UNIVERSITÁ POLITECNICA DELLE MARCHE

FACOLTÁ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di un algoritmo per il rilevamento dell'eye blink mediante camere ad eventi

Design and development of an algorithm to detect eye blinking using event cameras

Relatore:
Dott. Mancini Adriano

Laureando:
Mele Alessandro

Anno Accademico 2020-2021

Indice

Capitolo 1

Introduzione

Il XXI secolo viene definito dagli storici come *l'era della tecnologia*, la quale poggia le fondamenta sull'informatica: tale scienza si è sviluppata in tempi brevi ed in modo straordinario, diventando una componente fondamentale e indispensabile nella vita quotidiana.
Insieme alla tecnologia, l'informatica accompagnerà nei prossimi decenni la nostra vita e il nostro modo di vivere: in essa è racchiuso tutto il mondo, che ci sta trasformando e condizionando.
La tecnologia ha sviluppato metodi e sistemi avanzati, permettendo numerose possibilità prima inimmaginabili, dalla modernizzazione dei macchinari, ad esempio in campi come la medicina chirurgica, alla guida assistita e la ricerca scientifica in generale, come ad esempio calcolatori con elevatissima precisione.
Per quanto evoluto, un calcolatore esegue calcoli: legge ed elabora il contenuto informativo, fornendo in output un'opportuna rappresentazione secondo uno schema di codifica.
Una domanda lecita potrebbe essere: *"un calcolatore è in grado di pensare?"*.
La risposta non è scontata, anzi l'argomento è di ampio dibattito: l'*Intelligenza artificiale* è, oltre che per l'informatica, oggetto di studio anche della filosofia e della psicologia.
Per rispondere, si può osservare che l'intelligenza e la consapevolezza di vivere sono due concetti diversi:

- L'intelligenza è la capacità di risolvere autonomamente un nuovo problema o un imprevisto, mai affrontati prima ricorrendo all'esperienza e alla conoscenza acquisita;

- La consapevolezza di vivere invece è ben più complessa e probabilmente caratterizza soltanto gli esseri viventi.
  Tutto ciò che lega il settore a questi aspetti, al momento è oggetto di studio della filosofia, degli studi sperimentali oppure della fantascienza.

Per concludere, occorre quindi considerare l'*Intelligenza artificiale* come uno strumento per comprendere l'ambiente e prendere decisioni razionali, nulla di più.

1.1 Obiettivi

Lo scopo del seguente elaborato è la progettazione e lo sviluppo di un algoritmo per il rilevamento dell'*Eye blink* mediante camere ad eventi.
Per *Eye blink* si intende il movimento di apertura e chiusura dell'occhio, il quale, se prolungato, può rappresentare una situazione di stanchezza, sonnolenza o affaticamento da parte dell'uomo.
In particolare, l'algoritmo ha il compito di analizzare l'andamento nel tempo dei *keypoints* relativi agli occhi, tracciandone dei grafici per individuare i movimenti di apertura e chiusura prolungati.
In primo luogo, verrà fornita una panoramica generale sui sistemi di sicurezza e assistenza implementati a bordo dei veicoli moderni, per poi introdurre i concetti di *Intelligenza artificiale*, *Apprendimento automatico*, *Apprendimento approfondito*, *Visione artificiale*, camere *frame* ed *event-based*.
Per concludere, verranno mostrati gli strumenti software utilizzati per lo sviluppo dell'algoritmo ed il funzionamento dello stesso.
Per la visualizzazione e/o condivisione del progetto, il sottoscritto mette a disposizione il *repository* GitHub contenente il codice sorgente, previa autorizzazione, al seguente link: EC_Project.

Capitolo 2

Sistemi di rilevamento della stanchezza del guidatore

Il crescente sviluppo della sicurezza nei veicoli ha reso il conducente l'anello debole della catena.
Per aiutare gli automobilisti deconcentrati, stanchi o assonnati, i veicoli vengono prodotti con sistemi di assistenza alla guida: questi, noti come *ADAS* (*Advanced Driver Assistance Systems*) o *DAS*, fanno largo uso della *Visione artificiale*.
Le soluzioni attuali propongono:

- Sistemi di trasporto intelligenti, per garantire la sicurezza nella navigazione, aumentare l'efficienza ed evitare traffico;
- Guida assistita/autonoma, per sostituire, parzialmente e/o non, al conducente il veicolo stesso;
- Sistemi avanzati di assistenza alla guida, per monitorare il movimento e il flusso dei veicoli lungo la rete stradale.

L'obiettivo di tali soluzioni è supportare il conducente per mantenere condizioni di viaggio sicure e fornire un avviso tempestivo nel momento in cui emerge una situazione critica.
Si ipotizza che il conducente, avvisato mediante segnali visivi e/o sonori, segua le raccomandazioni dell'assistente automatico; ma se ciò non dovesse accadere, il sistema deve essere in grado di accorgersi della situazione ed effettuare azioni al posto del guidatore stesso.

2.1 Il caso Tesla

Una delle grandi sfide della tecnologia moderna è quella di permettere ad un veicolo di poter transitare per le strade autonomamente, senza la presenza di un conducente umano.
I moderni sistemi *ADAS* a bordo dei veicoli hanno, come precedentemente detto, l'obiettivo di **assistere** il conducente durante il viaggio e non di sostituirsi ad esso.
A sostegno di ciò, si espone il caso dell'incidente che ha coinvolto Tesla, ed in particolare, una vettura Model S.
Nella notte del 17 aprile 2021, il veicolo viaggiava ad alta velocità, quando non ha superato una curva ed è uscito dalla carreggiata: il veicolo si è poi schiantato contro un albero e andato in fiamme.
Lo scontro è stato fatale ed ha causato la morte di due passeggeri di 59 e 70 anni, uno sul sedile anteriore del passeggero, l'altro sul posteriore, senza quindi nessun conducente.

2.2 Riconoscimento dei segnali stradali

Gli incidenti stradali sono un grave problema della società: uno dei motivi principali è l'omissione della segnaletica stradale da parte dei conducenti.
Il riconoscimento della segnaletica stradale è un campo di ricerca attivo da diversi decenni ed i risultati ottenuti hanno permesso l'implementazione di alcune soluzioni in condizioni *real-time*.
Tuttavia, tali sistemi sono limitati a poche tipologie di segnali, quali limite di velocità circolari o di divieto di sorpasso; ciò accade perché molti segnali condividono lo stesso aspetto generale, quindi compongono un sottoinsieme di elementi molto simili tra loro e di difficile distinzione.
La situazione diventa ancor più ancora più complessa quando i segnali sono analizzati in particolari condizioni di illuminazione, prospettiva, sfocatura e condizioni atmosferiche.

Figura 2.1: Esempio di riconoscimento segnali stradali, fonte: ibn software

2.3 Prevenzione delle collisioni

I sistemi di prevenzione delle collisioni (*CAV*) sono i più avanzati sistemi di assistenza alla guida: avvertono il conducente al rilevamento della situazione pericolosa e sono in grado di utilizzare i sistemi di frenatura e sterzata per evitare una potenziale collisione.
L'aspetto fondamentale è il riconoscimento dell'ambiente intorno al veicolo: il sistema deve essere in grado di rilevare ostacoli quali pedoni, animali, altri veicoli o elementi statici dell'infrastruttura stradale.
Nelle odierne implementazioni dei sistemi *CAV*, sono utilizzate soluzioni ibride che fanno uso di sensori ambientali quali *RADAR*, *LiDAR* e videocamere.
Alcuni produttori di veicoli di classe premium offrono ai loro utenti come equipaggiamento opzionale delle *termocamere*, che consentono, in condizioni ambientali ostili, di offrire un'anteprima della strada oltre la gamma dei fari.

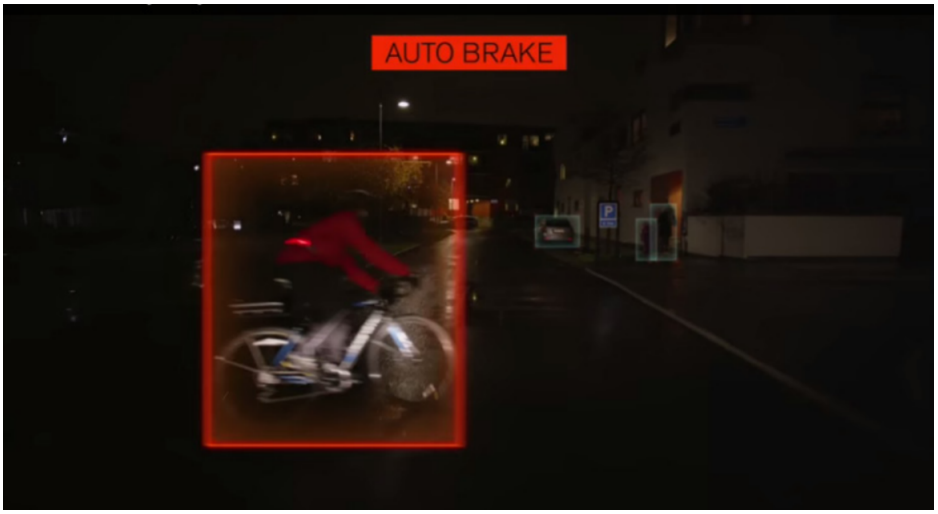

Figura 2.2: Esempio di rilevamento collisione, fonte: Adam Nowosielski

2.4 Avviso di deviazione dalla corsia

Un essere umano utilizza molti segnali percettivi durante la guida, come colore e consistenza della strada, confini e segnaletica orizzontale.
I sistemi di avviso di deviazione dalla corsia cercano di imitare lo stesso principio, dunque, una volta rilevata la strada, si procede analizzando numero di corsie, unioni, divisioni e terminazione delle corsie.
Lo schema di esecuzione consiste nell'elaborazione dell'immagine, applicando tecniche di rimozione dell'ombra o troncamento dell'area dell'immagine nella regione al di sotto dell'orizzonte: qui vengono rilevate le caratteristiche di basso livello per l'attività di rilevamento della corsia e strada.
Vengono poi estratte caratteristiche quali segni di colore e trame, che costituiscono la base per la fase di montaggio del modello della strada.
La fase finale dell'elaborazione è l'integrazione temporale, che tiene conto delle informazioni sul frame attuale e precedente.

Figura 2.3: Esempio di rilevamento di una corsia, fonte: Adam Nowosielski

2.5 Rilevamento dell'affaticamento del conducente

La stanchezza del conducente è un fattore che contribuisce notevolmente agli incidenti stradali: monitorando l'affaticamento è possibile migliorare la sicurezza del veicolo.
Successivamente al rilevamento, il sistema di assistenza alla guida può stimolare il conducente con alcuni segnali acustici, oppure, in situazioni critiche, fermare il veicolo.
Una possibile strategia è la misurazione dei parametri fisiologici del conducente come la frequenza cardiaca o l'attività cerebrale: queste tecniche richiedono però il posizionamento di alcuni scomodi sensori sul corpo.
I più convenienti e non invasivi sono i sistemi di monitoraggio basati sulla visione, ad esempio il movimento della testa, delle palpebre, dello sguardo e degli occhi.
Una volta rilevati il viso e gli occhi, l'attenzione si sposta sul tracciamento oculare, il quale consiste nel rilevare lo stato degli occhi (aperti, parzialmente aperti o chiusi): quando il conducente è assonnato, gli occhi iniziano a chiudersi; dunque se la chiusura è di lunga durata, può indicare una situazione di rischio.
Un limite di tale approccio risulta essere la visibilità degli occhi nel caso in cui si indossi occhiali da sole.

