UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Progettazione e Sviluppo di un algoritmo mediante eye-tracking per la determinazione del livello di attenzione di un guidatore

Design and Development of an algorithm of eye-tracking for the estimation of a driver attention level

Relatore:
Prof. Mancini Adriano

Laureando:
Pacioni Davide

Anno Accademico 2022-2023

Indice

Capitolo 1

Introduzione

Il presente lavoro di tesi tratta dello sviluppo di un'applicazione che implementa il tracciamento del volto, denominato di seguito come *head-tracking* e soprattutto del tracciamento degli occhi, o *eye-tracking*, nell'abito dei sistemi *automotive* al fine di riconoscere l'attenzione dell'utente alla guida.

Una particolare attenzione, infatti, va riservata a tutte quelle situazioni in cui vi sono elementi di distrazione del conducente, come ad esempio l'utilizzo del cellulare o conversazioni con altri passeggeri, i quali possono causare drastici cali di attenzione, oppure ai casi di sonnolenza durante la guida. Secondo il report 2022 del Ministero delle Infrastrutture e dei Trasporti la distrazione, il mancato rispetto delle regole di precedenza o del semaforo e la velocità troppo elevata sono le prime tre cause di incidente (escludendo il gruppo residuale delle cause di natura imprecisata). I tre gruppi costituiscono complessivamente il 40% dei casi.[1].

Al fine di prevenire eventuali incidenti stradali, negli ultimi anni l'attenzione di chi sviluppa la sicurezza degli autoveicoli si è estesa dai sistemi di sicurezza passiva (airbag, cinture di sicurezza, resistenza agli urti, ecc), già consolidati, ai sistemi avanzati di sicurezza attiva, pensati principalmente per prevenire situazioni di pericolo e incidenti[2]. Questi ausili elettronici vengono indicati con l'acronimo ADAS auto cioè *Advanced Driver Assistance Systems*, e con questa sigla si identificano tutti i dispositivi presenti a bordo delle nostre auto per incrementare il comfort di guida e i livelli di sicurezza.[3] Tra i vari sistemi esistenti che vengono installati sulle auto troviamo un sistema di allerta dell'attenzione, utile anche a prevenire casi di colpi di sonno alla guida, basato su una valutazione degli atteggiamenti del conducente attraverso l'elaborazione dei movimenti del capo e della direzione dello sguardo.

Attraverso dunque l'utilizzo di alcuni sensori montati sul cruscotto è possibile rilevare situazioni in cui il conducente è distratto ed emettere di conseguenza un segnale visivo-acustico al fine di riportare l'attenzione del conducente sulla guida del mezzo. Al fine di rilevare queste situazioni viene utilizzata una tecnologia che sfrutta il tracciamento degli occhi, o *eye-tracking*.

Il tracciamento degli occhi è il processo di misurazione del punto di osservazione (dove si sta guardando) o del movimento di un occhio rispetto alla testa. Il dispositivo che si occupa di misurare le posizioni degli occhi e il movimento degli occhi viene chiamato *eye-tracker*. La ricerca su *eye-tracking* risale al diciottesimo secolo, quando Louis Emile Javal studiò il movimento degli occhi durante la lettura. Edmund Huey fu il pioniere nella costruzione del primo strumento di *eye-tracking*, ossia delle lenti connesse ad un puntatore di alluminio[4].

Essi sono utilizzati nella ricerca sul sistema visivo come dispositivo di input per l'interazione uomo-computer e sempre più utilizzati per applicazioni riabilitative e *assistive* (correlate, ad esempio, al controllo di sedie a rotelle, bracci robotici e protesi). Esistono diversi metodi per misurare il movimento degli occhi. La variante più popolare utilizza immagini video da cui viene estratta la posizione dell'occhio.

In media l'essere umano esegue 3-5 movimenti oculari al secondo e questi movimenti sono fondamentali per aiutarci a gestire la grande quantità di informazioni che incontriamo nella nostra vita quotidiana. Negli ultimi anni, grazie allo sviluppo della tecnologia di *eye-tracking*, c'è stato un crescente interesse per il monitoraggio e la misurazione di questi movimenti, al fine di capire come ci occupiamo ed elaboriamo le informazioni visive che incontriamo[5]. Questo processo trova applicazioni in una vasta gamma di ambiti, ad esempio in applicazioni commerciali: nell'ultimo decennio c'è stata una rapida crescita nelle applicazioni commerciali della tecnologia *eye-tracking* per valutare l'efficacia degli sforzi di marketing visivo. I movimenti oculari sono strettamente accoppiati con l'attenzione visiva, il che li rende indicatori eminenti del processo di attenzione visiva nascosta[6]. In ambito *automotive* invece la ricerca sullo sviluppo di *eye-tracker* studia la riduzione del costo dei *tracker* ad infrarossi esistenti e l'aumento della loro precisione. La maggior parte degli *eye-tracker* disponibili in commercio usa sensori ad infrarosso per tracciare la direzione dello sguardo dell'utente. In ogni caso l'accuratezza del tracciamento in differenti condizioni ambientali è un problema non indifferente. La tecnica più comunemente utilizzata è basata sul centro della pupilla e sulla tecnica di riflessione della cornea. Il *tracker*, che ha un sensore ad infrarosso, illumina l'occhio e cattura l'immagine. L'infrarosso riflette una piccola quantità di luce proveniente dalla cornea, in questo modo durante il processamento dell'immagine si riesce ad individuare il centro della pupilla. Il *tracker* può dunque calcolare dove la persona sta rivolgendo il suo sguardo basandosi sulla posizione relativa del centro della pupilla e sulla riflessione della cornea[4].

1.1 Obiettivi

Con il presente lavoro di tesi viene utilizzata una *Intel Realsense D415* come strumento di *eye-tracker.* I sensori di tale dispositivo, che comprendono una

camera RGB, un sensore di profondità e due sensori ad infrarosso, vengono sfruttati utilizzando la libreria *realsense2*, utilizzata per manipolare il video, che verrà in seguito elaborato da GazeSense[7], un software che consente di tracciare il movimento e la direzione degli occhi utilizzando una semplice *webcam* o un sistema di rilevazione tridimensionale, integrabile in qualsiasi piattaforma e dispositivo. Inoltre fornisce in tempo reale i dati relativi alla posizione della testa, alla direzione dello sguardo, ci permette di rilevare il *blinking* dell'occhio e, con una opportuna configurazione, quale "regione" dell'ambiente circostante l'utente sta osservando. I dati subiscono infine un'ultima elaborazione allo scopo di mostrare graficamente all'utente il risultato dell'elaborazione, utilizzando la libreria *OpenCV*[8].

1.2 Struttura della tesi

La funzionalità principale del progetto di tesi è quella di sviluppare un sistema che riconosca, attraverso la direzione dello sguardo, su quale "oggetto" l'utente sta prestando attenzione e, eventualmente, segnalare una situazione di pericolo dovuta a distrazione. Inizialmente si recupera il **feed* video* dal sistema di visione posizionata sul cruscotto in modo tale da riprendere il volto del guidatore. Il **feed** viene processato attraverso l'utilizzo del **Software Development Kit* GazeSense* precedentemente citato, il quale effettua i calcoli necessari a rilevare la direzione dello sguardo del guidatore. Infine, attraverso la descrizione degli oggetti che il guidatore osserva, come ad esempio cruscotto, finestrini, specchietto e parabrezza, viene rilevato il grado di attenzione.

Il presente lavoro di tesi è strutturato nei seguenti capitoli:

- introduzione;
- strumenti utilizzati;
- sviluppo del progetto;
- conclusioni e sviluppi futuri.

Capitolo 2

Eye-tracking e Gaze-tracking

Il tracciamento degli occhi (*Eye-tracking*) è il processo di misurazione del punto di osservazione (dove si sta guardando) o del movimento di un occhio rispetto alla testa. Il dispositivo che si occupa di misurare le posizioni degli occhi e il movimento degli occhi viene chiamato *eye-tracker*. Essi sono utilizzati nella ricerca sul sistema visivo come dispositivo di input per l'interazione uomo-computer e sempre più utilizzati per applicazioni riabilitative e assistite (correlate, ad esempio, al controllo di sedie a rotelle, bracci robotici e protesi). Esistono diversi metodi per misurare il movimento degli occhi. La variante più popolare utilizza immagini video da cui viene estratta la posizione dell'occhio.

