POLITECNICO DI MILANO
Polo Regionale di Como
Facoltà di Ingegneria dell'Informazione
Corso di Studi in Ingegneria Informatica

Analisi di immagini per l'identificazione del volto e dei suoi movimenti

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Ing. Andrea Bonarini
Correlatore: Prof. Matteo Matteucci

Tesi di Laurea di:
Cristian Mandelli, matricola 671783

Anno Accademico 2008-2009

A mia mamma e mio papà
che mi hanno sempre sostenuto
in tutte le mie scelte.
A mio nonno Angelo

Sommario

Le tecnologie di analisi del volto sono state implementate in sistemi di face recognition e emotion recognition, ovvero in tutti quei sistemi che hanno la necessità di localizzare il volto in un'immagine o in un video.

Lo scopo di questa tesi è quello di proporre una tecnologia a basso costo che consenta di tracciare ed analizzare i movimenti del volto e dei suoi componenti, quali occhi e sopracciglia, da un filmato registrato da una webcam.

I risultati finali mostrano che utilizzando una sola webcam con una risoluzione di 640×480 è possibile estrarre l'immagine del volto dal filmato registrato, localizzando nell'immagine ottenuta gli occhi e le sopracciglia.

Ringraziamenti

Ai prof. A. Bonarini e M. Matteucci per avermi seguito passo passo per tutto il corso del presente progetto e per la passione per il loro lavoro che hanno saputo trasmetterimi. Da loro ho avuto il giusto stimolo per continuare gli studi anche dopo la laurea.

Un ringraziamento particolare anche ai miei "soci"; Giacomo per le numerose consulenze tecniche su computer, hifi, macchine e per le giornate passate a giocare con il wii in taverna; Simone, per le sue battute anni 50', per la compagnia nelle giornate di studio e per le scarse prestazioni sulla fascia nelle partite di calcetto; Roberto, per essere svizzero e avermi fatto risparmiare tantissimo negli ordini online e per l'aiuto nella preparazione del TOEFL; Massimo la pietra dello scandalo del gruppo! per le risate volontarie e sopratutto involontarie che ci hai regalato; Tristano, per il forum, per le chiaccherate in msn (mi devi ancora una birretta!!). Li ringrazio tutti per i bei momenti di relax (tanti) e di studio (pochi) passati in questi anni.

Grazie alla Lauretta e a Simone per averci messo letteralmente "la faccia". Ammetto che sono state due ottime cavie per testare il mio lavoro.

Grazie alla DeDe, per aver reso meravigliosi gli ultimi due anni di università, per la forza che mi ha saputo trasmettere per superare le barriere più "cificili", e per essermi stata sempre accanto. Ti Amo!!. P.S. visto che ci siamo riusciti??

Un ringraziamento particolare anche alla tua famiglia per avermi "sopportato" a casa durante i periodi d'esame e durante lo sviluppo di questo lavoro.

Infine e più importante, ringrazio la mia famiglia, per avermi seguito e confortato nei momenti difficili nel corso dei miei studi, li ringrazio di cuore e gli dedico questo mio primo traguardo.

Indice

Capitolo 1

Introduzione

"Ardo dal desiderio di spiegare, e la mia massima soddisfazione è prendere qualcosa di ragionevolmente intricato e renderlo chiaro passo dopo passo. è il modo più facile per chiarire le cose a me stesso."

Isaac Asimov

Le ricerche svolte nel campo della computer vision sono in gran parte dedicate all'implementazione di sistemi per il riconoscimento di movimenti o per l'identificazione di espressioni del volto umano e molte di queste sono condotte con lo scopo di aumentare la comodità e la sicurezza delle automobili. Lo scopo del sistema qui proposto è quello di offrire un economico e robusto algoritmo di analisi del volto umano, integrabile con un sistema (hw e sw) utilizzato per l'analisi delle emozioni durante la guida, al fine di fornire sensazioni di guida migliori al conducente.

Per ottenere una buona interazione uomo-macchina, ci si pone l'obiettivo di analizzare quante più risorse disponibili. Un contributo importante in questa direzione potrebbe venire da un sistema basato sull'analisi di segnali provenienti da sensori posti sul volante e sul sedile, affiancati ad un sistema di analisi dei movimenti del volto. Tale sistema potrebbe portare al riconoscimento di elevati livelli di stress in modo da modificare il comportamento del veicolo ed offrire sensazioni migliori all'autista.

Per questo motivo, è stato sviluppato un sistema che permette di catturare i movimenti del volto e dei suoi componenti, all'interno di in un filmato. Per il raggiungimento degli obiettivi sopra enunciati, sono stati utilizzati sistemi di face detection per il riconoscimento del volto in un filmato e algoritmi di blob analysis per l'estrazione delle informazioni circa occhi e sopracciglia. Il sistema presentato conduce uno studio basato su tre livelli: partendo dall'analisi del filmato "frame-by-frame" , si passa poi alla ricerca

ed all'estrazione della posizione del volto al suo interno. La seconda fase consiste nella localizzazione degli occhi e delle sopraciglia all'interno del volto estratto al passo precedente; la terza fase, infine, consiste nell'estrazione di tutte le informazioni circa i movimenti e le dimensioni delle caratteristiche localizzate (Figura 1.1).

Figura 1.1: Rappresentazione delle tre fasi di analisi.

I risultati ottenuti mostrano che per tali analisi non si necessita di immagini ad alta risoluzione, ma, con le attuali tecnologie, è possibile condurre un'analisi che varia dai 10 ai 15 frame per secondo, localizzando in modo preciso le posizioni delle varie feature all'interno del video, anche in casi in cui il volto assuma particolari pose. Questi mostrano inoltre che l'analisi può essere condotta indipendentemente dalle caratteristiche somatiche della persona.

La tesi è suddivisa in quattro capitoli fondamentali e si alterna tra aspetti tecnici molto dettagliati (capitoli 3 e 4) e altri più discorsivi, che analizzano alcune proposte offerte dalla letteratura negli ambiti della face recognition e dell'eye tracking (capitolo 2). Essa è così strutturata:

- nel **capitolo 2**, *Stato dell'arte della tecnologia di analisi del volto*, viene presentato lo stato dell'arte offerto dalla letteratura in merito a due problematiche fondamentali: la localizzazione degli occhi all'interno di un filmato o di un'immmagine; l'estrazione degli occhi dall'immagine del volto.

- nel **capitolo 3**, *Face Detection*, viene proposto l'algoritmo scelto per l'estrazione dell'immagine del volto dal filmato in analisi. Nella prima parte del capitolo viene analizzato l'algoritmo e gli aspetti ad esso correlati dal punto di vista teorico; nella seconda parte vengono riportate, invece, alcune parti del codice sorgente che hanno permesso l'implementazione delle sezioni dell'algoritmo più significative.

- nel **capitolo 4**, *Eye Tracking*, viene proposto l'algoritmo scelto per l'estrazione della posizione degli occhi e delle sopracciglia all'interno

dell'immagine del volto. Nella prima parte del capitolo viene analizzato l'algoritmo e le problematiche ad esso collegate dal punto di vista teorico; nella seconda parte vengono riportate, invece, alune parti del codice sorgente che hanno permesso l'implementazione delle sezioni dell'algoritmo più significative.

- nel **capitolo 5**, *Verifiche sperimentali e conclusioni*, vengono presentati i risultati ottenuti nei test effettuati per la valutazione della robustezza dell'algoritmo ottenuto.

Capitolo 2

Stato dell'arte della tecnologia di analisi del volto

"Un giorno avranno dei segreti, un giorno avranno dei sogni..."

Io Robot

Nell'interazione uomo-uomo la comunicazione visiva, come ad esempio la postura del corpo, i gesti e sopratutto il viso, contribuiscono enormemente alla qualità e all'efficienza della comunicazione. Lo scambio del solo contenuto verbale, privato di tutte le precedenti caratteristiche, porterebbe ad interpretazioni errate e fraintendimenti. Per questo motivo, per tutte le applicazioni tecnologiche che vogliono analizzare e migliorare l'interazione e la comunicazione uomo-uomo o uomo-computer, l'analisi del volto è una tappa di primaria importanza. Il solo volto umano fornisce sia informazioni come il genere, l'età, le origini etniche, sia informazioni relative ad emozioni come rabbia, felicità, paura, stress, concentrazione etc.

La tecnologia di analisi del volto è da sempre stata oggetto di attente ricerche da parte degli scienziati e rappresenta ancora oggi una delle più grandi sfide nella moderna computer vision. Infatti, già agli inizi degli anni '60, si trovano i primi lavori a partire da sistemi di face recognition (identificazione automatica di una persona in un video o in una immagine) per passare poi negli anni '70 all'eye-tracking con i primi studi del problema delle saccadi (rapidi ed improvvisi movimenti degli occhi), della pupilla o della rotazione della testa [4], [24], [25]. Sempre agli inizi degli anni '70 risalgono inoltre i primi studi di P. Ekman sulle espressioni del volto umano [6]. A quasi cinquant'anni di distanza, i passi fatti dalla tecnologia sono notevoli, permettendo un forte abbattimento dei costi di analisi e miglioramento

delle prestazioni. L'analisi del volto e delle sue componenti principali (quali occhi, sopraciglia etc.) possono trovare applicazione in svariati campi. Tra i principali possiamo citare:

- **Face Recognition:** essa è diventata, negli ultimi 10 anni, una popolarissima area di ricerca all'interno della computer vision. Un aspetto comune a tutte le problematiche di face recognition può essere formulato come segue: dato un video o un'immagine di una determiata scena, identificare o verificare una o più persone nella scena stessa attraverso l'utilizzo di un data base di volti [35];

- **Face emotion recognition:** sistemi che hanno come obiettivo quello di identificare determinate emozioni a partire da espressioni facciali visualizzate in video o in sequenze di immagini [34];

- **Affective computing:** essa è una particolare branca di studi dell'intelligenza artificiale la quale ha come obiettivo quello di studiare e/o creare sistemi e apparecchiature tecnologiche in grado di identificare, interpretare e processare le emozioni umane. Uno dei principali obiettivi di questa particolare corrente dell'intelligenza artificiale è, senza dubbio, quello di far interpretare alle macchine lo stato emotivo dell'uomo in modo da adattare il loro comportamento in riposta a quel particolare stato d'animo [33].

In quest'ultimo ambito (Affective Computing) si colloca il presente progetto che vuole offrire una particolare tecnologia di analisi del volto umano e delle sue componenti principali. In questo capitolo verranno presentati i diversi approcci offerti dalla letteratura per l'identificazione del volto, degli occhi e delle sopraciglia e per tenere traccia dei loro movimenti.

Verranno ora brevemente introdotte le librerie OpenCV le quali hanno giocato un fondamentale ruolo nello sviluppo e nella realizzazione di numerosi progetti di Computer Vision.

2.1 Le librerie OpenCv

Le librerie OpenCV hanno giocato un importante ruolo nella crescita e nello sviluppo della computer vision. Con il loro orientamento alla visione in tempo reale, le openCV hanno aiutato studenti e professionisti del settore nello sviluppo e nell'implementazione di progetti relativi alla computer vision e al machine learning, cosa che prima era possibile solamente in avanzati laboratori di ricerca.

Ma che cosa sono le OpenCV?
Le OpenCV sono delle librerie open-source che realizzano funzionalità di computer vision. Queste librerie sono scritte in C e C++ e sono supportate dai principali sistemi operativi quali Linux, Windows e Mac OS X. Quest'ultime sono state create ponendo particolare attenzione all'efficienza computazionale per la realizzazione di applicazioni real-time; esse sono state infatti ottimizzate con il tempo e godono ora dei vantaggi dei processori multicore. Per le architetture Intel, inoltre, sono state create delle particolari librerie che consistono in una serie di routine di basso livello atte ad ottimizzare le OpenCV (IPP Integrated Performance Primitives).

Uno dei principali successi raggiunti dalle OpenCV, è stato quello di offrire un più facile accesso alle infrastrutture della computer vision, facilitando e velocizzando la creazione di sofisticate applicazioni. Le librerie OpenCV contengono oltre 500 funzioni che abbracciano le principali aree della computer vision e, in aggiunta, funzioni specifiche per: interfaccia utente, sicurezza, immagini mediche, calibrazione della video-camera, stereo visione e robotica. Visti inoltre i continui collegamenti tra la computer vision ed il machine learning, le OpenCV contengono anche alcune librerie relative a quest'ultimo settore; queste sotto-librerie sono dedicate principalmente al clustering e alla pattern recognition.

Le librerie OpenCV sono suddivise in cinque componenti principali, quattro dei quali sono mostrati nella Figura 2.1.

- **CV:** componente che contiene numerosi algoritmi di image processing e di computer vision;
- **MLL:** librerie di machine learning che contengono numerosi classificatori statistici e strumenti per il clustering;
- **HighGui:** componente che contiene le funzioni di I/O atte alla me-

morizzazione e al caricamento di diverse tipologie di dati (Immagini, video etc.);

- **CXCore:** componente che contiene le strutture base utilizzate dalle librerie;
- **CVAux:** componente che contiene funzioni ed algoritmi riguardanti il riconoscimento del volto e alcuni algoritmi sperimentali.

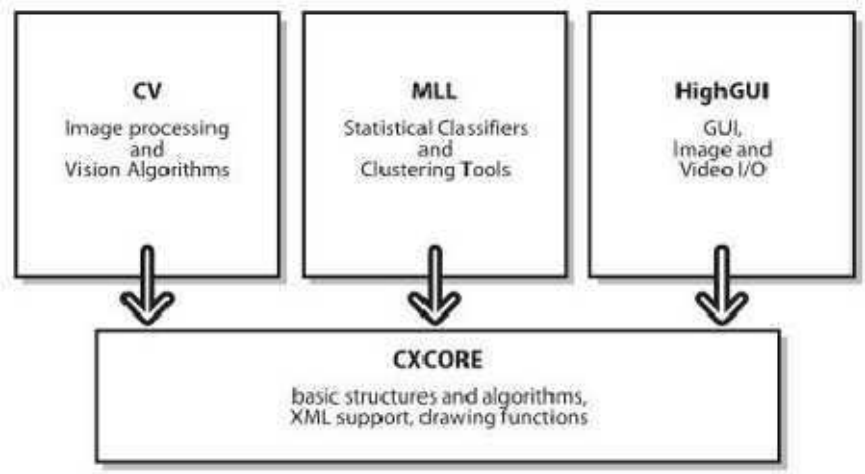


Figura 2.1: Struttura base delle OpenCV

Un'altra caratteristica fondamentale delle librerie OpenCV e delle relative applicazioni è la portabilità. Esse sono state originariamente scritte per funzionare su compilatori basati sulla tecnologia Intel e con il passaggio anche da parte della Apple all'architettura Intel, si può affermare che queste librerie sono supportate da tutti i principali sistemi operativi [15].

2.2 Face Detection

L'immagine del volto è essenziale per i sistemi intelligenti, basati sulla computer vision, che si occupano dell'interazione tra uomo e computer e le ricerche ad esse correlate includono svariati metodi tra i quali face tracking, face recognition, pose estimation ed espression recognition. Tali metodi assumono che il volto contenuto in un'immagine, o in una sequenza di immagini, debba essere localizzato ed identificato. Per la costruzione di un sistema completamente automatico che analizzi le informazioni contenute nell'immagine di un volto, sono richiesti robusti ed efficienti algoritmi di

face detection. Data una singola immagine, l'obiettivo della face detection consiste nell'identificare tutte le regioni dell'immagine contenenti un volto indipendentemente da posizione, orientamento e illuminazione. L'obiettivo proposto genera una problematica in continua mutazione poichè i volti non sono corpi rigidi e hanno un elevato grado di variabilità in dimensione, forma, colore e texture [27]. Sono state sviluppate numerose tecniche per individuare il volto in una singola immagine e l'obiettivo di questo capitolo è quello di catalogare e valutare tali algoritmi.

2.2.1 Metodi Top-Down basati sulla conoscenza

In questo approccio, l'algoritmo per il riconoscimento del volto è stato sviluppato a partire da regole ben definite dagli scienziati che studiano il volto umano. è infatti semplice arrivare ad una esaustiva descrizione di quest'ultimo partendo dalle relazioni che intercorrono tra le varie componenti del volto. Per esempio, un volto appare spesso in un'immagine con due occhi che sono tra loro simmetrici, un naso e una bocca. Le relazioni tra queste caratteristiche possono essere espresse tramite distanza e posizione. Essendo questo un metodo top-down, tutte le caratteristiche che identificano il volto sono estratte per prime e il volto viene estratto basandosi sulle regole sopra citate. Si è soliti utilizzare anche algoritmi di verifica per ridurre i casi di falsi positivi.

Una problematica di tale metodo è rappresentata dalla difficoltà di tradurre in regole le conoscenze circa il volto umano. La definizione di regole troppo restrittive, infatti, potrebbe portare al fallimento nel riconoscimento del volto, ma, allo stesso tempo, regole troppo generali potrebbero portare al riconoscimento di numerosi falsi postitivi. è inoltre difficile estendere tale metodologia per l'identificazione dei volti in diverse pose.

Yang e Huang usarono un metodo di apprendimento di tipo gerarchico [13]: il loro sistema consiste in tre differenti livelli di regole. Al livello più alto, l'immagine in input viene scansionata estraendone tutti i possibili candidati attraverso l'applicazione di alcune regole. Tali regole sono generalmente descrizioni di ciò che deve essere identificato come volto, mentre saranno le regole dei livelli inferiori a definire dettagliatamente le caratteristiche del volto stesso.

Figura 2.2: (a) n=1, immagine originale. (b) n=4. (c) n=8. (d) n=16. Immagine originale a bassa risoluzione. Ogni cella quadrata è formata da $n \times n$ pixel, per ognuno dei quali il valore di intensità è stato sostituito con il valore medio dell'intensità dei pixel nella cella.

- *Livello 1:* L'immagine a bassa risoluzione è utilizzata per la ricerca dei possibili candidati;
- *Livello 2:* Su tutti i candidati viene effettuata un'analisi per il riconoscimento dei bordi dei componenti dell'immagine(edge detection);
- *Livello 3:* I candidati sopravvissuti vengono quindi esaminati nuovamente attraverso nuove regole basate sulle caratteristiche generali del volto, come occhi e bocca.

Il precedente sistema è stato valutato attraverso l'analisi di un set di 60 immagini. Il sistema ha riconosciuto con successo 50 delle immagini utilizzate nel test, mentre ci sono stati 28 casi di falsi positivi dovuti principalmente alla difficoltà nella definizione delle regole. Sebbene il sistema non abbia un elevato grado di riconoscimento, l'idea di usare una gerarchia di immagini a risoluzioni differenti, accompagnate da un set di regole per guidare la ricerca, è alla base degli ultimi lavori nel campo del riconoscimento del volto [23].

Kotropoulus e Pitas [23] presentarono un altro metodo di localizzazione simile al precedente [13]. In questo caso, le caratteristiche del volto vengono identificate e localizzate attraverso un particolare metodo di proiezione utilizzato precedentemente con successo nella localizzazione dei limiti del volto [20]. Detto $I(x, y)$ il valore di intensità di una immagine $m \times n$ nella posizione (x, y), le proiezioni orizzontale e verticale dell'immagine sono definite come:

$$HI(x) = \sum_{y=1}^{n} I(x, y) \text{ e } VI(y) = \sum_{x=1}^{m} I(x, y)$$

Il calcolo di entrambi i profili, viene utilizzato per condurre una ricerca circa alcune caratteristiche del volto. Il *profilo orizzontale* è associato con la localizzazione dei minimi locali (brusche variazioni dei valori di HI) i quali

corrispondono esattamente al lato destro e sinistro del volto. In modo del tutto simile, viene calcolato anche il *profilo verticale* e i minimi locali vengono calcolati per la localizzazione delle labbra, della punta del naso e degli occhi. Tutte le caratteristiche localizzate all'interno dell'immagine di input, costituiscono un probabile volto ovvero un candidato. La Figura 2.3(a) mostra un esempio nel quale i contorni del volto risultano ben localizzati da questo sistema di riconoscimento, mentre la Figura 2.3(b) e Figura 2.3(c) mostrano due casi in cui il sistema utilizzato incontra numerose difficoltà nella localizzazione delle caratteristiche del volto e quindi nella ricerca del volto stesso.

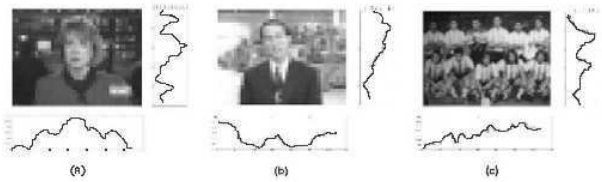


Figura 2.3: (a) e (b) n=8. (c) n=4. Profili orizzontali e verticali. è possibile localizzare un singolo volto ricercando i picchi nei due profili dell'immagine. Tuttavia lo stesso metodo trova alcune difficoltà se applicato in immagini con uno sfondo complesso oppure in presenza di più volti.

2.2.2 Metodi Botton-Up basati sulle caratteristiche del volto

In contrasto con i precedenti metodi basati sulla conoscenza scientifica del volto umano, si è cercato, con questo approccio di utilizzare le caratteristiche fisse del volto per la sua localizzazione. L'idea di base di tale approccio prende spunto dal fatto che l'occhio umano può, con facilità, identificare un volto o un determinato oggetto in diverse pose ed in differenti condizioni di illuminazione, quindi ci devono essere alcune caratteristiche che si preservano anche in condizioni molto variabili. Sono stati presentati numerosi sistemi che localizzano per prime le caratteristiche del volto (come occhi, naso e bocca) per poi inferire la presenza del volto stesso. Tuttavia la principale problematica comune a tutti questi sistemi è dovuta alla difficoltà della localizzazione delle caratteristiche del volto, la quale risulta spesso compromessa dalle continue variazioni dell'illuminazione, per la presenza di ombre e in generale per la presenza di rumore nell'immagine.

Sirohey propose un sistema per la localizzazione e l'estrazione del volto contenuto in una generica immagine in scala di grigi. Le tecniche di segmentazione del volto utilizzate per la sua estrazione, sono basate principalmente sulla struttura ellittica del volto umano e utilizzano sia le informazioni contenute in una mappa dei bordi (edge map) sia alcuni algoritmi per facilitarne l'estrazione. [31]. Un diagramma dell'algoritmo proposto da Sirohey è quello rappresentato in Figura 2.4

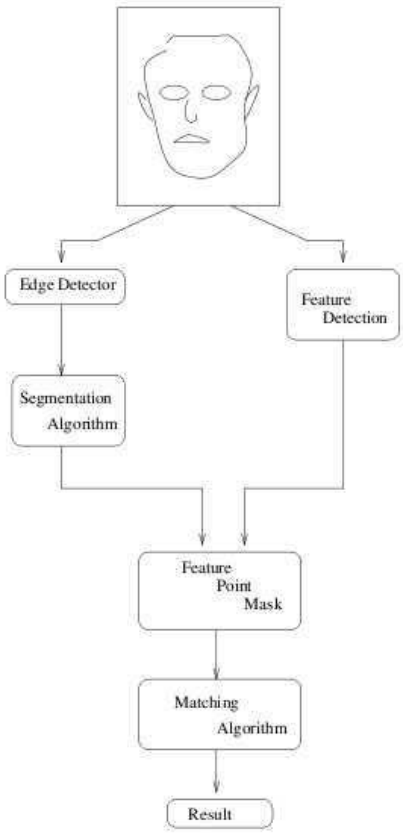


Figura 2.4: Diagramma dell'algoritmo presentato da Sirohey per l'identificazione di un volto in un'immagine.