Capitolo 3

Concetti preliminari

3.1 Artificial intelligence

Il termine *Intelligenza artificiale* indica la capacità di un sistema informatico di svolgere attività che normalmente prevedono la presenza o le capacità di un essere umano.
La descrizione risulta molto ampia e lascia spazio a diverse interpretazioni e sviluppi.

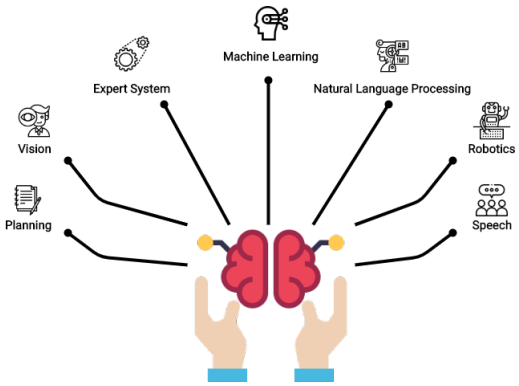

Figura 3.1: Intelligenza artificiale, fonte: medium

Le basi dell' *Intelligenza artificiale* risalgono al 1936, anno in cui Alan Turing formulò l'ipotesi di una macchina in grado di svolgere qualsiasi tipo di calcolo.
Tale macchina, detta *macchina di Turing*, rappresenta la genesi dei calcolatori moderni.
Turing andò ben oltre il concetto di macchina, ponendosi un interrogativo fondamentale: *"esiste la facoltà di pensiero nelle macchine?"*.
Lo studioso arrivò a formulare un teorema secondo il quale una *macchina pensante* riesce a fornire una sequenza logica, una concatenazione efficace ed empirica delle idee, arrivando addirittura ad esprimerle.

Famoso è il *test di Turing*, ossia un esperimento per riconoscere la presenza di una *macchina intelligente*: è il metodo più efficace per capire se siamo in presenza di un modello pensante, e nessuna macchina finora lo ha mai superato.

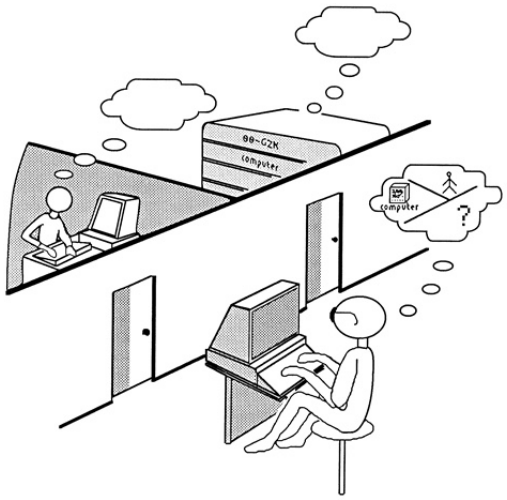

Figura 3.2: Test di Turing, fonte: nous

Il termine *Intelligenza artificiale* venne ufficialmente coniato nel 1956 da John McCarthy, professore di matematica, durante un seminario che ruotava attorno alla possibilità di far svolgere alle macchine dei compiti dove era necessaria l'intelligenza umana.
Questo seminario si pose degli obiettivi da raggiungere in un arco temporale di massimo dieci anni:

- Dimostrare teoremi matematici;
- Battere un campione di scacchi;
- Spiegare i comportamenti umani.

L'unico obiettivo raggiunto finora è stato battere il campione di scacchi Gasparov: impresa riuscita dalla macchina *Deep Blue*, costruita da IBM nel 1996.
L'*Intelligenza artificiale* si divide in due macro aree:

- L'*Intelligenza artificiale debole* è basata interamente su una singola frase: *"come se"*; ovvero agisce e pensa come se potesse pensare, ma in realtà non può.
  Lo scopo di questa intelligenza non è quindi di realizzare programmi che emulano il comportamento umano nella risoluzione di problemi, ma di creare sistemi in grado di agire con successo in alcune funzioni complesse solite di un essere umano, come la traduzione di testi.

Un aspetto fondamentale è la necessaria presenza di un essere umano, poiché il sistema non è in grado di pensare in maniera autonoma.

- L'*Intelligenza artificiale forte* invece trasforma il sistema rendendolo in grado di pensare come una mente umana vera e propria: ha quindi sia capacità cognitive che conoscenza dei propri limiti e capacità.
  Alla base di questa Intelligenza ci sono sistemi e algoritmi esperti, in grado di riprodurre le conoscenze e prestazioni delle persone capaci in determinati settori.
  Non esiste, al momento, nessuna *Intelligenza artificiale forte*.

3.2 Machine learning

Il *Machine learning* è un ramo dell' *Intelligenza artificiale* definito come una serie di meccanismi che permettono ad una *macchina intelligente* di imparare a svolgere in modo efficiente uno specifico *task*, tramite l'esperienza e le proprie capacità.
Il *Machine learning* a sua volta è diviso in diverse categorie, come la *Visione artificiale*, lo *Speech* ed il *Natural Language Processing*.
Si distinguono tre differenti sistemi di apprendimento automatico:

- L'*apprendimento supervisionato*, consiste nel fornire al sistema una serie di modelli ed esempi che permettono di costruire un vero e proprio *database* di informazioni e di esperienze.
  In questo modo, quando la macchina fronteggia un problema, attinge alle esperienze nel proprio sistema, le analizza, e decide quale risposta dare sulla base di esperienze già codificate.
  Questo tipo di apprendimento è, in qualche modo, fornito già e la macchina deve scegliere la migliore risposta in base allo stimolo dato.
  Esempi classici di questa tipologia sono la *regressione* e la *classificazione*, entrambi ricevono dati in input che possono essere continui o discreti:

  - La regressione fornisce in output risultati basati su dati continui, come ad esempio la previsione del prezzo di vendita di un bene;
  - La classificazione invece produce in output un risultato basato su dati discreti, come ad esempio la classificazione di un'immagine in una categoria, come cane o gatto.

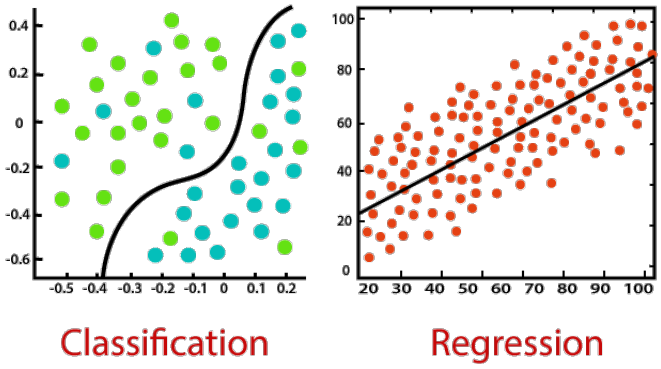

Figura 3.3: Confronto tra classificazione e regressione, fonte: javatpoint

- L'*apprendimento non supervisionato*, prevede che le informazioni inserite all'interno della macchina non siano codificate, ossia senza alcun esempio del loro utilizzo e conoscenza dei risultati attesi a seconda della scelta effettuata.
  È quindi compito della macchina stessa classificare le informazioni in proprio possesso, organizzarle, imparare il loro significato ed il risultato.
  L'apprendimento senza supervisione offre maggiore libertà di scelta alla macchina, che deve organizzare le informazioni in maniera intelligente ed apprendere i risultati per le differenti situazioni che si presentano.
  Alcune tecniche sono:

  – *Clustering*, tecnica che raggruppa dati privi di etichette in gruppi in base alle loro caratteristiche, le quali non sono note in numero e specifiche.

  – Ridimensionamento dei dati, tecnica che si utilizza quando il numero di caratteristiche è troppo elevato, si procede per raffinamento senza perdere le informazioni fondamentali.

- *L'apprendimento per rinforzo*, rappresenta il sistema più complesso, poiché prevede che la macchina sia dotata di sistemi e strumenti in grado di migliorare il proprio apprendimento e di comprendere le caratteristiche dell'ambiente circostante.
  La macchina riceve in input un obiettivo da raggiungere, ma non sa come, poiché non ha né un *dataset* di esempi per l'addestramento né una base di conoscenza pregressa.
  In primo luogo la macchina osserva l'ambiente che lo circonda e lo trasforma in un vettore di caratteristiche, dove ogni combinazione di elementi è un diverso stato dell'ambiente.
  Quando il sistema prende una decisione, analizza il cambiamento dello stato dell'ambiente valutando i *feedback* tramite una *funzione di rinforzo*.

Tale funzione misura il grado di successo di un'azione o decisione, rispetto a un obiettivo predeterminato, mediante:

- Ricompensa, se il *feedback* è positivo, il sistema si è avvicinato all'obiettivo dopo l'azione e la funzione assegna un valore reale positivo alla macchina;
- Penalizzazione, se il *feedback* è negativo, il sistema si è allontanato dall'obiettivo dopo l'azione e la funzione assegna un valore reale negativo alla macchina.

3.3 Deep learning

L'*Apprendimento approfondito* è una sotto categoria del *Machine learning* che si occupa della creazione di modelli di apprendimento su più livelli.
Questi algoritmi hanno lo scopo di comprendere il funzionamento del cervello umano, e di come esso riesca ad interpretare le immagini ed il linguaggio.
L'apprendimento così realizzato ha la forma di una piramide: fornisce la rappresentazione dei dati a livello gerarchico, dove i concetti più alti sono appresi a partire dai livelli più bassi.
Questa rappresentazione consente alla macchina di classificare i dati in input e in output, evidenziando quelli rilevanti ai fini della risoluzione del problema e scartando quelli irrilevanti.
L'elaborazione dei dati è simile alla mente umana, la quale elabora dati e conoscenze in maniera non lineare.
Il *Deep learning* poggia le sue fondamenta sulle *Reti neurali*, le quali consentono di "pensare" in maniera simile al funzionamento dei neuroni umani.

3.3.1 Reti neurali artificiali

Le *Reti neurali*, o *Reti neurali artificiali*, sono l'elemento centrale degli algoritmi di *Deep learning*.
Il loro nome e la loro struttura sono ispirati al cervello umano, imitando il modo in cui i neuroni biologici inviano segnali.
Le *Reti neurali artificiali* sono composte da livelli di nodi che contengono un livello di input, uno o più livelli nascosti e un livello di output.
Ciascun nodo, o *neurone artificiale*, si connette ad un altro e ha un peso e una soglia associati:

- se l'output di qualsiasi singolo nodo è al di sopra del valore di soglia specificato, tale nodo viene attivato, inviando i dati al successivo livello della rete;

- in caso contrario, non viene passato alcun dato al livello successivo della rete.

