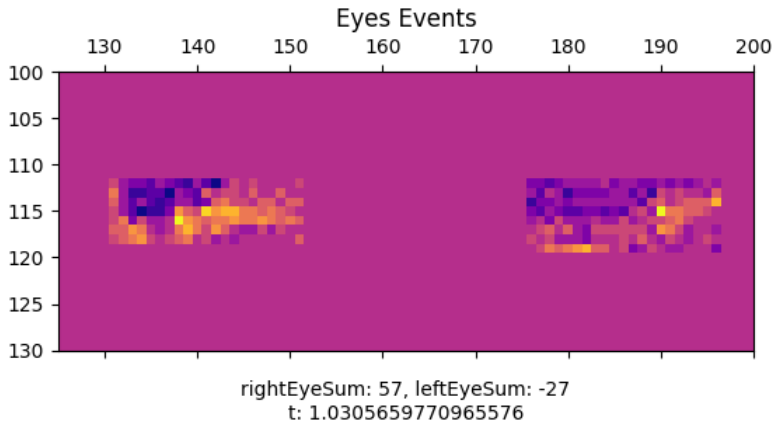


Figura 4.9: Eyes Events

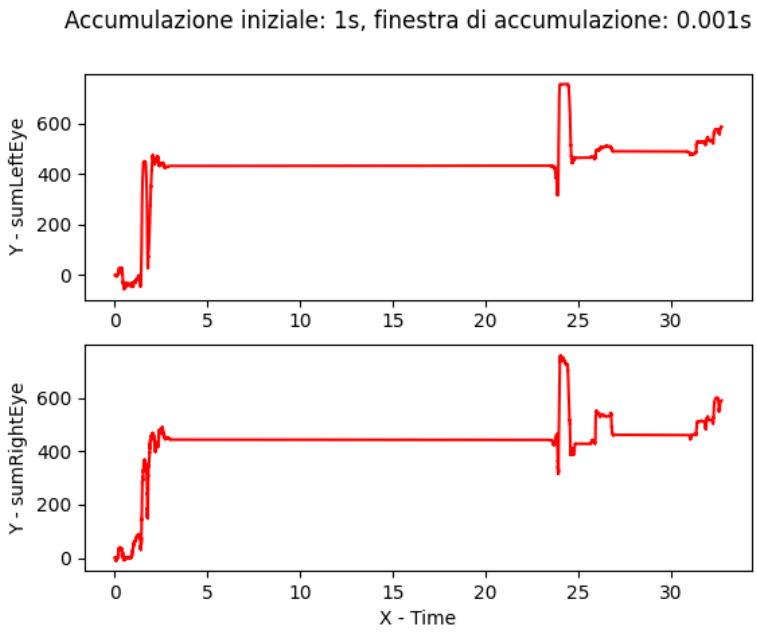


Figura 4.10: Andamento Polarità Occhi

4.4 Lips Meshing e Lips Processing

Per l'analisi delle labbra similmente a quanto fatto per gli occhi è stato sviluppato una serie di funzioni dedicate andando a filtrare per timestamp *reducedTimestamps = ft.getReducedTimestamps("...")* che si trova in `"eyeMeshing.py"` (Codice 4.5). Quindi si procede con lo script `"lipsMeshing.py"` il quale è diviso in due parti: la prima è una versione ridotta di `"faceMeshing.py"` (Codice 4.1), che mostra a video il volto circondato da un bounding boxes tempo variante, che delimita l'area di appartenenza degli eventi successivamente catturati.
I dati delle labbra vengono salvate nel file `"dvSaveLipsFrames.json"`, mentre le

altre vengono salvate nel file `"lipsBoundingBoxes.json"`.
La seconda parte analizza tutti gli eventi appartenenti agli intervalli delineati nel file `"analyzeLips.json"`, se tali eventi appartengono ai bounding boxes dei due occhi, allora vengono salvati nel file `"dvSaveLipsEvents.json"`.

```
import cv2
import utilities.landmarks as ld
import utilities.functions as ft
import mediapipe as mp
import numpy as np
from mediapipe.framework.formats import landmark_pb2
from dv import AedatFile

mp_drawing = mp.solutions.drawing_utils
mp_face_mesh = mp.solutions.face_mesh

image = 0
list,redDdict,count,baseDate = [], {}, 0, 0

bboxes,totDict = [],{}

with AedatFile(ft.genPath+"dvSave.aedat4") as recording:
    with mp_face_mesh.FaceMesh(min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as face_mesh:
        for frame in recording["frames"]:
            if count == 0:
                baseDate = frame.timestamp/1e6
                count = 1
            timestamp = (frame.timestamp/1e6)-baseDate
            image = cv2.cvtColor(frame.image, cv2.COLOR_GRAY2RGB)
            annotated_image = image.copy()
            results = face_mesh.process(image)
            if results.multi_face_landmarks:
                for face_landmarks in results.multi_face_landmarks
    :
                    dict,landmark_list = ft.calculateKeypoints(
    face_landmarks)
                    redDict = ft.getLipsValues(face_landmarks)

                    mp_drawing.draw_landmarks(image=
    annotated_image,landmark_list=landmark_pb2.
    NormalizedLandmarkList(landmark = landmark_list[1]),
    landmark_drawing_spec=mp_drawing.DrawingSpec(color=ld.RED_COLOR
    , thickness=1, circle_radius=1))

                    startLI,stopLI = ft.getLipsMaxMinXYValues(dict
    ,"lips",image)

                    totDict = {}

                    cv2.putText(annotated_image, 't: '+str(
    timestamp), (0,20), cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255))
                    cv2.rectangle(annotated_image,startLI,stopLI,
```

```
    ld.RED_COLOR,1)
40                      cv2.imshow("totMeshing", annotated_image)
41
42                      if(bool(redDict)):
43                          totDict["timestamp"] = timestamp
44                          totDict["lips"] = {'xMin':startLI[0],'yMin
    ':startLI[1],'xMax':stopLI[0],'yMax':stopLI[1]}
45                          bboxes.append(totDict)
46                          redDict["timestamp"] = timestamp
47                          list.append(redDict)
48                  cv2.waitKey(1)
49
50 ft.writeJson(list,"dvSaveLipsFrames")
51 ft.writeJson(bboxes,"lipsBoundingBoxes")
52
53 list, dict, count, idx = [], {}, 0, 0
54 bool2 = False
55
56 with AedatFile(ft.genPath+"dvSave.aedat4") as f:
57     events = np.hstack([packet for packet in f['events'].numpy()])
58     for event in events:
59         timestamp = (event["timestamp"]/1e6)-baseDate
60         string,bool2 = ft.filterLipsEventCoords(event["x"],event["
    y"],startLI,stopLI)
61         if bool2:
62             dict["timestamp"], dict["x"], dict["y"], dict["
    polarity"],dict["type"]  = timestamp,event["x"],event["y"],
    event["polarity"],string
63             list.append(dict)
64             dict = {}
65
66 ft.writeJson(list,"dvSaveLipsEvents",cls=ft.NumpyEncoder)
```

Listing 4.7: lipsMeshing.py

Quello che viene visualizzato nel debug è la Figura 4.11, in cui vengono soltanto evidenziati i landmarks delle labbra nei vari timestamp, e come questi landmarks si muovono coi movimenti del volto:

Successivamente è stato realizzato uno script che mostra un'animazione dell'andamento nel tempo degli eventi contenuti nel file `"dvSaveLipsEvents.json"`, mentre successivamente viene mostrato l'andamento complessivo delle polarità relative agli stessi eventi.

```
1 import utilities.functions as ft
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 bBoxes = []
6 with open(ft.jsonPath+"lipsBoundingBoxes.json", 'r') as file:
7     bBoxes = ft.load(file)
8 maxLI = ft.getLipsMaxMinBoundingBoxes(bBoxes)
9 stLI,timestampLI = [],[]
10 xDim,yDim = 346, 260
```

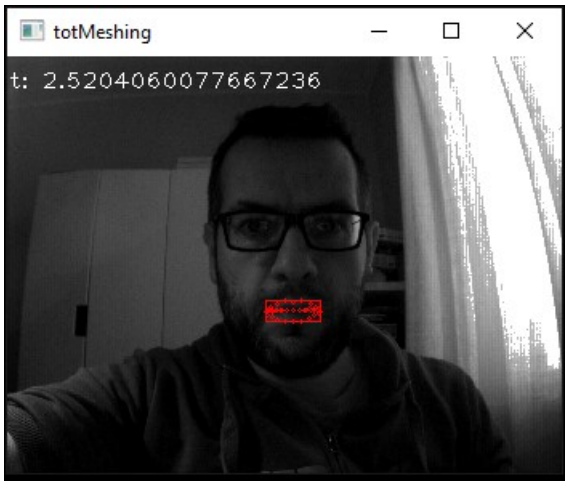


Figura 4.11: Lips Meshing

```
mat = np.zeros((yDim,xDim))
index, acc, sumLI = 0, 0, 0
step, initStep, valueStep = 0, 1, 0.5
with open(ft.jsonPath+"dvSaveLipsEvents.json", 'r') as file:
    data = ft.load(file)
    while index < len(data):
        acc,idx = 0, 0
        step = valueStep
        mat = np.zeros((yDim,xDim))
        action = ""
        if index == 0:
            step = initStep
        while (index+acc < len(data)) and (data[index+acc]["
timestamp"]<(data[index]["timestamp"]+step)):
            event = data[index+acc]
            if event["polarity"] == 1:
                mat[event["y"],event["x"]] += 1
            else:
                mat[event["y"],event["x"]] -= 1

            string = ft.recognizeLipsBoxes(event["x"],event["y"],
maxLI)
            if string == "lips":
                sumLI = int(np.sum(mat[maxLI[0][1]:maxLI[1][1],
maxLI[0][0]:maxLI[1][0]]))
                stLI.append(sumLI)
                timestampLI.append(event["timestamp"])
            acc += 1

        plt.xlim(125,200)
        plt.ylim(180,155)
        plt.title("Events")
        plt.xlabel('\nlipsSum: '+str(sumLI)+'\nt: '+str(event["
timestamp"]))
        plt.matshow(mat,fignum=False,interpolation='nearest',cmap=
```

```
    'plasma')
42          plt.pause(1)
43
44          index = index+acc+1
45  plt.close()
46
47  #il primo intervallo e un campione random
48  ft.printLipsTot(stLI,timestampLI,initStep,valueStep)
```

Listing 4.8: lipsProcessing.py

Quando si avvia il programma viene visulaizzata una finestra dove ci sono gli eventi per le diverse finestre di accumulazione nei vari `valueStep` come si vede nella Figura 4.12; mentre andamento complessivo delle polarità relative agli stessi eventi viene mostrato nella Figura 4.13 con finestra d'accumulazione 0.5

