Tesi di laurea

# Analisi Automatica di Identikit per il Riconoscimento di Identità

Laureando: Niero Andrea

Relatore: Ferrari Carlo

**Corso di laurea Magistrale**
**in Ingegneria Informatica**

*Anno accademico*
*2011/2012*

# Indice

# Capitolo 1

# Introduzione

L'applicazione di tecniche di *biometria* è una risposta alla sempre più diffusa esigenza di automatizzare le procedure di *riconoscimento di identità* e di adeguarle alle diverse situzioni in cui esse devono essere inserite. In questo campo, è stata presa in considerazione una delle caratteristiche che ha grande rilevanza negli studi recenti: il *volto*.

L'obiettivo è quello di proporre un nuovo approccio al problema del riconoscimento di identità attraverso l'analisi di un'immagine del volto, che sfrutti uno dei metodi usati dall'uomo per la ricostruzione dell'identità di un soggetto, ovvero l'*identikit*.

L'idea di partenza riprende la procedura usata per la costruzione di un identikit: si prende in considerazione una componente del volto alla volta e si associano dei modelli a partire da un catalogo che contiene un insieme tale da descrivere le diverse tipologie morfologiche per tale componente. La scelta cade su quegli elementi che risultano più somiglianti alla descrizione del soggetto, i quali vengono messi assieme fino a formare un volto completo.

La proposta è strutturata per far fronte a un problema di diverso tipo, ovvero di *verifica* dell'identità di un utente che dichiara di essere un certo individuo e fornisce un'immagine del volto per provarlo. Il sistema che viene proposto per questa procedura, all'immagine associa un identikit i cui componenti sono scelti attraverso uno specifico criterio di somiglianza rispetto al volto nell'immagine. Questa descrizione viene poi confrontata con l'identikit relativo all'identità dichiarata, dato che il sistema ha a disposizione.

Con questo approccio, dunque, si vuole arrivare a calcolare un grado di somiglianza tra due volti, che possa valutare se essi rappresentano lo stesso soggetto. Questo valore viene elaborato calcolando in una prima fase il grado di somiglianza di ciascuna componente dei due volti, rispetto ai modelli forniti dal catalogo per la costruzione dell'identikit. Questi indicatori di somiglianza, associati ai diversi elementi del catalogo inseriti nell'identikit, vengono poi confrontati tra loro e messi assieme per arrivare alla decisione finale.

Il progetto del sistema che viene presentato come prima realizzazione di questo approccio, vede la definizione di tre componenti principali:

- un modello geometrico di descrizione delle componenti del volto che permetta di confrontare l'immagine del volto e gli elementi di catalogo che andaranno a formare l'identikit;

- un modello di definizione dell'identikit a partire dai confronti tra immagine e elementi di catalogo;

- un metodo di confronto tra identikit relativi a due volti diversi, con l'obiettivo di stabilire se appartengono allo stesso soggetto.

Una delle proprietà che si vuole mantenere nella definizione delle tre componenti appena elencate, è l'indipendenza di ciascuna di esse dalle scelte progettuali specifiche fatte per le altre. In questo modo è possibile apportare futuri miglioramenti, senza stravolgere la struttura generale del sistema.

Dopo una fase di progettazione, il lavoro ha portato a una prima implementazione del sistema di riconoscimento in un software realizzato nel linguaggio di programmazione *C++*, che verrà descritto nel dettaglio e per il quale sono stati eseguiti dei test di validazione, di cui saranno riportati i dati.

Infine verranno discusse le potenzialità di miglioramento del software, che sono intesi come quegli aspetti che non sono stati trattati in dettaglio in questo lavoro ma che danno un'idea degli sviluppi futuri che può avere il modello di sistema proposto.

# Capitolo 2

# Stato dell'arte

Il problema che viene affrontato è quello del riconoscimento di identità attraverso il volto (*face recognition*). Si entra quindi nel campo più generale della *biometria* che analizza le caratteristiche fisiche e comportamentali della persona con lo scopo del riconoscimento di identità.

Partendo da un uso quasi esclusivamente in ambito criminale, le tecniche biometriche si sono diffuse anche in un contesto civile che spazia dal controllo degli accessi al mondo del divertimento.

Oltre al volto, le principali caratteristiche fisiche che vengono utilizzate nei sistemi biometrici sono le impronte digitali, l'iride, la retina, la mano e recentemente anche l'orecchio. Per quanto riguarda le caratteristiche comportamentali, invece, vengono analizzate voce, grafia, stile di battitura a tastiera e andamento. Ciascuna di esse viene presa in considerazione nel campo biometrico per una caratteristica peculiaria, che rende efficace il riconoscimento in determinate condizioni.

Impronte digitali, iride e retina sono descrizioni molto accurate che permettono di distinguere soggetti in una grande popolazione. Le procedure di rilevamento associate, però, risultano invadenti rispetto all'utente che deve essere identificato, soprattutto se il riconoscimento deve essere fatto su un cosiddetto soggetto "non collaborativo". La mano, più semplice da rilevare e con una procedura meno invadente, risulta adatta in un sistema limitato a pochi utenti. L'orecchio, invece, è studiato in quanto è una componente che risulta essere poco soggetta a cambiamenti nel tempo.

Le caratteristiche comportamentali soffrono di problemi di accuratezza e del fatto che nel tempo possono variare in base allo stato psico-fisico del soggetto in questione. Esse sono utilizzate per situazioni in cui è disponibile tale dato, ad esempio in sistemi di analisi di conversazioni registrate per la voce oppure in applicazioni per personal-computer in cui vi è l'uso della tastiera per lo stile di battitura.

Il volto ha avuto una certa rilevanza in questi ultimi anni, in quanto è un compromesso tra affidabilità e accettazione da parte dell'utente. Infatti richiede una collaborazione minima a chi deve essere identificato e certamente molto meno intrusiva di un rilevatore di iride o retina. Pertanto il riconoscimento del volto risulta avere un grande bacino di applicazione.

Per quanto riguarda l'accuratezza, questo campo non si può mettere allo stesso livello dell'impronta digitale, anche se sono stati ottenuti risultati molto buoni a livello prestazionale.

E' per questi motivi che l'identificazione automatica attraverso l'analisi del volto risulta una sfida affascinante alla naturale capacità umana di riconoscere un individuo.

## 2.1 Sistemi di riconoscimento

I problemi di riconoscimento di identità nel campo della biometria, possono essere generalizzati e schematizzati indipendentemente dalla caratteristica biometrica in questione e dal particolare approccio utilizzato per arrivare alla soluzione del problema.

In primo luogo è necessario definire il tipo di problema da affrontare e di conseguenza quali sono i dati in input e la risposta che si aspetta di ricevere in output. Successivamente è possibile definire una struttura del sistema a fasi sequenziali.

### Verifica vs Identificazione

Si possono definire due tipi principali di problema:

**Verifica:** si ha nel caso in cui viene rilevata la caratteristica biometrica di un soggetto che dichiara una certa identità e il sistema deve essere in grado di dare una risposta alla richiesta di autenticazione dell'identità, più precisamente deve verificare se la caratteristica biometrica corrisponde al soggetto identificato dalla dichiarazione. In questo caso i dati in ingresso al problema sono un soggetto e un'identità dichiarata, il dato in uscita è una risposta di tipo *sì/no*.

**Identificazione:** è un problema che richiede di determinare l'identità di un soggetto sconosciuto, a partire da un insieme di utenti per i quali si conosce la descrizione biometrica. In questo caso l'input è il soggetto da identificare, mentre l'output è l'identità attribuita oppure una risposta di riconoscimento mancato, nel caso in cui il soggetto non venga associato ad alcuna identità.

La differenza tra le due definizioni può essere considerata anche come problema di confronto tra descrizioni biometriche di soggetti. Nella *verifica* il confronto che avviene è di tipo $1:1$ (*one-to-one matching*), mentra il problema di *identificazione* propone un confornto $1:n$ (*one-to-many matching*).

### Struttura del sistema

Il sistema può essere pensato come un insieme di operazioni suddivise in tre fasi principali, sequenziali:

1. rilevazione dei dati relativi alla caratteristica biometrica del soggetto in questione;

2. estrazione delle informazioni necessarie per la descrizione della caratteristica biometrica, ai fini del riconoscimento;

3. confronto tra la descrizione del soggetto e le descrizioni che il sistema ha a disposizione.

La prima fase prevede in generale l'acquisizione di un dato grezzo, a partire dal quale deve essere rilevata la componente biometrica che deve essere analizzata (fase indicata spesso con il termine *detection*). Successivamente il dato necessita di un'ulteriore elaborazione per costruire un modello adatto al confronto (procedimento descritto spesso come features extraction). Infine avviene il confronto vero e proprio (*matching*) il quale è concentrato su una coppia specifica di descrizioni nel caso di un problema di verifica, mentre per un'identificazione è composto da una serie di confronti.

### Analisi dei risultati

Per la valutazione di un sistema di riconoscimento, vengono utilizzati i due indici di errore più significativi:

**FRR (False Reject Rate):** percentuale di casi in cui le descrizioni a confronto si riferiscono allo stesso soggetto ma il confronto dà esito negativo.

**FAR (False Accept Rate):** percentuale di casi in cui le descrizioni a confronto non si riferiscono realmente allo stesso soggetto ma il confronto dà esito positivo.

Questi due indici variano a seconda di come sono impostati i parametri in gioco nelle diverse fasi del sistema. In particolare, quando è in gioco un valore di soglia che distingue i soggetti accettati da quelli non accettati, si possono calcolare le due curve relative ai due indici, al variare della soglia. Le due curve hanno un andamento opposto, ovvero al crescere del valore di soglia, se un indice cresce, l'altro diminuisce.

Viene definito come *EER (Equal Error Rate)*, il valore dei due indici nella situazione in cui essi si equivalgono, ovvero la percentuale di errore in cui le due curve relative ai due indici si incontrano. Il valore di una soglia nel punto in cui viene individuato l'*EER* può essere considerato la scelta più ragionevole. In generale la combinazione di parametri che minimizzano tale errore si può considerare la più performante e dà un'indicazione della prestazione del sistema.

## 2.2 Il riconoscimento attraverso il volto

Il riconoscimento del volto ha una diffusione notevole in quanto viene utilizzato in numerosi campi di applicazione [3] [5]:

- sicurezza e controllo di accessi;
- sorveglianza;
- autenticazione attraverso "smart cards";
- ricerca su database di immagini e video;
- divertimento, applicazioni multimediali;
- divertimento;

- lotta al crimine, indagini.

Questo grande bacino di diffusione, ha fatto sì che questo settore della biometria abbia avuto attenzioni particolari da parte della ricerca, coinvolgendo aree disciplinari diverse: elaborazione di immagini, pattern recognition, intelligenza artificiale, antropometria, psicologia.

Un sistema suddiviso nelle tre fasi, come descritto in precedenza, può assumere contorni più dettagliati se si restringe il campo al riconoscimento del volto. Infatti è possibile ridefinire le tre fasi secondo la seguente impostazione [3]:

1. Rilevamento del volto.
2. Elaborazione dell'immagine del volto per la costruzione di un modello.
3. Confronto tra i modelli.

La prima fase richiede tecniche di elaborazione dell'immagine e pattern recognition che effettuino un'operazione di *object detection* per l'individuazione del volto all'interno dell'immagine data in input al sistema. In questo campo esistono diverse soluzioni, descritte in [7] [8] [9] [10] [11] [12].

Anche la seconda coinvolge tecniche di elaborazione di immagine che permettano di definire un'insieme di parametri descrittivi del volto con l'obiettivo di effettuare il riconoscimento attraverso il confronto finale. Queste due fasi si possono considerare il punto cruciale dell'intero riconoscimento e in letteratura sono presenti molti approcci diversi al problema.

## Approcci per le fasi di riconoscimento automatico del volto

In letteratura, i metodi vengono comunemente classificati in due categorie principali [4] [17]:

**Approccio olistico:** utilizza l'intera regione del volto come come dato su cui effettuare il riconoscimento.

**Approccio feature-based:** vengono estratte caratteristiche locali come occhi, naso e bocca, per poi valutarne le loro descrizioni geometriche o in generale il loro aspetto.

Per quanto riguarda i metodi che sfruttano un *approccio olistico*, si può individuare una sotto-categoria importante che comprende tutti i metodi che utilizzano tecniche di *Principal Component Analysis (PCA)*. Tra questi, il principale è certamente *Eigenfaces* [18]. Altri metodi che realizzano un approccio olistico sono *Fisherfaces/LDA* [19], *Support Vector Machines* [21] e *Neural Networks* [24] [25].

Tra i metodi *feature-based*, vi sono metodi puramente geometrici [20] [26], *Hidden Markov Models (HMM)* [22] [23] e *Dynamic Link Architecture (DLA)* [27]. I metodi di rilevamento delle caratteristiche geometriche delle componenti del volto coinvolgono spesso procedure di rilevamento di punti e contorni (*point/edge detection*) [13] [14] [15] [16].

In generale si può individuare la differenza tra le due categorie nel fatto che in un approccio *feature-based* le informazioni sono ricavate da determinate aree di

interesse, mentre generalmente un approccio *olistico* non è strutturato e parte dal fatto che ogni pixel dell'immagine ha la stessa valenza. I pro e i contro di un approccio rispetto all'altro sono collegati: l'approccio olistico, ad esempio, sfrutta tutta l'informazione che l'immagine può dare ma ha la difficoltà di saper riconoscere quale è utile ai fini del riconoscimento e quale potrebbe indurre all'errore. Viceversa, un'applicazione feature-based dispone di meno informazione anche se si suppone essere significativa. In questo caso, però, vi è la difficoltà aggiuntiva del rilevamento delle posizioni delle componenti del volto.

Inoltre talvolta non si dispone di più immagini per effettuare il "training" del sistema [6], passaggio che viene richiesto solitamente nei metodi che utilizzano un approccio olistico.

Esistono anche tecniche avanzate che mettono insieme i due approcci e per questo motivo sono chiamate *metodi ibridi*.

In tutti i casi, il sistema deve fare i conti con delle problematiche intrinseche del dato in input da elaborare ovvero l'immagine.

### Difficoltà principali

L'immagine può essere acquisita in condizioni diverse. Queste coinvolgono da una parte il soggetto di cui deve essere analizzato il volto, dall'altra l'ambiente circostante. Esistono quindi diversi fattori che devono essere considerati in fase di elaborazione dell'immagine [4] [5], poichè ne condizionano le caratteristiche:

- Variazioni nell'illuminazione del volto all'interno dell'immagine.
- Posizione del volto e quindi deformazione dello stesso rispetto alla classica visione frontale.
- Variazioni delle componenti del volto, dovute all'espressione specifica nel momento dell'acquisizione.
- Variazioni nel tempo del volto ed effetti dovuti all'invecchiamento.
- Fattori di occlusione che impediscono il rilevamento di alcune porzioni del volto.

Sebbene alcuni sistemi di riconoscimento del volto mostrino risultati molto buoni in un contesto che presenta determinate condizioni per questi fattori, può accadere che l'efficacia diminuisca considerevolmente quando il fuonzionamento avviene in circostanze diverse.

## 2.3 Riconoscimento del volto e identikit

Un aspetto importante, che viene preso in considerazione fin dall'inizio, è il rapporto tra riconoscimento del volto e costruzione di identikit.

Appena si pensa a questa associazione, si prende in considerazione l'applicazione che il riconoscimento del volto ha nelle procedure legate alla lotta alla criminalità nel riconoscimento di identità di persone coinvolte. In questi casi la costruzione di un identikit avviene a partire da una descrizione verbale che porta alla composizione

(manuale o automatizzata) di un disegno. La maggior parte dei temi che vengono proposti in letteratura [28] [29] [30] [31], riguardano il passaggio successivo alla costruzione dell'identikit ovvero l'associazione di un identikit all'immagine di un soggetto.

Inoltre sono stati sviluppati diversi software che permettono la costruzione di identikit a partire da cataloghi contenenti un gran numero di tipologie per le varie componenti del volto. Tra questi, viene preso come riferimento il software *Faces*, sviluppato dalla *IQ Biometrics* che rende disponibile una versione didattica: *Faces 4.0 Edu* [1].

# Capitolo 3

# Descrizione del processo di riconoscimento

L'obiettivo del processo di riconoscimento è quello di verifica dell'identità di un soggetto, a partire da una sua immagine del volto.

Il sistema dispone di una *descrizione* dei possibili soggetti che può riconoscere, i quali verranno chiamati *utenti del sistema*. Questa descrizione è tale da permettere il confronto con una descrizione analoga relativa al soggetto da identificare, al fine di verificarne la somiglianza e stabilire se il soggetto corrisponde o meno all'identità dichiarata.

L'idea proposta riguarda l'uso di un *identikit* del volto come *descrizione* del soggetto e di conseguenza come oggetto del confronto per la verifica dell'identità del soggetto stesso.

Un sistema di riconoscimento prevede due fasi diverse:

- la *registrazione* di un nuovo utente nel sistema;
- l'*autenticazione* di un soggetto come utente del sistema.

Prima di stabilire il funzionamento di entrambe le fasi, è necessario specificare in cosa consistono l'immagine di partenza e l'identikit come descrizione del soggetto.

## 3.1 Immagine di partenza

In ingresso al sistema viene fornita un'immagine in cui è presente il volto del soggetto da identificare.

Vengono fatte delle ipotesi sul volto all'interno dell'immagine:

- il volto è completamente contenuto nell'immagine;
- l'immagine rappresenta la visione frontale del volto;
- gli occhi sono aperti;
- l'espressione della bocca è normale, chiusa e rilassata. L'aspetto della parte bassa del volto non è influenzata da espressione particolari.

Non vengono posti vincoli per quanto riguarda la posizione del volto nell'immagine o la sua dimensione.

Si definisce come *asse verticale del volto*, la retta passante tra i due occhi e per il centro di naso e bocca, che divide in due il volto in maniera per quanto possibile simmetrica. Piccole rotazioni dell'asse rispetto al lato verticale dell'immagine sono consentite e non alterano il funzionamento del sistema.

Queste ipotesi sono verosimili nel caso in cui il sistema venga applicato per identificare utenti che sono consapevoli che un dispositivo sta rilevando la propria immagine al momento della richiesta di autenticazione dell'identità dichiarata.

## 3.2 Struttura dell'identikit

Un identikit è, per definizione, una procedura che ha l'obiettivo di ricostruire i tratti somatici del volto di un individuo a partire da una sua descrizione. Con il termine identikit viene indicato anche il risultato di tale procedura.

La costruzione avviene scegliendo una componente del volto alla volta, fino ad arrivare a formare l'intero volto. Poichè la procedura è utilizzata nell'identificazione di persone sconosciute, essa si basa sulla descrizione verbale del volto dell'individuo da identificare, al fine di ricavarne un'immagine attraverso la quale possa essere riconosciuto.

Inizialmente, a partire dalla descrizione, veniva direttamente disegnata l'immagine da parte di ritrattisti esperti. Successivamente, però, venne introdotto l'uso di un catalogo contenente immagini di diverse tipologie di ciascuna componente del volto. La composizione di questi elementi di catalogo, secondo le caratteristiche indicate, andava a formare l'identikit.

Nell'utilizzo dell'identikit per il riconoscimento automatico di identità sviluppato in questo lavoro, cambia il punto di partenza per la costruzione che non è una descrizione orale, bensì un'immagine. Rimangono, però, altre due componenti fondamentali cioè la struttura del risultato finale e l'uso di un catalogo di riferimento.

### Catalogo

Il catalogo è un insieme di elementi divisi per categorie, ciascuna delle quali rappresenta un tratto del volto e in seguito verrà indicata genericamente con il termine *componente*.

Per ogni categoria sono presenti un numero variabile di elementi che cercano di esprimere una varietà di tipologie della specifica componente, nella maniera più completa possibile. Per *completezza* si intende la proprietà dell'insieme di elementi di descrivere qualsiasi soggetto in maniera abbastanza accurata.

Ogni elemento è costituito da un'immagine ovvero il disegno della sola componente in questione secondo i caratteri specifici che esso vuole descrivere. Ogni disegno rappresenta la componente nella visione frontale del volto, coerentemente con l'immagine di partenza.

Graficamente, il disegno è in scala di grigi e possono essere definite aree trasparenti, per consentire la sovrapposizione delle varie componenti.

L'area complessiva di ciascuna immagine è di forma rettangolare e ristretta alla sola porzione del volto relativa alla componente in questione. Inoltre il posizion-

amento di ciascuna componente all'interno dell'immagine rettangolare è vincolato dalla seguente regola: l'asse verticale del volto è parallelo al lato verticale del rettangolo.

## Struttura del risultato finale

Un identikit vero e proprio è formato da un insieme di elementi, ciascuno relativo a una componente diversa. Le immagini relative vengono opportunamente selezionate, posizionate e scalate, per comporre il volto nella maniera più verosimigliante possibile rispetto al soggetto.

Solitamente, quindi, il volto risulta definito in tutte le sue componenti poichè deve essere valutato da una persona che compie un'analisi qualitativa sul volto definito nel suo complesso.

In questo caso particolare, l'identikit viene utilizzato all'interno di una procedura automatica, pertanto l'analisi che deve essere fatta non è paragonabile con il giudizio espresso da una persona in merito al riconoscimento.

Per questo motivo la struttura dell'identikit viene "generalizzata" rispetto a quanto appena descritto, in modo da adattarsi al suo utilizzo per il riconoscimento automatico. In quest'ottica, si può pensare a una struttura nella quale:

- l'identikit è composto da un sottoinsieme di componenti del volto, opportunamente scelte, le quali non devono necessariamente comporre l'intero volto ma allo stesso tempo devono essere sufficienti a rendere efficace il riconoscimento;

- le componenti individuate possono interessare aree del volto sovrapposte tra loro e non necessariamente contigue;

- per ciascuna componente possono essere associati uno o più elementi di catalogo scelti come gli elementi più somiglianti al soggetto in questione;

- l'identikit diventa una struttura che non si può rappresentare in un'unica immagine risultante ma, se le componenti sono $n$ e $k_1, k_2, ..., k_n$ sono le quantità di elementi di catalogo scelti per ciascuna componente, l'identikit individua $k_1 \cdot k_2 \cdot ... \cdot k_n$ immagini diverse corrispondenti alle possibili combinazioni degli elementi scelti;

- l'insieme di elementi selezionaie, non contengono in sè le informazioni riferite al loro posizionamento e a come devono essere scalati per andare a comporre il volto;

- i parametri relativi a posizionamento e scala degli elementi, possono essere aggiunti a corredo della struttura di identikit basata sugli elementi.

La struttura proposta presenta volutamente molti punti interrogativi e resta una definizione generica. La progettazione concreta della struttura dell'identikit verrà affinata con la costruzione dell'intero sistema, in quanto le scelte sono spesso giustificate da esigenze relative a ciascuna fase che verrà presa in esame.

## 3.3 Fase di registrazione

La fase di registrazione serve a creare la base di utenti del sistema, la cui descrizione viene recuperata nel momento della richiesta di autenticazione, per effettuare il riconoscimento.

Per questo motivo, se il confronto avviene tra identikit, per ogni utente che si registra deve essere definito un identikit di riferimento, il quale viene costruito sempre a partire da un'immagine del soggetto in questione. Questa procedura verrà descritta con maggior dettaglio nella sezione successiva.

Assieme all'identikit vengono memorizzati i dati relativi all'utente, perciò dovrà essere creata una base di dati del sistema, in cui una tabella conterrà record relativi agli utenti. I campi relativi a ciascun record possono essere ad esempio:

- Codice identificativo dell'utente;
- Cognome;
- Nome;
- Identikit.

Il campo *Identikit* dovrà essere strutturato per contenere tutte le informazioni relative all'identikit del soggetto.

In fase di autenticazione, ogni volta in cui sarà richiesto l'identikit di un utente, questo dovrà essere individuato nella base di dati e recuperato per consentirne il confronto.

## 3.4 Fase di autenticazione

La fase di autenticazione rappresenta il vero e proprio meccanismo di riconoscimento del sistema e parte dal presupposto che esso sia opportunamente corredato del catalogo e della base dati di utenti.

Il punto di partenza è costituito dall'immagine contenente il volto del soggetto da identificare e dall'identità dichiarata, che si vuole verificare. Il risultato deve essere una scelta, ovvero una risposta affermativa o negativa alla domanda di autenticazione.

Il procedimento si può suddividere a sua volta in una sequenza di cinque fasi:

1. recupero dell'identikit dell'utente dichiarato;
2. rilevazione del volto;
3. rilevazione delle componenti del volto;
4. costruzione dell'identikit;
5. matching tra identikit.

Segue una descrizione più dettagliata di ciascuna fase.

## 1) Recupero dell'identikit dell'utente dichiarato

**Input:** nome e cognome oppure codice identificativo dell'utente dichiarato.

**Output:** identikit relativo all'utente dichiarato.

**Dettagli:** questa fase prevede solamente la ricerca dell'identikit dell'utente che il soggetto dichiara di essere, all'interno base di dati degli utenti. L'identikit è già stato composto nella fase di registrazione, attraverso le fasi 2, 3 e 4 applicate all'immagine di riferimento dell'utente. Se nessun utente corrisponde ai dati dichiarati, la procedura viene interrotta, restituendo esito negativo.

## 2) Rilevazione del volto

**Input:** immagine contenente il volto del soggetto da identificare che rispetta le caratteristiche discusse in precedenza.

**Output:** immagine del volto del soggetto da identificare.

**Dettagli:** attraverso una procedura di *face-detection* viene individuata la porzione dell'immagine che isola il volto del soggetto. Tale area viene estratta e considerata come nuova immagine.

## 3) Rilevazione delle componenti del volto

**Input:** immagine del volto del soggetto da identificare.

**Output:** insieme di aree del volto relative a ciascuna componente da analizzare.

**Dettagli:** attraverso procedure di *features-detection* vengono individuate le porzioni del volto relative alle componenti scelte per la costruzione dell'identikit. Per ciascuna di esse, il termine *area* significa una descrizione che comprende sia l'immagine relativa alla porzione, che la posizione di essa all'interno dell'immagine del volto.

## 4) Costruzione dell'identikit

**Input:** insieme di aree del volto relative a ciascuna componente da analizzare.

**Output:** identikit relativo al soggetto da identificare.

**Dettagli:** una proposta per il passaggio da immagini a identikit viene descritta dettagliatamente nel capitolo 4. Tale procedura permette di associare a ciascuna componente l'insieme di elementi del catalogo più somiglianti alla rispettiva immagine in input, costruendo in tal modo l'identikit secondo la struttura specificata in precedenza. Le informazioni relative alle posizioni delle aree relative a ciascuna componente sono utilizzate nel caso in cui il procedimento coinvolga relazioni tra le diverse componenti.

### 5) Confronto tra identikit

**Input:** identikit relativi al soggetto da identificare e all'utente dichiarato.

**Output:** *sì/no*, risposta alla richiesta di autenticazione.

**Dettagli:** il confronto tra identikit viene descritto nel capitolo 5 e determina se gli identikit costruiti sono sufficientemente somiglianti oppure no. In caso positivo, il soggetto viene autenticato come utente del sistema.

Di primo acchito, si può ipotizzare che la prima fase possa essere eseguita parallelamente alle tre successive. Nella proposta per il confronto descritta nel capitolo 5, però, si sfrutta l'identikit dell'utente dichiarato, ottenuto come output nella prima fase, per la costruzione efficiente dell'identikit del soggetto nella fase numero 4. Pertanto si considererà che la fase 1 può essere eseguita parallelamente alla seconda e terza fase.

L'insieme di operazioni può essere schematizzato come in figura 3.1.

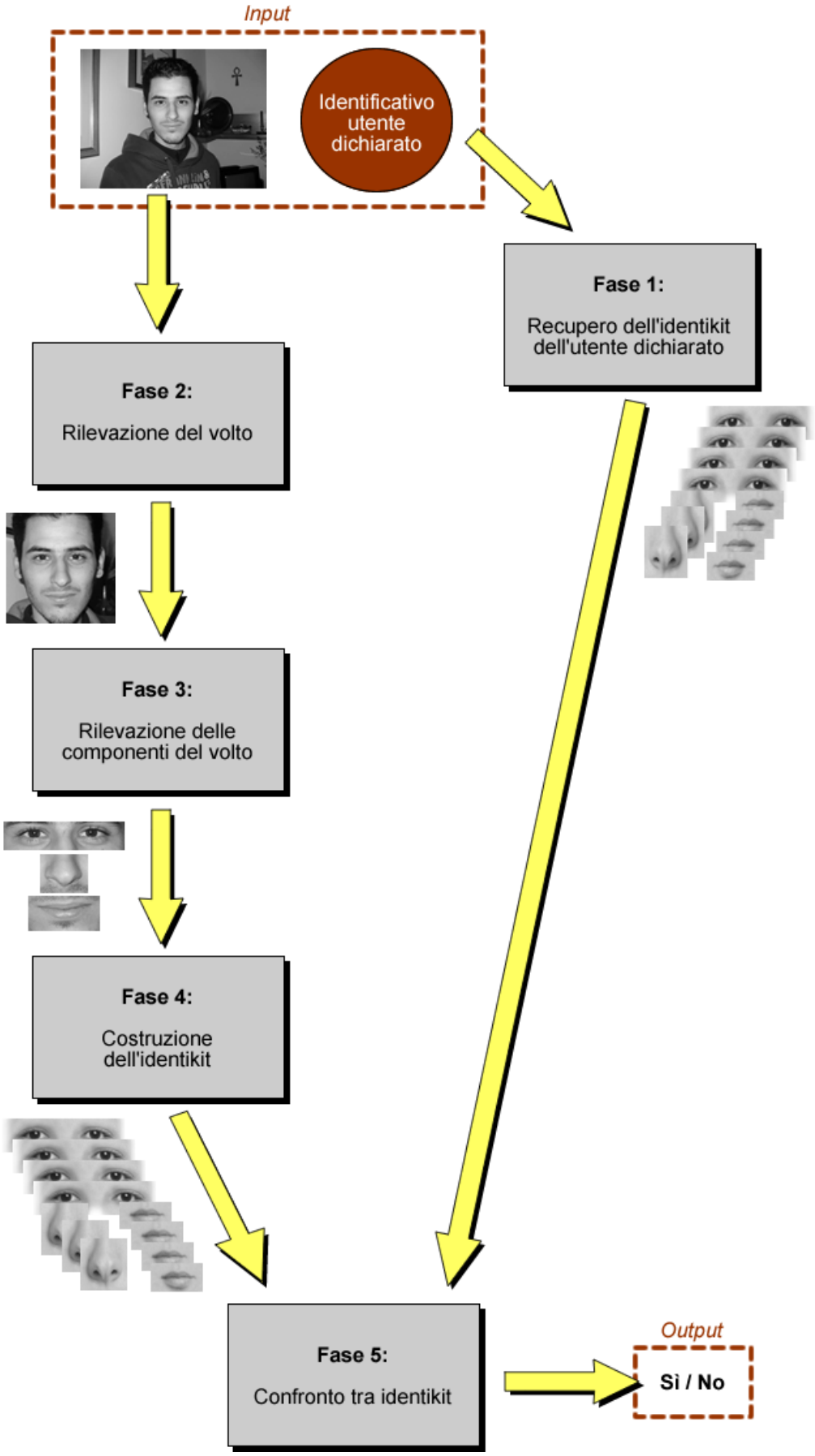


Figura 3.1: Fasi del processo di autenticazione.