Ogni singolo nodo viene rappresentato come un proprio modello di *Regressione lineare*, composto da dati di input, dei pesi, una distorsione e un output.
La formula vale:

$$\sum w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias$$

Mentre, la formula per determinare l'output vale:

$$f(x) = \begin{cases} 1 \text{ se } \sum w_1 x_1 + bias \geq 0 \\ 0 \text{ se } \sum w_1 x_1 + bias < 0 \end{cases}$$

Una volta determinato un livello di input vengono assegnati i pesi, che stabiliscono l'importanza della variabile ai fini del risultato (pesi maggiori contribuiscono in modo più significativo rispetto a pesi minori).
Una volta moltiplicati per i pesi, gli input vengono sommati ed il risultato viene passato ad una funzione di attivazione, che determina l'output finale.
Superata una determinata soglia, tale output attiva il nodo, passando i dati al livello successivo nella rete: l'output di un nodo diventa quindi l'input di quello successivo.
I più comuni tipi di reti neurali sono:

- *Percettrone*, la più antica rete neurale, ha un singolo neurone ed è la forma più semplice di una rete neurale;

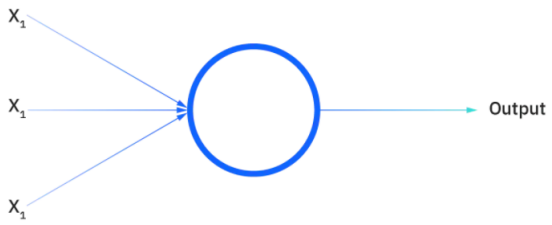

Figura 3.4: Percettrone, fonte: IBM

- Reti neurali *feedforward*, o *Percettroni multilivello*, sono formate da un livello di input, uno o più livelli nascosti e un livello di output.
  I nodi sono formati da neuroni *sigmoidi*, poiché la maggior parte dei problemi del mondo reale risulta essere non lineare.
  Sono la base per la *Visione artificiale*, il *Natural language processing* e altre reti neurali.

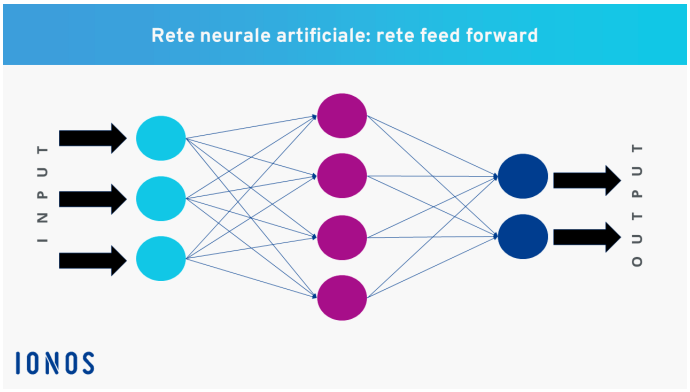

Figura 3.5: Rete neurale feedforward, fonte: ionos

- Reti neurali *convolutive*, sono simili alle reti *feedforward* e sfruttano i principi dall'algebra lineare (come la moltiplicazione della matrice) per identificare i modelli all'interno di un'immagine nella *Visione artificiale*.
- Reti neurali ricorrenti, sono identificate dai loro loop di feedback.
  Ci si avvale delle reti ricorrenti quando si utilizzano i dati per previsioni su risultati futuri, come ad esempio previsioni di vendita.

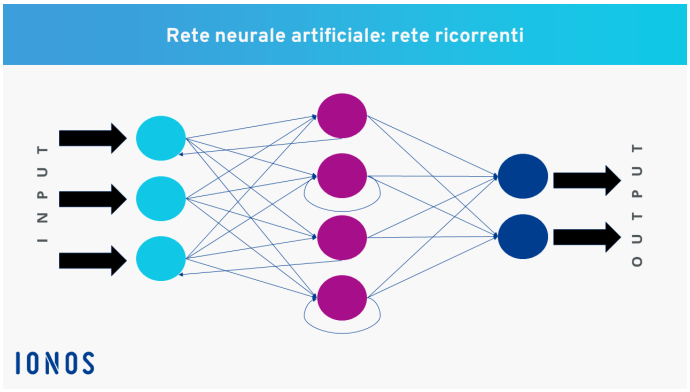

Figura 3.6: Reti neurali ricorrente, fonte: ionos

3.4 Computer vision

La *Visione artificiale* è un ramo del*Machine learning* che rende possibile il riconoscimento e la manipolazione delle informazioni derivanti da una sorgente di input visiva: l'obiettivo è di imitare il funzionamento del processo visivo dell'uomo.
Tali compiti vengono affrontati mediante tecniche di *Deep learning*.
Per definire il risultato di un processo, è importante che l'algoritmo effettui degli esperimenti e venga addestrato allo scopo per cui deve operare: per questo motivo la *Visione artificiale* è strettamente legata al *Machine learning*.
Alcune tecniche utilizzate sono:

- Classificazioni di immagini: dato un insieme di immagini (etichettate in una categoria), si cerca di prevedere se questa categoria appare in un nuovo set di immagini di prova misurandone l'accuratezza delle previsioni.
  L'addestratore insegna all'algoritmo a riconoscere un *dataset* di immagini, in modo che diventino abili da prevedere con elevata probabilità di cosa si tratti;

- Rilevazione degli oggetti: consiste nella definizione degli oggetti all'interno delle immagini, di solito tramite etichette.
  Esistono soluzioni che consentono di localizzare sia singoli che più oggetti.

Figura 3.7: Esempio di rilevazione di oggetti, fonte: nonteek

- Tracciamento di oggetti: si riferisce al processo di seguire uno o più oggetti di interesse in una determinata scena.
  Tradizionalmente ha applicazioni nelle interazioni video e nel mondo reale dove le osservazioni sono fatte dopo un rilevamento iniziale dell'oggetto; è fondamentale nello sviluppo di sistemi di guida autonoma.

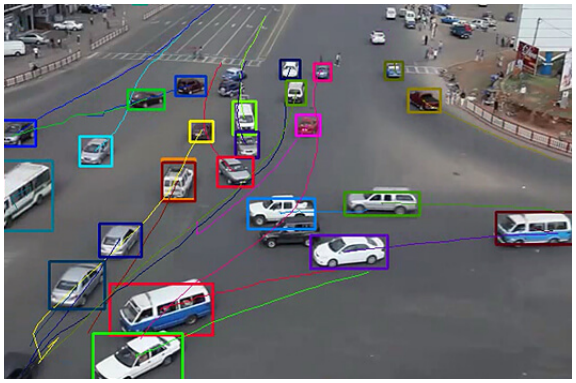

Figura 3.8: Esempio di tracciamento di veicoli, fonte: aidetic

- Segmentazione semantica: processo che consente di dividere intere immagini in gruppi di pixel, i quali possono essere etichettati e classificati.
  In particolare, la segmentazione semantica cerca di capire il ruolo di ciascun pixel nell'immagine: ad esempio, riconoscere persone, strade, veicoli ed alberi.

- Segmentazione dell'istanza: tecnica molto complessa che separa le diverse istanze delle classi, ad esempio l'etichettatura di dieci persone con dieci colori diversi.

Figura 3.9: Esempio di segmentazione semantica e dell'istanza, fonte: neptune

3.5 Camere frame-based

I sensori maggiormente diffusi nel mercato per l'acquisizione di immagini e video impiegano tecnologie basate su *frame*.
Un *frame* è una matrice contenente, per ogni cella, dati i cui valori rappresentano l'intensità associata alla stessa.

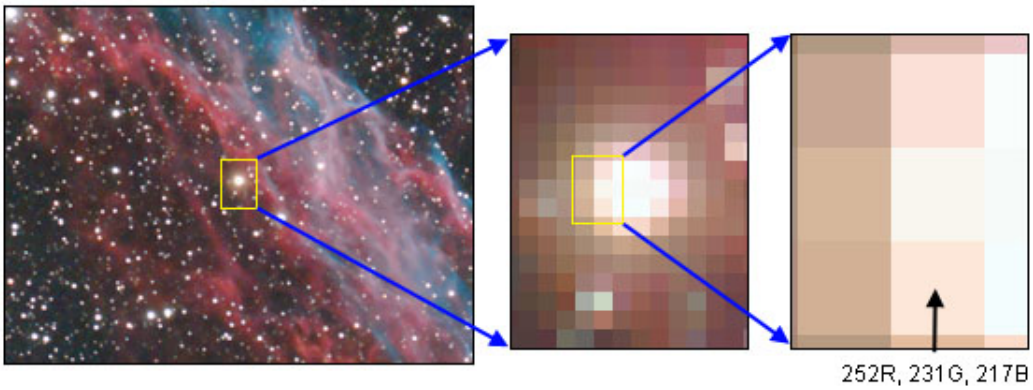

Figura 3.10: Esempio di immagine *frame-based*, fonte: astropix

Tali sensori campionano la luce nello spazio, nel tono e nel tempo:

- Nello spazio, poiché l'angolo di campo della camera è suddiviso in una griglia di pixel.
  I *photosite* all'interno del sensore corrispondono uno a uno con i pixel dell'immagine digitale quando viene prodotta: sono disposti in una matrice rettangolare che può essere immaginata come una scacchiera, dove i quadrati sono talmente piccoli che, se visti da lontano, ingannano l'occhio e il cervello umano nel pensare che l'immagine sia a tono continuo.
  Ogni pixel contiene dei numeri, che specificano la posizione nella griglia e la luminosità dei tre canali di colore rosso, verde e blu: la mescolanza di tali colori, a causa del modo in cui l'occhio e il cervello li percepiscono, consente di ricreare tutto l'arcobaleno.

- Nel tono, poiché i valori di luminosità sono discreti: se sono presenti abbastanza campioni, l'immagine viene percepita come una rappresentazione fedele alla scena originale.
  A causa del funzionamento del sistema visivo umano, se i toni continui vengono divisi in un numero sufficiente di passaggi discreti, è possibile ingannare l'occhio nel pensare che sia un tono continuo, anche se non lo è.

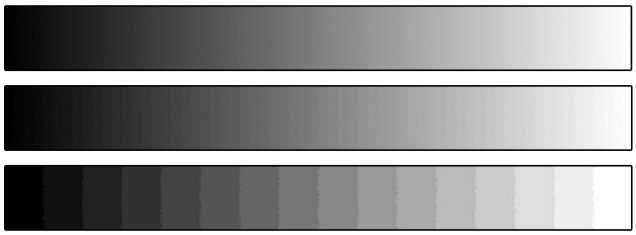

Figura 3.11: Esempio di scala di grigi a 256, 64 e 16 bit, fonte: astropix

- nel tempo, poiché l'esposizione ha una durata finita.

