Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Eyes Tracking
Tracciamento dello sguardo tramite fotocamera frontale dello smartphone

Relatore:
Dott.
Luca Bedogni

Presentata da:
Giovanni Fidanza

Sessione III
Anno Accademico 2017 - 2018

Abstract

Lo scopo di questa ricerca e di creare un algoritmo per il tracciamento approssimativo della direzione dello sguardo umano utilizzando la fotocamere di comuni smartphone.
Attualmente esistono varie tecnologie di Eye-tracking, a vari livelli di precisione, quasi sempre basate su costosi hardware dedicati.
L'obiettivo finale è quello di integrare l'algoritmo sulle app per smartphone e consentire a chi lo sta usando di essere allertato se qualcuno "sbircia" nel suo dispositivo.
L'idea di base è quella di utilizzare la fotocamera frontale dell'utente per "intercettare" ed analizzare i movimenti oculari dei i vari volti inquadrati, escludendo la faccia dell'utente mediante funzioni già presenti in Android.
L'algoritmo mira a fornire un Alert se rileva sguardi indiscreti verso il proprio smartphone e sollecitare un comportamento di protezione della privacy, ad esempio evitando di leggere dati sensibili o digitare password.

Indice

Capitolo 1

Introduzione

Cos'è l'Eyes Tracking

La relazione tra i movimenti dell'occhio umano e le emozioni che possono veicolare è un argomento oggetto di interesse fin dall'antichità e di grande interesse per la psicologia sociale e comportamentale. Con il progresso della tecnologia e delle tecniche di rilevamento anche la medicina ha iniziato a nutrire un forte interesse per i dati relativi ai movimenti oculari e considerarli come strumento di supporto per la diagnosi di alcune patologie.
Ovviamente le prime forme di rilevamento erano di tipo visivo, alle quali seguirono alla fine dell'ottocento esperimenti con particolari lenti a contatto collegati a strumenti che rilevavano i movimenti oculari. Nei primi decenni del novecento furono sperimentate le prime tecniche di rilevamento non intrusive, tramite la luce riflessa dalle pupille, con lo scopo di studiare le strategie che l'occhio mette in atto per "leggere" un testo scritto o un'immagine.
La terminologia medico scientifica che caratterizza il settore di studi non è oggetto di questo studio, tuttavia è bene chiarire alcuni concetti cardine che sono alla base anche delle più sofisticate tecniche moderne di Eyes tracking. L'osservazione dei movimenti dell'occhio, la contrazione e la dilatazione delle pupille, sono nati in un contesto clinico-medico con l'intento di capire il funzionamento del complesso sistema della visione umana. La luce penetra all'interno dell'occhio attraverso la pupilla e va a colpire i ricettori della la retina che trasmettono le le informazioni al cervello; in realtà solo la parte centrale della retina, la *fovea*, riesce a vedere nitidamente l'immagine mentre il resto della retina percepisce l'immagine quanto più è lontana dalla fovea. Per poter permettere al cervello di "ricostruire" l'immagine a fuoco nella sua completezza, l'occhio fa costantemente dei micro movimenti, solo apparentemente casuali; al termine di ogni movimento segue la cosiddetta fissazione,

della durata di 2-4 centesimi di secondo. Il movimento viene definito *saccade*, e avviene con una frequenza di 3-4 movimenti al secondo. A differenza del movimento oculare volontario, che ciascuno di noi fa per dirigere lo sguardo dove vuole, le *saccadi* hanno la caratteristica di essere involontari.
Il meccanismo di dilatazione e di contrazione della pupilla, direttamente correlata con il sistema nervoso autonomo e quindi anch'essa di natura involontaria, ha la funzione principale di filtrare la quantità di luce necessaria per una corretta visione; pertanto si dilata con l'oscurità e si restringe quando c'è molta luce. Inoltre, accanto a questa funzione lineare di filtro, le pupille possono avere dei comportamenti analoghi in situazioni emozionali e/o patologiche, fonte perciò di preziose informazioni all'osservatore. E' proprio la relazione tra i movimenti oculari e la sfera emozionale che ha di fatto ampliato i settori che sono interessati ai dati forniti dalle rilevazione del movimento oculare.

Capitolo 2

Stato dell'arte

2.1 Utilizzi dell' Eyes Tracking

Negli ultimi anni l'enorme sviluppo delle tecnologie di ripresa video e dell'informatica hanno creato le condizioni per una forte crescita di interesse verso le tecnologie di Eyes tracking da parte di settori disciplinari più svariati: dal marketing al Pc-gaming, dalla sicurezza stradale al miglioramento dell'interazione tra uomo e macchina. Le diverse motivazioni le specificità di ciascun campo di interesse trova il comune denominatore nella raccolta, analisi ed elaborazione delle informazioni sul movimento oculare e richiede algoritmi sempre più efficaci ed efficienti per tale scopo. La grande diffusione di applicativi che richiedono l'utilizzo di questa tecnologia è il motore di una continua ricerca di soluzioni non solo ad un livello altamente specializzato, con costosi hardware e software, ma anche per soluzioni innovative che prevedano l'utilizzo di software open source e strumenti tecnologici di ampia diffusione dai costi moderati.

Ad oggi i settori che utilizzano maggiormente le tecnologie di Eyes tracking e che ne stimolano lo sviluppo sono:

- **Automotive** La ricerca che ruota intorno all'automotive da molto tempo si avvale di tecnologie che permettono il tracciamento dello sguardo; lo scopo principale è orientato ad ottenere informazioni utili per:
  - Ottimizzare il *layout* dei cruscotti in modo da fornire le informazioni principali in modo semplice, facilmente leggibili e in completa sicurezza.
  - Gestire la sicurezza del guidatore rilevando comportamenti potenzialmente pericolosi: ad esempio la dilatazione della pupilla potrebbe indicare assunzione di droghe o alcol, oppure segnalare

stati di sonnolenza, perdita di concentrazione, ecc. Il fine ultimo è prevenire quegli incidenti stradali le cui cause potrebbero essere in relazione con queste rilevazioni

- **PC-gaming** Il tracciamento dello sguardo è stato recentemente introdotto anche nel mondo dei videogiochi per affinare le tecnologie di interazione uomo-macchina. Principalmente per studiare ed interpretare le emozioni del giocatore attraverso l'analisi delle dilatazioni della pupilla allo scopo di fornire agli sviluppatori il feedback per implementare contenuti emozionali. Inoltre sono oggetto di ricerca soluzioni sempre più sofisticate che consentono al giocatore il controllo del gioco attraverso i movimenti oculari, soprattutto in ambiti di realtà virtuali.

- **Ricerca medica**[1] [2] In campo medico la combinazione di Eye-tracking con altri strumenti di rilevazione, come ad esempio sensori biometrici, permette di perfezionare la diagnosi di diverse malattie, come ad esempio il deficit di attenzione, il disturbo ossessivo compulsivo, la schizofrenia, la sonnolenza, etc. Inoltre, mediante hardware specifici, è possibile anche fornire alle persone con handicap uno strumento per facilitare l'interazione con Pc o altre apparecchiature elettroniche

- **Usability** Nell'ambito del Web-design il tracciamento oculare si rivela ormai una tecnologia di enorme importanza: analizzare e rilevare in quale area e per quanto tempo l'attenzione di un utente si sofferma, ad esempio in una pagina web, contribuisce notevolmente al lavoro di costante miglioramento dell'interfaccia: correlando i dati di movimento oculare con altri dati, come ad esempio i movimenti del mouse, è possibile migliorare in modo considerevole l'*user-experience*.

- **Marketing** Negli ultimi anni l'EyesTracking è oggetto di grande interesse da parte del mondo del marketing, in particolare da grandi aziende. Queste sono le principali aree di utilizzo.

  - Retail. L'osservazione del comportamento del consumatore davanti ad uno scaffale è sempre stato oggetto di grande attenzione da parte della grande distribuzione. Mediante una telecamera che

[1] https://www.ncbi.nlm.nih.gov/pubmed/28606763
[2] https://ieeexplore.ieee.org/document/5373675

inquadra il consumatore frontalmente è possibile rilevare quali sono i prodotti che attirano maggiormente l'attenzione. Di conseguenza, analizzando i dati del movimento oculare, la scelta e la disposizione dei prodotti sugli scaffali può essere ottimizzata.

- Packaging. Anche nello studio del confezionamento dei prodotti informazioni sulle reazioni del consumatore contribuisce notevolmente a migliorare le performance di vendita.
- Pubblicità. In ambito pubblicitario gli studi sul movimento oculare sono di grande interesse. L'attenzione è focalizzata sia agli studi cognitivi che di marketing vero e proprio. Capire in che modo colori, grafica e rappresentazioni visive in generale influenzino l'attenzione del consumatore è di estrema importanza per progettare impostare campagne pubblicitarie sempre più efficaci. inoltre il rilevamento delle aree nelle quali si sofferma più frequentemente l'attenzione dell'utente/consumatore è un dato di crescente utilità per il posizionamento dei messaggi pubblicitari. Si pensi ad esempio al posizionamento ottimale dei banner in una pagina web.

2.2 Lavori correlati

Lo scenario degli studi sull' Eyes Tracking contempla numerosi studi da almeno venti anni. La letteratura in merito è corposa, soprattutto articoli scientifici che trattano algoritmi e sistemi di utilizzo del rilevamento oculare. I livelli applicativi della tecnologia spaziano dalle applicazioni più semplici a complessi algoritmi con hardware dedicati.

Hardware

Per effettuare un buon riconoscimento oculare viene spesso utilizzata la telecamera ad infrarossi di alta qualità che consente di stabilire con precisione colorazione, mappatura 3d e direzione dell'occhio. Questo consente di ottenere un'accuratezza elevata senza dover ricorrere all'utilizzo di costosi hardware dedicati.

Il problema del rapporto tra il costo e il risultato ottenuto è una costante che può influire notevolmente sulla commercializzazione: ad esempio l'azienda "pupil-labs[3]" ha prodotto un occhiale molto preciso che rileva la direzione dello sguardo per svariati utilizzi. Il vantaggio è quello di un'alta precisione, con dati millimetrici che consentono studi approfonditi ed affidabili, utilizzati soprattutto per la ricerca o per migliorare le esperienze di realtà virtuale. Il costo che attualmente si aggira intorno ai 2800 euro, è senz'altro un ostacolo

[3]https://pupil-labs.com/

per una diffusione di massa. Un altro esempio di hardware dedicato è quello dell'azienda "Tobii[4]" che ha sviluppato un dispositivo Eyes tracker, una sottile barra da montare sulla base del monitor, compatibile con i personal computer. Il costo di circa 150 dollari non è eccessivo, a fronte di una precisione di rilevazione moderata. Potrebbe senz'altro essere utile per migliorare l'*user-experience* in generale (gaming, streaming, accessibiltà, usability).
Da un punto di vista del soggetto osservato le tipologie di *eye tracking* possono essere suddivise in due categorie che potremmo definire *inter-attiva e passiva*: nella prima rientrano i due dispositivi appena citati per i quali l'utente è non solo consapevole del meccanismo di rivelazione in atto, ma attivamente coinvolto soprattutto a scopo interattivo. Mentre i casi in cui il soggetto è coinvolto, più o meno consapevolmente, in un processo di rilevazioni sostanzialmente finalizzato a fornire dati circa il suo comportamento spontaneo (ad esempio la telecamera davanti agli scaffali), possiamo definirli passivi. Questo implica che strumenti hardware che comportano una cooperazione del soggetto si prestano meno a rilevare dati in contesti nei quali è preferibile che una situazione di inconsapevolezza.