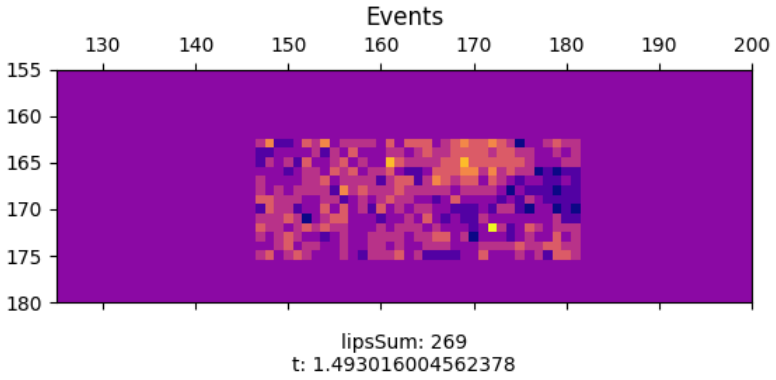


Figura 4.12: Lips Events

4.5 Total Meshing e Total Processing

Infine si effettua lo stesso processamento dove si analizzano sia occhi che le labbra. Si è sviluppato lo script `"totMeshing.py"` che è diviso in due parti:
la prima è una versione simile a `"faceMeshing.py"` (Codice 4.1), che mostra a video il volto circondato da tre bounding boxes tempo varianti, che delimiteranno l'area di appartenenza degli eventi successivamente catturati. Le informazioni relative agli occhi e le labbra vengono salvate nel file `"dvSaveTotFrames.json"`, mentre quelle relative ai bounding boxes sono salvate in `"totBoundingBoxes.json"`.
La seconda parte analizza tutti gli eventi appartenenti agli intervalli delineati nei file `"analyzeEyes.json"` e `"analyzeLips.json"`, se tali eventi appartengono ai bounding boxes dei due occhi e delle labbra, allora vengono salvati nel file `"dvSaveEyesEvents.json"`.

```
1  import cv2
2  import utilities.landmarks as ld
3  import utilities.functions as ft
```

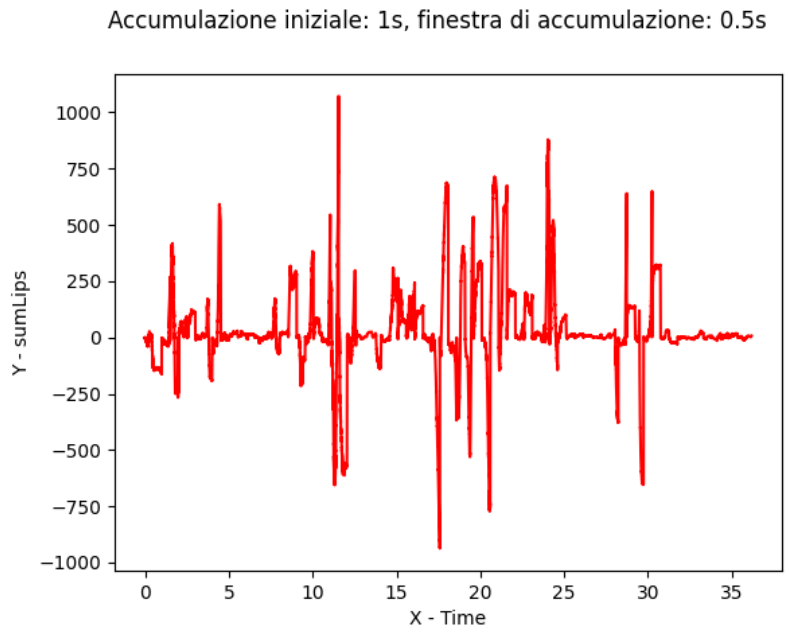


Figura 4.13: Andamento Polarità Labbra

```
import mediapipe as mp
import numpy as np
from mediapipe.framework.formats import landmark_pb2
from dv import AedatFile

mp_drawing = mp.solutions.drawing_utils
mp_face_mesh = mp.solutions.face_mesh

image = 0
list,redDdict,count,baseDate = [], {}, 0, 0

bboxes,totDict = [],{}

with AedatFile(ft.genPath+"dvSave.aedat4") as recording:
    with mp_face_mesh.FaceMesh(min_detection_confidence=0.5,
    min_tracking_confidence=0.5) as face_mesh:
        for frame in recording["frames"]:
            if count == 0:
                baseDate = frame.timestamp/1e6
                count = 1
            timestamp = (frame.timestamp/1e6)-baseDate
            image = cv2.cvtColor(frame.image, cv2.COLOR_GRAY2RGB)
            annotated_image = image.copy()
            results = face_mesh.process(image)
            if results.multi_face_landmarks:
                for face_landmarks in results.multi_face_landmarks
    :
                    dict,landmark_list = ft.calculateKeypoints(
    face_landmarks)
```

```
30                      redDict = ft.getTotValues(face_landmarks)
31
32                      mp_drawing.draw_landmarks(image=
    annotated_image,landmark_list=landmark_pb2.
    NormalizedLandmarkList(landmark = landmark_list[0]),
    landmark_drawing_spec=mp_drawing.DrawingSpec(color=ld.
    BLUE_COLOR, thickness=1, circle_radius=1))
33                      mp_drawing.draw_landmarks(image=
    annotated_image,landmark_list=landmark_pb2.
    NormalizedLandmarkList(landmark = landmark_list[2]),
    landmark_drawing_spec=mp_drawing.DrawingSpec(color=ld.
    YELLOW_COLOR, thickness=1, circle_radius=1))
34                      mp_drawing.draw_landmarks(image=
    annotated_image,landmark_list=landmark_pb2.
    NormalizedLandmarkList(landmark = landmark_list[1]),
    landmark_drawing_spec=mp_drawing.DrawingSpec(color=ld.RED_COLOR
    , thickness=1, circle_radius=1))
35
36
37                      startLE,stopLE = ft.getMaxMinXYValues(dict,"
    leftEye",image)
38                      startRE,stopRE = ft.getMaxMinXYValues(dict,"
    rightEye",image)
39                      startLI,stopLI = ft.getLipsMaxMinXYValues(dict
    ,"lips",image)
40
41                      totDict = {}
42
43                      cv2.putText(annotated_image, 't: '+str(
    timestamp), (0,20), cv2.FONT_HERSHEY_PLAIN, 1, (255, 255, 255))
44                      cv2.rectangle(annotated_image,startLE,stopLE,
    ld.BLUE_COLOR,1)
45                      cv2.rectangle(annotated_image,startRE,stopRE,
    ld.YELLOW_COLOR,1)
46                      cv2.rectangle(annotated_image,startLI,stopLI,
    ld.RED_COLOR,1)
47                      cv2.imshow("totMeshing", annotated_image)
48
49                      if(bool(redDict)):
50                          totDict["timestamp"] = timestamp
51                          totDict["leftEye"] = {'xMin':startLE[0],'
    yMin':startLE[1],'xMax':stopLE[0],'yMax':stopLE[1]}
52                          totDict["rightEye"] = {'xMin':startRE[0],'
    yMin':startRE[1],'xMax':stopRE[0],'yMax':stopRE[1]}
53                          totDict["lips"] = {'xMin':startLI[0],'yMin
    ':startLI[1],'xMax':stopLI[0],'yMax':stopLI[1]}
54                          bboxes.append(totDict)
55                          redDict["timestamp"] = timestamp
56                          list.append(redDict)
57                  cv2.waitKey(1)
58
59 ft.writeJson(list,"dvSaveTotFrames")
60 ft.writeJson(bboxes,"totBoundingBoxes")
```

```
61
62 list, dict, count, idx = [], {}, 0, 0
63 bool2 = False
64
65 with AedatFile(ft.genPath+"dvSave.aedat4") as f:
66     events = np.hstack([packet for packet in f['events'].numpy()])
67     for event in events:
68         timestamp = (event["timestamp"]/1e6)-baseDate
69         string,bool2 = ft.filterTotEventCoords(event["x"],event["y
    "],startLE,stopLE,startRE,stopRE,startLI,stopLI)
70         if bool2:
71             dict["timestamp"], dict["x"], dict["y"], dict["
    polarity"],dict["type"]  = timestamp,event["x"],event["y"],
    event["polarity"],string
72             list.append(dict)
73             dict = {}
74
75 ft.writeJson(list,"dvSaveTotEvents",cls=ft.NumpyEncoder)
```

Listing 4.9: totMeshing.py

Quello che viene visualizzato nel debug è la Figura 4.14, in cui vengono evidenziati i landmarks delle labbra e degli occhi in tutti i timestamp, e come questi landmarks si muovono al variare della posizione del volto:

Figura 4.14: Total Meshing

Successivamente è stato sviluppato uno script per mostrare un'animazione dell'andamento nel tempo degli eventi contenuti nel file `"dvSaveTotEvents.json"`, mentre successivamente viene mostrato l'andamento complessivo delle polarità relative agli stessi eventi. Inoltre si procede con la visualizzazione ogni finestra di accumulazione come cambiano le polarità ed in particolare sono state utilizzate le seguenti `finestre di accumulazione`: `1, 2, 5, 10, 25, 50, 100, 250, 500, 1000 ms`.

```
1 import utilities.functions as ft
```

```
import numpy as np
import matplotlib.pyplot as plt

bBoxes = []
with open(ft.jsonPath+"totBoundingBoxes.json", 'r') as file:
    bBoxes = ft.load(file)
maxLE,maxRE,maxLI = ft.getTotMaxMinBoundingBoxes(bBoxes)
stLE,timestampLE,stRE,timestampRE,stLI,timestampLI =
    [],[],[],[],[],[]
xDim,yDim = 346, 260
mat = np.zeros((yDim,xDim))
index, acc, sumLE, sumRE, sumLI = 0, 0, 0, 0, 0
step, initStep, valueStep = 0, 1, 1
with open(ft.jsonPath+"dvSaveTotEvents.json", 'r') as file:
    data = ft.load(file)
    while index < len(data):
        acc,idx = 0, 0
        step = valueStep
        mat = np.zeros((yDim,xDim))
        action = ""
        if index == 0:
            step = initStep
        while (index+acc < len(data)) and (data[index+acc]["
    timestamp"]<(data[index]["timestamp"]+step)):
            event = data[index+acc]
            if event["polarity"] == 1:
                mat[event["y"],event["x"]] += 1
            else:
                mat[event["y"],event["x"]] -= 1

            string0 = ft.recognizeLEBoxes(event["x"],event["y"],
    maxLE)
            string1 = ft.recognizeREBoxes(event["x"],event["y"],
    maxRE)
            string2 = ft.recognizeLipsBoxes(event["x"],event["y"],
    maxLI)
            if string0 == "leftEye":
                sumLE = int(np.sum(mat[maxLE[0][1]:maxLE[1][1],
    maxLE[0][0]:maxLE[1][0]]))
                stLE.append(sumLE)
                timestampLE.append(event["timestamp"])
            if string1 == "rightEye":
                sumRE = int(np.sum(mat[maxRE[0][1]:maxRE[1][1],
    maxRE[0][0]:maxRE[1][0]]))
                stRE.append(sumRE)
                timestampRE.append(event["timestamp"])
            if string2 == "lips":
                sumLI = int(np.sum(mat[maxLI[0][1]:maxLI[1][1],
    maxLI[0][0]:maxLI[1][0]]))
                stLI.append(sumLI)
                timestampLI.append(event["timestamp"])
            acc += 1

```

```
47          plt.xlim(125,200)
48          plt.ylim(180,100)
49          plt.title("Events")
50          plt.xlabel('\nrightEyeSum: '+str(sumRE)+', leftEyeSum: '+
    str(sumLE)+', lipsSum: '+str(sumLI)+'\nt: '+str(event["
    timestamp"]))
51          plt.matshow(mat,fignum=False,interpolation='nearest',cmap=
    'plasma')
52          plt.pause(1)
53          index = index+acc+1
54  plt.close()
55  ft.printSumTot(stLE,stRE,stLI,timestampLE,timestampRE,timestampLI,
    initStep,valueStep)
```

Listing 4.10: totProcessing.py

Quando si avvia il programma viene visualizzata una finestra dove ci sono gli eventi per le diverse finestre di accumulazione nei vari `valueStep` come si vede nella Figura 4.15.