2.1 Sviluppi sullo studio dell'Eye-tracking

Gli studi sul movimento degli occhi iniziano nel diciannovesimo secolo, quando Louis Émile Javal osservò che durante l'operazione di lettura di un documento, gli occhi non scorrevano in maniera fluida sul foglio, bensì effettuavano una serie di piccole pause, chiamate *fixation* e delle veloci saccadi, ovvero rapidi movimenti degli occhi dovuti all'esplorazione della scena visiva, permettendo di portare nuovi oggetti sulla fovea, acquisendo così informazioni[9]. Il comportamento relativo all'esplorazione visiva si riferisce alla raccolta delle informazioni visive attraverso la coordinazione dei movimenti di: occhi, testa e corpo[10].

2.2 Tipologie di tracker

Sono tre i principali modelli di *eye-tracker,* e si ottiene una diversa ottimizzazione dei risultati in base alla natura della ricerca che si vuole svolgere. Questi strumenti permettono di calcolare la posizione dell'occhio e dove il nostro sguardo si focalizza a seconda dell'interfaccia che ci viene mostrata. In questo modo si riesce a studiare in dettaglio il comportamento visivo e tutti i movimenti oculari[11].

2.2.1 Tracking desktop (screen based)

È la tipologia di *tracker* più comune. Lo schermo è la fonte principale di interesse, ed il *tracker* riesce a seguire il movimento degli occhi entro alcuni limiti particolari chiamati *headbox*. Il soggetto in questo caso viene posizionato davanti ad alcune immagini specifiche, e attraverso l'utilizzo di software viene calcolato il punto di interesse dell'utente in un determinato istante di tempo. In dettaglio, un *Eye Tracker Screen Based* (ETSB), funziona nel seguente modo[12]:

- Feature point selection: durante questa fase vengono registrate dimensione del viso e posizione degli occhi. La "mappa" ottenuta durante questa fase viene utilizzata dal software per registrare nel modo più accurato possibile la posizione degli occhi unica per ogni utilizzatore;

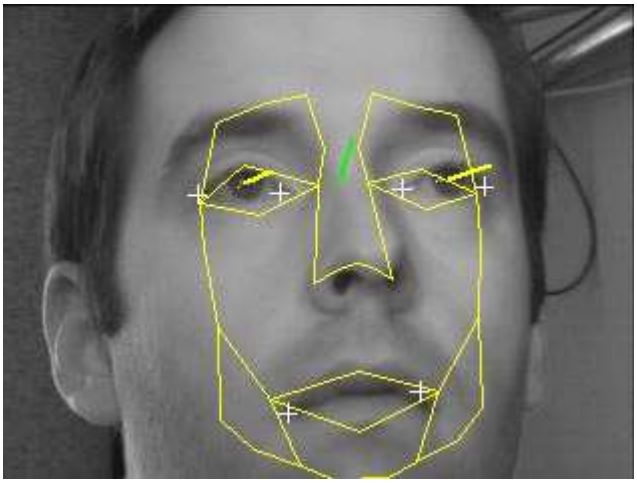

Figura 2.1: Esempio di Feature Point Selection

Fonte: [13]

- *Calibration*: durante questo passaggio vengono visualizzati alcuni punti in varie posizioni sullo schermo. Questo procedimento serve ad estrarre le immagini degli occhi e addestrare un processo gaussiano che rappresenta la mappatura tra l'immagine di un occhio e la posizione sullo schermo;

- *Tracking*: dopo che tutti i punti di calibrazione sono stati elaborati, il processo gaussiano produce una distribuzione predittiva, in modo che il punto di messa a fuoco dell'occhio previsto sul monitor del display possa essere stimato in base a una nuova immagine dello stesso. Le immagini dell'occhio vengono estratte (utilizzando i riferimenti inseriti durante la prima fase del processo) e lo sguardo viene previsto usando le immagini dell'occhio estratte e il processo gaussiano addestrato.

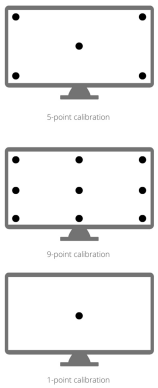

Figura 2.2: Esempio procedimento di Calibrazione

Fonte: [14]

2.2.2 Tracker indossabile (wearable)

È considerata la tipologia più dinamica, in quanto consente di studiare il comportamento visivo utilizzando ad esempio dei semplici occhiali. Essi sono dotati di telecamera ad infrarossi per registrare dove lo sguardo viene fissato durante la presentazione di alcuni stimoli visivi. Inoltre può essere indossato per periodi di tempo prolungati e non comportano restrizioni ai movimenti, preservando la visione periferica. Può essere utilizzato per studiare le performance in situazioni diverse, che vanno dalla cognizione dell'ambiente circostante alla concentrazione su un singolo oggetto[15].

Figura 2.3: Esempio Tracker indossabile

Fonte: [16]

2.2.3 Tracking webcam

Questo strumento utilizza, oltre ad avanzati software, anche la *webcam* del computer. Essa registra il movimento degli occhi ed è in grado di processare questo movimento traducendolo in una mappa di calore comunemente conosciuta come *heatmap* (Fig. 2.4. La mappa di calore mostra le parti più calde dove il nostro occhio si è concentrato maggiormente per comprendere quale parte dell'immagine mostrata ha catturato maggiormente l'attenzione. Gli *eye-trackers* a *webcam* dunque non hanno sensori o telecamere speciali, e non funzionano basandosi sul principio della riflessione corneale del centro della pupilla, ma lavorano attraverso il dispositivo *webcam* collegato o incorporato in un computer, con il supporto di uno specifico software.

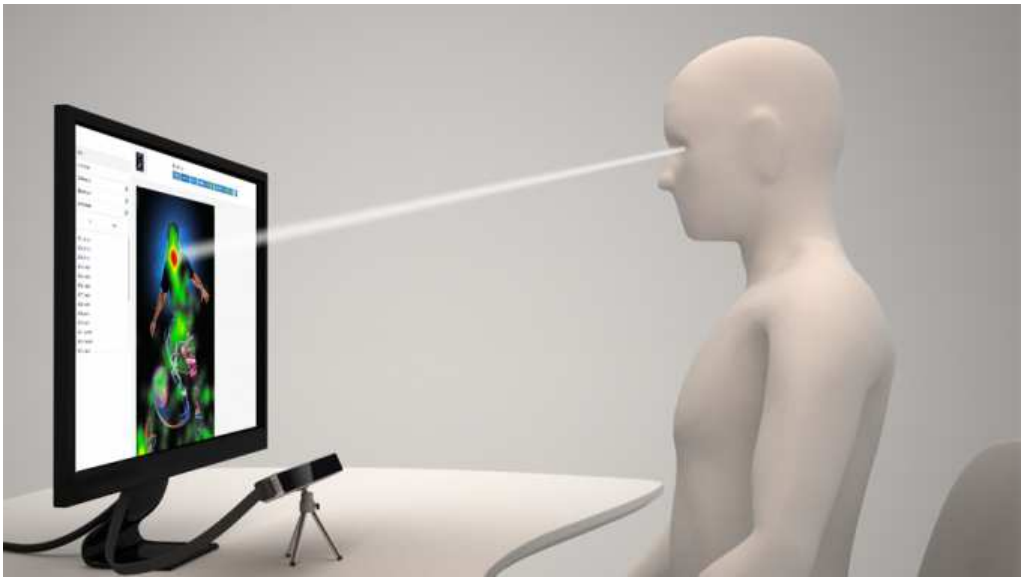

Figura 2.4: Rappresentazione della *Heatmap*

***Fonte:** [7]*

2.3 Tecnologie e tecniche utilizzate

Esistono diversi algoritmi che permettono ad un calcolatore di identificare gli occhi di un essere umano. L'identificazione degli occhi in un'immagine o in un video è basata su *modelli oculari*. Un *modello oculare* ottimale dovrebbe essere sufficientemente accurato, così da adattarsi alla variabilità dell'aspetto degli occhi, oltre ad essere efficiente dal punto di vista computazionale. Di fatto non è scontato realizzare questi modelli, in quanto esistono differenti fattori quali l'apertura degli occhi, la variabilità delle dimensioni della posa della testa e della

riflettività e dell'occlusione dell'occhio da parte delle palpebre. Inoltre bisogna considerare le condizioni di luce, l'etnia del soggetto e altri fattori ambientali[17].