[4]https://www.tobii.com/

Capitolo 3

Progettazione e sviluppo

L'obiettivo di questo studio è di realizzare un dispositivo di Eyes tracking che può utilizzare fotocamere ad uso commerciale senza avere la necessità di un hardware specifico. Ad esempio si possono utilizzare le webcam integrate nei PC oppure negli smartphone. Ho strutturato la presentazione dedicando una sezione ad ogni in test in modo da dettagliare il procedimento utilizzato e spiegare i processi che hanno determinato la scelta finale.

3.1 Ambiente di sviluppo

É stato scelto di utilizzare il linguaggio *python* per realizzare l'applicativo. Il motivo principale per il quale è stato scelto questo linguaggio è per la caratteristica di portabilità che lo caratterizza e per la possibilità di sviluppare ad alto livello. Inoltre le numerose librerie e tool che offre rappresentano un ulteriore elemento di vantaggio. L'applicativo è stato sviluppato in ambiente *unix*, utilizzando come editor "sublime-text"[1], In particolare la libreria principale utilizzata è "Opencv"[2].

3.2 OpenCV

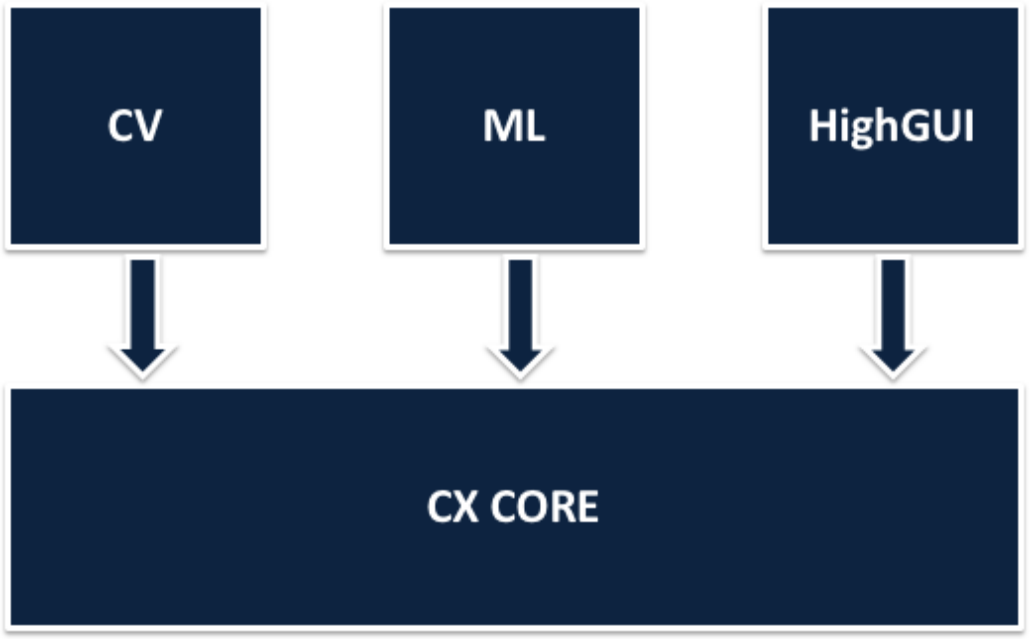
OpenCV (Open Source Computer Vision Library) è rilasciato sotto una licenza BSD e quindi OpenSource sia per uso accademico che commerciale. Ha interfacce C ++, Python e Java e supporta Windows, Linux, Mac OS, iOS e Android. OpenCV è stato progettato per l'efficienza computazionale e con una forte attenzione per le applicazioni in tempo reale. Scritta in C / C ++,

[1] https://www.sublimetext.com/
[2] https://opencv.org/

la libreria può trarre vantaggio dall'elaborazione multi-core. Abilitato con OpenCL, può sfruttare l'accelerazione hardware della piattaforma di calcolo eterogenea sottostante.

Adottato in tutto il mondo, OpenCV conta su una community di oltre 47 mila utenti e un numero stimato di download che superano i 14 milioni. L'utilizzo spazia dall'arte interattiva, all'ispezione delle mine, alle mappe dei punti sul Web o alla robotica avanzata.

Struttura di OpenCV:[3]



- CXCORE: implementa le strutture dati e le funzioni per gestire immagini e video.
- CV: modulo specializzato nell'elaborazione e l'analisi delle immagini, la calibrazione e il tracking.
- ML (Machine Learning): contiene molte funzioni sul Machine Learning e sul pattern recognition, quali il clustering e la classificazione
- HighGUI: implementa le definizioni delle interfacce utente (GUI).

Durante l'installazione è richiesta anche la libreria numpy (una libreria matematica molto potente ai fini di velocizzare i calcoli *real time*)

[3]https://www.domsoria.com/2018/05/face-recognition-and-tracking-con-opencv-prima-parte/

OpenCV è stato utilizzato per riconoscere, in una prima fase per la faccia e gli occhi presi da un'immagine dalla fotocamera o di un video. In una seconda fase è stato utilizzato anche per tracciare i movimenti della pupilla. Il tutto è stato realizzato utilizzando le funzioni del *framework* che servono a riconoscere oggetti presenti in un'immagine tramite file "haar cascade"

3.3 Haar Cascade

È fondamentalmente un classificatore che viene utilizzato per rilevare l'oggetto per il quale è stato addestrato, dalla sorgente. La tecnologia si basa nel sovrapporre l'immagine positiva a una serie di immagini negative. Il *training* (classificazione) consiste nel raccogliere dati di un oggetto che successivamente vogliamo riconoscere. Per avere risultati ottimali è necessaria una grande mole di dati (immagini ad alta risoluzione) dell'oggetto che si vuole riconoscere. Su github[4] ci sono vari files "HaarCascade" che possono essere scaricati e utilizzati.

3.4 Riconoscimento volto e occhi

Obbiettivo

L'obiettivo di questo algoritmo è quello di riconoscere tramite la fotocamera del computer se sono presenti volti e occhi e da evidenziare i risultati sullo schermo.

Modellazione e sviluppo

Inizialmente l'algoritmo per il riconoscimento è stato incentrato su due punti:

- Riconoscimento del volto
- Riconoscimento dell'occhio

I primi test sono stati effettuati su varie immagini di tipo "jpeg o png" con diversi soggetti e uno o più volti nella stessa immagine. Dopo aver ottenuto risultati positivi con immagini, lo step successivo è stato il passaggio ad un input video in tempo reale, mostrando i risultati a video con rettangoli blu per le facce e rossi per gli occhi (come mostrato in figura).

[4]https://github.com/opencv/opencv/tree/master/data/haarcascades

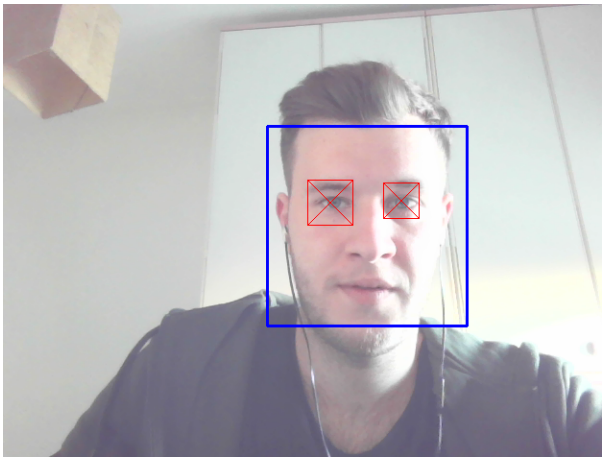


Figura 3.1: Screenshot dell'applicazione

Di seguito l'algoritmo in Python utilizzato per il riconoscimento

```
import numpy as np
import cv2

face_cascade = cv2.CascadeClassifier('
    haarcascade_frontalface_default.xml')


cap = cv2.VideoCapture(0)


while 1:
  #immagina
  ret, img = cap.read()

  if ret == True:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #riconoscimento faccia
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml'
    )
    eyes = eye_cascade.detectMultiScale(img,1.3,10)

    for (x,y,w,h) in faces:
      #x,y coordinate punto in alto a sinistra - w,h
    larghezza e altezza
      cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2) #
    rettangolo blu
    #riconoscimento occhi
    for (ex,ey,ew,eh) in eyes:
```

```
26        #ex,ey, coordinate punto in alto a sinistra - ew,eh
      larghezza e altezza
27        cv2.rectangle(img, (ex,ey), ((ex+ew),(ey+eh)),
      (0,0,255),1) #rettangolo rosso
28        cv2.line(img, (ex,ey), ((ex+ew,ey+eh)), (0,0,255),1)#
      linea per x
29        cv2.line(img, (ex+ew,ey), ((ex,ey+eh)), (0,0,255),1)#
      linea per x
30
31
32      cv2.imshow('img',img)
33      k = cv2.waitKey(30) & 0xff
34      if k == ord ('q'):
35        break
36
37  cap.release()
38  cv2.destroyAllWindows()
```

Conclusioni e problemi

Anche se nella maggior parte dei casi il riconoscimento è preciso e vengono correttamente individuati occhi e volti, si verificano casi di riconoscimento di oggetti all'esterno del volto che non sono occhi. Inoltre un movimento veloce del soggetto crea instabilità all'algoritmo che riconosce con difficoltà gli occhi e la faccia.

3.5 Creazione struttura dati

Obbiettivo

Poiché in varie casistiche il *framework* OpenCV erroneamente riconosce oggetti o forme come occhio, la presente struttura dati mira a creare una *cache* con lo storico delle ultime "n" posizioni per poter evitare gli errori.

Modellazione e sviluppo

E' stata creata una struttura dati in modo da consentire l'analisi delle ultime posizioni di un occhio. Per realizzare la struttura dati sono state creati classi differenti per faccia e occhi. Lo storico è dato da un parametro passato che, convenzionalmente, è stato fissato a 10. I dati vengono salvati in due *array* contenenti i dati delle rispettive classi.
Le classi contengono quindi le coordinate dell'oggetto riconosciuto e un valore chiamato "priorità". La priorità serve a determinare se e quando un

oggetto può essere considerato nuovo, vecchio o un falso positivo. Per ogni oggetto riconosciuto in un frame del video, viene effettuata una ricerca nella struttura dati; se l'oggetto era già presente (con un *offset* fissato) si aggiorna la struttura dati, altrimenti viene aggiunto in una struttura dati temporanea per poter essere valutato in seguito. Lo spostamento di un oggetto, dalla struttura dati temporanea a quella effettiva, avviene quando tale oggetto compare almeno in "n" frame raggiungendo così una "priorità" più alta.