Il contorno del volto umano può essere generalmente descritto come un ellisse. Per separare la porzione di immagine che contiene il volto, dal resto dell'immagine, Sirohey cercò di trovare e adattare il migliore ellisse nell'immagine di input intorno al volto cercato. Tra i vari metodi possibili per condurre tale analisi venne considerata inizialmente la Trasformata di Hough [3] [30].

La trasformata ellittica di Hough consiste nel trovare, dato un generico punto (x, y) nel piano, i parametri di un ellisse passante per tale punto. I

parametri sopra citati sono il centro (x_0, y_0), il semi-asse maggiore a e il semi-asse minone b dell'ellisse. Il metodo utilizzato è qui riportato:

1. L'equazione dell'ellisse è:

$$\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} = 1$$

2. Definendo $X = x - x_0$ e $Y = y - y_0$ la (1) diventa:

$$\frac{X^2}{a^2} + \frac{Y^2}{b^2} = 1$$

3. Derivando ora tutto rispetto a X otteniamo:

$$\frac{2X}{a^2} + \frac{2Y}{b^2}\frac{dY}{dX} = 0$$

Usando la (2) e la (3) e aggiungendo le informazioni relative al gradiente $\frac{dY}{dX}$ per ogni punto (x, y), è possibile risolvere per x_0, y_0 su un range di valori per a, b.

Questa tecnica utilizza le informazioni presenti nella mappa dei bordi dell'immagine originale e cerca di trovare il miglior ellisse dati tutti i pixel che rappresentano il contorno ed il loro gradiente. Potrebbe essere anche necessario ruotare l'ellisse se, ad esempio, i suoi assi non coincidessero con quelli dell'immagine. Quando questo metodo fu provato su immagini in real-time, anche con sfondo uniforme, la complessità computazionale cresceva velocemente con la dimensione del numero del range dei parametri per ogni pixel di bordo. Per limitare la crescita computazionale ad un livello adeguato, la mappa dei bordi fu limitata a contenere solamente i bordi più esterni del volto umano. Nel caso di immagini aventi uno sfondo uniforme, questa scelta è di facile ottenimento, ma, man mano che l'uniformità dello sfondo viene a mancare, il tempo richiesto per la computazione ricomincia a crescere. Per tali motivi venne scelta una strada alternativa alla trasformata di Hough.

Per risolvere i problemi incontrati con la trasformata di Hough fu utilizzato un approccio praticabile anche su immagini con sfondo non uniforme. Questo nuovo metodo, che ha alla base la natura ellittica del volto umano, utilizza informazioni dalla mappa dei bordi e si poggia su alcuni algoritmi esterni per etichettare tutti i segmenti contigui interni alla mappa stessa. Questi segmenti etichettati sono infine analizzati nel loro insieme per ricercare una possibile ellisse che si adatti alla loro forma Figura 2.5.

Figura 2.5: Mappa dei bordi (a), risultato dell'algoritmo (b).

I risultati ottenuti attraverso questo sistema mostrano che tale algoritmo è in grado di eliminare i contorni dritti dell'immagine.

La segmentazione del volto da un'immagine con uno sfondo uniforme non presenta particolari problematiche; infatti, la mappa dei bordi può restituire un'ottima approssimazione della regione contenente il volto. In questo caso, un semplice modo per determinare l'ellisse del volto sull'immagine consiste nel trovare i pixel più lontani sul contorno in entrambe le direzioni x e y. Definendo (x_i, y_i) la posizione di un pixel di bordo, allora:

$$x_0 = \sum_{i=0}^{N} \frac{x_i}{N}, y_0 = \sum_{i=0}^{N} \frac{y_i}{N} \tag{2.1}$$

dove N è il numero totale di pixel sul bordo nell'immagine e x_0, y_0 è il centro di massa del bordo dell'immagine. Ricordando l'obiettivo iniziale, ovvero ricercare la miglior ellisse possibile che approssimi i segmenti intorno al volto, la seguente equazione deve essere risolta:

$$\frac{(x_i - x_0)^2}{a^2} + \frac{(y_i - y_0)^2}{b^2} = 1 \tag{2.2}$$

dove (x_0, y_0) è il centro dell'ellisse, a rappresenta il semi-asse maggiore e b il semi-asse minone dell'ellisse. Il centro dell'ellisse è approssimabile dalla [2.1] se, come in questo caso, l'unico oggetto nell'immagine è il volto. I rimanenti parametri a e b possono essere ottenuti risolvendo il seguente sistema sovra determinato.

$$\begin{bmatrix} (x_1 - x_0)^2 & (y_1 - y_0)^2 \\ \vdots & \vdots \\ (x_n - x_0)^2 & (y_n - y_0)^2 \end{bmatrix} \begin{bmatrix} \frac{1}{a^2} \\ \vdots \\ \frac{1}{b^2} \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{2.3}$$

La precedente equazione matriciale risulta essere del tipo $AX = C$ e può essere quindi risolta come $X = (A^tA)^{-1}A^tC$

I risultati ottenuti tramite l'applicazione del precedente metodo, sono ottimi in presenza di un bordo del volto conforme ad una struttura ellittica. In presenza di elementi che modificano tale forma, come ad esempio capelli molto lunghi e voluminosi, i risultati ottenuti non sono sempre del tutto accurati.

Dalla Figura 2.4 e da [31] è possibile vedere come il riconoscimento dell'ellisse del volto e l'estrazione delle caratteristiche dello stesso siano due processi indipendenti (inizialmente).

Il sistema di riconoscimento del volto in [31] è stato sviluppato tramite l'utilizzo di alcuni algoritmi pre-esistenti sviluppati da Manjunath e Chellappa [5] ed in aggiunta utilizza anche una funzione di Gabor (Gabor wavelet decomposition). L'estrazione delle caratteristiche dall'immagine avviene dunque attraverso l'utilizzo di una funzione di Gabor, ovvero una funzione Gaussiana modulata attraverso una sinusoide complessa. La 2-D Funzione di Gabor può essere scritta come:

$$g(x, y : u_0, v_0) = exp(-[x^2/2\sigma_x^2 + y^2/2\sigma_y^2] + 2\pi i[u_0 x + v_0 y]) \tag{2.4}$$

dove σ_x rappresenta lo spazio delle ampiezze della gaussiana e (u_0, v_0) è la frequenza della sinusoide complessa.

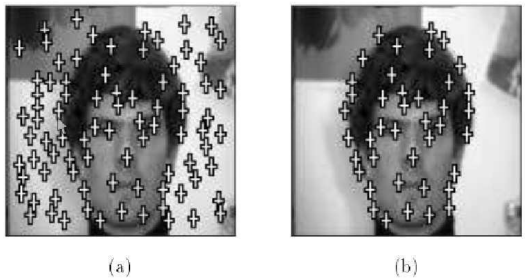


Figura 2.6: Caratteristiche dell'immagine (a) senza segmentazione dell'immagine, (b) con segmentazione.

Come mostrato in figura Figura 2.6 le caratteristiche estratte dall'immagine in input sono numerose e molte di esse sono esterne alla zona di interesse. L'utilizzo anche dell'algoritmo di segmentazione permette di confinare tali

caratteristiche solamente all'interno della regione di interesse. Inoltre, senza l'utilizzo dell'algoritmo di segmentazione, anche le caratteristiche esterne al volto, ovvero quelle dello sfondo, dovranno essere considerate. Questi punti di interesse non rappresenterebbero però informazioni pertinenti con l'obiettivo dell'algoritmo.

In Figura 2.7 sono rappresentati buoni risultati dell'algoritmo di segmentazione, mentre in Figura 2.8 sono riportati alcuni casi in cui l'algoritmo ha qualche difficoltà.

Figura 2.7: Buoni risultati dell'algoritmo di segmentazione. (a) immagine in input (b) Immagine estratta.

Figura 2.8: Scarsi risultati dell'algoritmo di segmentazione. (a) immagine in input (b) Immagine estratta.

Per quanto riguarda l'algoritmo di riconoscimento, esso è stato testato su un set di 30 immagini. Nel primo test l'algoritmo non è stato accompagnato dal sistema di segmentazione e si sono ottenuti buoni risultati solamente nel 50% circa dei casi. Secondariamente è stato accompagnato con l'algoritmo di segmentazione che ha migliorato notevolmente i risultati: oltre al 70%. La Figura 2.9 riporta tutti i passi dell'algoritmo. La Figura 2.9(a) mostra l'immagine in input, in Figura 2.9(b) vengono mostrati tutti i punti di interesse sull'intera immagine (senza l'utilizzo dell'algoritmo di segmentazione), in Figura 2.9(c) viene attivata la segmentazione ed infine nella Figura 2.9(d) vengono mostrati i risultati, ovvero l'immagine riconosciuta.

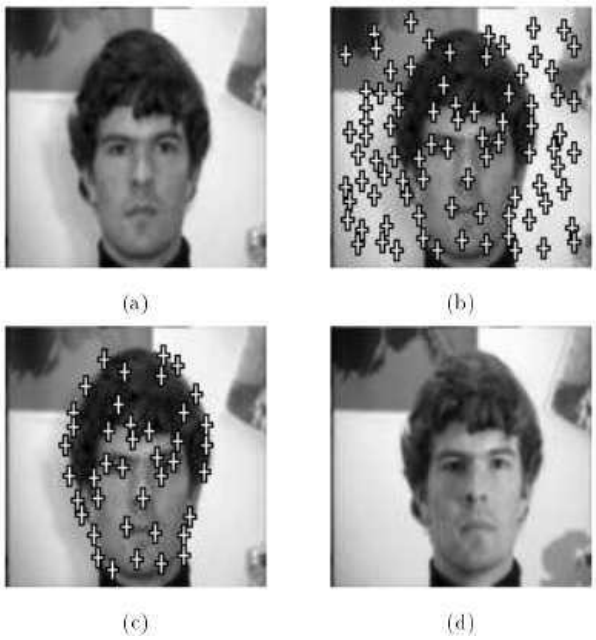


Figura 2.9: (a) Immagine in input, (b) punti di interesse senza segmentazione, (c) con segmentazione, (d) immagine riconosciuta.

2.2.3 Metodi basati su Algoritmi Genetici

In [9] Cheng Yuan Tang, Jeng Horng Change Hui Wen Jeng, proposero un algoritmo genetico per apprendere il modello del volto umano e con esso le sue caratteristiche principali quali occhi, naso e bocca. Nella formulazione presentata in [9], è stato scelto un genoma a due dimensioni per la codifica dei cromosomi dell'immagine del volto umano ed alcune proprietà, tra cui la posizione degli occhi e del loro centro, per valuare la funzione di fitness. Il riconoscitore a Massima-Verosimiglianza (Maximum Likelihood, ML) è stato utilizzato per l'individuazione di alcuni ellissi da proporre come possibili volti candidati, mentre l'algoritmo genetico è stato utilizzato per la verifica di quest'ultimi.

Algoritmi Genetici: Molto brevemente, gli algoritmi genetici [10] sono sistemi pensati per introdurre, nel campo tecnologico, alcuni processi evolutivi osservabili in natura. Una descrizione molto schematica del funzionamento di tali algoritmi è riportata di seguito

1. *Inizializzazione:* della popolazione di cromosomi;

2. *Valutazione:* di ogni cromosoma facente parte della popolazione;

3. *Evoluzione:* creazione di nuovi cromosomi attraverso l'unione di due cromosomi della popolazione selezionata. Vengono poi applicati altri sistemi per la ricombinazione del patrimonio genetico e mutazione;

4. *Eliminazione:* di alcuni membri della popolazione corrente per fare posto ai nuovi membri;

5. *Valutazione:* dei nuovi cromosomi e inserimento di questi nella popolazione;

6. *Verifica:* se il tempo è finito si ferma il processo e viene restituito il cromosoma migliore di tutta la popolazione, altrimenti si riparte dal punto 3.

Nell'algoritmo genetico proposto in [9], l'ML riconoscitore è stato utilizzato per l'identificazione di forme ellittiche e dai test fatti si è potuto vedere che, senza la presenza di alcuni oggetti di forma ellittica intorno al volto, è possibile ottenere buoni risultati. Verranno ora mostrate le varie fasi dell'algoritmo in questione.

Inizializzazione: Per prima cosa sono stati determinati e settati alcuni parametri fondamentali per l'algoritmo genetico. In questo caso si è scelto:

- Dimensione della popolazione: 50 individui;
- Numero di nuovi individui: 20;
- Mutazione: 0.00025/pixel nell'immagine 2-D
- Dimensione dell'immagine 60×60

Il genoma utilizzato è mostrato in Figura 2.10. Esso rappresenta la dimensione della porzione di immagine contenente il volto. La scelta è dovuta al database utilizzato per l'apprendimento all'interno del quale le immagini del volto hanno dimensioni di 60×60.

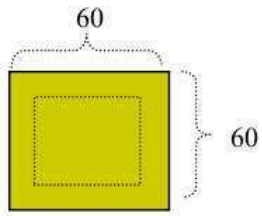


Figura 2.10: Genoma 2-D utilizzato in [9].

Nella fasi di inizializzazione sono stati creati 50 modelli di features del volto tra i quali è possibile citare: un modello vuoto, nel quale la popolazione è stata inizializzata con punti casuali ognuno avente l'1% di probabilità di essere scelto (Figura 2.11) e un secondo modello, nato da un insieme di mappe dei bordi (edge map) ottenibili attarverso l'applicazione, all'immagine del volto, dell'operatore Solber o del codice di SUSAN [32]. Figura 2.12.

Figura 2.11: Alcuni esempi di modelli vuoti.

Come detto precedentemente nel sistema proposto, alcune mappe dei bordi (4 nello specifico) sono state utilizzate per la creazione di un modello Inoltre ogni pixel appartenente a quest'ultimo è stato confrontato con un valore di soglia; se il pixel risultava essere superiore o uguale a tale soglia veniva rappresentato in bianco, altrimenti in nero.

Figura 2.12: A sinistra: regione del volto nell'immagine in input. A destra: mappa dei bordi tramite applicazione di [32].

Valutazione: Nel sistema presentato, la funzione di fitness è stata definita come il rapporto tra la posizione degli occhi e l'attesa posizione degli stessi. è stato assunto, inoltre, che il vero centro degli occhi fosse il pixel più scuro interno alla regione della pupilla. Nella Figura 2.13, C_l rappresenta il vero centro dell'occhio sinistro e C_r quello dell'occhio destro mentre $\tilde{\mathrm{C}}_l$ rappresenta posizione attesa del centro dell'occhio sinistro e $\tilde{\mathrm{C}}_r$ quella dell'occhio destro. d_l rappresenta invece la distanza Euclidea tra C_l e $\tilde{\mathrm{C}}_l$, mentre d_r quella tra C_r e $\tilde{\mathrm{C}}_r$.

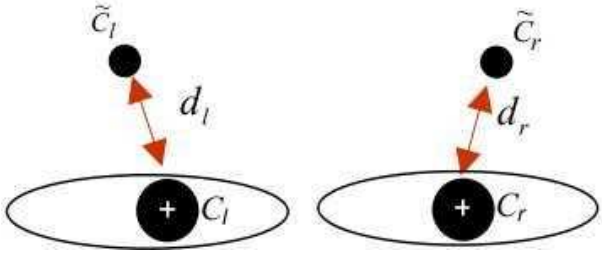


Figura 2.13: Posizione degli occhi stimata e reale.

La reale posizione degli occhi è stata localizzata tramite un'attenta analisi di un database di volti, dalla quale è stato possibile estrappolare un insieme di valori che mettono in relazione la posizione degli occhi con il volto. Una volta acquisita tale regione dell'immagine, il centro dell'occhio è stato localizzanto cercando il pixel più scuro. Successivamente è stata determinata la posizione stimata degli occhi, definita come il centro del contorno dell'occhio risultante dalla mappa dei bordi

Selezione dei genitori: lo scopo della selezione dei genitori in un algoritmo genetico è quello di migliorare le possibilità di riproduzione e quindi di creazione dei nuovi individui nella popolazione. Nel sistema proposto si è scelto di utilizzare un metodo di selezione abbastanza comune: la Roulette Wheel [10] la quale può essere schematizzata come segue:

1. viene calcolata la *total fitness* ovvero la somma della funzione di fitness di tutti i membri della popolazione;

2. viene generato casualmente un numero n compreso tra 0 ed il valore della total fitness;

3. viene restituito il primo individuo della popolazione il cui valore di fitness, aggiunto al valore di fitness dei membri della popolaione precedente, è maggiore o uguale a n.

La prima tabella in Figura 2.14 mostra il valore di fitness per 9 cromosomi e il valore di fitness totale. La seconda tabella mostra invece i cromosomi che sono scelti tramite la Roulette Wheel utilizzando i valori di fitnes su un esempio di 6 numeri generati casualmente.

chromosome	1	2	3	4	5	6	7	8	9
fitness	2	6	3	5	2	1	8	6	7
Running total	2	8	11	16	18	19	27	33	40

Random number	20	35	16	7	10	25
Chromosome chosen	7	9	4	2	3	7

Figura 2.14: Esempio della selezione dei genitori tramite Roulette Wheel.

Crossover e Mutazione: In natura il sistema di crossover avviene quando due genitori si scambiano parti dei loro cromosomi. Negli algoritmi genetici questo sistema è utilizzato per ricombinare il patrimonio genetico dei due genitori al fine della creazione di un nuovo individuo. Nel sistema qui trattato è stato utilizzato un crossover one-point. Per quanto riguarda invece il processo di mutazione, esso è responsabile del fatto che i cromosomi di un nuovo individuo sono diversi dai cromosomi dei suoi antenati.

Rimpiazzo: A causa della limitatezza della popolazione utilizzata, il processo di rimpiazzamento dei vecchi individui in favore dei nuovi deve essere controllato ed è utilizzato per verificare quale individuo deve essere tolto dalla popolazione per lasciare il posto a quello nuovo.

Il sistema descritto precedentemente è stato testato su 100 immagini di volti umani usate come mezzo di apprendimento. Il modello dei bordi del volto è stato utilizzato come modello iniziale e la posizione degli occhi è stata utilizzata come funzione di valutazione. Nella parte sinistra della Figura 2.15 sono rappresentati i risultati dopo 1000 epoche di apprendimento da parte dell'algoritmo genetico, mentre nella parte destra sono rappresentati i risultati ottenuti dopo 1000 epoche di apprendimento utilizzando il modello vuoto come modello iniziale.

Figura 2.15: Risultati dopo 1000 epoche di apprendimento. A destra è stato utilizzato il modello dei bordi del volto, a sinistra il modello vuoto.

Nel riconoscimento del volto l'ML riconoscitore è stato usato per ottenere alcuni volti candidati e successivamente è stato utilizzato il template creato mediante l'algoritmo genetico per la verificare le veridicità di tali candidati. Le Figura 2.16 e 2.17 e mostrano i risultati di tale operazione.

Figura 2.16: Immagine di input per testare il sistema.

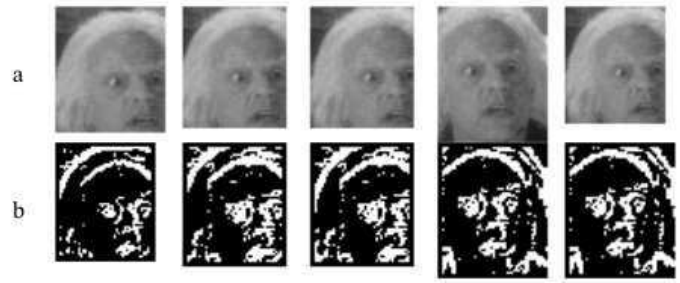


Figura 2.17: (a) Candidati che sono stati riconosciuti come volti (b) mappe dei bordi di tali candidati.

2.3 Eye Tracking

In passato [38] l'unica modalità tramite cui si poteva condurre una precisa analisi sui movimenti degli occhi, era quella di utilizzare alcune tecniche invasive, come ad esempio: l'elettro-oculografia (tecnica che prevede l'uso di alcuni elettrodi posizionati sul capo della persona per misurare la diferenza di potenziale generata dal movimento degli occhi), l'uso di particolari lenti a contatto che mettevano in risalto l'occhio, o tramite sistemi di specchi e telecamere montate sulla testa della persona.

I primi esperimenti condotti con tali tecniche invasive degni di nota, sono stati compiuti da *Kaufman et al.* [4], i quali applicarono degli elettrodi sui lati della testa per misurare la differenza di potenziale che si registra con il movimento deglio occhi; *K. Y. Kim, S. Y. Lee, e H. C. Kim* [22] usarono una tecnica del tutto diversa dalla precedente. Essi posero sull'occhio del soggetto una particolare lente a contatto contenente un magnete, la cui posizione veniva registrata da due sensori di rilevamento del campo magnetico generato dalla calamita. Nel più recente passato sono stati raggiunti anche ottimi risultati dai sistemi proposti da *Li et al.* [12], *Lijima et al.* [1], *Asai et al.* [17], in cui vengono usate una o più telecamere disposte sulla testa dell'utente che, tramite un sistema di specchi e occhiali particolare, riescono a catturare il movimento dell'occhio come se le telecamere fossero disposte frontalmente al viso.

Con l'introduzione negli ultimi 10-15 anni dei primi sensori CCD, per l'acquisizione di immagini in formato digitale, sono diminuiti notevolmente sia i costi delle telecamere, sia le loro dimensioni, spingendo fortemente la ricerca ad abbandonare tutti i precedenti metodi invasivi. Il tutto è stato inolte affiancato da una veloce crescita delle prestazioni dei computer e delle qualità video delle telecamere. Grazie a tutte queste innovazioni tec-

nologiche dell'ultimo decennio, oggi è possibile mantenere una risoluzione dell'immagine elevata pur stando a debita distanza dall'occhio umano.

Nei prossimi paragrafi verranno presentate alcune delle varie soluzioni che offre la letteratura in merito alle precedenti problematiche, seguendo la falsa riga di quanto già detto nel paragrafo precedente sulla Face Detection.

2.3.1 Metodi generici, senza l'uso di infrarossi

Michael Chau e Margrit Betke [26] proposero un sistema non intrusivo per utenti disabili, per offrire la possibilità di interagire con numerose applicazioni attraverso il movimento della testa e degli occhi in particolare. Tramite l'utilizzo di un robusto algoritmo utilizzato da *Grauman et al.* [18], [21]. Il sistema prevede un primo step di inizializzazione nel quale si conduce un'analisi di un battito di palpebere volontario da parte dell'utente. Tale battito verrà poi utilizzato dall'algoritmo per creare un modello di occhio in modo da effettuare il tracking degli stessi. L'algoritmo utilizzato può essere così rappresentato Figura 2.18.