Il file digitale risultante è una raccolta di numeri che rappresentano i valori di posizione e luminosità per ogni cella della griglia: a seconda del sensore utilizzato, è possibile definire alcune caratteristiche principali.

- Velocità di acquisizione dei frame: caratteristica intrinseca del sensore, è un valore costante;
- Tempo di esposizione: varia in base all'ambiente di applicazione del dispositivo, ad esempio scene buie richiedono tempi di esposizione maggiori rispetto ad altre luminose;
- Range dinamico: intervallo di valori tale per cui il sensore rappresenta fedelmente l'ambiente; è un rapporto adimensionale e viene espresso in *dB* per permettere la rappresentazione di un ampio intervallo;
- FPS: è il numero di *frame* visualizzati per ogni istante di tempo, dipende dalla velocità di acquisizione del sensore.

3.6 Camere ad eventi

Chiamate anche sensori di visione dinamica, sono particolari tipologie di videocamere che permettono di misurare, per ogni pixel, in maniera asincrona, la variazione di luminosità ad esso legata.
Un evento dunque è una capsula contenente una coppia di coordinate, un tempo e la polarità ad esso associata.
Sinteticamente: $E(x,y,t,p)$

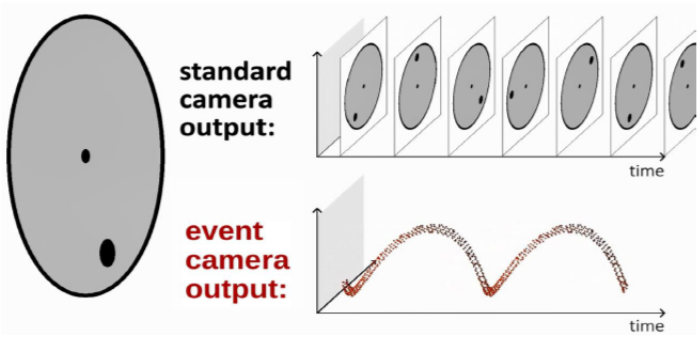

Figura 3.12: Confronto tra il flusso di output di camere frame-based ed event-based, fonte: Scaramuzza

Poniamo ora l'attenzione in un contesto industriale, dove è necessaria la presenza di sensori per l'analisi di:

- Movimenti rapidi: se la velocità di esposizione è minore della velocità dell'oggetto in questione, si manifesta perdita di informazione;
- Relativi a oggetti piccoli: vengono acquisite informazioni relative all'ambiente circostante, dunque vengono conservati dati di non interesse;
- Posti in condizioni di scarsa luminosità: i tempi di esposizione risultano elevati, con conseguente rallentamento nell'elaborazione delle informazioni.

Da queste considerazioni, si comprende la necessità di utilizzare strumenti differenti, in grado di garantire la completezza dell'informazione e la raccolta dei soli dati di interesse con una latenza bassa.
Queste caratteristiche sono largamente soddisfatte dalle *event-cameras*, poiché:

- Le camere ad eventi attualmente in commercio hanno risoluzioni temporali dell'ordine dei microsecondi, garantendo una latenza molto bassa ed un range dinamico di 120dB contro i 60dB delle camere *frame-based*;
- Le camere ad eventi memorizzano esclusivamente dati relativi alle variazioni, dunque il contenuto statico, ossia l'ambiente circostante, non viene preso in considerazione.

Ciò consente di risparmiare in termini di spazio di archiviazione e ottenere una maggiore velocità di elaborazione.
Le camere ad eventi vengono utilizzate in molteplici campi applicativi, quali:

- Internet of Things (IoT), grazie al basso consumo energetico;
- Guida autonoma, grazie alla bassa latenza e le ridotte dimensioni di *storaging*;
- Realtà aumentata e virtuale, grazie alla bassa latenza ed al ridotto consumo energetico;
- Automazione industriale, grazie alla velocità nell'elaborazione delle informazioni.

Name	**Event output**	**Frames**	**Color**	**Imu**	**Manufacturer**
DVS128	Polarity	No	No	No	Inivation
sDVS128	Polarity	No	No	No	CSIC
DAVIS240	Polarity	Yes	No	Yes	Inivation
DAVIS346	Polarity	Yes	No	Yes	Inivation
SEES	Polarity	Yes	No	Yes	Insightness
Samsung DVS	Polarity	No	No	Yes	Samsung
Onboard	Polarity	No	No	Yes	Prophesee
Celex	Intensity	Yes	No	Yes	CelePixel

Fonte: Wikipedia

Per lo sviluppo dell'algoritmo, è stato utilizzato un file con estensione *.aedat4* prodotto dalla camera ad eventi *DAVIS346*, la quale consente di acquisire sia *frame* che *eventi*.

Capitolo 4

Strumenti e librerie software

4.1 Linguaggio Python

Python è un linguaggio di programmazione ad alto livello multi-paradigma: il *binding* variabile-tipo è dinamico, poiché il tipo di dato viene identificato e convertito, se occorre, a *runtime*; mentre non esistono puntatori, anche se le variabili sono implementate tramite il loro meccanismo.
Il linguaggio comprende implementazioni di strutture come *liste*, *dizionari*, *tuple* e *set*.
Una caratteristica fondamentale è che tutti i tipi di dato sono oggetti, ed essendo tali, il linguaggio mette a disposizione funzionalità per la manipolazione degli stessi.
È presente un *Garbage-collector*, che consente di ottimizzare e gestire in maniera opportuna la memoria.
È possibile definire *moduli*: ossia file contenenti attributi e metodi che possono essere riutilizzati e inseriti in programmi diversi, senza possibilità di ambiguità fra le variabili del programma principale e quelle dei moduli stessi.
Python contiene una vasta libreria di funzioni, permette inoltre l'inclusione di moduli esterni disponibili nel *Python Package Index* e installabili mediante linea di comando attraverso il comando `pip install [...]`, permette l'esecuzione di *System calls* con quasi tutti i *Sistemi operativi* e l'esecuzione di codice scritto in altri linguaggi.
Alcuni ambienti di utilizzo del linguaggio sono *Framework Web* (Django, Pyramid, Flask e Bottle), supporto del protocollo *Internet* nella libreria standard per JSON, HTML, XML, FTP, IMAP e socket, *Data science*, *Machine learning* (SciPy, Pandas, IPython, NumPy,MatPlotLib) e altro ancora.

4.2 Visual Studio Code

Editor proprietario Microsoft, è compatibile con Windows, Linux e MacOs, il sorgente è disponibile su GitHub e scaricabile al seguente link selezionando la versione adatta al proprio *Sistema operativo*.
Per prima cosa, è buona norma selezionare una cartella di lavoro (*workspace*), che ospiterà i vari progetti.
Il software permette l'auto completamento del testo, mediante la tecnologia *Intellisense*, e l'aggiunta di funzionalità mediante estensioni scaricabili all'interno del *Marketplace*.
L'installazione dell'editor include l'estensione per il linguaggio Python.

4.3 SDK DV-Python

Il *Software Development Kit* (SDK) DV-Python è una libreria che consente di:

- ottenere flussi di dati, grezzi o processati, con sorgente una camera *Dynamic Vision Systems* (DV) e destinazione l'applicazione Python;
- l'apertura di file generati da dispositivi recenti, come l'estensione *.aedat4*, oppure meno recenti, come l'estensione *.aedat3*.

La libreria è piuttosto recente, perciò le funzionalità disponibili sono al momento limitate.
L'installazione ha come requisiti sia una versione minima di Python pari alla 3 che l'installazione del pacchetto *Microsoft Build Tools per Visual Studio 2017*, scaricabile per il *Sistema operativo* Windows al seguente link.
In alternativa, lo strumento è integrato nell'installazione di una versione di *Microsoft Visual Studio* pari alla 2017 o successive.
L'installazione dell' SDK avviene tramite linea di comando dall'interprete Python mediante il comando `pip3 install dv`.
Nello sviluppo dell'algoritmo, l'SDK viene utilizzato per l'apertura di un file generato dalla camera ad eventi *DAVIS346* nel più recente formato con estensione *.aedat4*.

4.4 MediaPipe

MediaPipe è un progetto open-source, disponibile su GitHub al seguente link che offre soluzioni *cross-platform* basate su *Machine-learning* per la *Visione artificiale*.
Le soluzioni comprendono:

- rilevazione di volti, pose, mani, occhi, testo ed oggetti generici;
- manipolazione di volti e tracciamento in tempo reale di movimenti relativi ad oggetti.

La soluzione di interesse, dunque utilizzata nello sviluppo dell'algoritmo, è ***FaceMesh***, compatibile con il linguaggio Python.
L'installazione avviene mediante linea di comando dell'interprete Python, con il comando: `pip install mediapipe`.
FaceMesh è una soluzione che stima 468 punti 3D di riferimento del viso in tempo reale: utilizza il *Machine-learning* per dedurre la geometria della superficie 3D, richiedendo il solo input della camera, senza necessità di un sensore di profondità.

4.4.1 ML Pipeline

La *pipeline* è composta da due reti neurali che operano contemporaneamente in real time:

- un rilevatore, che lavora sull'immagine completa e calcola le posizioni dei volti;
- un modello 3D di punti di riferimento del volto, che opera su tali posizioni e prevede la geometria approssimativa della superficie, tramite *regressione*.

Lo scopo è ottenere un volto accuratamente ritagliato, poiché un'elevata precisione riduce drasticamente il numero di trasformazioni come rotazioni, traslazioni e modifiche di scala; inoltre ciò consente alla rete di dedicare la maggior parte della sua capacità all'accuratezza della previsione delle coordinate.
Nella *pipeline*, i ritagli possono essere generati anche sulla base dei punti di riferimento del volto identificati nel *frame* precedente, in maniera tale da invocare il rilevatore solo quando il modello non è più in grado di identificare il volto.
Per i punti di riferimento del viso viene impiegato il *Machine learning* e addestrata una rete con gli obiettivi di prevedere simultaneamente le coordinate del punto di riferimento 3D sui dati sintetici renderizzati ed i contorni semantici 2D sui dati annotati del mondo reale.
La rete dei punti di riferimento 3D riceve come input un *frame* ritagliato senza profondità e fornisce in output le posizioni dei punti 3D.

4.4.2 Face Geometry Module

Il *Face Landmark Model* rileva i punti di riferimento facciali a telecamera singola nello spazio delle coordinate dello schermo: le coordinate X e Y sono normalizzate, mentre la coordinata Z è relativa e viene ridimensionata come la coordinata X sotto il modello della telecamera di proiezione prospettica debole.
Il modulo si allontana poi dallo spazio delle coordinate dello schermo verso uno spazio 3D metrico e fornisce le *primitive* necessarie per gestire il volto come un normale oggetto 3D: diventa quindi possibile proiettare la scena 3D finale nello spazio delle coordinate dello schermo con la garanzia che le posizioni dei punti di riferimento del viso non vengano modificate.
Lo spazio 3D metrico è ortonormale destrorso: all'interno di esso è presente una telecamera prospettica virtuale situata all'origine dello spazio e puntata nella direzione negativa dell'asse Z.
Nella *pipeline* si presume che i *frame* della telecamera siano osservati esattamente dalla telecamera virtuale, dunque che i suoi parametri vengano successivamente utilizzati per convertire le coordinate del punto di riferimento dello schermo nello spazio metrico 3D.