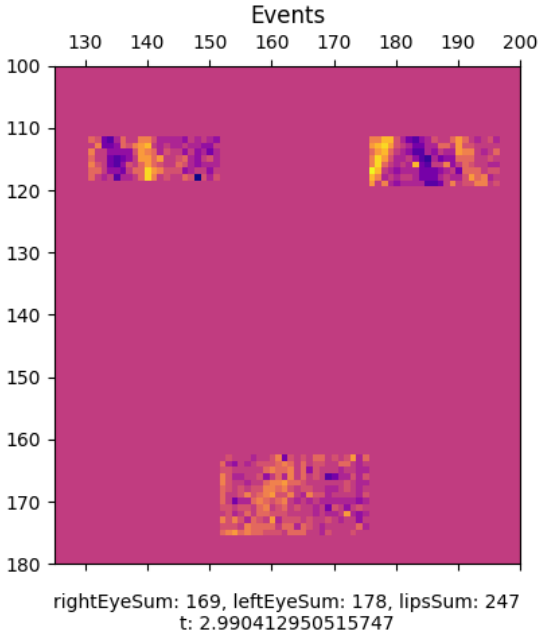


Figura 4.15: TotalEvents

Capitolo 5

Risultati

Vediamo ora i risultati ottenuti i vari script di con i diversi valori di integrazione ovvero `1, 2, 5, 10, 25, 50, 100, 250, 500, 1000 ms`. I grafici dell'andamento delle polarità relative ai vari eventi sono i seguenti:

5.1 Grafici Eye Processing

Di seguito sono riportati i grafici di `"eyeProcessing.py"` nei timestamp di apertura e di chiusura degli occhi. Si usa la Funzione 5.1 che è una porzione di codice di `"functions.py"`; viene effettuato il plot generando un grafico che mostra le polarità degli eventi. Questa funzione viene richiamata nel Programma `"eyeProcessing.py"`.

```
1 def printSumEyes(mat1,mat2,time1,time2,init,delta):
2     fig, (ax1,ax2) = plt.subplots(2,1)
3     fig.suptitle('Accumulazione iniziale: '+str(init)+'s, finestra
      di accumulazione: '+str(delta)+'s')
4     ax1.set_ylabel('Y - sumLeftEye')
5     ax1.plot(np.array(time1),np.array(mat1),color = 'r')
6     ax2.set_ylabel('Y - sumRightEye')
7     ax2.set_xlabel('X - Time')
8     ax2.plot(np.array(time2),np.array(mat2),color = 'r')
9     plt.show()
```

Listing 5.1: functions.py: printSumEyes()