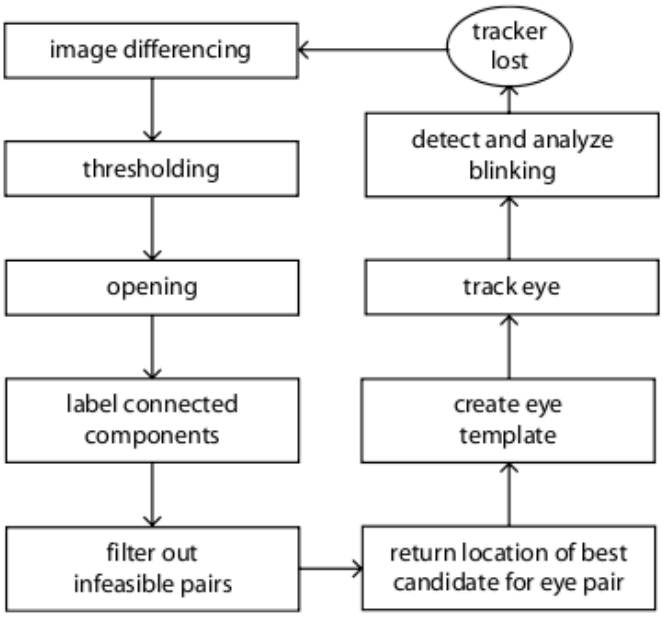


Figura 2.18: Schema dei principali step dell'algoritmo.

La fase di inizializzazione serve all'algoritmo per il rilevamento della posizione delgli occhi all'interno dell'immagine. Per questo motivo viene calcolata la differenza tra ogni frame e il suo precedente, ottenendo come risultato un'immagine binaria che mostra le regioni in cui si è verificato un movimento. Tale sistema viene poi accompagnato da altri algoritmi per eliminare il rumore, nell'immagine ottenuta, dovuto al naturale tremolio del volto dell'utente o da piccoli cambiamenti di luce. Nell'ambito in cui è stato sviluppato questo sistema, ovvero mirato per persone disabili o semi-paralizzate, il risultato dell'analisi sul movimento produce ottimi risultati visibili nellaFigura 2.19.

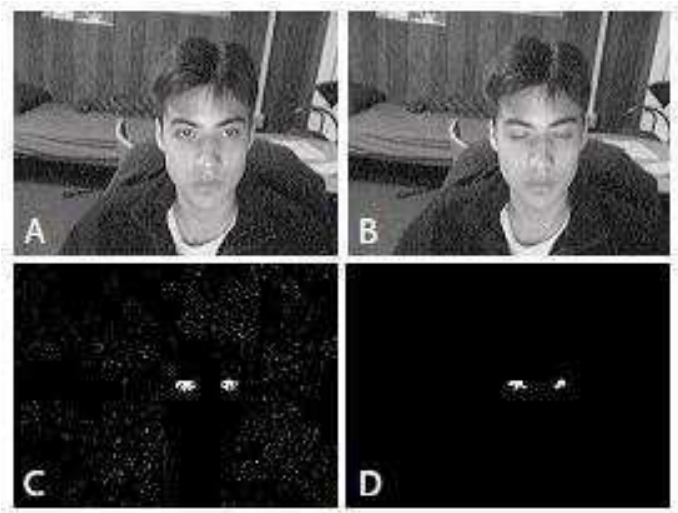


Figura 2.19: Differenza tra due frame consecutivi per il rilevamento degli occhi.

L'algoritmo proposto utilizza anche un particolare sistema per scartare eventuali falsi positivi, il quale si basa sostanzialmente sull'analisi di alcuni parametri, quali la lunghezza e l'altezza dei componenti analizzati e la distanza orizzontale e verticale tra i centri dei componenti. Se il passo precedente produce come risultato due componenti che rispettano tutti i controlli fatti, allora con buona approssimazione gli occhi sono stati localizzati correttamente e la regione contenente il componente più grosso viene presa per la crazione di un modello di occhio aperto.

Per effettuare l'eye Tracking Michael Chau utilizza un algoritmo [18] basato sul calcolo di un particolare coefficiente:

$$\frac{\sigma_{x,y}[f(x,y) - f^1_{u,v}][t(x-u, y-v) - t^1]}{\sqrt{\sigma_{x,y}[f(x,y) - f^1_{u,v}]^2 \sigma_{x,y}[t(x-u, y-v) - t^1]^2}} \tag{2.5}$$

dove $f(x,y)$ rappresenta la luminosità nel frame video nel punto (x,y), $f^1_{u,v}$ rappresenta il valore medio del frame video nella regione interessata dall'analisi, mentre $t(x,y)$ rappresenta la luminosità del modello nel punto (x,y) e t^1 è il valore medio del modello. Il valore ottenuto sarà un valore compreso tra -1 e 1 e rappresenta la somiglianza tra il modello e i punti trovati nella regione di ricerca. Risultati prossimi allo zero saranno quindi indicatori di un basso livello di somiglianza tra modello e candidato, mentre livelli prossimi a 1, corrisponderanno ad un elevato grado di somiglianza. I risultati ottenuti sono visibili in Figura 2.20.

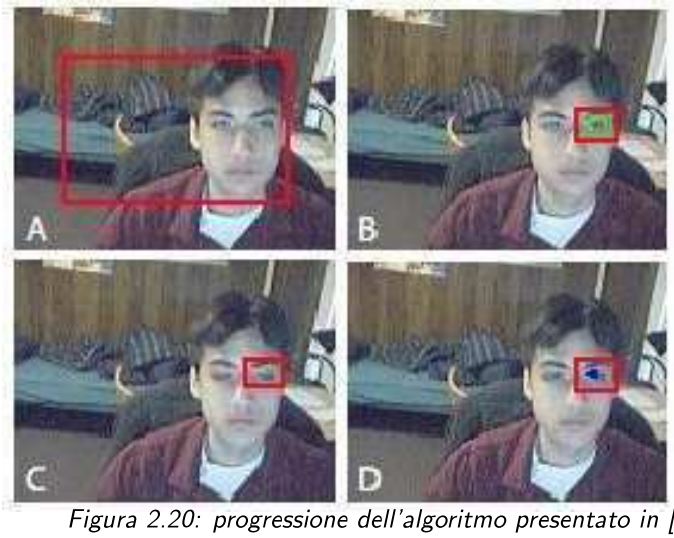


Figura 2.20: progressione dell'algoritmo presentato in [26]

- Figura 2.20(a): Sistema durante la fase di analisi del movimento. Il rettangolo rosso rappresenta la regione che è considerata durante lo step di inizializzazione;
- Figura 2.20(b): Una volta localizzato l'occhio il sistema entra in questo stato. Il rettangolo verde rappresenta il componente utilizzato per la creazione del modello dell'occhio, mentre il rettangolo rosso rappresenta la nuova area di ricerca;

- Figura 2.20(c): Battito di palpebre dell'utente al frame f;
- Figura 2.20(c): Al frame $f + 1$ il sistema rappresenta il recedente battito delle palbere posizionando un pallino giallo sull'occhio.

2.3.2 Metodi basati sull'uso degli infrarossi

Molti sistemi di eye-tracking offerti dalla letteratura utilizzano, con modalità a volte diverse, la luce infrarossa. Lavorare nello spettro dell'infrarosso effettivamente rapresenta un ottimo espediente per condurre un'attenta analisi sui movimenti degli occhi, ed in particolare della pupilla. Tale sistema risulta essere molto usato proprio a causa della particolare struttura dell'occhio umano. Il Raggio IR infatti attraversa la pupilla e successivamente, grazie all'azione del cristallino, viene focalizzato in un punto della retina dell'occhio. Quest'ultima è un tessuto particolarmente riflettente il quale normalmente riflette (sullo stesso asse con cui arriva) gran parte della luce ricevuta. In questo modo una telecamera sensibile agli infrarossi può vedere sulla pupilla lo stesso punto luminoso impresso sulla retina, che noi normalmente non vediamo. Il risultato ottenuto è visibile in figura Figura 2.21

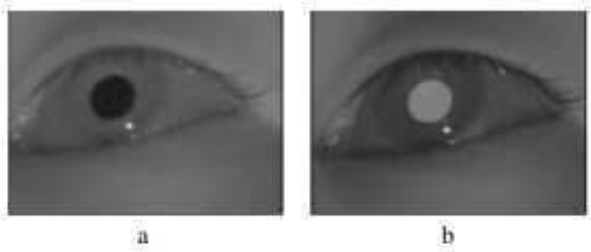


Figura 2.21: (a) Illuminazione IR fuori asse della telecamera (dark pupil). (b) illuminazione in asse (bright pupil)

Nella Figura 2.21(b) la sorgente luminosa è in asse con la telecamera e la pupilla si distingue da tutto il resto come l'area più luminosa. Tale effetto viene chiamato *effetto bright pupil.* Nel caso in cui la sorgente luminosa fosse fuori asse rispetto alla telecamera, la luce sarebbe riflessa in una direzione non riconosciuta dalla telecamera producendo quindi l'effetto contrario, ovvero *dark pupil* Figura 2.21(a). [36]

Per isolare in modo efficace la posizione della pupilla, pur rimanendo a debita distanza, *Morimoto et al.* [7] usano una telecamera schermata

alla luce visibile, posta di fronte al computer e due led IR che si illuminano in alternanza. I due led sono posti a distanze differenti dalla telecamera in modo da sfruttare entrambi gli effetti rappresentati in Figura 2.21. In questo modo la pupilla si comporta come un disco che lampeggia in modo sincrono con i due led IR. La frequenza con cui i due led IR si alternano è stata scelta uguale al frame rate della videocamera. Con tale sistema diventa molto semplice identificare la pupilla tramite sottrazione di due frame successivi. Inoltre essendo la telecamera posizionata ad una certa distanza dall'utente, ha un campo visivo molto elevato permettendo di inquadrare tutto il volto della persona.

Yoo e Chung [11] hanno recentemente proposto un sistema composto da 5 led IR e due telecamere CCD, una centrata sull'occhio con un campo visivo risretto e l'altra con copertura maggiore, entrambe con due gradi di libertà per seguire i movimenti dell'utente (Figura 2.22). Quattro led sono stati messi inoltre sui rispettivi angoli dello schermo del computer per creare la riflessione sulla superficie della cornea (effetto *dark pupil*); il quinto è utilizzato per creare *l'effetto brigh pupil* e quindi si trova in corrispondenza dell'asse della videocamera usata per zoomare sull'occhio. Anche in questo caso, come in [7], entrambi gli effetti riflessivi della pupilla sono stati utilizzati per la localizzazione e l'estrazione della pupilla stessa.

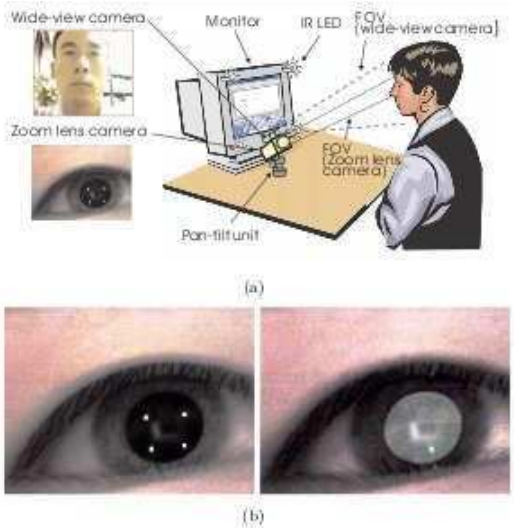


Figura 2.22: (a) Descrizione sistema proposto da [11]. (b) Risultati sull'occhio dell'utente

2.3.3 Metodi basati sull'uso di reti neurali

Una tecnica molto usata che conduce a ottimi risultati, ma con un carico computazionale leggermente più elevato rispetto alle precedenti, è quella che ricorre all'uso di reti neurali [2], [29]. La rete neurale, molto similmente a quello che avveniva in [9] per la Face Detection, viene qui utilizzata per l'apprendimento del modello dell'occhio passatogli in ingresso. Alla rete neurale infatti viene passato un insieme di immagini dell'occhio in differenti posizioni e condizioni di illuminazione, dal quale la rete dovrà apprendere i vari pattern per il riconoscimento. Questa soluzione, molto diversa dalle precedenti, ha portato a grandi risulati nello studio del centro dell'iride e anche nello studio della direzione dello sguardo.

Capitolo 3

Face Detection

"Spooner: Le emozioni non sembrano essere una simulazione molto utile per un robot...Non vorrei mai che il mio tostapane o l'aspirapolvere fossero così emotivi..."

Io Robot

In questo capitolo verrà descritto nel dettaglio l'algoritmo proposto, ossia l'algoritmo progettato per la localizzazione del volto umano all'interno di un video. Successivamente verranno analizzate le caratteristiche principali, ad esempio, le sue dimensioni. La sorgente di informazione a disposizione è un semplice filmato 640×480, registrato dalla webcam integrata nel portatile, avente un frame rate di 25 frame al secondo circa.

Tra tutte le possibili alternative offerte dalla letteratura, si è scelta una particolare tecnica implementata nelle librerie OpenCV, che permette la localizzazione di generici oggetti in un'immagine o, come in questo caso, in un video; l'Haar classifier. La scelta è stata guidata principalmente dalla facilità di implementazione di tale algoritmo e dalle sue alte prestazioni. L'algoritmo proposto, infatti rappresenta un ottimo sistema di riconoscimento del volto in pose differenti, anche in condizioni di luminosità molto variabili e risulta ottimale per sistemi in real time.

3.1 Algoritmo proposto

La computer vision è una vasta area in continua evoluzione. Per tale motivo, alcune sezioni delle openCV, proponenti algoritmi per l'implementazione di specifiche funzioni, rischiano, dopo poco tempo, di essere considerate obsolete. L'algoritmo di *face detection*, invece, risulta facilmente adattabile al riconoscimento di vari oggetti (volti, macchine, lettere, figure geometriche, etc.) all'interno di immagini o video, attraverso la creazione di nuovi riconoscitori.

Le librerie openCV implementano una versione dell'algoritmo di face detection sviluppato inizialmente da *Paul Viola e Michael Jones* e per questo, comunemente riconosciuto come *Viola-Jones Detector* [28]. Successivamente tale versione venne estesa da Rainer Lienhart e Joched Maydt, dai quali venne chiamata Haar Classifier. Tale nominativo deriva dalle *haar feature*: un insieme di caratteristiche derivate dalle trasformate di Haar. Il presente algoritmo si basa principalmente sulle operazioni di somma e sottrazione di regioni di immagini, per passare poi alla computazione dei risultati; esso è contraddistinto da tre caratteristiche principali:

- **Immagine integrale:** nuova rappresentazione di un'immagine che permette alte prestazioni nell'analisi delle feature le quali possono essere analizzate ad ogni scala in un tempo costante. Questa rappresentazione è ottenibile con semplici operazioni sui pixel dell'immagine originale;
- **Classificatore:** il classificatore è costruito attraverso la selezione di alcune feature estratte dall'algoritmo AdaBoost [14];
- **Classificatore complesso:** viene utilizzato un particolare metodo per la combinazione in cascata di più classificatori. Tale struttura incrementa fortemente le prestazioni del riconoscimento focalizzando l'attenzione dell'algoritmo solamente nelle regioni dell'immagine più promettente.

3.1.1 Features

L'algoritmo di *face detection*, proposto da Viola-Jones, utilizza un'insieme di feature dette haar-like-features derivate dalle trasformate di haar. Questo approccio nasce dagli studi di *Papageorgiou er al.* [8] per l'analisi di immagini attraverso un numero ristretto di features. Nel caso dell'algoritmo per la localizzazione dei volti il classificatore viene definito come un albero decisionale con almeno due foglie, che utilizza le feature per determinare se una

zona dell'immagine analizzata è compatibile con qualche immagine da esso riconosciuta in fase di apprendimento. Le features utilizzate da un classificatore sono definite dalla loro forma, dalla posizione all'interno della regione di interesse e dal fattore di scala. Ad esempio, nel caso 2c in Figura 3.1, la risposta è calcolata come differenza tra la somma dei valori di intensità nei pixel di immagine coperti da tutta la caratteristica e la somma dei pixel coperti dalla zona nera moltiplicati per tre, per compensare le differenze di formato delle zone.

Le somme dei valori di intensità dei pixel coperti dalle regioni rettangolari, possono essere calcolate in maniera molto rapida se si utilizza una rappresentazione particolare per l'immagine di input: l'immagine integrale.

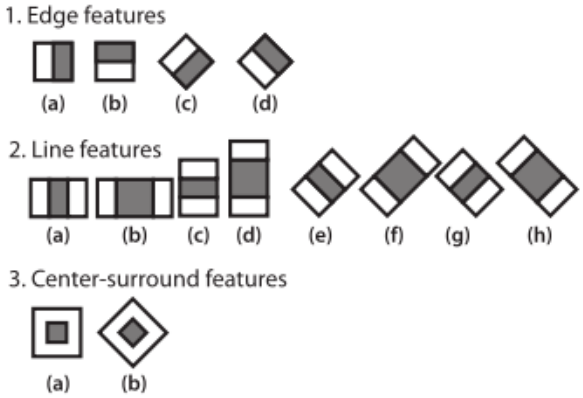


Figura 3.1: Features haar utilizzabili.

Nello specifico, nell'algoritmo presentato da Viola e Jones sono state utilizzate tre differenti tipologie di feature (Figura 3.2):

- *Two-rectangular-feature*: il suo valore corrisponde alla differenza tra la somma dei pixel all'interno delle due regioni rettangolari;
- *Three-rectangular-feature*: il suo valore corrisponde alla somma dei due rettangoli esterni meno il rettangolo centrale;
- *Four-rectangular-feature*: il suo valore corrisponde alla differenza tra le coppie di rettangoli posti lungo la diagonale.

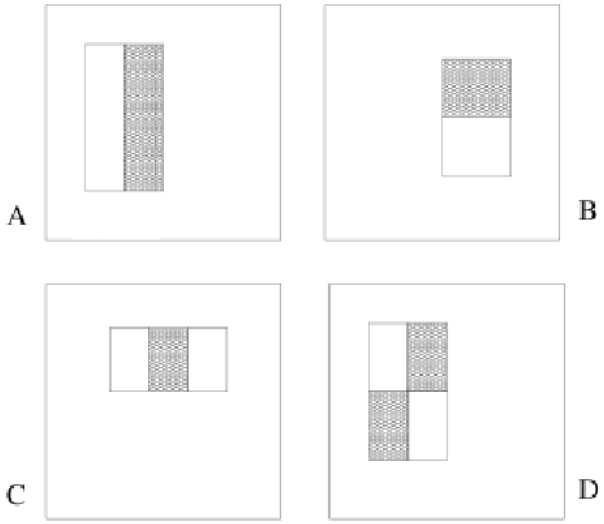


Figura 3.2: Esempi di rectangular-features mostrate in una finestra chiusa. (a) e (b) mostrano una two-rectangular features, (c) mostra una tree-rectangular feature e (d) una four-rectangular feature.

3.1.2 Immagine Integrale

Le precedenti rectangular-feature possono essere analizzate molto rapidamente usando una rappresentazione dell'immagine chiamata *immagine integrale*, la quale alla posizione (x, y), rappresenta la somma dei pixel contenuti al di sopra e alla sinistra del punto (x, y) incluso (Figura 3.3).
è quindi possibile esprimere l'immagine integrale come

$$ii(x, y) = \sum_{x' \leq x, y' \leq y} i(x', y')$$

dove $ii(x, y)$ è l'immagine integrale e $i(x, y)$ rappresenta l'immagine originale. L'immagine integrale può essere calcolata in un unico passo a partire dall'immagine originale utilizzando le due seguenti formule ricorsive:

$$s(x, y) = s(x, y - 1) + i(x, y) \tag{3.1}$$

$$ii(x, y) = ii(x - 1, y) + s(x, y) \tag{3.2}$$

dove $s(x, y)$ è la somma comulativa delle righe, $s(x - 1) = 0$ e $ii(-1, y) = 0$.

Utilizzando l'immagine integrale, ogni somma di rettangoli può essere calcolata attraverso quattro riferimenti ad un array (Figura 3.4).

Prendendo come riferimento la Figura 3.4, la somma dei pixel nel rettangolo D può essere calcolato con semplici operazioni sulle coordinate dei

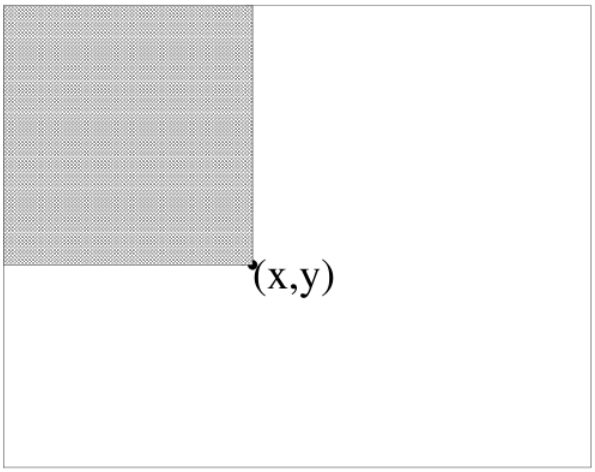


Figura 3.3: Il valore dell'immagine integrale al punto (x, y) è la somma di tutti i pixel al di sopra e alla sinistra del punto (x, y) incluso.

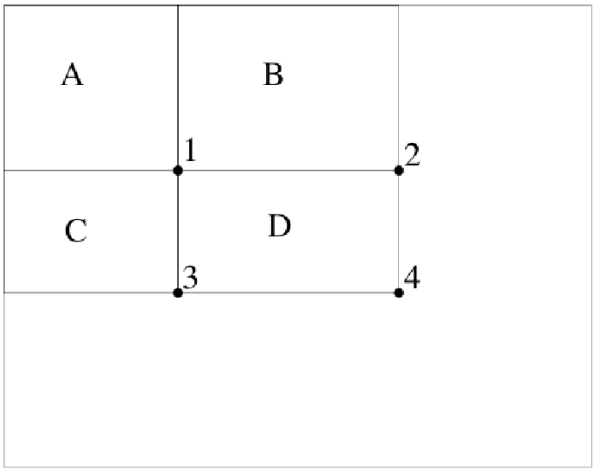


Figura 3.4: Esempio di calcolo attraverso l'immagine integrale.

punti 1,2,3,4. Il valore dell'immagine integrale nel punto 1, è la somma dei pixel presenti nel rettangolo A, mentre, nel punto 2, il valore è dato dalla somma dei rettangoli A e B, così come nel punto 3, è calcolabile come $A+C$ ed infine, nel punto 4, è calcolabile come $A + B + C + D$. Viceversa, per il valori dei singoli rettangoli, si ottiene che:

- $A = 1$
- $B = 2 - 1$
- $C = 3 - 1$
- $D = 4 - A - B - C$

Il valore D può essere dunque calcolato come $D = 4 + 1 - (2 + 3)$.

Per quanto riguarda le *feature-haar-like*, queste possono essere calcolate attraverso una semplice operazione di somma e sottrazione, ricorrendo alla rappresentazione descritta dalle equazioni ricorsive 3.1 e 3.2; esse, infatti, rappresentano differenze e somme di sotto-finestre dell'immagine originale.