# Capitolo 4

# Da immagine a identikit

Questa fase necessita della progettazione di un modello di descrizione delle caratteristiche del volto, il quale gioca un ruolo importante nello sviluppo del sistema di riconoscimento di identità che si basa sull'analisi di questa parte del corpo.

È necessario che tenga conto di quale dato ha come input, di quali caratteristiche deve avere l'insieme di dati che vengono forniti come output e della funzione che questa fase ha nell'intero sistema.

Per prima cosa verrannno illustrate le esigenze che sono state riscontrate esaminando questi due aspetti. All'analisi iniziale, seguirà la descrizione del modello sviluppato che giustificherà dettagliatamente le scelte fatte.

## 4.1 Individuazione dei requisiti

Il processo di riconoscimento richiede il passaggio tra due descrizioni diverse del soggetto che prende in esame:

- l'immagine del volto;
- l'identikit, secondo la struttura descritta nel capitolo precedente.

La costruzione dell'identikit prevede la scelta di un sottoinsieme di elementi di catalogo per ogni componente del volto, mirando ad associare al soggetto gli elementi più somiglianti all'immagine del volto. Di conseguenza, risulta necessario trovare un metodo di confronto tra l'immagine del volto e l'immagine di un elemento di catalogo, con una metrica che possa valutarne la somiglianza.

La prima questione che deve essere risolta riguarda senza dubbio quali sono le componenti del volto che compongono l'identikit. Come conseguenza, questa scelta implica di quale tipo di elementi (occhi, nasi, bocche, orecchie, menti, ecc...) debba essere fornito il catalogo.

La seconda scelta che deve essere fatta è un passaggio successivo perché, per ciascuna componente del volto inserita nel catalogo, è necessario stabilire quali siano i parametri da usare nella valutazione della somiglianza tra l'immagine del volto e un elemento di catalogo per tale componente.

Infine, a partire dell'insieme di parametri per ciascuna componente, si deve arrivare a una decisione del tipo *sì/no* come esito del confronto tra i parametri riferiti

ancora una volta all'immagine del volto da una parte e un elemento di catalogo dall'altra. Tale risultato si può considerare come risposta a una domanda del tipo: *il naso dell'elemento di catalogo numero 14 assomiglia al naso del soggetto nell'immagine data come input?*

Nelle prossime sezioni verranno esaminate queste tre questioni, tenendo conto che l'insieme di confronti di questo tipo sull'immagine di un soggetto, stabilisce quali elementi compongono l'identikit associato al soggetto. Per questo motivo, bisogna tenere conto delle seguenti considerazioni:

- l'identikit deve descrivere il soggetto accuratamente ovvero soggetti diversi dovrebbero corrispondere a identikit diversi;
- gli identikit costruiti a partire da immagini diverse dello stesso soggetto dovrebbero essere quantomeno molto simili;
- la costruzione dell'identikit non deve essere condizionata da fattori come la dimensione dell'immagine e la posizione e dimensione relativa del volto all'interno dell'immagine.

## 4.2 Scelta delle componenti del volto

L'immagine di partenza e gli elementi dell'identikit corrispondono ad una visione frontale del volto.

A partire da questo, si deve cercare un insieme di componenti che abbia le seguenti proprietà:

**Persistenza:** le componenti non devono essere soggette a cambiamenti nel tempo per lo stesso soggetto, ovvero non si può fare affidamento su caratteristiche che possano variare da un giorno all'altro.

**Individuabilità:** l'obiettivo è arrivare a un'analisi automatica del volto, pertanto le componenti devono essere rilevate in maniera automatica.

**Distinguibilità:** le componenti nell'insieme devono essere caratterizzanti per il soggetto, ovvero idealmente devono poterlo identificare univocamente.

Il software *Faces* [1] costruisce un'immagine frontale di un volto partendo da una lista di componenti di cui segue un elenco nel quale sono stati omessi elementi che non interessano l'aspetto fisionomico (ad esempio occhiali e orecchini):

- capelli;
- forma della testa;
- sopracciglia;
- occhi;
- naso;
- bocca;
- forma del mento;
- baffi;
- barba;
- pizzetto;
- linee sulla fronte;
- linee sotto gli occhi;
- linee sulle guance;
- linee agli estremi della bocca;
- linee sul mento;
- nèi;
- cicatrici.

Scegliendo di tenere questo elenco come riferimento che indica le possibili componenti di un identikit, è necessario effettuare una valutazione che tenga conto del problema che si sta trattando e quindi delle proprietà elencate in precedenza.

Per prima cosa, il riconoscimento del volto deve fare affidamento su caratteristiche persistenti. Per un soggetto non è opportuno, quindi, considerare identificanti elementi come capelli, sopracciglia, baffi, barba e pizzetto.

Inoltre, per la proprietà di individuabilità, si possono escludere componenti come la forma della testa e del mento poichè possono essere nascoste.

Un problema analogo porta ad escludere componenti come cicatrici e nèi perché non hanno forme e posizioni determinate, pur essendo segni visibili e che caratterizzano il soggetto.

Le linee del volto relative a fronte, regione oculare, guance, bocca e mento sono riconducibili, al contrario, ad aree prestabilite. Esse, però, soffrono ugualmente di un problema di individuabilità poichè non sono ben definite e dipendono molto dalla direzione della luce e in generale dalla luminosità nella situazione in cui l'immagine è scattata.

Secondo questa breve analisi, rimangono solo tre componenti iniziali che rispettano le proprietà di persistenza e individuabilità, cioè: occhi, naso e bocca. Infatti, ciascuna di esse è facilmente individuabile, come ad esempio viene proposto in [**?**] [11]. Inoltre, gli inevitabili cambiamenti dovuti all'invecchiamento si possono considerare minimi anche in un periodo abbastanza lungo, quindi trascurabili. Infine si può supporre che nell'immagine non ci siano variazioni dovute all'espressione del volto come ad esempio occhi chiusi, bocca aperta o sorridente. Sotto queste ipotesi, per le tre componenti vale la proprietà di persistenza.

Rimane da verificare se l'insieme *occhi-naso-bocca* garantisce la proprietà di distinguibilità. Per quanto detto finora, questo rappresenta il più grande sottoinsieme di componenti che verificano le altre due proprietà. Più fattori si considerano, più aumenta la distinguibilità, di conseguenza di tutti i sottoinsiemi considerabili è quello che massimizza quest'ultima caratteristica richiesta.

Nel modello che verrà proposto, saranno *occhi*, *naso* e *bocca* le tre componenti che andranno a formare l'identikit. Nelle prossime sezioni si dà per scontato che sia possibile confrontare l'immagine di un elemento di catalogo per una certa componente con l'immagine del volto ristretta all'area interessata dalla componente considerata.

L'analisi informale descritta in questa sezione può essere arricchita e migliorata. La struttura del modello per il volto che viene progettato può cambiare e richiedere ulteriori analisi ad hoc per ciascuna componente inserita. Un'eventuale evoluzione in questa scelta può essere apportata senza modificare il modello per gli altri livelli di progettazione dell'intero sistema di riconoscimento.

## 4.3 Un modello per ciascuna componente

Per confrontare due immagini, è necessario orientarsi in primo luogo sulla tipologia di tecnica da usare. Dunque, la prima scelta deve essere fatta tra l'utilizzo di un approccio *features-based* e un approccio *olistico* (in riferimento alla classificazione proposta in sezione 2.2).

Il passaggio alla descrizione tramite identikit, prevede il confronto tra due immagini di natura diversa: l'immagine del volto acquisita come fotografia e il disegno

di una componente appartenente al catalogo. Questa prima osservazione mette in discussione un possibile approccio *olistico* che si basa sulla somiglianza di immagini nel loro insieme.

Un'ulteriore considerazione da fare è che una tecnica *olistica* prende in considerazione l'intera immagine del volto, quindi un confronto diretto tra immagini avrebbe più senso rispetto al passaggio a identikit e alla scomposizione in componenti.

Per queste motivazioni, il modello proposto applica un approccio *features-based* basato sul confronto di parametri geometrici relativi alle diverse componenti del volto da prendere in esame per la costruzione dell'identikit. Scelto questo metodo, è necessario definire i parametri geometrici che caratterizzano ciascuna componente.

Segue una proposta dettagliata componente per componente, fondata sull'analisi del software *Faces* [1], usato come riferimento per la costruzione degli identikit.

## Occhi

Per la componente occhi, gli elementi di catalogo del software di riferimento vengono suddivisi nelle categorie elencate in tabella 4.1.

| Categorie | Descrizione |
|---:|---|
| *Narrow* | Altezza dell'occhio molto piccola in proporzione alla larghezza. |
| *Deep-set* | Occhi incavati. |
| *Oriental* | Occhi di carattere orientale. |
| *Heavy Lids* | Palpebra superiore abbassata. |
| *Average Blue* | Occhi chiari (medi per i caratteri occidentali) |
| *Average Brown* | Occhi scuri (medi per i caratteri occidentali) |
| *Almond-shaped Blue* | Occhi chiari a mandorla. |
| *Almond-shaped Brown* | Occhi scuri a mandorla. |
| *Bulging* | Occhi sporgenti. |
| *Hooded* | Occhi socchiusi. |

Tabella 4.1: Classificazione della componente *occhi* secondo il software *Faces*.

La classificazione che viene fatta è qualitativa e si basa sull'aspetto generale, consono al discernimento da parte di una persona. Ne è una prova il fatto che si possono trovare occhi molto simili in categorie diverse.

L'analisi automatica, invece, richiede il rilevamento di parametri che corrispondano a metriche specifiche definite sulla struttura del volto. Per questi motivi la classificazione dell'occhio in base alle categorie sopraccitate, non è efficace ai fini del confronto.

Dalle categorie, però, è possibile dedurre quali possano essere le caratteristiche che distinguono i diversi elementi di catalogo tra loro:

1. Proporzione tra altezza e larghezza;

2. Angolo di inclinazione della retta passante per il punto più esterno e il punto più interno;

3. Forma della palpebra superiore;
4. Forma della palpebra inferiore;
5. Visibilità e forma dell'orbita oculare (parte superiore);
6. Visibilità e forma dell'orbita oculare (parte inferiore);
7. Colore dell'iride (più chiaro o più scuro in una scala di grigi).

Nel software la dimensione degli occhi è fissa e deve essere di riferimento per le dimensioni delle restanti parti del volto. Come conseguenza le altre componenti vengono regolate in proporzione agli occhi.

Si sottolinea la differenza tra la proporzione tra altezza e larghezza dell'occhio e la grandezza dell'occhio in sè. Infatti la prima determina la tipologia di occhio, la seconda è un parametro che ha valenza nel rapporto con le altre parti del volto e non va ad influenzare il rapporto di proporzione interno all'occhio stesso.

Il software, inoltre, dà la possibilità di modificare un parametro importante cioè la distanza tra i due occhi, che verrà considerata come distanza tra i due estremi interni dell'occhio sinistro e dell'occhio destro. Questo parametro non distingue i diversi elementi di catalogo ma, scelto uno di essi, è possibile avvicinare o separare gli occhi in un range predefinito di valori. Pertanto questo parametro non sarà considerato nel modello per l'occhio che si va a delineare in questa fase.

Anche in questa fase, dato l'insieme di caratteristiche costruito a partire da un'analisi informale, è necessario verificare quali tra esse, nel caso specifico che si sta trattando, rispettano le tre proprietà precedentemente definite: *persistenza*, *individuabilità* e *distinguibilità* dell'insieme scelto.

Si suppone che l'occhio sia aperto, quindi non ci siano variazioni rilevanti rispetto alla chiusura delle palpebre. Con questo presupposto, si può affermare che l'unica caratteristica non *persistente* risulta essere il colore dell'iride, in quanto le immagini sono in scala di grigi e variazioni di luminosità nelle immagini, ombre e luce riflessa possono alterarne l'aspetto. È da sottolineare anche che l'informazione sul colore rilevata dall'immagine è ridotta già dal fatto che le immagini sono in scala di grigi.

Per quanto riguarda l'*individuabilità*, risulta complicata la rilevazione di una forma non lineare come può essere il perimetro dell'occhio, di conseguenza si decide di scartare dal modello i fattori di forma corrispondenti ai punti 3, 4, 5, 6.

Le caratteristiche rimanenti (1, 2) si possono invece considerare facilmente individuabili: infatti è sufficiente stabilire la posizione di quattro punti dell'occhio per calcolare i due parametri geometrici ad esse associati.

Siano dunque $p$ e $\alpha$ i parametri relativi rispettivamente alle caratteristiche 1 e 2. Allora l'occhio può essere rappresentato con un modello *a rombo*, definito secondo le seguenti caratteristiche:

- La diagonale maggiore $a$ del rombo è fissata, in quanto si suppone che tale parametro sia di riferimento per i restanti, coerentemente al software.
- La diagonale minore $b$ è data dalla proporzione tra altezza e larghezza dell'occhio come: $b = p \cdot a$, dove $p = altezza/larghezza$.
- La rotazione del rombo sul piano, ovvero l'angolo tra la diagonale principale e a l'asse orizzontale di riferimento, è data dall'angolo $\alpha$.

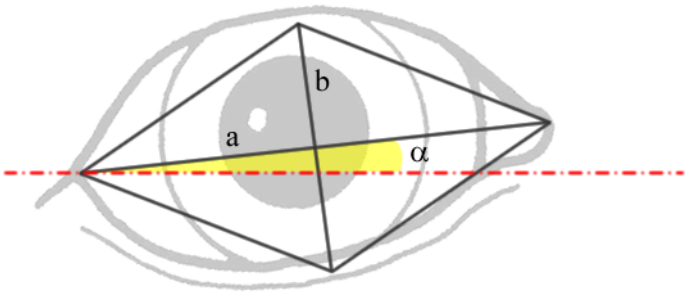


Figura 4.1: Modello *"a rombo"* per l'occhio.

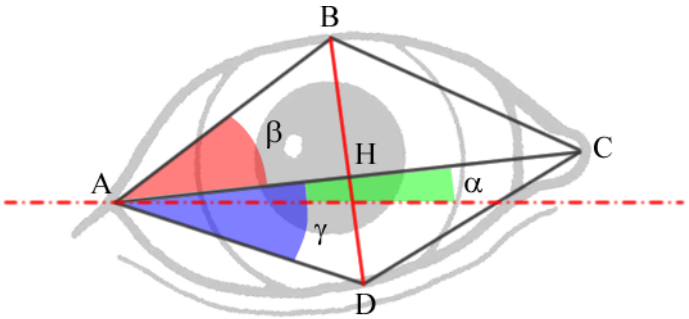


Figura 4.2: Modello *"a triangoli"* per l'occhio.

Come per l'analisi fatta nella sezione precedente, si potrebbe sostenere che il sottoinsieme di caratteristiche considerato è il più grande che verifica le proprietà di persistenza e individuabilità, pertanto è uno di quelli che massimizza la *distinguibilità*. In questo caso, però, escludendo la forma del perimetro dell'occhio e considerando solamente la proporzione tra altezza e larghezza, si perdono informazioni relative al grado di simmetria tra l'arco superiore e quello inferiore, dato che non è scontato.

Si può dunque estendere il modello *a rombo* per l'occhio con un nuovo modello costruito da due triangoli isosceli aventi la base in comune. In questo modo i due triangoli descrivono separatamente l'arcata superiore e quella inferiore.

Si osserva che l'informazione relativa alla proporzione tra altezza e larghezza è comunque rappresentata dalla somma delle due altezze dei due triangoli.

Nella scelta dei parametri geometrici che descrivono il modello, è necessario tenere conto che questi non devono essere influenzati dalla grandezza assoluta dell'occhio. Per questo motivo, si considerano gli angoli alla base di entrambi i triangoli. Infatti, se una figura geometrica viene fatta variare secondo un fattore di scala, gli angoli rimangono invariati.

Per semplicità di calcolo, si considererà il seno di ciascun angolo in questione. Poichè ciascuno di essi è certamente compreso nell'intervallo $[-90°, 90°]$ in cui la funzione seno è biettiva, tale passaggio non danneggia la descrizione.

I parametri geometrici che descrivono il modello per l'occhio, in riferimento alla figura 4.2, sono quindi:

- $\sin\alpha$, dove $\alpha$ è l'angolo compreso tra la base dei due triangoli e l'asse orizzontale di riferimento, ovvero tra il segmento $AC$ e l'asse orizzontale passante per il punto A (estremo esterno).

- $\sin\beta$, dove $\beta$ è l'angolo compreso tra la base dei due triangoli e un lato del triangolo superiore, ovvero l'angolo $C\widehat{A}B$.
- $\sin\gamma$, dove $\gamma$ è l'angolo compreso tra la base dei due triangoli e un lato del triangolo inferiore, ovvero l'angolo $D\widehat{A}C$.

Fissati i parametri geometrici, bisogna considerare che essi vengono rilevati per entrambi gli occhi. Nella costruzione dell'identikit con il software *Faces*, per questa componente non viene data la possibilità di impostare un occhio diverso dall'altro. Si suppone, ragionevolmente, che esista un grado elevato di simmetria tra essi.

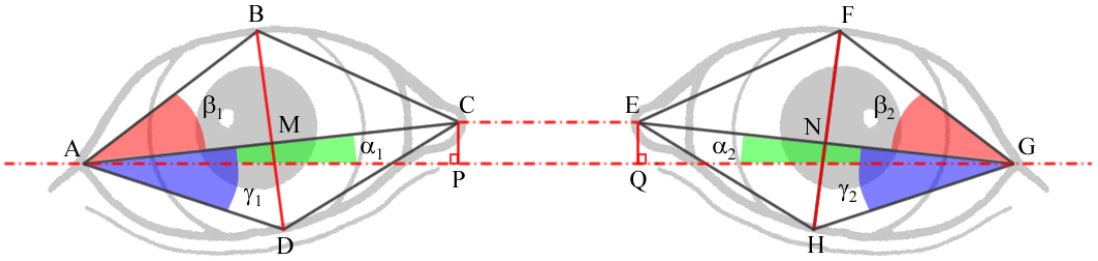


Figura 4.3: Modello definitivo per gli occhi.

Nel modello finale (figura 4.3) , ovvero quello che costituirà l'insieme di parametri usati per confrontare immagine del volto e elementi di catalogo, si mantiene la stesso approccio e si calcola ciascuno dei tre parametri come media tra lo stesso parametro nei due occhi:

$$\sin\alpha = \frac{\sin\alpha_1 + \sin\alpha_2}{2} \qquad \sin\beta = \frac{\sin\beta_1 + \sin\beta_2}{2} \qquad \sin\gamma = \frac{\sin\gamma_1 + \sin\gamma_2}{2}$$

dove:

$$\sin\alpha_1 = \begin{cases} \frac{PC}{AC}, & \text{se } y_C \geq y_P \\ -\frac{PC}{AC}, & \text{se } y_C < y_P \end{cases} \qquad \sin\alpha_2 = \begin{cases} \frac{QE}{EG}, & \text{se } y_E \geq y_Q \\ -\frac{QE}{EG}, & \text{se } y_E < y_Q \end{cases}$$

$$\sin\beta_1 = \frac{MB}{AB} \qquad \sin\beta_2 = \frac{NF}{GF} \qquad \sin\gamma_1 = \frac{MD}{AD} \qquad \sin\gamma_2 = \frac{NH}{GH}$$

Nella definizione si considera inoltre che gli angoli $\beta_1$, $\beta_2$, $\gamma_1$, $\gamma_2$ siano compresi nell'intervallo $[0°, 90°]$, mentre $\alpha_1$ e $\alpha_2$ possano assumere valori in $[-90°, 90°]$, ovvero sia positivi che negativi.

## Naso

Anche per il naso, gli elementi di catalogo del software di riferimento vengono suddivisi in categorie (tabella 4.2).

La classificazione che viene fatta è ancora una volta qualitativa. Per le stesse ragioni spiegate per gli occhi, vengono delineate delle caratteristiche che non corrispondono perfettamente alla classificazione proposta ma derivano da essa:

1. Larghezza della base rispetto alla lunghezza del naso;

| Categorie | Descrizione |
|---|---|
| *Narrow* | Sottile. |
| *Average with Round Base* | Base tonda. |
| *Average with Broad Base* | Base medio-larga. |
| *Average Pointed* | Naso a punta fina. |
| *Hooked, Nostrils not Showing* | Naso ad uncino (punta verso giù) e narici nascoste. |
| *Hooked, Nostrils Showing* | Naso ad uncino (punta verso giù) e narici visibili |
| *Slightly Flared Nostrils* | Narici grandi. |
| *Very Flared Nostrils* | Narici molto grandi. |
| *Round* | Naso tondo. |
| *Large* | Naso largo. |
| *Wide Base, Nostrils Showing* | Base larga, narici visibili. |
| *Wide Base, Nostrils not Showing* | Base larga con narici nascoste. |

Tabella 4.2: Classificazione della componente *naso* secondo il software *Faces*.

2. Visibilità e grandezza delle narici;
3. Forma e dimensioni della punta;
4. Forma e dimensioni della base.

Oltre a queste, si possono considerare altri due parametri per il naso ovvero la sua grandezza nel complesso e la posizione nel volto. Come per la distanza tra gli occhi, questi non sono legati alla discriminazione tra un elemento di catalogo e un altro, ma alla composizione del volto nel suo complesso e quindi al rapporto tra le diverse componenti. Per questo motivo, non saranno considerati nel modello sviluppato per il confronto della componente in sè.

Tornando all'elenco, ancora una volta esso presenta caratteristiche non utilizzabili ai fini dell'analisi automatica: le forme della punta e della base del naso sono difficili da individuare. Allo stesso modo è difficile individuare le narici con una precisione tale da confrontarne la grandezza.

Il naso, a differenza degli occhi, presenta contorni meno definiti. Pertanto risulta difficile rilevarne le caratteristiche geometriche. Ad esempio, la visione frontale non permette di stabilire quale sia la radice del naso, ovvero quel punto che nella visione di profilo è facilmente individuabile come vertice dell'angolo formato dalla fronte e dal dorso del naso.

Per stabilire un riferimento superiore per misurare l'altezza del naso, si utilizzerà quindi la posizione degli occhi. Per la precisione si considera l'asse orizzontale usato come riferimento anche per gli stessi occhi, ovvero l'asse passante per gli estremi esterni dei due occhi. Il punto di riferimento può essere quindi calcolato come punto medio del segmento che unisce tali punti rilevati sugli occhi (punti A e G in figura 4.3).

È possibile invece rilevare facilmente alcuni punti relativi alla base del naso, in maniera tale da stabilirne la larghezza e l'altezza. Si può considerare un modello geometrico costituito da un solo triangolo isoscele in cui la base corrisponde alla

larghezza della base del naso, mentre i due lati obliqui rettificano il profilo inferiore della base fino al limite inferiore del naso.

Il modello che viene definito utilizza una struttura che considera sia il triangolo appena descritto, che il punto di riferimento individuato a partire dagli occhi ed è rappresentato in figura 4.4.

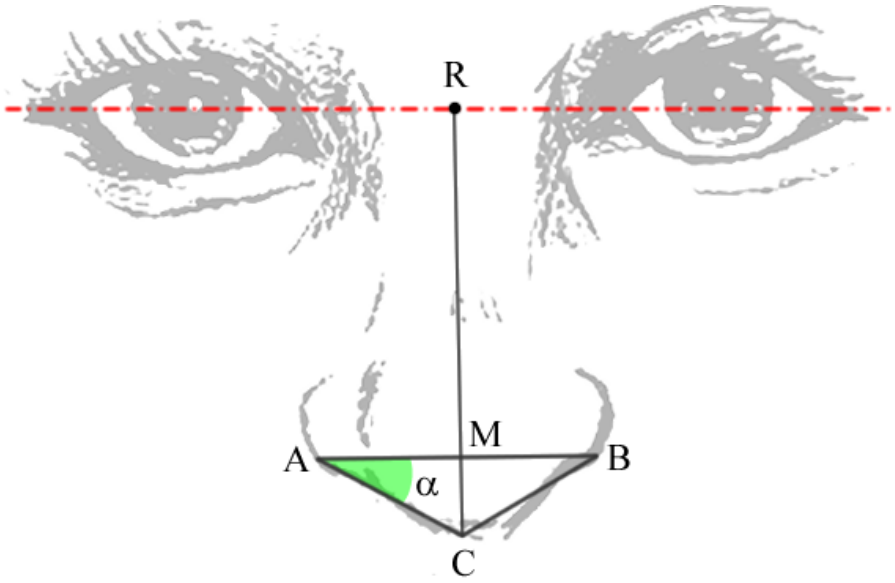


Figura 4.4: Modello definitivo per il naso.

I punti $A$, $B$, $C$ individuano il triangolo isocele ed il punto $R$ è il riferimento superiore calcolato come specificato in precedenza. $M$ è il punto medio del segmento $AB$, base del triangolo isoscele, e non necessariamente è un punto del segmento verticale $RC$. In questo modo $MC$ rappresenta l'altezza del triangolo.

A partire da questo modello, i parametri geometrici che lo descrivono sono:

- $p$, rapporto tra larghezza e altezza del naso;
- $\sin\alpha$, dove $\alpha$ è l'angolo compreso tra la base del triangolo e uno dei lati obliqui.

Questi valori possono essere calcolati a partire dai punti, secondo le seguenti equazioni:

$$p = \frac{AB}{RC} \qquad \sin\alpha = \frac{MC}{AC}$$

Si può osservare che la definizione del triangolo isoscele utilizzato nel modello, non dà un'esatta informazione sull'altezza effettiva della base del naso in quanto non tiene conto dell'altezza delle ali del naso. Questa scelta è dettata dalla difficoltà di individuare punti significativi che delimitino superiormente quest'area che abbiamo definito nel complesso come base del naso.

## Bocca

Per prima cosa, si prende in considerazione la classificazione fatta dal software *Faces* che viene descritta in tabella 4.3.

| Categorie | Descrizione |
|---|---|
| *Small* | Bocca stretta e labbra più spesse in proporzione. |
| *Thin* | Labbra sottili. |
| *Even* | Labbra medie e simmetriche in grandezza rispetto all'asse orizzontale. |
| *Raised Upper Lip* | Labbro superiore più grande rispetto a quello inferiore. |
| *Overhanging Upper Lip* | Labbro superiore sporgente. |
| *Heart-Shaped* | Forma a cuore con labbro superiore che forma un archetto detto di Cupido. |
| *Thin Upper Lip* | Labbro superiore più sottile rispetto a quello inferiore. |
| *Wide* | Bocca larga e labbra più sottili in proporzione. |
| *Thick* | Labbra grandi. |

Tabella 4.3: Classificazione della componente *bocca* secondo il software *Faces*.

A differenza dei casi precedenti, la suddivisione mette in risalto caratteri legati a metriche come dimensioni e propozioni, pur rimanendo di carattere qualitativo. Risulta quindi molto utile per ricavarne un insieme di parametri legati alla struttura geometria della componente.

Nella sua stesura, si tiene conto di due supposizioni: la bocca è chiusa, non caratterizzata da espressioni particolari e si considera simmetrica rispetto all'asse vericale che passa per il centro. Pertanto si considerano le seguenti caratteristiche:

1. Proporzione tra altezza e larghezza della bocca;
2. Altezza del labbro superiore
3. Altezza del labbro inferiore;
4. Forma del labbro inferiore;
5. Forma del labbro superiore e forma dell'archetto di Cupido.

Oltre a queste, come per occhi e bocca, si possono considerare altri due parametri che non sono legati alla discriminazione tra un elemento di catalogo e un altro, ma alla composizione del volto nel suo complesso e quindi al rapporto tra le diverse componenti. Questi sono la grandezza della bocca nel complesso e la sua posizione nel volto e per ora non saranno considerati.

Ancora una volta, le caratteristiche relative alla forma di un contorno, come in questo caso quello delle labbra, risultano difficili da individuare attraverso una procedura automatica.

Le tre caratteristiche rimanenti (1, 2, 3), riguardanti le dimensioni della bocca possono essere rappresentate ancora una volta con un modello "a triangoli" simile a quello costruito per l'occhio. Infatti, come per l'occhio vengono trattati separatamente l'arcata superiore e quella inferiore, per la bocca vengono rappresentate le labbra con due triangoli isosceli diversi, come viene mostrato in figura 4.5.

Tale modello viene descritto dai seguenti parametri geometrici:

- $\sin\alpha$, dove $\alpha$ è l'angolo compreso tra la base dei due triangoli e un lato del triangolo superiore, ovvero l'angolo $C\widehat{A}B$.