Conclusioni e problemi

Il riconoscimento dell'occhio è più preciso, a discapito della velocità: movimenti rapidi faticano ad associare l'occhio al suo rispettivo storico presente nella struttura dati; tuttavia gli errori e i falsi positivi sono stati notevolmente ridotti. Una parte degli errori, che ancora sono numerosi, vanno imputati alle alterazioni di luce e colori presenti in un'immagine RGB e al "forzare" il riconoscimento dell'occhio.

3.6 Applicazione di filtri fotografici

Obbiettivo

Ridurre il rumore dell'immagine per poter migliorare il riconoscimento al variare di condizioni ambientali quali luminosità, contrasto, esposizione, ecc..

Modellazione e sviluppo

I colori di un'immagine non sono utili al riconoscimento dell'occhio. Ho preferito pertanto di utilizzare come input solamente immagini in tonalità di grigio. Una volta modificata l'immagine è stato applicato il filtro fotografico Gaussian Blur[5]

BLUR: è un'elaborazione dell'immagine basta su una formula matematica gaussiana che riduce il rumore dell'immagine; l'effetto visivo di questa tecnica è una sfocatura omogenea. Di seguito 2 esempi di immagini alle quali è stato applicato il filtro.

[5]https://en.wikipedia.org/wiki/Gaussianblur

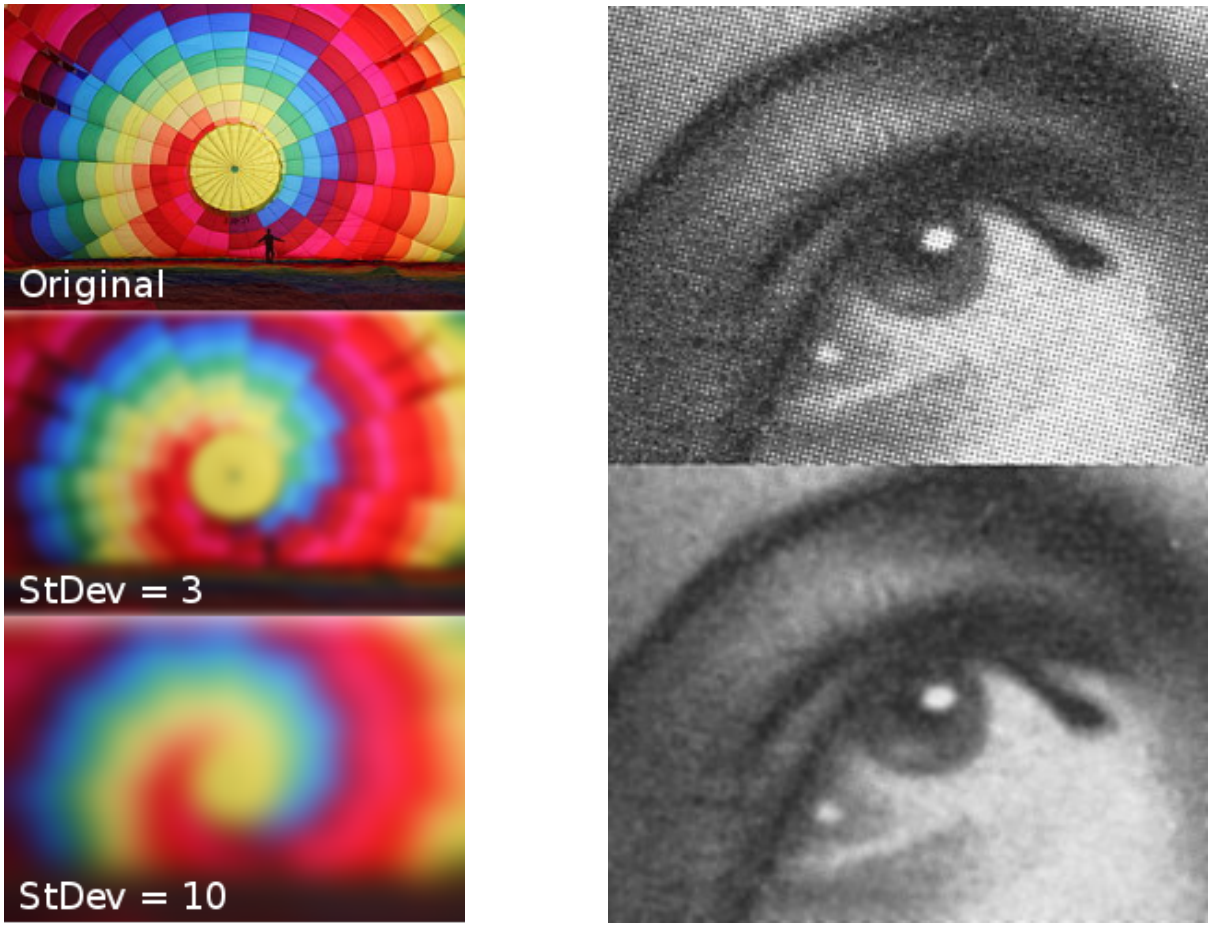


Figura 3.2: esempi del filtro Gaussian Blur

Conclusioni

Il rumore dell'immagine è stato quindi notevolmente ridotto. Durante lo sviluppo è stato rimosso il riconoscimento della faccia perché non utile ai fini dell'algoritmo: in particolare la casistica di un soggetto che guarda la fotocamera senza che il suo volta sia interamente inquadrato rappresenta una criticità.

3.7 Riconoscimento dell'iride

Obbiettivo

Riconoscere geometricamente la posizione dell'iride all'interno dell'occhio.

Modellazione e sviluppo

Per determinare la direzione dello sguardo è necessario ricercare l'iride. Per raggiungere questo obbiettivo è stata utilizzata la funzione "HoughCircles" di OpenCV che consente di cercare dei cerchi all'interno di una determinata immagine. Nello specifico sono state create "n" immagini, tante quanti sono gli occhi presenti nella struttura dati, in modo tale da limitare la ricerca alla figura geometrica del cerchio che corrisponde alla forma dell'iride o della pupilla.

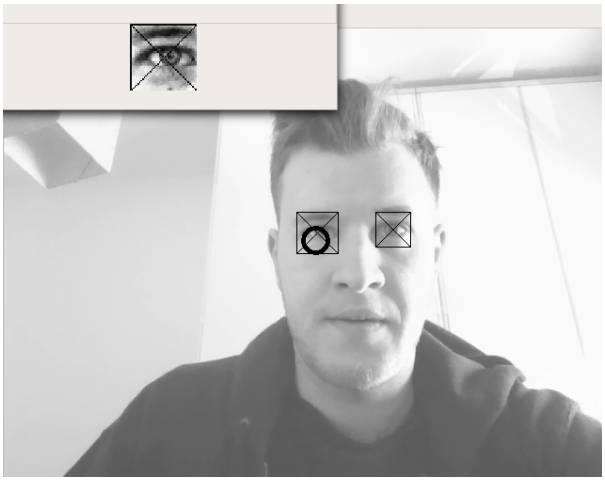


Figura 3.3: Screenshot dell'applicazione

Conclusioni e problemi

Come si può vedere anche dall'immagine il riconoscimento non è preciso. L'errore è determinato principalmente da due fattori:

1. L'iride non sempre è perfettamente circolare poiché una parte di essa può essere coperta dalla palpebra superiore o inferiore.

2. Il riconoscimento forme (della funzione "HoughCircle") non è efficiente a causa di una grande quantità di pixel da analizzare; di conseguenza i risultati sono instabili.

Il primo problema potrebbe essere risolto istruendo l'algoritmo a cercare prima una forma ellittica (bulbo oculare) e successivamente il cerchio della pupilla, ma purtroppo OpenCV non permette la ricerca di ellisse come for-

ma geometrica; di conseguenza non è possibile individuare nello specifico i contorni della sclera[6]).

3.8 Rilevamento contorni

Obbiettivo

"Pulire" l'immagine per poter rendere più efficace il riconoscimento dell'iride

Modellazione e sviluppo

Per "pulire" l'immagine sono stati applicati filtri specifici particolarmente adatti a ridurre al minimo il rumore ed evidenziare i contorni di sclera e iride. In questo modo risulta più facile poter "ricercare" le forme desiderate in quanto i contorni riconosciuti più pertinenti alla realtà.
Gli effetti utilizzati, in aggiunta al blur, sono: CLAHE e Canny.

- CLAHE (Contrast Limited Adaptive Histogram Equalization)[wikipedia]: L'equalizzazione adattiva dell'istogramma è una tecnica di elaborazione digitale delle immagini usata per migliorarne il contrasto. Differisce dall'ordinaria equalizzazione dell'istogramma poiché il metodo adattivo calcola diversi istogrammi, ognuno corrispondente a una distinta sezione dell'immagine, e li usa per distribuire i valori di luminosità. È anche utilizzabile per migliorare il contrasto locale di una immagine e aggiungere maggior dettaglio.

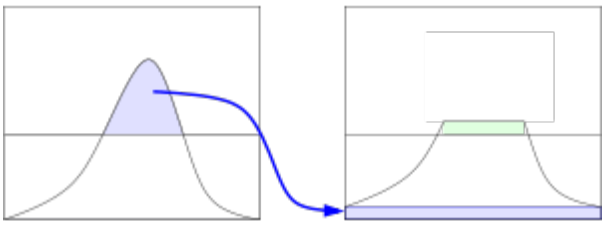


Figura 3.4: Equalizzazione adattiva dell'istogramma

[6]anche detto "bianco dell'occhio"

- CANNY: l'algoritmo di Canny è un operatore per il riconoscimento dei contorni (edge detection) ideato nel 1986 da John F. Canny. Utilizza un metodo di calcolo multi-stadio per individuare contorni di molti dei tipi normalmente presenti nelle immagini reali.