Un'ulteriore caratteristica che rende molto vantaggioso l'utilizzo dell'immagine integrale, è stata dimostrata nelle ricerche di *Simard et al.* in [37], secondo cui l'immagine integrale è l'integrale doppio dell'immagine originale. Tuttavia, per la grande quentità di feature (intorno alle 40.000) associate ad ogni sottofinestra dell'immagine originale 24x24, l'ipotesi di utilizzare alcuni sistemi di riconoscimento proposti negli anni e trovati nella letteratura è da considerarsi proibitiva in quanto, anche una rapida analisi delle feature derivanti delle varie sotto-finestre dell'immagine (in numero superiore al numero dei pixel dell'immagine stessa) porterebbe a tempi computazionali proibitivi, soprattutto per applicazioni orientate al real-time. L'ipotesi fatta risulta dunque la seguente: solo un numero ridotto di caratteristiche possono essere combinate nella costruzione di un classificatore affinchè questo sia efficiente sia in termini di riconoscimento del volto, che in termini di prestazioni.

Nel sistema proposto da Viola-Jones è stata utilizzata una variante di un algoritmo già presente in letteratura, l'*AdaBoost*, usato sia nella selezione delle feature da utilizzare, sia nell'addestramento dei vari classificatori. Nella sua forma originale, l'AdaBoost viene utilizzato per migliorare le prestazioni di un singolo algoritmo di apprendimento. Questo viene reso possibile attraverso la combinazione di funzioni di classificazione definite deboli (ossia che utilizzano un singolo algoritmo di apprendimento), al fine di formare un classificatore definito forte. L'utilizzo dell'AdaBoost da parte di Viola-Jones,

può essere anche inteso però come un semplice processo greedy di selezione. Se si considera, infatti, il generico problema del migliorare (Boost) l'apprendimento, l'obiettivo posto è quello di associare grandi pesi alle funzioni di classificazione che portano ad un buon riconoscimento e bassi pesi alle altre. Tramite questo processo, si va a selezionare un ristretto insieme di funzioni di classificazione. Nella variante dell'AdaBoost proposto da Viola-Jones, l'algoritmo di apprendimento debole, e quindi il classificatore debole, viene utilizzato per la selezione di una singola feature rettangolare che meglio separa gli esempi positivi da quelli negativi. Per ogni feature, l'algoritmo di apprendimento debole determina la migliore funzione di classificazione che funge da soglia, in modo tale che un numero minimo di esempi siano classificati in maniera errata. Per tali motivi è possibile definire l'algoritmo AdaBoost un eccellente sistema che permette la selezione di un ristretto insieme di feature definite ottime.

Un Classificatore debole $h_j(x)$, consiste in una feature f_j, una soglia θ_j e una parità che indica il verso del segno nella diseguaglianza:

$$h_j(x) = \begin{cases} 1, & \text{se } p_j f_j(x) < p_j \theta_j \\ 0, & \text{altrimenti} \end{cases} \tag{3.3}$$

In questo caso x indica una sotto-finestra di un'immagine.

Nella realtà, ovviamente, non è possibile ricorrere ad una singola feature per ottenere un classificatore che commetta un errore tale da poter essere etichettato come trascurabile. Le feature selezionate inizialmente nel processo, hanno quindi percentuali di errore che variano dallo 0.1 allo 0.3, mentre quelle selezionate negli ultimi step, quando l'apprendimento diviene più complesso, hanno percentuali d'errore più elevate (nell'ordine di $0.4 - 0.5$). Di seguito viene mostrato, in pseudo codice, l'algoritmo di apprendimento utilizzato.

3.1.3 Algoritmo di apprendimento della funzione di classificazione

Dato un'insieme di feature ed un training set di esempi positivi e negativi, si possono utilizzare moltissimi approcci per la costruzione di una funzione che li classifichi correttamente. *Sung e Poggio* in [19], proposero, ad esempio, un modello di classificazione composto da un'intreccio di diverse gaussiane, mentre *H. Rowley e Kanade* in [16] utilizzarono un piccolo set di feature ed una rete neurale per la loro classificazione.

- Date le immagini di esempio $(x_1, y_1), ..., (x_n, y_n)$ dove $y_i = 0, 1$ per esempi negativi e positivi rispettivamente;
- Inizializzazione dei pesi $w_{1,i} = \frac{1}{2m}, \frac{1}{2l}$ per $y_i = 0, 1$ rispettivamente, dove m e l rappresentano il numero degli esempi positivi e negativi;
- **for**(t=1,...,T)
  1. Normalizzazione dei pesi
     $w_{t,i} \leftarrow \frac{w_{t,i}}{\sum_{j=1}^{n} w_{t,j}}$
     in modo tale che w_t possa essere interpretato come una distribuzione di probabilità
  2. Per ogni feature j, si addestra il classificatore h_j il quale è obbligato, per costruzione, ad utilizzare una sola feature. L'errore viene valutato rispetto a w_t
     $\varepsilon_j = \sum_i w_i \mid h_j(x_i) - y_i \mid$
  3. Selezione del classificatore h_t avente la più piccola percentuale di errore ε_t
  4. Aggiornamento dei pesi
     $w_{t+1,i} = w_{t,i}\beta_t^{1-e_i}$
     dove $e_i = 0$ se l'esempio x_i è classificato correttamente, altrimenti $e_i = 1$ e $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$
- Il classificatore finale *forte* risulta dunque essere:

$$h(x) = \begin{cases} 1, & \text{se } \sum_{t=1}^{T} \alpha_t h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} \alpha_t \\ 0, & \text{altrimenti} \end{cases} \tag{3.4}$$

dove $\alpha_t = \log \frac{1}{\beta_t}$

Si ipotizza che le T ipotesi siano strutturate attraverso l'uso di una singola feature; L'ipotesi finale risulta dunque caratterizzata da una combinazione lineare di pesi delle ipotesi T, dove i pesi selezionati sono inversamente proporzionali al rate di errore.

In Figura 3.5 è riportato un esempio circa i risultati dei primi nove cicli dell'algoritmo sopra presentato. La prima caratteristica scelta è quella più presente in tutte le immagini; essa in particolare mette in evidenza la differenza di tonalità tra occhi, fronte e guance.

Figura 3.5: Caratteristiche rilevate durante i primi nove cicli dell'algoritmo di AdaBoost.

La parte degli occhi, infatti, risulta più scura della parte che li circonda, e ciò è ricorrente in tutte le immagini. Alla seconda interazione, l'algoritmo rileva la differenza di tonalità tra gli occhi e il naso, identificato come una una zona più chiara in tutte le immagini. Proseguendo con le iterazioni, le caratteristiche evidenzieranno le differenze tra la bocca e le zone comprendenti il naso e il mento, per poi soffermarsi sui contorni del viso, del collo e delle sopraciglia, fino a discriminare dettagli sempre più piccoli.

I primi test fatti su un classificatore avente le caratteristiche mostrate nell'equazione 3.4, hanno mostrato che attraverso l'utilizzo di 200 feature è possibile realizzare un classificatore con ottimi risultati (Figura 3.6).

3.1.4 La combinazione in cascata dei classificatori

In questo paragrafo verrà presentato l'algoritmo utilizzato da Viola-Jones per la costruzione della cosiddetta *cascata di classificatori*, la quale ha il compito di incrementare le prestazioni del riconoscimento del volto all'interno di un'immagine, ma, allo stesso tempo, deve anche cercare di ridurre al minimo i tempi computazionali, sempre di primaria importanza per le applicazioni orientate al real-time. Il concetto che sta alla base dell'implementazione di una cascata di classificatori è semplice: Per raggiungere gli obiettivi prestazionali e computazionali fissati, infatti, servono piccoli, veloci ed efficaci classificatori, i quali hanno il compito di reiettare quanto prima la maggior parte delle sotto-finestre negative, in modo da focalizzare l'attenzione su quelle positive. Tutti questi classificatori devono inoltre essere opportunamente accoppiati per ottenere performance di alto livello.

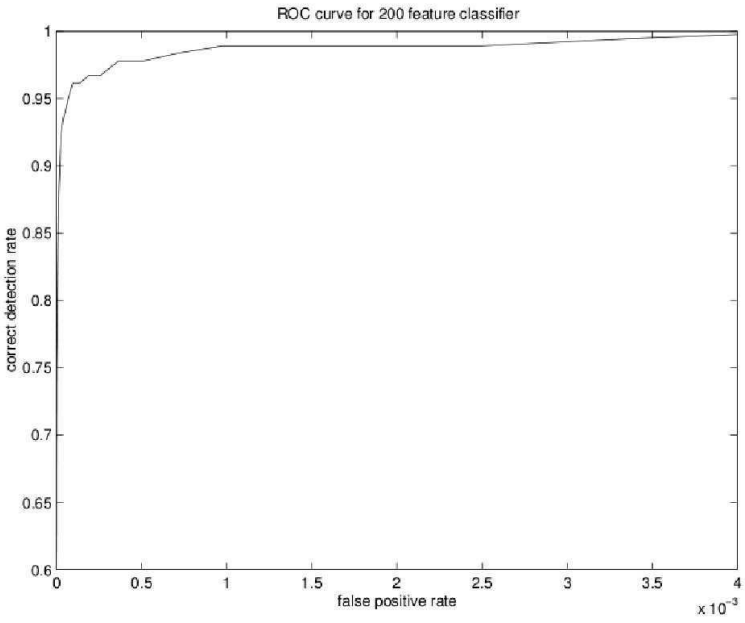


Figura 3.6: Risultati ottenuti tramite l'utilizzo di un classificatore di tipo 3.4.

Per tale motivo, si è scelto di utilizzare classificatori molto semplici, i quali hanno il compito di scremare l'enorme insieme di sotto-finestre dell'immagine analizzata e successivamente, di utilizzare classificatori più complessi per raggiungere un basso livello di falsi positivi. Per l'apprendimento dei vari classificatori ai vari livelli della cascata, viene ovviamente utilizzato l'algoritmo di AdaBoost precedentemente illustrato.

Il funzionamento della cascata così ottenuta è abbastanza semplice; per tutte le sottofinestre dell'immagine vengono applicati una serie di classificatori: per ognuna di queste infatti, si passa all'analisi di un secondo classificatore solamente in caso di risposta positiva da parte del classificatore precedente. Tale meccanismo a cascata prosegue poi fintantochè il risultato dei test associati ai vari classificatori incrementali non ha un esito totalmente positivo, mentre si passa all'eliminazione della sotto-finestra corrente alla prima occorrenza di un risultato negativo da parte di un classificatore (Figura 3.7).

Aumentando il numero di strati nella cascata, ovvero aumentando di fatto il numero delle verifiche su ogni sottofinestra, aumenta anche l'attendibilità del risultato ottenuto. In questo caso, però, si va ad aumentare anche il carico computazionale dell'applicazione. Risulta dunque necessario travare un giusto bilanciamento tra le richieste di bassi tempi di elabora-

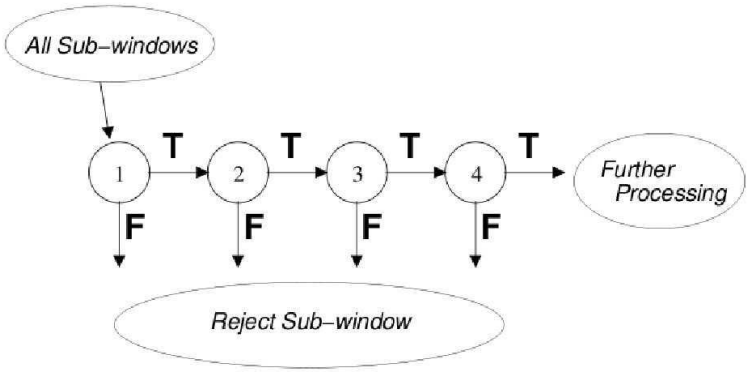


Figura 3.7: Descrizione schematica della cascata di riconoscimento. Una serie di classificatori sono applicati ad ogni sottofinestra. Il primo classificatore elminina un gran numero di esempi negativi con un bassissimo costo computazionale. Gli strati seguenti eliminano ulteriori esempi negativi, ma con un tempo computazionale via via crescente.

zione e un elevato grado di affidabilità nel rilevamento dei volti, ovvero un basso tasso di falsi positivi.

3.1.5 Algoritmo di apprendimento della cascata di classificatori

Come detto nella parte finale del precedente paragrafo, la costruzione della cascata di classificatori è strettamente correlata con il tipo di riconoscimento che si deve eseguire e con gli obiettivi prestazionali e computazionali richiesti. Il numero degli strati della cascata deve essere tale da permettere il raggiungimento di bassi tassi di falsi positivi (nell'ordine di $10^{-5}, 10^{-6}$), minimizzando però i tempi necessari alla computazione.

Data una cascata di classificatori, il numero (rate) di falsi positivi della cascata può essere calcolato come:

$$F = \prod_{i=1}^{k} f_i$$

dove k rappresenta il numero di classificatori e f_i rappresenta il rate di falsi positivi dell'i-esimo classificatore.
Allo stesso modo è possibile esprimere anche il rate di riconoscimento

$$D = \prod_{i=1}^{k} d_i$$

dove d_i rappresenta il rate di riconoscimenti corretti dell'i-esimo classificatore.

Dato dunque un preciso obiettivo circa il numero complessivio di falsi positivi ed il numero di riconoscimenti corretti, i loro reali valori, ottenibili tramite una cascata di classificatori, sono calcolabili a partire dalle caratteristiche di ogni classificatore. Per fare un esempio, un rate di riconoscimento di 0.9 può essere raggiunto tramite la costruzione di una cascata formata da dieci classificatori, ognuno dei quali ha un rate di riconoscimento di 0.99 permettendo però di ottenere un rate di falsi riconoscimenti minore rispetto a quello del singolo classificatore.

Viene ora riportato un possibile approccio per la costruzione della cascata:

- l'utente seleziona il rate massimo accettabile per f_i e il minimo valore di d_i accettabile;
- per ogni strato della cascata viene utilizzato l'algoritmo di AdaBoost
- il rate di f_i e d_i trovato viene poi valutato testando il riconoscitore ottenuto sino ad ora sul validation-set
- se i livelli di riconoscimento e di falsi positivi ottenuti non sono conformi alle specifiche inizialmente poste, si aggiunge un ulteriore strato alla cascata, permettendo così una riduzione dei falsi riconoscimenti.

Il precedente elenco può essere meglio rappresentato in pseudo codice come segue:

- l'utente seleziona il valore di f_i (massimo rate accettabile di falsi positivi per ogni strato) e di d_i (minimo rate accettabile di riconoscimenti corretti per strato);
- L'utente seleziona anche il massimo rate complessivo di falsi positivi F_{target};
- P = insieme di esempi positivi;
- N = insieme di esempi negativi;
- $F_0 = 1.0$; $D_0 = 1.0$;
- i = 0;
- **while**$(F_i > F_{target})$
  - $i \leftarrow i + 1$;
  - $n_j = 0; F_i = F_{i-1}$;
  - **while**$(F_i > f \times F_{i-1})$;
    - $n_i \leftarrow n_i + 1$;
    - vengono utilizzati P e N per allenare il classificatore con n_i feature usando l'AdaBoost;
    - si valuta la cascata ottenuta sul validation-set per determinare F_i e D_i;
    - si diminuisce la soglia per l'i-esimo classificatore fintanto che la corrente cascata ha un rate di riconoscimento di almeno $d \times D_{i-1}$.
  - $N \leftarrow 0$;
  - **if**$(F_i > F_{target})$;
    - Si valuta la cascata di classificatori su un set di esempi non contenenti volti inserendo poi eventuali falsi positivi riscontrati nell'insieme N.

3.2 Strutture Base delle OpenCV

Prima di procedere con l'analisi dell'algoritmo qui proposto, si rende necessaria una rapida trattazione di quelle che sono le principali strutture utilizzate dalle librerie OpenCV.

3.2.1 Strutture dati primitive

Le librerie OpenCV mettono a disposizione dell'utente numerose strutture dati primitive che, pur essendo complicate dal punto di vista del codice in C, sono di per sè relativamente semplici e semplificano notevolmente il lavoro di chi decide di cimentarsi nel loro utilizzo. Le strutture dati primitive, schematizzate in Figura 3.8, sono le seguenti:

- **cvPoint:** è una semplice struttura dati molto utile per la rappresentazione di punti nello spazio. I suoi parametri sono infatti due interi, x e y che rappresentano le coordinate del punto. Vi sono inoltre altre due strutture dati legate ad essa, ovvero:
  - **cvPoint2D32f:** i due parametri, in questo caso, sono due floating point;
  - **cvPoint3D32f:** ai due parametri x e y in rappresentazione floating point, si aggiunge un terzo parametro z;
- **cvSize:** indica le dimensioni di un'immagine. I suoi parametri sono width e height e vengono dichiarati di tipo intero. Come nel caso della strutture cvPoint esiste una variante;
  - **cvSize2D32f:** i cui membri sono in rappresentazione floating point;
- **cvRect:** rappresenta una porzione di un'immagine. Essa risulta strettamente collegata con le due strutture precedenti, i suoi parametri, infatti, sono x, y, width e height.
- **cvScalar:** utile per la rappresentazione dei valori RGBa di un'immagine. Tale struttura è un insieme di quattro double point accessibili attraverso il puntatore *val*, unico parametro della struttura, il quale punta ad un array contenente i quattro valori.

Structure	Contains	Represents
CvPoint	int x, y	Point in image
CvPoint2D32f	float x, y	Points in $\Re^2$
CvPoint3D32f	float x, y, z	Points in $\Re^3$
CvSize	int width, height	Size of image
CvRect	int x, y, width, height	Portion of image
CvScalar	double val[4]	RGBA value

Figura 3.8: Strutture primitive delle librerie OpenCV.

3.2.2 Rappresentazione di matrici ed immagini

Le OpenCV mettono a disposizione una gerarchia di classi utilizzabili per la rappresentazione di matrici, immagini e non solo. La gerarchia è riportata in Figura 3.9.

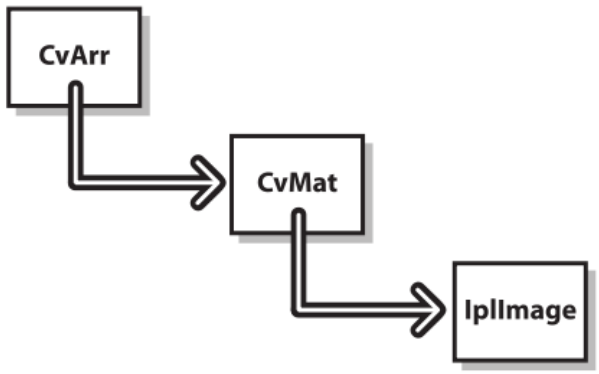


Figura 3.9: Gerarchia utilizzata dalle openCV per la rappresentazione di matrici ed immagini.

Partendo dal basso abbiamo:

- **IplImage:** è forse la struttura più utilizzata, essa permette infatti di rappresentare in maniera semplice, e soprattutto funzionale, quelle che vengono comunemente chiamate immagini, indipendentemente dalla loro tipologia: in scala di grigi, a colori, four-channel, etc.;
- **CvMat:** la relazione che intercorre tra questa struttura e quella precedente è molto simile al concetto di *ereditarietà* introdotta nel C++. Questa struttura in particolare, viene utilizzata per la rappreentazione di matrici.
- **CvArr:** questa classe può essere vista come la classe astratta dalla quale deriva poi CvMat. Solitamente in tutti i metodi disponibili per analizzare o modificare immagini o matrici, si fa riferimento a

CvArr, anche se nella pratica si utilizzano solamente le sue due strutture derivate CvMat e IplImage, che verranno comunque accettate come parametro.

CvMat: Struttura per le matrici

è necessario introdurre due aspetti fondamentali prima di poter utilizzare e analizzare tale struttura dati:

1. all'interno delle librerie openCV non sono implementate strutture appositamente dedicate alla rappresentazione di vettori. Pertanto, quando si è costretti ad utilizzare tale struttura, è necessario ricorrere ad una matrice avente una sola colonna;

2. il concetto stesso di matrice, nelle suddette librerie, è molto più astratto di quello comunemente spiegato nei vari corsi di algebra. In particolare, non è detto che gli elementi di una matrice siano solamente dei numeri interi. Ogni posizione della matrice stessa potrà infatti contenere una serie di valori multipli, i quali, a loro volta, potranno essere dichiarati di un qualsiasi tipo. Il motivo principale, ma non l'unico, che ha portato a tale *generalizzazione* del concetto di matrice, è rappresentato dal fatto che la struttura a matrice nelle openCV è fatta appositamente per la rappresentazione di immagini, quindi deve permettere la rappresetazione di tutte le possibili tipologie di immagini: RGB, scala di grigi etc.

L'implementazione della CvMat è la seguente

```
typedef struct CvMat {
    int type;
    int step;
    int* refcount;  //for internal use only

    union {
        uchar* ptr;
        short* s;
        int* i;
        float* fl;
        double* db;
    } data;

    union {
```

```
        int rows;
        int height;
    };

    union {
        int cols;
        int width;
    };
}; CvMat;
```

IplImage: Struttura per immagini

Se guardata con attenzione la struttura *IplImage* non si differenzia di molto da quella precedente. Essa infatti contiene alcuni parametri che le consentono di essere interpretata come un'immagine.

```
typedef struct IplImage {
    int nSize;
    int ID;
    int nChannels;
    int alphaChanel;
    int depth;
    char colorModel [4];
    char channelModel [4];
    int  dataOrder;
    int origin;
    int align;
    int width;
    int height;
    struct IplROI* roi;
    struct IplImage* maskROI;
    void* imageId;
    struct IplTitleInfo* titleInfo;
    int imageSize;
    char* imageData;
    int widthStep;
    int BorderMode [4];
    int BorderConst [4];
    char* imageDataOrigin;
} IplImage;
```

I parametri principali di tale struttura sono:

- **depth:** identifica la tipologia di immagine, accettado uno dei valori riportati in Figura 3.10.

- **origin:** accetta due possibili valori:

  - IPL_ ORIGIN_ TL: Se l'origine delle coordinate è ipotizzato nell'angolo in alto a sinistra dell'immagine;
  - IPL_ ORIGIN_ BL: Se l'origine delle coordinate è ipotizzato nell'angolo in basso a sinistra dell'immagine;

  La mancanza di uno standard per l'interpretazione dell'origine delle coordinate, è una delle principali cause d'errore nella creazione di applicazioni di computer vision;

- **dataOrder:** accetta due possibili valori

  - IPL_ DATA_ ORDER_ PIXEL
  - IPL_ DATA_ ORDER_ PLANE

  Tali valori indicano rispettivamente il modo in cui i dati dovranno essere raggruppati in multicanali; uno dopo l'altro per ogni pixel, oppure prima tutti i singoli valori di un singolo piano seguiti dai valori del successivo;

- **widthStep:** rappresenta il numero di byte tra i punti della stessa colonna tra righe successive.

Vi è anche un'ultimo importante membro della struttura IplImage, che indica una particolare regione, interna all'immagine rappresentata, avente un valore significativo; tale regione viene chiamata **ROI (Region Of Interest)**. Questo parametro risulta essere fondamentale, nelle fasi di analisi o di modifica dell'immagine, infatti se una ROI è stata impostata, si potrà scegliere di applicare una qualsivoglia operazione solamente a tale regione piuttosto che su tutta l'immagine.