5.1.1 Andamento con Finestra di Accumulazione a 1ms

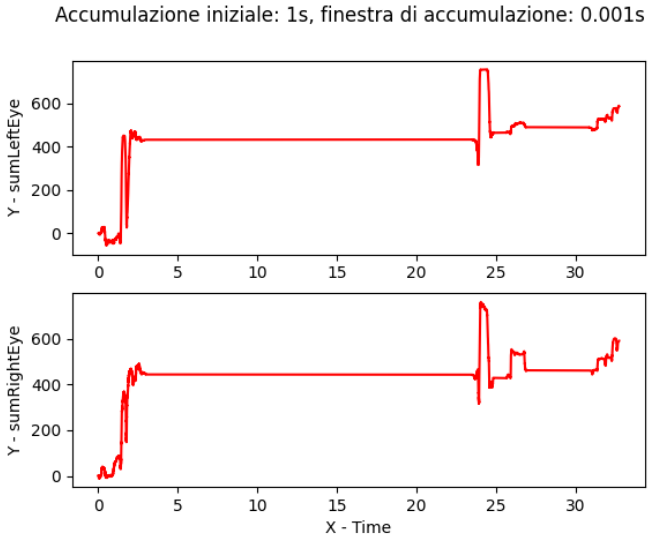


Figura 5.1: Grafico Eyes 1ms

5.1.2 Andamento con Finestra di Accumulazione a 2ms

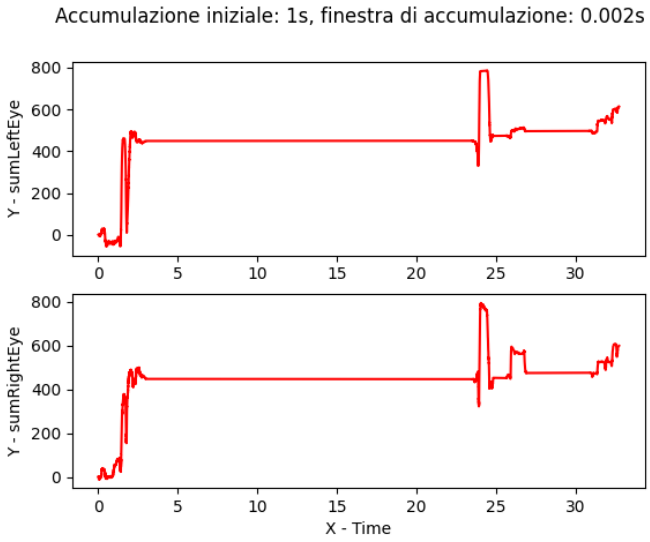


Figura 5.2: Grafico Eyes 2ms

5.1.3 Andamento con Finestra di Accumulazione a 5ms

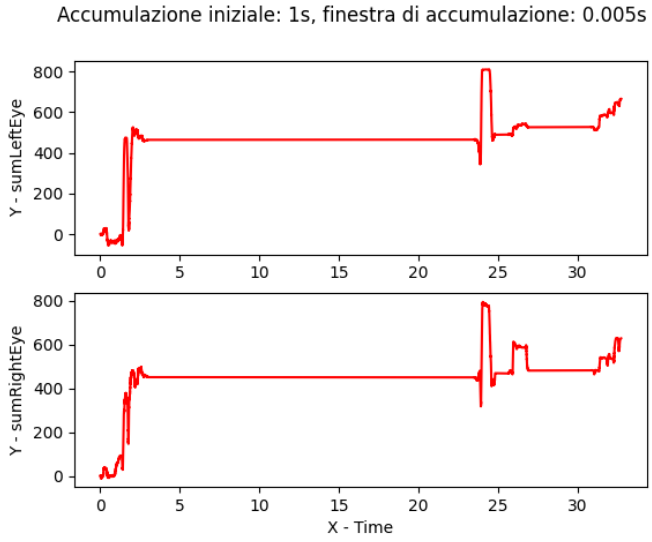


Figura 5.3: Grafico Eyes 5ms

5.1.4 Andamento con Finestra di Accumulazione a 10ms

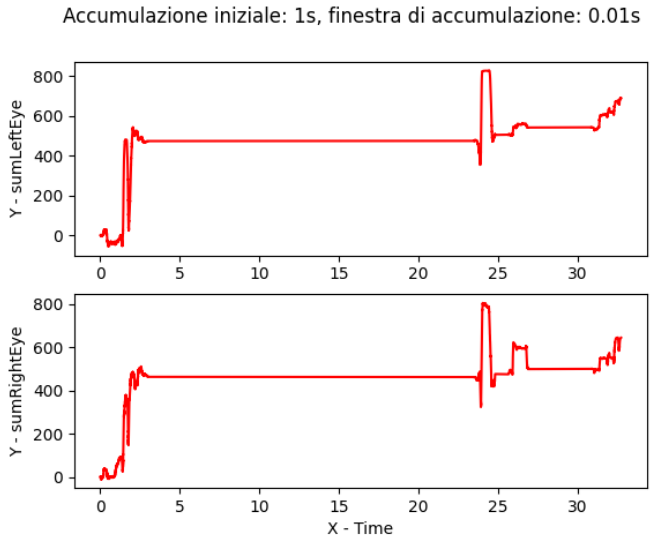


Figura 5.4: Grafico Eyes 10ms

5.1.5 Andamento con Finestra di Accumulazione a 25ms

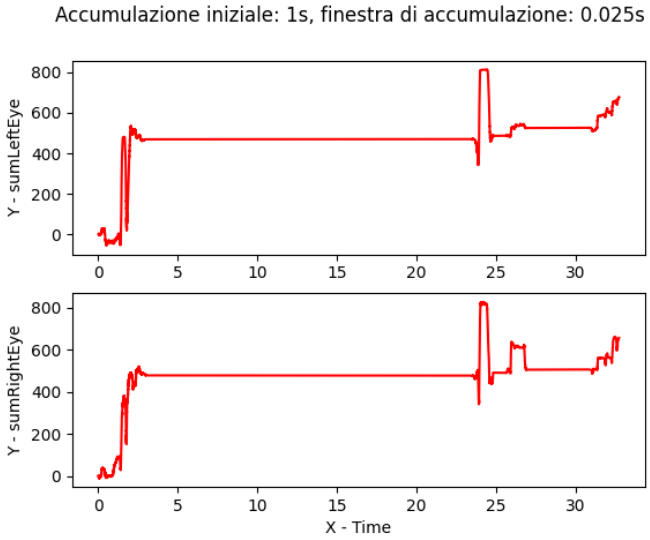


Figura 5.5: Grafico Eyes 25ms

5.1.6 Andamento con Finestra di Accumulazione a 50ms

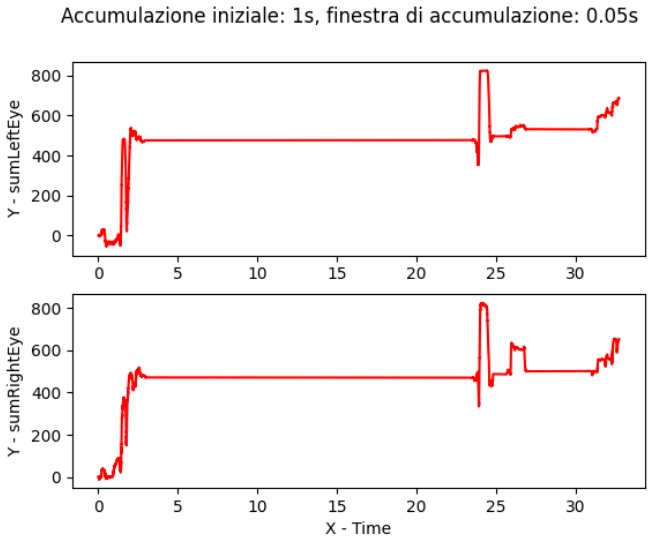


Figura 5.6: Grafico Eyes 50ms

5.1.7 Andamento con Finestra di Accumulazione a 100ms

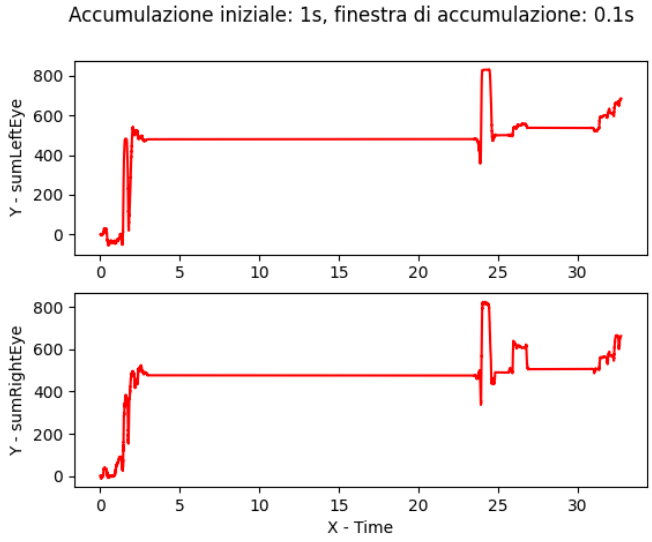


Figura 5.7: Grafico Eyes 100ms

5.1.8 Andamento con Finestra di Accumulazione a 250ms

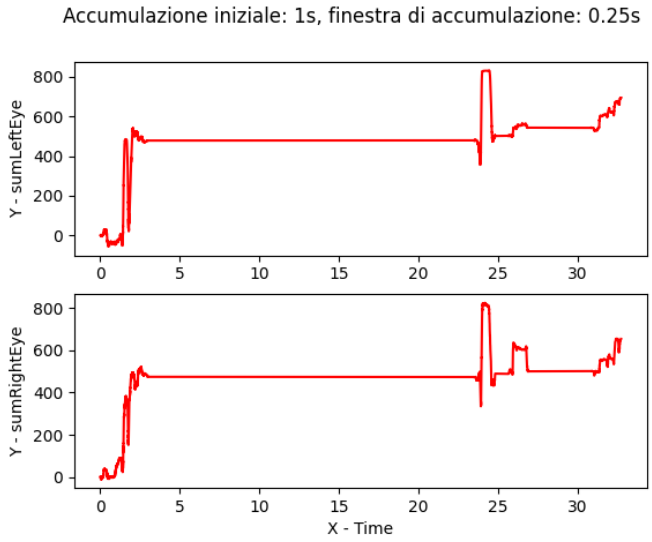


Figura 5.8: Grafico Eyes 250ms

5.1.9 Andamento con Finestra di Accumulazione a 500ms

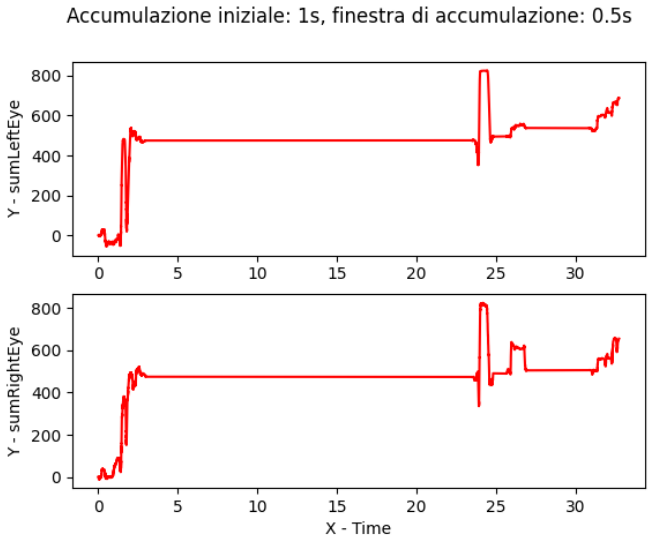


Figura 5.9: Grafico Eyes 500ms

5.1.10 Andamento con Finestra di Accumulazione a 1000ms

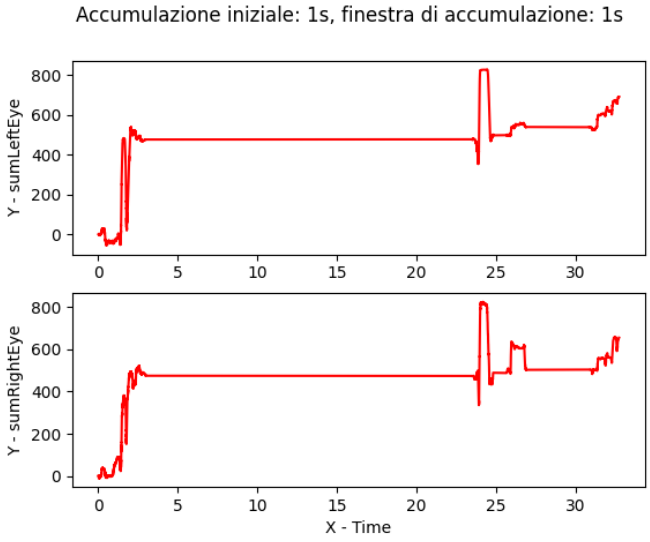


Figura 5.10: Grafico Eyes 1000ms

5.2 Grafici Lips Processing

Di seguito sono riportati i grafici di `"lipsProcessing.py"` lungo l'intero video; con il marker verde è segnato il timestamp in cui avviene l'apertura della bocca. Si usa la Funzione 5.2 che è una porzione di codice di `"functions.py"`; viene fatto

il plot con cui si genera un grafico che visualizza le polarità degli eventi. Questa funzione viene richiamata nello script `"lipsProcessing.py"`.

```
def printSumLips(mat3,time3,init,delta):
    x2 = np.array([4.120306015014648, 28.399940967559814,
    29.83999490737915])
    #y2 = np.array([-20, -20, -20])
    #y2 = np.array([-50, -50, -50])
    #y2 = np.array([-200, -200, -200])
    y2 = np.array([-400, -400, -400])
    fig, (ax3) = plt.subplots(1,1)
    fig.suptitle('Accumulazione iniziale: '+str(init)+'s, finestra
    di accumulazione: '+str(delta)+'s')
    ax3.set_ylabel('Y - sumLips')
    ax3.plot(np.array(time3),np.array(mat3),color = 'r')
    ax3.plot(x2,y2,color = 'g', marker='o', linestyle = 'None')
    ax3.set_xlabel('X - Time')
    plt.show()
```

Listing 5.2: functions.py: printSumLips()