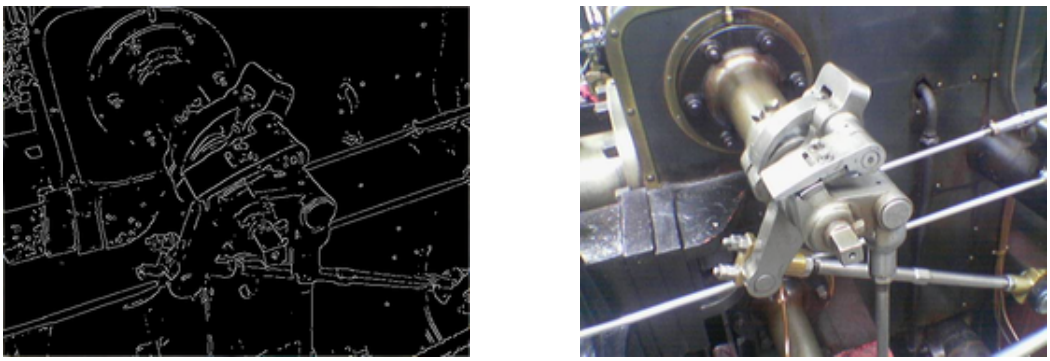


Figura 3.5: esempio del filtro Canny

L'inserimento di questi due effetti aggiuntivi ha permesso di rendere l'immagine dell'occhio più nitida ed ha consentito l'individuazione dei contorni, come mostrato nella seguente immagine:

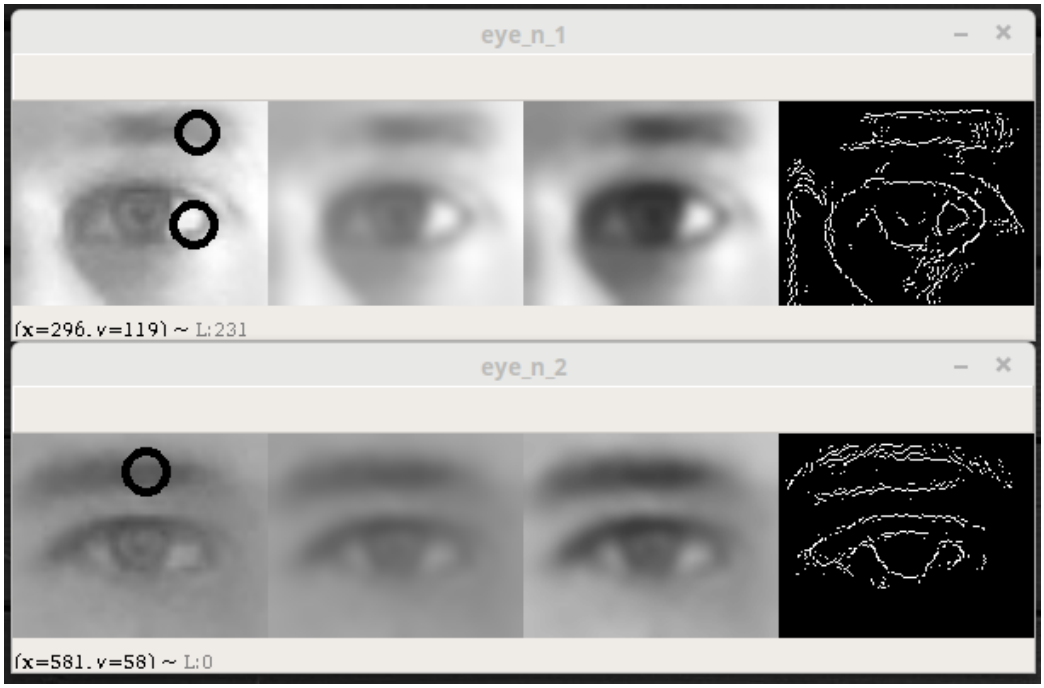


Figura 3.6: Screenshot dell'applicazione

Conclusioni e problemi

Una volta applicati gli effetti i contorni riconosciuti sono abbastanza fedeli alla realtà. Il problema principale, come anche mostrato in figura, denota che c'è molta differenza tra i risultati in relazione ai parametri utilizzati per calibrare gli effetti; di conseguenza il riconoscimento risulta errato in alcune casistiche; inoltre il risultato finale è fortemente condizionato dal elementi esterni come ad esempio luminosità, contrasto, colori, etc.

3.9 Analisi dei parametri su filtri Blur, CLAHE, Canny

Obbiettivo

Le diverse combinazioni possibili dei vari filtri fotografici influiscono notevolmente sul risultato finale. L'obiettivo di questa fase è stato di strutturare queste informazioni e stabilire dei range numerici che potessero aumentare l'efficienza del riconoscimento.

Modellazione e sviluppo

In questa fase ho focalizzato l'attenzione sugli attributi e parametri degli effetti, studiando l'andamento dei risultati al variare di uno o più parametri. Per fare ciò sono stati effettuati i seguenti step:

1. Registrazione di video con luminosità e contrasti differenti.

2. Salvataggio della posizione dell'iride reale.

3. Riconoscimento della posizione dell'iride mediante algoritmo variando i parametri degli effetti.

4. Salvataggio dei dati nel formato csv[7].

5. Analisi dei dati con grafici.

Sono stati generati una considerevole quantità di grafici, grazie all'utilizzo di R, che hanno permesso di analizzare le relazioni tra i vari filtri e scegliere i migliori parametri.

Script per la generazione di grafici:

```
library(ggplot2)

colnames <- c('CN1','CN2','BL1','BL2','CL','PP1','PP2','FRAME
    ','ES','ER')
df <- read.csv('es.csv', col.names=colnames)
summary(df)
df$P_ES = df$ES/(df$FRAME * 2)

g <- ggplot(data= df[df$CN1==10 & df$CN2==10 & df$CL==28,],
    aes(x=BL1,y=P_ES, color=factor(BL2))) + geom_line()
ggsave('result.png',g)
```

Qui di seguito alcuni esempi di grafici:

[7](Comma-Separated Values) formato di file utile per rappresentare dati numerici

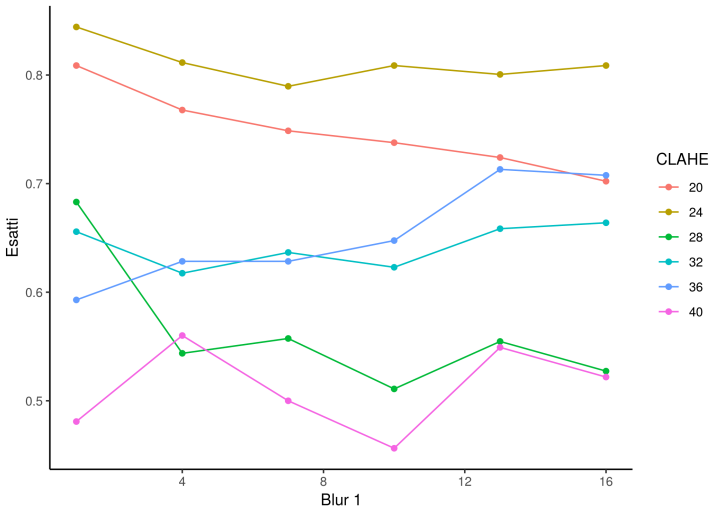


Figura 3.7: **CLAHE e BLUR 1**. Si può notare come il valore CLAHE=24 dia risultati positivi a prescindere dal variare del parametro BLUR 1

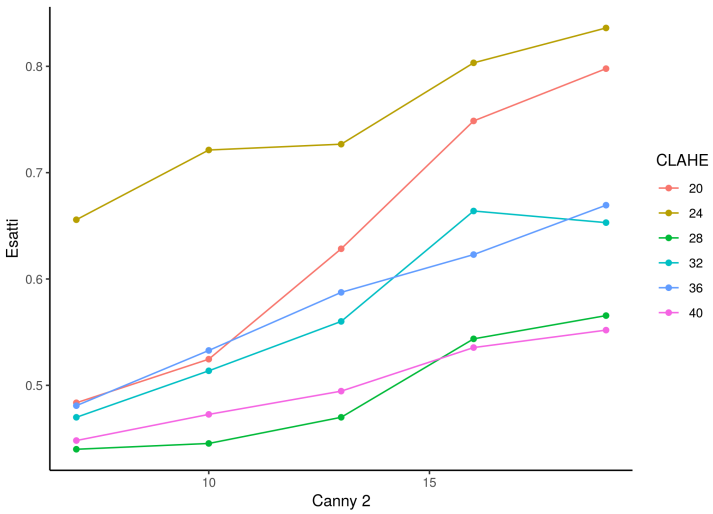


Figura 3.8: **CLAHE e CANNY 2**. Un altro esempio in cui il valore CLAHE=24 dia risultati migliori rispetto agli altri, inoltre CANNY 2 restituisce dei risultati positivi con valori più alti.

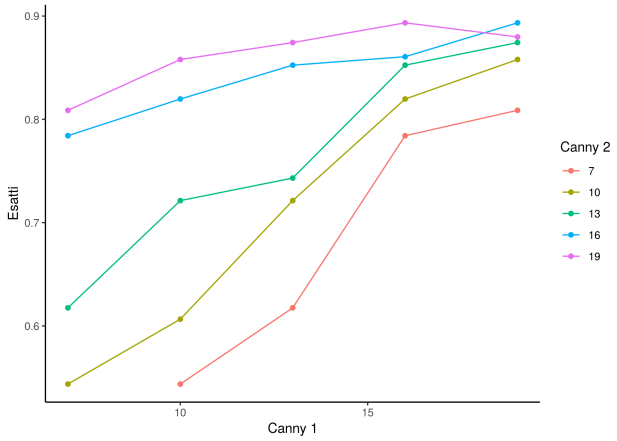


Figura 3.9: **CLAHE e BLUR 1**. Da questo grafico si evince che il BLUR 1 restituisce risultati migliori con valori più bassi

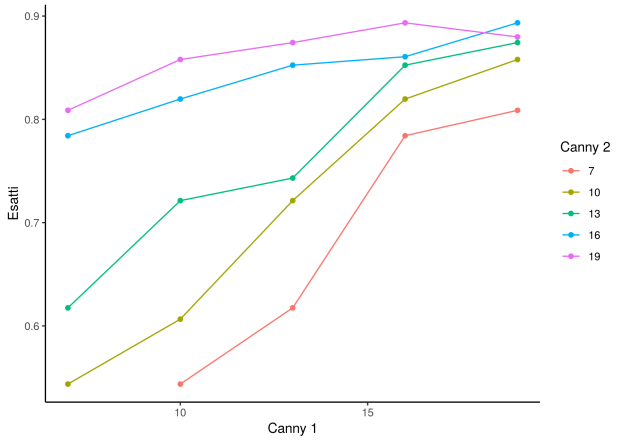


Figura 3.10: **CANNY 1 e CANNY 2**. In questo esempio invece viene messo in risalto il rapporto tra CANNY 1 e 2 confermando che il CANNY 2 ha migliori prestazioni per valori più alti.(~ 20)

Conclusioni

Dopo un'analisi dei grafici che sono stati generati dallo Script in R è stato possibile stabilire dei *range* ottimali per ogni parametro. In modo da rendere ottimale il riconoscimento dell'iride e stabilire la direzione. I valori utilizzati per l'analisi sono quindi:
CANNY 1 = 16
CANNY 2 = 10

BLUR 1 = 1
BLUR 2 = 16
CLAHE = 24

3.10 Riconoscimento dell'iride (2)

Obbiettivo

Riconoscere geometricamente la posizione dell'iride, dopo aver applicato gli effetti fotografici illustrati nei precedenti algoritmi.

Modellazione e sviluppo

Una volta scelti i valori ottimali per ogni parametro (descritti nel capitolo "Analisi dei parametri") il riconoscimento dell'iride è stato implementato sull'effetto finale del Canny. In questo modo è risultato molto più efficiente stabilire quali fossero realmente i contorni dell'occhio, in particolare iride e sclera.
Per ottenere la media delle ultime posizioni dell'iride è stata creata una nuova struttura dati contenente lo storico degli ultimi "n" riconoscimenti sugli assi x ed y. Questa struttura viene salvata nella matrice multidimensionale, contenente già i dati degli occhi, essendo il riconoscimento dell'iride strettamente legato ad un particolare occhio.