2.3.1 Tecniche basate sulla forma dell'occhio (Shape-Based)

Generalmente utilizzata per applicazioni semplici di tracciamento degli occhi. Questi modelli definiscono le forme dell'iride e della pupilla, oltre all'ellissi che delinea l'intero occhio. Tipicamente queste tecniche utilizzano un modello ellittico per descrivere l'occhio, da cui viene poi estratto il bordo della pupilla utilizzando tecniche come l'*edge detection*. La limitazione principale di questa tecnica è il dover indicare, almeno inizialmente, la regione di spazio intorno all'occhio, onde evitare i cosiddetti falsi positivi, ovvero l'identificazione errata di una forma simile all'occhio umano.

2.3.2 Tecniche basate sulle caratteristiche (Feature-Based)

Queste tecniche si basano sull'identificazione e l'utilizzo di un set di caratteristiche uniche dell'occhio umano. La caratteristica principale per la localizzazione degli occhi è quella della riflessione della cornea. Generalmente queste tecniche vengono spesso utilizzate in ambienti *indoor*, in quanto producono risultati soddisfacenti, mentre nei casi *outdoor* le performance sono limitate. La pupilla e l'iride sono generalmente le parti più scure dell'immagine o del video di riferimento, e sono quindi considerate come un punto di riferimento affidabile per il tracciamento degli occhi. Si sviluppa dunque un algoritmo per rilevare le le pupille semplicemente cercando l'area più scura dell'immagine. Questa tecnica, in caso di scarsa luminosità o in caso il soggetto sia distante dal dispositivo di acquisizione e quindi si abbia differenti parti dell'immagine con diverse aree scure, si riscontrano performance discutibili, pertanto viene richiesto che il soggetto sia ben illuminato e che sia abbastanza vicino al dispositivo di acquisizione. Tuttavia per superare queste limitazioni, in particolare quella sull'illuminazione, viene usato generalmente un emettitore *IR* (infrarosso)

2.4 Differenza tra Eye-tracking e Gaze-tracking

Banalmente con l'*eye-tracking* si fa riferimento al tracciamento della posizione dell'occhio all'interno di un immagine o di un video, mentre con il termine *gaze tracking* si fa riferimento al tracciamento dello sguardo, quindi del punto di attenzione del soggetto. La Figura mostra il tipico schema di un occhio. L'obiettivo del *gaze tracking* è quello di identificare il punto di attenzione, detto *Point of Regard* (POG) dell'utente. A questo scopo sono molto importanti i movimenti

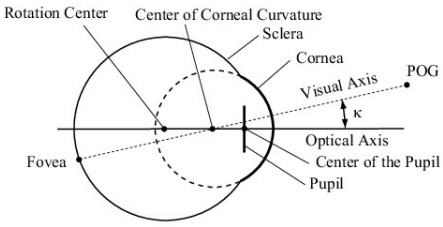

Figura 2.5: Struttura dell'occhio umano

Fonte: [18]

degli occhi, quali le saccadi e i *fixation*. Inoltre è importante conoscere alcuni parametri necessari per effettuare l'operazione di calibrazione. La calibrazione è un processo in cui vengono definiti i parametri necessari per il tracciamento, in particolare è necessario effettuare:

- calibrazione geometrica per determinare la posizione dei dispositivi quali fonti di luce e dispositivi di acquisizione delle immagini;
- calibrazione della fotocamera per recuperare i parametri caratteristici della stessa

Capitolo 3

Strumenti utilizzati

In questo capitolo vengono approfonditi gli strumenti utilizzati durante lo sviluppo dell'intero progetto.

3.1 Intel Realsense D415 Depth Camera

La Intel Realsense D415 (Fig. 3.1)è un sistema di fotocamere che ha due moduli principali: un sensore RGB che cattura l'immagine a colori e un modulo di profondità, o modulo *Depth*, che utilizza due sensori (*Left* e *Right imager*) per calcolare la profondità (Fig. 3.2).

Figura 3.1: Camera RGB-D Intel RealSense D415

Fonte: [19]

Inoltre è presente un proiettore ad infrarossi per aumentare la precisione delle rilevazioni in scenari di scarsa illuminazione. I due sensori catturano la scena e inviano i dati dell'immagine al processore integrato che calcola il valore della profondità per ogni pixel dell'immagine[20] (Fig 3.3).

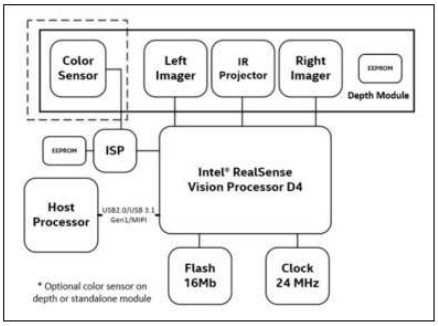

Figura 3.2: Struttura Intel RealSense D415

Fonte: [19]

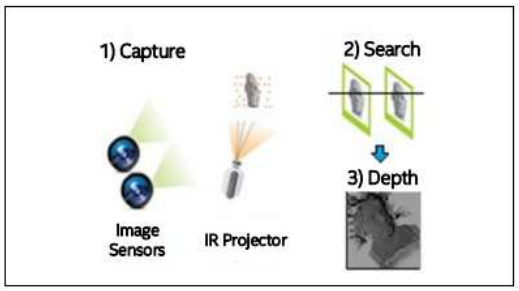

Figura 3.3: Funzionamento sensori IR

Fonte: [21]

3.2 GazeSense

GazeSense è un software per il tracciamento della testa e degli occhi sviluppato in linguaggio C++, e fornisce la codifica dello sguardo in tempo reale senza la necessità di indossare occhiali e traccia la posa della testa e lo sguardo degli occhi in 3D (Fig. 3.4). Utilizzando fotocamere con rilevamento della profondità standard, consente di tracciare più soggetti, etichettare oggetti di interesse nello spazio aperto e tracciare l'attenzione verso di loro in tempo reale[22]. Per questo progetto è stata utilizzata inizialmente la versione grafica del software, il quale ha fornito immediatamente una panoramica delle potenzialità del prodotto, successivamente si è optato per l'utilizzo dell'SDK (*Software Development Kit*), in modo da avere un controllo più granulare sul comportamento dell'applicazione per adattarla al meglio alle specifiche di progetto.

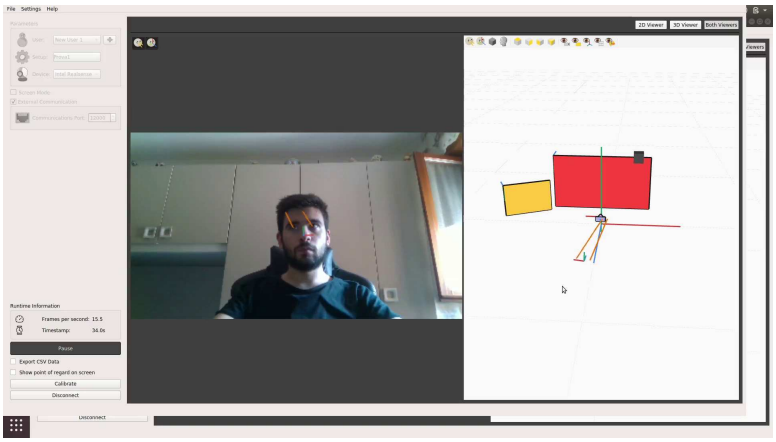

Figura 3.4: Software GazeSense

3.3 RealSense Viewer

Intel RealSense Viewer è un software incluso in Intel RealSense SDK 2.0, ed è stato utilizzato allo scopo di accedere velocemente ai sensori della fotocamera, registrare e riprodurre i flussi video e configurare le impostazioni della fotocamera (Fig. 3.5). In particolare sono stati utilizzati i sensori di profondità ed RGB per estrarre la sequenza di immagini che compongono il video, le quali sono state successivamente elaborate dal software GazeSense per il *tracking* degli occhi e del volto[23].