5.2.1 Andamento con Finestra di Accumulazione a 1ms

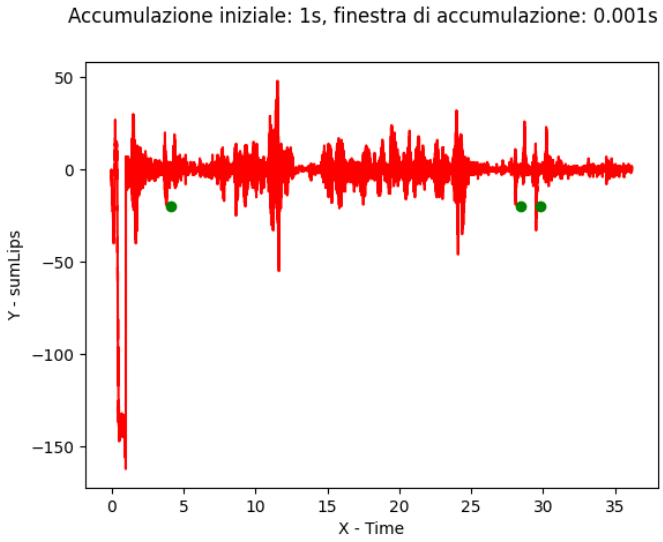


Figura 5.11: Grafico Lips 1ms

5.2.2 Andamento con Finestra di Accumulazione a 2ms

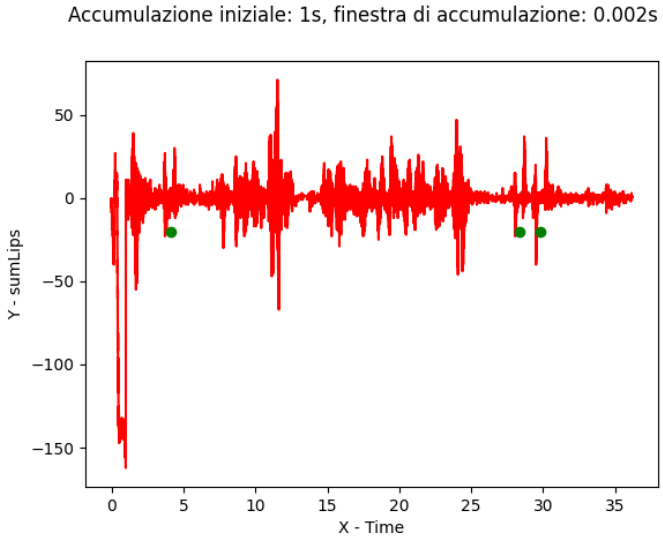


Figura 5.12: Grafico Lips 2ms

5.2.3 Andamento con Finestra di Accumulazione a 5ms

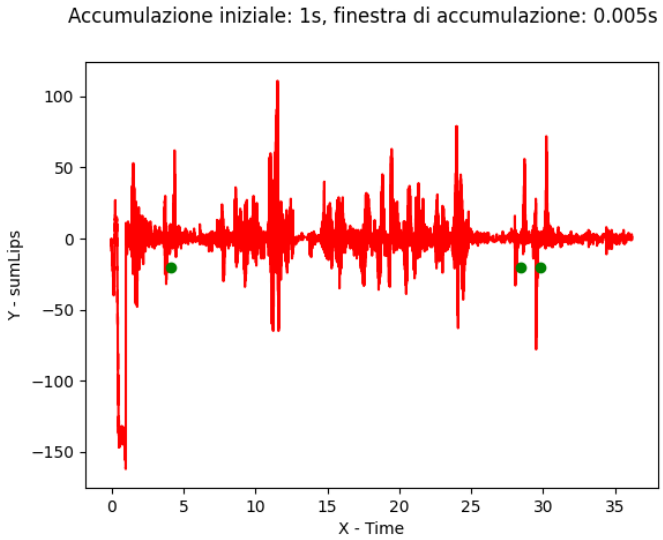


Figura 5.13: Grafico Lips 5ms

5.2.4 Andamento con Finestra di Accumulazione a 10ms

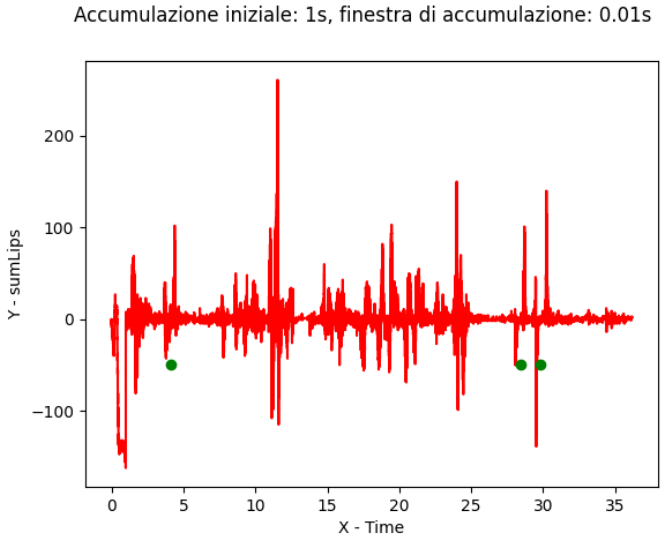


Figura 5.14: Grafico Lips 10ms

5.2.5 Andamento con Finestra di Accumulazione a 25ms

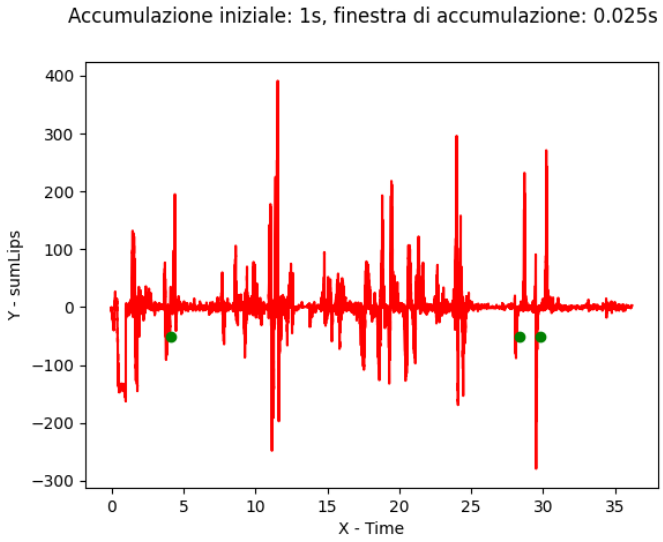


Figura 5.15: Grafico Lips 25ms

5.2.6 Andamento con Finestra di Accumulazione a 50ms

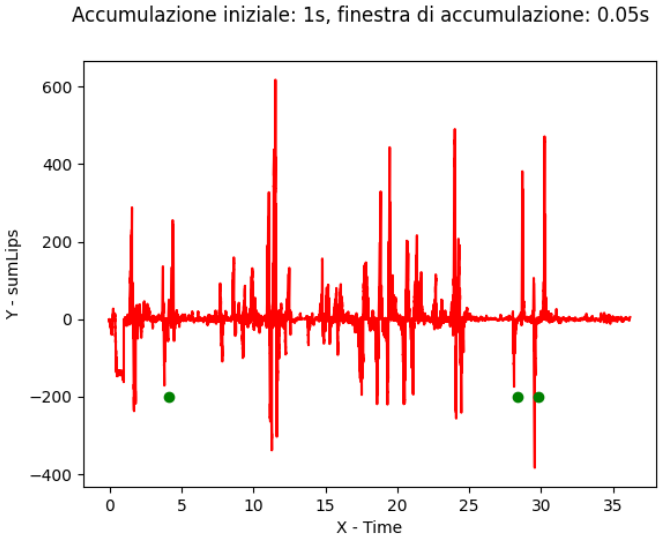


Figura 5.16: Grafico Lips 50ms

5.2.7 Andamento con Finestra di Accumulazione a 100ms

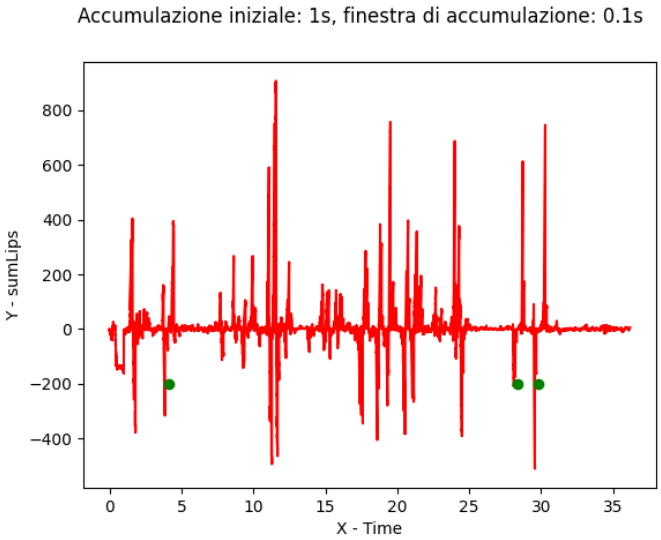


Figura 5.17: Grafico Lips 100ms

5.2.8 Andamento con Finestra di Accumulazione a 250ms

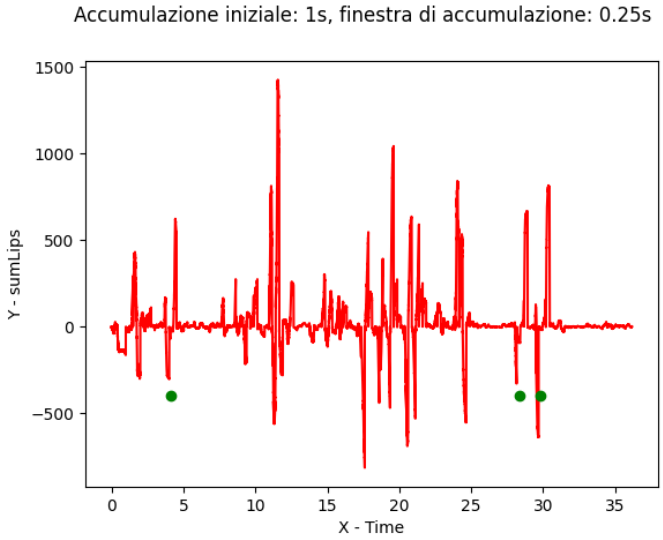


Figura 5.18: Grafico Lips 250ms

5.2.9 Andamento con Finestra di Accumulazione a 500ms

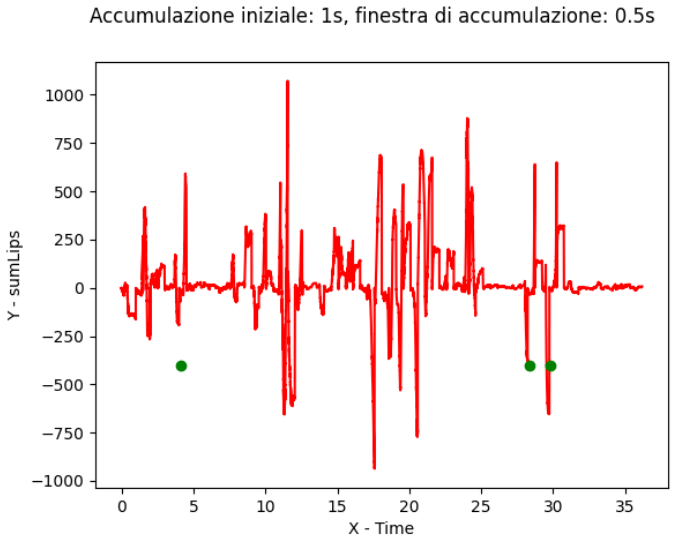


Figura 5.19: Grafico Lips 500ms

5.2.10 Andamento con Finestra di Accumulazione a 1000ms

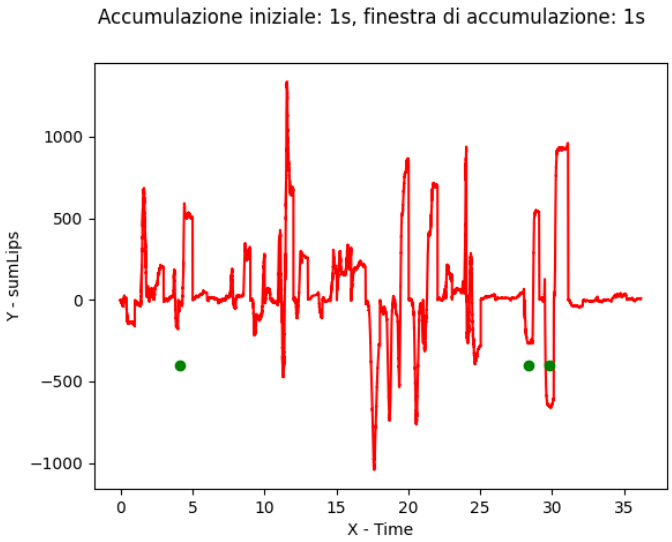


Figura 5.20: Grafico Lips 1000ms

5.3 Grafici Total Processing

Di seguito vi sono i grafici di `"totProcessing.py"` lungo l'intero video; con il marker verde è segnato il timestamp in cui avviene l'apertura della bocca e la chiusura degli occhi. Si usa la Funzione 5.3 che è una porzione di codice di `"functions.py"` viene effettuato il plot con cui si genera un grafico che consente di visualizzare le polarità degli eventi. Questa funzione viene richiamata nel Programma `"totProcessing.py"`.

```
1  def printSumTot(mat1,mat2,mat3,time1,time2,time3,init,delta):
2      x1 = np.array([23.72014594078064, 25.96000099182129,
       31.119995832443237, 32.35993695259094])
3      x2 = np.array([4.120306015014648, 28.399940967559814,
       29.83999490737915])
4      #y1 = np.array([20, 20, 20, 20])
5      #y1 = np.array([50, 50, 50, 50])
6      #y1 = np.array([100, 100, 100, 100])
7      #y1 = np.array([200, 200, 200, 200])
8      y1 = np.array([300, 300, 300, 300])
9      #y2 = np.array([-20, -20, -20])
10     #y2 = np.array([-50, -50, -50])
11     #y2 = np.array([-100, -100, -100])
12     #y2 = np.array([-200, -200, -200])
13     y2 = np.array([-300, -300, -300])
14     fig, (ax1,ax2,ax3) = plt.subplots(3,1)
15     fig.suptitle('Accumulazione iniziale: '+str(init)+'s, finestra
       di accumulazione: '+str(delta)+'s')
16     ax1.set_ylabel('Y - sumLeftEye')
```

```
17      ax1.plot(np.array(time1),np.array(mat1),color = 'r')
18      ax1.plot(x1,y1,color = 'g', marker='o', linestyle = 'None')
19      ax2.set_ylabel('Y - sumRightEye')
20      ax2.plot(np.array(time2),np.array(mat2),color = 'r')
21      ax2.plot(x1,y1,color = 'g', marker='o', linestyle = 'None')
22      ax3.set_ylabel('Y - sumLips')
23      ax3.plot(np.array(time3),np.array(mat3),color = 'r')
24      ax3.plot(x2,y2,color = 'g', marker='o', linestyle = 'None')
25      ax3.set_xlabel('X - Time')
26      plt.show()
```

Listing 5.3: functions.py: printSumTot()