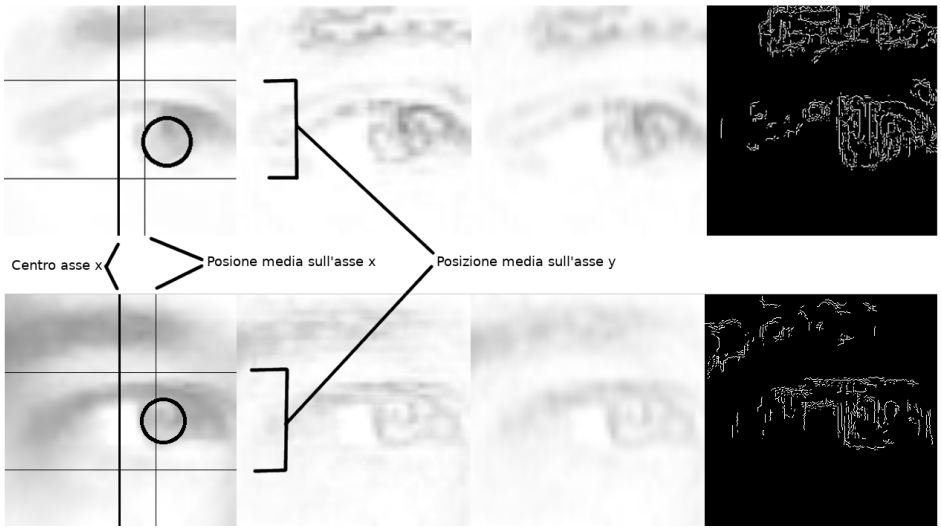


Figura 3.11: Screenshot del riconoscimento dell'iride

Conclusioni e problemi

Sulla base dei dati approssimativi sulla posizione dell'iride è stato possibile rilevare un riconoscimento approssimativo della direzione dello sguardo (sinistra, destra, centro); la procedura è stata di considerare la metà dell'asse x (il centro orizzontale dell'immagine) e aggiungendo o togliendo un offset prefissato(ad esempio 30px).

I risultati non sono stati molto soddisfacenti: l'occhio non perfettamente centrato poiché le funzioni di OpenCV non sono molto precise nel rilevare la posizione, inoltre la struttura dati inserita precedentemente ha come conseguenza un'approssimazione della posizione che, è utile per eliminare i falsi positivi, ma può decentrare di poco il riconoscimento dell'occhio se il soggetto compie dei movimenti.
Come conseguenza, stabilire la direzione in base fattori legati alla dimensione e centro dell'immagine, ha portato alla generazione di errori di natura geometrica.
Un'esempio di errore è evidente nella Figura 3.12: si può notare che, nonostante il soggetto stia guardando a sinistra, l'algoritmo riconosce (erroneamente) che lo sguardo è diretto verso il centro / leggermente a destra.

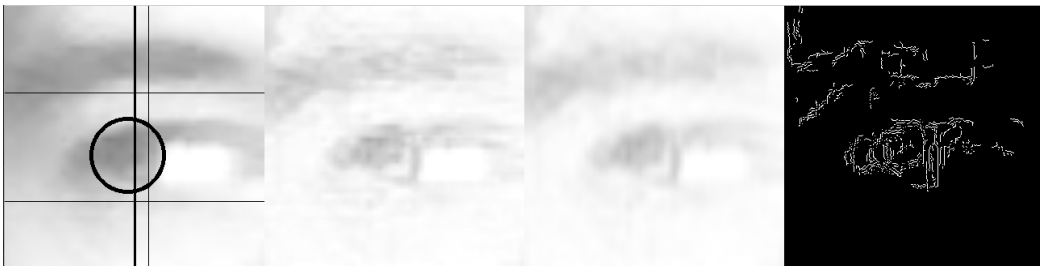


Figura 3.12: Screenshot del riconoscimento dell'iride

3.11 Riconoscimento basato su "media colore"

Obbiettivo

Riconoscere la direzione (centro, destra, sinistra) in base alla media del colore di un'area selezionata dinamicamente.

Modellazione e sviluppo

I risultati con il riconoscimento dell'iride, descritto nel capitolo precedente, non sono stati quelli desiderati; pertanto ho focalizzato l'attenzione sul colore. Ho modificato l'algoritmo in modo da fornire la media del colore in aree definite calcolate in base alla media delle ultime posizioni dell'iride (utilizzando i dati prodotti dall'algoritmo descritto nel capitolo "Riconoscimento dell'iride(2)").
In particolare, vengono designati 2 quadrati a destra e sinistra di una determinata posizione. L'asse y di questa posizione è dato dalla media della posizione dell'iride negli ultimi frame. Per l'asse x invece la posizione è calcolata in percentuale rispetto alla larghezza dell'occhio riconosciuto.

Nella Figura 3.13 è possibile vedere i due quadrati calcolati come descritto. Le linee, verticale e orizzontale, rappresentano la media della posizione dell'iride rispettivamente sull'asse y e sull'asse x. Il cerchio invece rappresenta l'iride riconosciuta nell'ultimo frame.

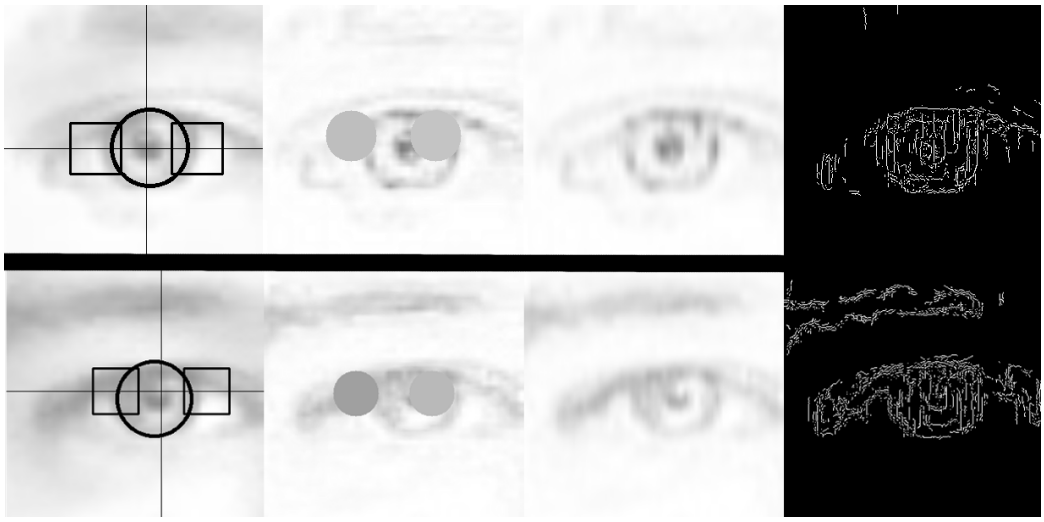


Figura 3.13: Screenshot del riconoscimento dell'iride

Una volta designate le aree quadratiche è stato possibile calcolare la media del colore all'interno di esse come mostrato nella seguente figura.

Figura 3.14: Screenshot del riconoscimento dell'iride

Si può notare che la media del rettangolo a sinistra è notevolmente più scura della media del rettangolo a destra; questo accade poiché essendo la direzione dello sguardo verso sinistra, l'iride va a coprire l'area del rettangolo di sinistra rendendone la "media colore" più scura.

Quindi si può stabilire la direzione dello sguardo, sottraendo i valori delle rispettive aree. Ovviamente il risultato ottenuto sarà compreso tra -255 e 255 (valori massimi del colore espressi in esadecimale).
Un valore lontano dallo 0 (ad esempio -80) indicherà che lo sguardo è rivolto da un lato, viceversa avendo valori vicini allo 0 (ad esempio 13) lo sguardo è rivolto verso il centro. Nello specifico i valori negativi indicano come direzione sinistra mentre i positivi destra.

Conclusioni

I risultati di questo algoritmo sono stati molto soddisfacenti in quanto è stato possibile stabilire la direzione dello sguardo (destra, sinistra, centro) in modo chiaro.
Gli intervalli utilizzati sono stati:
da -255 a -30 per sinistra
da -30 a 30 per il centro
da 30 a 255 per destra

L'algoritmo è stato opportunamente modificato in modo da salvare su un file "csv" i dati relativi ad ogni singolo frame di un particolare video consentendo un'analisi più approfondita.

Capitolo 4

Analisi dei dati

4.1 Valutazioni

Per poter valutare l'efficienza dell'algoritmo sono state utilizzate le seguenti metriche: Precision, Recall, F-measure.
Queste tre metriche sono strettamente correlate tra loro. I termini di misura utilizzati sono: direzione effettiva (la direzione dello sguardo effettiva del soggetto), *eye1*,*eye2* (la direzione riconosciuta dall'algoritmo per ogni occhio presente), il tutto valutato sul numero di *frame* totali.

Metriche

Precision

La *precision* misura, relativamente ai dati, il rapporto tra veri positivi (n° di volte che la direzione è stata rilevata in modo corretto) e la somma tra i veri positivi e falsi positivi(n° di volte che la direzione è stata rilevata, ma in modo errato).

$$\text{Precision} = \frac{\text{n° veri positivi}}{\text{n° veri positivi + n° falsi positivi}}$$

Recall

La *recall* misura, relativamente ai dati, il rapporto tra i veri positivi e la somma tra vari positive e falsi negativi(direzione dell'occhio non riconosciuta).

$$\text{Precision} = \frac{\text{n° veri positivi}}{\text{n° veri positivi + n° falsi negativi}}$$

F-measure

La *F-measure* rappresenta il rapporto tra le due precedenti appena descritte.

$$\text{F-measure} = \frac{\text{n° veri positivi}}{\text{n° veri positivi} + \text{n° falsi negativi}}$$

Tutti i risultati delle formula precedenti sono compresi tra 0 e 1 e sono considerati migliori quanto più si avvicinano al valore 1.

4.2 Risultati

Le metriche sopra-descritte sono state utilizzate in particolare per 5 video, con differenti soggetti, calcolando per ognuno:

1. N° di frame totali.

2. N° di frame in cui il soggetto guarda al centro

3. N° di frame in cui il soggetto guarda a destra

4. N° di frame in cui il soggetto guarda a sinistra

Una volta ottenuti questi dati è stato opportunamente modificato l'algoritmo, per ognuno dei video, in modo tale da poter calcolare i successi e gli insuccessi di riconoscimento della direzione; i dati sono stati quindi convertiti in un file "csv" per poter essere analizzati facilmente con fogli di calcolo.