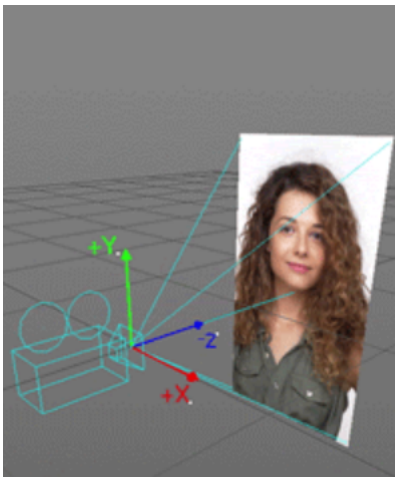

Figura 4.1: Spazio 3D metrico, fonte: GitHub

Il **Canonical Face Model** è un modello 3D statico del volto umano, che segue la topologia dei 468 punti di riferimento del *Face Landmark Model*.
Il modello ha due importanti funzioni:

- definire le unità metriche: la scala del *Canonical Face Model* definisce l'unità dello spazio Metrico 3;l'unità metrica predefinita utilizzata dal modello vale un centimetro;
- collegare gli spazi statici e di *runtime*: la *Face pose transformation matrix* è una mappa lineare del *Canonical Face Model* stimata su ciascun fotogram-

ma a *runtime*.
In questo modo, le risorse 3D virtuali modellate attorno al *Canonical Face Model* possono essere allineate con una faccia tracciata applicando loro la *Face pose transformation matrix*.

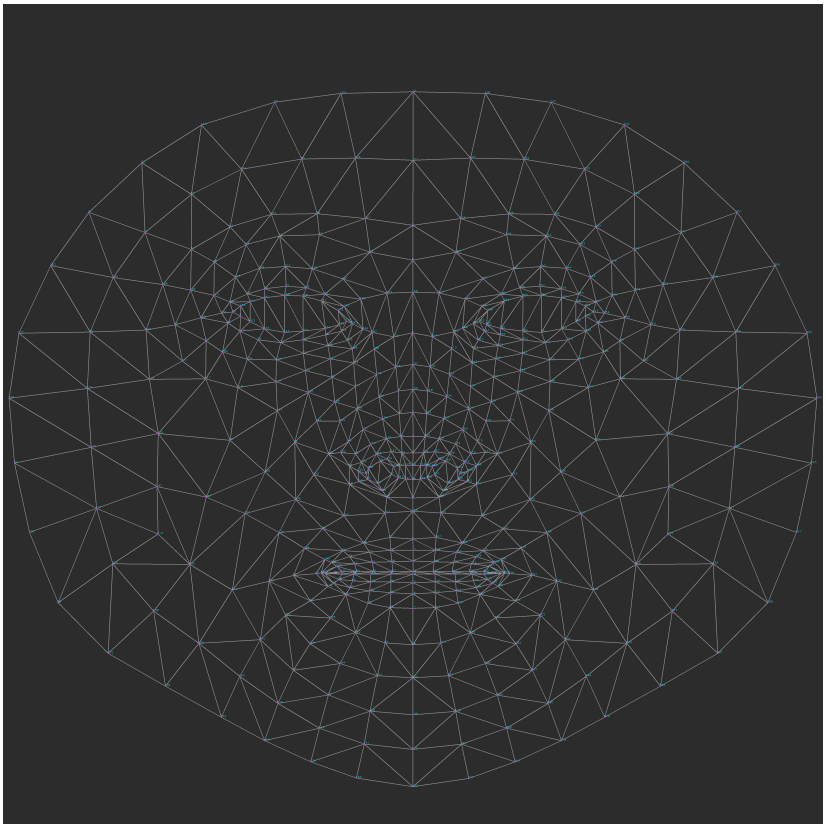

Figura 4.2: Mappa dei 468 punti di riferimento del viso, fonte: GitHub

La **Geometry Pipeline** è la componente responsabile della stima degli oggetti della geometria facciale all'interno dello spazio metrico 3D.
Per ogni *frame*, vengono eseguiti i seguenti passaggi:

- Le coordinate dello schermo relative ai punti del volto vengono convertite in coordinate dello spazio metrico 3D;
- Viene stimata la *Face pose transformation matrix*;
- viene creato un *Face mesh* utilizzando i punti di riferimento metrici calcolati come posizioni dei vertici, mentre sia le coordinate della trama dei vertici che la topologia triangolare sono ereditate dal *Canonical face model*.

Il formato della geometria facciale è definito come *Protocol Buffer message*.

Capitolo 5

Sviluppo Modulo Software

5.1 Considerazioni

Lo sviluppo dell'algoritmo è stato diviso in due fasi:

- Nella prima si è operato esclusivamente sui *frame*:
  - Applicando l'algoritmo *FaceMesh*: i risultati prodotti hanno permesso di poter estrapolare le informazioni relative ai *keypoints* appartenenti ai diversi *landmarks* per ogni *timestamp*;
  - Tali informazioni sono state poi utilizzate per calcolare il modulo della velocità e la direzione di ogni *keypoints*: inizialmente per ogni *timestamp* e, successivamente, per determinati intervalli temporali;
  - Per ultimo, si è operato esclusivamente sugli occhi, concentrandosi su determinati *keypoints* delimitati da rettangoli (*bounding boxes*), utilizzati nella successiva fase di implementazione.
- La seconda fase si è concentrata esclusivamente sugli eventi relativi agli occhi, dove si è:
  - Analizzato ogni *evento* per determinare l'appartenenza agli intervalli temporali di interesse: in caso di esito positivo, si è effettuato un ulteriore controllo per l'appartenenza ad un *bounding box*;
  - In caso di esito positivo per entrambe le verifiche, tale evento viene memorizzato in un file *.json*;
  - Per concludere, dal file *.json* precedente, sono state realizzati grafici ed animazioni delle polarità relative agli *eventi*, mostrando l'andamento degli stessi nel tempo.

5.1.1 Utilizzo SDK DV-Python

In particolare, l'SDK *DV-Python* non consente di poter eseguire funzionalità specifiche sul file *.aedat4*, dunque l'utilizzo è stato limitato nell'accesso alle informazioni contenute nelle strutture dati dello stesso.
Da una prima analisi, è stata estrapolata la struttura del file *.aedat4*, che risulta essere la seguente:

```
dvSave.aedat4 = {
        "frames": [{
                      "image": array[[[ ]]],
                      "position": (x,y),
                      "size":(x,y),
                      "timestamp_end_of_exposure": value,
                      "timestamp_end_of_frame": value,
                      "timestamp_start_of_exposure": value,
                      "timestamp_start_of_frame": value,
                      "timestamp": value
                      },
                      ...
                ],
        "imu": [{
                      "accelerometer": (x,y,z),
                      "gyroscope": (x,y,z),
                      "magnetometer":(x,y,z),
                      "temperature": value,
                      "timestamp": value
                      },
                      ...
                ],
        "events": [{
                      "polarity": True,
                      "x": value,
                      "y": value,
                      "timestamp": value
                      },
                      ...
                ],
        "triggers": [{
                      "type": value,
                      "timestamp": value
                      },
                      ...
                ]}
```

I dati di interesse sono contenuti nelle strutture *frames* ed *events*.
L'importazione della libreria e l'apertura del file *.aedat4* avvengono con le seguenti modalità:

```
from dv import AedatFile
with AedatFile(<Path to aedat file>) as f:
```

5.2 Implementazione

5.2.1 utilities

La directory *utilites* contiene i file *landmarks.py* e *functions.py*, realizzati per offrire una miglior leggibilità del codice.
Essi contengono rispettivamente tutti i *keypoints* relativi ai diversi *landmarks* e tutte le funzionalità implementate per la realizzazione dell'algoritmo.

5.2.2 json

La directory *json* contiene i file con estensione *.json* prodotti dall'elaborazione delle informazioni nei successivi algoritmi, incluse le strutture di filtraggio relative ad occhi e bocca ed i *bounding boxes* relativi agli occhi.
Relativamente ai file di filtraggio degli occhi e della bocca, la struttura risulta essere la seguente:

```
[
    {
        "type":string,
        "events":[
        {
            "timestamp": value,
            "event": string
        },
        {
            "timestamp": value,
            "event": string
        },
        {
            "timestamp": value,
            "event": string
        }]
    },
    ...
]
```

La stringa associata alla chiave *type* identifica il *landmark* in questione (*eyes* o *lips*), mentre quella relativa a *events* identifica sia intervalli random, con i valori *random1*, *random2* e *random3*, sia intervalli di apertura e chiusura di occhi e labbra, identificati da *open*, *close* e *reOpen*.
Il file contenente i *bounding boxes* relativi ai *timestamp* di interesse contiene, per ogni *timestamp*, le posizioni delle coordinate massime e minime dei *keypoints* di entrambi gli occhi. La struttura, risulta essere la seguente:

```
[
  {
    "leftEye": {
```

```
      "xMax": value,
      "xMin": value,
      "yMax": value,
      "yMin": value
    },
    "rightEye": {
      "xMax": value,
      "xMin": value,
      "yMax": value,
      "yMin": value
    },
    "timestamp": value
  },
  ...
]
```

5.2.3 faceMeshing.py

Il seguente algoritmo procede partendo dalla lettura del file *dvSave.aedat4*, analizzando ed applicando l'algoritmo *FaceMesh* su ogni*frame* rilevando i *keypoints* relativi al volto, per poi salvare le informazioni di interesse in un apposito file *.json*.
In particolare, viene analizzato ogni *frame*, controllando mediante una variabile contatore se il *frame* in questione è il primo ad essere scansionato:

- In caso di esito positivo, viene definita una variabile che contiene il *timestamp* di partenza, ovvero l'istante di tempo convertito dal formato *epoch* in secondi, moltiplicando per e^{-6}.

- in alternativa, il *timestamp* viene convertito come nel caso precedente, al cui risultato viene sottratto il valore del *timestamp* di partenza precedentemente definito.