- $\sin\beta$, dove $\beta$ è l'angolo compreso tra la base dei due triangoli e un lato del triangolo inferiore, ovvero l'angolo $D\widehat{A}C$.

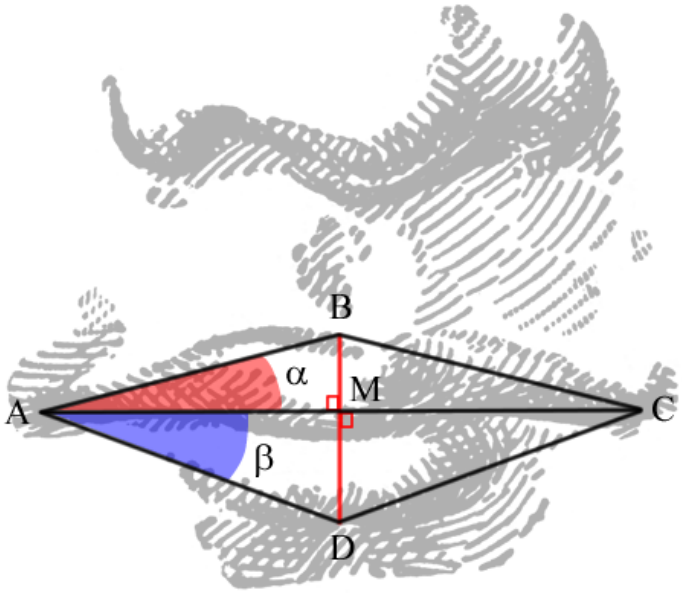


Figura 4.5: Modello definitivo per la bocca.

A partire dai quattro punti individuati, i parametri possono essere così calcolati:

$$\sin\alpha = \frac{BM}{AB} \qquad \sin\beta = \frac{MD}{AD}$$

Le tre caratteristiche non scartate per motivi legati all'individuabilità (1, 2, 3), si sono ridotte a due parametri poichè l'altezza complessiva della bocca è un'informazione che può essere dedotta dall'altezza delle singole labbra. Di conseguenza sarebbe ridondante inserire un ulteriore parametro per tale caratteristica.

## Parametri globali

Nei tre modelli vengono presi in considerazione parametri geometrici che riguardano proporzioni all'interno della componente stessa e sono stati volutamente tralasciati fattori come la grandezza e la posizione rispetto al volto stesso. Questa scelta è giustificata dal fatto che i tre modelli per occhi, naso e bocca devono essere derivati da immagini di natura diversa e non devono essere dipendenti dalla scala dell'immagine. A maggior ragione, per i parametri scartati, non si può parlare di dimensione e posizione rispetto al volto, in quanto uno dei due oggetti a confronto è un elemento di catalogo e non ha un volto di riferimento.

Giustificata la scelta, si può obiettare che i modelli non verifichino la proprietà di distinguibilità. Pensando alla bocca, per esempio, si può correttamente sostenere che il modello descritto dai soli due angoli $\alpha$ e $\beta$ consideri simili una bocca stretta e sottile e una bocca larga e con le labbra spesse.

Affinché il modello complessivo tenga conto anche delle caratteristiche che sono state scartate, è possibile corredare l'analisi che utilizza gli identikit con un insieme di

parametri geometrici definiti come *parametri globali*, in quanto non si riferiscono alle caratteristiche interne di ciascuna componente ma mettono in gioco più componenti tra loro.

Se si prende come riferimento ancora una volta il software *Faces*, si può osservare che ciascuna componente non viene solo scelta come un elemento di catalogo, ma in generale può essere scalata e spostata all'interno del volto.

Come specificato in precedenza, il riferimento principale nel software sono gli occhi: infatti essi non possono essere spostati sull'asse verticale e la loro grandezza non può essere variata. L'unico parametro che può essere modificato è la distanza tra i due punti interni dei due occhi, ovvero essi possono essere allontanati o avvicinati, mantenendo la simmetria rispetto all'asse verticale passante al centro del volto.

Per quanto concerne la posizione di naso e bocca, si suppone che essi si trovino sull'asse verticale al centro del volto. La loro posizione può variare dunque sull'asse verticale. Si tiene dunque come punto di riferimento il punto medio del segmento che unisce gli estremi esterni dei due occhi.

Riguardo al naso, per costruzione del modello, la distanza da tale punto è sempre pari a zero, in quanto il riferimento è proprio il punto che viene identificato come "estremo superiore del naso". Di conseguenza, non è necessario considerare un parametro che ne identifichi la posizione.

Per la bocca, invece, si può considerare come parametro la distanza tra il centro della bocca e il punto di riferimento fissato sugli occhi.

Per quanto riguarda la dimensione di ciascuna componente, gli occhi saranno considerati come riferimento. Per la dimensione di naso e bocca, invece, è opportuno considerare la lunghezza del naso e la larghezza della bocca.

I due parametri di posizione e i due di dimensione definiti informalmente finora, devono rispettare a loro volta la proprietà di indipendenza dalla scala dell'immagine considerata. Per definire correttamenti ciascun parametro, è necessario che ciascuno sia rapportato a una grandezza di riferimento.

Per quanto discusso, la scelta cade nuovamente sugli occhi e la *distanza di riferimento* scelta è la distanza tra l'estremo esterno dell'occhio destro e il punto simmetrico dell'occhio sinistro. Ogni parametro viene calcolato quindi come rapporto tra la distanza che descrive e la distanza di riferimento.

Riassumendo e formalizzando i concetti sulla base della figura 4.6, i *parametri globali* sono così definiti:

- $d_o = \frac{CD}{AB}$, rappresenta la distanza tra i due punti più interni dei due occhi;
- $l_n = \frac{RE}{AB}$, rappresenta la lunghezza del naso;
- $l_m = \frac{FH}{AB}$, rappresenta la larghezza della bocca;
- $d_m = \frac{RG}{AB}$, rappresenta la posizione della bocca come distanza dagli occhi sull'asse verticale.

Si specifica che, in riferimento alla figura 4.6, i punti $R$, $E$ e $G$ non necessariamente appartengono alla stessa retta.

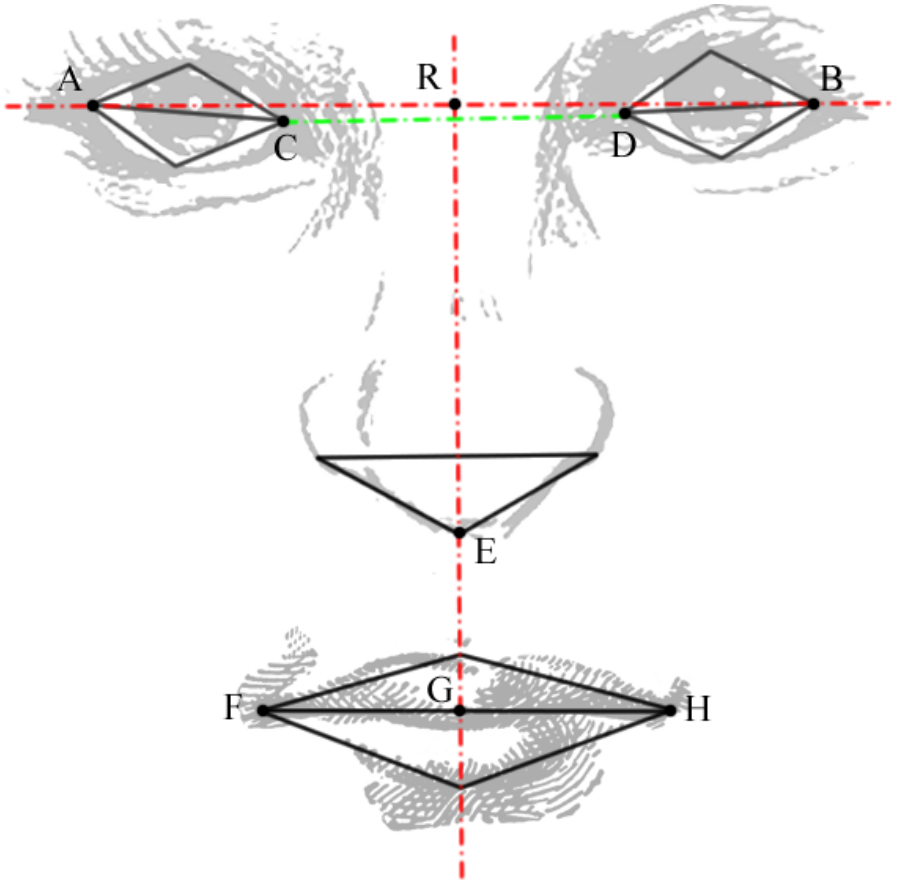


Figura 4.6: Modello per la definizione dei *parametri globali*.

## 4.4 Il confronto su ciascuna componente

Fissato il modello, per ciascuna componente è necessario stabilire una funzione che, data una componente del volto, riceve come input i parametri geometrici di un'immagine e di un elemento di catalogo e fornisce una risposta di tipo *sì/no* alla domanda: *"Tale elemento di catalogo è abbastanza somigliante alla componente del volto del soggetto nell'immagine data come input?"*

Per descrivere questo confronto, viene definita una descrizione formale per i modelli costruiti nella sezione precedente. La descrizione proposta è tale da essere adattabile a qualsiasi modello. Pertanto, qualora i modelli fossero modificati secondo scelte differenti, la descrizione resterebbe coerente.

### Descrizione formale per il modello

Siano $n$ le componenti del volto scelte, ciascuna delle quali è identificata dal simbolo $c_i$, con $1 \leq i \leq n$. Sia $k_i$ il numero di parametri geometrici associati a $c_i$.

Sia un'immagine, identificata con una lettera maiuscola dell'alfabeto (ad esempio $A$, $B$, ...) e sia $\Gamma_{im}$ l'immagine relativa all'$m$-esimo elemento di catalogo per la componente $c_i$.

I parametri geometrici per le diverse componenti sono indicati dal simbolo $p_{ij}$ al

variare degli indici $i$ e $j$, con il seguente significato:

$$p_{ij}(A) = \{\text{Valore del parametro } j\text{-esimo della componente } c_i \text{ nell'immagine } A\}$$
$$\text{con:} \quad 1 \leq i \leq n \quad \text{e} \quad 1 \leq j \leq k_i$$

Per il modello sviluppato nella sezione precedente, si considerano dunque:

| | | | |
|---|---|---|---|
| $n = 3$ | $c_1 = \text{occhi}$ | $c_2 = \text{naso}$ | $c_3 = \text{bocca}$ |
| | $k_1 = 3$ | $k_2 = 2$ | $k_3 = 2$ |

## Calcolo dell'esito del confronto

Fissata una componente $c_i$, il confronto avviene tra due immagini di input $A$ e $\Gamma_{im}$, per un certo soggetto rappresentato nell'immagine $A$ e per l'elemento di catalogo relativo alla componente $c_i$ che ha indice $m$.

Il confronto è determinato da una funzione che valuta i parametri relativi alla componente data. Tale funzione vuole essere una misura della distanza tra le due immagini in input, secondo i parametri stabiliti. Essa può essere definita nel modo seguente:

$$D_i(A, \Gamma_{im}) = \sqrt{\sum_{j=1}^{k_i} \Big[\alpha_{ij}\Big(p_{ij}(A) - p_{ij}(\Gamma_{im})\Big)\Big]^2} \qquad \text{per } 1 \leq i \leq n$$
$$\text{con: } \alpha_{ij} \text{ coefficiente (peso) associato al parametro } p_{ij}$$

La scelta dei pesi da associare ai parametri meriterebbe un'analisi accurata che non è trattata in questo lavoro. La soluzione più semplice consisterebbe nell'impostare ogni coefficiente al valore 1. In tal caso la formula per la distanza diventa la classica *distanza Euclidea*.

Una proposta più articolata, potrebbe essere quella di normalizzare ciascun parametro in base alla statistica del parametro calcolata nell'insieme di elementi del catalogo. In tal caso si può passare alla *distanza Euclidea normalizzata* impostando i coefficienti in base alla deviazione standard:

$$\alpha_{ij} = \frac{1}{\sigma_{ij}}$$

Si può calcolare $\sigma_{ij}$ come deviazione standard relativa all'insieme di valori dati dal parametro $p_{ij}$ calcolato su ciascun elemento di catalogo per la componente $c_i$, ovvero l'insieme: $\{p_{ij}(\Gamma_{im}) \mid \Gamma_{im} \text{ è un elemento di catalogo}\}$.

L'esito del confronto è dato dalla comparazione della distanza con una soglia prefissata $th_i$ per ciascuna componente. Solo se la distanza $D_i(A, \Gamma_{im})$ è inferiore a tale soglia, l'elemento di catalogo $\Gamma_{im}$ è associato all'identikit del soggetto nell'immagine $A$ per la componente $c_i$.

A partire da questa descrizione formale, l'identikit $I(A)$, associato al soggetto di un'immagine $A$, si può definire in questo modo:

$$I(A) = \Big(I_1(A), I_2(A), ..., I_n(A)\Big)$$
$$\text{con:} \quad I_i(A) = \{\Gamma_{im} \mid D_i(A, \Gamma_{im}) < th_i\} \quad \forall i \text{ t.c. } 1 \leq i \leq n$$

### Scelta della soglia $th_i$

La scelta di $th_i$ deve rispettare un limite inferiore affinchè il modello di identikit sia accettabile.

Si pensi agli elementi di catalogo per una componente $i$ generica: essi sono punti nello spazio $k_i$-dimensionale definito dai parametri geometrici per tale componente. Quando un soggetto viene analizzato, anch'esso viene associato a un punto dello spazio e il suo identikit è composto da quegli elementi di catalogo i cui punti sono "abbastanza vicini".

Questa logica viene compromessa nel caso in cui una componente non abbia elementi di catalogo "abbastanza vicini". Immaginando un'ipersfera di raggio $th_i$ nello spazio $k_i$-dimensionale per ogni elemento di catalogo $\Gamma_{im}$ con centro il punto relativo a $\Gamma_{im}$ stesso, se l'unione di tali sottospazi ha qualche buco interno si può verificare la situazione descritta. La soglia $th_i$ deve essere abbastanza grande da far sì che questo non accada.

In ogni caso l'unione delle ipersfere genera un sottospazio limitato, poichè il numero di elementi di catalogo e $th_i$ sono limitati. Questo non comporta un problema se il catalogo è abbastanza fornito da descrivere le tipologie per ciascuna componente. Nello spazio generato dai parametri di ciascuna componente, la "regione esterna", in cui un punto non viene associato ad alcun elemento di catalogo, può essere considerata come la regione in cui la componente non è ben definita. Ad esempio, una bocca i cui parametri corrispondono ad un punto collocato in questa cosiddetta "regione esterna", viene considerata come "non bocca", ovvero un dato rilevato come una certa componente ma che non dimostra caratteristiche accettabili per essere identificato come tale.

# Capitolo 5

# Confronto tra identikit

Il confronto tra identikit è il punto cruciale del processo di riconoscimento. Questa fase ha come input due identikit: quello relativo al soggetto da identificare e quello relativo all'utente dichiarato, recuperato dal sistema.

L'approccio che viene utilizzato va ad analizzare l'intersezione dei due identikit. Prima di enunciare l'algoritmo di confronto basato sull'intersezione, è necessario estendere il modello di identikit presentato nei capitoli precedenti.

La modifica proposta per la definizione dell'identikit, non influisce sulle scelte fatte nel capitolo precedente ma aggiunge informazione a tale modello, che sarà fondamentale in fase di confronto.

Il capitolo inoltre tratterà in dettaglio l'algoritmo di confronto e come è inserito nel processo di autenticazione.

## 5.1 Estensione del modello per l'identikit

La necessità di aggiungere informazione al modello previsto per l'identikit, nasce dal fatto che esso è composto da un insieme di elementi di catalogo, suddivisi per componente. Per ogni componente, quindi, vengono selezionati un certo numero di elementi del catalogo, ovvero quelli considerati "più vicini" all'immagine del soggetto rispetto alla metrica definita come *distanza*.

La struttura dell'identikit, però, non mantiene l'informazione relativa a "quanto vicini" sono gli elementi del catalogo ma tutti quelli selezionati vengono messi sullo stesso piano.

D'ora in poi il concetto di "vicinanza" verrà indicato con il termine *verosimiglianza*, in quanto la metrica utilizza è indicatrice della somiglianza tra immagine e elemento di catalogo in questione.

Il modello è quindi esteso associando a ciascun elemento di catalogo un *valore di verosimiglianza*. Riprendendo la descrizione formale descritta nella sezione 4.4, l'identikit può essere ridefinito in questo modo:

$$I(A) = \Big(I_1(A), I_2(A), ..., I_n(A)\Big)$$

$$\text{con:} \quad I_i(A) = \big\{\big(\Gamma_{im}, v_{im}(A)\big) \mid D_i(A, \Gamma_{im}) < th_i\big\} \quad \forall i \text{ t.c. } 1 \leq i \leq n$$

in cui $v_{im}(A)$ è la verosimiglianza di $\Gamma_{im}$ relativa all'immagine $A$.

Ogni identikit ha una struttura che rimane una $n$-upla, in cui ogni posizione è associata a una diversa componente e contiene coppie composte da un elemento di catalogo ed il valore di verosimiglianza associato.

## Verosimiglianza e logica fuzzy

Associare il valore di verosimiglianza a ciascun elemento del catalogo, è come considerare ciascun elemento di catalogo in logica fuzzy.

Per ciascun elemento di catalogo, si consideri la proprietà: *l'elemento di catalogo* $\Gamma_{im}$ *è somigliante al volto nell'immagine* $A$. Per il modello definito nei capitoli precedenti, tale proprietà sarebbe considerata in logica classica. Infatti si può stabilire che è *vera* per gli elementi di catalogo appartenenti all'identikit $I(A)$ e che è *falsa* per tutti gli altri.

Il passo che viene fatto è quello di assegnare un *grado di verità* a questa proprietà, che corrisponde proprio al significato dato al valore di verosimiglianza. In quest'ottica, dati un'immagine $A$, un elemento di catalogo $\Gamma_{im}$ e il valore di verosimiglianza associato $v_{im}(A)$, si può sostenere concettualmente che: $\Gamma_{im}$ *è somigliante con grado* $v_{im}(A)$ *al volto nell'immagine* $A$.

Per $\Gamma_{im} \notin I_i(A)$, il valore di verosimiglianza $v_{im}(A)$ è nullo per definizione, in quanto possiamo dire che tale elemento di catalogo non assomiglia per niente al volto in $A$.

Se invece $\Gamma_{im} \in I_i(A)$, $v_{im}(A)$ deve essere calcolato come valore compreso nell'intervallo $(0, 1]$ come per definizione di grado di verità nella logica fuzzy.

## Calcolo del valore di verosimiglianza

Il calcolo della verosimiglianza, per come tale valore è stato definito, deve rispettare i seguenti vincoli:

- $0 \leq v_{im} \leq 1 \qquad \forall\, i, m;$
- $\Gamma_{im} \notin I_i(A)$ se e solo se $v_{im} = 0$;
- dati due elementi di catalogo $\Gamma_{i1}$, $\Gamma_{i2}$ per una qualsiasi componente $i$-esima, se $D_i(A, \Gamma_{i1}) \geq D_i(A, \Gamma_{i2})$ allora $v_{i1} \leq v_{i2}$.

La soluzione più semplice da adottare è quella di considerare la distanza nell'intervallo $[0, th_i]$ (dove $th_i$ è la soglia per la distanza sulla componente $i$-esima) e far variare il grado di verosimiglianza lineramente in maniera tale che la massima verosimiglianza si abbia se la distanza è nulla, la minima per una distanza pari a $th_i$.

In questo modo si ottiene una relazione lineare tra verosimiglianza e distanza nell'intervallo di interesse, come quella rappresentata nel grafico in figura 5.1, che è rappresentata dalla funzione:

$$v_{im}(A) = \begin{cases} 1 - \frac{D_i(A,\Gamma_{im})}{th_i} & \text{se } D_i(A, \Gamma_{im}) < th_i, \\ 0 & \text{se } D_i(A, \Gamma_{im}) \geq th_i. \end{cases}$$

Il significato della verosimiglianza nell'iperspazio $k$-dimensionale generato dai $k$ parametri geometrici della singola componente del volto, corrisponde a considerare

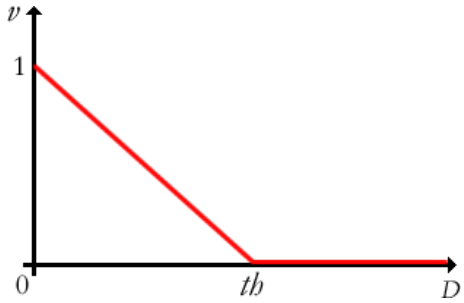


Figura 5.1: Verosimiglianza al variare della distanza.

un volume in cui ciascun punto ha un valore di verosimiglianza maggiore di zero. All'esterno di tale porzione di iperspazio, la verosimiglianza è pari a zero.

Prendendo in consierazione il caso più semplice in cui la distanza è euclidea, il volume rappresenta un'ipersfera che ha raggio $th_i$ e centro nel punto individuato dai parametri geometri caratterizzanti il volto del soggetto nell'immagine.

Nell'iperspazio sono posti anche i punti relativi agli elementi di catalogo per la componente in questione. I punti che si trovano all'interno dell'ipersfera corrispondono a quelli che compongono l'identikit e per i quali la verosimiglianza è diversa da zero.

Se i parametri geometrici sono due, l'ipersfera si riduce a un cerchio, come mostrato in figura 5.2 in cui il punto rosso indica il punto individuato dal soggetto, i punti blu corrispondono agli elementi di catalogo e il colore del cerchio indica la verosimiglianza con il colore nero che rappresenta il valore 1, il colore bianco il valore 0 e la scala di grigi i valori intermedi.

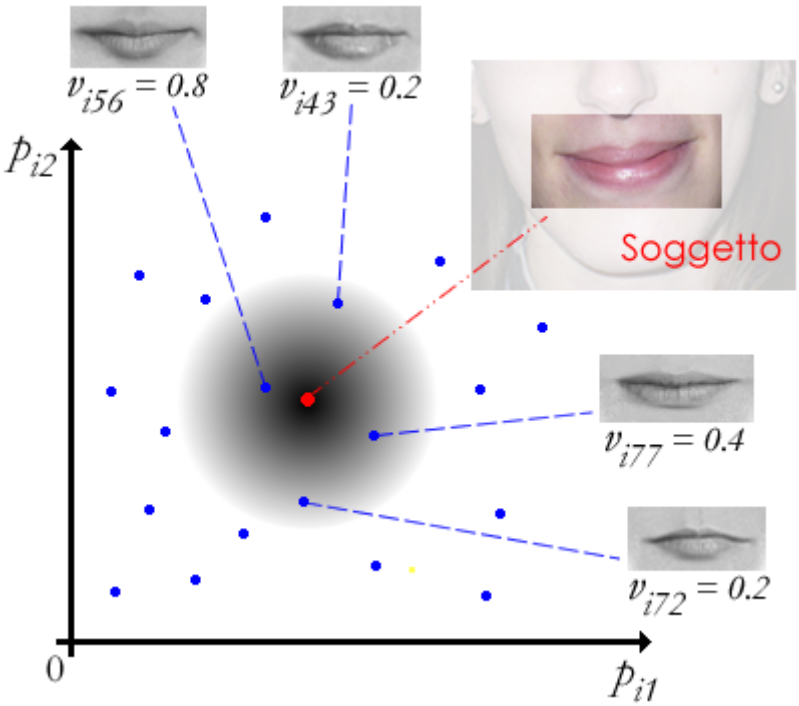


Figura 5.2: Verosimiglianza nello spazio di due parametri geometrici relativi a una singola componente.

## 5.2 Intersezione tra due identikit

Definito in maniera completa l'identikit, è possibile passare all'intersezione tra due, che corrisponde al primo passo del confronto. L'idea di fondo è che più ampia è l'intersezione, più somiglianti sono i due identikit.

Essendo ciascun identikit composto da un insieme di elementi di catalogo, è possibile considerare l'intersezione insiemistica tra due identikit. Questa è formata da tutti gli elementi di cataolgo che compaiono sia nell'uno, che nell'altro identikit.

Siano $I(A)$ e $I(B)$ gli identikit relativi a due soggetti generici $A$ e $B$. A ciascun elemento $\Gamma_{im}$ dell'intersezione tra $I(A)$ e $I(B)$ si vuole associare un valore che indichi il grado di somiglianza tra i due soggetti $A$ e $B$ a partire dalla verosimiglianza di ciascuno rispetto all'elemento di catalogo $\Gamma_{im}$.

Lo scopo è quello di partire da questi indici di somiglianza sui singoli elementi di catalogo per poi arrivare a un indice di somiglianza su ciascuna componente e infine mettere insieme il tutto per arrivare a un grado di somiglianza globale tra i due identikit. L'approccio risulta quindi gerarchico rispetto ai tre livelli:

1. livello di elemento di catalogo;
2. livello di componente del volto;
3. livello globale del volto.

L'intersezione e il calcolo del grado di somiglianza per ciascun elemento di tale insieme, è da considerarsi il passo da fare per il primo livello.

### Struttura dell'intersezione tra identikit

L'intersezione, pur avendo la stessa struttura di un identikit, non ha lo stesso significato. Infatti può essere anche una struttura vuota o comunque avere alcune componenti prive di elementi.

Inoltre non descrive più un soggetto ma il confronto tra soggetti (potenzialmente diversi). Anche il grado di verità associato a ciascun elemento di catalogo dell'intersezione ($\Gamma_{im}$) prende questo significato e, per distinzione, verrà chiamato come *grado di somiglianza* e indicato con $s_{im}$.

Siano $I(A)$ e $I(B)$ due identikit. La loro intersezione è così definita:

$$I(A) \cap I(B) = \Big( I_1(A) \cap I_1(B), I_2(A) \cap I_2(B), ..., I_n(A) \cap I_n(B) \Big)$$

$$\text{con:} \quad I_i(A) \cap I_i(B) = \Big\{ \Big( \Gamma_{im}, s_{im}\big(I(A), I(B)\big) \Big) \mid \Gamma_{im} \in I_i(A) \wedge \Gamma_{im} \in I_i(B) \Big\}$$

$$\forall i \text{ t.c. } 1 \leq i \leq n$$

in cui $s_{im}(A)$ è la somiglianza di $I(A)$ e $I(B)$ a livello dell'elemento $\Gamma_{im}$.

### Grado di somiglianza

Il grado di somiglianza nell'intersezione tra gli identikit $I(A)$ e $I(B)$ deve descrivere la somiglianza tra i relativi soggetti rispetto a un certo elemento di catalogo.

Scelta una componente ($i$) e un elemento dell'intersezione ($\Gamma_{im}$), ciascun identikit fornisce il proprio valore di verosimiglianza: ($v_{im}(A)$ e $v_{im}(B)$.

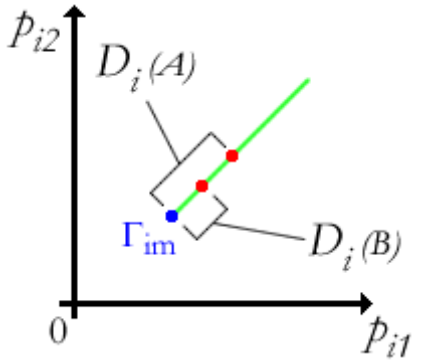


Figura 5.3: Caso migliore per la distanza tra $A$ e $B$.

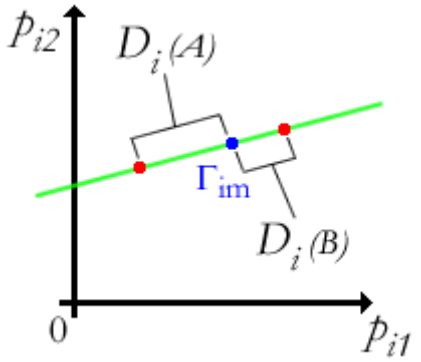


Figura 5.4: Caso peggiore per la distanza tra $A$ e $B$.

Il calcolo di questi due valori, è stato eseguito in base alle distanze tra i parametri geometrici della componente $i$-esima dei soggetti $A$ e $B$ e i parametri geometrici dell'elemento di catalogo $\Gamma_{im}$. Tali distanze sono indicate rispettivamente da $D_i(\Gamma_{im}, A)$ e $D_i(\Gamma_{im}, B)$.

Mantenendo lo stesso criterio, la misura di somiglianza tra $A$ e $B$ deve essere una misura coerente con la distanza $D_i(A, B)$ tra i parametri geometrici per la componente $i$-esima di $A$ e di $B$.

A partire dalle distanze $D_i(\Gamma_{im}, A)$ e $D_i(\Gamma_{im}, B)$, si può affermare che la distanza $D_i(A, B)$ nel caso migliore è pari a $|D_i(\Gamma_{im}, A) - D_i(\Gamma_{im}, B)|$ e accade nella situazione in cui nello spazio $k$-dimensionale dei $k$ parametri geometrici per la componente $i$-esima, i punti relativi ad $A$ e $B$ sono allineati su una stessa semiretta con origine nel punto relativo a $\Gamma_{im}$ (in figura 5.3 è raffigurato il caso per $k = 2$).

Il caso peggiore avviene invece quando i tre punti relativi ad $A$, $B$ e $\Gamma_{im}$ sono allineati su una stessa retta con $\Gamma_{im}$ nel segmento che unisce gli altri due punti (schematizzato in figura 5.4, sempre per $k = 2$). In tal caso, si ha che $D_i(A, B)$ è pari a $D_i(\Gamma_{im}, A) + D_i(\Gamma_{im}, B)$.