Video n°1

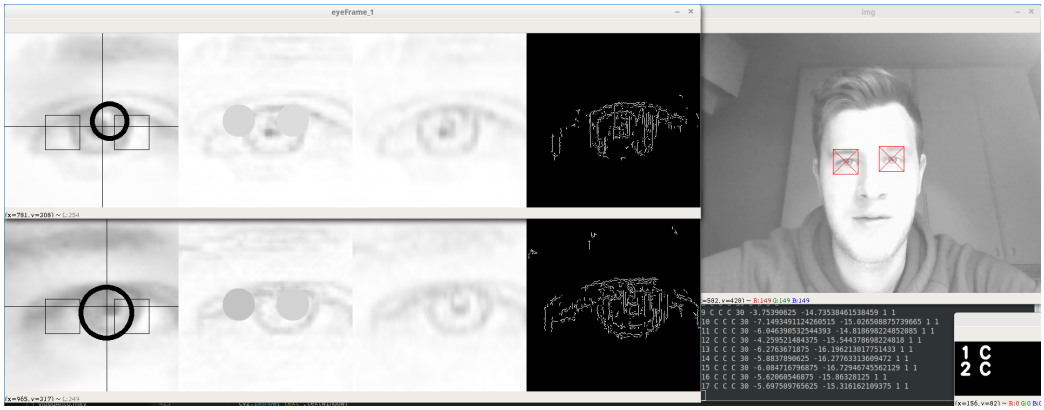


Figura 4.1: Screenshot dell'applicazione (video 1)

Figura 4.2: Dati video 1

Precision	Recall	F-measure
0,90	0,88	0,89

Video n°2

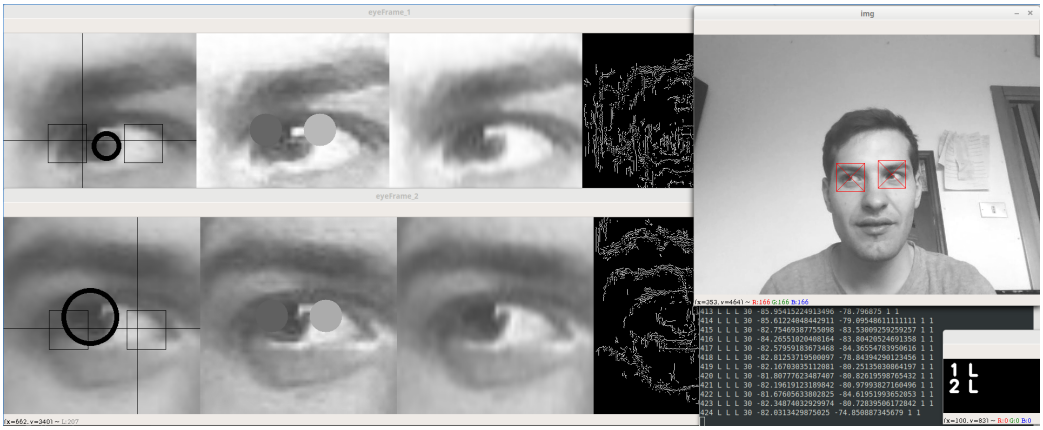


Figura 4.3: Screenshot dell'applicazione (video 2)

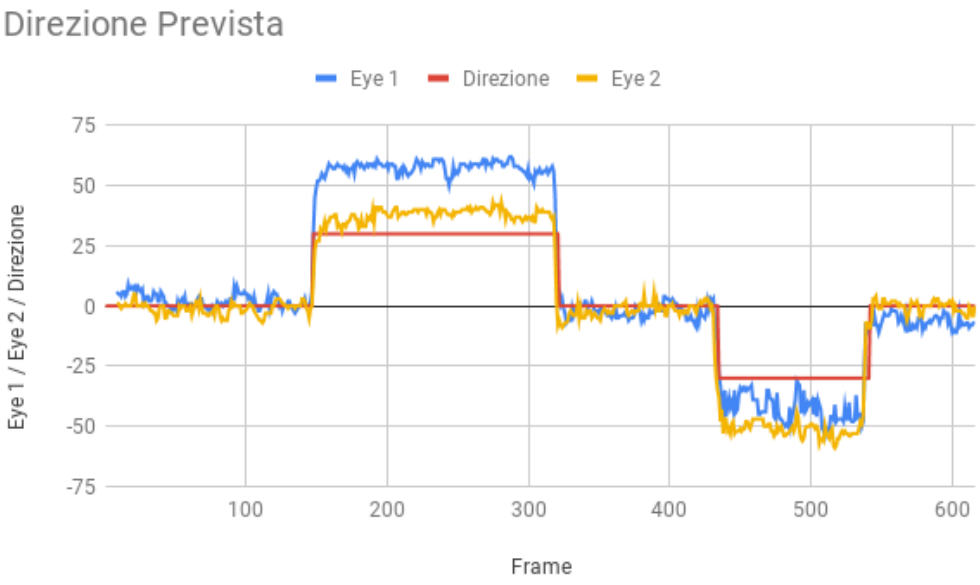


Figura 4.4: Dati video 2

Precision	Recall	F-measure
0,99	0,97	0,98

Video n°3

Figura 4.5: Screenshot dell'applicazione (video 3)

Figura 4.6: Dati video 3

Precision	Recall	F-measure
0,80	0,79	0,8

Video n°4

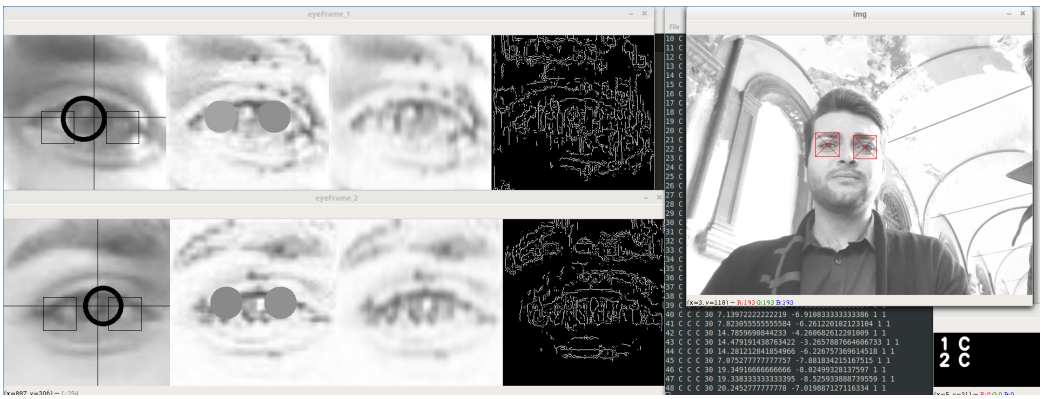


Figura 4.7: Screenshot dell'applicazione (video 4)

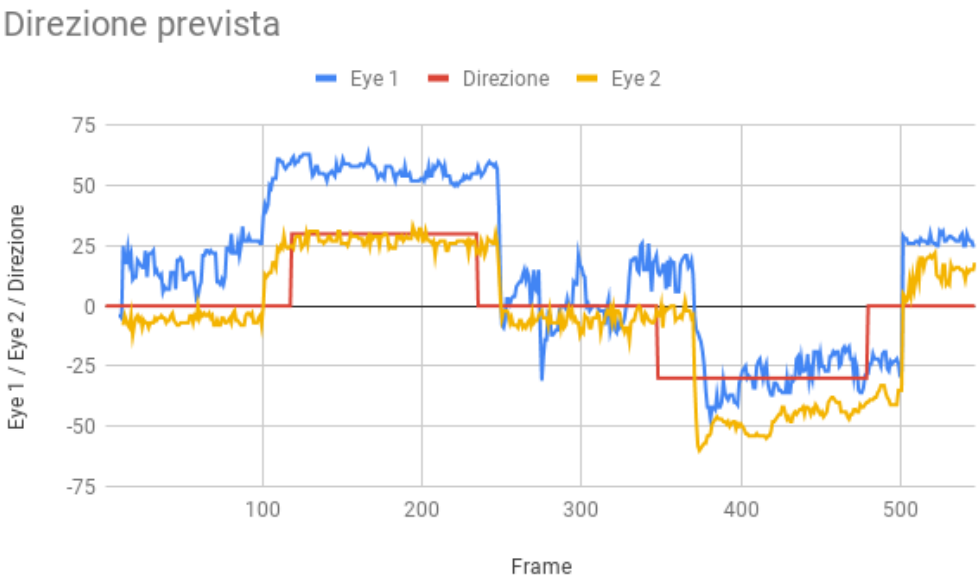


Figura 4.8: Dati video 4

Precision	Recall	F-measure
0,76	0,74	0,75

4.2.1 Tabella riassuntiva

	N° Frame	Eye 1	Eye 2	**Precision**	**Recall**	**F-measure**
Video n° 1	484	430	216	0,76	0,74	0,75
Video n° 2	616	473	466	0,99	0,97	0,98
Video n° 3	547	384	464	0,80	0,79	0,80
Video n° 4	546	362	344	0,76	0,74	0,75

Nella Tabella 4.2.1 sono stati evidenziati e riassunti i risultati relativi ai video. Come si può notare Precision, Recall e F-measure sono sempre sopra il 75% in tutti i video, dimostrando che l'algoritmo ha un buona efficacia. I risultati possono variare perché, come già detto, fattori esterni (ad esempio, luminosità esposizione, ecc..) possono influenzare il riconoscimento; nonostante queste variazioni l'efficacia raggiunta è molto alta.

Conclusioni

L'oggetto del presente studio è stata ricerca di un algoritmo più efficace possibile nel riconoscere la direzione dello sguardo.
Ho innanzitutto svolto ricerche di software già esistenti al fine di valutare se esistessero o meno algoritmi di Eyes Tracking con fotocamere di uso comune, come ad esempio quelle degli smartphone. Questa riflessione è importante proprio perché lo scopo è quello di non utilizzare hardware dedicati ma consentire l'uso dell'algoritmo sfruttando i dispositivi già diffusi negli usi comuni.
Notata l'assenza di algoritmi efficaci il lavoro è stato orientato appunto alla creazione di tale algoritmo: l'obbiettivo prefissato era quindi l'individuazione della direzione dello sguardo in modo approssimativo.
Sono stati creati diversi algoritmi, sopra illustrati, valutandone di ognuno l'efficacia e l'efficienza. I vari percorsi seguiti sono stati in parte progressivi e in parte paralleli: Per alcuni test è stato possibile individuare aspetti positivi che potevano essere utili e altri negativi che rendevano l'efficacia dell'algoritmo non soddisfacente rispetto agli obbiettivi prefissati; Alcuni test invece sono stati svolti in parallelo poiché prevedevano percorsi differenti.
Alla luce delle varie prove è stato quindi definito un algoritmo finale che rispecchia i requisiti iniziali.
Di conseguenza, rispetto agli obbiettivi prefissati i risultati possono essere ritenuti raggiunti: la precision, recall e F-measure, come mostrato nel precedente capitolo sono sufficientemente alte.
Inoltre la scelta del linguaggio Python e del framework OpenCV permettono di avere una portabilità notevole consentendo all'algoritmo di essere esportato su varie piattaforme e sistemi operativi come ad esempio Android.