Macro	Image pixel type
IPL_DEPTH_8U	Unsigned 8-bit integer (8u)
IPL_DEPTH_8S	Signed 8-bit integer (8s)
IPL_DEPTH_16S	Signed 16-bit integer (16s)
IPL_DEPTH_32S	Signed 32-bit integer (32s)
IPL_DEPTH_32F	32-bit floating-point single-precision (32f)
IPL_DEPTH_64F	64-bit floating-point double-precision (64f)

Figura 3.10: Tipologie di immagine riconosciute dalle openCV.

3.3 Analisi dell'algorimo di Haar utilizzato

Tramite l'uso delle librerie openCV, è stato possibile implementare l'algoritmo di Haar, presentato nel paragrafo precedente, per la localizzazione del volto nei filmati a disposizione. Per come è stata strutturata l'applicazione, l'algoritmo viene inizializzato un'unica volta all'avvio dell'applicazione e successivamente viene utilizzato su ogni frame del filmato. Una volta finita l'analisi, viene restituito un valore booleano vero o falso a seconda che sia stato localizzato un possibile candidato o meno (Figura 3.11).

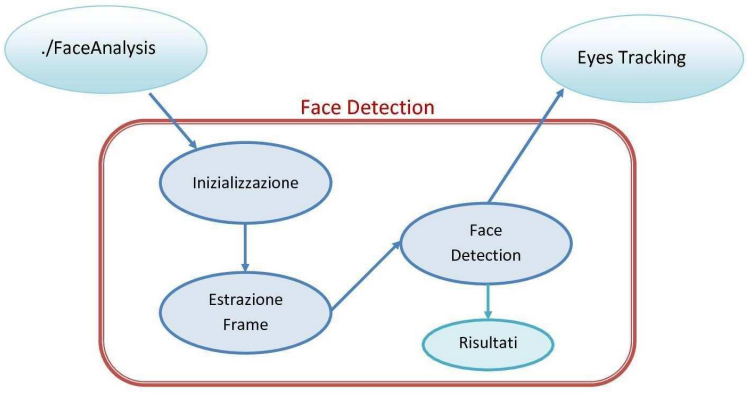


Figura 3.11: Diagramma dell'algoritmo implementato per la localizzazione del volto.

3.3.1 Fase di inizializzazione

Come detto precedentemente, i parametri dell'algoritmo vengono inizializzati all'avvio dell'applicazione. L'algoritmo è contenuto all'interno della classe Videograbber e tramite il suo costruttore,

```
VideoGrabber* vb = new VideoGrabber();
```

vengono impostati sia il video nel quale condurre l'analisi di face analysis, sia la cascata di classificatori da utilizzare nell'algoritmo di Haar per il riconoscimento del volto.

```
cvCapture* capture;
capture = cvCaptureFromAvi("path_file_video.avi");

string cascadeFace = "classificatore.xml";
```

Le librerie openCV permettono, in modo del tutto analogo, di associare alla variabile *capture* un qualsivoglia dispositivo di acquisizione video (usb o firewire), come per esempio una webcam. In questo modo è possibile condure un'analisi real-time sul filmato acquisito,

```
capture = cvCaptureFromCAM(int index);
```

dove index rappresenta quale dispositivo collegato al computer si deve utilizzare per l'acquisizione (utile in presenza di più dispositivi ad esempio webcam usb più webcam integrata). è anche possibile utilizzare più sorgenti di acquisizione contemporaneamente, come ad esempio 2 webcam aventi due prospettive differenti della scena ripresa.

Come detto, in questa fase di inizializzazione, viene anche indicato il classificatore che l'algoritmo di Haar dovrà utilizzare per condurre l'analisi sul filmato in ingresso. Le librerie in esame offrono una discreta scelta di classificatori al loro interno:

- **haarcascadefrontalfacealt1.xml**: utilizzabile per il riconoscimento del volto
- **haarcascadefrontalfacealt2.xml**: utilizzabile per il riconoscimento del volto (sono stati utilizzati esempi 24x24)
- **haarcascadeprofileface.xml**: utilizzabile per il riconoscimento del profilo del volto
- **haarcascadefullbody.xml**: utilizzabile per il riconoscimento di tutto il corpo
- **haarcascadelowerbody.xml**: utilizzabile per il riconoscimento del corpo escluso il volto
- **haarcascadeupperbody.xml**: utilizzabile per il riconoscimento del corpo dalla vita in su.

Nel presente progetto, pur sapendo che in rete si possono trovare molti classificatori per il riconoscimento del volto, si è scelto di utilizzare il classificatore con campioni 24x24 offerto dalle openCV, che per la tipologia di filmati a disposizione, si è dimostrato il più performante.

3.3.2 Estrazione del frame dal video sorgente

Il passo successivo consiste nell'estrazione del singolo frame dal video in ingresso. L'algoritmo di Haar, infatti, può unicamente lavorare su una singola immagine alla volta e quindi, in caso di filmati, si rende necessario estrrarre un frame alla volta. L'estrazione avviene ricorrendo alle librerie openCV, che mettono a disposizione una serie di metodi molto utili a tale scopo:

```
cvGrabFrame(capture);
```

Il metodo *cvGrabFrame* permette l'estrazione del frame dal video associato alla variabile capture e viene mantenuto in memoria. Per applicare su tale frame l'algoritmo di face detection, occorre salvarlo all'interno della struttura utilizzata dalle openCV per la rappresentazione dell'immagine: l'*IplImage*

```
IplImage* frame;

frame = cvRetrieveFrame(capture);
```

3.3.3 Il cuore dell'algoritmo di Haar

Finita la fase di inizializzazione e di estrazione del frame dal video in ingresso, viene utilizzato l'algorimo di Haar per localizzare il volto all'interno dell'immagine. Il metodo, interno alla classe videoGrabber, che implementa l'algoritmo di face detection è il seguente:

```
void VideoGrabber::detectObject_andDrow(IplImage* &img,
                    string cascadeObject,
                    CvPoint &pt1, CvPoint &pt2,
                    int frameId);
```

Vediamo i signiicati dei vari parametri

- **img**: rappresenta l'immagine estratta dal video in ingresso, alla quale si deve applicare l'algoritmo di face detection;

- **cascadeObject**: rappresenta il classificatore che verrà utilizzato per localizzare il volto nell'immagine. Il classificatore può essere visto come ciò che comunica all'algoritmo di Haar di localizzare il volto nell'immagine piuttosto che altri elementi;

- **pt1, pt2**: parametri utilizzati dalle openCV per rappresentare i punti nel piano, pt1 e pt2 vengono impiegati in una fase successiva al fine di disegnare sul frame del video il rettangolo di interesse localizzato dall'algorimo;

- **frameId**: rappresenta un semplice contatore per tenere traccia della posizione del frame in analisi all'interno del video.

Per avviare l'analisi occorre, per prima cosa, identificare la tipologia di classificatore e caricare lo stesso:

```
cvHaarClassifierCascade* cascadeObj;

cascadeObj = (cvHaarClassifierCascade*)
             cvLoad(cascadeObject);
```

Occorre inoltre inizializzare una particolare struttura dinamica, messa a disposizione dalle librerie utilizzate, per memorizzare tutte le regioni di interesse localizzate all'interno dell'immagine.

```
cvSeq* object;
```

Il cuore dell'algoritmo di face detection è rappresentato dalle seguenti righe di codice:

```
CvSeq* object = cvHaarDetectObjects(const cvArr* image,
                cvHaarClassifierCascade* cascade,
                cvMemStorage*storage,
                double scale_factor=1.1,
                int min_neighbors=3,
                int flags=0,
                cvSize min_size=cvSize(0,0));
```

Vediamo i significati dei vari parametri

- **image**: rappresenta l'immagine su cui deve essere condotta la ricerca del volto;
- **cascade**: è il classificatore utilizzato per l'analisi;
- **storage**: rappresenta la porzione di memoria allocata per ospitare il salvataggio delle regioni d'immagine restituite dall'algoritmo;
- **scale_ factor**: valore che indica quanto una finestra deve essere scalata tra un'analisi l'altra. Il valore di default 1.1 indica che essa verrà ingrandita del 10% ad ogni passo;
- **min_ neighbors**: consente di controllare il numero di rettangoli d'interesse vicini, trovati nell'immagine in analisi. Se impostato a -1, i gruppi di rettangoli più piccoli verranno eliminati, mentre se impostato a 0, verrà disabilitata tale funzione;
- **flags**: rappresenta la modalità in cui l'algoritmo di face detection deve operare. Nella versione delle openCV utilizzata, è possibile utilizzare una sola funzione, *cvHaarDoCannyPruning*. Tale parametro permette di utilizzare il Canny edge detector, al fine di scartare quelle porzioni di immagini aventi troppi o troppo pochi bordi al loro interno;
- **min_ size**: indica la grandezza minima della finestra. Di default le openCV impostano tale parametro alla grandezza degli esempi utilizzati dai classificatori (circa $20x20$ per la face detection).

I parametri precedentemente citati, possono essere combinati in svariate modalità permettendo di agire sia sull'efficienza che sulle prestazioni computazionali dell'algoritmo. Si è notato, che nelle applicazioni in real-time, è possibile ottenere ottime prestazioni computazionali ed un'ottima efficienza attraverso l'utilizzo dei seguenti valori:

- scale_ factor = 1.2
- min_ neighbors = 3
- flag = 0

Nell'applicazione qui presentata, dove si è cercato di trovare un compromesso tra le prestazioni computazionali dell'algoritmo e l'efficienza dello stesso, dopo alcuni tentativi sono stati selezionati i seguenti valori.

- scale_ factor = 1.1
- min_ neighbors = 4
- flag = CvHaarDoCannyPruning
- min_ size = cvSize(100,100)

L'alto valore assegnato ai due parametri *min_ neighbors* e *min_ size* è dovuto al fatto che, nei video analizzati all'interno del presente progetto, può comparire un solo volto, le cui dimensioni sono tali da occupare gran parte dell'immagine. Queste impostazioni, al contrario di quelle precedentemente descritte, hanno dato la possibilità di scartare alcuni falsi positivi, dovuti principalmente a differenti condizioni di luce e sfondi particolari.

3.3.4 Rappresentazione del volto nel video

Una volta conclusa l'analisi dell'immagine corrente, i risultati ottenuti, vengono tracciati sull'immagine e stampati a video. Per fare questo, le librerie openCV, mettono a disposizione una particolare struttura *cvRect*, che consente di creare un rettangolo leggendo i risultati ottenuti dall'analisi precedente ed opportunamente salvati nella struttura dati dinamica *cvSeq*

```
CvRect* r = (CvRect*)cvGetSeqElem( object, i );
```

Vengono poi estratti i due punti *pt1* e *pt2*

```
pt1.x = r->x*scale;
pt2.x = (r->x+r->width)*scale;
pt1.y = r->y*scale;
pt2.y = (r->y+r->height)*scale;
```

ed infine si procede con il disegnare il rettangolo contenente il volto sull'immagine.

```
cvRectangle( img, pt1, pt2, CV_RGB(0,255,0), 3, 8, 0 );
```

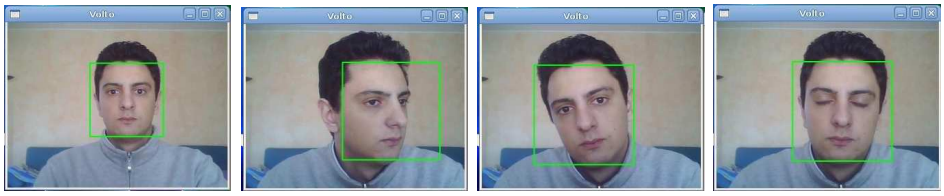
3.3.5 Estrazione e salvataggio dei dati

Per quanto riguarda il volto, si è scelto di salvare alcune informazioni estrapolate dall'analisi di ogni frame del video in ingresso, come la larghezza e l'altezza del volto, il numero di pixel e il centro del rettangolo di interesse. Tali caratteristiche sono infatti calcolabili a partire dai risultati dell'algoritmo di face detection precedentemente illustrato.

```
//Dimensioni del volto nel frame
r->width;
r->height;

//Numero di pixel
numPixel = (r->width)*(r->height);

//Centro del volto rispetto al frame analizzato
c.x = (pt1.x)+(pt2.x/2);
c.y = (pt1.y)+(pt2.y/2);


//Salvo i risultati su file
fileDimVolto<<frameId<<","<<r->width<<","
            <<r->height<<","<<numPixel<<endl;
filePosCentro<<frameId<<","<<c.x<<","<<c.y<<endl;
```

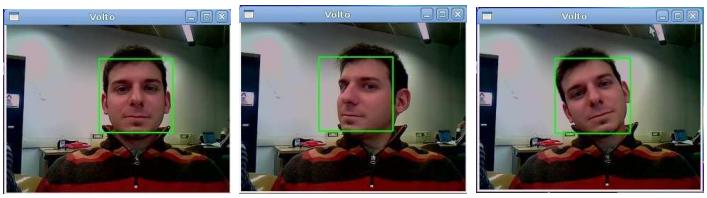
3.4 Risultati ottenuti

Come anticipato all'inizio del capitolo, l'algoritmo di face detection di Viola-Jones implementato in questa applicazione, si è rilevato molto performante, sia in termini di efficienza, che in termini di prestazioni computazionali molto buone. Come è possibile vedere dalle immagini dei test riportate in Figura 3.12 Figura 3.13 Figura 3.14, l'algoritmo ha dato buoni risultati riconoscendo correttamente il volto in numerose pose ed in differenti condizioni di luce indipendentemente dallo sfondo in cui esso era inserito.



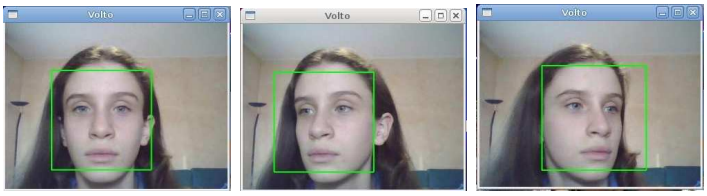
(a) (b) (c) (d)

Figura 3.12: Test 1



(a) (b) (c)

Figura 3.13: Test 2



(a) (b) (c)

Figura 3.14: Test 3

Capitolo 4

Eye Tracking

"Morpheus: Che vuol dire reale? Dammi una definizione di reale. Se ti riferisci a quello che percepiamo, a quello che possiamo odorare, toccare e vedere, quel reale sono semplici segnali elettrici interpretati dal cervello. Questo è il mondo che tu conosci. Il mondo com'era alla fine del XX secolo. E che ora esiste solo in quanto parte di una neuro-simulazione interattiva che noi chiamiamo Matrix. Sei vissuto in un mondo fittizio, Neo."

Matrix

In questo capitolo, verrà descritto nel dettaglio l'algoritmo proposto per la localizzazione degli occhi all'interno dell'immagine del volto precedentemente estratta. La sorgente d'informazione è rappresentata, in questo caso, da una regione d'immagine di dimensioni non fissate, all'interno della quale, dovranno essere localizzate le posizioni degli occhi. Tra le possibili alternative offerte dalla letteratura, di cui si è trattato nel corso del Capitolo 2, si è scelto di utilizzare un particolare algoritmo, precedentemente scritto dal prof. Matteucci, opportunamente modificato per affrontare le problematiche qui trattate.

L'algoritmo di *Blob Analysis* proposto, è stato utilizzato per la localizzazione, all'interno di un'immagine, di alcune aree aventi caratteristiche associabili a quelle degli occhi; tale localizzazione avviene mediante l'utilizzo di alcune tecniche di *Image processing* ed, in particolare, di tecniche di *segmentazione dell'immagine*.

4.1 Image Processing: La Segmentazione dell'immagine

Nel campo della Computer Vision vengono trattate tecniche di *Segmentazione dell'immagine*, in riferimento a particolari processi di partizionamento di un'immagine digitale in regioni multiple.

L'obiettivo di questo processo è quello di utilizzare una specifica rappresentazione digitale dell'immagine, al fine di semplificarne l'analisi, soprattutto per quanto riguarda quei processi mirati alla localizzazione di oggetti o contorni all'interno di immagini o video. I risultati del processo di segmentazione, consistono in un insieme di regioni (ossia un raggruppamento di pixel) che, se considerate nel loro complesso, ricoprono l'intera immagine o, in alternativa, l'insieme di contorni estratti dall'immagine originale. Ogni pixel appartenente alla stessa regione, sarà caratterizzato dalla stessa peculiarità, in base alla caratteristica con cui l'immagine è stata analizzata (colore, intensità, texture, etc.), mentre, le regioni vicine, saranno caratterizzate da valori differenti l'una dalle altre.

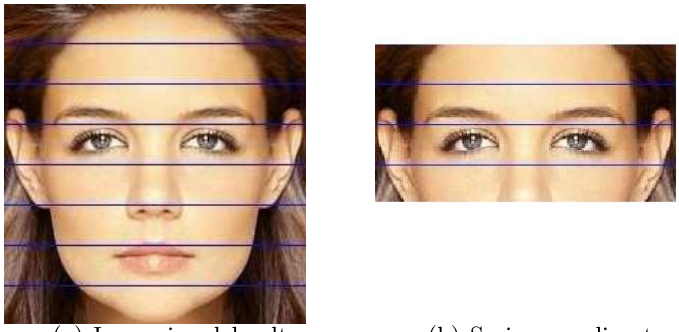
Le tecniche di segmentazione dell'immagine trovano utilizzo principalmente in quei processi atti alla localizzazione di oggetti in immagini o video. Tra le principali applicazioni è possibile citare:

- **Medical Imaging:** Particolari processi per l'analisi d'immagini del corpo umano a scopi clinici come:
  - localizzazione di tumori e patologie simili
  - misurazioni di tessuti organici
  - studi sulla struttura anatomica del corpo
- **Localizzazione di oggetti in immagini satellitari:** Negli ultimi anni, sono state sviluppate numerose applicazioni per l'analisi di immagini satellitari, sia per uso scientifico che per uso domestico, come, ad esempio, quelle utilizzate dai più moderni GPS, ossia:
  - localizzazione di strade;
  - localizzazione delle zone verdi come boschi e foreste;
  - localizzazione di edifici;
  - sviluppo di sistemi per il controllo del traffico;

4.2 Algoritmo Proposto

Per determinare la posizione degli occhi all'interno dell'immagine del volto, l'algoritmo di **Blob Analysis** è stato suddiviso in tre differenti parti:

- **Background subtraction:** questa prima analisi, in realtà, viene svolta in un modo del tutto automatico dall'algorimo di Face Detection descritto nel capitolo 3. Tale algoritmo, infatti, permette di estrarre una regione dell'immagine originale contenente il volto, eliminando le altre regioni esterne. Per ridurre ulteriormente le dimensioni della regione da analizzare, tramite l'utilizzo dell'algoritmo di Blob Analysis, l'immagine contenente il volto è stata ulteriormente ritagliata come evidenziato in Figura 4.1



(a) Immagine del volto (b) Sezione analizzata

Figura 4.1: Riduzione della regione d'immagine analizzata dall'algoritmo di Blob Analysis.

- **Blob Detection:** sistema che permette la localizzazione degli oggetti presenti all'interno dell'immagine analizzata, indicando, per ognuno di essi, la forma, il colore, la posizione, etc. Per limitare il numero di oggetti riconosciuti, è possibile analizzare l'immagine con l'algoritmo di background subtraction prima di applicare l'algoritmo di blob detection.

- **Object Tracking:** questa terza sezione, applicata dopo gli algoritmi citati nei punti precedenti, ha lo scopo d'identificare e seguire solo alcuni degli oggetti tra tutti quelli trovati nell'immagine precedentemente analizzata.

Un diagramma rappresentante le tre differenti fasi dell'algoritmo è mostrato in Figura 4.2

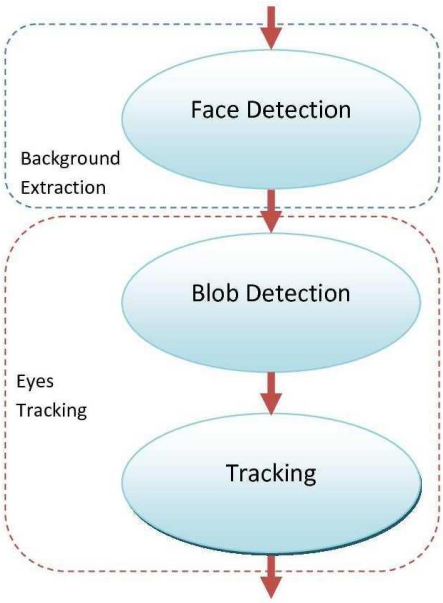


Figura 4.2: Le tre sezioni dell'algoritmo di Blob Analysis implementato.

4.2.1 Identificazione e riconoscimento dei blob

Tramite la localizzazione dei blob, è possibile identificare nell'immagine in analisi la posizione di punti e/o regioni, caratterizzati da particolari valori di intensità, colore, ect. che li rendono differenti da ciò che li circonda. Il principale motivo, per cui ancora oggi si conducono attente ricerche sugli algoritmo di blob detection, è rappresentato dal fatto che questi permettono di accedere ad informazioni inerenti a particolari regioni dell'immagine, le quali non sarebbero disponibili attraverso l'utilizzo dei normali algoritmi di image processing come l'edge detection o il contour detection. Una delle principali applicazioni degli algoritmi di Blob Detection è rappresentata dal loro utilizzo nell'estrazione di regioni d'interesse, ossia quelle zone che potrebbero contenere un particolare oggetto da utilizzare in analisi future.

Il grafico dell'algoritmo utilizzato per l'identificazione degli occhi nell'immagine in Figura 4.1(b), è mostrato nella Figura 4.3.

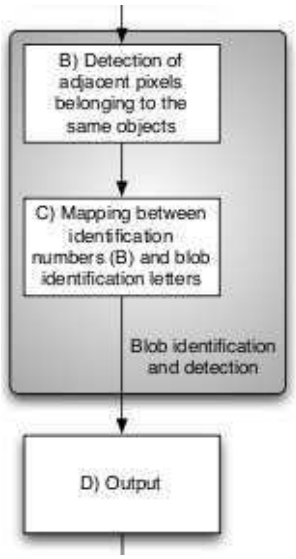


Figura 4.3: Identificazione dei blob ed algoritmo di riconoscimento.

- Nella prima fase (Figura 4.3(B)), l'immagine ritagliata del volto, mostrata in Figura 4.1(b), viene analizzata al fine di associare ad ogni pixel appartenente allo stesso oggetto (alla stessa regione) il medesimo numero d'identificazione. Alla fine di questa prima analisi, vengono create alcune liste (una per ogni oggetto o regione localizzata all'interno dell'immagine) contenenti le etichette che identificano tutti i pixel interni a ciascuna regione d'interesse. Queste ultime verranno poi analizzate nuovamente dalla seconda parte dell'algoritmo.

- La seconda fase (Figura 4.3(C)), esegue una seconda analisi dell'immagine, al fine di creare una mappa (identificata da un'etichetta), di tutti gli identificatori numerici che contraddistingono i pixel contenuti in una determinata regione d'interesse. Alla fine di questa seconda analisi, l'immagine elaborata, è suddivisa in regioni, i cui pixel sono identificati da una sola etichetta, che ne permette la distinzione.

Al fine di fornire una migliore spiegazione del precedente sistema d'identificazione ed analisi dei blob contenuti in un'immagine, verrà ora mostrato un piccolo esempio. Si consideri, a tale scopo, un'immagine di 144 pixel (16×9).