In uno spazio 1-dimensionale, questi due casi sono gli unici possibili. Non potendo risalire al caso in questione, si considera il valore medio della distanza.

$$D_i(A, B) = \frac{\big(D_i(\Gamma_{im}, A) + D_i(\Gamma_{im}, B)\big) + \big(|D_i(\Gamma_{im}, A) - D_i(\Gamma_{im}, B)|\big)}{2}$$
$$= \mathrm{MAX}\big(D_i(\Gamma_{im}, A)\ ,\ D_i(\Gamma_{im}, B)\big)$$

Passando da distanza a grado di somiglianza, si vuole utilizzare la stessa relazione usata per la verosimiglianza, ottenendo che la somiglianza tra $A$ e $B$ per l'elemento di catalogo $\Gamma_{im}$ è data dal valore minimo tra $v_{im}(A)$ e $v_{im}(B)$.

$$\begin{aligned} s_{im}\Big(I(A), I(B)\Big) &= 1 - \frac{D_i(A,B)}{th_i} \\ &= 1 - \frac{\mathrm{MAX}\big(D_i(\Gamma_{im}, A)\ ,\ D_i(\Gamma_{im}, B)\big)}{th_i} \\ &= \mathrm{MIN}\left(1 - \frac{D_i(\Gamma_{im}, A)}{th_i}\ ,\ 1 - \frac{D_i(\Gamma_{im}, B)}{th_i}\right) \\ &= \mathrm{MIN}\Big(v_{im}(A)\ ,\ v_{im}(B)\Big) \end{aligned}$$

In uno spazio $k$-dimensionale (con $k > 1$) il calcolo del valore medio risulta complicato in quanto l'informazione sulla distanza dall'elemento del catalogo fornisce un insieme di punti dello spazio $k$-dimensionale che corrisponde ad una ipersfera $k$-dimensionale centrata nel punto corrispondente ai parametri geometrici dell'elemento di catalogo $\Gamma_{im}$.

Infatti, fissati i punti $\Gamma_{im}$ nello spazio e (senza perdita di generalità) $A$ a distanza $D_i(\Gamma_{im}, A)$, sapendo che $B$ è distante $D_i(\Gamma_{im}, B)$ da $\Gamma_{im}$, $B$ può essere un punto qualsiasi appartenente all'ipersfera centrata in $\Gamma_{im}$ che ha raggio $D_i(\Gamma_{im}, B)$. La distanza media tra $A$ e $B$ dovrebbe essere calcolata quindi come media con $A$ fissato e $B$ che varia tra tutti i punti dell'ipersfera.

Per $k > 1$, dunque, si sceglie di calcolare il grado di somiglianza sempre come media tra caso migliore e caso peggiore, risultando così un'approssimazione ottimistica del valore medio calcolato su tutti i casi possibili.

## 5.3 Funzione di confronto finale

A partire dall'intersezione tra i due identikit, corredata di grado di somiglianza per ciascun elemento di catalogo, è necessario passare ai livelli successivi, ovvero a un indicatore di somiglianza per componente e successivamente a livello globale.

In questa organizzazione gerarchica, ogni livello sintetizza le informazioni del livello precedente fino a formare un indicatore per un elemento di tale livello.

### Da livello di elemento di catalogo a livello di componente

Si consideri l'insieme $I_i(A) \cap I_i(B)$ di elementi di catalogo dell'intersezione tra i due identikit $I(A)$ e $I(B)$ riferiti alla generica componente $i$-esima del volto. Tale insieme può essere l'insieme vuoto oppure può contenere un insieme di coppie $(\Gamma_{im}, s_{im})$ con $\Gamma_{im}$ elemento di catalogo e $s_{im}$ il grado somiglianza ad esso associato.

A partire da tale insieme, si vuole calcolare un grado di somiglianza riferito alla componente $i$-esima. È evidente che se $I_i(A) \cap I_i(B) = \emptyset$, il grado somiglianza debba essere nullo.

Nel caso l'insieme contenga almeno una coppia, una prima idea è quella di calcolare la media delle somiglianze sui singoli elementi di catalogo. Questo porta a un risultato assurdo: si pensi ad esempio al caso in cui si mette a confronto $I(A)$ con

se stesso. Banalmente si ottiene che l'intersezione è una struttura analoga ad $I(A)$ stesso. Il confronto dovrebbe dare un grado di somiglianza che in ogni componente tende ad 1. Se $I(A)$ contiene elementi di catalogo con verosimiglianza bassa, questi abbasseranno la media delle somiglianze portando a un risultato non propriamente corretto.

Si può invece considerare l'elemento di catalogo che si presume sia il punto d'incontro dei due identikit su ciascuna componente. Anche nel ragionamento umano, quando si hanno due descrizioni dello stesso soggetto e c'è un elemento comune e abbastanza sicuro da entrambi i lati, si è portati a pensare che le due descrizioni siano coerenti. Per questi motivi si sceglie come elemento di catalogo rappresentante ciascuna componente, quello comune più significativo, ovvero quello che ha grado di somiglianza maggiore.

A livello di componente, il risultato, che verrà indicato con $R(I(A), I(B))$, può essere descritto da una $n$-upla di coppie, in cui per ciascuna componente viene considerato l'elemento di catalogo scelto e il suo grado di somiglianza.

$$R\Big(I(A), I(B)\Big) = \Big(\big(\Gamma_{1m_1}, s_{1m_1}\big), \big(\Gamma_{2m_2}, s_{2m_2}\big), ..., \big(\Gamma_{nm_n}, s_{nm_n}\big)\Big)$$

$$\text{con:} \quad m_i \text{ t.c. } s_{im_i} = \text{MAX}\{s_{im} \mid (\Gamma_{im}, s_{im}) \in I_i(A) \cap I_i(B)\} \quad \forall i \text{ t.c. } 1 \leq i \leq n$$

## Da livello di componente a livello globale

Il passaggio al livello globale passa attraverso l'analisi del risultato precedente con lo scopo di ottenere un grado di somiglianza globale. Questa volta, però, si considera un insieme di valori dai significati differenti, ovvero corrispondenti a componenti diverse. Quindi ha senso sommare i contributi in quanto ogni contributo dà un'informazione aggiuntiva indipendente dall'altra. Per valutare la somiglianza globale si può utilizzare una funzione che sommi i gradi di somiglianza riscontrati per ciascuna componente.

Poichè le componenti hanno caratteristiche diverse, si dà la possibilità di associare un coefficiente a ciascuna componente in modo da dare più peso alle componenti significative per l'efficacia del riconoscimento.

Sia $\beta_i$ il coefficiente associato alla componente $i$-esima, si può definire la funzione che calcola la somiglianza globale in questo modo:

$$f\big(I(A), I(B)\big) = \sum_{i=1}^{n} \beta_i s_{im_i}$$

Per la scelta finale ancora una volta deve essere scelta una soglia $TH$ tale che:

- se $f\big(I(A), I(B)\big) > TH$, allora $I(A)$ e $I(B)$ rappresentano lo stesso soggetto e la risposta al problema di riconoscimento è *sì*;
- se $f\big(I(A), I(B)\big) \leq TH$, allora $I(A)$ e $I(B)$ non sono abbastanza somiglianti pertanto la risposta al problema di riconoscimento è *no*.

## 5.4 Confronto e procedura di autenticazione

In questa sezione vengono presentati una serie di algoritmi in pseudocodice che compongono la procedura di autenticazione, utlizzando una struttura per l'identikit più adatta al linguaggio informatico.

All'algoritmo principale, seguono una serie di funzioni ausiliarie che nel complesso assolvono ai vari passi descritti fino a questo punto.

**Algoritmo 1** *Autenticazione*

**Input:** un'immagine $A$ del soggetto che si dichiara come utente $u$.
**Output:** *sì/no* come risposta alla domanda di autenticazione.

1: $I_U \leftarrow CaricaIdentikit(u)$
2: $I_A \leftarrow NuovoIdentikitCR(A, I_U)$
3: $n \leftarrow$ numero di componenti di un identikit
4: Crea un nuovo array $R$ di dimensione $n$
5: **for** $i = 0 \rightarrow n - 1$ **do**
6: $L_U \leftarrow$ lista della componente $i$-esima di $I_U$
7: $L_A \leftarrow$ lista della componente $i$-esima di $I_A$
8: $k \leftarrow$ numero di elementi di $L_U$
9: Crea un nuovo array $S$ di dimensione $k$
10: **for** $j = 0 \rightarrow k - 1$ **do**
11: $\Gamma \leftarrow$ id dell'elemento $j$-esimo di $L_U$
12: **if** $\Gamma \in L_A$ **then**
13: $v_U \leftarrow$ verosimiglianza dell'elemento $\Gamma$ in $L_U$
14: $v_A \leftarrow$ verosimiglianza dell'elemento $\Gamma$ in $L_A$
15: $S[j] \leftarrow Somiglianza(v_U, v_A)$
16: **else**
17: $S[j] \leftarrow 0$
18: **end if**
19: **end for**
20: $R[i] \leftarrow Max(S)$
21: **end for**
22: Crea un nuovo array $\beta$ di dimensione $n$
23: $f \leftarrow 0$
24: **for** $i = 0 \rightarrow n - 1$ **do**
25: $\beta[i] \leftarrow$ peso relativo alla componente $i$-esima del volto
26: $f \leftarrow f + (\beta[i] \cdot R[i])$
27: **end for**
28: $TH \leftarrow$ soglia prefissata
29: **if** $f > TH$ **then**
30: **return** *sì*
31: **else**
32: **return** *no*
33: **end if**

L'algoritmo 1 verifica se l'utente relativo all'immagine $A$ è identificabile come utente $u$ del sistema.

L'identikit viene considerato come un insieme di liste, ciascuna associata ad una componente del volto. Ogni lista contiene l'insieme di elementi del catalogo associati al membro del sistema per la componente del volto che la lista rappresenta. Infine un elemento della lista è costituito da una coppia di valori corrispondenti all'id dell'elemento di catalogo e dal grado di verosimiglianza associato a tale elemento.

Inizialmente per l'utente del sistema, viene recuperato l'identikit $I_U$ costruito in fase di registrazione, mentre a partire dall'immagine $A$ del soggetto viene costruito un nuovo identikit $I_A$.

La struttura dell'algoritmo rispecchia quella dell'identikit appena descritta: il ciclo esterno (righe 5-21) determina la componente del volto che viene presa in considerazione, mentre il ciclo interno (righe 10-19) esamina ciascun elemento della lista associata.

Data una certa componente del volto $i$, vengono considerate le liste $L_U$ e $L_A$ relative a tale componente. Per ciascun elemento $\Gamma$ di $L_U$, viene calcolato il grado di somiglianza a livello di elemento di catalogo tra $I_U$ e $I_A$. Questo valore è posto a zero se la lista $L_A$ relativa al soggetto non presenta tale elemento. In caso contrario, il grado di somiglianza viene calcolato a partire dai gradi di verosimiglianza assegnati, nei due identikit $I_U$ e $I_A$, all'elemento di catalogo $\Gamma$ preso in considerazione.

Tra tutti gli elementi valutati per la stessa componente (array $S$), viene scelto il valore di somiglianza massimo e assegnato come grado di somiglianza alla componente in questione (array $R$).

Infine, i valori di somiglianza associati alle componenti vengono sommati tra loro e opportunamente pesati attraverso i coefficienti moltiplicativi dell'array $\beta$. Il valore che si ottiene, confrontato con la soglia $TH$, determina se l'utente viene accettato o meno come utente del sistema.

## Funzioni inserite nell'algoritmo principale

Negli algoritmi che seguono, non sono state considerate le situazioni eccezionali o problematiche che possono accadere nel corso dell'intera procedura. Saranno però analizzate in via descrittiva.

**Algoritmo 2** *CaricaIdentikit*

**Input:** $u$: dato che identifica un utente registrato nel sistema.
**Output:** $I_U$: identikit dell'utente $u$ recuperato dalla base di dati del sistema.

La funzione *CaricaIdentikit* (algoritmo 2) permette di recuperare l'identikit di un utente del sistema, a partire da un suo dato identificativo come ad esempio un codice oppure nome e cognome. La procedura potrebbe essere estesa facendo sì che gestisca il caso in cui l'input non corrisponda ad alcun utente del sistema, arrestando la procedura di autenticazione in quanto il dato inserito non risulta avere una corrispondenza tra gli utenti.

A differenza dell'algoritmo 2, *NuovoIdentikitCR* (algoritmo 3) crea un identikit a partire dall'immagine $A$, secondo il metodo descritto nel capitolo 4. Anche in questo caso si può estendere il metodo in modo che interrompa la procedura di autenticazione nel caso in cui non riesca a individuare un volto all'interno dell'immagine.

**Algoritmo 3** *NuovoIdentikitCR*

**Input:** l'immagine $A$ del soggetto e un catalogo ridotto ai soli elementi di $I_{CR}$.
**Output:** $I_A$: identikit del soggetto in $A$ creato sulla base del catalogo ridotto $I_{CR}$.

```
1: n ← numero di componenti di un identikit
2: Crea un nuovo identikit I_A come insieme di n liste vuote
3: for i = 0 → n − 1 do
4:    p_A ← EstraiParametriGeometrici(A, i)
5:    L_CR ← lista della componente i-esima di I_CR
6:    L_A ← lista della componente i-esima di I_A
7:    k ← numero di elementi di L_CR
8:    for j = 0 → k − 1 do
9:       Γ ← id dell'elemento j-esimo di L_CR
10:      v ← Verosimiglianza(Γ, p_A, i)
11:      if v > 0 then
12:         Inserisci in L_A la coppia (Γ, v)
13:      end if
14:   end for
15: end for
16: return I_A
```

Il metodo proposto introduce uno schema nuovo, ovvero cerca di rendere più efficiente la procedura di autenticazione evitando di considerare elementi di catalogo per l'identikit $I_A$ per i quali si può affermare a priori che non faranno parte dell'intersezione tra $I_A$ e l'identikit relativo all'utente $I_U$.

Ha senso ridurre il catalogo ai soli elementi dell'identikit $I_U$, poichè qualsiasi altro elemento di catalogo si possa considerare per $I_A$, certamente non farà parte dell'intersezione tra i due identikit.

Una conseguenza potrebbe essere quella di avere qualche componente del volto che non presenta elementi di catalogo pur essendo associata a un punto nello spazio dei parametri geometrici appartenente alla regione in cui si può considerare come componente *ben definita*.

**Algoritmo 4** *EstraiParametriGeometrici*

**Input:** un'immagine $A$ del volto di un soggetto e l'indice $i$ della componente in questione.
**Output:** l'array di parametri geometrici rilevati in $A$ rispetto alla componente $i$-esima.

L'algoritmo 4, *EstraiParametriGeometrici*, esegue tutte le operazioni necessarie al passaggio dall'immagine all'insieme di parametri del modello geometrico impostato per la componente specifica indicata. In questo metodo è inclusa anche la fase di *feature detection* che potrebbe dare esito negativo. In tal caso non si ha informazione sulla componente e si può trattare come nel caso in cui essa non venga considerata *ben definita*.

In questa logica, i parametri geometrici in output possono essere un valore speciale che indichi, all'algoritmo chiamante, che per tale componente venga restituita una lista vuota.

**Algoritmo 5** *CaricaParametriGeometrici*

**Input:** un elemento di catalogo $\Gamma$ e l'indice $i$ della componente in questione.

**Output:** l'array di parametri geometrici calcolati per l'elemento di catalogo $\Gamma$ relativo alla componente $i$-esima.

*CaricaParametriGeometrici* (algoritmo 5), a differenza dell'algoritmo 4, deve solo recuperare dati che sono già disponibili nella base di dati relativa agli utenti del sistema.

Infine gli algoritmi 6, 7, 8 eseguono rispettivamente il calcolo della *distanza* nello spazio dei parametri geometrici per la stessa componente, della *verosimiglianza* rispetto a un elemento di catalogo e della *somiglianza* tra due identikit a livello di elemento di catalogo.

**Algoritmo 6** *Distanza*

**Input:** due array di parametri $p_A, p_B$ e l'indice $i$ della componente a cui si riferiscono.

**Output:** la distanza tra $p_A$ e $p_B$.

1: $m \leftarrow$ numero di parametri geometrici per la $i$-esima componente del volto
2: Crea un nuovo array $\alpha$ di dimensione $k$
3: $d \leftarrow 0$
4: **for** $j = 0 \rightarrow k - 1$ **do**
5: $\quad \alpha[j] \leftarrow$ peso relativo al parametro $j$-esimo della componente $i$-esima
6: $\quad d \leftarrow d + (\alpha[j] \cdot (p_A[j] - p_B[j]))^2$
7: **end for**
8: **return** $\sqrt{d}$

**Algoritmo 7** *Verosimiglianza*

**Input:** un elemento di catalogo $\Gamma$, un array di parametri geometrici $p_A$ di un soggetto e l'indice $i$ che specifica la componente a cui si riferiscono $\Gamma$ e $p_A$.

**Output:** verosimiglianza del soggetto $A$ a cui si riferiscono i parametri in $p_A$, rispetto all'elemento di catalogo $\Gamma$.

1: $p_\Gamma \leftarrow CaricaParametriGeometrici(\Gamma, i)$
2: $d \leftarrow Distanza(p_A, p_\Gamma, i)$
3: $th \leftarrow$ soglia per la componente $i$-esima
4: **if** $d < th$ **then**
5: $\quad$ **return** $1 - \frac{d}{th}$
6: **else**
7: $\quad$ **return** $0$
8: **end if**

**Algoritmo 8** *Somiglianza*

**Input:** due valori di verosimiglianza $v_A$ e $v_B$, relativi allo stesso elemento di catalogo ma a due identikit distinti.

**Output:** la somiglianza tra i due identikit a livello di elemento di catalogo.

1: **if** $v_A < v_B$ **then**
2: **return** $v_A$
3: **else**
4: **return** $v_B$
5: **end if**

In questa trattazione della procedura di autenticazione, sono stati omessi i dettagli relativi alle modalità di elaborazione del'immagine in input $A$ per rilevare l'area relativa al volto e le regioni relative alle singole componenti.

Il modello, inoltre, rimane generico ovvero non viene presentato come dipendente da fattori come le componenti del volto scelte, i parametri geometrici, i valori delle costanti come soglie e coefficienti.

Nella realizzazione di un programma che permetta di eseguire le operazioni finora descritte, in base alle scelte che vengono fatte, devono essere sviluppate procedure specifiche. Ad esempio tutte le procedure che si riferiscono ad una specifica componente (algoritmi 4, 5, 6, 7) devono essere realizzate ad hoc per ciascuna di esse, poichè ciascuna presenta una struttura diversa.

# Capitolo 6

# Realizzazione del software

Il software è realizzato in linguaggio C++ in un insieme di file che nel corso del capitolo verranno descritti nel dettaglio.

Il programma implementa le due funzioni principali per il sistema di riconoscimento: la *registrazione* e l'*autenticazione*. Inoltre permette di costruire il catalogo a partire da immagini relative ai singoli elementi.

Per memorizzare i dati necessari al sistema, riguardanti gli utenti registrati e il catalogo, verranno utilizzati file di testo e sintassi predefinite. Per questo primo approccio pratico, non viene impiegata una base di dati strutturata che garantirebbe una gestione più efficiente degli stessi.

Come supporto, viene utilizzata la libreria *OpenCV* [2] nella fase di acquisizione delle immagini, di rilevamento del volto e delle sue componenti e nell'estrazione dei parametri geometrici.

## 6.1 Il programma principale

Il programma principale interagisce con l'utente esterno per determinare l'operazione da effettuare scegliendo tra tre possibilità:

1. Autenticazione di un soggetto che si dichiara utente del sistema attraverso nome e cognome e di cui viene rilevata l'immagine del volto.

2. Registrazione di un nuovo utente del sistema che fornisce la sua immagine e i sui dati (nome e cognome).

3. Aggiunta di elementi al catalogo.

Nel codice, questo viene sviluppato attraverso l'uso del terminale che propone un menù con le operazioni selezionabili. In base alla scelta viene eseguita una delle operazioni, al termine della quale viene riproposto il menù iniziale finchè l'utente non decide di terminare l'esecuzione.

All'inizio del programma viene inizializzato il catalogo come oggetto di una classe (*Catalog*) appositamente creata, la cui struttura verrà descritta in seguito.

Segue il codice del programma principale (codice 6.1) che corrisponde al metodo *main* all'interno del file *facerec.cpp*, in riferimento al quale verrà fatta una descrizione più dettagliata.

Codice 6.1: **facerec.cpp - main**

```
#include "identikit.h"
#include "cv.h"
#include "highgui.h"
#include <stdio.h>

int main(char** argv)
{
    //Costruzione del catalogo
    Catalog* cat = new Catalog();
    int op;
    do
    {
        //Reset della schermata
        if(system("CLS"))
            system("clear");

        //Scelta dell'operazione da eseguire
        printf("IDENTIFICAZIONE TRAMITE RICONOSCIMENTO DEL VOLTO E
            COSTRUZIONE DI IDENTIKIT\n\n");
        printf("Scegliere l'operazione da eseguire:\n 1) Accesso al
            sistema\n 2) Registrazione nuovo utente\n 3)
            Aggiornamento del catalogo\n 4) Termina\n");
        scanf("%d", &op);

        if(op == 1) //Richiesta di autenticazione
        {
            // Acquisizione dell'immagine
            char nm[20];
            char snm[20];
            char filename[20];
            printf("\nACCESSO AL SISTEMA\nImmagine da identificare
                (nome file): ");
            scanf("%s", filename);
            char imgaddress[50];
            sprintf(imgaddress, "db/foto/%s.bmp", filename);

            // Richiesta di nome e cognome
            printf("Nome dichiarato: ");
            scanf("%s", nm);
            printf("Cognome dichiarato: ");
            scanf("%s", snm);

            // Caricamento delle caratteristiche dell'utente
                dichiarato
            User* u = new User();
            bool exist = u->loadUser(nm, snm);

            if(exist) // Se l'utente e' stato trovato
            {
                // Costruzione dell'identikit rispetto al volto
                    nell'immagine
                Identikit* newidentikit = new Identikit(imgaddress,
                    cat, u->identikit);

                // Confronto tra i due identikit
                bool auth = identify(newidentikit, u->identikit);
```

```
50              if(auth)
51                  printf("\nAccesso autorizzato!\nBuongiorno %s %s,
                        utente numero %d!\n", nm, snm, u->id);
52              else
53                  printf("\nAccesso non autorizzato!\n");
54          }
55          else // Se l'utente non e' stato trovato
56          {
57              printf("\nL'utente non esiste!\n");
58          }
59          getchar();
60      }
61      else if(op == 2) // Registrazione nuovo utente
62      {
63          char nm[20];
64          char snm[20];
65          char filename[20];
66
67          // Richiesta dei dati
68          printf("\nREGISTRAZIONE NUOVO UTENTE\nNome: ");
69          scanf("%s", nm);
70          printf("Cognome: ");
71          scanf("%s", snm);
72
73          // Acquisizione dell'immagine
74          printf("Immagine (nome file): ");
75          scanf("%s", filename);
76          char imgaddress[50];
77          sprintf(imgaddress, "db/foto/registrazione/%s.bmp",
                filename);
78
79          // Costruzione dell'identikit e salvataggio dell'utente
80          Identikit* identikit = new Identikit(imgaddress, cat);
81          User* u = new User(nm, snm, identikit);
82          u->saveUser();
83      }
84      else if(op == 3) // Aggiornamento del catalogo
85      {
86          int comp;
87          int from;
88          int to;
89
90          // Scelta della componente e dell'intervallo di immagini
                da caricare
91          printf("\nAGGIORNAMENTO CATALOGO: digitare 1 per occhi, 2
                per naso, 3 per bocca. ");
92          scanf("%d", &comp);
93          printf("\nIntervallo immagini da caricare: ");
94          scanf("%d %d", &from, &to);
95
96          // Esecuzione dell'operazione richiesta
97          if(comp == 1)
98              insertNewEyes(from, to);
99          else if(comp == 2)
100             insertNewNoses(from, to);
101         else if(comp == 3)
102             insertNewMouths(from, to);
```

```
103
104             // Aggiornamento del catalogo
105             cat = new Catalog();
106         }
107         cvWaitKey(0);
108         cvDestroyAllWindows();
109         scanf("%*[^\n]");
110     }while(op != 4);
111     return 0;
112 }
```

## Autenticazione

La fase di autenticazione prevede la seguente sequenza di operazioni preliminari che necessitano l'interazione con l'utente esterno che deve essere autenticato.

Per prima cosa deve essere fornita un'immagine del soggetto stesso. In un sistema di riconoscimento, questa fase comporterebbe la rilevazione attraverso una fotografia scattata da un dispositivo connesso al sistema e che fornisce il dato automaticamente. In questo caso l'immagine, sempre corrispondente a una foto del volto, viene selezionata da una cartella del sistema in cui l'immagine è stata precedentemente salvata in formato *bitmap*. Per questo passo, deve solo essere indicato il nome del file in questione.

Il soggetto deve fornire i dati necessari a identificare l'utente che si dichiara di essere. Pertanto vengono richiesti nome e cognome, supponendo che non ci siano casi di omonimia. L'alternativa può essere quella di richiedere un codice identificativo, come ad esempio il numero di matricola se la base di utenti è composta da studenti che afferiscono a un polo universitario.

Eseguite queste operazioni preliminari, i dati dell'utente dichiarato possono essere caricati. Nel caso in cui i dati non corrispondano ad alcun profilo nella base di dati del sistema, l'output della funzione che carica l'utente sarà il valore booleano *falso*.

Se, dunque, i dati inseriti non sono corretti, la procedura di autenticazione non viene avviata e il problema viene segnalato a terminale. In caso contrario la procedura di autenticazione vera e propria può iniziare, secondo queste operazioni:

1. Viene calcolato l'identikit per il soggetto dell'immagine in input, tenendo conto di un catalogo ridotto ai soli elementi dell'identikit dell'utente dichiarato.

2. Viene verificata l'identità attraverso il confronto tra l'identikit appena creato e quello relativo all'utente.

3. Viene segnalato a terminale l'esito della verifica.

## Registrazione

La registrazione prevede ancora la rilevazione dei dati necessari alla definizione dell'utente. In questo caso vengono richiesti solamente nome e cognome ma la procedura potrebbe essere estesa per richiedere dati aggiuntivi in base alle esigenze del sistema. Inoltre deve essere acquisita l'immagine come base per la costruzione

dell'identikit dell'utente, operazione che viene effettuata fornendo nuovamente il nome del file che deve essere inserito in un'apposita cartella.

La registrazione viene effettuata in tre passi:

1. Viene creato l'identikit a partire dall'immagine caricata.

2. Viene creato un nuovo utente a partire dai dati inseriti dall'utente e dall'identikit appena costruito.

3. Viene salvato l'utente nella base di dati del sistema.

### Aggiornamento del catalogo

Il programma prevede l'opportunità di aggiornare il catalogo. In realtà questa operazione deve essere eseguita in fase di inizializzazione del sistema, ovvero prima di ogni registrazione o autenticazione. Il motivo è che la costruzione dell'identikit deve essere basata sullo stesso catalogo che si suppone completo fin dall'inizio.

Attraverso il terminale, vengono richiesti la componente per la quale si vogliono inserire nuovi elementi e l'intervallo di immagini da caricare. Ancora una volta si suppone che le immagini si trovino all'interno di una cartella specifica e che siano identificabili da un indice.

Anche il catalogo viene memorizzato su file e, a seconda della componente scelta, viene aggiornato il file relativo inserendo gli elementi corrispondenti all'intervallo indicato. Infine viene ricreato il catalogo in maniera tale che l'oggetto di classe *Catalog* venga aggiornato secondo le modifiche apportate.

A questa prima descrizione che non entra nei dettagli dell'implementazione ma che descrivendo il programma principale introduce quali sono le funzionalità richieste dall'intero sistema, seguirà l'analisi delle altre parti del software con un approccio *bottom-up*.

## 6.2 Gestione delle immagini con OpenCV

Per l'acquisizione e l'elaborazione di immagini vengono sfruttate le classi e i metodi della libreria *OpenCV*, software opensource realizzato anche nel linguaggio C++.

Segue una trattazione descrittiva non troppo dettagliata per le strutture dati e gli algoritmi utilizzati nel presente lavoro. Per i dettagli si rimanda alla documentazione della libreria.

### L'immagine

Un'immagine viene rappresentata da un oggetto di classe *IplImage*, che la rappresenta come matrice in cui ciascun elemento corrisponde a un pixel. L'immagine è caratterizzata da tre parametri:

- la dimensione, data da un parametro di tipo *CvSize* che rappresenta larghezza e altezza in pixel e che viene definita dal metodo *cvSize(altezza,larghezza)*.

- il numero di canali di colore, che può variare da 1 a 4. Ad esempio un'immagine in scala di grigi è rappresentabile con un solo canale che ne indica la luminanza, un'immagine a colori in formato BGR viene rappresentata da tre canali relativi ai colori blu, verde e rosso nell'ordine.

- la profondità in bit di ciascun pixel ovvero il numero di bit e quindi l'intervallo di valori che descrive ciascun pixel.

Per la creazione dell'immagine vengono utilizzati due metodi diversi. Il primo è *cvLoadImage* che carica un'immagine da file; il secondo, *cvCreateImage*, crea un'immagine vuota secondo i parametri stabiliti.

Per l'accesso al contenuto dell'immagine, viene utilizzato il metodo *cvGet2D* che, date le coordinate di un pixel all'inetrno dell'immagine a partire dall'angolo superiore sinistro, estrae il contenuto del pixel in un dato di tipo *CvScalar*. Questo permette di accedere ai valori del pixel su ciascun canale ($i$) considerato per l'immagine, attraverso l'attributo *val[i]* dell'oggetto di tipo *CvScalar*. Ogni attributo rappresenta un valore che si deve considerare in base alla profondità in bit dell'immagine.

Altri metodi che vengono utilizzati per la gestione dell'immagine nel suo complesso sono:

- *cvCopy*, per la copia di un'immagine.