Sviluppi futuri

Le possibili strade sulla quale si potrebbe lavorare in futuro possono essere riassunte in 2 direzioni:

1. Efficienza ed efficacia dell'algoritmo

   - Come già descritto, la struttura di OpenCV permette di sfruttare al meglio l'hardware sul quale viene installato; questo consente di adattarsi agli sviluppi futuri della tecnologia. Ad esempio analizzare una grande mole di dati che attualmente è difficile fare a causa del costo elevato di elaborazione.
   - L'efficacia dell'algoritmo può essere ancora migliorata. Un esempio pratico potrebbe essere la scelta della "soglia del colore" con la quale viene stabilita la direzione. Ovvero, attualmente la soglia è statica ed è scelta in base a delle valutazioni basate sui dati raccolti; rendere questa soglia dinamica potrebbe migliorare notevolmente l'efficacia. Per farlo si potrebbero usare algoritmi di *learning* o sfruttare regole statistiche.(adriana mi aveva detto una cosa interessante, ricordami di chiederglielo)

2. Applicazioni
   Come illustrato nel capitolo "stato dell'arte" l'utilizzo di un algoritmo di Eyes tracking può avere moltissime applicazioni. In particolare quello creato, sfruttando la portabilità e l'hardware di uso comune, potrebbe aprire nuove strade a soluzioni già esistenti riducendone i costi. Un esempio pratico può essere quello dell'Automotive. Oggi sempre più persone utilizzano come strumento di navigazione il proprio smartphone sul cruscotto: Utilizzando l'Eye tracking si potrebbe fornire una maggiore sicurezza alla guida, valutando lo sguardo del guidatore e prevenire ad esempio colpi di sonno accidentali, emettendo un alert sonoro o visivo. Oppure prevedere la disattivazione automatica dello smartphone qualora il guidatore tenesse per troppo tempo lo sguardo su di esso.

Appendice

Algoritmo finale:

```
import numpy as np
import cv2, sys, getopt
import time, numpy
from parameters import *
from checker import *

#livello zoom per finestra occhio
zoom=7

face_cascade = cv2.CascadeClassifier('
    haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
cap = cv2.VideoCapture(myVideoFile)
#funzione per debugging
def prettyPrintAll(allEyesArray,string):
  print "\n-----",string,"-------"
  for x in allEyesArray:
    for i in x:
      print i
    print "..................."
    stringColorAverage=" - "
    for colorAverage in x[0][3]:
      stringColorAverage=stringColorAverage+
   getDirectioByNumber(colorAverage)+" - "
    print stringColorAverage + "\n"
  print "----------------------------\n"

#stabilisce la direzione dell'occhio mediante una soglia
def getDirectioByNumber(number):
  if number>differenceParams:
    return "R"
  elif number< -differenceParams:
    return "L"
  else:
    return "C"

```

```
35 #controlla se i nuovi dati riguardano vecchi occhi in base
       all'offset
36 def checkEyeOffset (oldData, newData, offset):
37   if len(newData) != len(oldData) != 4:
38     print "ERROR on len()"
39     return False
40   for i in range(0,4):
41     if newData[i] > oldData[i] +offset or newData[i] <
       oldData[i] - offset:
42       return False
43   return True
44
45 #aggiorna la lista delle coordinate dell'Eye
46 def updateEyeList(singleEyeList,newData):
47   if len(singleEyeList)==0:
48     #priority
49     priority=[5,0,0,0]
50     singleEyeList.insert(0,priority)
51   #aggiunge in testa
52   singleEyeList.insert(1,newData)
53   #rimuove l'ultimo se va oltre "n_frameMemory"
54   if len(singleEyeList)> n_frameMemory +1:
55     singleEyeList.pop()
56   return singleEyeList
57
58 #aggiorna la pririta' mantenendola tra max e min
59 def updatePriority(array,index,value):
60   if array[index]<=max_priority and value>0:
61     array[index]=array[index]+value
62   elif array[index] >= min_priority and value<0:
63     array[index]=array[index]+value
64   return array[index]
65 #cerca l'occorrenza
66 def searchIndex(array,newData):
67   index=None
68   onlyOneEye=False
69   for i in range(0,len(array)):
70     if checkEyeOffset(array[i][1],newData,offset) and not
       onlyOneEye:
71       index=i
72       onlyOneEye=True
73       #aumenta priorita'
74       updatePriority(array[i][0],0,len(allEyesArray))
75     else:
76       #diminuisce priorita'
77       updatePriority(array[i][0],0,-1)
78   return index
79
80 #cerca se i nuovi dati sono presenti
```

```
81 def checkIfEye (allEyesArray, newData):
82   foundedEye=False
83 
84   index = searchIndex(allEyesArray,newData)
85   #-------TMP----------
86   if index is None:
87     indexTmp=searchIndex(tmpAllEyesArray,newData)
88     if indexTmp == None:
89       #nuovo dato, nuova lista occhio in tmpAllEyesArray
90       newList=[]
91       newList = updateEyeList(newList,newData)
92       tmpAllEyesArray.insert(0,newList)
93       indexTmp=0
94     else:
95       #se e' nel tmp
96       tmpAllEyesArray[indexTmp] = updateEyeList(
    tmpAllEyesArray[indexTmp],newData)
97 
98     #inserisco nell'array ufficiale se ho gia n_frame
99     if len( tmpAllEyesArray[indexTmp]) >= n_frameNewEye:
100      allEyesArray.insert(0,tmpAllEyesArray[indexTmp])
101      del tmpAllEyesArray[indexTmp]
102  #-------OFFICIAL-------
103  else:
104    #aggiorna la lista di quell'occhio
105    allEyesArray[index] = updateEyeList(allEyesArray[index],
    newData)
106 
107 #copia solo gli occhi con priorita' richiesta
108 def checkRightEyes(allEyesArray):
109   tmp=[]
110   for eye in allEyesArray:
111     if eye[0][0] > min_priority:
112       tmp.append(eye)
113   return tmp
114 
115 #media pesata sulle coordinate
116 def getFinalCoordinates(singleEyeList):
117   peso = len(singleEyeList)*pesoJump
118   pesoTot = 0
119   somma=0
120   x=y=h=w=0
121   for i in range(1,len(singleEyeList)):
122     x = x + (singleEyeList[i][0]*peso)
123     y = y + (singleEyeList[i][1]*peso)
124     h = h + (singleEyeList[i][2]*peso)
125     w = w + (singleEyeList[i][3]*peso)
126     pesoTot = pesoTot + peso
127     peso = peso - pesoJump
```

```
128
129   x=x/( pesoTot )
130   y=y/( pesoTot )
131   h=h/( pesoTot )
132   w=w/( pesoTot )
133   return x,y,h,w
134
135 #ritorna le ultime 10 posizione del riconoscimento
136 def save_pupil_XY(last10Pupil,x,y):
137   if type(last10Pupil) is int:
138     last10Pupil=[[x,y]]
139   else:
140     last10Pupil.insert(0,[x,y])
141     if len(last10Pupil) > 10:
142       last10Pupil.pop()
143   return last10Pupil
144
145 #funzione usata per disegnare
146 def drowLines(x,y,h,w,img):
147   #ex,ey, 1 punto in alto a sinistra - ew,eh larghezza e
      altezza
148   cv2.rectangle(img, (x,y), ((x+w),(y+h)), (0,0,255),1) #
      rettangolo rosso
149   cv2.line(img, (x,y), ((x+w,y+h)), (0,0,255),1)#linea per x
150   cv2.line(img, (x+w,y), ((x,y+h)), (0,0,255),1)#linea per x
151
152
153 def debugPriority():
154   list=[]
155   for x in allEyesArray:
156     list.append(x[0][0])
157   return list
158
159 #chiude le finestre se non c'e' piu' l'occorrenza nell'array
160 def closePupilWindows(len_allEyesArray,openedWindows):
161   r=dict(openedWindows)
162   if (len_allEyesArray < len(openedWindows)):
163     cv2.destroyAllWindows()
164     r={}
165   return r
166
167 #media asse x o y della pupilla
168 def get_average(data,asseXoY):
169   if asseXoY != "x" and asseXoY !="y":
170     print "ERRORE MEDIA ASSE ERRATO"
171     return 0
172   if data != 0:
173     tot=0
174     for i in data:
```

```
175       if asseXoY=="x":
176         tot=tot+i[0]
177       else:
178         tot=tot+i[1]
179     return tot/len(data)
180   else: return 0
181 
182 #per ogni occhio stampa la coordinata della pupilla e la
    larghezza della finestra riferita all'ultimo frame
183 def distanceFuntion(allEyesArray):
184   for i in allEyesArray:
185     print "binary"
186     binaryFunction(i[0][3], i[0][3]*zoom)
187     print "percentage"
188     percentagePoint = percentageFunction(i[0][3], i[0][3]*
    zoom)
189     print percentagePoint
190     print "hyperboleFunction"
191     hyperboleFunction(percentagePoint)
192   print "\n\n"
193 
194 def binaryFunction(x,width):
195   center=width/2
196   binaryPoint=(center)*30/100 #(30% punto di inversione
    binaria)
197   if x > binaryPoint and x<center+binaryPoint:
198     print binaryPoint,center,center+binaryPoint,width," ",x,"
    =",1
199   else:
200     print binaryPoint,center,center+binaryPoint,width," ",x,"
    =",0
201 
202 def percentageFunction(x,width):
203   center=width/2
204   #proporzione "width-center : 100 = x-center : INCOGNITA"
205   percentage = 100 * (x-center) / (width-center)
206   return percentage
207 
208 def hyperboleFunction(percentagePoint):
209   print 1/float(percentagePoint) * 100
210 
211 def save_last_direction(array,averageSX,averageDX):
212   difference_Color=averageSX-averageDX
213   if array == 0:
214     array = []
215   if len(array)>=5:
216     array.pop()
217 
218   array.insert(0,difference_Color)
```

```
219   return array
220
221 def main(argv):
222   global allEyesArray
223   global openedWindows
224   global apertura_finestre
225   global turbo_mode
226   global arguments
227   try:
228     opts, args = getopt.getopt(argv,"",["canny1=","canny2=","
      blur1=","blur2=","clahe=","pupil1=","pupil2=","window=","
      turbo="])
229   except getopt.GetoptError:
230     print 'error in args'
231     sys.exit(2)
232
233   for opt, arg in opts:
234     stringArg = opt[2:]
235     if stringArg in arguments:
236       arguments[stringArg]=int(arg)
237
238     else:
239       if stringArg == "window":
240         if arg=="false":
241           apertura_finestre=False
242         elif arg=="true":
243           apertura_finestre=True
244
245       if stringArg == "turbo":
246         if arg=="false":
247           turbo_mode=False
248         elif arg=="true":
249           turbo_mode=True
250   #-----------------------------------------------
251   frame=0
252   while (1):
253     ret, img = cap.read()
254     frame=frame+1
255     if ret == True:
256       gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
257       img = cv2.cvtColor(gray, cv2.COLOR_GRAY2BGR)
258       pupilFrame=gray
259       clahe=gray
260       blur=gray
261
262       id_pupilWindow=1
263       yDesktopPosition=0
264
265       #riconoscimento occhi
```

```
266       eyes = eye_cascade.detectMultiScale(img,1.3,10)
267       #per ogni eye nel frame
268       for eye in eyes:
269         #cerca se l'occhio e presente o meno e aggiunge a tmp
      o aggiorna i dati
270         checkIfEye(allEyesArray,eye)
271         #rimuove gli occhi con bassa priorita'
272         allEyesArray=checkRightEyes(allEyesArray)
273       #ogni occhio nell'array
274       for singleEyeList in allEyesArray:
275         x,y,h,w= getFinalCoordinates(singleEyeList)
276
277         #ritaglio ,ridimensionamento e posizione dell'occhio
278         pupilFrame = (gray[y:y+h, x:x+w])
279         #zoom x7
280         pupilFrame = cv2.resize(pupilFrame,None,fx=zoom,fy=
    zoom)
281         gray2=pupilFrame.copy()
282
283         #effetti
284         #clahe
285         cl1 = cv2.createCLAHE(clipLimit=1.0, tileGridSize=(
    arguments["clahe"],arguments["clahe"]))
286         clahe = cl1.apply(pupilFrame)
287         #sfuma
288         blur = cv2.blur(clahe,(arguments["blur1"],arguments["
    blur2"]))
289         #rileva contorni
290         canny = cv2.Canny(blur, arguments["canny1"],arguments
    ["canny2"])
291
292         #riconoscimento iride
293         circles = cv2.HoughCircles(canny ,cv2.HOUGH_GRADIENT
    ,1,20,param1=arguments["pupil1"],param2=arguments["pupil2"
    ],minRadius=20,maxRadius=50) #houghcircles
294
295         if circles is not None:
296         #converte le coordinate del cerchio in x,y,r
297           circles = np.round(circles[0, :]).astype("int")
298           #i cerchi riconosciuti sono ordinati in base alla "
    correttezza"
299           #di conseguenza il primo elemento e' quello piu'
    corretto
300           cx=circles[0][0]
301           cy=circles[0][1]
302           cr=circles[0][2]
303
304           #salvataggio cerchio riconosciuto in singleEyelist
    [0,x,0,0]
```

```
305         singleEyeList[0][1]=singleEyeList[0][1]+1
306
307
308         if cr != 0:
309           #salvo le coordinate della pupilla in
    alleyesArray [0,0,[x,y],0] come vettore
310           singleEyeList[0][2]=save_pupil_XY(singleEyeList
    [0][2],cx,cy)
311
312           cv2.circle(pupilFrame, (cx, cy), cr, (0, 255, 0),
     8)
313           # cv2.line(pupilFrame, (0,(h*zoom)/3), ((w*zoom,(
    h*zoom)/3)), (0,255,0),1)
314           # cv2.line(pupilFrame, (0,3*(h*zoom)/4), ((w*zoom
    ,3*(h*zoom)/4)), (0,255,0),1)
315
316       #calcolo media delle coordinate della pupilla sull'
    asse x
317       average_pupilX = get_average(singleEyeList[0][2],"x")
318       average_pupilY = get_average(singleEyeList[0][2],"y")
319       cv2.line(pupilFrame, (average_pupilX,0), ((
    average_pupilX,(h*7))), (0,255,0),1)
320       cv2.line(pupilFrame, (0,average_pupilY), ((w*7),
    average_pupilY ), (0,255,0),1)
321
322       #           |
323       #           a
324       #           |
325       #
326       #  >--c---  o-----d-----o
327       #          |           |
328       #          b           |
329       #          |           |
330       #          o-----------o
331       #
332
333       h_dynamyc=h*zoom/5
334       w_dynamyc=w*zoom/5
335       c_dynamyc=w_dynamyc/2
336
337       #COORDINATE SX
338       a_sx= h*zoom/2 - 10
339       b_sx= a_sx + h_dynamyc
340       c_sx= w*zoom/2 - c_dynamyc - w_dynamyc +10
341       d_sx= c_sx + w_dynamyc
342       #COORDINATE DX
343       a_dx= h*zoom/2 - 10
344       b_dx=a_dx + h_dynamyc
345       c_dx= w*zoom/2 + c_dynamyc +10
```

```
346         d_dx= c_dx+w_dynamyc
347 
348         misure=[a_sx,b_sx,c_sx,d_sx]
349 
350         #DISEGNO SUL CLAHE
351         sx = gray2.copy()[a_sx : b_sx, c_sx : d_sx ]
352         dx = gray2.copy()[a_dx : b_dx, c_dx : d_dx ]
353         try:
354           averageSX=sx.mean(axis=0).mean(axis=0)
355           averageDX=dx.mean(axis=0).mean(axis=0)
356         except Exception as e:
357           print e
358 
359         singleEyeList[0][3]= save_last_direction(
    singleEyeList[0][3],averageSX,averageDX)
360 
361         cv2.circle(clahe,(h*zoom/2 - 50,w*zoom/2), 30,
    averageSX, -1)
362         cv2.circle(clahe,(h*zoom/2 + 50,w*zoom/2), 30,
    averageDX, -1)
363 
364         drowSquare(pupilFrame, c_sx,a_sx,w_dynamyc,h_dynamyc
    ,1)
365         drowSquare(pupilFrame, c_dx,a_dx,w_dynamyc,h_dynamyc
    ,1)
366 
367         final = np.hstack((pupilFrame,clahe,blur,canny))
368 
369         #apertura finestra e salvataggio dati
370         if (apertura_finestre):
371           #apertura finestra con nome "id_pupilWindow"
372           cv2.imshow('eyeFrame_'+str(id_pupilWindow), final)
373           cv2.moveWindow("eyeFrame_"+str(id_pupilWindow), 0,
    yDesktopPosition)
374           yDesktopPosition = yDesktopPosition + 340
375           #aggiungo la nuova finestra al dizionario
376           openedWindows[id_pupilWindow]='eyeFrame_'+str(
    id_pupilWindow)
377           openedWindows= closePupilWindows(len(allEyesArray),
     openedWindows)
378 
379           id_pupilWindow=id_pupilWindow+1
380 
381           drowLines(x,y,h,w,img)
382 
383           difference_Color = averageSX-averageDX
384 
385           directionEyes[id_pupilWindow] = difference_Color
386 
```

```
387           font = cv2.FONT_HERSHEY_SIMPLEX
388           textWindow = np.zeros((100,500,3), np.uint8)
389           counter=30
390           for eye in directionEyes:
391             cv2.putText(textWindow, str(eye-1) + " " + str(
    getDirectioByNumber(directionEyes[eye])),(10,counter),
    font, 1,(255,255,255),3,cv2.LINE_AA)
392             counter=counter+30
393
394           cv2.imshow("text",textWindow)
395           cv2.moveWindow("text", 0,1400)
396
397       eye1=eye2=0
398       reco1=reco2=0
399       n1=n2=0
400       if len(allEyesArray)>0:
401         eye1=1
402         n1=allEyesArray[0][0][3][0]
403         reco1=getDirectioByNumber(n1)
404       if len(allEyesArray)>1:
405         eye2=1
406         n2=allEyesArray[1][0][3][0]
407         reco2=getDirectioByNumber(n2)
408
409       #funzione che ha memorizzate le direzioni di sguardo
    del video di riferimento
410       #utile per generare il csv in output
411       videoMio(eye1,reco1,n1,eye2,reco2,n2,frame,
    differenceParams)
412
413
414       if (apertura_finestre):
415         cv2.imshow('img',img)
416         cv2.moveWindow("img", 1400,200)
417         cicle=False
418         k = cv2.waitKey(30) & 0xff
419         if k == ord ('q'):
420           cap.release()
421           cv2.destroyAllWindows()
422           exit()
423         elif k == ord ('p'):
424           prettyPrintAll(allEyesArray,"all")
425         elif k== ord('s'):
426           cicle=True
427
428         while cicle:
429           s = cv2.waitKey(30) & 0xff
430           if s == ord ('s'):
431             cicle=False
```

```
            elif s== ord('p'):
              prettyPrintAll(allEyesArray,"all")
            elif s== ord('o'):
              prettyPrintAll(allEyesArray,"all")
              print misure
    else:
      break


if __name__ == "__main__":
  main(sys.argv[1:])
```