Successivamente, per poter applicare le funzionalità della libreria *cv2*, si è resa necessaria la conversione del *frame* da scala di grigi in RGB: da tale immagine si è poi creata una copia sulla quale applicare l'algoritmo *FaceMesh*.
La presenza di risultati in *FaceMesh* viene controllata mediante l'analisi della struttura *results.multi_face_landmarks*: se vuota, non si è rilevato nessun volto. In caso contrario si verifica, per ogni elemento del risultato, il *landmark* di appartenenza, proseguendo con il salvataggio dei dati in apposite strutture.
Al termine della scansione del *frame*, viene mostrato a video un immagine contenente il *frame* copiato, il *timestamp* ed i *keypoints* del volto.
I dati relativi ai *keypoints* dei vari *landmarks*, vengono salvati nel file *dvSaveFrames.json*

```
apertura file dvSave.aedat4
```

```
    apertura modulo faceMesh
        per ogni frame
            if count == 0:
                conversione timestamp base epoch -> secondi
                count = 1
            conversione il timestamp
            conversione immagine in RGB
            applicazione algoritmo FaceMesh
            Se ci sono risultati:
                determino landmark di appartenenza del keypoint
                scrittura keypoint sull'immagine
                scrittura timestamp sull'immagine
                scrittura a video dell'immagine
                aggiunta dei dati alla struttura
scrittura dati in dvSaveFrames.json
```

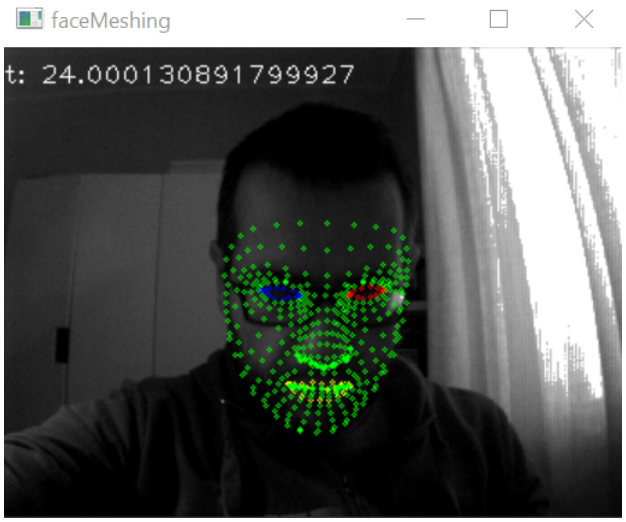

Figura 5.1: Applicazione dell'algoritmo *FaceMesh*.

La struttura del file *dvSaveFrames.json* è del tipo:

```
[
  {
    "leftEye": [
      {
        "x": value,
        "y": value
      },
      ...
    ],
     "lips": [
      {
        "x": value,
        "y": value
```

```
      },
      ...
    ],
      "rightEye": [
      {
        "x": value,
        "y": value
      },
      ...
    ],
    "timestamp": value,
  },
      ....
]
```

5.2.4 faceProcessing.py

Il seguente algoritmo ha il compito di processare le informazioni contenute nel file *dvSaveFrames.json*, calcolando per ogni *timestamp*, la direzione ed il modulo della velocità di ogni *keypoint* relativo alla bocca e agli occhi.
In particolare, vengono dapprima calcolate le dimensioni dei tre *tensori* (uno per ogni *landmark*) che ospiteranno i dati presenti nel file:

- La prima dimensione rappresenta il numero di elementi presenti nel file, che corrisponde al numero di *timestamp* (e quindi di *frame*) presenti;
- La seconda dimensione rappresenta il numero di coppie di *keypoints* del *landmark* in questione;
- La terza ed ultima dimensione rappresenta il numero di elementi che identificano le coordinate.

Una volta assegnati i valori ai *tensori* si prosegue calcolando, per ogni *timestamp*, per ogni *landmark*, per ogni coppia di *keypoint*, i valori del modulo della velocità (mediante il metodo delle *Differenze centrali*) e della direzione (calcolata come $\arctan(\frac{Vy}{Vx})$.
I risultati vengono mostrati a video mediante dei grafici, e salvati nel file *dvSaveDerivatives.json*.

```
apertura file dvSaveFrames.json
    estrapolazione lista con timestamp
    valorizzazione tensori
    calcolo delle velocità e direzioni
    scrittura informazioni in dvSaveDerivatives.json
stampa a video dei grafici
```

La struttura del file *dvSaveDerivatives.json* risulta essere la seguente:

```
[
  {
    "leftEye": [
      {
        "direction": value,
        "modSpeed": value
      },
      ...
      ],
      "lips": [
      {
        "direction": value,
        "modSpeed": value
      },
      ...
      ],
      "rightEye": [
      {
        "direction": value,
        "modSpeed": value
      },
      ...
      ],
      "timestamp": value
      },
      ...
]
```

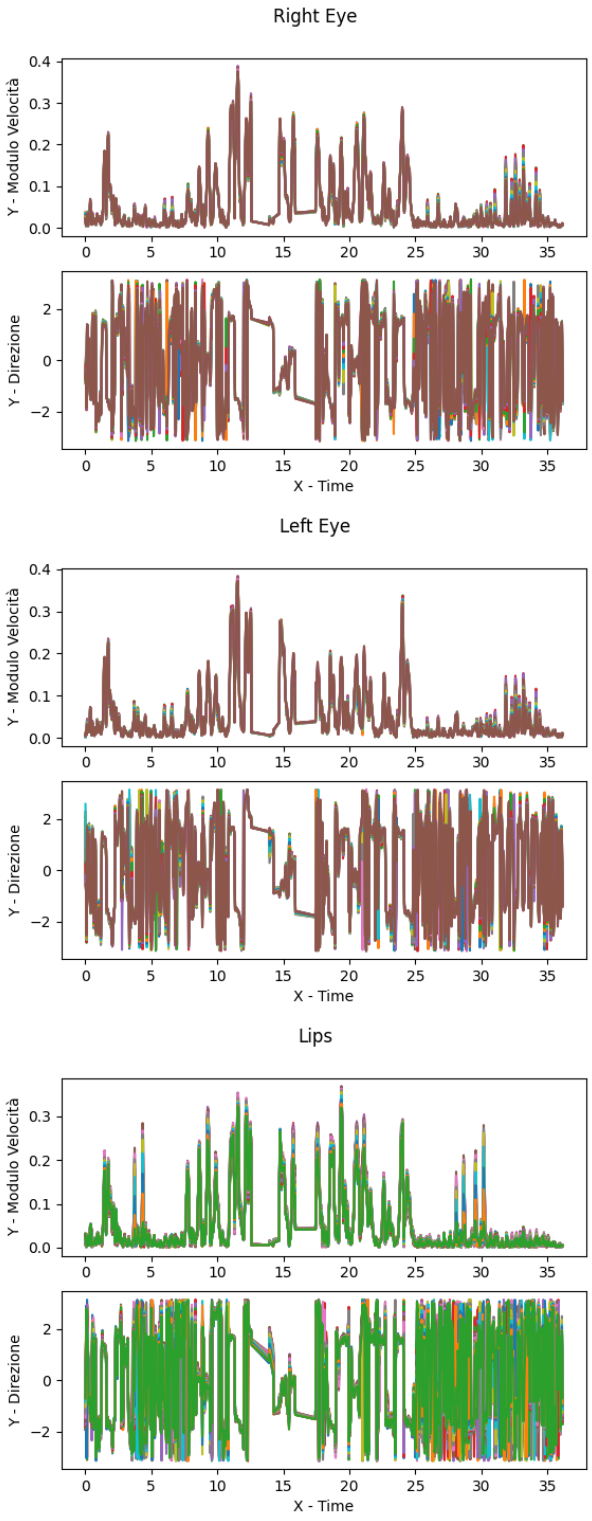

Figura 5.2: Velocità e direzione dei *keypoints* relativi all'occhio destro, occhio sinistro e labbra.

5.2.5 filteredProcessing.py

Il seguente algoritmo è una versione ridotta di *FaceProcessing* che ha il compito di processare le informazioni contenute nel file *dvSaveFrames.json*, **limitatamente** agli intervalli temporali contenuti nei file *analyzeEye.json* e *analyzeLips.json*.
Tali file contengono intervalli temporali relativi sia a movimenti di apertura e chiusura delle labbra e degli occhi, sia randomici.
Per far ciò, si è operato definendo una lista di *timestamp* estratti manualmente dal file di interesse, per poi proseguire come nel caso precedente.
Si scansionano i *frame* verificando, mediante un filtro, se il *timestamp* in questione appartiene o meno alla alla lista dei *timestamp* ridotti.
In caso affermativo, allora la cella del *tensore* in questione viene valorizzata, altrimenti il suo valore permane, come in fase di inizializzazione, allo zero.
Successivamente vengono calcolati i valori del modulo della velocità e della direzione come nel caso precedente, con l'accortezza di escludere dal calcolo la frontiera per evitare risultati inconsistenti.
Analogamente, vengono mostrati gli andamenti nel tempo relativi al modulo della velocità e alla direzione, oscurando le parti di non interesse mediante appositi meccanismi.
Le informazioni vengono salvate nel file *dvSaveReducedDerivatives.json*.

```
1 liste con intervalli di interesse
2 apertura file dvSaveFrames.json
3     valorizzazione tensori
4     calcolo delle velocità e direzioni negli intervalli di
          interesse
5     scrittura informazioni nel file dvSaveReducedDerivatives.json
6 apertura file dvSaveReducedDerivatives.json
7     valorizzazione tensori
8     filtraggio dei valori di non interesse
9     stampa a video dei grafici
```

Si osserva che la struttura del file *dvSaveReducedDerivatives.json* risulta essere la medesima del file *dvSaveDerivatives.json*, con la differenza che negli intervalli di non interesse i valori del modulo della velocità e della direzione sono posti a zero.

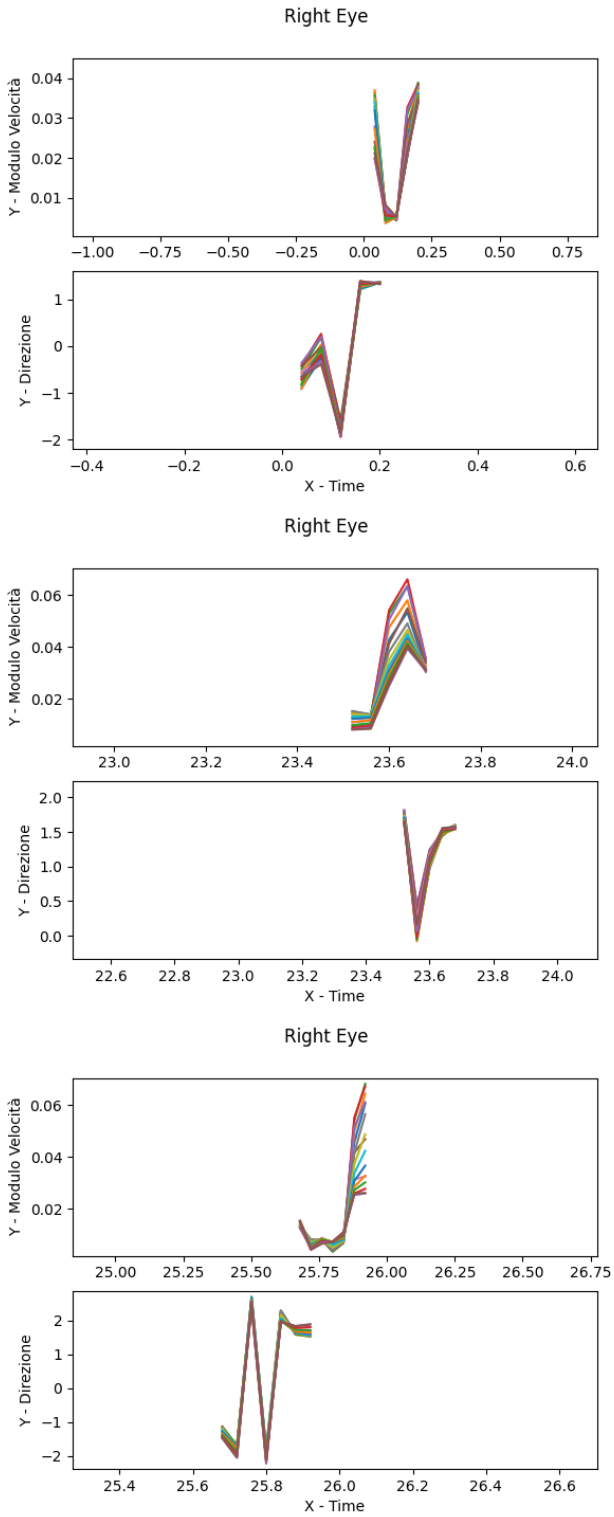

Figura 5.3: Velocità e direzioni dei *keypoints* filtrate relative all'occhio destro (1).