- *cvCvtColor*, per la conversione dal formato RGB per il colore adottato nei file *bitmap* al formato in scala di grigi utilizzato nell'elaborazione dell'immagine.

- *cvReleaseImage*, per eliminare l'immagine, liberando lo spazio di memoria occupato.

## Punti e aree nel piano cartesiano

Ogni immagine si può considerare come un'area nel piano bidimensionale con gli assi a valori interi. Ogni punto del piano interno a tale area corrisponde a un pixel dell'immagine.

Per valutare le caratteristiche geometriche dell'immagine, vengono utilizzati oggetti che rappresentano punti e aree del piano per mezzo delle classi *CvPoint* e *CvRect*.

Un punto è definito dalla coppia di coordinate utilizzando il metodo *cvPoint(x,y)* o modificando gli attributi $x$ e $y$.

Un rettangolo è definito da quattro valori ovvero le due coordinate del vertice superiore sinistro, la larghezza e l'altezza che corrispondono rispettivamente agli attributi *x, y, width* e *height*. Anche un rettangolo può essere creato attraverso un metodo (*cvRect*) in cui vengono forniti in ingresso questi quattro parametri.

Ciò che è diverso rispetto ad una trattazione classica in un piano cartesiano, è l'orientamento degli assi. Data un'immagine, infatti, l'origine è considerata come l'estremo superiore sinistro con l'asse delle ascisse sul lato superiore orientato verso destra e l'asse delle ordinate sul lato sinistro orientato verso il basso, come schematizzato nell'immagine 6.1.

Punti e rettangoli sono fondamentali in fase di estrazione dei parametri del volto a partire dalle immagini, in quanto porzioni di immagini relative a una certa componente del volto sono rappresentate da rettangoli e all'interno di queste vengono poi individuati i punti significativi per il calcolo dei parametri geometrici prestabiliti.

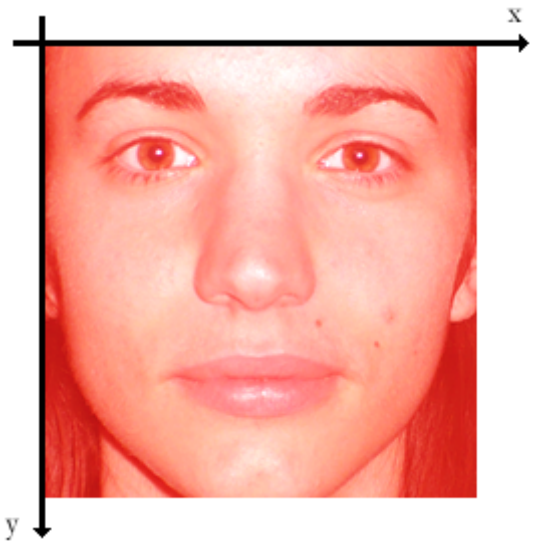


Figura 6.1: Orientamento degli assi.

## Metodi particolari di elaborazione delle immagini

All'interno delle specifiche procedure di rilevazione delle componenti del volto e di individuazione dei punti caratteristici, vengono utilizzati altri metodi che elaborano un'immagine. Segue una descrizione di tre algoritmi, il cui utilizzo all'interno delle diverse procedure viene descritto assieme alle procedure stesse.

- *cvSobel*: calcola la derivata dell'immagine eseguendo la convoluzione dell'immagine con una matrice (kernel) opportuna.

- *cvGoodFeaturesToTrack*: è una procedura di "corner detection" ovvero estrae un insieme di punti dell'immagine che ritiene appropriati come "spigoli" delle forme contenute nell'immagine.

- *cvHaarDetectObjects*: è una procedura basata su un classificatore che individua aree dell'immagine in cui è presente un oggetto, una forma o una componente associati al classificatore specifico.

## Finestre per la visualizzazione delle immagini

*OpenCV* fornisce anche un supporto utile per la visualizzazione delle immagini in finestre. Nel programma verranno create delle finestre attraverso il comando *cvNamedWindow* che verranno visualizzate con il comando *cvShowWindow* attraverso il quale viene assegnata un'immagine specifica alla finestra in questione.

Per la chiusura automatica verrà infine utilizzato il comando *cvDestroyAllWindows*. Questa istruzione è anticipata dal comando *cvWaitKey* che interrompe il programma finchè non viene premuto un tasto mentre è selezionata una delle finestre di *OpenCV* aperte.

## 6.3 Metodi ad hoc per l'elaborazione di immagini

Nell'elaborazione dell'immagine per l'estrazione dei parametri geometrici, vengono utilizzati dei metodi che sono stati sviluppati ad hoc. Questi vengono inseriti all'interno dello stesso file *utilities.cpp* che può essere importato all'occorrenza in quanto dispone di un header file relativo.

A questo primo gruppo di metodi, si aggiungono quelli relativi alle procedure di rilevamento del volto all'interno di un'immagine e delle singole componenti all'interno dell'immagine del volto. Questi metodi sono contenuti, invece, nel file *areasdetection.cpp*.

### Analisi dell'immagine per riga

Nella fase di estrazione dei parametri geometrici, viene analizzata l'immagine per riga, ovvero viene effettuata un'operazione sui pixel della stessa riga che ne associa un valore caratterizzante. Indipendentemente dalla funzione utilizzata nel calcolo di tale valore, il risultato dell'operazione sarà un vettore che ha come dimensione l'altezza dell'immagine.

I valori per riga che saranno utilizzati sono riferiti al processo di elaborazione di un'immagine in scala di grigi con profondità pari a 8bit, pertanto ogni pixel sarà rappresentato da un unico valore di luminanza contenuto nel primo canale (attributo *val[0]*) con un valore nell'intervallo $[0, 255]$. Le funzioni utilizzate sono:

- la somma dei valori dei pixel di una stessa riga, in cui l'apporto di ogni singolo elemento è dato dal valore di luminanza normalizzato nell'intervallo $[0, 1]$ rapportandolo con il valore massimo possibile (cioè 255, associato al colore bianco). Il metodo che esegue questa operazione è *sumRow* nel codice 6.2.

- la varianza dei valori dei pixel di riga normalizzati come nel caso precedente. Il metodo *varRow* (codice 6.3), che implementa la funzione, prima calcola il valore medio, poi la varianza.

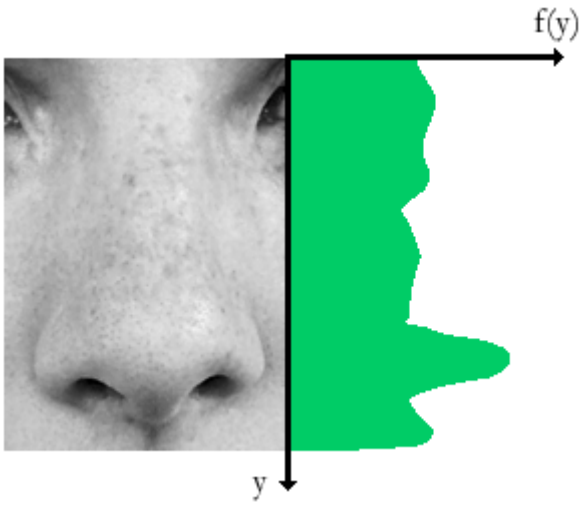


Figura 6.2: Istogramma relativo all'analisi di riga di un'immagine

Codice 6.2: **utilities.cpp - sumRow**

```
/* CALCOLO DELLA SOMMA PER RIGA NELL'IMMAGINE src IN INPUT.
 * TALI VALORI VENGONO MEMORIZZATI IN s[].
 */
void sumRow(IplImage* src, double s[])
{
    int w = src->width;
    int h = src->height;
    int i,j,c;
    for(i=0; i<h; i++) // Viene presa in esame un riga
    {
        // Calcolo della somma
        s[i] = 0;
        for(j=0; j<w; j++)
        {
            double pixel = (cvGet2D(src, i, j).val[0]/255);
            s[i] += pixel;
        }
    }
}
```

Codice 6.3: **utilities.cpp - varRow**

```
/* CALCOLO DELLA VARIANZA PER RIGA DELL'IMMAGINE src IN INPUT.
 * TALI VALORI VENGONO MEMORIZZATI IN v[].
 */
void varRow(IplImage* src, double v[])
{
    int w = src->width;
    int h = src->height;
    int i,j;
    double s,c;
    for(i=0; i<h; i++) // Viene presa in esame un riga
    {
        // Calcolo della media
        s = 0;
        for(j=0; j<w; j++)
        {
            double pixel = (cvGet2D(src, i, j).val[0]/255);
            s += pixel;
        }
        s = s/w;

        // Calcolo della varianza
        v[i] = 0;
        for(j=0; j<w; j++)
        {
            double pixel = (cvGet2D(src, i, j).val[0]/255);
            v[i] += pow(pixel - s, 2);
        }
        v[i] = v[i]/w;
    }
}
```

I risultati delle operazioni sulle righe, contenuti in un array a livello di programmazione, può essere rappresentato graficamente come un istogramma in cui ogni barra corrisponde alla funzione calcolata sulla riga. Un esempio è mostrato in figura 6.2.

Lo studio della conformazione dell'istogramma è utile nell'individuazione delle componenti e dei loro punti caratteristici. La forma dell'istogramma viene analizzata, infatti, attraverso il rilevamento di quelli che graficamenti si possono considerare "picchi", corrispondenti ai massimi relativi della funzione al variare dell'indice di riga.

Per questo viene definita la struttura *peak* (codice 6.4) che memorizza l'indice della riga relativa al massimo relativo (attributo *top*) e gli indici che corrispondono alle righe di inizio e fine del "picco" (attributi *start* e *end*) ovvero rispettivamente il punto di minimo relativo che precede e il punto di minimo relativo che segue il massimo considerato.

Attraverso il metodo *findNextPeak* (codice 6.5), dato un array e una posizione iniziale (*init*), viene cercato il primo picco a partire da tale posizione. Se viene individuato, viene assegnato all'oggetto *peak pkc* e viene dato *true* come output. In caso contrario il metodo ritorna il valore booleano *false*. Per fare questo, il metodo parte ad esaminare l'array a partire dalla posizione indicata come inizio, fino a trovare un minimo relativo che sarà l'inizio del picco. L'algoritmo prosegue cercando il successivo punto di massimo relativo e infine il minimo relativo che segue.

Codice 6.4: **utilities.cpp - struct peak**

```
4  typedef struct peak
5  {
6      int start;
7      int top;
8      int end;
9  };
```

Codice 6.5: **utilities.cpp - findNextPeak**

```
96  /* NELL'ARRAY s[] DI DIMENSIONE dim, A PARTIRE DALLA POSIZIONE
       init, VIENE CERCATO IL PRIMO PICCO E POSTO IN pk.
97   * IL METODO RITORNA false SE NESSUN PICCO VIENE INDIVIDUATO, true
       IN CASO POSITIVO.
98   */
99  bool findNextPeak(double s[], int dim, int init, peak* pk)
100 {
101     int i = init;
102
103     // Ricerca del primo minimo relativo
104     while((i+1 < dim) && (s[i] >= s[i+1]))
105        i++;
106     pk->start = i;
107
108     // Ricerca del primo massimo relativo
109     while((i+1 < dim) && (s[i] <= s[i+1]))
110        i++;
111     if(i+1 >= dim)
112     {
```

```
113         // Array terminato senza aver individuato il picco
114         return false;
115     }
116     else
117     {
118         // Trovato un massimo relativo
119         pk->top = i;
120 
121         // Ricerca del minimo relativo successivo
122         while((i+1 < dim) && (s[i] > s[i+1]))
123             i++;
124         pk->end = i;
125         return true;
126     }
127 }
```

Nell'analisi dei picchi, un problema che si presenta molto facilmente è quello di ottenere un istogramma che presenti minimi e massimi relativi molto vicini tra loro e con una variazione molto piccola. Anche queste piccole variazioni vengono identificate come tanti piccoli picchi che possono risultare un disturbo per l'analisi dell'istogramma, il quale dovrebbe essere graficamente "lisciato". Per ovviare a questo problema, viene filtrato l'array che implementa l'istogramma attraverso il metodo *smoothing* (codice 6.6) il quale esegue la convoluzione secondo un particolare nucleo gaussiano.

Codice 6.6: **utilities.cpp - smoothing**

```
72 /* ALL'ARRAY s[] DI DIMENSIONE dim VIENE APPLICATO UN FILTRO
       GAUSSAIANO MONODIMENSONALE.
73  * IL RISULTATO VIENE POSTO IN s2[].
74  */
75 void smoothing(double s[], double s2[], int dim)
76 {
77     int i,j;
78     double c;
79 
80     // Definizione del kernel gaussiano
81     double m[9] = {0.025, 0.075, 0.125, 0.150, 0.250, 0.150, 0.125,
           0.075, 0.025};
82     for(i=0; i<dim; i++) // Viene presa in esame un riga
83     {
84         s2[i] = 0;
85         int first = (i < 4) ? 0 : i-4;
86         int last = (i > dim-5) ? dim : i+5;
87 
88         // Applicazione del filtro
89         for(j=first; j<last; j++)
90         {
91             s2[i] += m[j-i+4]*s[j];
92         }
93     }
94 }
```

## Analisi di aree

Nell'individuazione dei punti, viene valutato l'intorno di un punto in un immagine calcolando il vlore medio dei pixel in un'area quadrata centrata nel punto. Questa valutazione viene fatta attraverso il metodo *avgArea* (codice 6.7) in cui è possibile determinare l'ampiezza dell'area da considerare: dato il parametro di ingresso *dim*, il quadrato avrà lato di lunghezza $2dim + 1$.

Codice 6.7: **utilities.cpp - avgArea**

```
129 /* CALCOLO DELLA MEDIA DEI VALORI DEI PIXEL DELL'IMMAGINE img
       NELL'INTORNO DEL PUNTO p DI DIMENSIONE dim.
130  */
131 double avgArea(IplImage* img, CvPoint p, int dim)
132 {
133     int w = img->width;
134     int h = img->height;
135     int x_init = (p.x >= dim) ? p.x-dim : 0;
136     int y_init = (p.y >= dim) ? p.y-dim : 0;
137     int x_end = (p.x+dim < w) ? p.x+dim : w-1;
138     int y_end = (p.y+dim < h) ? p.y+dim : h-1;
139     double s = 0;
140     int count = 0;
141     int i,j;
142     for(i=y_init; i<y_end; i++)
143     {
144         for(j=x_init; j<x_end; j++)
145         {
146             s += cvGet2D(img, i, j).val[0];
147             count++;
148         }
149     }
150     return s/count;
151 }
```

## Analisi geometrica

Una volta passati dall'immagine di una componete a una descrizione per punti per la stessa, è necessario calcolare i parametri geometrici. In questa fase, vengono coinvolti diversi metodi ausiliari inseriti nelle porzioni di codice 6.8, 6.9, 6.10 e 6.11.

- *midPoint*: dati due punti, ne calcola il punto medio.
- *distance*: calcola la distanza euclidea tra due punti.
- *isoscelesTriangleVertex*: dati due punti che identificano la base di un triangolo, calcola la coordinata $x$ del vertice rimanente conoscendo la coordinata $y$, in maniera tale da ottenere un triangolo isoscele.
- *triangleHeight*: dati tre punti che vengono considerati i vertici di un triangolo, calcola l'altezza del triangolo rispetto al primo vertice. Il segno associato a tale valore è determinato dalla posizione del primo vertice rispetto al lato limitato dai rimanenti vertici, come raffigurato nell'immagine 6.3. Se il vertice si trova sopra, il segno sarà positivo. In caso contrario il segno risulterà negativo.

Codice 6.8: **utilities.cpp - midPoint**

```
153 /* CALCOLO DI m COME PUNTO MEDIO TRA a E b.
154  */
155 void midPoint(CvPoint a, CvPoint b, CvPoint* m)
156 {
157    m->x = (a.x + b.x)/2;
158    m->y = (a.y + b.y)/2;
159 }
```

Codice 6.9: **utilities.cpp - distance**

```
161 /* CALCOLO DELLA DISTANZA EUCLIDEA TRA I PUNTI a E b.
162  */
163 double distance(CvPoint a, CvPoint b)
164 {
165    return sqrt(pow((a.x - b.x),2) + pow((a.y - b.y),2));
166 }
```

Codice 6.10: **utilities.cpp - isoscelesTriangleVertex**

```
168 /* CALCOLO DELLA COORDINATA X DEL VERTICE c DI UN TRIANGOLO
       ISOSCELE CON a E b COME VERTICI ALLA BASE E y_c COME COORDINATA
       Y DI c.
169  */
170 int isoscelesTriangleVertex(CvPoint a, CvPoint b, int y_c)
171 {
172    int x_c_num = pow(b.x,2) - pow(a.x,2) + pow((b.y - y_c),2) -
          pow((a.y - y_c),2);
173    int x_c_den = 2*(b.x - a.x);
174    return (int)(x_c_num/x_c_den);
175 }
```

Codice 6.11: **utilities.cpp - triangleHeight**

```
177 /* CALCOLO DELL'ALTEZZA RISPETTO AL VERTICE a DEL TRIANGOLO
       DEFINITO DAI PUNTI a, b, c.
178  */
179 double triangleHeight(CvPoint a, CvPoint b, CvPoint c)
180 {
181    double ab = distance(a, b);
182    double bc = distance(b, c);
183    double ca = distance(c, a);
184    double p = (ab + bc + ca)/2;
185
186    // Attribuzione del segno
187    int sign = 1;
188    double m = (b.y - c.y)/(b.x - c.x);
189    double q = c.y - (m*c.x);
190    if(a.y > m*a.x + q)
191       sign = -1;
192
193    return sign*2*sqrt(p*(p-ab)*(p-bc)*(p-ca))/bc;
194 }
```

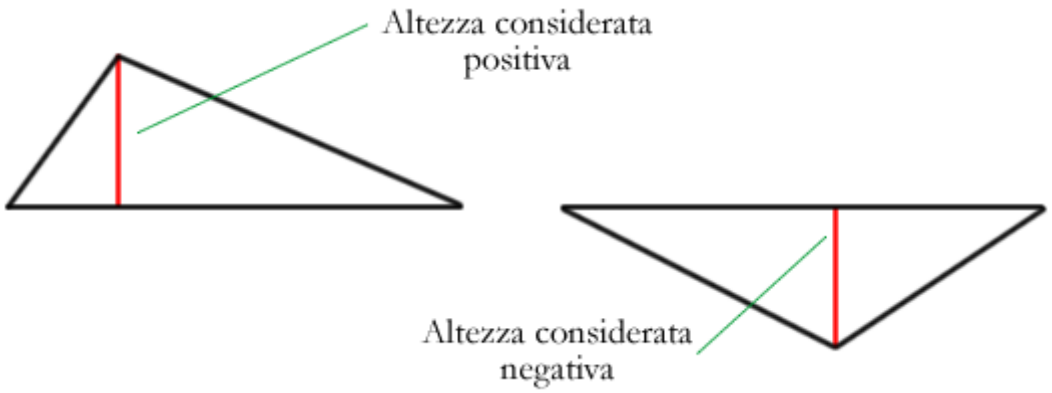


Figura 6.3: Segni associati all'altezza dei triangoli.

## Rilevamento del volto e delle sue componenti

Un'operazione preliminare rispetto all'estrazione dei parametri geometrici è il rilevamento del volto all'interno dell'immagine data in input dal sistema e della porzione del volto in cui sono presenti le diverse componenti.

Prendendo in considerazione l'immagine globale, in generale acquisita dal sistema e in questo caso caricata a partire da un file *bitmap*, non si possono conoscere a priori la posizione e la dimensione del volto. L'unica informazione disponibile è la presenza di un volto visto di fronte.

Per quanto riguarda le componenti del volto considerate (occhi, naso e bocca), data l'area del volto, si può supporre che abbiano una disposizione prefissata. Infatti, eseguendo una scansione dall'alto al basso dell'immagine del volto, si dovrebbero individuare occhi, naso e bocca nell'ordine.

Il risultato migliore con un'elaborazione del genere, sarebbe troppo grossolano poichè, per l'estrazione dei punti caratterizzanti ciascuna componente, è necessario arrivare a determinare un'area precisa che contenga esattamente la componente in questione.

Un'altro fattore di disturbo è senzaltro la possibilità che il volto abbia diverse rotazioni rispetto a un asse di riferimento parallelo al lato dell'immagine.

La risoluzione di questo problema richiede l'utilizzo di qualche tecnica avanzata nel campo dell'"object detection". *OpenCV* fornisce il metodo *cvHaarDetectObjects* (presentato nella sezione 6.2) che permette di definire un'area all'interno di un'immagine come volto, occhi, naso o bocca, attraverso l'uso di un classificatore specifico.

La stessa procedura viene utilizzata per tutti e quattro i casi attraverso i metodi *detectFace*, *detectEyes*, *detectNose*, *detectMouth*, rispettivamente in riferimento ai codici 6.12, 6.13, 6.14, 6.15.

Il risultato di tale operazione è una sequenza di rettangoli *CvRect*, tra i quali viene considerato solamente il primo risultato. Per esportare l'area rettangolare individuata in una nuova immagine, è stato sviluppato il metodo aggiuntivo *getSubImage* (codice 6.16).

Codice 6.12: **areasdetection.cpp - detectFace**

```
19 /* RILEVAMENTO DELLA PORZIONE DELL'IMMAGINE img CORRISPONDENTE A
       UN VOLTO.
20  * IL RISULTATO E' UN RETTANGOLO ALL'INTERNO DELL'IMMAGINE.
21  */
22 CvRect* detectFace(IplImage* img)
23 {
24     // Inizializzazione del classificatore
25     CvMemStorage* storage = 0;
26     CvHaarClassifierCascade* cascade = 0;
27     const char* cascade_name =
           ".../haarcascade_frontalface_alt2.xml";
28     cascade = (CvHaarClassifierCascade*)cvLoad(cascade_name, 0, 0,
           0);
29     storage = cvCreateMemStorage(0);
30     cvClearMemStorage(storage);
31
32     // Ricerca
33     CvRect* r;
34     if(cascade)
35     {
36         CvSeq* faces = cvHaarDetectObjects(img, cascade, storage,
               1.1, 3, 0, cvSize(30, 30));
37         if(faces && (faces->total == 1))
38             r = (CvRect*)cvGetSeqElem(faces, 0);
39         else
40             r = NULL;
41     }
42     else
43     {
44         r = NULL;
45     }
46     return r;
47 }
```

Codice 6.13: **areasdetection.cpp - detectEyes**

```
49 /* RILEVAMENTO DELLA PORZIONE DELL'IMMAGINE img CORRISPONDENTE AD
       OCCHI.
50  * IL RISULTATO E' UN RETTANGOLO ALL'INTERNO DELL'IMMAGINE.
51  */
52 CvRect* detectEyes(IplImage* img)
53 {
54     // Inizializzazione del classificatore
55     CvMemStorage* storage_e = 0;
56     CvHaarClassifierCascade* cascade_e = 0;
57     const char* cascade_name =
           ".../haarcascade_mcs_eyepair_small.xml";
58     cascade_e = (CvHaarClassifierCascade*)cvLoad(cascade_name, 0,
           0, 0);
59     storage_e = cvCreateMemStorage(0);
60     cvClearMemStorage(storage_e);
61
62     // Ricerca
63     CvRect* r;
64     if(cascade_e)
```

```
65      {
66          CvSeq* eyes = cvHaarDetectObjects(img, cascade_e, storage_e,
                1.1, 3, 0, cvSize(20, 10));
67          if(eyes)
68              r = (CvRect*)cvGetSeqElem(eyes, 0);
69          else
70              r = NULL;
71      }
72      else
73      {
74          r = NULL;
75      }
76      return r;
77  }
```

Codice 6.14: **areasdetection.cpp - detectNose**

```
79  /* RILEVAMENTO DELLA PORZIONE DELL'IMMAGINE img CORRISPONDENTE A
        UN NASO.
80   * IL RISULTATO E' UN RETTANGOLO ALL'INTERNO DELL'IMMAGINE.
81   */
82  CvRect* detectNose(IplImage* img)
83  {
84      // Inizializzazione del classificatore
85      CvMemStorage* storage_n = 0;
86      CvHaarClassifierCascade* cascade_n = 0;
87      const char* cascade_name = ".../haarcascade_mcs_nose.xml";
88      cascade_n = (CvHaarClassifierCascade*)cvLoad(cascade_name, 0,
            0, 0);
89      storage_n = cvCreateMemStorage(0);
90      cvClearMemStorage(storage_n);
91
92      // Ricerca
93      CvRect* r;
94      if(cascade_n)
95      {
96          CvSeq* noses = cvHaarDetectObjects(img, cascade_n,
                storage_n, 1.1, 3, 0, cvSize(30, 30));
97          if(noses)
98              r = (CvRect*)cvGetSeqElem(noses, 0);
99          else
100             r = NULL;
101     }
102     else
103     {
104         r = NULL;
105     }
106     return r;
107 }
```

Codice 6.15: **areasdetection.cpp - detectMouth**

```
109 /* RILEVAMENTO DELLA PORZIONE DELL'IMMAGINE img CORRISPONDENTE AD
        UNA BOCCA.
110  * IL RISULTATO E' UN RETTANGOLO ALL'INTERNO DELL'IMMAGINE.
```

```
111  */
112 CvRect* detectMouth(IplImage* img)
113 {
114    // Inizializzazione del classificatore
115    CvMemStorage* storage_m = 0;
116    CvHaarClassifierCascade* cascade_m = 0;
117    const char* cascade_name = ".../haarcascade_mcs_mouth.xml";
118    cascade_m = (CvHaarClassifierCascade*)cvLoad(cascade_name, 0,
          0, 0);
119    storage_m = cvCreateMemStorage(0);
120    cvClearMemStorage(storage_m);
121
122    // Ricerca
123    CvRect* r;
124    if(cascade_m)
125    {
126       CvSeq* mouths = cvHaarDetectObjects(img, cascade_m,
             storage_m, 1.1, 3, 0, cvSize(50, 20));
127       if( mouths )
128          r = (CvRect*)cvGetSeqElem(mouths, 0);
129       else
130          r = NULL;
131    }
132    else
133    {
134       r = NULL;
135    }
136    return r;
137 }
```

Codice 6.16: **areasdetection.cpp - getSubImage**

```
139 /* ESTRAZIONE DI UNA SOTTOIMMAGINE CORRISPONDENTE ALL'AREA DEL
       RETTANGOLO r NELL'IMMAGINE src.
140  */
141 IplImage* getSubImage(IplImage* src, CvRect* r)
142 {
143    CvPoint center;
144    center.x = cvRound((r->x + r->width*0.5));
145    center.y = cvRound((r->y + r->height*0.5));
146    IplImage* dst = cvCreateImage(cvSize(r->width, r->height), 8,
          1);
147    cvGetRectSubPix(src, dst, cvPointTo32f(center));
148    return dst;
149 }
```

## 6.4 Parametri geometrici

Data un'immagine che si suppone essere un'area rettangolare che contiene una data componente, l'estrazione dei parametri geometrici è un processo che prevede due fasi interne. La prima parte dall'immagine e individua un insieme prefissato di punti caratterizzanti, come descritto nel modello geometrico proposto. La seconda

fase, a partire dai punti, calcola i parametri geometrici per la componente in questione, associandoli alla descrizione del soggetto (descrizione non definitiva in quanto vi è un passaggio successivo cioè la costruzione dell'identikit).

I parametri geometrici di ciascuna componente, vengono strutturati come una classe in cui gli attributi sono proprio gli stessi parametri. Un oggetto della classe corrisponde alla descrizione geometrica associata a un'immagine di partenza. L'immagine può essere associata a un soggetto (in fase di registrazione o autenticazione) oppure ad un elemento del catalogo.

Avendo modelli diversi tra loro, le tre componenti vengono implementate attraverso tre classi diverse: *EyesFeatures*, *NoseFeatures* e *MouthFeatures* con riferimento ai codici 6.17, 6.18 e 6.19 rispettivamente. Nelle stesse porzioni di codice, sono inseriti i metodi costruttori e di assegnazione dei parametri.