Parametri:

```

#file video
myVideoFile = './titoloVideo.mkv'

#array in cui sono salvati gli occhi trovati (ultimi 10 frame)
allEyesArray =[]

#array per verificare se una nuova rilevazione non sia un falso positivo
tmpAllEyesArray=[]

#numero di frame dell'array
n_frameMemory=10

#numero di frame minimo per essere spostato da "tmp" a "definitivo"
n_frameNewEye = 10

#offset per riconoscimento del vecchio occhio
offset=40

#priorita' per essere tolto dall'array
min_priority=-10
max_priority=10

#jump della media pesata (piu' alto-> piu' importanza all' ultimo)
pesoJump=10

#differenza colori per decidere la direzione
differenceParams = 30

#dizionario con elenco delle finestre aperte
openedWindows ={}

```

```
33 #dizionario per la direzione degli occhi
34 directionEyes = {}
35
36 #parametri per esecuzione in background
37 apertura_finestre = True
38 turbo_mode=False
39
40 #parametri filtri fotografici
41 arguments = {"canny1":16,"canny2":10,"blur1":1,"blur2":16,"
      clahe":24,"pupil1":50,"pupil2":30}
```

Bibliografia

[1] *Che cos'è l'Eye-Tracking*
URL: https://economia.tesionline.it/economia/article.jsp?id=12668

[2] K. Harezlak, P. Kasprowski; *Application of eye tracking in medicine: A survey, research issues and challenges.*
URL: https://www.ncbi.nlm.nih.gov/pubmed/28606763

[3] Vidas Raudonis, Rimvydas Simutis, Gintautas Narvydas; *Discrete eye tracking for medical applications*, in IEEE Conference, 2010
URL: https://ieeexplore.ieee.org/document/5373675

[4] Wikipedia. *Precision and recall*
URL: https://en.wikipedia.org/wiki/Precision_and_recall

[5] Wikipedia. *Gaussian Blur*
URL: https://en.wikipedia.org/wiki/Gaussian_blur

[6] Wikipedia. *Canny edge detector*
URL: https://en.wikipedia.org/wiki/Canny_edge_detector

[7] D. Soriano; *Face Recognition and Tracking with OpenCV*
URL:https://www.domsoria.com/2018/05/face-recognition-and-tracking-con-opencv-prima-parte/

[8] A. Matheus; *Eye Tracking for Mouse Control in OpenCV*
URL:https://picoledelimao.github.io/blog/2017/01/28/eyeball-tracking-for-mouse-control-in-opencv/

[9] K.Krafka, A. Khosla, P. Kellnhofer, H. Kannan, S. Bhandarkar, W. Matusik and A. Torralba; *Eye Tracking for Everyone*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016
URL:http://gazecapture.csail.mit.edu/index.php

[10] T. Hume; *Simple, accurate eye center tracking in OpenCV*
URL:http://thume.ca/projects/2012/11/04/simple-accurate-eye-center-tracking-in-opencv/

Ringraziamenti

"Ringrazio innanzitutto il Dott. Bedogni, con il quale ho collaborato per questo lavoro, che mi dedicato del tempo dandomi preziosi consigli e insegnato molto per quanto riguarda tutto l'aspetto tecnico intorno a cui ruota questo elaborato.
Un ringraziamento speciale va a tutti gli amici. Ai Compagni di corso quali Andrea, Davide Natale e tutti gli altri con cui ho condiviso studi e tempo libero. Agli amici di più vecchia data, che mi sono sempre stati vicini nonstante la distanza.
Il ringraziamento più grande lo dedico alla mia famiglia che mi ha sempre supportato nelle scelte personali ed aiutato nei momenti più difficili. "

A Gianni, Anna e Adriana.