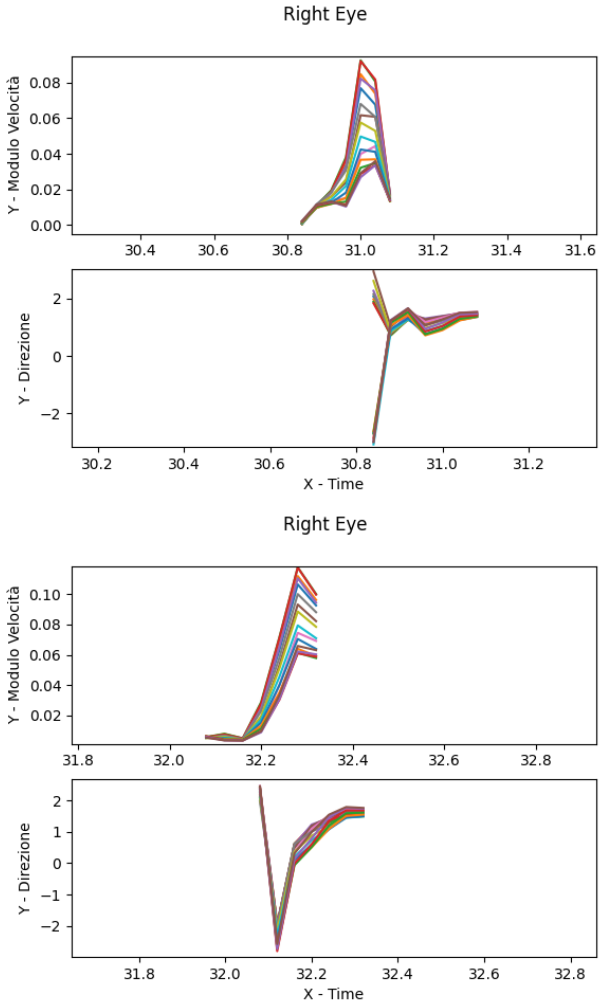

Figura 5.4: Velocità e direzioni dei *keypoints* filtrate relative all'occhio destro (2).

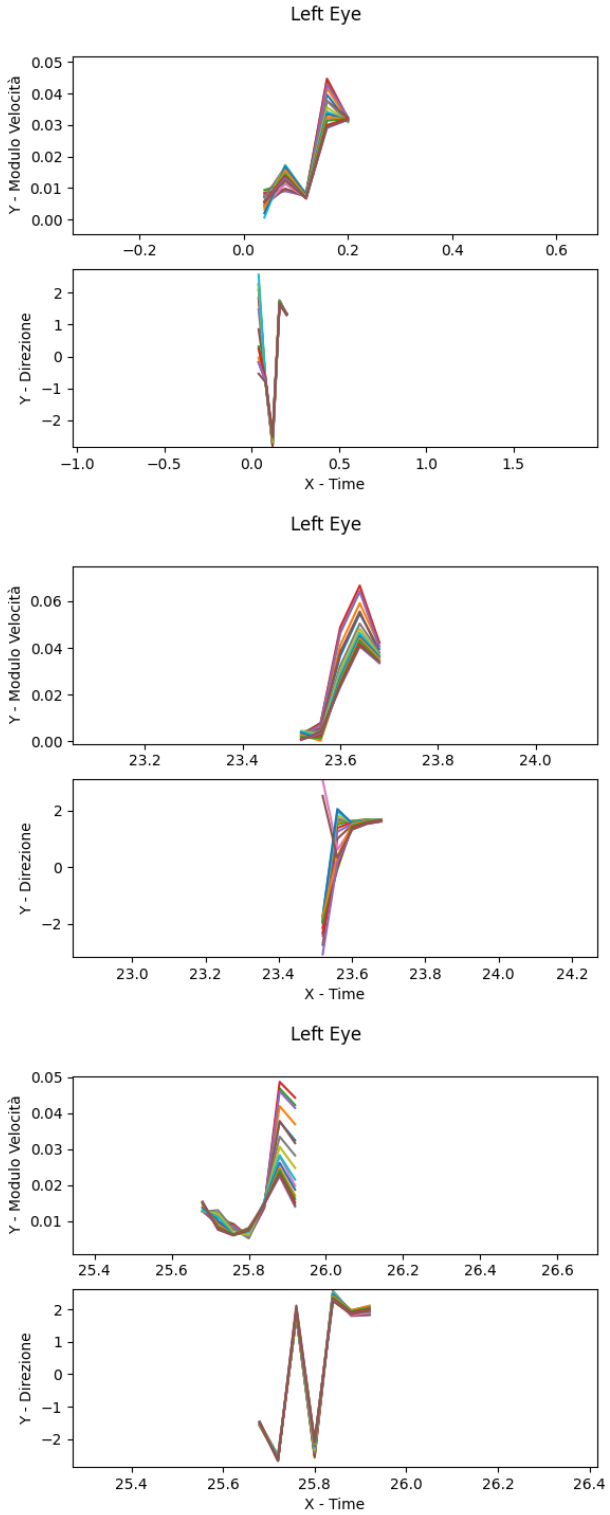

Figura 5.5: Velocità e direzioni dei *keypoints* filtrate relative all'occhio sinistro (1).

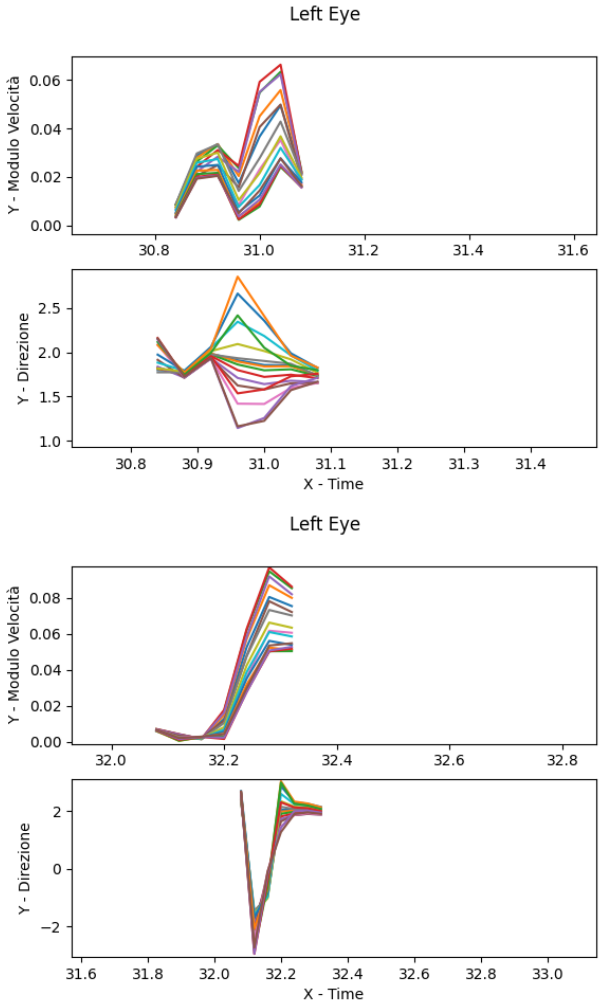

Figura 5.6: Velocità e direzioni dei *keypoints* filtrate relative all'occhio sinistro (2).

5.2.6 eyeMeshing.py

Il seguente algoritmo è diviso in due parti:

- La prima opera scansionando ogni *frame*, convertendo i *timestamp* ed estraendo i *keypoints* relativi esclusivamente agli occhi.
  Viene mostrato a video un risultato analogo a *faceMeshing.py*, con la differenza che:
  - i *keypoints* mostrati riguardano esclusivamente gli occhi;
  - gli occhi sono circondati dai *bounding boxes*.
    Le coordinate di tali rettangoli vengono calcolate, per ogni *frame*, combinando i valori massimi e minimi di tutti i *keypoints*.

  Successivamente si verifica se il *timestamp* del *frame* attuale appartiene alla lista dei *timestamp* ridotti, estratti precedentemente dal file *analyzeEyes.json*: in caso affermativo sia *keypoints* che *bounding boxes* del *frame* in questione vengono estratti e salvati in apposite strutture.
  Infine, le informazioni vengono salvate rispettivamente nei file *dvSaveEyesFrames.json* e *eyesBoundingBoxes.json*.

- La seconda parte opera sugli *eventi* estratti mediante una funzionalità messa a disposizione dall'*SDK* per aumentare la *produttività*.
  In particolare, si scansiona ogni *evento* e si verifica se:
  - il valore del *timestamp* (convertito e normalizzato rispetto al valore base del *timestamp* del primo *frame*) appartiene ad uno degli intervalli temporali contenuti nel file *analyzeEyes.json*;
  - In caso di esito positivo, si procede verificando che le coordinate dell'*evento* siano contenute in uno dei *bounding boxes*;
  - In caso affermativo, allora tale *evento* viene salvato nel file *dvSaveEyesEvents.json*.

```
1  apertura file dvSave.aedat4
2      apertura modulo FaceMesh
3          per ogni frame:
4              if count == 0:
5                  timestamp di base da epoch -> secondi
6                  count = 1
7              conversione timestamp
8              applicazione algoritmo FaceMesh
9              se ci sono risultati:
10                 estrapolazione e scrittura a video dei keypoints
                       e dei bounding boxes
11                 se ci sono risultati filtrati:
12                     salvataggio informazioni ridotte
```

```
scrittura dei file dvSaveEyesFrames.json e eyesBoundingBoxes.json
apertura file dvSave.aedat4
    per ogni evento
        estrapolazione timestamp
        se l'evento appartiene agli intervalli di interesse:
            se l'evento appartiene ad un bounding box:
                salvo informazioni
scrittura dati in dvSaveEyesEvents.json
```

Figura 5.7: Occhi delimitati dai bounding boxes.