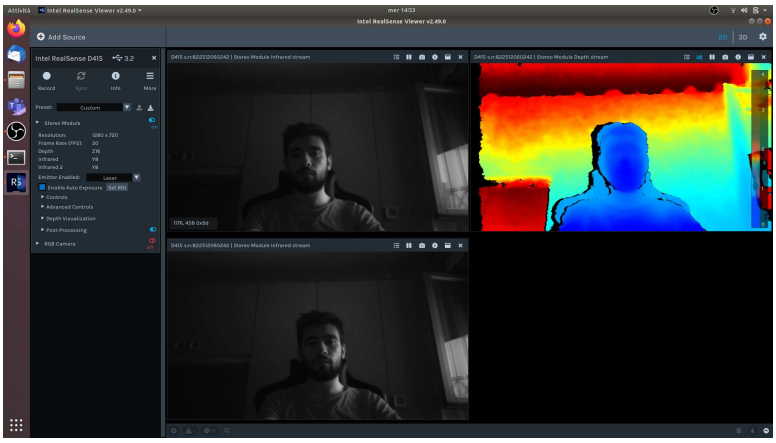

Figura 3.5: Software RealSense Viewer

3.4 Librerie

3.4.1 OpenCV

OpenCV o *OpenSource Computer Vision Library* è una libreria che nel progetto è stata utilizzata per la visualizzazione in tempo reale del video elaborato con i dati del *tracking*.

3.4.2 librealsense2

Libreria incluso in Intel RealSense SDK 2.0, insieme al software RealSense Viewer citato in precedenza, ed è stata utilizzata all'interno del programma per definire la configurazione della fotocamera e i diversi parametri, tra cui la risoluzione e quanti frame recuperare al secondo, di configurazione della fotocamera. Permette inoltre di recuperare i parametri caratteristici della fotocamera, in modo da ottenere una precisione maggiore in fase di elaborazione delle immagini da parte del software GazeSense.

3.5 Linguaggi Utilizzati

Il principale linguaggio di programmazione utilizzato in questo lavoro è stato C++. C++ è un linguaggio di programmazione *general purpose* sviluppato da Bjarne Stroustrup nel 1983 ed è nato come evoluzione del linguaggio C, aggiungendone la programmazione object-oriented. È tutt'oggi uno dei linguaggi di programmazione più utilizzato, accompagnato da C, C#, Python, Java e JavaScript.

3.6 Visual Studio Code

Per lo sviluppo del progetto è stato utilizzato l'editor di sviluppo VisualStudio Code (Fig. 3.6), al fine di sviluppare il codice C++ dell'applicazione. Si tratta di un IDE (*Integrated Development Environment*) *cross-platform*, dunque compatibile con i sistemi operativi Windows, MacOS e Linux, ed è possibile sviluppare con esso in qualsiasi linguaggio. Essendo un IDE ha anche funzionalità quali supporto per il *debug* e integrazione con i principali software di *versioning*, come ad esempio Git.

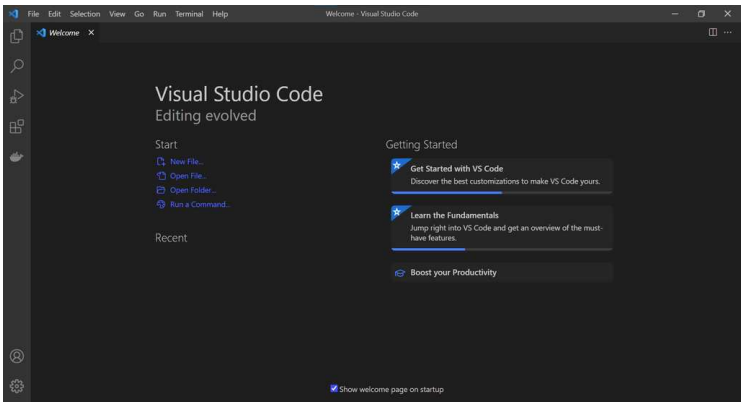

Figura 3.6: IDE Visual Studio Code

Capitolo 4

Sviluppo del progetto

Utilizzando gli strumenti visti nel capitolo precedente si presenta in questa sezione lo sviluppo del progetto. L'obiettivo, come già introdotto, è quello di implementare un meccanismo che rilevi l'attenzione del guidatore e che ci fornisca informazioni su cosa sta osservando in particolare.

4.1 Setup iniziale

Essendo il progetto sviluppato in ambito desktop, si è cercato di simulare e di effettuare test nella maniera più simile possibile ad un utilizzo in ambito *automotive*. Inizialmente, al solo fine di verificare il corretto funzionamento di tutte le componenti e di effettuare test più concisi, viene decisa una configurazione contenente uno schermo, che in questo caso rappresenta il monitor del computer, ed una camera posizionata centralmente nella parte inferiore del monitor, come illustrato in Fig. 4.1 Sempre per motivi di semplicità, la camera viene posizionata in

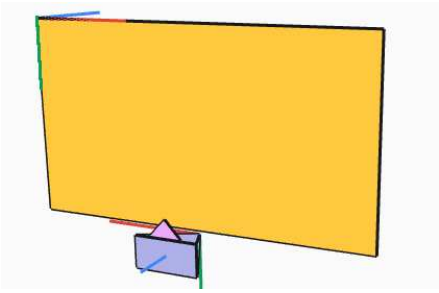

Figura 4.1: Disposizione iniziale schermo e camera

maniera esattamente perpendicolare al bordo inferiore dello schermo, senza alcun angolo di inclinazione o rotazione. Avendo ora posizionato gli elementi hardware del progetto in maniera corretta, si è provveduto a verificarne il funzionamento utilizzando il software *RealSense Viewer*, il quale ha confermato il corretto funzionamento di tutti i sensori presenti nella camera *Intel RealSense D415* come si

può vedere dalla Fig. 4.2 Inoltre attraverso l'utilizzo di questo software è possi-

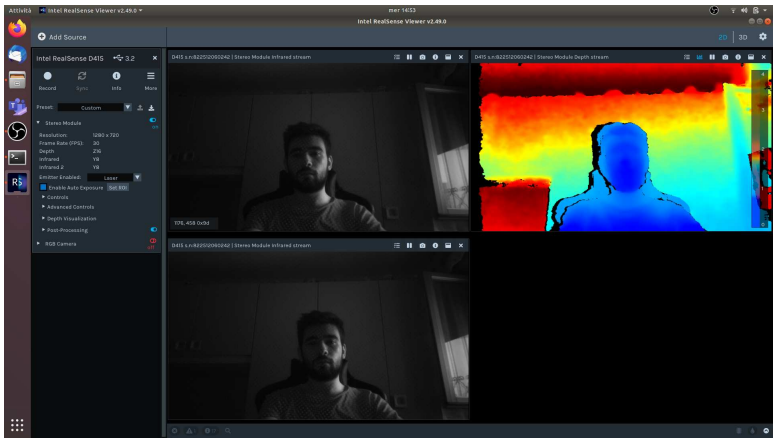

Figura 4.2: RealSense Viewer

bile provare differenti configurazioni della camera, come ad esempio la potenza dell'emettitore IR, ecc...

4.2 Test GazeSense GUI

Una volta concluso il *setup* iniziale ho sperimentato il software *GazeSense*, basato sull'omonimo SDK. L'applicazione permette di misurare il grado di attenzione dell'utente in tempo reale attraverso la combinazione di fattori quali il riconoscimento del punto e della direzione in cui l'utente sta ponendo il suo sguardo e del grado di apertura degli occhi, e conoscere di conseguenza se l'utilizzatore non stia ponendo l'attenzione su un punto di particolare rilevanza o se stia, ad esempio, per abbassare lo sguardo a causa della stanchezza. Attraverso l'interfaccia grafica è possibile inserire gli oggetti che compongono l'ambiente circostante, incluse le relative distanze, prendendo come punto di riferimento centrale la camera RealSense. Il primo test è stato effettuato prendendo in considerazione un solo schermo come illustrato in Fig. 4.3 Il test ha prodotto esito positivo, riportando correttamente la direzione dello sguardo e il punto inquadrato dello schermo. In seguito è stato possibile, attraverso l'editor grafico *built-in*, simulare l'utilizzo in ambiente *automotive*, ricreando gli elementi come: parabrezza; cruscotto; volante; specchietti etc. In questo modo è possibile conoscere verso quale elemento è rivolta l'attenzione dell'utente, in particolare è possibile dunque sapere quando, e per quanto tempo, il guidatore non sta rivolgendo la sua attenzione alla guida. Durante il tracciamento è possibile recuperare le informazioni relative agli eventi che avvengono in real-time, come il movimento della testa o la chiusura degli occhi. Un esempio delle informazioni che è possibile recuperare è mostrato in Fig. 4.4 Utilizzando delle *Network API* sviluppate in Python è possibile dunque