L'immagine da analizzare, consiste di quattro oggetti di forma diversa due dei quali sono caratterizzati dallo stesso colore. (Figura 4.4). Tale figurarappresenta il risultato ottenuto tramite la precedente tecnica di background subtraction e, nel caso in analisi, rappresenterebbe la regione di volto ritagliata in Figura 4.1(b).

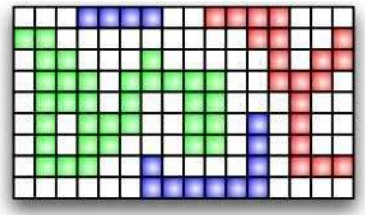


Figura 4.4: Esempio di immagine da analizzare.

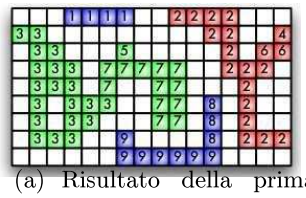
Fase 1: Blob Detection

Come detto precedentemente, la prima fase dell'algoritmo, consiste nel riconoscimento dei pixel adiacenti appartenenti allo stesso oggetto. Questa analisi può essere compiuta associando un particolare valore ad ogni pixel, secondo le seguenti politiche:

- Il primo pixel nell'angolo in alto a sinistra (ipotizzato come origine delle coordinate) nell'immagine in analisi, verrà identificato dal valore 0 se appartenente allo sfondo, oppure dal valore 1 se appartenente ad una regione d'interesse.
- Ogni altro pixel della prima riga verrà identificato dal valore:
  - 0 se appartenente allo sfondo;
  - con lo stesso valore del pixel alla sua sinistra se entrambi appartengono ad una regione d'interesse;
  - con un nuovo valore (mai utilizzato prima) se il pixel appartiene ad una regione d'interesse, ma il pixel alla sua sinistra risulta identificato dal valore 0;
- Ogni altro pixel appartenente alle righe successive verrà identificato, dal valore:
  - 0 se appartenente allo sfondo;
  - con lo stesso valore del pixel alla sua sinistra se entrambi appartengono ad una regione d'interesse;

- con lo stesso valore del pixel immediatamente al di sopra se entrambi appartengono ad una regione d'interesse ma quello alla sua sinistra risulta identificato dal valore 0;
- con un nuovo valore (mai utilizzato prima) se esso appartiene ad una regione d'interesse e il pixel alla sua sinistra risulta identificato dal valore 0;

La Figura 4.5(a) mostra i risultati ottenibili tramite l'applicazione della precedente analisi sulla Figura 4.4



(a) Risultato della prima fase di analisi

```
A -> 1
B -> 2, 4, 6
C -> 3, 5, 7
D -> 8, 9
```

(b) Mappa dei valori dei singoli pixel con relativa etichetta assegnata ad ogni regione d'interesse

Figura 4.5:

Fase 2: Blob Identification

Una volta conclusa la prima fase di riconoscimento di tutte le regioni d'interesse all'interno dell'immagine Figura 4.4, si passa alla fase successiva, la quale ha l'obiettivo di etichettare, con la stessa lettera, tutti i valori dei pixel appartenenti ad una stessa regione. (Figura 4.5(b)). Dalla Figura 4.5(a) è possibile vedere che il valore 1 può essere associato all'etichetta A, mentre la regione B contiene al suo interno solamente i pixel precedentemente identificati dai valori 2, 4, e 6. Allo stesso modo, la regione C raggruppa tutti i pixel identificati dai valori 3, 5 e 7, mentre l'ultima regione D contiene i pixel identificati con i valori 8 e 9. La Figura 4.6 mostra i risultati ottenuti tramite questa seconda analisi.

4.2.2 Blob Tracking

Viene indicato con *Object Tracking* quel processo che permette la localizzazione di oggetti in movimento mediante l'utilizzo di una telecamera. Un algoritmo che renda realizzabile il suddetto processo, deve quindi poter analizzare il frame video, rendendo poi disponibili, come risultati, le posizioni delle varie regioni d'interesse presenti nel frame analizzato. L'algoritmo

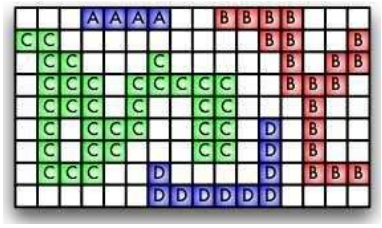


Figura 4.6: Risultato della seconda fase di analisi.

proposto per la realizzazione di un sistema di Object Tracking può essere schematizzato come in Figura 4.7

L'algoritmo qui proposto per l'object tracking, come quello di Blob detection, si compone di due fasi principali.

Analisi di primo livello

La prima fase dell'algoritmo, consiste nell'analizzare l'intera lista di blob, risultante dal processo di blob detection di cui si è parlato nel paragrafo precedente, al fine di localizzare le sole regioni d'interesse. Nel caso in analisi, le sole regioni utili sono quelle contenenti gli occhi, le quali dovranno essere estratte e successivamente analizzate. A tal fine, si è scelto di utilizzare un sistema di selezione basato su alcune caratteristiche che i blob d'interesse dovranno soddisfare per essere considerati tali:

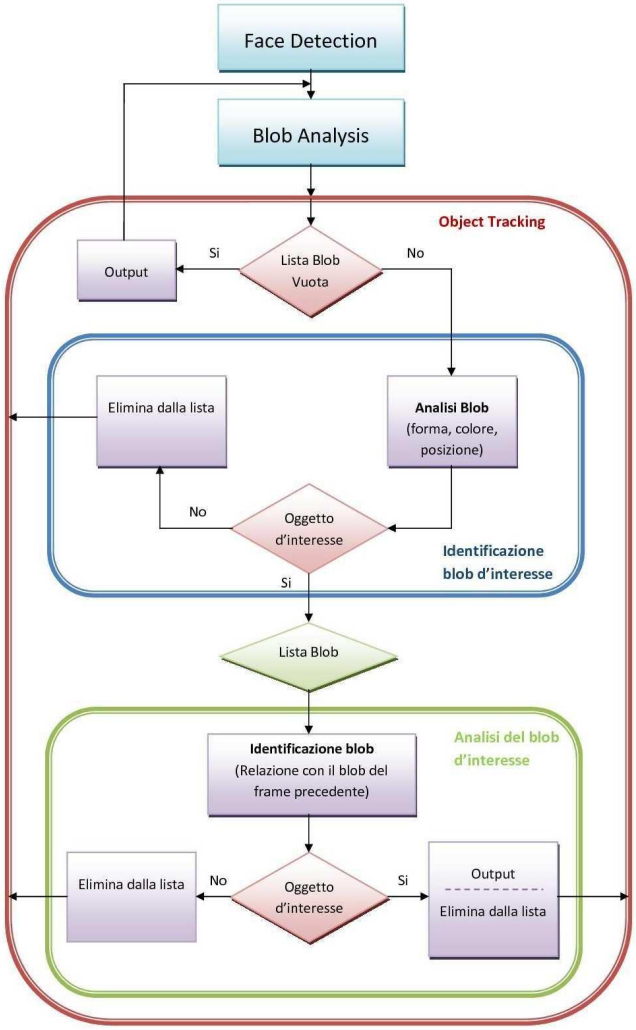


Figura 4.7: Algoritmo di Object Tracking proposto.

- **Classe:** Rappresenta l'etichetta assegnata dalla seconda fase dell'algoritmo di blob detection, la quale identifica tutti i pixel contenuti in una determinata regione d'interesse. L'algoritmo di blob detection qui utilizzato, assegna la stessa etichetta K a tutte le regioni caratterizzate da pixel aventi colori simili a quelli della pupilla dell'occhio umano (pixel molto scuri). La scelta di ricercare tutte le possibili regioni scure, all'interno del volto umano, è dovuta al fatto che, oltre alla pupilla (sempre di colore nero), tutta la regione intorno all'occhio è tra le più scure di tutta l'immagine, a causa della presenza delle ciglia e delle cavità oculari. (Figura 4.8). Tra tutti i blob presenti nella lista ottenuta

con l'algoritmo di blob detection vengono quindi considerati solamente quelli aventi K come etichetta.

Figura 4.8: Identificazione delle regioni con tonalità scusa (in rosso) e delle regioni leggermente più chiare (in giallo).

- **Forma:** La posizione delle regioni d'interesse, classificate come K contenute nella lista, identificano il rettangolo che meglio approssima la regione d'immagine contenente il blob trovato; tali rettangoli saranno il punto di partenza per la seconda analisi, che infatti, ne studierà la forma e la dimensioni. L'occhio, di fatto, ha di per sè una forma allungata simile ad un'ellisse racchiuso all'interno di una zona più scura (ciglia), come mostrato in Figura 4.9. è stato dunque immediato ipotizzare che i rettangoli d'interesse, rappresentanti gli occhi, dovranno avere la forma di un rettangolo posto orizzontalmente (ossia con il lato maggiore alla base).

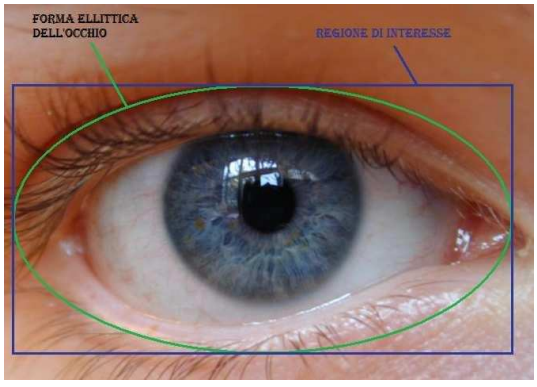


Figura 4.9: In verde è rappresentata la caratteristica forma ellittica dell'occhio umano, mentre in blu il rettangolo che meglio approssima tale ellisse.

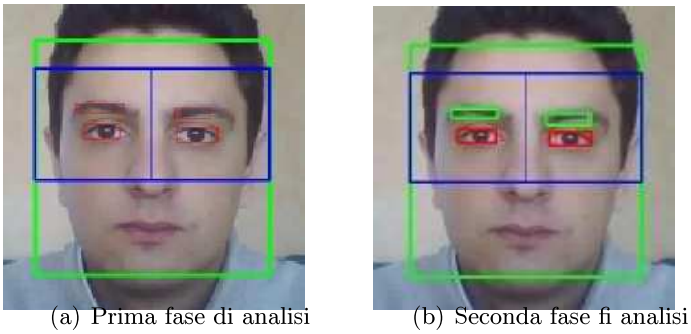
- **Dimensione:** Anche la dimensione del rettangolo, approsimante il blob localizzato all'interno dell'immagine analizzata, può essere utilizzata per eliminare le regioni più piccole trovate dall'algoritmo di blob detection. Scartando, infatti, tutte le regioni che non contengono un certo numero di pixel, è possibile eliminare facilmente i blob dovuti a piccole ombre o a piccole macchie della pelle, o in generale dovute a tutte le caratteristiche del volto umano, che sono considerate rumore per l'analisi qui affrontata. Dopo alcuni test, è stato trovato un valore di soglia α al di sotto del quale le regioni verranno scartate; nei filmati in analisi, tale valore è stato sempre impostato a 100 pixel ($\alpha = 100$).

Una volta conclusa l'analisi precedentemente descritta, la lista di blob localizzati nell'immagine dovrebbe contenere al massimo due regioni di interesse: una per ogni occhio. Tuttavia, in presenza di persone aventi determinate caratteristiche somatiche (come ad esempio sopracciglie folte e scure), la lista potrebbe contenere regioni aggiuntive; tale problematica è stata risolta attraverso l'introduzione della seconda fase di analisi dell'algoritmo di object tracking.

Analisi di secondo livello

Considerando l'immagine analizzata dall'algoritmo di blob detection (Figura 4.1), si nota come, una volta localizzati gli occhi, l'unica regione scura sia rappresentata dalle sopracciglia. Risulta inoltre possibile notare che le regioni contenenti al loro interno le sopracciglia, sono molto simili a quelle rappresentanti gli occhi; queste infatti sono caratterizzate dalla stessa classe K, in quanto sono zone molto scure, hanno dimensioni molto simili alle regioni degli occhi ed infine, anche la forma del rettangolo che ne approssima la regione, è molto simile a quella del rettangolo che approssima l'ellisse dell'occhio. In questo particolare caso, si è scelto di non disperedere le informazioni inerenti alle sopracciglia (quando esse vengono riconosciute), in quanto potrebbero costituire fonte di informazione utile, nell'ambito in cui il nostro progetto si vuole collocare.

La seconda analisi compiuta dall'algoritmo di object tracking, si pone l'obiettivo di distinguere le regioni d'interesse rappresentanti gli occhi da quelle che localizano le sopracciglia, come mostrato in Figura 4.10



(a) Prima fase di analisi (b) Seconda fase fi analisi

Figura 4.10:

L'evoluzione dell'algoritmo può essere così descritta:

- viene ipotizzato che, al primo frame, vengano riconosciuti correttamente entrambi gli occhi;
- la lista di blob risultante dall'analisi di primo livello, viene nuovamente analizzata, in modo da selezionare, tra le regioni localizzate, le quattro aventi dimensioni maggiori. Queste vengono a loro volta suddivise in due liste differenti, a seconda della loro posizione rispetto al volto. Vengono quindi create due liste, la prima contenente tutte le regioni d'interesse della parte sinistra del volto, l'altra contenete quelle localizzate nella parte destra;
- entrambe le liste vengono analizzate indipendentemente, una dopo l'altra e, ogni componente, viene confrontato con quelli della lista elaborata al frame precedente, applicando le seguenti politiche:
  - se la lista contiene un solo blob, in base alla sua posizione, si determina quale tipo di caratteristica rappresenta, tramite un confronto con il frame precedente;
    - inizialmente viene ipotizzato che il blob analizzato rappresenti una regione d'interesse contenente l'occhio;
    - si confronta quindi la sua posizione con quella dello stesso occhio nel frame precedente;
    - se il confronto dà esito positivo, il blob viene etichettato come occhio e si procede con l'analisi del componente successivo;
    - se il confronto dà esito negativo si ipotizza, invece, che il blob in analisi rappresenti un sopracciglio e si procede con l'analisi del componente successivo;

  - se la lista contiene più di una regione d'interesse:
    * quello che occupa la posizione più in alto, all'interno dell'immagine analizzata, rappresenta il sopraciglio;
    * quello che occupa la posizione più in basso, all'interno dell'immagine analizzata, rappresenta l'occhio;

Il confronto viene eseguito tra un blob del frame n ed un blob del frame $n - m$ (con $m > 0$), a cui viene applicato un ingrandimento pari ad un valore $2\Delta m$, al fine di tenere conto del movimento del volto. Tale confronto avviene con le seguenti modalità:

- Se il blob in analisi è contenuto all'interno di uno dei componenti, appartenenti alla lista del frame $n - m$, allora la regione d'interesse analizzata viene etichettata con la medesima etichetta del componente in cui essa è contenuta.

- se il blob in analisi non è completamente contenuto all'interno di un componente, appartenente alla lista del frame $n - m$, allora la regione d'interesse in analisi non appartiene alla stessa classe del componente con cui è stata confrontata.

- se il blob in analisi non è completamente contenuto all'interno di nessuno dei componenti, appartenenti alla lista del frame $n - m$, allora la regione d'interesse in analisi viene scartata, perchè rappresenta un falso positivo.

Lo sviluppo della seconda parte dell'algoritmo sopra descritta, è profondamente legato all'efficienza della prima analisi e alle caratteristiche del filmato analizzato. Il filtraggio effettuato durante la prima analisi (basato sulla classe, sulla forma e sulle dimensioni delle singole regioni di interesse localizzate), ha permesso, infatti, di scartare tutti i possibili falsi positivi localizzati al di sopra delle sopracciglia (come ad esempio l'attaccatura dei capelli) e al di sotto degli occhi (come ad esempio le narici o la bocca). In secondo luogo, il frame rate dei filmati utilizzati per testare l'algoritmo (tra i 25 ed i 30 fps), ha permesso di condurre un'analisi tra frame successivi, considerando un Δ molto piccolo (nell'ordine di 10/20 pixel) tra la posizione di un blob nel frame n e la posizione dello stesso blob nel frame $n + 1$.

4.3 Analisi dell'algoritmo di Blob Analysis

L'algoritmo di blob detection analizzato, nel paragrafo precedente, è stato sviluppato dal prof. Matteucci e viene qui modificato per meglio adattarlo alle diverse problematiche da affrontare. In particolare, come emerso dall'analisi condotta precedentemente, si è reso necessario affiancare la blob detection con un algoritmo di object tracking, utilizzato per l'identificazione delle differenti zone d'interesse localizzate nell'immagine analizzata. Prima di condurre un'analisi sui frammenti di codice che hanno permesso la realizzazione delle principali funzionalità offerte, è necessario soffermarsi sulla particolare tecnica utilizzata per addestrare l'algoritmo di blob detection, ovvero sulla fase di creazione del classificatore di colore.

4.3.1 Creazione di un classificatore

Un classificatore, in questo caso, è da intendersi come una legenda consultabile dall'algoritmo di blob detection, indispensabile per identificare e classificare correttamente le varie regioni d'interesse localizzate nell'immagine in analisi. Tale classificazione avviene infatti su alcune caratteristiche proprie di ciascun pixel, come il colore, l'intensità etc. La creazione del classificatore avviene, come mostrato in Figura 4.11, in tre differenti fasi:

- il primo passo consiste nella selezione, all'interno del filmato da analizzare, di un insieme di immagini raffiguranti il volto in diverse pose e condizioni di luce, all'interno delle quali, risultano ben chiare le caratteristiche ricercate (in questo caso gli occhi);
- dalle immagini selezionate, vengono estratte alcune regioni (ossia degli insiemi di pixel) contenenti:
  - pixel rappresentanti porzioni di pelle del volto (come ad esempio: pelle della fronte, del naso, delle guance, etc.);
  - pixel rappresentanti porzioni di pupilla (di entrambi gli occhi se visibili).

  Tutte le precedenti regioni vengono poi esaminate e classificate in base a cosa rappresentano:
  - le regioni rappresentanti la pelle vengono classificate con la lettera R (rosso);
  - le regioni rappresentanti la pupilla vengono classificate con la lettera K (nero).

Durante questa analisi, per tutti i pixel contenuti all'interno delle parti scelte, vengono memorizzati su file i valori dei tre canali RGB e la classe alla quale appartengono;

- il terzo ed ultimo passo, consiste nella creazione del classificatore a partire dai valori estratti nel passo precedente. Il classificatore così creato, verrà utilizzato dall'algoritmo di blob detection per l'identificazione delle varie regioni d'interesse contenute all'interno dei frame del filmato in analisi.

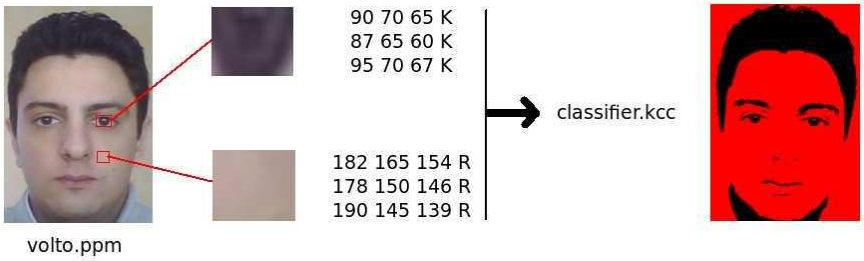
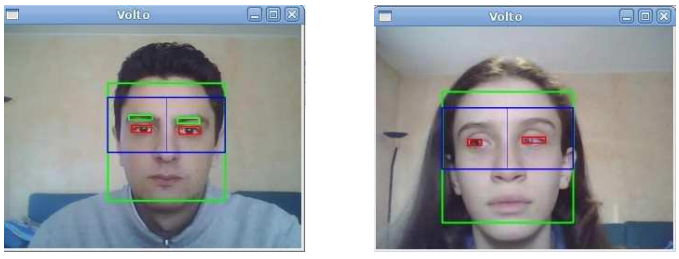


Figura 4.11: Le varie fasi della creazione di un classificatore colore.

Nel caso in esame, al fine di ottenere buone prestazioni nel riconoscimento delle varie regioni di interesse, il classificatore dovrà essere creato in base alle caratteristiche del volto all'interno del video da analizzare. Tuttavia, in casi in cui le caratteristiche somatiche degli utenti non sono eccessivamente diverse, l'utilizzo dello stesso classificatore, come mostrato in Figura 4.12, ha portato ad ottimi risultati.



(a) (b)

Figura 4.12: Risultati ottenuti mediante lo stesso classificatore su due volti aventi differenti caratteristiche di colore.

Il codice utiulizzato per la creazione del classificatore è il seguente:

```
//inizializzazione
ColorDataset* cd = new ColorDataset();
KnnColorClassifier* cc = new KnnColorClassifier;

//Caricamento del file contenente i valori dei
                                    //pixel campione
cd->load("Cris03.dts");

//Creazione e salvataggio del classificatore
cc->fast_build(cd, 5, 5, false);
cc->save_matrix("classifier.kcc");
```

4.3.2 Background subtraction: Estrazione della zona occhi

Come spiegato nel corso dei capitoli precedenti, l'estrazione della regione d'immagine da analizzare con l'algoritmo di blob analysis avviene in due tempi: in un primo momento, l'algoritmo di face detection estrae dal frame in ingresso la regione d'immagine contenente il volto, ottenendo così una prima esclusione dello sfondo. Successivamente, per limitare ulteriormente la zona di analisi, viene ridotta la regione del volto in cui localizzare gli occhi e le sopracciglia. Tale riduzione è stata possibile grazie alle varie proporzioni che il volto umano e i suoi componenti mantengono nel tempo, indipendentemente dalla persona. L'immagine contenente il volto viene dunque suddivisa in otto sezioni orizzontali e per le analisi successive, vengono considerate solamente le sezioni 2, 3, 4 e 5 (Figura 4.1). Il frammento di codice che permette tali operazioni è qui riportato:

```
//Coordinate del volto restituite dall'algoritmo di
CvPoint pt1, pt2;

//Calcolo della regione contenente gli occhi
eyePlace1.x = pt1.x;
```

```
eyePlace1.y = pt1.y - (imgHeight/8) ;
eyePlace2.x = pt2.x;
eyePlace2.y = pt2.y + (3*(imgHeight/8))-10;

//Rappresentazione della zona occhi
cvRectangle(img, eyePlace1, eyePlace2,
            CV_RGB(0,0,255), 2, 8, 0 );
```

4.3.3 Blob Detection

Concluso l'algoritmo di face detection, se il valore restituito è vero (ossia se il volto è stato riconosciuto) viene avviato, sulla regione localizzata, l'algoritmo di blob detection, qui riportato:

```
FindEyes* fe = new FindEyes(true, true);

//Tracking del volto
vbDone = vb->analyzeCurrentFrame(...);

if (vbDone == true)
{
        //BlobDetection
        feDone = fe->analyzeCurrentImage(
                                KnnColorClassifier* cc,
                                PixelMap pm*,
                                CvPoint pt1,
                                CvPoint pt2,
                                IplImage volto*,
                                int frameId);
}
```

I parametri ivi citati hanno il seguente significato:

- **fe:** riferimento alla classe *FindEyes*, all'interno della quale è implementato tutto l'algoritmo di blob analysis (blob detection e blob tracking).