5.3.1 Andamento con Finestra di Accumulazione a 1ms

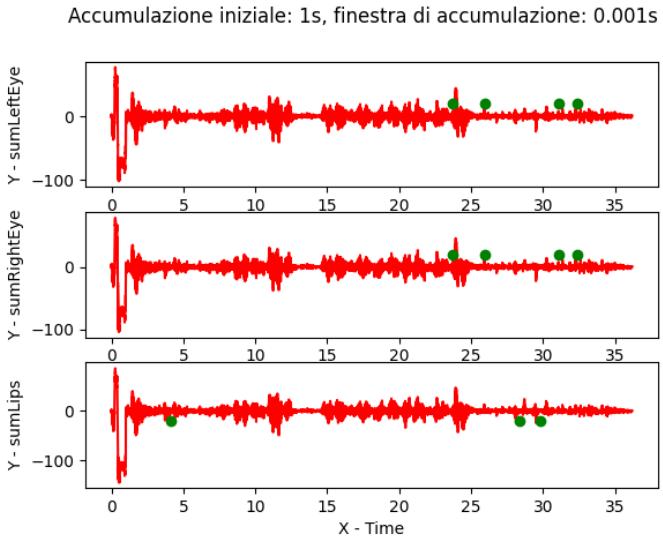


Figura 5.21: Grafico Tot 1ms

5.3.2 Andamento con Finestra di Accumulazione a 2ms

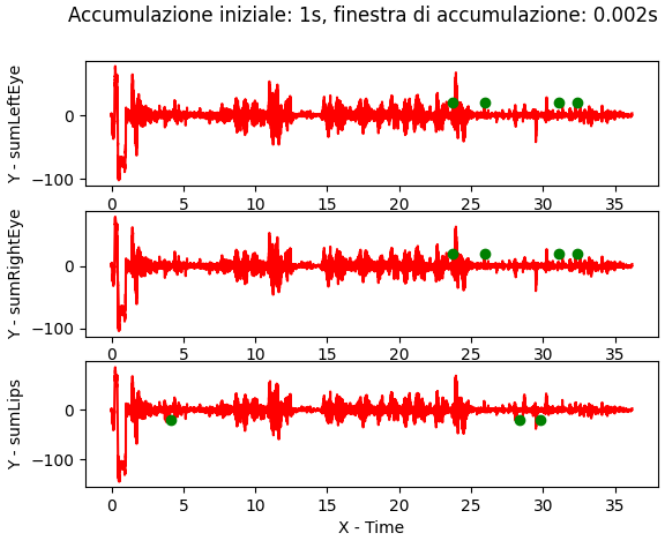


Figura 5.22: Grafico Tot 2ms

5.3.3 Andamento con Finestra di Accumulazione a 5ms

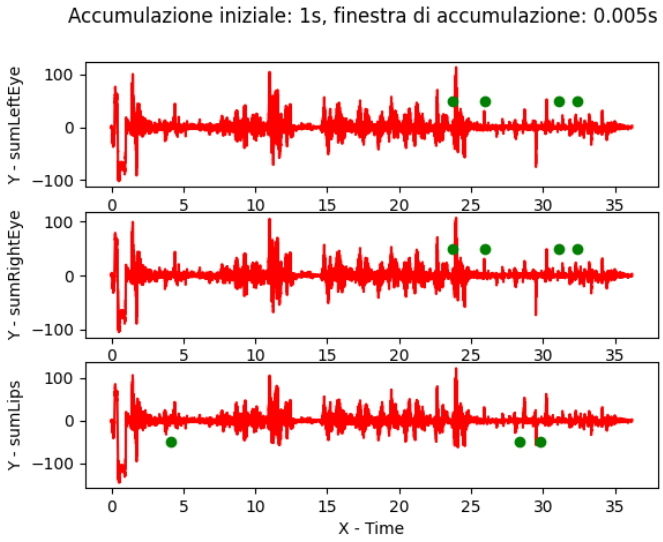


Figura 5.23: Grafico Tot 5ms

5.3.4 Andamento con Finestra di Accumulazione a 10ms

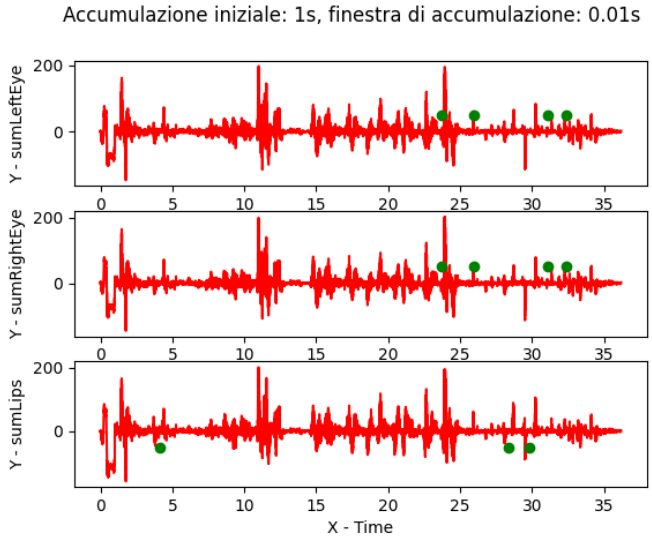


Figura 5.24: Grafico Tot 10ms

5.3.5 Andamento con Finestra di Accumulazione a 25ms

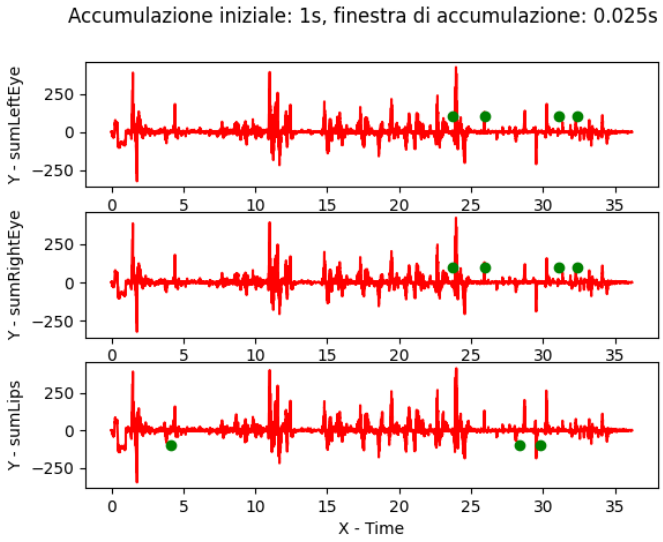


Figura 5.25: Grafico Tot 25ms

5.3.6 Andamento con Finestra di Accumulazione a 50ms

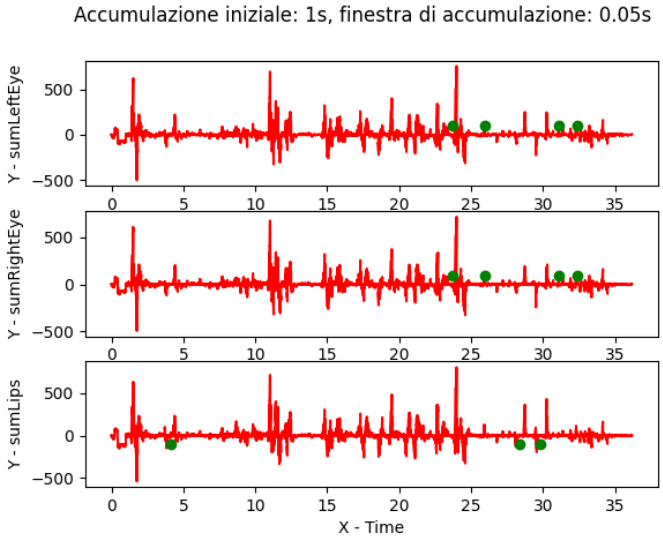


Figura 5.26: Grafico Tot 50ms

5.3.7 Andamento con Finestra di Accumulazione a 100ms

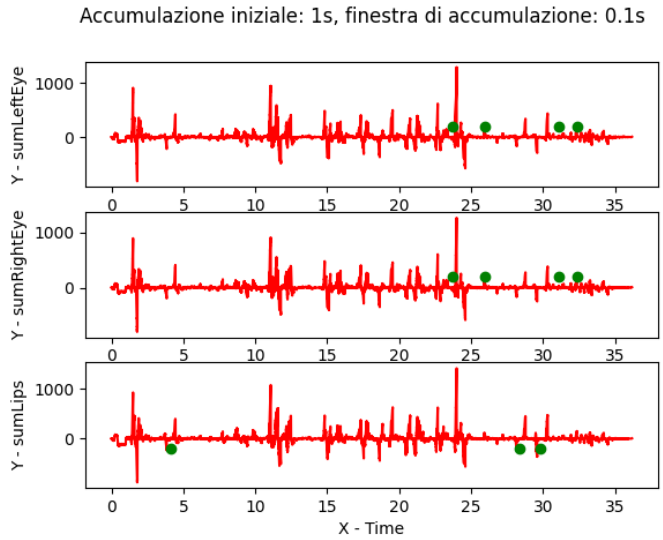


Figura 5.27: Grafico Tot 100ms

5.3.8 Andamento con Finestra di Accumulazione a 250ms

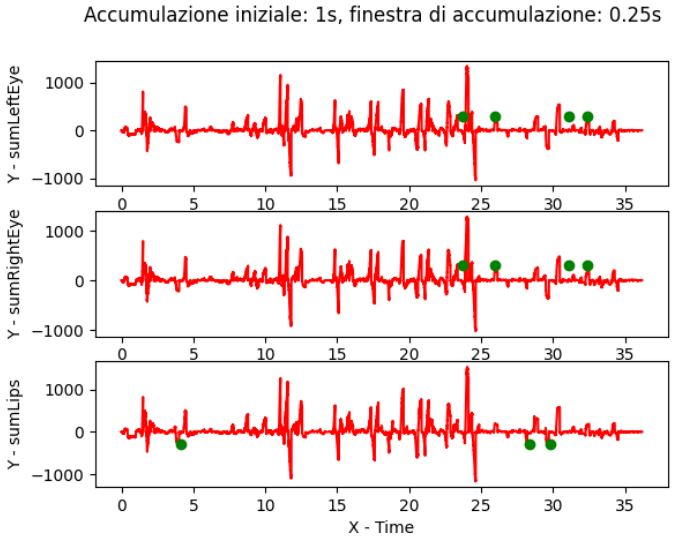


Figura 5.28: Grafico Tot 250ms

5.3.9 Andamento con Finestra di Accumulazione a 500ms

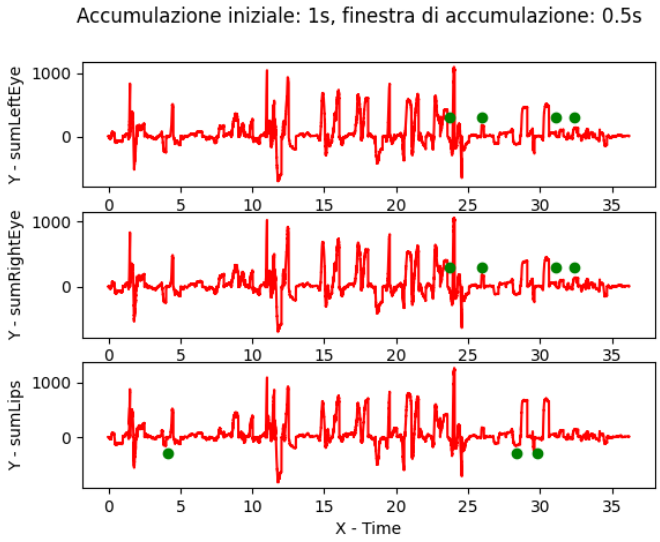


Figura 5.29: Grafico Tot 500ms

5.3.10 Andamento con Finestra di Accumulazione a 1000ms

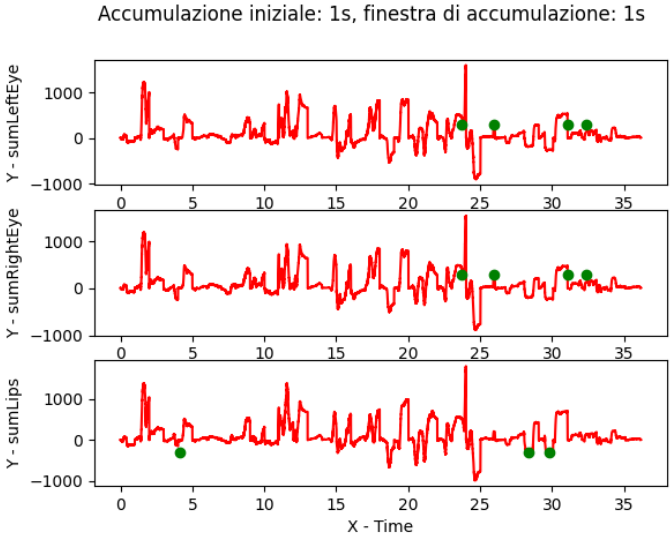


Figura 5.30: Grafico Tot 1000ms

Conclusioni

Lo sviluppo di un algoritmo per il di volti mediante eventi e frame da camere DVS è stata un'esperienza interessante ed altamente formativa che ha messo in mostra le potenzialità di una tecnologia relativamente nuova con una ricerca costante ed in continua evoluzione.
Grazie alle notevoli funzionalità messe a disposizione da *Python* e dai ricercatori che operano nel mondo delle camere ad eventi, probabilmente molte tecniche future saranno realizzate al suo interno e sarà quindi utile avere a disposizione un modulo software in grado di far interagire le Camere ad Eventi DVS con tale ambiente e quindi con i vari algoritmi implementati.
Il risultati finali hanno mostrato seppur in uno stadio preliminare come Le camere ad eventi riescano a catturare dinamiche veloci che sono di fondamentale importanza per scopo di tracking.