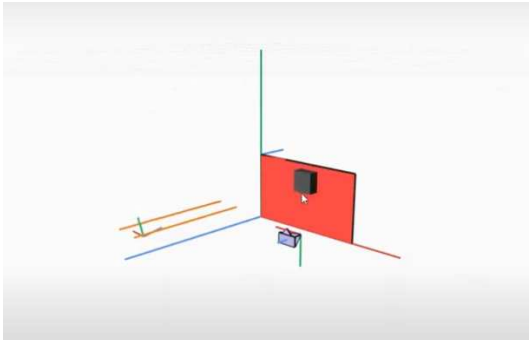

Figura 4.3: Primo test *tracking* con GazeSense GUI

User	Time	Screen_X	Screen_Y	Intersection_X	Intersection_Y	Intersection_Z	HeadPosition_X	HeadPosition_Y	HeadPosition_Z	HeadRotation_X	HeadRotation_Y
Generic	0.34	300	1439	-0.47	0.71	-0.44	-0.44	0.74	0.60	0.42	-4.03
Generic	0.44	299	1439	-0.47	0.71	-0.44	-0.44	0.74	0.59	0.22	-3.73
Generic	0.56	543	1439	-0.33	0.71	-0.44	-0.44	0.74	0.59	0.00	-3.54
Generic	0.64	569	1439	-0.31	0.71	-0.44	-0.44	0.74	0.59	-0.30	-3.28
Generic	0.72	589	1439	-0.32	0.71	-0.44	-0.44	0.74	0.59	-0.30	-3.38
Generic	0.81	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.31	-2.80
Generic	0.88	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.36	-2.88
Generic	0.95	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.37	-2.69
Generic	1.05	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.47	-2.30
Generic	1.12	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.55	-2.26
Generic	1.18	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.60	-2.49
Generic	1.25	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.68	-2.16
Generic	1.33	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.51	-2.22
Generic	1.40	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.59	-2.06
Generic	1.49	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.61	-1.96
Generic	1.53	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.37	-2.28
Generic	1.63	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.49	-1.71
Generic	1.70	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.41	-1.92
Generic	1.77	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.40	-1.74
Generic	1.83	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.42	-1.58
Generic	1.90	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.35	-1.63
Generic	1.97	-999	-999	-999.00	-999.00	-999.00	-0.44	0.74	0.59	-0.51	-1.94
Generic	2.03	504	1439	-0.36	0.71	-0.44	-0.44	0.74	0.59	-0.47	-1.66
Generic	2.10	494	1439	-0.37	0.71	-0.44	-0.44	0.74	0.59	-0.46	-1.64

Figura 4.4: Dati grezzi del *tracking*

definire un *listener* agli eventi provenienti dall'applicazione *GazeSense*. Per ogni evento lo script può effettuare un'azione come ad esempio riprodurre un suono o stampare a video qualcosa nel momento in cui si verifica un particolare evento. Per fare ciò il programma *GazeSense* ha bisogno di un *host*, generalmente il *localhost*, e una porta alla quale inviare i dati. Lo script Python registrerà dunque il *listener* sulla medesima porta per recuperare i dati. Un esempio di script Python per il recupero dei dati è il seguente:

```
1 from gazesense_client import Client, TrackerListener, TrackingEvent import time
2 class MyListener(TrackerListener): def __init__(self):
3 super().__init__()
4 def on_track_ready(self, event: TrackingEvent) -> None:
5 print("No of people tracked: ", len(event.people)) if len(event.people) < 1:
6 return
7 screen_gaze_is_lost = event.people[0].tracking_info.screen_gaze.is_lost print("
      - Lost track: ", screen_gaze_is_lost)
8 screen_gaze = event.people[0].tracking_info.screen_gaze
9 if not screen_gaze_is_lost:
10 print(" print(" print("
11 - Screen Id:
12 - X Coordinate:
13 - Y Coordinate:
14 ", screen_gaze.screen_id)
15 ", screen_gaze.screen_gaze.x) ", screen_gaze.screen_gaze.y)
16 print("=====================================================")
17 # Global Parameters
18 host = "localhost"
19 listener_port = 12000
20 # Building client and listener
```

```
client = Client(host, listener_port)
listener = MyListener() client.register_tracker_listener(listener)
# Making sure the script remains open for a long time time.sleep(100000)
```

4.3 Utilizzo dell'SDK GazeSense

Considerando l'utilizzo in ambito *automotive*, e dunque la necessità di eseguire il programma su un hardware con dimensioni ridotte e capacità computazionali limitate, si è optato per l'utilizzo dell'SDK al software "completo", essendo quest'ultimo oneroso su hardware di dimensioni ridotte. Utilizzando dunque l'SDK non è stato più necessario avere l'interfaccia grafica che gravasse sulle prestazioni, lasciando l'hardware con meno carico computazionale, oltre ad avere un grado di controllo maggiore sul comportamento dell'applicazione. La struttura dell'SDK si articola in classi, le quali ci forniscono gli strumenti e dunque i metodi per accedere alle informazioni elaborate dal software in merito al *tracking*. Vengono definite inoltre le classi inerenti al *listener*, alle informazioni del *tracking*, alla definizione dei tipi etc. Le classi che racchiudono le funzionalità principali sono tre:

- *realsense_streaming.h*: file *header* che consente di recuperare e gestire il flusso video proveniente dalla camera *Intel RealSense D415* utilizzando l'apposita libreria *librealsense2* e adattarlo all'utilizzo con *OpenCV*;
- *show_gaze.cpp*: classe contenente lo script il quale si occupa del *tracking* in tempo reale durante una sessione di guida;
- *screen_calibration.cpp*: classe che contiene lo script che effettua la calibrazione del *tracking*.

4.3.1 Recupero del flusso video dalla RealSense D415

Al fine di recuperare il video in tempo reale dalla camera *Intel RealSense D415* per poter elaborare successivamente il video, viene creata questa classe, la quale definisce utilizzando l'apposita libreria *librealsense2* le strutture dati che ospiteranno le informazioni recuperate dalla camera. Vengono innanzitutto abilitati, e quindi accesi i sensori che ci servono, come il sensore *RGB* ed il sensore di profondità *Depth*, specificandone le dimensioni, in questo caso 1920x1080 per il sensore *RGB* e 640x480 per il sensore *Depth*, oltre che il numero di frame al secondo (FPS), in questo caso 30, al fine di ottenere un tracciamento fluido. Viene dunque recuperato il flusso di immagini e la dimensione effettiva delle stesse, oltre ai parametri *intrinsics* ed *extrinsics* della fotocamera. Questi parametri sono diversi per ogni camera utilizzata e descrivono le caratteristiche della stessa.

4.3.1.1 Parametri Intrinsic

$$K = \begin{bmatrix} \alpha_x & \gamma & u_0 & 0 \\ 0 & \alpha_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

La matrice degli *intrinsic* K contiene 5 parametri *instrinsics* di una fotocamera. Questi parametri comprendono la *lunghezza focale (focal length)*, il *formato del sensore* ed il *punto principale (principal point)*. I parametri $a_x = f \cdot m_x$ e $a_y = f \cdot m_y$ rappresentano la lunghezza focale in termini di pixel, mentre m_x e m_y sono l'inverso di larghezza ed altezza di un pixel sul piano di proiezione ed f è la lunghezza focale in termini di distanza. γ rappresenta il coefficiente di distorsione tra gli assi x e y, e generalmente è 0. u_0 e v_0 rappresentano il punto principale, il quale è generalmente al centro dell'immagine.

4.3.1.2 Parametri Extrinsic

$$\begin{bmatrix} R_{3\times3} & T_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}_{4\times4}$$

R, T sono i parametri *extrinsics*, e definiscono le trasformazioni del sistema di coordinate 3D. Definiscono inoltre la posizione del centro della camera. Quando viene utilizzata una fotocamera, la luce dell'ambiente è concentrata sul piano dell'immagine e catturata. Questo processo riduce la dimensione dei dati elaborati dalla fotocamera (da 3D ad un'immagine in 2D). Ogni pixel corrisponde dunque ad un raggio di luce della scena catturata nell'immagine.