Il suo costruttore riceve in ingresso due parametri booleani che permettono di abilitare e disabilitare alcune funzioni implementate come, ad esempio, la visualizzazione, all'interno del video, della traccia di occhi e sopracciglia e il riconoscimento di queste ultime.

- **KnnColorClassifier*:** viene indicato il classificatore precedentemente creato da utilizzarsi per l'analisi dell'immagine;
- **PixelMap*:** riferimento alla classe *PixelMap* all'interno della quale è implementato l'algoritmo di blob detection sviluppato dal prof. Matteucci;
- **cvPoint pt1, pt2:** punti nel piano che identificano la regione d'immagine contenente il volto;
- **IplImage*:** frame video in analisi;
- **int frameId:** Id del frame in analisi.

L'algoritmo di blob detection utilizzato è implementato all'interno della classe *PixelMap* e necessita di una prima fase di inizializzazione che deve essere eseguita ad ogni analisi, in quanto, le dimensioni dell'immagine da analizzare non sono fissate. Tale caratteristica è evidenziata nella seguente porzione di codice:

```
//Inizializzazione dell'immagine
pm->SetImage(unsigned char* img,
             int width,
             int height,
             ColorClassifier* cc,
             int ppb);
```

dove i parametri citati hanno il seguente significato:

- **unsigned char*:** rappresenta un array monodimensionale (1-D) utilizzato all'interno della struttura *IplImage* al fine di identificare i valori dei canali RGB di ogni pixel dell'immagine. Per tale motivo, le sue dimensioni dipendono da quelle dell'immagine stessa, ovvero da:

$$(img \rightarrow width) \times (img \rightarrow height) \times 3$$

dove, la moltiplicazione per tre è dovuta al numero d'informazioni contenute in ciascun pixel, ovvero i valori dei canali RGB;

- **int width, height:** rappresentano le dimensioni (lunghezza e altezza) dell'immagine da analizzare;

- **ColorClassifier*:** indica il classificatore colore precedentemente creato al fine di localizzare i blob all'interno dell'immagine;

- **int ppb:** pixel per byte.

All'interno della funzione *SetImage(...)* viene inoltre definita la struttura sulla quale l'algoritmo di blob detection opera:

```
//Riferimento all'immagine classificata
unsigned char* mColorImage;

...
...

mColorImage=(unsigned char*)
            malloc
            (width*height*sizeof(unsigned char));
```

Nel precedente frammento di codice, *mColorImage*, rappresenta un'immagine segmentata di dimensioni pari a quelle dell'immagine originale, rappresentabile attraverso un array monodimensionale di char. Ogni pixel dell'immagine in questione è rappresentato da un carattere, utilizzato come etichetta per la segmentazione. Nel presente algoritmo possono essere utilizzate come etichette tutte le lettere dell'alfabeto, ad eccezione della lettera U riservata per le regioni di sfondo, ovvero quelle regioni dell'immagine non classificate (unclassified).

L'analisi vera e propria ha inizio tramite la funzione *BlobGrowing*, qui parzialmente riportata:

```
pm->BlobGrowing(int step,
                int StartWindowX,
                int StartWindowY,
                int EndWindowX,
                int EndWindowY);
```

dove i parametri citati hanno il seguente significato:

- **int step:** indica la modalità con cui analizzare l'immagine. Se impostato al valore 1, esso indica che l'analisi sarà condotta analizzando tutti i pixel, mentre se impostato al valore 2, l'analisi verrà condotta ogni due pixel, e così via;

- **int StartWindowX, StartWindowY, EndWindowX, EndWindowY:** tramite questi parametri è possibile specificare all'algoritmo di blob detection la regione dell'immagine da analizzare. Questo risulta molto utile quando, come in questo caso, si conosce a priori la regione dove i componenti d'interesse sono localizzati, evitando così l'analisi dell'intera immagine; ciò sarà possibile, invece, solo se i parametri in questione saranno impostati al valore -1.

Il codice dell'algoritmo di *BlobGrowing* è riportato interamente nell'appendice B.

Una volta conclusa l'analisi, i risultati ottenuti dal precedente algoritmo vengono memorizzati in un'opportuna struttura dati:

```
map < char, map< int, Blob* > >* GetBlobs()
        {return &mBlobs;};

map <int, Blob*>* GetBlobs(char c)
        {return &mBlobs[c]; };
```

4.3.4 Blob Tracking

L'algoritmo implementato per il blob tracking ha lo scopo di identificare quali regioni, tra quelle localizzate dall'algoritmo di blob detection, rappresentano informazioni utili ai fini dell'applicazione qui presentata. La risorsa d'informazione in ingresso a tale algoritmo è dunque rappresentata dalla lista di blob fornita dalla precedete fase di blob detection. L'analisi è condotta, come spiegato nei paragrafi precedenti, in due fasi differenti, applicate in sequenza: durante la prima fase, la lista di blob identificati nell'immagine viene analizzata con l'obiettivo di riconoscere i blob d'interesse in base ad alcune loro caratteristiche. L'analisi svolta sui vari componenti della lista è la seguente:

```
//Analisi dei blob trovati
for(BlobsItr1=pm->GetBlobs()->begin();
    BlobsItr1!=pm->GetBlobs()->end();
    BlobsItr1++)

  for(BlobsItr2 = BlobsItr1->second.begin();
      BlobsItr2!=BlobsItr1->second.end();
      BlobsItr2++)

   if(BlobsItr2->second->GetValid() &&
      BlobsItr2->second->GetNumPix()>MIN_BLOB_SIZE)
          {
      //Inizializzazione variabili dei filtri
      eyePlace3 = false;
      eyePlace4 = false;
      rect = false;
      clasBlob = false;

     //Coordinate del blob
     pt3.x = BlobsItr2->second->GetMinX();
     pt3.y = BlobsItr2->second->GetMaxY();
     pt4.x = BlobsItr2->second->GetMaxX();
     pt4.y = BlobsItr2->second->GetMinY();

     //Pixel del Blob
     numPixel = BlobsItr2->second->GetNumPix();
```

```
        //Calcolo delle dimensioni del blob
        lengthBlob = (BlobsItr2->second->GetMaxX())
                    -(BlobsItr2->second->GetMinX());
        heightBlob = (BlobsItr2->second->GetMaxY())
                    -(BlobsItr2->second->GetMinY());

        //(1) forma
        rapp = lengthBlob/heightBlob;
        if (rapp > 0.9) rect = true;

        //(2) interno alla zona occhi
        if(((pt3.x >= eyePlace1.x)  &
            (pt3.x <= eyePlace2.x)) &
            ((pt3.y >= eyePlace1.y) &
            (pt3.y <= eyePlace2.y)))
        {
            eyePlace3 = true;
        }


        if(((pt4.x >= eyePlace1.x) &
           (pt4.x <= eyePlace2.x)) &
           ((pt4.y >= eyePlace1.y) &
           (pt4.y <= eyePlace2.y)))
        {
            eyePlace4 = true;
        }

        //(3) Classe del blob
        classificationBlob = BlobsItr1->first;
        if( classificationBlob == clas )
        {
                clasBlob = true;

        }
}
```

Come si può notare dal precedente frammento di codice, i controlli eseguiti sono quattro:

1. all'interno del ciclo *if* iniziale, viene imposto che l'analisi venga compiuta solamente sui blob aventi dimensioni maggiori della costante *MIN_BLOB_SIZE* precedentemente impostata:

2. la seconda verifica è condotta basandosi sulla forma del blob analizzato. Tale verifica impone che il rapporto tra la lunghezza e l'altezza del rettangolo rappresentante la regione in analisi, debba risultare maggiore di un valore di soglia, impostato in questo caso a 0.9. Tale rapporto è stato estrapolato dai risultati effettuati su varie immagini di test, al fine di ottenere buoni risultati di riconoscimento, anche in casi in cui il volto assume pose particolari (ad esempio, il volto girato verso destra o sinistra). In alcune condizioni infatti, può capitare che la regione contenente l'occhio non venga rappresentata tramite un rettangolo, in quanto, la rotazione della testa potrebbe nascondere parte del volto. In questi casi la regione riconosciuta può essere approssimabile ad un quadrato;

3. un altro parametro di verifica è rappresentato dalla classe associata, in fase di analisi, al blob. Le regioni d'interesse sono infatti contraddistinte dall'etichetta K, indicante le zone più scure localizzate all'interno dell'immagine;

4. viene effettuata infine un'analisi con l'obiettivo di verificare che il blob in questione, risulti localizzato all'interno della regione del volto contente occhi e sopracciglia (Figura 4.1(b)).

Quando il componente in analisi supera i quattro controlli sopra presentati, questo viene salvato all'interno di una lista.

```
//Se tutti (1) (2) (3) sono rispettati
if(eyePlace3 == true &&
   eyePlace4 == true &&
   rect == true &&
   clasBlob == true)
{
```

```
        //Memorizzazione delle informazioni
        listaBlobs [numBlobFrame].a.x = pt3.x;
        listaBlobs [numBlobFrame].a.y = pt3.y;
        listaBlobs [numBlobFrame].b.x = pt4.x;
        listaBlobs [numBlobFrame].b.y = pt4.y;
        listaBlobs [numBlobFrame].numPixel =
                BlobsItr2->second->GetNumPix();
}
```

La lista così creata rappresenta il parametro di ingresso per la seconda parte dell'algoritmo di blob tracking, la quale, ha il compito di analizzare le regioni d'interesse in essa contenute, determinando cosa rappresentano. Tale analisi viene eseguita ricorrendo alla seguente funzione:

```
BlobsDetection(int numBlobs,
               BlobInfo listaBlobs [],
               int mediana);
```

dove i parametri citati hanno il seguente significato:

- **int numBlob:** rappresenta il numero di blob identificati nel frame in analisi;
- **BlobInfo listaBlob[]:** rappresenta la struttura dati contenente tutte le informazioni relative ai blob localizzati;
- **int mediana:** linea che divide verticalmente il volto in due parti: parte destra e sinistra.

La lista ottenuta dalla prima parte dell'algoritmo viene analizzata una prima volta per dividere i blob in essa contenuti, in base alla loro posizione nell'immagine. Vengono dunque create due nuove liste: la prima contenete i blob localizzati nella parte destra del volto e la seconda contenente quelli localizzati nella parte sinistra.

```
for (i=0; i<numBlob; i++)
{
    //sono a sinistra
    if (listaBlobs[i].a.x < mediana)
    {
                listaSx[sx] = listaBlobs[i];
                sx++;
    }

        //o sono a destra
        else
        {
                listaDx[dx] = listaBlobs[i];
                dx++;
        }
}
```

L'analisi delle due liste viene ora condotta indipendentemente l'una dall'altra. Prendendo come esempio la sola lista contenete i blob localizzati sulla destra del volto, sono state implementate due tipologie di controlli, il primo dei quali è rappresentato dal seguente frammento di codice:

```
if (dx == 1)
{
    //Hp. blob dell'occhio
    temp[3] = listaDx[0];

   //Confronto con la posizione dell'occhio
   //dx nei frame precedenti
   if ( ConfrontaPrec(3) == 'K') //corrisponde
   {
      temp[3].classe = 'K';
      cvRectangle(img, temp[3].a,
                  temp[3].b,
                  CV_RGB(255,0,0), 2, 8, 0 );
   }
```

```
    if ( ConfrontaPrec(3) == 'S')  //non corrisponde
    {
        temp[1] = listaDx[0];
        temp[1].classe = 'S';
        cvRectangle(img, temp[1].a,
                    temp[1].b,
                    CV_RGB(0,255,0), 2, 8, 0 );



        //Cancellazione della previsione sbagliata
        temp[3].a.x=-1;
        temp[3].a.y=-1;
        temp[3].b.x=-1;
        temp[3].b.y=-1;
        temp[3].classe = 'P';
    }
}
```

Tale controllo permette di identificare la regione d'interesse, nel caso in cui essa sia l'unica contenuta nella lista analizzata. Il controllo eseguito dal precedente frammento di codice, infatti, è basato sul confronto tra il blob in analisi ed il suo corrispondente localizzato all'interno del frame precedente. Il secondo controllo, invece, è riservato alle sole liste contenenti più di una regione d'interesse ed è riportato nel seguente frammento di codice:

```
if (dx > 1)
{
    temp[3] = listaDx[0];

    //localizzazione del blob piu' basso nella
    //parte destra
    for (j=0; j<dx; j++)
    if(temp[3].a.y <= listaDx[j].a.y)
   {
```

```
      temp[3] = listaDx[j];
      temp[3].classe = 'K';        //e' occhio
    }

    else { j++; }
}

cvRectangle(img, temp[3].a,
                    temp[3].b,
                   CV_RGB(255,0,0), 2, 8, 0 );
...
...
if (dx > 1)
{
    temp[1] = listaDx[0];
    //localizzazione del blob piu' alto
    //nella parte destra
    for (j=0; j<dx; j++)
    {
      if((temp[1].a.y >= listaDx[j].a.y) &&
      (listaDx[j].a.y >= 0))
      {
        temp[1] = listaDx[j];
        temp[1].classe = 'S'; //e' sopracciglio
      }

      else { j++; }
    }

cvRectangle(img, temp[1].a,
                    temp[1].b,
                   CV_RGB(0,255,0), 2, 8, 0 );

}
```

Come visibile dal codice, questa volta non sono stati implementati controlli tra i blob analizzati e i corrispettivi localizzati al frame precedente. Questo perchè, nei risultati ottenuti dai test, non sono mai state individuate, in

questa fase di analisi, regioni d'interesse al di sopra delle sopracciglia o al di sotto degli occhi. è stato riscontrato, inoltre, che, in particolar modo per la localizzazione delle sopracciglia, il confronto con i blob relativi ai frame precedenti è fonte di mancati riconoscimenti con conseguente perdita di informazioni.

4.3.5 Risultati ottenuti e salvataggio dei dati

Verranno ora riportati i riultati ottenuti attraverso l'applicazione dell'algoritmo di blob analysis ai filmati registrati. La prima sequenza di immagini (Figura 4.13, Figura 4.14, Figura 4.15, Figura 4.16) mostra come l'algoritmo implementato localizzi correttamente la posizione di occhi e sopracciglia all'interno dell'immagine, indipendentemente dalla posizione del volto e dalla sua inclinazione. Il rettangolo più esterno di colore verde rappresenta la regione d'immagine restituita dall'algoritmo di face detection all'interno della quale, viene condotta la blob detection. Il rettangolo più interno, rappresentato in blu, rappresenta la regione d'immagine utilizzata per la ricerca dei blob di interesse.

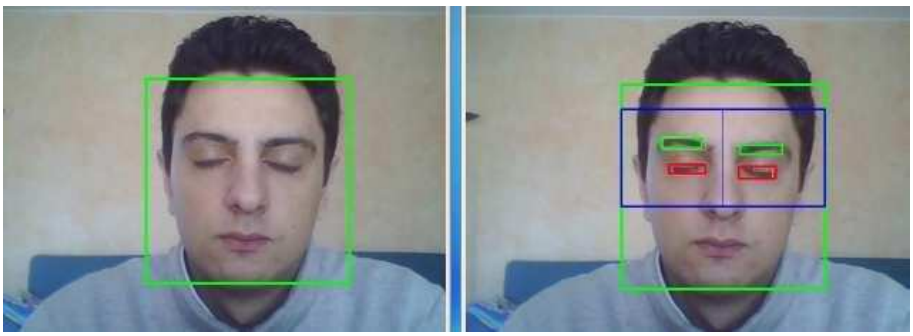


Figura 4.13: Risultati algoritmo blob Analysis immagine del volto frontale con occhi chiusi.

Figura 4.14: Risultati algoritmo blob Analysis immagine del volto rivolto a sinistra.

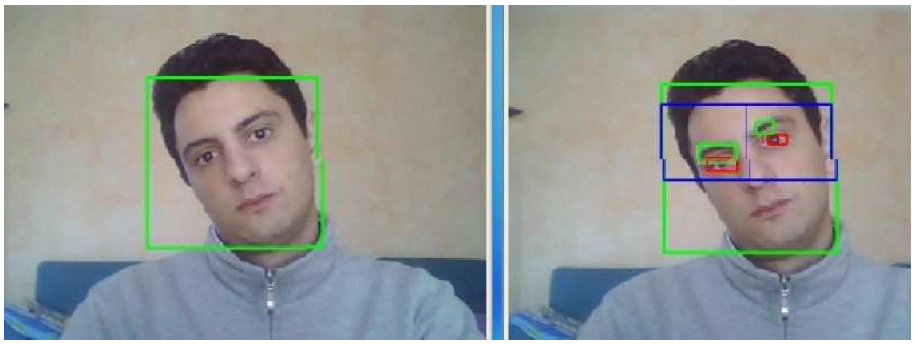


Figura 4.15: Risultati algoritmo blob Analysis immagine del volto inclinato a destra.

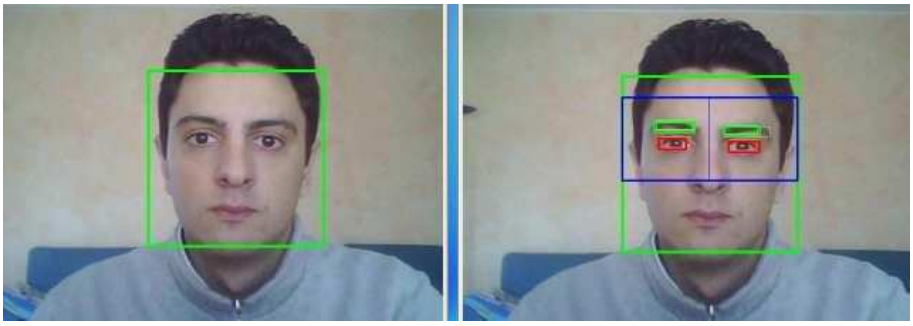


Figura 4.16: Risultati algoritmo blob Analysis immagine del volto frontale.

La sequenza successiva, mostra i risultati ottenuti analizzando, con lo stesso classificatore, un video contenente un volto avente caratteristiche di colore nettamente differenti dal precedente. A differenza di quest'ultimo, le sopracciglia non vengono qui localizzate, fatta eccezione della Figura **??**. Anche in questo caso la localizzazione degli occhi avviene correttamente nelle differenti posizioni ed inclinazioni assunte dal volto.

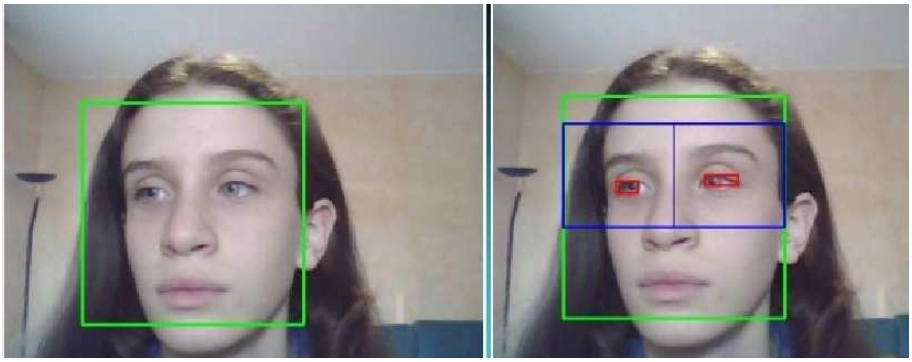


Figura 4.17: Risultati algoritmo blob Analysis immagine del volto rivolto a destra.

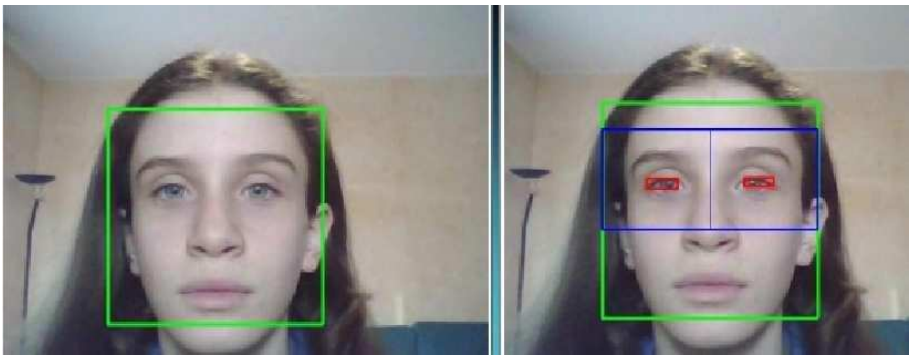


Figura 4.18: Risultati algoritmo blob Analysis immagine del volto frontale.

Capitolo 5

Verifiche sperimentali e conclusioni

"Sonny: Mio padre ha provato ad insegnarmi le emozioni umane. Sono...difficili"

Io Robot

Al termine della realizzazione del sistema complessivo, analizzato nei capitoli precedenti, sono state condotte alcune verifiche sperimentali con lo scopo di misurare le prestazioni e l'accuratezza dell'algoritmo proposto.

5.1 Descrizione del sistema di acquisizione dati

I filmati sono stati realizzati con due webcam differenti aventi una risoluzione di 640×480 ed un frame rate variabile dai 25 ai 30 fps. In tutti gli esperimenti fatti, si richiede che il soggetto sia disposto frontalmente all'obiettivo e ad una distanza utile per l'inquadratura di tutto il volto. Il filmato acquisito viene poi analizzato frame per frame ottenendo, come risultato dell'algoritmo qui implememntato, un filmato di circa 10/15 fps con in evidenza le caratteristiche localizzate. I filmati sono stati realizzati in ambienti luminosi con luci statiche per evitare che la presenza di ombre possa disturbare l'algoritmo.

5.2 Classificatori utilizzati

Sono stati utilizzati due classificatori differenti a seconda dalle caratteristiche dell'utente nel video. I classificatori sono stati realizzati entrambi attraverso

la selezione di 10 frame dal video da analizzare, dai quali sono state poi estratte alcune caratteristiche di colore appartenente alla pelle e alle pupille.

5.3 Test effettuati

Verranno ora mostrati e commentati alcuni risultati ottenuti sui video ed sulle immagini registrate tramite webcam. Per ogni test verrà proposta l'immagine originale del frame, quella del volto e il risultato finale con tutte le feature localizzate.

5.3.1 Test 1

Il primo test è stato eseguito su un filmato che ritrae il volto in diverse pose ed orientamenti.

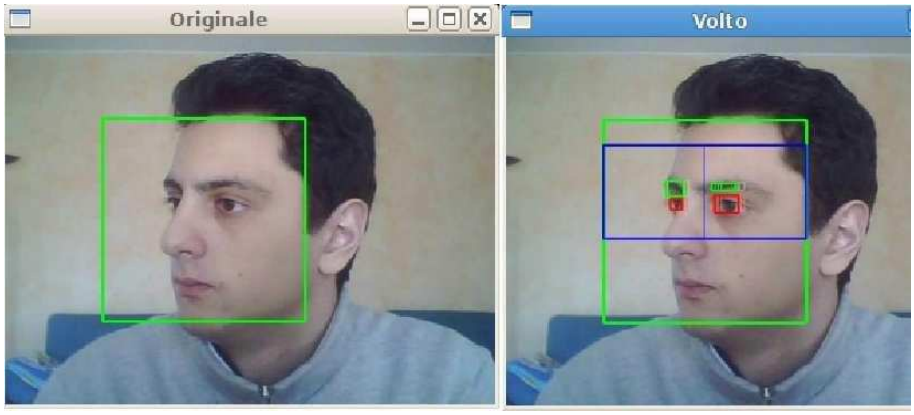


Figura 5.1: Risultati test 1: Volto orientato verso destra con occhi e sopracciglia localizzati correttamente

I risultati ottenuti sono stati soddisfacenti sia in termini di localizzazione di occhi e sopracciglia, sia per quanto riguarda l'estrazione del volto dall'immagine.