Codice 6.17: **eyes.cpp - class EyesFeatures**

```
8  #define MAX_EYES_DISTANCE (0.1)
9
10 /* CLASSE EYESFEATURES
11  * Ha come attributi i parametri geometrici impostati per gli
        occhi.
12  * I metodi permettono di estrarre e modificare tali parametri.
13  */
14 class EyesFeatures
15 {
16    public:
17       double upperEyelid;
18       double lowerEyelid;
19       double eyeAxisDirection;
20       EyesFeatures();
21       void setValues(double ue, double le, double ead);
22       double getEyesFeatures(IplImage* img, IplImage* dst,
            CvPoint* ext);
23       void calcValues(CvPoint* ext, CvPoint* in, CvPoint* up,
            CvPoint* down);
24 };
25
26 double compareEyes(EyesFeatures* a, EyesFeatures* b);
27 double calcEyesLikelihood(EyesFeatures* a, EyesFeatures* b);
28
29 /* INIZIALIZZAZIONE SENZA PARAMETRI.
30  */
31 EyesFeatures::EyesFeatures()
32 {
33    eyeAxisDirection = 0;
34    upperEyelid = 0;
35    lowerEyelid = 0;
36 }
37
38 /* ASSEGNAZIONE DEI PARAMETRI IN INPUT.
39  */
40 void EyesFeatures::setValues(double ue, double le, double ead)
41 {
42    eyeAxisDirection = ead;
43    upperEyelid = ue;
44    lowerEyelid = le;
```

```
45 }
```

Codice 6.18: **nose.cpp - class NoseFeatures**

```
8  #define MAX_NOSE_DISTANCE (0.1)
9
10 /* CLASSE NOSEFEATURES
11  * Ha come attributi i parametri geometrici impostati per il naso.
12  * I metodi permettono di estrarre e modificare tali parametri.
13  */
14 class NoseFeatures
15 {
16    public:
17       double width;
18       double base;
19       NoseFeatures();
20       void setValues(double w, double b);
21       double getNoseFeatures(IplImage* img, IplImage* dst,
            CvPoint* rif);
22       void calcValues(CvPoint* ext, CvPoint* cent);
23 };
24
25 double compareNose(NoseFeatures* a, NoseFeatures* b);
26 double calcNoseLikelihood(NoseFeatures* a, NoseFeatures* b);
27
28 /* INIZIALIZZAZIONE SENZA PARAMETRI.
29  */
30 NoseFeatures::NoseFeatures()
31 {
32    width = 0;
33    base = 0;
34 }
35
36 /* ASSEGNAZIONE DEI PARAMETRI IN INPUT.
37  */
38 void NoseFeatures::setValues(double w, double b )
39 {
40    width = w;
41    base = b;
42 }
```

Codice 6.19: **mouth.cpp - class MouthFeatures**

```
8  #define MAX_MOUTH_DISTANCE (0.05)
9
10 /* CLASSE MOUTHFEATURES
11  * Ha come attributi i parametri geometrici impostati per la bocca.
12  * I metodi permettono di estrarre e modificare tali parametri.
13  */
14 class MouthFeatures
15 {
16    public:
17       double supLip;
18       double infLip;
19       MouthFeatures();
```

```
20        void setValues(double s, double i);
21        void getMouthFeatures(IplImage* img, IplImage* dst, CvPoint*
             rif, double* d);
22        void calcValues(CvPoint* ext, CvPoint* cent);
23 };
24
25 double compareMouth(MouthFeatures* a, MouthFeatures* b);
26 double calcMouthLikelihood(MouthFeatures* a, MouthFeatures* b);
27
28 /* INIZIALIZZAZIONE SENZA PARAMETRI.
29  */
30 MouthFeatures::MouthFeatures()
31 {
32    supLip = 0;
33    infLip = 0;
34 }
35
36 /* ASSEGNAZIONE DEI PARAMETRI IN INPUT.
37  */
38 void MouthFeatures::setValues(double s, double i)
39 {
40    supLip = s;
41    infLip = i;
42 }
```

Ciascuna classe comprende anche i metodi che permettono di impostare i parametri geometrici. Questi verranno presentati componente per componente.

## Calcolo dei parametri geometrici per gli occhi

Data in ingresso l'immagine degli occhi, la prima operazione che viene eseguita è quella di "tagliarla a metà", ovvero considerare due nuove immagini relative all'occhio sinistro e all'occhio destro. Per ciascuna di esse verranno effettuate le stesse operazioni. La ragione che porta a separare le due immagini è che, attraverso i metodi presentati nella sezione 6.3, devono essere eseguite delle operazioni per riga al fine di calcolarne i limiti superiore e inferiore. In generale non è vero che i due occhi sono perfettamente allineati orizzontalmente, di conseguenza i limiti non sono relativi alla stessa riga e i due occhi devono essere considerati separatamente.

In ogni occhio si vogliono individuare quattro punti che definiscano i due triangoli isosceli con base in comune, in riferimento al modello proposto nel capitolo 4.

La prima operazione che viene eseguita è proprio quella accennata: si vuole trovare un limite superiore e inferiore all'occhio in maniera tale da determinare:

- la coordinata $y$ del punto superiore;
- la coordinata $y$ del punto inferiore;
- un intervallo di righe dell'immagine da considerare come area in cui individuare i punti esterno e interno.

Viene quindi calcolato il vettore che prende in considrazione la varianza dei pixel in ogni riga attraverso il metodo *varRow*, al quale viene successivamente applicato un

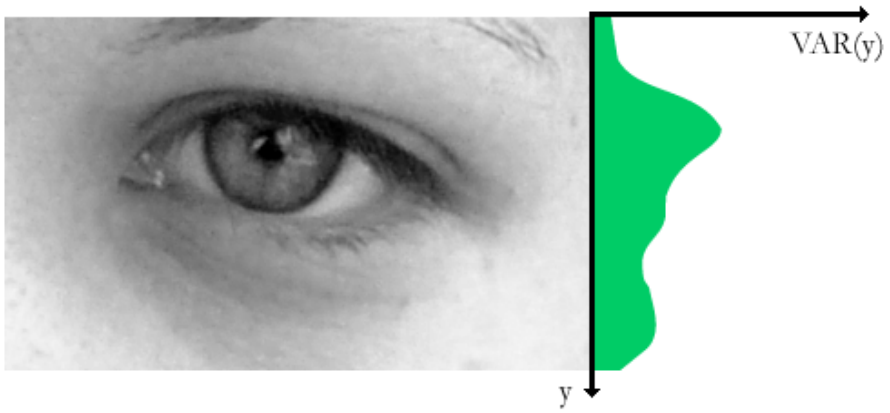


Figura 6.4: Istogramma relativo all'analisi dell'occhio.

filtro gaussiano attraverso il metodo *smoothing*. Ciò che si ottiene è un istogramma che presenta un picco in corrispondenza dell'occhio, come mostrato in figura 6.4.

Il modo in cui viene cercato il picco relativo all'occhio è la scansione del vettore alla ricerca dei picchi che sono presenti, con l'obiettivo di individuare il picco "più alto" ovvero quello in cui l'attributo *top* determina la posizione dell'array in cui il valore è massimo. Questa ricerca viene implementata con un ciclo in cui vengono cercati picchi consecutivi attraverso il metodo *findNextPeak* con indice iniziale che, di volta in volta, corrisponde alla posizione finale dell'ultimo picco individuato. Confrontando tra loro i massimi relativi, viene mantenuto il picco con massimo relativo maggiore che, alla fine del ciclo, corrisponde al picco che contiene il massimo valore dell'array.

Il picco selezionato fornisce la posizione approssimativa dell'occhio. Per determinare il limite superiore, a partire dalla posizione data dall'attributo *top* del picco massimo, si ripercorre l'array all'indietro (considerando righe superiori) fino ad arrivare a un limite in cui il valore dell'array scende al di sotto di una soglia data da una percentuale del valore massimo precedentemente individuato. Lo stesso procedimanto viene applicato per il limite inferiore, sempre partendo dalla posizione del massimo, passando a indici di riga superiori e quindi scendendo nell'immagine fino ad un limite analogamente fissato. Le percentuali utilizzate sono state determinate empiricamente.

Alla fine di questa prima parte, si individuano due limiti per cisascun occhio che corrispondono alle variabili *l_up*, *l_down* per l'occhio sinistro, *r_up*, *r_down* per il destro. Tali valori saranno le ordinate dei punti superiore e inferiore dei due occhi. L'area in cui individuare i punti rimanenti, è data dall'estensione della fascia determinata dai due limiti da entrambi i lati, di una percentuale rispetto alla propria dimensione (rappresentata dalla variabile *offset*).

Per il rilevamento dei punti rimanenti, viene utilizzata la funzione di "corner detection" *cvGoodFeaturesToTrack* in cui vengono impostati parametri, con valori scelti empiricamente in modo tale che i punti esterno e interno dell'occhio compaiano tra i risultati. Infatti, la funzione restituisce un insieme di punti dell'immagine che la funzione valuta come "spigoli".

Il risultato viene affinato cercando di considerare solamente i punti significativi che appartengono all'occhio. Con un ciclo che scorre ciascun risultato, si considera

un punto solo se appartiene alla fascia orizzontale definita nel passo precedente e se l'intorno del pixel corrisponde a un bordo vero e proprio. I punti individuati esternamente all'occhio sono caratterizzati da un basso valore di variazione nell'intorno, di conseguenza questo controllo permette di escludere un certo numero di falsi positivi. L'implementazione del controllo usa la funzione *avgArea* descritta in precedenza, che calcola il valore medio dei pixel nell'intorno del punto dato, a partire dall'immagine alla quale viene applicato un filtro di Sobel. Questo crea una nuova immagine in cui vengono esaltati i contorni delle forme, pertanto la valutazione dell'intorno di un punto in questa immagine dà un riscontro sull'appartenenza o meno a un bordo nell'immagine iniziale. Ancora una volta viene determnato un valore di soglia opportuno per il risultato della funzione *avgArea*.

I punti che superano positivamente questo controllo, si suppongono appartenenti all'occhio. Pertanto, da questo sottoinsieme selezionato, vengono estratti il punto con ascissa maggiore e quello con ascissa minore che individuano l'estremo interno e quello esterno dell'occhio in questione.

Si osserva che, affinchè i punti si riferiscano all'immagine completa degli occhi, ai punti individuati per l'occhio sinistro deve essere aggiunto un offset all'ascissa pari alla dimensione della porzione di immagine (circa metà) relativa all'occhio destro.

Alla fine di questa seconda fase dell'estrazione dei punti significativi per la costruzione del modello, si hanno il punto più esterno dell'occhio, quello più interno e le due ordinate per i punti inferiore e superiore di ciascun occhio. I rimanenti valori da calcolare sono le ascisse degli ultimi due punti elencati.

Si osserva, che per rispettare il modello a "triangoli isosceli", il vertice superiore deve avere la stessa distanza dai due punti della base ovvero i punti interno ed esterno. Data l'ordinata per tale punto, esiste un solo valore per l'ascissa che rispetti questo vincolo del modello. Per valutare le ascisse di tali punti, viene utilizzato il metodo *isoscelesTriangleVertex* (codice 6.10).

Determinati i quattro punti per ciascun occhio, si calcolano i parametri geometrici attraverso il metodo *calcValues* che verrà descritto in seguito. Infine viene fornito come output il rapporto tra la distanza tra i due punti più interni e quella tra i due più esterni degli occhi. Questo valore non rientra nel modello per la componente ma può essere utilizzato come parametro globale, pertanto viene fornito alla procedura chiamante considerando che la descrizione dell'occhio per punti non viene memorizzata e tale parametro non sarebbe più calcolabile se non ripetendo la procedura fin qui descritta.

Codice 6.20: **eyes.cpp - getEyesFeatures**

```
47 /* ESTRAZIONE DEI PARAMETRI GEOMETRICI A PARTIRE DALL'IMMAGINE img.
48  * I PUNTI RILEVATI VENGONO INDICATI NELL'IMMAGINE dst.
49  * ext E' LA COPPIA DI PUNTI RILEVATI COME ESTREMI ESTERNI.
50  * IL METODO FORNISCE IN OUTPUT IL PARAMETRO GLOBALE ASSOCIATO
        AGLI OCCHI.
51  */
52 double EyesFeatures::getEyesFeatures(IplImage* img, IplImage* dst,
       CvPoint* ext)
53 {
54     cvCopy(img, dst, NULL);
55     int w = img->width;
56     int h = img->height;
```

```
57    CvPoint in[2];
58    CvPoint up[2];
59    CvPoint down[2];
60
61    CvScalar target_color[3] = {{{255,255,255,255}}, {{0,0,0,0}},
          {{128,128,128,128}}};
62    int radius = 1;
63
64    // Divisione dell'immagine in due immagini distinte per occhio
          sinistro e occhio destro
65    int rw = (int)(w/2);
66    int lw = w - rw;
67    CvRect l_eye_rect = cvRect(rw, 0, lw, h);
68    CvRect r_eye_rect = cvRect(0, 0, rw, h);
69    IplImage* l_eye = getSubImage(img, &l_eye_rect);
70    IplImage* r_eye = getSubImage(img, &r_eye_rect);
71
72    // Ricerca degli estremi superiore e inferiore dell'occhio
          sinistro
73    double l_row_var[h];
74    double l_row_var2[h];
75    varRow(l_eye, l_row_var2);
76    smoothing(l_row_var2, l_row_var, h);
77    peak pkc;
78    peak pkinit = {0, (int)(h/2), h};
79    peak pk1 = pkinit;
80    int init = 0;
81    bool found;
82    do
83    {
84       found = findNextPeak(l_row_var, h, init, &pkc);
85       if(l_row_var[pkc.top] > l_row_var[pk1.top])
86       {
87          pk1 = pkc;
88       }
89       init = pkc.end;
90    }while(found && (pkc.end < h));
91    int l_up = pk1.top;
92    int l_down = pk1.top;
93    double top_val = l_row_var[pk1.top];
94    while(((l_up - 1) >= 0) && (l_row_var[l_up-1] > (top_val*0.6)))
95       l_up--;
96    while(((l_down + 1) < h) && (l_row_var[l_down+1] >
          (top_val*0.3)))
97       l_down++;
98
99    // Ricerca degli estremi superiore e inferiore dell'occhio
          destro
100   double r_row_var[h];
101   double r_row_var2[h];
102   varRow(r_eye, r_row_var2);
103   smoothing(r_row_var2, r_row_var, h);
104   pk1 = pkinit;
105   init = 0;
106   do
107   {
108      found = findNextPeak(r_row_var, h, init, &pkc);
```

```
109         if(r_row_var[pkc.top] > r_row_var[pk1.top])
110         {
111            pk1 = pkc;
112         }
113         init = pkc.end;
114      }while(found && (pkc.end < h));
115      int r_up = pk1.top;
116      int r_down = pk1.top;
117      top_val = r_row_var[pk1.top];
118      while((r_up-1 >= 0) && (r_row_var[r_up-1] > (top_val*0.6)))
119         r_up--;
120      while((r_down+1 < h) && (r_row_var[r_down+1] > (top_val*0.3)))
121         r_down++;
122
123      // Parametri della funzione di "corner detection"
124      const int MAX_CORNERS = 20;
125      double quality_level = 0.001;
126      double min_distance = (int)(w/20);
127      int eig_block_size = 5;
128      int use_harris = false;
129
130      IplImage* l_eig_image = cvCreateImage(cvSize(lw,h),
            IPL_DEPTH_32F, 1);
131      IplImage* l_temp_image = cvCreateImage(cvSize(lw,h),
            IPL_DEPTH_32F, 1);
132      IplImage* r_eig_image = cvCreateImage(cvSize(rw,h),
            IPL_DEPTH_32F, 1);
133      IplImage* r_temp_image = cvCreateImage(cvSize(rw,h),
            IPL_DEPTH_32F, 1);
134
135      // Ricerca degli estremi laterali nell'occhio sinistro
136      CvPoint2D32f l_corners[MAX_CORNERS] = {0};
137      int l_corner_count = MAX_CORNERS;
138      cvGoodFeaturesToTrack(l_eye, l_eig_image, l_temp_image,
            l_corners, &l_corner_count, quality_level, min_distance,
            NULL, eig_block_size, use_harris);
139      CvPoint p_l_int = cvPoint(w, 0);
140      CvPoint p_l_ext = cvPoint(0, 0);
141      IplImage* l_edges = cvCreateImage(cvSize(lw,h), 8, 1);
142      cvSobel(l_eye, l_edges, 0, 1, 3);
143      for(int i=0; i<l_corner_count; i++)
144      {
145         CvPoint p = cvPoint((int)(rw + l_corners[i].x + 0.5f),
               (int)(l_corners[i].y + 0.5f));
146         CvPoint p2 = cvPoint((int)(l_corners[i].x + 0.5f),
               (int)(l_corners[i].y + 0.5f));
147         double offset = (l_down - l_up)*0.2;
148         if((p.y >= l_up-offset) && (p.y <= l_down+offset) &&
               (avgArea(l_edges, p2, 6) >= 20))
149         {
150            cvCircle(dst, p, radius, target_color[1]);
151            if(p.x <= p_l_int.x)
152               p_l_int = p;
153            if(p.x >= p_l_ext.x)
154               p_l_ext = p;
155         }
156         else
```

```
157         cvCircle(dst, p, radius, target_color[2]);
158     }
159
160     // Ricerca degli estremi laterali nell'occhio destro
161     CvPoint2D32f r_corners[MAX_CORNERS] = {0};
162     int r_corner_count = MAX_CORNERS;
163     cvGoodFeaturesToTrack(r_eye, r_eig_image, r_temp_image,
            r_corners, &r_corner_count, quality_level, min_distance,
            NULL, eig_block_size, use_harris);
164     CvPoint p_r_ext = cvPoint(w, 0);
165     CvPoint p_r_int = cvPoint(0, 0);
166     IplImage* r_edges = cvCreateImage(cvSize(rw,h), 8, 1);
167     cvSobel(r_eye, r_edges, 0, 1, 3);
168     for(int i=0; i<r_corner_count; i++)
169     {
170         CvPoint p = cvPoint((int)(r_corners[i].x + 0.5f),
                (int)(r_corners[i].y + 0.5f));
171         double offset = (r_down - r_up)*0.2;
172         if((p.y >= r_up-offset) && (p.y <= r_down+offset) &&
                (avgArea(r_edges, p, 6) >= 20))
173         {
174             cvCircle(dst, p, radius, target_color[1]);
175             if(p.x <= p_r_ext.x)
176                 p_r_ext = p;
177             if(p.x >= p_r_int.x)
178                 p_r_int = p;
179         }
180         else
181             cvCircle(dst, p, radius, target_color[2]);
182     }
183
184     // Definizione dei punti per l'occhio destro
185     ext[0] = p_r_ext;
186     in[0] = p_r_int;
187     up[0] = cvPoint(isoscelesTriangleVertex(p_r_int, p_r_ext,
            r_up), r_up);
188     down[0] = cvPoint(isoscelesTriangleVertex(p_r_int, p_r_ext,
            r_down), r_down);
189     cvCircle(dst, ext[0], radius, target_color[0]);
190     cvCircle(dst, in[0], radius, target_color[0]);
191     cvCircle(dst, up[0], radius, target_color[0]);
192     cvCircle(dst, down[0], radius, target_color[0]);
193
194     // Definizione dei punti per l'occhio sinistro
195     ext[1] = p_l_ext;
196     in[1] = p_l_int;
197     up[1] = cvPoint(isoscelesTriangleVertex(p_l_int, p_l_ext,
            l_up), l_up);
198     down[1] = cvPoint(isoscelesTriangleVertex(p_l_int, p_l_ext,
            l_down), l_down);
199     cvCircle(dst, ext[1], radius, target_color[0]);
200     cvCircle(dst, in[1], radius, target_color[0]);
201     cvCircle(dst, up[1], radius, target_color[0]);
202     cvCircle(dst, down[1], radius, target_color[0]);
203
204     // Calcolo dei parametri geometrici
205     this->calcValues(ext, in, up, down);
```

```
206
207     // Output del parametro globale per gli occhi
208     return distance(in[0], in[1])/distance(ext[0], ext[1]);
209 }
```

Dati i quattro punti per entrambi gli occhi, i parametri geometrici possono essere così calcolati (riferimento al codice 6.21):

- *upperEyelide*: per l'occhio destro è il rapporto tra l'altezza del triangolo dato dai punti *ext[0], in[0], up[0]*, rispetto al punto *up[0]*, e la lunghezza del segmento che unisce *up[0]* e *ext[0]*. Essendo un triangolo isoscele, si può calcolare l'altezza come distanza tra *up[0]* e il punto medio tra *ext[0]* e *in[0]*. Per l'occhio sinistro il calcolo è lo stesso a partire dai punti *ext[1], in[1], up[1]*.

- *lowerEyelide*: per l'occhio destro è il rapporto tra l'altezza del triangolo dato dai punti *ext[0], in[0], down[0]*, rispetto al punto *down[0]*, e la lunghezza del segmento che unisce *down[0]* e *ext[0]*. Analogamente al caso precedente, si può calcolare l'altezza come distanza tra *down[0]* e il punto medio tra *ext[0]* e *in[0]*. Per l'occhio sinistro il calcolo è lo stesso a partire dai punti *ext[1], in[1], down[1]*.

- *eyeAxisDirection*: per l'occhio destro è il rapporto tra l'altezza del triangolo dato dai punti *in[0], ext[0], ext[1]*, rispetto al punto *in[0]*, e la lunghezza del segmento che unisce *in[0]* e *ext[0]*. Il triangolo non è isoscele, pertanto il calcolo dell'altezza deve essere fatto per mezzo della funzione *triangleHeight* con input i tre vertici *in[0], ext[0], ext[1]* nell'ordine. Tale funzione assegna il segno al parametro coerentemente al modello. Per l'occhio sinistro il calcolo è lo stesso a partire dai punti *in[1], ext[1], ext[0]*.

Per ogni parametro, il valore finale che viene assegnato all'attributo della classe è la media dei due valori calcolati sul singolo occhio.

Codice 6.21: **eyes.cpp - calcValues**

```
211 /* CALCOLO DEI PARAMETRI GEOMETRICI A PARTIRE DAI PUNTI
       CARATTERISTICI.
212  */
213 void EyesFeatures::calcValues(CvPoint* ext, CvPoint* in, CvPoint*
       up, CvPoint* down)
214 {
215     // Calcolo della direzione dell'asse maggiore dell'occhio
          rispetto all'asse di riferimento (come media dei valori
          relativi ai due occhi)
216     double r_eyeAxisDirection = triangleHeight(in[0], ext[0],
          ext[1])/distance(in[0], ext[0]);
217     double l_eyeAxisDirection = triangleHeight(in[1], ext[1],
          ext[0])/distance(in[1], ext[1]);
218     eyeAxisDirection = (r_eyeAxisDirection + l_eyeAxisDirection)/2;
219
220     CvPoint m_r;
221     CvPoint m_l;
222     midPoint(ext[0], in[0], &m_r);
223     midPoint(ext[1], in[1], &m_l);
```

```
224
225     // Calcolo dell'ampiezza del semiasse minore superiore (come
            media dei valori relativi ai due occhi)
226     double r_upperEyelid = distance(up[0], m_r)/distance(ext[0],
            up[0]);
227     double l_upperEyelid = distance(up[1], m_l)/distance(ext[1],
            up[1]);
228     upperEyelid = (r_upperEyelid + l_upperEyelid)/2;
229
230     // Calcolo dell'ampiezza del semiasse minore inferiore (come
            media dei valori relativi ai due occhi)
231     double r_lowerEyelid = distance(down[0], m_r)/distance(ext[0],
            down[0]);
232     double l_lowerEyelid = distance(down[1], m_l)/distance(ext[1],
            down[1]);
233     lowerEyelid = (r_lowerEyelid + l_lowerEyelid)/2;
234 }
```

## Calcolo dei parametri geometrici per il naso

Per il naso, il processo è simile a quello usato per il singolo occhio, con la differenza sostanziale che il punto superiore è determinato. Infatti, assieme all'immagine dell'area del naso rilevata, vengono forniti come input i due punti degli occhi corrispondenti ai due estremi esterni. A partire da questi, può essere calcolata la loro distanza (di riferimento per alcuni parametri) e il loro punto medio (definito come estremo superiore del naso). Non è detto che l'immagine in input relativa al naso contenga tali punti collegati agli occhi; in tal caso verranno considerati opportunamente con un valore negativo per l'ordinata in maniera tale da poter calcolare correttamente le distanze tra essi e i punti rilevati all'interno dell'immagine.

Come per l'occhio, è possibile determinare il limite inferiore del naso considerando una proprietà delle righe dell'immagine. Infatti, considerando l'immagine alla quale viene applicato un filtro di Sobel, si può considerare l'istogramma rappresentato dall'array ottenuto attraverso il metodo *sumRow*, ovvero il risultato della somma dei valori dei pixel su ciascuna riga. L'istogramma che risulta presenta un picco in corrispondenza della base del naso, in particolare nella fascia in cui la linea di definizione del naso risulta più evidente dall'ombra data dalle narici.

Ancora una volta, dopo aver filtrato l'immagine, calcolato la somma per riga e ottimizzato l'istogramma con il metodo *smoothing*, viene cercato il picco massimo attraverso un ciclo che cerca tutti i picchi presenti e ne determina il massimo. In generale, la posizione del massimo rappresenta il limite inferiore per il naso. Può accadere che il picco massimo non corrisponda esattamente al termine del naso, ma che esista un altro picco in posizione inferiore che determini tale limite. Per questo motivo viene tenuta traccia anche del secondo picco più rilevante e, nel caso in cui si trovi in posizione inferiore (quindi con indici di riga superiori) e non troppo distante rispetto al massimo assoluto, viene considerato l'indice *top* di questo secondo picco come limite.

Al termine di questa fase, si ottiene un indice di riga che indica il limite inferiore del naso e viene utilizzato come ordinata del punto inferiore del triangolo isoscele

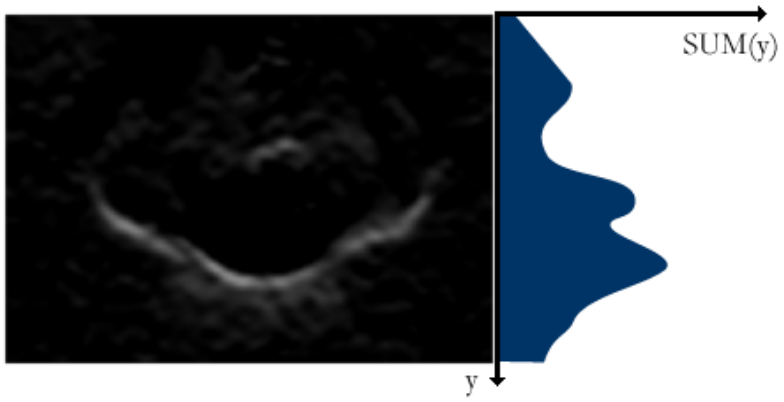


Figura 6.5: Istogramma relativo all'analisi del naso.

del modello e come riferimento per il rilevamento dei punti rimanenti, ovvero gli estremi laterali della base del naso.

L'individuazione di questi punti viene effettuata tramite la funzione *cvGoodFeaturesToTrack* e il successivo affinamento dei risultati tramite l'esclusione dei punti che non appartengono alla fascia orizzontale individuata dai due limiti costruiti a partire dal limite inferiore calcolato in precedenza. Inoltre, come per gli occhi, vengono esclusi i punti che non rispettano la soglia per il valore medio dei pixel dell'intorno, calcolato attraverso il metodo *avgArea*.

I risultati che superano il controllono sono considerati come il sottoinsieme che appartiene alla base del naso, pertanto vengono presi in considerazione i due estremi laterali del sottoinsieme che individuano la base del triangolo isoscele previsto dal modello.

Anche in questo caso l'ascissa del vertice mancante del triangolo isoscele, viene calcolata attraverso il metodo *isoscelesTriangleVertex* con i due estremi laterali e il limite inferiore individuato nella prima fase come input. Una volta definiti i quattro punti di riferimento, attraverso il metodo *calcValues* vengono impostati opportunamente i parametri geometrici della componente.

Infine il modello prevede un parametro globale anche per il naso, ovvero il rapporto tra altezza del naso e la distanza di riferimento degli occhi. Per entrambi i valori si conoscono gli estremi dei segmenti relativi, infatti il primo è dato dalla distanza tra l'estremo superiore e l'estremo inferiore, il secondo dalla distanza tra i due punti passati in ingresso al metodo.

Codice 6.22: **nose.cpp - getNoseFeatures**

```
44 /* ESTRAZIONE DEI PARAMETRI GEOMETRICI A PARTIRE DALL'IMMAGINE img.
45  * I PUNTI RILEVATI VENGONO INDICATI NELL'IMMAGINE dst.
46  * rif E' LA COPPIA DI PUNTI RILEVATI COME ESTREMI ESTERNI NEGLI
        OCCHI E USATA COME RIFERIMENTO.
47  * IL METODO FORNISCE IN OUTPUT IL PARAMETRO GLOBALE ASSOCIATO AL
        NASO.
48  */
49 double NoseFeatures::getNoseFeatures(IplImage* img, IplImage* dst,
       CvPoint* rif)
50 {
51     cvCopy(img, dst, NULL);
```

```
52    int w = img->width;
53    int h = img->height;
54    CvPoint ext[2];
55    CvPoint cent[2];
56
57    CvScalar target_color[3] = {{{255,255,255,255}}, {{0,0,0,0}},
          {{128,128,128,128}}};
58    int radius = 1;
59
60    // Definizione dell'estremo superiore
61    midPoint(rif[0], rif[1], &cent[0]);
62
63    // Sobel
64    IplImage* edges = cvCreateImage(cvSize(w,h), 8, 1);
65    cvSobel(img, edges, 0, 1, 3);
66
67    // Calcolo e normalizzazione dell'integrale di riga
68    double row[h];
69    double row_filt[h];
70    sumRow(edges, row);
71    smoothing(row, row_filt, h);
72
73    // Posizione coordinata y dell'estremo inferiore
74    int y_c;
75    int init = 0;
76    int th = h-1;
77    peak pkinit;
78    bool found = findNextPeak(row_filt, th, init, &pkinit);
79    peak pk1 = pkinit;
80    peak pk2 = pkinit;
81    init = pkinit.end;
82    double value_top_pk1 = row_filt[pk1.top];
83    double value_top_pk2 = row_filt[pk2.top];
84    peak pkc;
85    do
86    {
87       found = findNextPeak(row_filt, th, init, &pkc);
88       if(found && (row_filt[pkc.top] > value_top_pk1))
89       {
90          pk2 = pk1;
91          value_top_pk2 = value_top_pk1;
92          pk1 = pkc;
93          value_top_pk1 = row_filt[pkc.top];
94       }
95       else if(found && (row_filt[pkc.top] > value_top_pk2))
96       {
97          pk2 = pkc;
98          value_top_pk2 = row_filt[pkc.top];
99       }
100      init = pkc.end;
101   }while(found && (pkc.start < th));
102   y_c = pk1.top;
103   if((pk2.top > pk1.top) && (pk2.top-pk1.top<0.3*h))
104      y_c = pk2.top;
105
106   // Parametri della funzione di "corner detection"
107   const int MAX_CORNERS = 50;
```

```
108     double quality_level = 0.002;
109     double min_distance = (int)(w/30);
110     int eig_block_size = 3;
111     int use_harris = false;
112 
113     IplImage* eig_image = cvCreateImage(cvSize(w,h), IPL_DEPTH_32F,
            1);
114     IplImage* temp_image = cvCreateImage(cvSize(w,h),
            IPL_DEPTH_32F, 1);
115 
116     // Ricerca estremi laterali
117     CvPoint2D32f corners[MAX_CORNERS] = {0};
118     int corner_count = MAX_CORNERS;
119     cvGoodFeaturesToTrack(img, eig_image, temp_image, corners,
            &corner_count, quality_level, min_distance, NULL,
            eig_block_size, use_harris);
120     double limit1 = 0.35*h;
121     double limit2 = 0;
122     ext[0] = cvPoint(w, 0);
123     ext[1] = cvPoint(0, 0);
124     for(int i=0; i<corner_count; i++)
125     {
126         CvPoint p = cvPoint((int)(corners[i].x +
                0.5f),(int)(corners[i].y + 0.5f));
127         if((avgArea(edges, p, 6) >= 15) && (y_c-p.y <= limit1) &&
                (p.y-y_c <= limit2))
128         {
129             cvCircle(dst, p, radius, target_color[1]);
130             if(p.x <= ext[0].x)
131                 ext[0] = p;
132             if(p.x >= ext[1].x)
133                 ext[1] = p;
134         }
135         else
136         {
137             cvCircle(dst, p, radius, target_color[2]);
138         }
139     }
140     cvCircle(dst, ext[0], radius, target_color[0]);
141     cvCircle(dst, ext[1], radius, target_color[0]);
142 
143     // Definizione dell'estremo inferiore
144     int x_c = isoscelesTriangleVertex(ext[0], ext[1], y_c);
145     cent[1] = cvPoint(x_c, y_c);
146     cvCircle(dst, cent[1], radius, target_color[0]);
147 
148     // Calcolo dei parametri geometrici
149     this->calcValues(ext, cent);
150 
151     // Output del parametro globale per gli occhi
152     return distance(cent[0], cent[1])/distance(rif[0], rif[1]);
153 }
```

Dati i due estremi laterali *ext[0]*, *ext[1]* e quelli sull'asse centrale *cent[0]*, *cent[1]*, è possibile calcolare i parametri geometrici del naso secondo le seguenti operazioni (implementate nel codice 6.23):

- *base*: è il rapporto tra l'altezza del triangolo isoscele definito dai punti *cent[1], ext[0], ext[1]* e la lunghezza del segmento che unisce *cent[1]* e *ext[0]*. Per calcolare l'altezza del triangolo è sufficiente calcolare la distanza tra *cent[1]* e il punto medio tra *ext[0]* e *ext[1]*.
- *width*: è la larghezza della base del naso in proporzione all'altezza e viene calcolata come il rapporto tra la distanza tra *ext[0]* e *ext[1]* e la distanza tra *cent[0]* e *cent[1]*.