Il codice che permette di recuperare il video dalla *RealSense D415* è il seguente:

```
1  bool open() {
2          if (m_is_open) {
3              return true;
4          }
5
6          rs2::config config;
7          config.enable_stream(RS2_STREAM_COLOR, 1920, 1080, RS2_FORMAT_RGB8, 30);
8          config.enable_stream(RS2_STREAM_DEPTH, 640, 480, RS2_FORMAT_Z16, 30);
9
10         if (!config.can_resolve(m_pipe)) {
11             std::cout << "RealSense device unavailable..." << std::endl;
12             return false;
13         }
14
15         rs2::pipeline_profile pipe_profile = m_pipe.start(config);
16
17         rs2::stream_profile texture_profile_generic = pipe_profile.get_stream(
       RS2_STREAM_COLOR);
18         rs2::stream_profile depth_profile_generic = pipe_profile.get_stream(
       RS2_STREAM_DEPTH);
19
20         rs2::video_stream_profile texture_profile =
21             texture_profile_generic.as<rs2::video_stream_profile>();
22         rs2::video_stream_profile depth_profile =
23             depth_profile_generic.as<rs2::video_stream_profile>();
24
```

```
25          m_rs_texture_width = texture_profile.width();
26          m_rs_texture_height = texture_profile.height();
27
28          m_rs_depth_width = depth_profile.width();
29          m_rs_depth_height = depth_profile.height();
30
31          m_rs_texture_intrinsics = texture_profile.get_intrinsics();
32          m_rs_depth_intrinsics = depth_profile.get_intrinsics();
33
34          m_rs_texture_extrinsics = depth_profile.get_extrinsics_to(
    texture_profile);
35
36          for (rs2::sensor &sensor : pipe_profile.get_device().query_sensors()) {
37              if (sensor.is<rs2::depth_sensor>()) {
38                  rs2::depth_sensor dpt_sensor = sensor.as<rs2::depth_sensor>();
39                  m_depth_resolution_in_meters = dpt_sensor.get_depth_scale();
40                  break;
41              }
42          }
43
44          m_is_open = true;
45
46          return true;
47      }
```

Una volta completato il processo di inizializzazione della fotocamera, i frame dei due sensori vengono tradotti, dal formato realsense2 ad un formato compatibile con OpenCV, ossia vengono copiati in due strutture dati separate chiamate matrici, definite nella libreria OpenCV, in attesa di elaborazione durante la fase di calibrazione o di *tracking*.

```
1  bool read_one_frame() {
2          if (!m_is_open) {
3              return false;
4          }
5
6          rs2::frameset frameset;
7          const unsigned int timeout_ms = 100;
8          try {
9              frameset = m_pipe.wait_for_frames(timeout_ms);
10         } catch (const std::exception &e) {
11             std::cout << "Did not get frames: " << e.what() << std::endl;
12             return false;
13         }
14
15         if (!update_opencv_frames(frameset.get_color_frame(), frameset.
       get_depth_frame())) {
16             std::cout << "Problem updating OpenCV frames" << std::endl;
17             return false;
18         }
19         m_frame_idx++;
20         return true;
21     }
```

```
1  bool update_opencv_frames(rs2::frame rs_texture_frame, rs2::depth_frame
       rs_depth_frame) {
2          if (!m_is_open) {
3              return false;
4          }
5
6          // Texture frame, from realsense2 format to OpenCV
```

```
7         const int texture_width = rs_texture_frame.as<rs2::video_frame>().
      get_width();
8         const int texture_height = rs_texture_frame.as<rs2::video_frame>().
      get_height();
9         cv::Mat texture_cv = cv::Mat(cv::Size(texture_width, texture_height),
      CV_8UC3,
10                                 const_cast<void *>(rs_texture_frame.get_data()));
11        texture_cv.copyTo(m_texture_cv);
12
13        // Depth frame, from realsense2 format to OpenCV
14        const int depth_width = rs_depth_frame.as<rs2::depth_frame>().get_width
      ();
15        const int depth_height = rs_depth_frame.as<rs2::depth_frame>().
      get_height();
16        cv::Mat depth_cv = cv::Mat(cv::Size(depth_width, depth_height), CV_32FC1
      );
17        unsigned short *depth_data_short =
18            reinterpret_cast<unsigned short *>(const_cast<void *>(rs_depth_frame
      .get_data()));
19        float *out_buffer = reinterpret_cast<float *>(depth_cv.data);
20        unsigned short *in_buffer = depth_data_short;
21        for (int k = 0; k < depth_width * depth_height; k++, out_buffer++,
      in_buffer++) {
22            *out_buffer = static_cast<float>(*in_buffer);
23        }
24        depth_cv.copyTo(m_depth_cv);
25
26        return true;
27    }
```

4.3.2 Calibrazione (calibration)

Il processo di calibrazione è necessario al fine di rendere il *tracking* più preciso. Lo script consiste nel mostrare a schermo nove diversi punti, uno alla volta, a cui l'utente deve puntare il suo sguardo. Un esempio è mostrato in Fig. 4.5 Lo

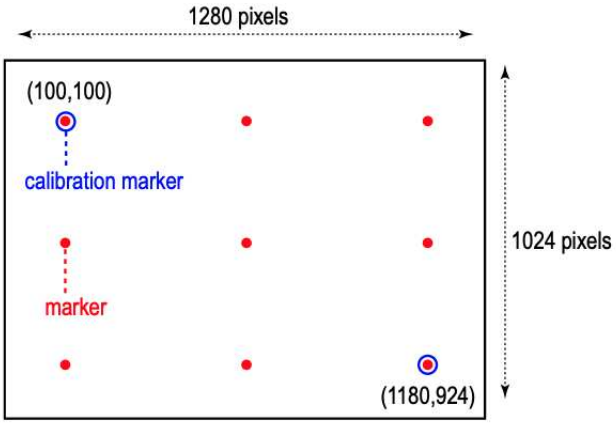

Figura 4.5: Schema del processo di calibrazione

script dunque recupera i frame precedentemente elaborati ed associa una posizione nota a priori, ossia quella del punto definito sullo schermo, la cui disposizione

è anch'essa nota a priori, alla posizione relativa dell'utente che lo sta osservando, in modo tale da aumentare la precisione del *tracking*. Dunque è necessario conoscere a priori due informazioni: la disposizione degli schermi o degli oggetti che l'utente potrebbe guardare (es. finestrini, cruscotto, specchietti ecc...) ed i punti di riferimento, in questo caso nove, per poter effettuare le dovute associazioni. Una volta terminata l'esecuzione dello script viene salvato un file contenente la configurazione finale *post-calibration*, il quale verrà utilizzato durante il *tracking* semplicemente caricando la configurazione salvata. Questo procedimento può essere svolto *una tantum* in modo tale da avere, ad esempio, una configurazione per ogni tipologia di autovettura. Basti pensare che la disposizione degli elementi caratteristici di un automobile varia per ogni modello. in questo modo non si rende necessario ripetere la calibrazione.

```
while (rs_streaming.read_one_frame()) {
        //int frame_idx = 0;
        std::cout << "Processing frame " << rs_streaming.get_frame_idx() << std
    ::endl;
        gazesense::CalibrationPoint *calibration_point = nullptr;
        gazesense::CalibrationState calibration_state;
        get_calibration_point_and_state(rs_streaming.get_frame_idx(), tracker.
    get(),
                                        &calibration_point, calibration_state);

        try {

            if (calibration_point) {
                tracker->process_frame_with_calibration(
                    rs_streaming.get_texture_cv().data,
                    rs_streaming.get_depth_cv().data, timestamp, *
    calibration_point);
            } else {
                tracker->process_frame(rs_streaming.get_texture_cv().data,
                                       rs_streaming.get_depth_cv().data,
    timestamp);
            }

        } catch (const std::exception &e) {
            std::cout << "Error processing frame: " << e.what() << std::endl;
            return EXIT_FAILURE;
        }

        /**
         * Finally, display the screen gaze and expected ground truth.
         */
        const bool screen_calibration_in_progress =
            (calibration_state != gazesense::CalibrationState::COMPLETE);
        display_screen_gaze_with_ground_truth(screen_config, listener->
    screen_gaze_info,
                                              rs_streaming.get_frame_idx(),
                                              screen_calibration_in_progress);

        timestamp += FRAME_DELTA_IN_SECONDS;
    }
```