Bibliografia

[1] Seong G. Kong nad Senior Member Ralph Oyini Mbouna. «Visual Analysis of Eye State and Head Pose for Driver Alertnes Monitoring». In: *IEEE* 14 (2013), pp. 1462–1469.

[2] A. De Paola S. Vitabile e F. Sorbello. «A real-time non-intrusive FPGA-based Drowsiness system». In: *J Ambient Intell Human Comput, Springer, University of Palermo* (2011), pp. 251–262.

[3] A. Yilmaz et al. «Automatic feature detection and pose recovery for faces». In: *ACCV2002* 6 (2002).

[4] P. Smith et al. «Determining Driver Visual Attention with one camera». In: *IEEE Trans. on Intelligent Transportation Systems* 4 (2003).

[5] Qiang Ji et al. «Real-time eye gaze and face pose tracking for monitoring driver vigilance». In: *Real-Time Imaging* 8 (2002).

[6] Qiang Ji et al. «3D pose estimation and tracking from a monocular camera». In: *Image and Vision Computing* (2002).

[7] M. Kaneda et al. «Development of a drowsiness warning system». In: *11th Int. conf. on Enhanced Safety of Vehicles* (1994).

[8] R. Grace. «A drowsy driver detection system for heavy vehicles». In: *conf. on Ocular Measures of Driver Alertness* (1999).

[9] D. Cleveland. «Unobstusive eyelid closure and visual of regard measurement system». In: *conf. on Ocular Measures of Driver Alertness* (1999).

[10] P. Sherry et al. «Fatigue Countermeasures in the Railroad Industry: Past and Current Developments». In: *Published by Association of American Railroads* (2000).

[11] Arun Sahayadhas e Kenneth Sundaraj. «Detecting Driver Drowsiness Based on Sensors A Review». In: *ISSN 1424-8220, Malaysia 2012* (2012), pp. 16937–16953.

[12] Anirban dasgupta e Anjith George. «A Vision Based System For Monitoring The Loss Of Attention in Automotive Drivers». In: *IEEE Transaction* 14.2 (2013).

[13] Antoine Picot e Sylvie Charbonnier. «"On-Line Detection of Drowsiness Using Brain and Visual Information». In: *IEEE Transaction on systems, man and cybernetics part a: systems and humans* 42.3 (2012).

[14] Boon-Giin Lee e Wan-Young Chung. «Driver Alertness Monitoring Using Fusion of Facial Features and Bio-Signals». In: *(IEEE) Sensors journal* 12.7 (2012).

[15] Seong G. Kong Ralph Oyini Mbouna e Senior Member. «Visual Analysis of Eye State and Head Pose for Driver AlertnessMonitoring». In: *IEEE transactions on intelligent transportation systems* 14.3 (2013).

[16] Sara Lal Rami N. Khushaba Sarath Kodagoda e Gamini Dissanayake. «Driver Drowsiness Classification Using Fuzzy Wavelet-Packet-Based Feature-Extraction Algorithm». In: *(IEEE) Transactions* 58.1 (2011).

[17] D.J Sanghvi Raoul Lopes e Aditya Shah. «"Drowsiness Detection based on Eye Movement, Yawn Detection and Head Rotation». In: 2.6 (2012).

[18] Bo Cheng Wei Zhang e Yingzi Lin. «Driver Drowsiness Recognition Based on Computer Vision Technology». In: *IEEE* 17.3 (2012).

[19] Karamjeet Singh e Rupinder Kaur. *Physical and Physiological Drowsiness Detection Methods*. Vol. 2. 2013, pp. 35–43.

[20] Dr. Xiong (Bill) Yu e P.E. «Non-Contact Driver Drowsiness Detection System». In: *safety IDEA* (2012).

[21] Di Huang Student Member e Caifeng Shan an IEEE Member. «Local Binary Patterns and Its Application to Facial Image Analysis A Survey». In: *IEEE* 41 (2011), pp. 765–781.

[22] Jarek Krajewski e David Sommer. «Steering Wheel Behavior Based Estimation Of Fatigue». In: *Fifth International Driving Symposium on Human Factors in Driver Assessment, Training and Vehicle Design, Germany* ().

[23] J. Nuevo L. Bergasa et al. «RealTime System for Monitoring Driver Vigilance». In: *(IEEE) Transactions on Intelligent Transportation Systems* 7.1 (2006).

[24] Artem A. Lenskiy e Jong-Soo Lee. «Driver's Eye Blinking Detection Using Novel Color and Texture Segmentation Algorithms». In: *International Journal of Control, Automation and Systems* (2012), pp. 317–327.

[25] Paul R. Davidson Amol M. Malla et al. «Automated Video-based Measurement of Eye Closure for Detecting Behavioral Microsleep». In: *presented at 32nd Annual International Conference of the IEEE EMBS Buenos Aires, Argentina* (2010).

[26] Medialt. *Eye-Blink Detection System for Human-Computer Interaction*. URL: `http://www.medialt.no/pub/uikt/u2010/011-Krolak/index.html`.

[27] L. Farkas. «Anthropometry of the Head and Face». In: *New-York:Raven Press* (1994).

[28] A. Sohail e P. Bhattacharya. «Localization of Facial Feature Regions using Anthropmetric Face Model». In: *Int. conf. on Multidisciplinary Information Sciences and Technologies* (2006).

[29] S. Birchfield. «Elliptical head tracking using intensity gradients and color histograms». In: *IEEE CVPR* (1998).

[30] X. Yanjun Z. Xin e D. Limin. «Locating Facial Features with Color Information». In: *IEEE Int. conf. on Signal Processing* 2 (1998).

[31] M. Swain e D. Ballard. «Color Indexing». In: *Int. Journal of Computer Vision* 7.2 (1991).

[32] V. Vezhnevets e A. Degtiareva. «Robust and Accurate Eye Contour Extraction». In: *Moscow, Russia:Graphicon-2003* (2003).

[33] P. Kuo e J. Hannah. «An Improved Eye Feature Extraction Algorithm based on Deformable Templates». In: *ICIP* (2005).

[34] H. Liu Y. Wu e H. Zha. «A new method of human eyelids detection based on deformable templates». In: *IEEE Int conf. on Systems Man and Cybernetics (SMC'04)* (2004).

[35] G. Feng e P. Yuen. «Variance Projection Function and its application to eye detection for human face recognition». In: *Pattern Recognition Letters* 19.9 (1998).

[36] P. Sherry et al. «Fatigue Countermeasures in the Railroad Industry: Past and Current Developments». In: *Published by Association of American Railroads* (2000).

[37] D. Kriegman M.-H. Yang e N. Ahuja. «Detecting Faces in Images: A Survey». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24 (2002), pp. 34–58.

[38] P. Viola e M. Jones. «Rapid Object Detection Using a Boosted Cascade of Simple Features». In: *Proceedings of the International Conference on Computer Vision and Pattern Recognition* 1 (2001), pp. 511–518.

[39] S. Baluja H. Rowley e T. Kanade. «Neural Network-Based Face Detection». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20 (1998), pp. 23–38.

[40] M. Pietikainen A. Hadid e T. Ahonen. «A Discriminative Feature Space for Detecting and Recognizing Faces». In: *Proceedings of the International Conference on Computer Vision and Pattern Recognition* 2 (2004), pp. 797–804.

[41] K. Levi e Y. Weiss. «Learning Object Detection from a Small Number of Examples: The Importance of Good Features». In: *Proceedings of the International Conference on Computer Vision and Pattern Recognition* 2 (2004), pp. 53–60.

[42] M. Everingham Sivic e A. Zisserman. «Person Spotting: Video Shot Retrieval for Face Sets». In: *Proceedings of the International Conference on Image and Video Retrieval* (2005), pp. 226–236.

[43] M. Everingham Sivic, J. Sivic A. M. Everingham e A. Zisserman. «Hello, My name is... Buffy' – Automatic Naming of Characters in Tv Video». In: *Proceedings of the British Machine Vision Conference* (2006).

[44] C. Schmid R. Choudhury e K. Mikolajczyk. «Face Detection and Tracking in a Video by Propagating Detection Probabilities». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25 (2003), pp. 1215–1228.

[45] G.D. Fiore G. Boccignone V. Caggiano e A. Marcelli. «Probabilistic Detection and Tracking of Faces in Video». In: *Proceedings of the International Conference on Image Analysis and Processing* (2005), pp. 687–694.

[46] C. Huang Y. Li H. Ai e S. Lao. «Robust Head Tracking with Particles Based on Multiple Cues Fusion». In: *ECCV Workshop on HCI* (2006), pp. 29–39.

[47] É. Mémin E. Arnaud B. Fauvet e P. Bouthemy. «A Robust and Automatic Face Tracker Dedicated to Broadcast Videos». In: *roceedings of the International Conference on Image Processing* (2005), pp. 429–432.

[48] Z. Zhu e Q. Ji. «Robust Real-Time Eye Detection and Tracking Under Variable Lighting Conditions and Various Face Orientations». In: *Computer Vision and Image Understanding* 98 (2005), pp. 124–154.

[49] Z. Zhu Y. Tong Y. Wang e Q. Ji. «Robust Facial Feature Tracking Under Varying Face Pose and Facial Expression». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40 (2007), pp. 3195–3208.

[50] F. Dornaika e J. Ahlberg. «Fast and Reliable Active Appearance Model Search for 3-D Face Tracking». In: *IEEE Transactions on Systems, Man and Cybernetics* 34 (2004), pp. 1838–1853.

[51] F. Dornaika e F. Davoine. «On Appearance Based Face and Facial Action Tracking». In: *IEEE Transactions on Circuits and Systems for Video Technology* 16 (2006), pp. 1107–1124.

[52] O. Arandjelovic e A. Zisserman. «Automatic Face Recognition for Film Character Retrieval in Feature-Length Films». In: *Proceedings of the International Conference on Computer Vision and Pattern Recognition* 1 (2005), pp. 860–867.

[53] S. Satoh. «Comparative Evaluation of Face Sequence Matching for Content-Based Video Access». In: *Proceedings of the International Conference on Automatic Face and Gesture Recognition* (2000), pp. 163–168.