5.3.2 Test 2

Il secondo test consiste nell'analisi di un filmato, simile al precedente, nel quale l'utente ha caratteristiche somatiche molto chiare come occhi azzurri e sopracciglia e capelli biondi.

Anche in questo caso i risultati sono stati soddisfacenti sia in termini di estrazione del volto dall'immagine origianle, che in termini di localizzazione degli occhi. Le sopracciglia, invece, come atteso, non sono state riconosciute in quanto caratterizzate da una colorazione molto chiara.

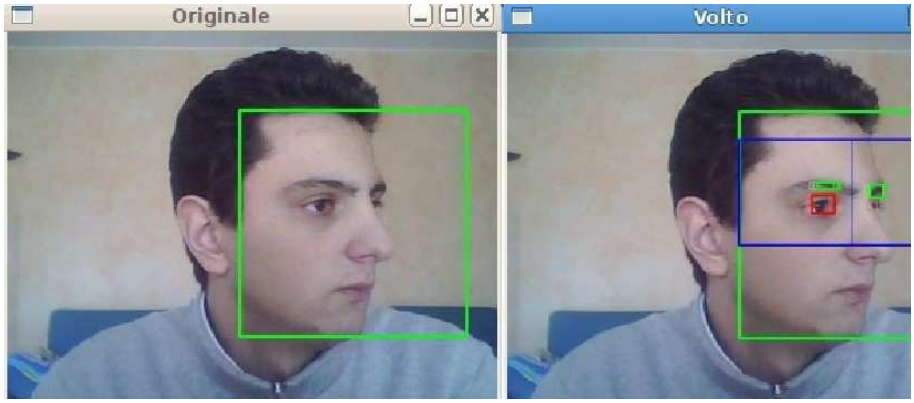


Figura 5.2: Risultati test 1: Volto orientato verso sinistra con sopracciglia localizzate correttamente, non viene invece rilevato l'occhio sinistro

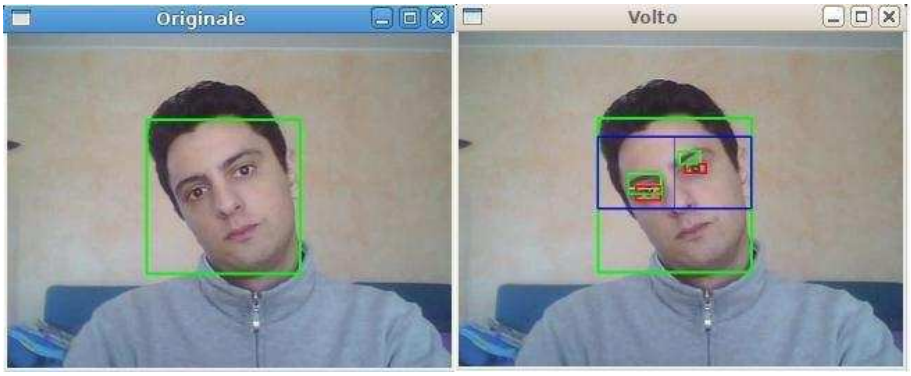


Figura 5.3: Risultati test 1: Volto inclinato verso destra con sopracciglia localizzate correttamente, non viene invece rilevato l'occhio sinistro

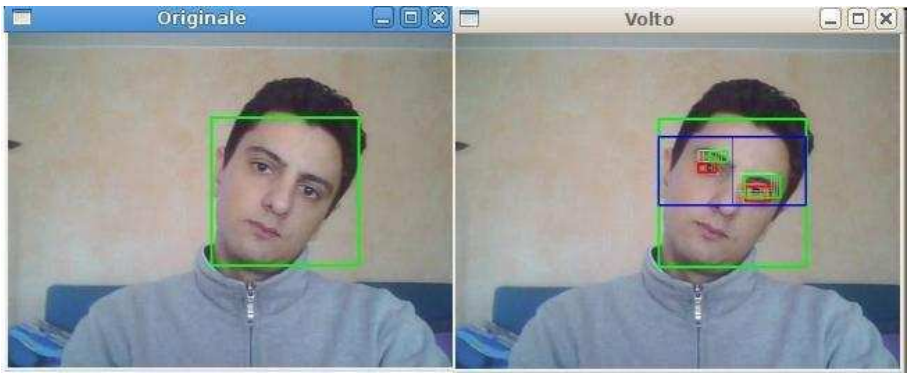


Figura 5.4: Risultati test 1: Volto inclinato verso sinistra con occhi e sopracciglia localizzati correttamente.

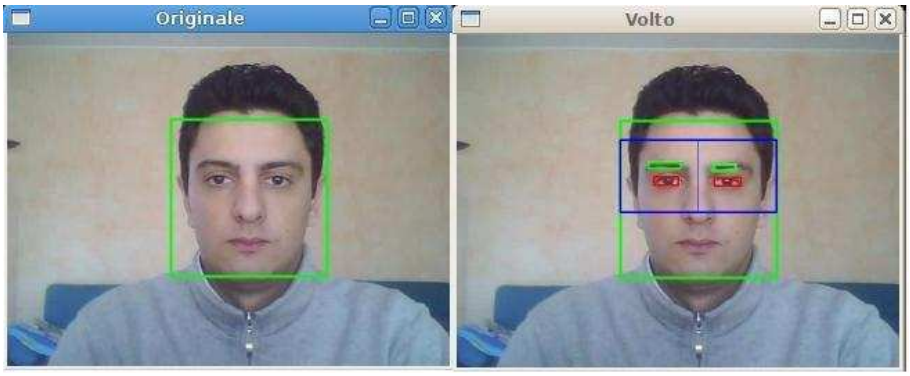


Figura 5.5: Risultati test 1: Immagine frontale del volto con occhi e sopracciglia localizzati correttamente.

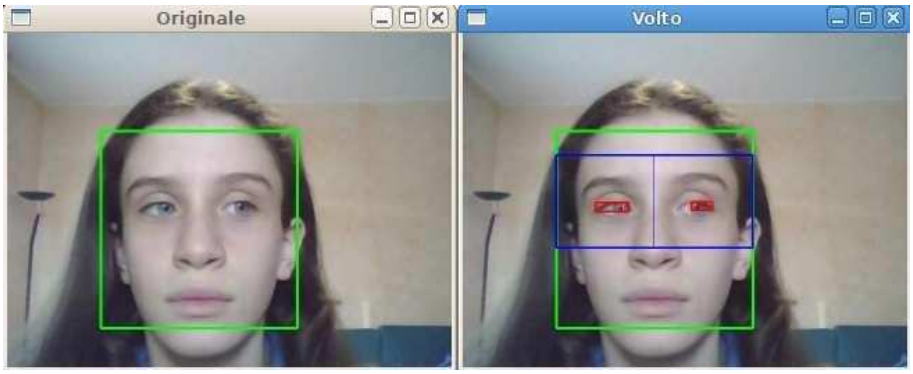


Figura 5.6: Risultati test 2: Immagine frontale del volto con occhi localizzati correttamente.

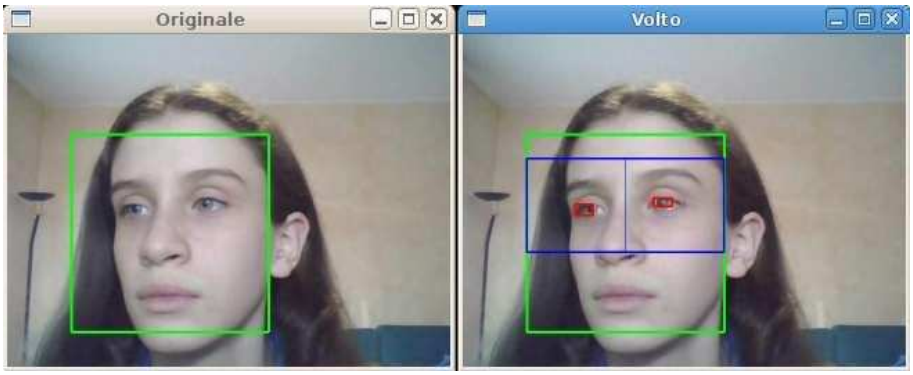


Figura 5.7: Risultati test 2: Immagine del volto orientato verso destra con occhi localizzati correttamente.

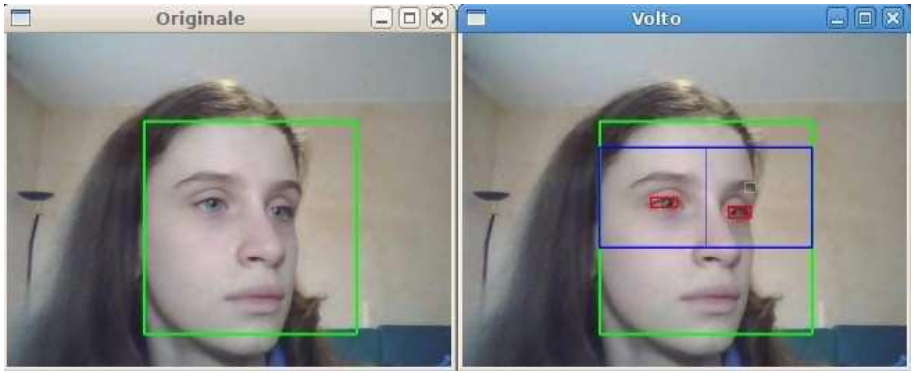


Figura 5.8: Risultati test 2: Immagine del volto orientato verso sinistra con occhi localizzati correttamente.

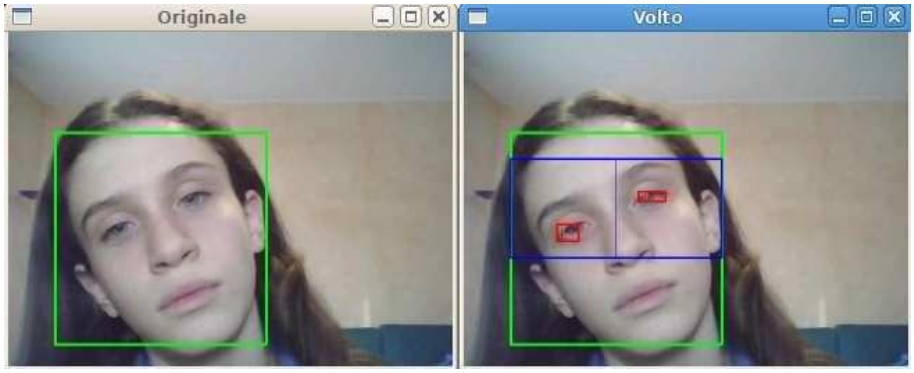


Figura 5.9: Risultati test 2: Immagine del volto inclinato verso destra con occhi localizzati correttamente.

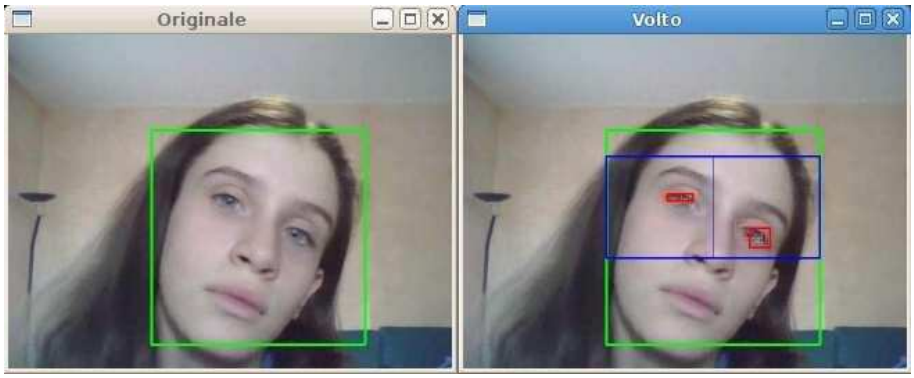


Figura 5.10: Risultati test 2: Immagine del volto inclinato verso sinistra con occhi localizzati correttamente.

5.3.3 Test 3

Il terzo test è stato condotto su una persona con occhiali, per testare la robustezza dell'algoritmo anche in presenza di riflessi nella zona degli occhi.

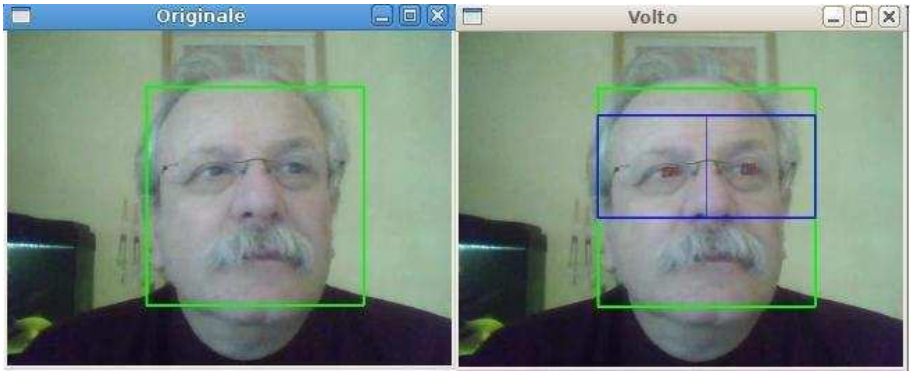


Figura 5.11: Risultati test 3: Immagine del volto orientato verso sinistra con occhi localizzati correttamente.

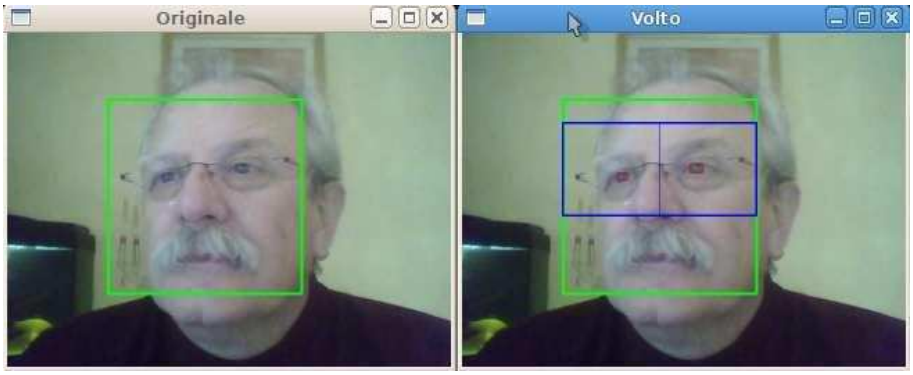


Figura 5.12: Risultati test 3: Immagine del volto orientato verso destra con occhi localizzati correttamente.

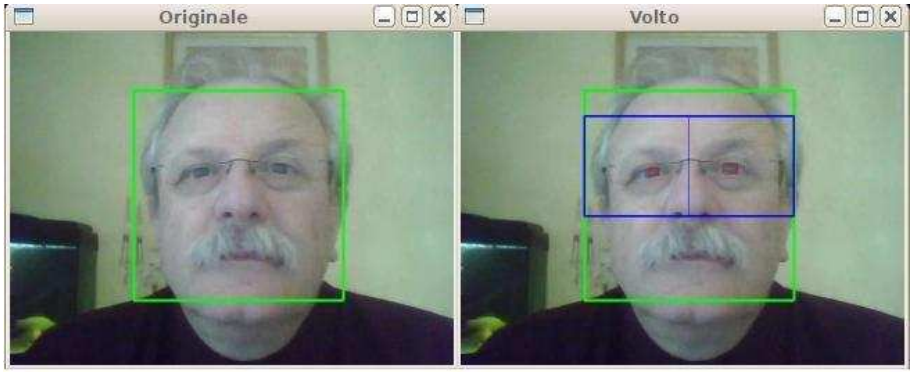


Figura 5.13: Risultati test 3: Immagine frontale del volto con occhi localizzati correttamente.

Figura 5.14: Risultati test 3: Immagine del volto inclinato a sinistra con localizzazione errata dell'occhio sinistro.

I risultati ottenuti mostrano che la presenza della lente non è causa di disturbo. Tuttavia, in particolari condizioni, esse generano un riflesso che porta ad una errata o mancata localizzazione degli occhi.

5.4 Conclusioni

L'analisi di filmati contenenti volti di persone è da sempre stata oggetto di attenzione di molti. Nonostante questo sia un'argomento di studio da più di 30' anni, ancora oggi non sono disponibili sul mercato (con costi ragionevoli) dispositivi che utilizzino tali tecnologie.

Lo scopo principale di questa tesi è stato quello di ricercare, studiare ed implementare un sistema di analisi del volto real time, avente buone prestazioni e buona efficienza nell'analisi. Non è stato dedicato tempo, in questa sede, ad un'attenta analisi delle prestazioni degli algoritmi implementati, in

quanto si è sempre lavorato su computer desktop e portatili ad alte prestazioni. Questa rimane una porta aperta per eventuali analisi future atte, ad esempio, all'ottimizzazione del codice per la sua esecuzione su dispositivi portatili, nei quali, la potenza di calcolo è limitata anche per questioni energetiche.

I video realizzati, inoltre, sono stati registrati in ambienti luminosi con una luce statica e successivamente analizzati attraverso l'uso di classificatori di colore creati ad hoc per la localizzazione di determinate caratteristiche. Un possibile sviluppo futuro potrebbe consistere nell'implementazione di classificatori dinamici che si adattino alle condizioni di luci ed ombre presenti nel filmato in analisi.

Tale progetto si propone quindi come possibile applicazione per i sistemi atti all'analisi del volto ed aventi l'obiettivo di identificare i diversi stati emotivi. In tali sistemi, infatti, l'analisi dei movimenti del volto e dei suoi componenti costituiscono informazioni fondamentali per l'identificazione e lo studio di alcuni comportamenti dell'utente dovuti ad una particolare condizione emotiva.

> *"Andrew: Come robot avrei potuto vivere per sempre, ma dico a tutti voi oggi, che preferisco morire come uomo, che vivere per tutta l'eternità come macchina."*
>
> L'uomo bicentenario

Bibliografia

[1] N. Ishikawa A. Lijima M. Haida. Head mounted goggle system with liquid crystal display for evaluation of eye tracking functions on neurological disease patients. *Proceedings of the 25th Annual Int. Conf. of the IEEE*, 2003.

[2] Shumeet Baluja ad Dean Pomerleau. Non intrusive gaze tracking using neural networks. *School of Computer Science*, 1994.

[3] D.H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 1981.

[4] Amit Bandopadhay Bernard D. Shaviv A. E. Kaufman. *Research Frontier in Virtual Reality Workshop Proceedings.* 1993.

[5] C. von der Malsburg B.S. Manjunath, R. Chellappa. A feature-based approach to face recognition. *Proc. IEEE Conf. Computer Vision Pattern Recognition*, 1992.

[6] Psychol. Bull. *P. Ekman Strong evidence for universals in facial expression.* 1994.

[7] D. Koons C. Marimoto. Keeping an eye for hci. 1999.

[8] T. Poggio C. Papageorgiou, M. Oren. A general framework for object detection. *International conference on Computer Vision*, 1998.

[9] Hui Wen Jeng Cheng Yuan Tang, Jeng Horng Chang. Learning the facial model for face detection using genetic algorithms. *Department of Information Management*, 2003.

[10] L. Davis. Handbook of genetic algorithms. *Van Nostrand, New York*, 1991.

[11] Myung Jin Chung Dong Hyun Yoo. Non intrusive eye gaze estimation without knowledge of eye pose. 1975.

[12] Winfield D. Dongheng Li. A hybrid algorithm for video basedeye tracking combining feature-based and model-based approaches. *Computer Vision and Pattern Recognition*, 2005.

[13] G. Yang e T.S.Huang. Human face detection in complex background. *Pattern Recognition*, 27, 1994.

[14] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application on boosting. *Computational learning theory*, 1995.

[15] Adrian Kaehler Gary Bradski. *Learning OpenCV, Computer Vision with the OpenCV Library.* O'Reilly, 2008.

[16] T.Kanade H. Rowley, S. Baluja. Neural network-based face detection. *IEEE Patt. Anal.Mach.Intell.*, 1998.

[17] H. Takahashi K. Asai, N. Osawa. Eye aker pointer in immersive projection display. *Virtual Reality 2000*, 2000.

[18] M. Betke K. Grauman. Communication via eye blink - detection and duration in real time. *Computer Vision and Pattern Recognition*, 2002.

[19] T.Poggio K. Sung. Example-based learning for view-based face detection. *IEEE Patt. Anal.Mach.Intell.*, 1998.

[20] T. Kanade. Pictures processing by computer complex and recognition of human faces. *PhD thesis, Kyoto Univ.*, 1973.

[21] M. Betke K.Grauman. Communication via eye blink and eyebrown raise. *Universal Acces in the Information Society*, 2001.

[22] Seung Yup Lee Keun Young Kim and Hee Chan Kim. A wireless measurement system for three-dimensional ocular movement using the magnetic contact lens sensing technique. 2004.

[23] C. Kotropoulus and I. Pitas. Rule-based face detection in frontal views. *Proc. Int'l Conf. Acoustics, Speech and Signal Processing*, 4, 1997.

[24] M.Stark L. Clark. Automatic control, ieee transictions. 20, 1975.

[25] Young Laurence R. and David Sheena. *Behaviour Research Methods and Instrumentation.* 1975.

[26] Margrit Betke Michael Chau. Real time eye tracking and blick detection with usb cameras. *Computer Science Departement, Boston University*, 2005.

[27] Narendra Ahuja Ming-Hsuan Yang, David J. Kriegman. Detecting faces in images: A survey. *IEE Transactions on pattern Analysis and machines intelligence*, 24, 2002.

[28] Michael J. Jones Paul Viola. Robust real-time object detection. 2001.

[29] S. Baluja D. Pomerleau. Neural information processing system. 1994.

[30] S.D. Shapiro. Feature space transforms for curve detection. *Pattern Recognition*, 1978.

[31] S.A. Sirohey. Human face segmentation and identification. *Univ. of Maryland*, 1993.

[32] S.M. Smith and J.M. Brady. Susan. a new approach to low level image processing. *International Journal of Computer Vision*, 1997.

[33] Wikipedia. *Affective Computing.*

[34] Wikipedia. *Face emotion Recognition.*

[35] Wikipedia. *Face Recognition.*

[36] Wikipedia. *Human eye and IR.*

[37] Patrick Haffner Y. Patrice, Lon Bottou. A fast convolution algorithm for signal processing and neural networks. *Advances in Neural Information Process Systems*, 1999.

[38] Laurence R. Young and David Sheena. Survey of eye movements recording methods. behaviour research methods and instrumentation. 1975.