Gli schermi oggetto di calibrazione possono descrivere qualsiasi piano del mondo reale, e a questo proposito vengono utilizzati per descrivere il parabrezza, i finestrini, gli specchietti ed il cruscotto. Per descrivere questi elementi è sufficien-

te conoscere la loro dimensione, ossia l'altezza e la larghezza, la distanza rispetto alla fotocamera e l'angolazione. Un esempio di configurazione di uno schermo è il seguente:

```
gazesense::ScreenConfig get_screen_config() {
    gazesense::ScreenConfig screen_config;

    screen_config.resolution.width = 1920;
    screen_config.resolution.height = 1080;

    screen_config.width_in_meters = 0.600000f;
    screen_config.height_in_meters = 0.3375000f;

    float alphaYawGradi=0;
    float betaPitchGradi=0;
    float gammaRollGradi=0;

    float pi = 3.141592653589793;
    //angoli in radianti
    float alpha=alphaYawGradi/180*pi;
    float beta=betaPitchGradi/180*pi;
    float gamma=gammaRollGradi/180*pi;

    screen_config.transform_scs_to_wcs.rotation[0][0] = -(cos(alpha)*cos(beta));
    screen_config.transform_scs_to_wcs.rotation[0][1] = (cos(alpha)*sin(beta)*
    sin(gamma)-(sin(alpha)*cos(gamma)));
    screen_config.transform_scs_to_wcs.rotation[0][2] = (cos(alpha)*sin(beta)*
    cos(gamma)+(sin(alpha)*sin(gamma)));
    screen_config.transform_scs_to_wcs.rotation[1][0] = sin(alpha)*cos(beta);
    screen_config.transform_scs_to_wcs.rotation[1][1] = (sin(alpha)*sin(beta)*
    sin(gamma)+(cos(alpha)*cos(gamma)));
    screen_config.transform_scs_to_wcs.rotation[1][2] = (sin(alpha)*sin(beta)*
    cos(gamma)-(cos(alpha)*sin(gamma)));
    screen_config.transform_scs_to_wcs.rotation[2][0] = -sin(beta);
    screen_config.transform_scs_to_wcs.rotation[2][1] = cos(beta)*sin(gamma);
    screen_config.transform_scs_to_wcs.rotation[2][2] = cos(beta)*cos(gamma);

    screen_config.transform_scs_to_wcs.translation.x = 0.3000000f;
    screen_config.transform_scs_to_wcs.translation.y = -0.3375000f;
    screen_config.transform_scs_to_wcs.translation.z = 0.0000000f;

    return screen_config;
}
```

Viene descritta di fatto la situazione citata precedentemente, come riportato in Figura 4.1.

Essendo la descrizione relativa ad un oggetto tridimensionale, c'è bisogno di utilizzare tre parametri per descrivere l'inclinazione dell'oggetto, denominati *Roll, Pitch* e *Yaw*, i quali descrivono l'inclinazione dell'oggetto rispettivamente per l'asse X, Y e Z, come illustrato in Fig. 4.6. Il sistema di riferimento in questione è in particolare un sistema di riferimento relativo, ossia la descrizione della posizione degli oggetti, in questo caso degli schermi, e della loro inclinazione, viene effettuata in base d un punto di riferimento ben preciso nello spazio, che in questo caso viene rappresentato dalla posizione del sistema di fotocamere. Prendendo quest'ultima come punto di riferimento assoluto nello spazio, vengono successivamente definite le posizioni e le angolazioni degli schermi, specificando di quanto siano traslati e ruotati rispetto al punto di riferimento. Per la traslazione viene

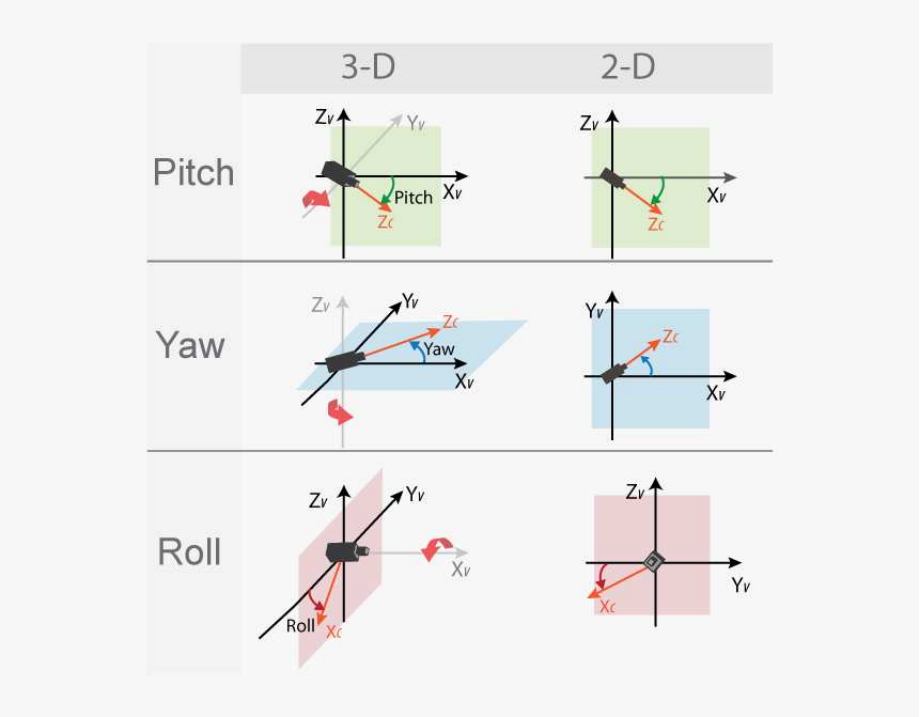

Figura 4.6: Yaw Pitch e Roll

Fonte: [24]

utilizzato un semplice valore numerico che definisce di quanto sia traslato verso sinistra o destra semplicemente utilizzando un valore rispettivamente positivo o negativo. Per descriverne l'inclinazione invece viene utilizzata la seguente *matrice di rotazione*. Essendo la matrice in funzione di 3 coefficienti α, β e γ è sufficiente implementare tre variabili che indicano il grado di inclinazione dell'oggetto espresso in gradi e successivamente convertite in radianti.

$$\begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}$$

4.3.3 Tracking

Avendo definito gli elementi chiamati *schermi* oggetto di attenzione dell'utente, lo script che esegue il *tracking* recupera i frame provenienti dalla camera e li elabora, identificando il numero di persone inquadrate nei frame ed estrapola le informazioni relative alla direzione dello sguardo, alla condizione degli occhi (aperti o chiusi) e della posizione della testa. Queste informazioni vengono raccolte dal *listener*, il quale espone le informazioni che diventano oggetto di un'ulteriore elaborazione, ad esempio se l'utente non sta osservando gli elementi come il parabrezza o il finestrino, c'è la possibilità che esso sia distratto, e dunque è possibile gestire questo evento emettendo un semplice segnale acustico.

Capitolo 5

Conclusioni

Questo capitolo riporta le conclusioni relative al progetto. Durante lo sviluppo si questo progetto ho avuto la possibilità di approfondire la conoscenza e l'utilizzo del linguaggio C++, ed in particolar modo di conoscere nuove applicazioni per tecnologie apparentemente di banale realizzazione come quella dell'*eye-tracking*. Ho inoltre potuto riscontrare attraverso le ricerche statistiche sugli incidenti stradali dovuti a distrazione o stanchezza, quanto la sicurezza stradale sia di fondamentale importanza, e quanto sia possibile apportare delle migliorie attraverso la tecnologia, al fine di rendere più sicuro l'utilizzo quotidiano di un veicolo. Infine la possibilità di relazionarmi con un ambiente esterno a quello accademico, accumulando esperienze preziose. Terminato lo sviluppo di questo progetto si è dunque realizzato un algoritmo che rileva il livello di attenzione di un utente alla guida di un mezzo attraverso l'utilizzo di un sistema di fotocamere, il quale rileva come mostrato in Fig. 5.15.25.3 la direzione dello sguardo del conducente e indica il punto su cui il conducente sta rivolgendo la propria attenzione, come ad esempio i finestrini laterali o il parabrezza. Viene inoltre emesso un segnale acustico ogni volta che il conducente tiene gli occhi chiusi per un periodo di tempo prefissato, al fine di riportare l'attenzione dell'utente verso la guida specialmente nei casi in cui venga colto da stanchezza.