[54] O. Arandjelovic e R. Cipolla. «Automatic Cast Listing in Feature-Length Films with Anisotropic Manifold Space». In: *in Proc. Intl. Conf. on Computer Vision and Pattern Recognition* 2 (2006), pp. 1513–1520.

[55] Y. Nakamura S. Satoh e T. Kanade. «Name-It: Naming and Detecting Faces in News Videos». In: *IEEE MultiMediae* 6 (1999), pp. 22–35.

[56] J. Yang e A.G. Hauptmann. «Naming every individual in news video monologues». In: *in Proc. ACM Int. Conf. on Multimedia* (2004), pp. 580–587.

[57] J. Fisher O. Arandjelovic G. Shakhnarovich, R. Cipolla e T. Darrell. «Face Recognition with Image Sets Using Manifold Density Divergence». In: *Proceedings of the International Conference on Computer Vision and Pattern Recognition* 1 (2005), pp. 581–588.

[58] M. Turk e A. Pentland. «Face Recognition Using Eigenfaces». In: *Proceedings of the International Conference on Computer Vision and Pattern Recognition* (1991), pp. 586–591.

[59] M.E. Houle D.D. Le S. Satoh e D.P. Nguyen. «Finding Important People in Large News Video Databases Using Multimodal and Clustering Analysis». In: *Proceedings of the Second IEEE International Workshop on Multimedia Databases and Data Managemen* (2007), pp. 127–136.

[60] Nicolas Roussel Gery Casiez e Daniel Vogel. «A simple speed-based low-pass filter for noisy input in interactive systems». In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, New York, NY, USA* (2012), pp. 2527–2530.

[61] Thomas Gerig. «A morphable model for the synthesis of 3D faces». In: *In Proceedings of 36th Internaional Conference and Exhibition on Computer Graphics and Interactive Techniques* (1999), pp. 187–194.

[62] Volker Blanz e Thomas Vetter. «Morphable face models - an open framework». In: *arXiv preprint arXiv 1709.08398* (2017).

[63] Edwin Catmull e Jim Clark. «Recursively generated B-spline surfaces on arbitrary topological meshes». In: *Computer-Aided Design, 10(6)* (1978), pp. 350–355.

[64] Blazeface. «Sub-millisecond neural face detection on mobile GPUs». In: *CVPR 2019 - Third Workshop on Computer Vision for AR/VR. Workshop Submission ID 5* ().

[65] Michael D. Grossberg e Shree K. Nayar. «What is the space of camera response functions?» In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2 (2003), pp. 1–602.

[66] Github. *Face Geometry*. URL: `https://github.com/google/mediapipe/tree/master/mediapipe/modules/face_geometry`.

[67] Github. *Canonical Face Model*. URL: `https://github.com/google/mediapipe/blob/master/mediapipe/modules/face_geometry/data/canonical_face_model.fbx`.

[68] Github. *Geometry Pipeline Calculator*. URL: `https://github.com/google/mediapipe/blob/master/mediapipe/modules/face_geometry/geometry_pipeline_calculator.cc`.

[69] Github. *Subgraph*. URL: `https://github.com/google/mediapipe/blob/master/mediapipe/modules/face_geometry/face_geometry_from_landmarks.pbtxt`.

[70] Github. *Message*. URL: `https://github.com/google/mediapipe/blob/master/mediapipe/modules/face_geometry/protos/face_geometry.proto`.

[71] Google. *Google Colab*. URL: `https://colab.research.google.com/drive/1FCxIsJS9i58uAsgsLFqDwFmiPO14Z2Hd`.

[72] Wikipedia. *Event Camera*. URL: `https://en.wikipedia.org/wiki/Event_camera`.

[73] Christian Brandli - Raphael Berner - Minhao Yang - Shih Chii Liu - Tobi Delbruck. «A 240 × 180 130 dB 3 µs latency global shutter spatiotemporal vision sensor». In: *IEEE J. Solid-State Circuits, 49(10)* (2014), pp. 2333–2341.

[74] Kirk Y.W. Scheper Bas J. Pijnacker Hordijk e Guido C.H.E. de Croon. «Vertical landing for micro air vehicles using event-based optical flow». In: *J. Field Robotics, 35(1)* (2018), pp. 69–90.

[75] Elias Mueggler - Nathan Baumli - Flavio Fontana - Davide Scaramuzza. «Towards evasive maneuvers with quadrotors using dynamic vision sensors». In: *In Eur. Conf. Mobile Robots* (2015), pp. 1–8.

[76] Kevin Klever Davide Scaramuzza Davide Falanga. «Vertical landing for micro air vehicles using event-based optical flow». In: *Sci. Robotics, 5(40)* (2020).

[77] David Weikersdorfer - David Adrian - Daniel Cremers - Jorg Conradt. «Event-based 3D SLAM with a depthaugmented dynamic vision sensor». In: *In IEEE Int. Conf. Robotics Autom* (2014), pp. 359–364.

[78] Michael Milford - Hanme Kim - Stefan Leutenegger - Andrew Davison. «Towards visual SLAM with event-based cameras». In: *RSS Workshop on the Problem of Mobile Sensors* (2015).

[79] Garrick Orchard - Cedric Meyer - Ralph Etienne-Cummings - Christoph Posch - Nitish Thakor - Ryad Benosman. «HFirst: A temporal approach to object recognition». In: *IEEE Trans. Pattern Anal. Mach. Intell., 37(10)* (2015), pp. 2028–2040.

[80] C. Posch P. Lichtsteiner e T. Delbruck. «A 128×128 120 dB 15 µs latency asynchronous temporal contrast vision sensor». In: *IEEE J. Solid-State Circuits* 43.2 (2008), pp. 566–576.

[81] Github. *Event-based Vision Resources*. URL: `https://github.com/uzh-rpg/event-based_vision_resources`.

[82] B. Son - Y. Suh - S. Kim - H. Jung - J.S. Kim - C. Shin - K. Park - K. Lee - J. Park - J. Woo - Y. Roh - H. Lee - Y. Wang - I. Ovsiannikov e H. Ryu. «A 640x480 dynamic vision sensor with a 9µm pixel and 300Meps address-event representation». In: *IEEE Intl. Solid-State Circuits Conf. (ISSCC)* (2017).

[83] Workshop. *First International Workshop on Event-based Vision.* URL: `http://rpg.ifi.uzh.ch/ICRA17_event_vision_workshop.html`.

[84] C. Brandli - R. Berner - M. Yang - S.C. Liu e T. Delbruck. «A 240x180 130dB 3µs latency global shutter spatiotemporal vision sensor». In: *IEEE J. Solid-State Circuits* 49.10 (2014), pp. 2333–2341.

[85] M. Mahowald. «VLSI analogs of neuronal visual processing: A synthesis of form and function». In: *Ph.D. dissertation, California Institute of Technology, Pasadena, California* (1992).

[86] T. Delbruck e C. A. Mead. «Time-derivative adaptive silicon photoreceptor array». In: *Proc. SPIE, Infrared sensors: Detectors, Electron., and Signal Process.* 1541 (1991), pp. 92–99.

[87] D. Matolin C. Posch e R. Wohlgenannt. «A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS». In: *IEEE J. SolidState Circuits* 49.1 (2011), pp. 259–275.

[88] D. Matolin C. Posch e R. Wohlgenannt. «A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS». In: *IEEE J. SolidState Circuits* (2010), pp. 400–401.

[89] E. Mueggler - H. Rebecq - G. Gallego - T. Delbruck e D. Scaramuzza. «The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry and SLAM». In: *Int. J. Robot. Research* 36.2 (2017), pp. 142–149.

[90] M. Gehrig G. Gallego e D. Scaramuzza. «Focus is all you need: Loss functions for event-based vision». In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)* (2019).

[91] V. Koltun H. Rebecq R. Ranftl e D. Scaramuzza. «High speed and high dynamic range video with an event camera». In: *IEEE Trans. Pattern Anal. Mach. Intell.* (2019).

[92] K. G. Derpanis D. Gehrig A. Loquercio e D. Scaramuzza. «End-to-end learning of representations for asynchronous eventbased data». In: *Int. Conf. Comput. Vis. (ICCV)* (2019).

[93] M. Liu e T. Delbruck. «Adaptive time-slice block-matching optical flow algorithm for dynamic vision sensors». In: *British Mach. Vis. Conf. (BMVC)* (2018).

[94] A. Aimar - H. Mostafa - E. Calabrese - A. Rios-Navarro - R. Tapiador Morales - I. Lungu - M. B. Milde - F. Corradi - A. Linares-Barranco - S. Liu e T. Delbruck. «NullHop: A flexible convolutional neural network accelerator based on sparse representations of feature maps». In: *IEEE Trans. Neural Netw. Learn. Syst.* 30.3 (mar. 2019), pp. 644–656.

[95] C. Sulzbachner J. Kogler e W. Kubinger. «Bio-inspired stereo vision system with silicon retina imagers». In: *Int. Conf. Comput. Vis. Syst. (ICVS)* (2009), pp. 174–183.

[96] R. Benosman - C. Clercq - X. Lagorce - S.H. Ieng e C. Bartolozzi. «Event-based visual flow». In: *IEEE Trans. Neural Netw. Learn. Syst* 25.2 (2014), pp. 407–417.

[97] K. Hara Y. Sekikawa e H. Saito. «EventNet: Asynchronous recursive event processing». In: *IEEE Conf. Comput. Vis. Pattern Recog. (CVPR)* (2019).

[98] M. Litzenberger - C. Posch - D. Bauer - A. N. Belbachir - P. Schon - B. Kohn e H. Garn. «Embedded vision system for real-time object tracking using an asynchronous transient vision sensor». In: *Digital Signal Processing Workshop* (2006), pp. 173–178.

[99] Z. Ni - A. Bolopion - J. Agnus - R. Benosman e S. Regnier. «Asynchronous event-based visual shape tracking for stable haptic feedback in microrobotics». In: *IEEE Trans. Robot.* 28.5 (2012), pp. 1081–1089.

[100] Z. Ni - S.H. Ieng - C. Posch - S. Regnier e R. Benosman. «Visual tracking using neuromorphic asynchronous event-based cameras». In: *Neural Computation* 27.4 (2015), pp. 925–953.

[101] E. Mueggler D. Tedaldi G. Gallego e D. Scaramuzza. «Feature detection and tracking with the dynamic and active-pixel vision sensor (DAVIS)». In: *Int. Conf. Event-Based Control, Comm. Signal Proc. (EBCCSP)* (2016).

[102] G. Gallego B. Kueng E. Mueggler e D. Scaramuzza. «Lowlatency visual odometry using event-based feature tracks». In: *IEEE Int. Conf. Intell. Robot. Syst. (IROS)* (2016).

[103] Gitlab. *DV-Python*. URL: `https://gitlab.com/inivation/dv/dv-python`.

[104] Wikipedia. *Python*. URL: `https://it.wikipedia.org/wiki/Python`.

[105] Unibo. *Python*. URL: `https://datascienceunibo.github.io/dia/download/slides/00-lab-python_setup.pdf`.

[106] Inivation. *DV*. URL: `https://inivation.gitlab.io/dv/dv-docs/docs/getting-started.html`.

[107] Python. *Python*. URL: `https://www.python.org/downloads/`.

[108] Microsoft. *Visual Studio Code*. URL: `https://code.visualstudio.com/download`.

Elenco delle figure