La struttura del file *dvSaveEyesFrames.json* risulta essere la medesima del file *dvSaveFrames.json*, con la differenza che i dati salvati riguardano esclusivamente gli occhi.
Il file *dvSaveEyesEvents.json* contiene *polarità* rappresentate con un valore booleano, *timestamp*,*type* (che identifica occhio destro o sinistro) e le coordinate dello stesso; in sintesi si ha una struttura del tipo:

```
[
  {
    "polarity": boolean,
    "timestamp": value,
    "type": string,
    "x": value,
    "y": value
  },
  ....
 ]
```

5.2.7 eyeProcessing.py

Il seguente algoritmo mostra sia un'animazione delle *polarità* relative agli *eventi* contenuti nel file *dvSaveEyesEvents.json*, sia un grafico sintetizzante tale andamento nel tempo.
Inizialmente, ipotizzando che gli scostamenti degli occhi tra un *frame* ed il successivo siano trascurabili, si è operato calcolando due soli *bounding boxes* valutando tutti i valori massimi e minimi delle coordinate relative ai *keypoints* di entrambi gli occhi: tali rettangoli, distinti, ospiteranno i valori delle *polarità* associati agli *eventi*.
Successivamente, si è inizializzata una matrice con dimensioni analoghe ad ai *frame*, per poi valorizzarla nelle seguenti modalità:

- Nella fase iniziale, si è operato fissando un intervallo di accumulazione pari a 0.5 secondi per fornire una condizione di partenza accettabile nell'animazione;
- Tale finestra di accumulazione consiste nell'aggiungere, a partire da un *timestamp* associato ad un *evento*, una quantità tale per cui tutti gli eventi con *timestamp* minore del limite preimpostato vengano mostrati simultaneamente nell'animazione; una volta superato, il limite viene ricalcolato e l'iterazione continua.
- A seguito dell'accumulazione iniziale, tale finestra è stata modificata con un valore pari a 5 millisecondi, in maniera tale da poter osservare con fluidità le variazioni delle polarità;
- Osserviamo che ad ogni iterazione, viene controllato se il contatore associato all'*evento* supera il numero di elementi contenuti nel file, in maniera tale da poter arrestare l'esecuzione in caso di termine anticipato.
- Se la *polarità* associata all'*evento* vale uno, allora la cella della matrice corrispondente incrementa il suo valore di uno; altrimenti decrementa di uno.

La seconda parte, infine, riassume il tutto mostrando a video l'andamento complessivo di tali *polarità*.

```
1 Estrapolazione valori massimi e minimi bounding boxes
2 apertura file
3     controllo del contatore
4         Se contatore = 0:
5             accumulazione iniziale
6         controllo del contatore durante l'accumulazione
7             calcolo valore cella della matrice
8             controllo se l'evento appartiene ad un bounding box
9             Se si:
```

```
                somma dei valori del bounding
                salvataggio timestamp
            incremento contatore durante l'accumulazione
        animazione delle polarità
        incremento contatore
grafico complessivo delle polarità
```

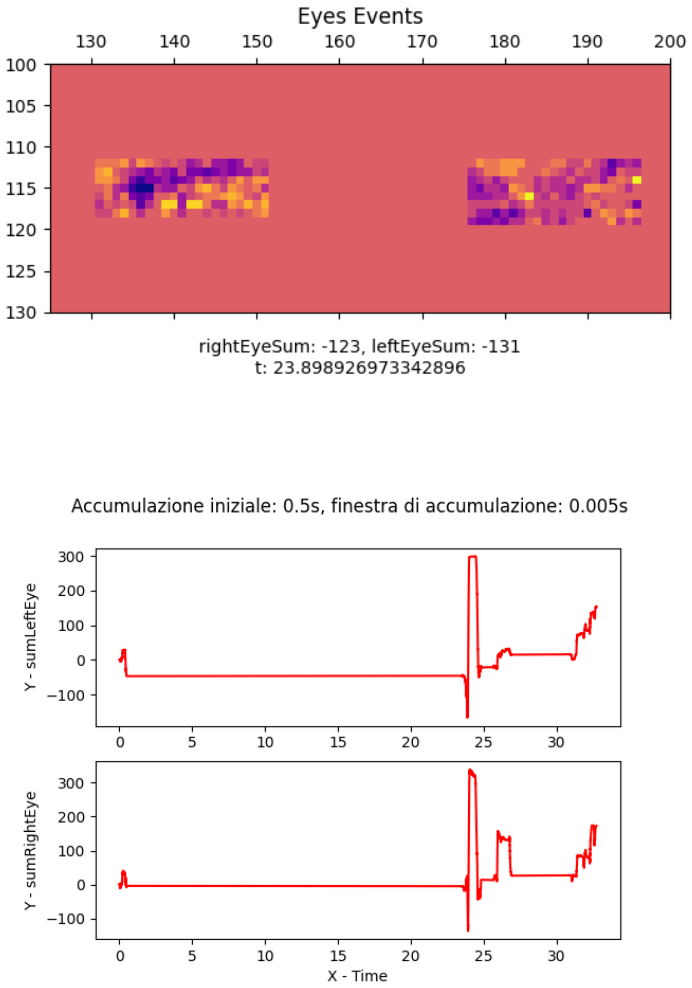

Figura 5.8: Animazione e andamento delle polarità relative agli eventi dell'occhio.

Capitolo 6

Conclusioni

6.1 Conclusioni e sviluppi futuri

L'algoritmo, come precedentemente osservato, estrapola da un file con estensione *.aedat4* sia *frame* che *eventi*.
Partendo dai *frame*, è stato applicato l'algoritmo *FaceMesh*, il quale ci ha consentito di poter determinare i *keypoints* e dunque le coordinate relative agli occhi, disegnare i loro contorni (*bounding boxes*) ed infine costruire un meccanismo di filtraggio, utilizzato in seguito per il processamento degli *eventi*.
Il risultato finale è stato mostrare sia un'animazione delle *polarità* relative agli*eventi* di interesse, sia un grafico sintetizzante l'andamento delle stesse nel tempo, distinguendo tra occhio destro e sinistro.
Un importante limite di tale applicazione, è il dover necessariamente analizzare gli *eventi* relativi agli occhi partendo dai *frame*: non è possibile trascurarli e quindi operare direttamente sugli *eventi*.
Una volta risolta tale problematica, un'interessante futura applicazione potrebbe coinvolgere il rilevamento in tempo reale, tramite camera ad *eventi* posta sotto al volante di un automobile, dei segni di cedimento o distrazione del guidatore tramite analisi:

- gli occhi, analizzando movimenti di chiusura prolungati, in modo tale da poter rilevare situazioni tali per cui il guidatore accusi sonno improvviso;
- la bocca, analizzando movimenti di apertura prolungata della bocca, per la rilevazione dello sbadiglio;
- l'intero volto, analizzando ad esempio i movimenti relativi ai *keypoints* esterni, per rilevare se il conducente è distratto da situazioni esterne e non abbia il focus sulla strada davanti a sé.

6.2 Ringraziamenti

Oggi, al termine di questo percorso lungo e tortuoso, desidero ringraziare tutte quelle persone che hanno contribuito a far sì che tutto ciò potesse avverarsi.
In primo luogo, ringrazio la mia famiglia, per il supporto fondamentale fornito durante tutto il periodo universitario.
Ricordo benissimo le crisi attraversate durante il primo anno, e senza di loro questo giorno non sarebbe mai arrivato: hanno sempre creduto in me, spesso e volentieri quando nemmeno io sono stato capace.
Ringrazio i colleghi universitari ed i coinquilini, per aver addolcito questi tre anni, alternando periodi di studio con momenti di puro svago e divertimento che porterò sempre dentro di me.
Ringrazio gli amici, con i quali avrei voluto trascorrere più tempo, ma ciò purtroppo non è stato sempre possibile a causa degli impegni universitari; anche a distanza di settimane, avevo sempre la certezza che si trovassero nel solito posto ad aspettarmi.
Ringrazio i compagni del ballo, per avermi accolto nel loro gruppo facendomi riscoprire una passione, dopo un lungo periodo di interruzione.
Ringrazio la professoressa Ciccanti, colei che mi ha fatto appassionare all'informatica durante le scuole superiori.
Infine, ringrazio il mio relatore Adriano Mancini per avermi concesso la possibilità di svolgere il seguente elaborato sotto la sua guida e supervisione, fornendomi costante supporto e repentina disponibilità per ogni evenienza.

Bibliografia

[1] Rapporto uomo-tecnologia https://scuola.repubblica.it/lombardia-bergamo-iisserafinoriva/2012/05/08/il-rapporto-tecnologia-uomo/.

[2] Intelligenza artificiale debole e forte https://www.intelligenzaartificialeitalia.net/post/intelligenza-artificiale-debole-e-forte.

[3] Reti neurali https://www.ibm.com/it-it/cloud/learn/neural-networks#toc-come-funzi-mMuAWTuZ.

[4] Macchina pensante https://www.andreaminini.com/ai/una-macchina-puo-pensare#cos'%C3%A8_una_macchina?

[5] Deep learning https://www.intelligenzaartificiale.it/deep-learning/#Cos8217e_il_deep_learning.

[6] https://www.intelligenzaartificiale.it/machine-learning/.

[7] Tipologie intelligenza artificiale https://medium.com/atobit/i-diversi-tipi-di-intelligenza-artificiale-d942e17d6b64.

[8] Machine learning https://www.oracle.com/it/data-science/machine-learning/what-is-machine-learning/.

[9] Clustering https://www.geeksforgeeks.org/clustering-in-machine-learning/.

[10] Apprendimento rinforzato https://www.andreaminini.com/ai/machine-learning/apprendimento-con-rinforzo.

[11] Regressione e classificazione https://www.eage.it/machine-learning/la-differenza-tra-regressione-e-classificazione.

[12] Computer vision https://www.lorenzogovoni.com/computer-vision/.

[13] Adam Nowosielski. Vision-based solutions for driver assistance. *Journal of Theoretical and Applied Computer Science*, 8:35–44, 11 2014.

[14] Immagini frame based
https://www.astropix.com/html/astrophotography/how.html.

[15] Gallego, Delbruck, Bartolozzi Orchard, Taba, Censi, Leutenegger, Davison, Conradt, Daniilidis, and Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[16] Python https://opensource.com/resources/python.

[17] Python feature
http://www.helldragon.eu/marcello/galli_python/03-Uso.html.

[18] Visual studio code https://www.ilsoftware.it/articoli.asp?tag=Visual-Studio-Code-cos-e-e-come-funziona_19189.

[19] Libreria dv-python https://gitlab.com/inivation/dv/dv-python/-/blob/master/README.md.

[20] Facemesh
https://google.github.io/mediapipe/solutions/face_mesh.html.

Elenco delle figure