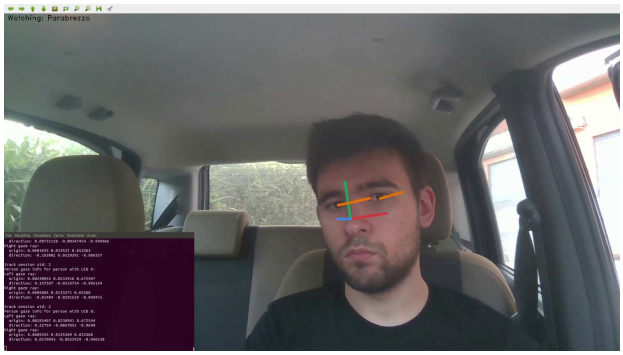

Figura 5.1: Sguardo sul parabrezza

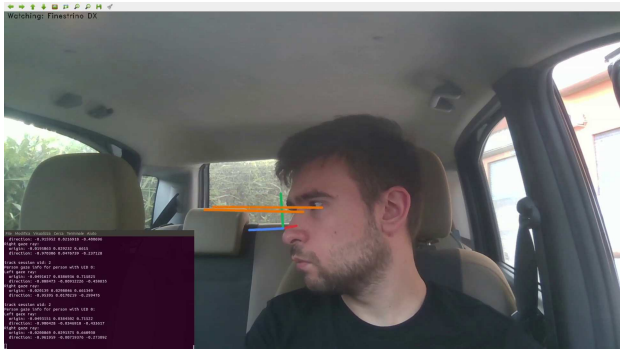

Figura 5.2: Sguardo sul finestrino destro

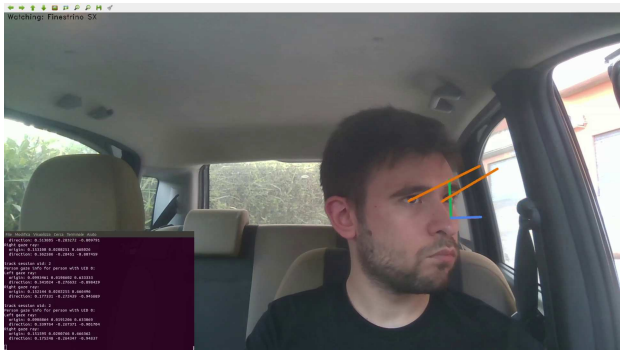

Figura 5.3: Sguardo sul finestrino sinistro

5.1 Difficoltà riscontrate

Durante lo sviluppo del progetto sono state riscontrate diverse difficoltà, sia nella fase di utilizzo del SDK GazeSense, sia nella fase di sviluppo del codice, le quali hanno reso necessario uno studio approfondito dei problemi come distorsione delle immagini ed il funzionamento di fotocamere digitali.

5.1.1 Utilizzo SDK

Nella fase di utilizzo le maggiori difficoltà riscontrate sono la comprensione del funzionamento dello stesso attraverso l'utilizzo degli esempi forniti con l'SDK, i quali fornivano inizialmente informazioni su uno specifico caso di utilizzo, ossia l'acquisizione delle informazioni relative allo sguardo e l'elaborazione delle informazioni attraverso unicamente l'utilizzo di un feed video pre-registrato. Inoltre le informazioni relative ai parametri intrinsic ed extrinsic non erano dinamici in base al sistema di fotocamere utilizzato. Pertanto è stato necessario uno studio approfondito relativo al funzionamento di una fotocamera digitale ed i problemi relativi alla distorsione dell'immagine prodotta.

5.1.2 Sviluppo del codice

Le principali difficoltà durante lo sviluppo del codice sono state l'implementazione dell'elaborazione dei dati relativi allo sguardo attraverso l'utilizzo di un flusso di dati provenienti da un video in tempo reale utilizzando il sistema di fotocamere Intel RealSense, ed attraverso l'utilizzo di apposite librerie, l'importazione dei parametri corretti per la tipologia di fotocamere utilizzate. Inoltre l'incorretto processo di calibrazione iniziale che impediva di ottenere risultati consistenti. La corretta definizione degli elementi che compongono lo spazio circostante, come il parabrezza ed i finestrini, descrivendoli in base ad un sistema di riferimento tridimensionale.

5.1.3 Limiti di utilizzo

I limiti per cui è possibile ottenere dei risultati incorretti sono principalmente stati riscontrati in condizioni di scarsa luminosità oppure durante l'utilizzo di occhiali con lenti scure, come ad esempio occhiali da sole, i quali rendono difficile la corretta identificazione dell'occhio umano, e viene dunque prodotto un risultato errato facendo pertanto scattare l'allarme acustico indicando che il conducente ha gli occhi completamente chiusi. Questo avviene specialmente nei casi in cui il sistema di fotocamere non presenta un sensore ad infrarosso, il quale facilita la corretta rilevazione nei casi di scarsa luminosità come ad esempio nei tratti di strada in cui sono presenti delle gallerie o semplicemente nelle ore notturne.

5.2 Sviluppi futuri

Il progetto può essere ulteriormente sviluppato, e diventa la base per un altro tirocinio, in cui le informazioni real-time del *tracking* vengono inviate ad un server per ulteriori elaborazioni. Inoltre trova un possibile ulteriore sviluppo andando ad ampliare le azioni in risposta a particolari eventi rilevati dal software, e migliorando l'accuratezza delle rilevazioni nei casi citati precedentemente quali scarsa luminosità, validando così i risultati in un ambiente operativo complesso.

Bibliografia

[1] MIT. Incidenti stradali. 2022. https://www.mit.gov.it/node/18093.

[2] https://www.newsauto.it/guide/adas-quali-sono-significato-2020-111281/.

[3] https://www.automobile.it/magazine/come-funziona/adas-sistemi-avanzati-assistenza-guida-3382.

[4] P. Biswas. *Exploring the Use of Eye Gaze Controlled Interfaces in Automotive Environments*. SpringerBriefs in Computer Science. Springer International Publishing, 2016.

[5] K. Holmqvist, R. Andersson, M. Nystrom, and H. Jarodzka. *Eye Tracking: A Comprehensive Guide to Methods and Measures*. Oxford University Press, 2015.

[6] M. Wedel and R. Pieters. *Eye Tracking for Visual Marketing*. Foundations and trends in marketing. Now Publishers, 2008.

[7] https://eyeware.tech/gazesense-eye-tracking-software-for-webcams-3d-sensor/.

[8] https://opencv.org/.

[9] https://www.di.univr.it/documenti/OccorrenzaIns/matdid/matdid753292.pdf.

[10] https://www.sciencedirect.com/science/article/abs/pii/S0079742120300244.

[11] https://www.net-informatica.it/migliori-tool-di-eye-tracking/.

[12] https://www.ai4business.it/intelligenza-artificiale/eye-tracker-come-funziona-la-tecnologia-che-legge-il-movimento-degli-occhi/.

[13] https://www.researchgate.net/figure/Using-of-a-head-model-to-detect-eye-and-gaze-orientation_fig4_29642053.

[14] https://www.eyelogicsolutions.com/
what-is-calibration-and-why-we-need-it/.

[15] https://scholarworks.rit.edu/other/487/.

[16] https://www.tobii.com/products/eye-trackers/wearables/
tobii-pro-glasses-3.

[17] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6960643/.

[18] https://www.semanticscholar.org/paper/
Gaze-estimation-method-based-on-an-aspherical-model-Nagamatsu-Iwamoto/
49deb46f1b045ee3b2f85d62015b9c42eb393e76.

[19] https://www.intel.it/content/www/it/it/
architecture-and-technology/realsense-overview.html.

[20] https://www.intelrealsense.com/wp-content/uploads/2022/03/
Intel-RealSense-D400-Series-Datasheet-March-2022.pdf.

[21] https://www.intelrealsense.com/wp-content/uploads/2022/11/
Intel-RealSense-D400-Series-Datasheet-November-2022.pdf.

[22] https://eyeware.tech/it/gazesense.

[23] https://www.intelrealsense.com/sdk-2/.

[24] https://de.mathworks.com/help/driving/ref/monocamera.html.

[25] https://dl.acm.org/doi/fullHtml/10.1145/3290607.3312942.

[26] https://www.researchgate.net/publication/29642053_A_history_
of_eye_gaze_tracking/.

Elenco delle figure

Ringraziamenti

In questo capitolo vorrei ringraziare innanzitutto il mio professore e relatore *Adriano Mancini* per la sua infinita disponibilità e per tutta l'esperienza che mi ha trasmesso durante questo percorso.

Inoltre mi piacerebbe ringraziare e dedicare questa tesi ai miei genitori, per tutti gli immensi sacrifici e sforzi i quali mi hanno aiutato ad arrivare fin qui, e per aver sempre creduto in me, aiutandomi nei momenti difficili, con la speranza, un giorno, di ripagarvi per tutto ciò che avete fatto.

Ringrazio inoltre i miei amici per avermi fatto compagnia in questa esperienza, ed infine *Federica* per essere stata al mio fianco quando ne avevo bisogno, e per aver condiviso con me i momenti di tristezza e di gioia.