Codice 6.23: **nose.cpp - calcValues**

```
155 /* CALCOLO DEI PARAMETRI GEOMETRICI A PARTIRE DAI PUNTI
       CARATTERISTICI.
156  */
157 void NoseFeatures::calcValues(CvPoint* ext, CvPoint* cent)
158 {
159     CvPoint m;
160     midPoint(ext[0], ext[1], &m);
161
162     // Calcolo dell'ampiezza dell'angolo tra l'estremo laterale e
           l'estremo inferiore
163     base = distance(cent[1], m)/distance(cent[1], ext[0]);
164
165     // Calcolo della larghezza (in proporzione all'altezza)
166     width = distance(ext[0], ext[1])/distance(cent[0], cent[1]);
167 }
```

## Calcolo dei parametri geometrici per la bocca

Per la bocca, il procedimento di estrazione dei punti significativi dall'immagine si può considerare molto simile a quello per l'occhio in quanto il modello è sempre quello dell'individuazione di due triangoli isosceli con la base in comune.

Anche in questo caso viene preso in considerazione l'array che valuta la varianza in ciascuna riga, ottimizzato per mezzo del metodo *smoothing* che applica un filtro gaussiano. Con la stessa tecnica utilizzata per l'occhio vengono determinati un limite superiore e un limite inferiore per la bocca i quali, oltre a determinare le ordinate dei due vertici relativi nel triangolo isoscele, sono utilizzati per la definizione dell'area all'interno della quale individuare i punti rimanenti.

Infatti, per gli estremi laterali, viene applicato nuovamente il metodo di "corner detection" *cvGoodFeaturesToTrack* e i risultati iniziali vengono affinati attraverso l'analisi dei pixel dell'intorno dell'area e con l'esclusione dei punti che non appartengono alla fascia prefissata. Tra i punti in cui questo controllo dà esito positivo, vengono scelti i due estremi in quanto si presume individuino gli estremi laterali della bocca.

Vengono poi calcolate le ascisse dei punti superiore e inferiore, sempre attraverso il metodo *isoscelesTriangleVertex* che tiene conto del fatto che i due triangoli sono isosceli. I punti ottenuti, assieme agli estremi laterali, vengono dati in input al metodo *calcValues* per calcolare i parametri geometrici.

Infine vengono calcolati i due parametri globali legati alla bocca. Come per il naso, in input al metodo viene fornita la coppia di punti di riferimento relativi agli

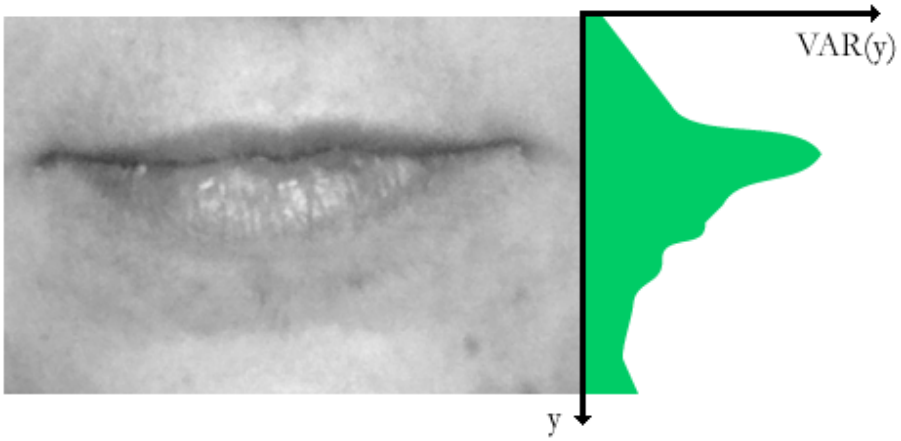


Figura 6.6: Istogramma relativo all'analisi della bocca.

occhi, ovvero i due estremi esterni. A partire da questi, viene calcolato il loro punto medio e la loro distanza che sarà considerata come "distanza di riferimento". I due parametri vengono poi calcolati nel seguente modo:

- la posizione della bocca è il risultato del rapporto tra la distanza occhi-bocca e la distanza di riferimento. La distanza occhi-bocca è calcolata tra il punto medio definito per gli occhi e il punto medio tra i due estremi laterali della bocca.

- la larghezza viene calcolata come la distanza tra i due estremi laterali della bocca e la distanza di riferimento.

Codice 6.24: **nose.cpp - getMouthFeatures**

```
44 /* ESTRAZIONE DEI PARAMETRI GEOMETRICI A PARTIRE DALL'IMMAGINE img.
45  * I PUNTI RILEVATI VENGONO INDICATI NELL'IMMAGINE dst.
46  * rif E' LA COPPIA DI PUNTI RILEVATI COME ESTREMI ESTERNI NEGLI
        OCCHI E USATI COME RIFERIMENTO.
47  * IN d VIENE CALCOLATA LA COPPIA DI PARAMETRI GLOBALI ASSOCIATI
        ALLA BOCCA.
48  */
49 void MouthFeatures::getMouthFeatures(IplImage* img, IplImage* dst,
       CvPoint* rif, double* d)
50 {
51     cvCopy(img, dst, NULL);
52     int w = img->width;
53     int h = img->height;
54
55     CvScalar target_color[3] = {{{255,255,255,255}}, {{0,0,0,0}},
          {{128,128,128,128}}};
56     int radius = 1;
57
58     // Ricerca punto superiore e inferiore
59     double row_var[h];
60     double row_var2[h];
61     varRow(img, row_var2);
62     smoothing(row_var2, row_var, h);
```

```
63    peak pkc;
64    peak pkinit = {0, (int)(h/2), h};
65    peak pk1 = pkinit;
66    int init = 0;
67    bool found;
68    do
69    {
70       found = findNextPeak(row_var, h, init, &pkc);
71       if(row_var[pkc.top] > row_var[pk1.top])
72       {
73          pk1 = pkc;
74       }
75       init = pkc.end;
76    }while(found && (pkc.end < h));
77    int up = pk1.top;
78    int down = pk1.top;
79    double top_val = row_var[pk1.top];
80    while((up-1 >= 0) && (row_var[up-1] > (top_val*0.2)))
81       up--;
82    while((down+1 < h) && (row_var[down+1] > (top_val*0.15)))
83       down++;
84
85    // Parametri della funzione di "corner detection"
86    const int MAX_CORNERS = 20;
87    double quality_level = 0.001;
88    double min_distance = (int)(w/22);
89    int eig_block_size = 3;
90    int use_harris = false;
91
92    IplImage* edges = cvCreateImage(cvSize(w,h), 8, 1);
93    cvSobel(img, edges, 0, 1, 3);
94    IplImage* eig_image = cvCreateImage(cvSize(w,h), IPL_DEPTH_32F,
         1);
95    IplImage* temp_image = cvCreateImage(cvSize(w,h),
         IPL_DEPTH_32F, 1);
96
97    // Ricerca degli estremi laterali
98    CvPoint2D32f corners[MAX_CORNERS] = {0};
99    int corner_count = MAX_CORNERS;
100   cvGoodFeaturesToTrack(img, eig_image, temp_image, corners,
         &corner_count, quality_level, min_distance, NULL,
         eig_block_size, use_harris );
101   CvPoint p_left = cvPoint(w, 0);
102   CvPoint p_right = cvPoint(0, 0);
103   double limit1 = up - (down-up)*0.1;
104   double limit2 = down + (down-up)*0.1;
105   for(int i=0; i<corner_count; i++)
106   {
107      CvPoint p = cvPoint((int)(corners[i].x + 0.5f),
            (int)(corners[i].y + 0.5f));
108      if((avgArea(edges, p, 6) >= 20) && (p.y >= limit1) && (p.y
            <= limit2))
109      {
110         cvCircle(dst, p, radius, target_color[1]);
111         if(p.x <= p_left.x)
112            p_left = p;
113         if(p.x >= p_right.x)
```

```
114                 p_right = p;
115         }
116         else
117         {
118             cvCircle(dst, p, radius, target_color[2]);
119         }
120     }
121
122     // Definizione dei punti
123     CvPoint ext[2];
124     CvPoint supinf[2];
125     ext[0] = p_right;
126     ext[1] = p_left;
127     supinf[0] = cvPoint(isoscelesTriangleVertex(p_right, p_left,
            up), up);
128     supinf[1] = cvPoint(isoscelesTriangleVertex(p_right, p_left,
            down), down);
129     cvCircle(dst, ext[0], radius, target_color[0]);
130     cvCircle(dst, ext[1], radius, target_color[0]);
131     cvCircle(dst, supinf[0], radius, target_color[0]);
132     cvCircle(dst, supinf[1], radius, target_color[0]);
133
134     // Calcolo dei parametri geometrici
135     this->calcValues(ext, supinf);
136
137     // Definizione dei riferimenti: centro occhi e distanza di
            riferimento
138     CvPoint cent[2];
139     midPoint(rif[0], rif[1], &cent[0]);
140     cvCircle(dst, cent[1], radius, target_color[0]);
141     double dist_rif = distance(rif[0], rif[1]);
142
143     // Calcolo parametri: posizione e larghezza
144     midPoint(ext[0], ext[1], &cent[1]);
145     d[0] = distance(cent[0], cent[1])/dist_rif;
146     d[1] = distance(ext[0], ext[1])/dist_rif;
147 }
```

Dati i due estremi laterali *ext[0]*, *ext[1]* e quelli superiore e inferiore *cent[0]*, *cent[1]*, è possibile calcolare i parametri geometrici della bocca attraverso le operazioni implementate nel codice 6.25, ovvero:

- *supLip*: è il rapporto tra l'altezza del triangolo isoscele definito dai punti *cent[0]*, *ext[0]*, *ext[1]* e la lunghezza del segmento che unisce *cent[0]* e *ext[0]*. Per calcolare l'altezza del triangolo è sufficiente calcolare la distanza tra *cent[0]* e il punto medio tra *ext[0]* e *ext[1]*.

- *infLip*: è il rapporto tra l'altezza del triangolo isoscele definito dai punti *cent[1]*, *ext[0]*, *ext[1]* e la lunghezza del segmento che unisce *cent[1]* e *ext[0]*. Analogamente, per calcolare l'altezza del triangolo è sufficiente calcolare la distanza tra *cent[1]* e il punto medio tra *ext[0]* e *ext[1]*.

Codice 6.25: **mouth.cpp - calcValues**

```
149 /* CALCOLO DEI PARAMETRI GEOMETRICI A PARTIRE DAI PUNTI
       CARATTERISTICI.
150  */
151 void MouthFeatures::calcValues(CvPoint* ext, CvPoint* cent)
152 {
153     CvPoint m;
154     midPoint(ext[0], ext[1], &m);
155 
156     // Calcolo dell'ampiezza dell'angolo tra l'estremo laterale e
           l'estremo superiore
157     supLip = distance(cent[0], m)/distance(cent[0], ext[0]);
158 
159     // Calcolo dell'ampiezza dell'angolo tra l'estremo laterale e
           l'estremo inferiore
160     infLip = distance(cent[1], m)/distance(cent[1], ext[0]);
161 }
```

## Metodi per il calcolo di distanza e verosimiglianza

Per ogni componente, vengono definiti due metodi per confrontare tra loro due oggetti della classe relativa.

Il primo di questi calcola la distanza tra i due oggetti nello spazio definito dai parametri geometrici della componente, ovvero dagli attributi dell'oggetto. La metrica utilizzata è la semplice distanza euclidea, infatti a ciascun elemento non viene attribuito un coefficiente.

Il secondo metodo, invece, calcola il grado di verosimiglianza tra i due oggetti e viene usato per confrontare la descrizione geometrica di un'immagine con quella di un elemento di catalogo. Il calcolo, come indicato nel capitolo 5, viene eseguito a partire dalla distanza tra i due oggetti. Il valore viene poi confrontato con una soglia prefissata e, solo nel caso in cui non venga superata, si passa a determinare la verosimiglianza. Il valore della soglia sarà argomento dell'analisi sperimentale del software.

Seguono i codici degli algoritmi per le tre componenti del volto: 6.26, 6.27 e 6.28. I metodi non vengono definiti internamente alla classe poichè concettualmente non si riferiscono ad un oggetto ma rappresentano un confronto tra una coppia di essi.

Codice 6.26: **eyes.cpp - distance, likelihood**

```
236 /* CALCOLO DELLA DISTANZA TRA DUE MODELLI GEOMETRICI PER GLI OCCHI.
237  */
238 double compareEyes(EyesFeatures* a, EyesFeatures* b)
239 {
240     return sqrt(pow((a->lowerEyelid - b->lowerEyelid),2) +
           pow((a->upperEyelid - b->upperEyelid),2) +
           pow((a->eyeAxisDirection - b->eyeAxisDirection),2));
241 }
242 
243 /* CALCOLO DELLA VEROSIMIGLIANZA TRA DUE MODELLI GEOMETRICI PER
       GLI OCCHI.
244  */
245 double calcEyesLikelihood(EyesFeatures* a, EyesFeatures* b)
```

```
246 {
247    double d = compareEyes(a, b);
248    if(d < MAX_EYES_DISTANCE)
249       return (1 - d/MAX_EYES_DISTANCE);
250    else
251       return 0;
252 }
```

Codice 6.27: **nose.cpp - distance, likelihood**

```
169 /* CALCOLO DELLA DISTANZA TRA DUE MODELLI GEOMETRICI PER IL NASO.
170  */
171 double compareNose(NoseFeatures* a, NoseFeatures* b)
172 {
173    return sqrt(pow((a->width - b->width),2) + pow((a->base -
          b->base),2));
174 }
175
176 /* CALCOLO DELLA VEROSIMIGLIANZA TRA DUE MODELLI GEOMETRICI PER IL
       NASO.
177  */
178 double calcNoseLikelihood(NoseFeatures* a, NoseFeatures* b)
179 {
180    double d = compareNose(a, b);
181    if(d < MAX_NOSE_DISTANCE)
182       return (1 - d/MAX_NOSE_DISTANCE);
183    else
184       return 0;
185 }
```

Codice 6.28: **mouth.cpp - distance, likelihood**

```
163 /* CALCOLO DELLA DISTANZA TRA DUE MODELLI GEOMETRICI PER LA BOCCA.
164  */
165 double compareMouth(MouthFeatures* a, MouthFeatures* b)
166 {
167    return sqrt(pow((a->supLip - b->supLip),2) + pow((a->infLip -
          b->infLip),2));
168 }
169
170 /* CALCOLO DELLA VEROSIMIGLIANZA TRA DUE MODELLI GEOMETRICI PER LA
       BOCCA.
171  */
172 double calcMouthLikelihood(MouthFeatures* a, MouthFeatures* b)
173 {
174    double d = compareMouth(a, b);
175    if(d < MAX_MOUTH_DISTANCE)
176       return (1 - d/MAX_MOUTH_DISTANCE);
177    else
178       return 0;
179 }
```

Figura 6.7: Esempi di elemento di catalogo.

## 6.5 Il catalogo

Il catalogo viene implementato attraverso la classe *Catalog*, per la quale vengono definiti come attributi tre array e altrettanti interi (codice 6.29). Gli array sono composti da oggetti delle classi relative alla descrizione geometrica delle tre componenti, ovvero *EyesFeatures*, *NoseFeatures*, *MouthFeatures*. Contengono, infatti, tutte le descrizioni geometriche degli elementi del catalogo, divisi per componente. Gli interi non sono altro che i contatori del numero di elementi presente in ciascuna lista.

In questo modo il catalogo si interfaccia come una struttura indicizzata nella quale è facilmente rintracciabile un elemento se si conosce la sua posizione nell'array.

Codice 6.29: **catalog.cpp - class Catalog**

```
9  /* CLASSE CATALOG
10  * Ha come attributi i tre array di elementi di catalogo relative
       alle componenti del volto e un contatore per ciascuno di essi.
11  * Prevede il metodo costruttore per il caricamento del catalogo
       dalla base di dati.
12  */
13 class Catalog
14 {
15    public:
16       EyesFeatures* eyesCat;
17       int countEyes;
18       NoseFeatures* nosesCat;
19       int countNoses;
20       MouthFeatures* mouthsCat;
21       int countMouths;
22       Catalog();
23 };
24
25 void insertNewEyes(int from, int to);
26 void insertNewNoses(int from, int to);
27 void insertNewMouths(int from, int to);
```

Questa struttura rappresenta l'implementazione temporanea del catalogo nell'intervallo di tempo in cui il programma è in funzione. Tutti i dati relativi alle descrizioni geometriche, invece, devono essere memorizzati su una base fissa che, per semplicità, viene realizzata attraverso file di testo. Questi sono strettamente collegati con gli attributi della classe, infatti i file *counteyes.txt*, *countnoses.txt* e *countmouths.txt* memorizzano il numero di elementi per ciascuna componente men-

tre *eyes.txt*, *noses.txt* e *mouths.txt* contengono le liste di parametri geometrici degli elementi di catalogo.

L'informazione che viene mantenuta, dunque, è la descrizione geometrica. Le immagini degli elementi non sono più necessarie una volta calcolati i parametri caratterizzanti ciascun elemento. Per fare questo in maniera precisa, i punti da rilevare all'interno dell'immagine vengono segnati manualmente con colori diversi per ciascuno di essi. Nell'immagine, essendo in scala di grigi, si riesce a distinguere un pixel che ha un colore determinato.

All'interno dei file relativi agli elementi di catalogo, ciascuno viene memorizzato in una riga diversa, secondo la seguente struttura:

*[indice dell'elemento] (parametro 1) (parametro 2) ... (parametro k)*

La classe catalogo contiene un solo metodo ovvero il metodo costruttore (codice 6.30), il quale carica tutte le informazioni dai file. Tale metodo effettua le stesse operazioni per ogni componente:

1. Rileva il numero di elementi di catalogo salvati per la componente in questione dal file specifico e assegna tale valore all'attributo che fa da contatore per la componente.

2. Crea l'array di descrizioni geometriche per la componente, di dimensione pari al valore appena rilevato.

3. Apre il file relativo agli elementi di catalogo per la componente, in lettura.

4. Per ogni riga del file, rileva i parametri geometrici dell'elemento di catalogo riconoscendo la sintassi specifica. Dopodichè assegna i valori all'elemento dell'array corrispondente, attraverso il metodo *setValues* per la componente in questione.

5. Quando vengono estratti tutti i dati e quindi definite tutte le posizioni dell'array, il ciclo termina e si passa alla componente successiva secondo le stesse fasi.

Codice 6.30: **catalog.cpp - Catalog, metodo costruttore**

```
29 /* INIZIALIZZAZIONE DEL CATALOGO A PARTIRE DAI DATI SALVATI.
30  */
31 Catalog::Catalog()
32 {
33    FILE * countFile = NULL;
34    FILE * elementsFile = NULL;
35
36    // Occhi
37    countFile = fopen("db/counteyes.txt", "r");
38    int count_e = 0;
39    if (countFile != NULL)
40    {
41       fscanf(countFile, "%i", &count_e);
42       fclose(countFile);
43    }
```

```
44    countEyes = count_e;
45    eyesCat = new EyesFeatures[count_e];
46
47    if(count_e > 0)
48    {
49       elementsFile = fopen("db/eyes.txt", "r");
50       double ue;
51       double le;
52       double ead;
53       char str_e[55];
54       for(int i=0; i<count_e; i++)
55       {
56          char* line = fgets(str_e, 50, elementsFile);
57          if(line == NULL)
58             break;
59          sscanf(line, "%*i %lf %lf %lf", &ue, &le, &ead);
60          eyesCat[i].setValues(ue, le, ead);
61       }
62       fclose(elementsFile);
63    }
64
65    // Naso
66    countFile = fopen("db/countnoses.txt", "r");
67    int count_n = 0;
68    if (countFile != NULL)
69    {
70       fscanf(countFile, "%i", &count_n);
71       fclose(countFile);
72    }
73    countNoses = count_n;
74    nosesCat = new NoseFeatures[count_n];
75
76    if(count_n > 0)
77    {
78       elementsFile = fopen("db/noses.txt", "r");
79       double n_width;
80       double n_base;
81       char str_n[55];
82       for(int i=0; i<count_n; i++)
83       {
84          char* line = fgets(str_n, 50, elementsFile);
85          if(line == NULL)
86             break;
87          sscanf(line, "%*i %lf %lf", &n_width, &n_base);
88          nosesCat[i].setValues(n_width, n_base);
89       }
90       fclose(elementsFile);
91    }
92
93    // Bocca
94    countFile = fopen("db/countmouths.txt", "r");
95    int count_m = 0;
96    if (countFile != NULL)
97    {
98       fscanf(countFile, "%i", &count_m);
99       fclose(countFile);
100   }
```

```
101     countMouths = count_m;
102     mouthsCat = new MouthFeatures[count_m];
103
104     if(count_m > 0)
105     {
106         elementsFile = fopen("db/mouths.txt", "r");
107         double sup;
108         double inf;
109         char str_m[55];
110         for(int i=0; i<count_m; i++)
111         {
112             char* line = fgets(str_m, 50, elementsFile);
113             if(line == NULL)
114                 break;
115             sscanf(line, "%*i %lf %lf", &sup, &inf);
116             mouthsCat[i].setValues(sup, inf);
117         }
118         fclose(elementsFile);
119     }
120 }
```

Oltre a questo, vengono inseriti nello stesso file tre metodi, uno per componente, che permettono di eleborare le immagini degli elementi del catalogo per derivarne la descrizione geometrica. Questi non interagiscono direttamente con la classe *Catalog* ma aggiornano i file di testo contenenti la base di dati.

Come accennato in precedenza, l'estrazione dei parametri non avviene secondo il procedimento automatico che viene eseguito per le immagini degli utenti ma i punti vengono inseriti manualmente nelle immagini. Queste operazioni devono essere realizzate solo una volta in fase di inizializzazione del sistema, pertanto il fatto che non siano efficienti è poco rilevante rispetto al grado di accuratezza che garantiscono. Il nome dei file delle immagini e composto dal nome della componente, seguito da un numero che indica l'indice dell'elemento di catalogo.

La procedura è molto simile per le tre componenti pertanto la descrizione che verrà fatta è comune. L'implementazione specifica viene riportata attraverso i codici 6.31 per gli occhi, 6.32 per il naso, 6.33 per la bocca.

In ingresso, dai parametri *from* e *to*, viene fornito un intervallo di indici che corrisponde alle immagini da considerare. La prima operazione che viene eseguita è l'aggiornamento del file relativo al contatore di elementi per la componente in questione. Per fare questo viene aperto il file in lettura e caricato il numero di elementi già presenti. A questo valore viene sommato il numero di elementi dell'intervallo in ingresso e il risultato viene sovrascritto al valore precedente, riaprendo il file del contatore in scrittura.

Successivamente viene aperto il file relativo alle descrizioni degli elementi di catalogo e, per ogni immagine dell'intervallo in input, vengono eseguite le seguenti operazioni:

1. L'immagine viene caricata in un oggetto *IplImage*.

2. In un doppio ciclo nidificato che scorre ogni riga e ogni colonna, viene considerato ciascun pixel dell'immagine. Se il colore del pixel corrisponde a un particolare colore di un punto caratteristico, il punto viene salvato nella variabile *CvPoint* relativa.

3. Terminata la rilevazione, viene eseguita un'operazione che adatta i punti al modello stabilito. Pertanto, per ogni punto corrispondente al vertice non appartenente alla base di un triangolo isoscele, viene mantenuta la sua ordinata mentre la sua ascissa viene aggiornata rispettando la proprietà del triangolo, attraverso il metodo *isoscelesTriangleVertex*.

4. Attraverso i punti ottenuti e adattati al modello, viene creato un nuovo oggetto della classe relativa alla descrizione geometrica della componente (*EyesFeatures, NoseFeatures* o *MouthFeatures*) e vengono calcolati e assegnati i parametri geometrici a partire dai punti, attraverso il metodo *calcValues*.

5. Infine viene aggiornato il file aggiungendo una nuova riga al termine e inserendo i parametri geometrici dell'elemento di catalogo in questione, rispettando la sintassi predefinita.

Una particolarità della rilevazione dei punti nelle immagini degli occhi, è che per ogni colore vengono segnati due punti, uno per occhio. Il ciclo più esterno fa variare le colonne dell'immagine, pertanto si otterrà certamente che il primo punto rilevato per un certo colore corrisponde all'occhio destro, il secondo all'occhio sinistro. Risulta semplice gestire la rilevazione associando a ciascun colore un array di punti di dimensione due e un contatore di punti rilevati.

Un'altra precisazione va fatta sulla modifica dei punti rilevati al fine di adattarli al modello. Questa operazione può sembrare una forzatura in quanto il punto modificato potrebbe non essere più caratterizzante per la componente. In realtà le immagini del catalogo sono orientate quasi perfettamente lungo gli assi orizzontale e verticale. Manualmente è facile ottenere un'approssimazione molto buona di un punto che rispetti la proprietà del triangolo isoscele che definisce. Le modifiche, pertanto, risultano minime e in ogni caso rimangono significative per la componente in questione.

Codice 6.31: **catalog.cpp - insertNewEyes**

```
122 /* INSERIMANTO NELLA BASE DI DATI DI NUOVI ELEMENTI PER LA
       COMPONENTE OCCHI.
123  */
124 void insertNewEyes(int from, int to)
125 {
126    // Acquisizione del numero di elementi gia' salvati
127    FILE * countFile = NULL;
128    int count;
129    countFile = fopen("db/counteyes.txt", "r");
130    if ( countFile != NULL )
131    {
132       fscanf(countFile, "%i", &count);
133       count += to - from + 1;
134       fclose(countFile);
135    }
136    else
137    {
138       count = to - from + 1;
139    }
140
141    // Aggiornamento del contatore
```

```
142     countFile = fopen("db/counteyes.txt", "w");
143     if (countFile != NULL)
144     {
145        fprintf(countFile, "%i", count);
146        fclose(countFile);
147     }
148
149     char str[55];
150     FILE * eFile = NULL;
151     eFile = fopen("db/eyes.txt", "a");
152     for(int num=from; num<=to; num++)
153     {
154        // Viene preso in considerazione un elemento da inserire
155        sprintf(str, "cat/eyes%i.bmp", num);
156        const char* filename = str;
157        IplImage* img = cvLoadImage(filename, CV_LOAD_IMAGE_COLOR);
158        int w = img->width;
159        int h = img->height;
160        int i,j;
161        int c_ext = 0;
162        int c_in = 0;
163        int c_up = 0;
164        int c_down = 0;
165        CvPoint ext[2];
166        CvPoint in[2];
167        CvPoint up[2];
168        CvPoint down[2];
169
170        // Analisi dell'immagine pixel per pixel per l'estrazione
              dei punti
171        for(j=0; j<w; j++)
172        {
173           for(i=0; i<h; i++)
174           {
175              CvScalar pixel = cvGet2D(img, i, j);
176              if(c_ext < 2 && pixel.val[0] == 255 && pixel.val[1] ==
                    0 && pixel.val[2] == 0) // blu - estremo esterno
177              {
178                 ext[c_ext] = cvPoint(j, i);
179                 c_ext++;
180              }
181              if(c_in < 2 && pixel.val[0] == 0 && pixel.val[1] ==
                    255 && pixel.val[2] == 0) // verde - estremo interno
182              {
183                 in[c_in] = cvPoint(j, i);
184                 c_in++;
185              }
186              if(c_up < 2 && pixel.val[0] == 0 && pixel.val[1] == 0
                    && pixel.val[2] == 255) // rosso - estremo superiore
187              {
188                 up[c_up] = cvPoint(j, i);
189                 c_up++;
190              }
191              if(c_down < 2 && pixel.val[0] == 0 && pixel.val[1] ==
                    255 && pixel.val[2] == 255) // giallo - estremo
                    inferiore
192              {
```

```
193                 down[c_down] = cvPoint(j, i);
194                 c_down++;
195             }
196         }
197     }
198
199     // Aggiustamento dei punti rispetto al modello geometrico
200     for(j=0; j<2; j++)
201     {
202         int y_up = up[j].y;
203         int x_up = isoscelesTriangleVertex(ext[j], in[j], y_up);
204         up[j].x = x_up;
205         int y_down = down[j].y;
206         int x_down = isoscelesTriangleVertex(ext[j], in[j],
                y_down);
207         down[j].x = x_down;
208     }
209
210     // Calcolo dei parametri geometrici e salvataggio su file
211     EyesFeatures* catalog_eyes = new EyesFeatures();
212     catalog_eyes->calcValues(ext, in, up, down);
213     if (eFile != NULL)
214     {
215         fprintf(eFile, "%i %f %f %f\n", num,
                catalog_eyes->upperEyelid, catalog_eyes->lowerEyelid,
                catalog_eyes->eyeAxisDirection);
216     }
217 }
218 fclose(eFile);
219 }
```

Codice 6.32: **catalog.cpp - insertNewNoses**

```
221 /* INSERIMANTO NELLA BASE DI DATI DI NUOVI ELEMENTI PER LA
       COMPONENTE NASO.
222  */
223 void insertNewNoses(int from, int to)
224 {
225     // Acquisizione del numero di elementi gia' salvati
226     FILE * countFile = NULL;
227     int count;
228     countFile = fopen("db/countnoses.txt", "r");
229     if (countFile != NULL)
230     {
231         fscanf(countFile, "%i", &count);
232         count += to - from + 1;
233         fclose(countFile);
234     }
235     else
236     {
237         count = to - from + 1;
238     }
239
240     // Aggiornamento del contatore
241     countFile = fopen("db/countnoses.txt", "w");
242     if (countFile != NULL)
```

```
243     {
244         fprintf(countFile, "%i", count);
245         fclose(countFile);
246     }
247 
248     char str[55];
249     FILE * eFile = NULL;
250     eFile = fopen("db/noses.txt", "a");
251     for(int num=from; num<=to; num++)
252     {
253         // Viene preso in considerazione un elemento da inserire
254         sprintf(str, "cat/nose%i.bmp", num);
255         const char* filename = str;
256         IplImage* img = cvLoadImage(filename, CV_LOAD_IMAGE_COLOR);
257         int w = img->width;
258         int h = img->height;
259         int i,j;
260         bool sx = true;
261         bool dx = true;
262         bool up = true;
263         bool down = true;
264         int c_in = 0;
265         CvPoint ext[2];
266         CvPoint cent[2];
267 
268         // Analisi dell'immagine pixel per pixel per l'estrazione
               dei punti
269         for(j=0; j<w; j++)
270         {
271             for(i=0; i<h; i++)
272             {
273                 CvScalar pixel = cvGet2D(img, i, j);
274                 if(dx && pixel.val[0] == 255 && pixel.val[1] == 0 &&
                       pixel.val[2] == 0) // blu - estremo sinistro
275                 {
276                     ext[0] = cvPoint(j, i);
277                     dx = false;
278                 }
279                 if(sx && pixel.val[0] == 0 && pixel.val[1] == 255 &&
                       pixel.val[2] == 0) // verde - estremo destro
280                 {
281                     ext[1] = cvPoint(j, i);
282                     sx = false;
283                 }
284                 if(up && pixel.val[0] == 0 && pixel.val[1] == 0 &&
                       pixel.val[2] == 255) // rosso - estremo superiore
285                 {
286                     cent[0] = cvPoint(j, i);
287                     up = false;
288                 }
289                 if(down && pixel.val[0] == 0 && pixel.val[1] == 255 &&
                       pixel.val[2] == 255) // giallo - estremo inferiore
290                 {
291                     cent[1] = cvPoint(j, i);
292                     down = false;
293                 }
294             }
```

```
295         }
296         // Aggiustamento dei punti rispetto al modello geometrico
297         int y_down = cent[1].y;
298         int x_down = isoscelesTriangleVertex(ext[0], ext[1], y_down);
299         cent[1].x = x_down;
300 
301         // Calcolo dei parametri geometrici e salvataggio su file
302         NoseFeatures* catalog_nose = new NoseFeatures();
303         catalog_nose->calcValues(ext, cent);
304         if (eFile != NULL)
305         {
306             fprintf(eFile, "%i %f %f\n", num, catalog_nose->width,
                    catalog_nose->base);
307         }
308     }
309     fclose(eFile);
310 }
```

Codice 6.33: **catalog.cpp - insertNewMouths**

```
312 /* INSERIMANTO NELLA BASE DI DATI DI NUOVI ELEMENTI PER LA
       COMPONENTE BOCCA.
313  */
314 void insertNewMouths(int from, int to)
315 {
316     // Acquisizione del numero di elementi gia' salvati
317     FILE * countFile = NULL;
318     int count;
319     countFile = fopen("db/countmouths.txt", "r");
320     if (countFile != NULL)
321     {
322         fscanf(countFile, "%i", &count);
323         count += to - from + 1;
324         fclose(countFile);
325     }
326     else
327     {
328         count = to - from + 1;
329     }
330 
331     // Aggiornamento del contatore
332     countFile = fopen("db/countmouths.txt", "w");
333     if (countFile != NULL)
334     {
335         fprintf(countFile, "%i", count);
336         fclose(countFile);
337     }
338 
339     char str[55];
340     FILE * eFile = NULL;
341     eFile = fopen("db/mouths.txt", "a");
342     for(int num=from; num<=to; num++)
343     {
344         // Viene preso in considerazione un elemento da inserire
345         sprintf(str, "cat/mouth%i.bmp", num);
346         const char* filename = str;
```

```
347         IplImage* img = cvLoadImage(filename, CV_LOAD_IMAGE_COLOR);
348         int w = img->width;
349         int h = img->height;
350         int i,j;
351         bool sx = true;
352         bool dx = true;
353         bool up = true;
354         bool down = true;
355         int c_in = 0;
356         CvPoint ext[2];
357         CvPoint cent[2];
358
359         // Analisi dell'immagine pixel per pixel per l'estrazione
               dei punti
360         for(j=0; j<w; j++)
361         {
362            for(i=0; i<h; i++)
363            {
364               CvScalar pixel = cvGet2D(img, i, j);
365               if(dx && pixel.val[0] == 255 && pixel.val[1] == 0 &&
                     pixel.val[2] == 0) // blu - estremo sinistro
366               {
367                  ext[0] = cvPoint(j, i);
368                  dx = false;
369               }
370               if(sx && pixel.val[0] == 0 && pixel.val[1] == 255 &&
                     pixel.val[2] == 0) // verde - estremo destro
371               {
372                  ext[1] = cvPoint(j, i);
373                  sx = false;
374               }
375               if(up && pixel.val[0] == 0 && pixel.val[1] == 0 &&
                     pixel.val[2] == 255) // rosso - estremo superiore
376               {
377                  cent[0] = cvPoint(j, i);
378                  up = false;
379               }
380               if(down && pixel.val[0] == 0 && pixel.val[1] == 255 &&
                     pixel.val[2] == 255) // giallo - estremo inferiore
381               {
382                  cent[1] = cvPoint(j, i);
383                  down = false;
384               }
385            }
386         }
387         // Aggiustamento dei punti rispetto al modello geometrico
388         int y_up = cent[0].y;
389         int x_up = isoscelesTriangleVertex(ext[0], ext[1], y_up);
390         cent[0].x = x_up;
391         int y_down = cent[1].y;
392         int x_down = isoscelesTriangleVertex(ext[0], ext[1], y_down);
393         cent[1].x = x_down;
394
395         // Calcolo dei parametri geometrici e salvataggio su file
396         MouthFeatures* catalog_mouth = new MouthFeatures();
397         catalog_mouth->calcValues(ext, cent);
398         if (eFile != NULL)
```

```
399         {
400             fprintf(eFile, "%i %f %f\n", num, catalog_mouth->supLip,
                    catalog_mouth->infLip);
401         }
402     }
403     fclose(eFile);
404 }
```

## 6.6 L'identikit

L'identikit e tutte le procedure relative, rappresentano il punto cruciale del programma in quanto vengono implementati gli algoritmi che realizzano la fase di creazione dell'identikit e di confronto tra essi.

Viene perciò definita la classe *Identikit* che permette di gestire tutte queste operazioni. A questa si aggiunge la classe *User* che rappresenta un utente registrato nel sistema e che interagisce con la base di dati per il salvataggio e il caricamento delle informazioni. Infatti, per quanto riguarda gli utenti, esistono ancora una volta due file di riferimento: *users.tx* e *countUsers.txt*. Il primo memorizza tutti i dati relativi agli utenti, il secondo tiene conto del numero di utenti registrati e quindi presenti nel primo file.

L'identikit viene strutturato come un insieme di liste, una per componente, che contengono coppie di valori ovvero un identificatore di un elemento di catalogo e il valore di verosimiglianza associato, come definito nel modello formale presentato nei capitoli precedenti. Pertanto viene realizzata una classe specifica che implementa una lista così strutturata.

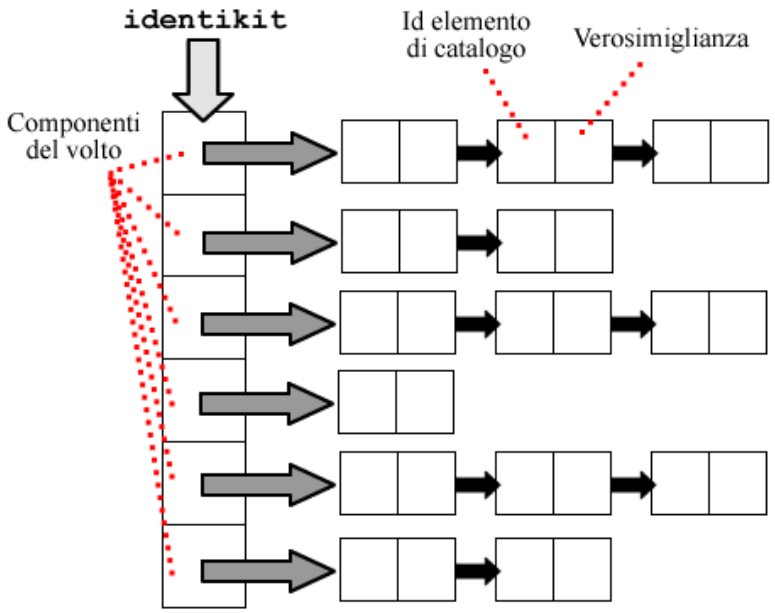


Figura 6.8: Schema della struttura che rappresenta l'identikit.

## Un modello ad hoc di lista concatenata

Una lista viene definita come un insieme di elementi, chiamati nodi, collegati tra loro. Un nodo è composto da due parti fondamentali: l'informazione relativa al nodo stesso e un puntatore al nodo successivo nella lista. In questo caso l'informazione è data dalla coppia formata dall'identificatore di un elemento di catalogo e dalla verosimiglianza associata.

L'implementazione di un nodo della lista avviene attraverso la classe *ListNode* (codice 6.34) che ha come attributi un intero *id* che identifica un elemento di catalogo, un parametro reale *lh* che indica la verosimiglianza e un puntatore *next* al nodo successivo in lista.

La classe, inoltre, fornisce i metodi per creare, modificare il nodo e per accedere ai sui attributi.

Codice 6.34: **list.cpp - class ListNode**

```
3  /* CLASSE LISTNODE
4   * Nodo di una lista che contiene un puntatore a un nodo della
        stessa classe e due valori che caratterizzano il nodo: un
        identificatore intero per un elemento di catalogo e un
        parametro legato alla verosimiglianza.
5   */
6  class ListNode
7  {
8     public:
9        ListNode(int index, double likelihood);
10       ListNode(int index, double likelihood, ListNode* n);
11       int getID();
12       double getLikelihood();
13       ListNode* getNext();
14       void setNext(ListNode* n);
15
16    private:
17       int id;
18       double lh;
19       ListNode* next;
20 };
21
22 ListNode::ListNode(int index, double likelihood)
23 {
24    id = index;
25    lh = likelihood;
26    next = NULL;
27 }
28
29 ListNode::ListNode(int index, double likelihood, ListNode* n)
30 {
31    id = index;
32    lh = likelihood;
33    next = n;
34 }
35
36 int ListNode::getID()
37 {
38    return id;
```

```
39 }
40
41 double ListNode::getLikelihood()
42 {
43    return lh;
44 }
45
46 ListNode* ListNode::getNext()
47 {
48    return next;
49 }
50
51 void ListNode::setNext(ListNode* n)
52 {
53    next = n;
54 }
```

Definita la struttura di un nodo, la lista viene realizzata mantenendo l'informazione del nodo in testa (attributo *head*) dal quale è possibile accedere a tutti gli elementi attraverso i puntatori contenuti in ogni nodo. Inoltre viene memorizzato il numero totale di elementi inseriti (attributo *len*) e si dà la possibilità di impostare un limite massimo a questo numero (attraverso l'attributo *maxlen*). Quest'ultimo attributo viene impostato al valore $-1$ se non si vuole porre alcun limite al numero di elementi in lista. Se invece il valore è non negativo, indica il numero massimo di nodi inseribili nella lista. Questo porta adottare una politica di gestione degli inserimenti poichè la lunghezza dell'array deve rimanere sotto una certa soglia.

Il codice 6.35 implementa la lista fornendo tutti i metodi necessari per la creazione della lista e l'inserimento, l'eliminazione e l'accesso ai nodi. In particolare il metodo *insert* inserisce un nuovo nodo, tenendo conto delle considerazioni fatte sul limite possibile per la lunghezza.

La politica che viene adottata è quella di mantenere la lista ordinata secondo il valore di verosimiglianza in maniera tale che, in caso di un inserimento che faccia superare il limite previsto, vengano mantenuti i *maxlen* elementi con verosimiglianza maggiore. La lista avrà sempre in testa il nodo con verosimiglianza minore, ovvero il candidato ad essere eliminato nel caso in cui si verificasse l'inserimento di un elemento con verosimiglianza superiore con la lunghezza della lista che supera il limite consentito.

L'algoritmo, quindi, esegue operazioni diverse in base allo stato in cui si trova la lista:

- se la lista è vuota ed il limite sul numero di elementi è diverso da zero, l'elemento viene inserito in testa alla lista ($maxlen = 0$ in realtà è un caso degenere per la lista che sarebbe forzata a rimanere sempre vuota). In output viene dato il valore *true* in conferma del corretto inserimento.

- se la lista non è vuota e l'elemento in testa ha verosimiglianza maggiore di quella dell'elemento da inserire ma l'elemento può essere ugualmente inserito (perchè $maxlen = -1$ oppure la lista non ha ancora raggiunto il limite massimo di elementi), allora l'elemento viene inserito in testa alla lista e viene legato al precedente nodo in testa, nuovamente restituendo *true* in output.

- se l'elemento in testa ha verosimiglianza minore di quello da inserire, allora l'elemento va certamente inserito nella lista. In tal caso si passa da un elemento all'altro fino a raggiungere la posizione corretta del nuovo elemento affinchè la lista rimanga ordinata. Attraverso la gestione dei puntatori dei nodi il nuovo elemento viene inserito e in tal modo va ad incrementare la lunghezza della lista. In questo caso può accadere che si superi il limite massimo consentito e viene eliminato dalla lista il primo elemento, ovvero quello con verosimiglianza minore. Anche in questo caso l'output è *true*.

- Non si verificano le situazioni precedenti nel caso in cui la lista risulta piena e il nuovo elemento non ha un valore di verosimiglianza sufficiente ad entrarvi. In tal caso la lista non viene modificata e viene dato in output il valore *false*.

Il valore massimo viene impostato dal metodo che crea la lista ed è uno dei parametri che sarà oggetto dell'analisi sperimentale.

Codice 6.35: **list.cpp - class List**

```
56 /* CLASSE LIST
57  * Struttura che implementa una lista concatenata di ListNode, con
      inserimento in ordine di parametro di verosimiglianza dal
      minore (in testa) al maggiore (in coda) e con la possibilita'
      di limitare il numero di elementi mantenendo quelli con
      parametro maggiore.
58  */
59 class List
60 {
61    public:
62       List();
63       List(int maxlength);
64       bool isEmpty();
65       bool insert(int index, double likelihood);
66       ListNode* getFirst();
67       int getElementAt(int index);
68       int dropFirst();
69       int length();
70
71    private:
72       ListNode* head;
73       int len;
74       int maxlen;
75 };
76
77 List::List()
78 {
79    head = NULL;
80    len = 0;
81    maxlen = -1;
82 }
83
84 List::List(int maxlength)
85 {
86    head = NULL;
87    len = 0;
88    maxlen = maxlength;
```

```
89 }
90
91 bool List::isEmpty()
92 {
93    return (len == 0);
94 }
95
96 bool List::insert(int index, double likelihood)
97 {
98    if(this->isEmpty() && (maxlen != 0))
99    {
100      head = new ListNode(index, likelihood);
101      len++;
102      return true;
103   }
104   else if((head->getLikelihood() >= likelihood) && (maxlen < 0 ||
         len < maxlen))
105   {
106      ListNode* Y = new ListNode(index, likelihood, head);
107      head = Y;
108      len++;
109      return true;
110   }
111   else if(head->getLikelihood() < likelihood)
112   {
113      ListNode* p = head;
114      while((p->getNext() != NULL) &&
            (p->getNext()->getLikelihood() < likelihood))
115      {
116         p = p->getNext();
117      }
118      ListNode* Y = new ListNode(index, likelihood, p->getNext());
119      p->setNext(Y);
120      len++;
121      if((maxlen > 0) && (len > maxlen))
122         this->dropFirst();
123      return true;
124   }
125   else
126   {
127      return false;
128   }
129 }
130
131 ListNode* List::getFirst()
132 {
133    return head;
134 }
135
136 int List::getElementAt(int index)
137 {
138    ListNode* n = head;
139    for(int i=0; i<index; i++)
140       n = n->getNext();
141    return n->getID();
142 }
143
```

```
144 int List::dropFirst()
145 {
146    ListNode* first = head;
147    head = first->getNext();
148    len--;
149    return first->getID();
150 }
151
152 int List::length()
153 {
154    return len;
155 }
```

## La classe identikit

L'dentikit viene costruito secondo il modello proposto nel capitolo 5 ovvero costituito da tre liste (*eyes, nose, mouth*) che conterranno gli elementi di catalogo selezionati per andare a comporre l'identikit. Oltre a queste, vengono considerati anche i quattro attributi relativi ai parametri globali individuati nell'immagine del soggetto per il quale viene calcolato l'identikit.

In questa prima versione del software, questi parametri non vengono considerati nel confronto per l'identificazione poichè l'attenzione è concentrata sul modello di identikit per componente, coerentemente agli algoritmi sviluppati nel capitolo 5. Sono stati rilevati e considerati fino al livello di struttura dell'identikit poichè possono essere opportunamente inseriti a corredo del modello di confronto di identikit per componente.

La classe *Identikit* (codice 6.36) fornisce tre metodi costruttori:

- il primo non riceve nulla come parametro di ingresso e inizializza le tre liste senza limiti di lunghezza. Questo metodo viene utilizzato quando l'identikit viene caricato dalla base di dati relativa agli utenti in cui i limiti di lunghezza delle liste sono già garantiti.

- il secondo riceve in ingresso un'immagine contenente un volto e il catalogo. Questo metodo viene utilizzato quando deve essere creato un identikit in fase di registrazione considerando tutti gli elementi di catalogo. Vengono così inizializzate le tre liste con i limiti di lunghezza dati dalle costanti prestabilite. Infine viene avviata la procedura di costruzione dell'identikit a partire dall'immagine attraverso il metodo *createIdentikit* in cui l'ultimo parametro in ingresso impostato al valore *false* indica che l'identikit deve essere calcolato a catalogo completo.

- il terzo metodo costruttore, a differenza del secondo, riceve in ingresso anche un identikit. In questo caso l'indicazione che viene data è quella di costruire l'identikit basandosi su un catalogo ridotto ai soli elementi dell'identikit in input. Le liste non vengono create con il limite di lunghezza in quanto la riduzione del catalogo prevede che il numero di elementi per componente sia già opportunamente limitato. Infine viene fatto eseguire il metodo *createIdentikit* con l'indicazione di mantenere un catalogo ridotto che viene passata attraverso il terzo e quarto parametro in ingresso.

Codice 6.36: **identikit.cpp - class Identikit**

```
8  #define MAX_LENGTH_EYES_LIST (10)
9  #define MAX_LENGTH_NOSE_LIST (10)
10 #define MAX_LENGTH_MOUTH_LIST (10)
11 #define AUTHENTICATION_TH (0.310)
12
13 /* CLASSE IDENTIKIT
14  * Ha come attributi le tre liste di elementi di catalogo relative
       alle componenti del volto e i quattro parametri globali.
15  * Prevede i metodi per la costruzione dell'identikit.
16  */
17 class Identikit
18 {
19    public:
20       List* eyes;
21       double eyesDistance;
22       List* nose;
23       double noseHeight;
24       List* mouth;
25       double mouthPosition;
26       double mouthWidth;
27       Identikit();
28       Identikit(char* imgaddress, Catalog* cat);
29       Identikit(char* imgaddress, Catalog* cat, Identikit*
            identikitCR);
30
31    private:
32       int createIdentikit(char* imgaddress, Catalog* cat,
            Identikit* identikit_rc, bool rc);
33 };
34
35 bool identify(Identikit* saved_id, Identikit* new_id);
36
37 /* INIZIALIZZAZIONE DI UN IDENTIKIT VUOTO.
38  */
39 Identikit::Identikit()
40 {
41    eyes = new List();
42    nose = new List();
43    mouth = new List();
44 }
45
46 /* INIZIALIZZAZIONE DI UN IDENITKIT PER LA FASE DI REGISTRAZIONE A
       PARTIRE DALL'IMMAGINE imgaddress.
47  */
48 Identikit::Identikit(char* imgaddress, Catalog* cat)
49 {
50    eyes = new List(MAX_LENGTH_EYES_LIST);
51    nose = new List(MAX_LENGTH_NOSE_LIST);
52    mouth = new List(MAX_LENGTH_MOUTH_LIST);
53    createIdentikit(imgaddress, cat, NULL, false);
54 }
55
56 /* INIZIALIZZAZIONE DI UN IDENITKIT PER L'AUTENTICAZIONE A PARTIRE
       DALL'IMMAGINE imgaddress E CON identikit_rc COME CATALOGO
       RIDOTTO.
57  */
```

```
58 Identikit::Identikit(char* imgaddress, Catalog* cat, Identikit*
       identikit_rc)
59 {
60     eyes = new List();
61     nose = new List();
62     mouth = new List();
63     createIdentikit(imgaddress, cat, identikit_rc, true);
64 }
```

La classe, nel momento in cui deve creare un identikit a partire dall'immagine, deve avviare la procedura che utilizza i metodi indicati nelle sezioni precedenti per il calcolo dei parametri geometrici. A partire da questi, poi, dovrà inserire gli elementi di catalogo opportuni nelle liste e calcolarne la verosimiglianza. Queste operazioni vengono implementate all'interno del metodo *createIdentikit* (codice 6.37).

Per prima cosa viene importata l'immagine in un oggetto *IplImage*. L'immagine a colori viene poi convertita in una in scala di grigi poichè nei metodi di estrazione dei parametri geometrici è stata fatta questa assunzione.

Viene poi avviata la procedura di rilevamento del volto all'interno dell'immagine attraverso il metodo *detectFace*. Se il risultato della ricerca è positivo, il rettangolo che viene restituito è ben definito. In caso contrario viene restituito il valore *NULL* e la procedura termina lasciando vuote le liste relative all'identikit.

In caso positivo, invece, viene estratta la porzione di immagine relativa al volto. Vengono poi definiti dei punti che nella procedura verranno impostati come i punti di riferimento degli occhi. Questi sono gli unici punti che, prelevati dall'analisi degli occhi, vengono utilizzati nell'analisi delle altre due parti del volto. Vengno definiti in tre coppie di variabili diverse perchè in base alla porzione di immagine di riferimento, le coordinate devono essere opportunamente modificate.

A partire dall'immagine del volto viene rilevata la prima componente: gli occhi. Il metodo *detectEyes* fornisce un rettangolo all'interno del volto che corrisponde all'area in cui vengono individuati gli occhi. Se tale rettangolo è ben definito, si passa all'estrazione dei parametri e alla costruzione della lista dell'identikit relativo alla componente.

Viene quindi creata un nuovo oggetto di tipo *EyesFeatures* al quale viene assegnati i valori dei parametri geometrici attraverso il metodo *getEyesFeatures*. Lo stesso metodo permette di impostare il valore del parametro globale relativo agli occhi *eyesDistance* e di assegnare alla coppia di variabili *rif* i due punti di riferimento rilevati come estremi esterni negli occhi.

A questo punto la variabile *e_ft* contiene i parametri geometrici rilevati ed è possibile creare l'identikit. La costruzione varia in base ai parametri di ingresso del metodo *createIdentikit*, ovvero alla presenza o meno di un identikit con il ruolo di catalogo ridotto.

Se il catalogo è ridotto ai soli componenti dell'identikit *identikit_rc*:

- Viene considerata la sua lista relativa alla componente occhi e attraverso un ciclo vengono esplorati i nodi.

- Per ciascuno di essi viene estratto l'identificativo dell'elemento di catalogo, corrispondente alla posizione del'elemento nell'array relativo agli occhi del catalogo. Dal catalogo, infatti, viene estratta la descrizione geometrica dell'ele-

mento ovvero l'oggetto di classe *EyesFeatures* relativo alla posizione indicata dall'identificativo dell'elemento di catalogo.

- Si calcola la verosimiglianza dei parametri geometrici del soggetto rispetto a quelli appena caricati, relativi all'elemento di catalogo in questione, per mezzo del metodo *calcLikelihood*.

- Se la verosimiglianza è maggiore di zero, la coppia formata dall'elemento di catalogo e dalla verosimiglianza associata viene inserita come un nuovo nodo nella lista *eyes* dell'identikit. In caso contrario l'elemento non deve essere inserito.

- Si ripetono le operazioni per ogni elemento della lista relativa agli occhi dell'identikit *identikit_rc*.

Nel caso in cui si debba calcolare l'identikit su tutto il catalogo, il metodo rimane analogo, con l'unica differenza che non c'è un sottoinsieme prefissato di elementi di catalogo ma l'operazione di calcolo della verosimiglianza deve essere effettuata per tutti gli elementi di catalogo dell'array relativo alla componente occhi.

Dopo la porzione di codice che mostra l'elemento di catalogo che viene valutato più verosimigliante, viene effettuata un'operazione che facilita il rilevamento del naso all'interno dell'immagine del volto. Infatti l'area viene ristretta in verticale alla porzione di volto inferiore all'area individuata per gli occhi, in orizzontale alla fascia determinata dalla stessa porzione. Nel caso in cui gli occhi per qualche motivo non vengono individuati, lo spazio di ricerca del naso viene impostato come l'intera immagine del volto.

Data una nuova immagine iniziale per la ricerca, si passa al rilevamento della componente del naso e alla costruzione dell'identikit con la stessa procedura utilizzata per gli occhi. L'unica accortezza è che il punto di riferimento relativo agli occhi, utilizzato nella procedura di estrazione dei parametri geometrici *getNoseFeatures*, deve essere considerato in riferimento alla nuova immagine ovvero la porzione del volto rilevata come naso.

Anche al termine della costruzione della lista *nose* relativa al naso, che include il calcolo del parametro globale *noseHeight*, viene opportunamente ristretta l'area del volto in cui riercare la bocca.

Infine per la bocca viene utilizzato ancora una volta lo stesso approccio per arrivare a determinare gli elementi di catalogo e i valori di verosimiglianza da inserire nella lista *mouth* e i due parametri globali *mouthPosition* e *mouthWidth*.

Segue il codice 6.37, relativo al metodo appena descritto.

Codice 6.37: **identikit.cpp - createIdentikit**

```
66 /* CREAZIONE DELL'IDENTIKIT A PARTIRE DALL'IMMAGINE imgaddress
      SECONDO IL CATALOGO COMPLETO cat SE rc E' FALSO, CON IL
      CATALOGO cat LIMITATO AI COMPONENTI DI identikit_rc SE rc E'
      VERO.
67  */
68 int Identikit::createIdentikit(char* imgaddress, Catalog* cat,
      Identikit* identikit_rc, bool rc)
69 {
70     cvNamedWindow("Face", 1);
```

```
71    cvNamedWindow("Eyes", 1);
72    cvNamedWindow("Nose", 1);
73    cvNamedWindow("Mouth", 1);
74    IplImage* face;
75    IplImage* eyes_img;
76    IplImage* nose_img;
77    IplImage* mouth_img;
78
79    // Acquisizione dell'immagine e conversione in scala di grigi
80    const char* filename = imgaddress;
81    IplImage* original_img = cvLoadImage(filename, 1);
82    IplImage* image = cvCreateImage(cvSize(original_img->width,
          original_img->height), 8, 1);
83    cvCvtColor(original_img, image, CV_RGB2GRAY);
84
85    // Rilevazione del volto
86    CvRect* face_rect = detectFace(image);
87
88    if(face_rect)
89    {
90       // Estrazione della porzione dell'immagine relativa al volto
91       face = getSubImage(image, face_rect);
92       cvShowImage("Face", face);
93
94       CvRect nose_init_space;
95       IplImage* nose_init_img;
96       CvRect mouth_init_space;
97       IplImage* mouth_init_img;
98
99       // Estremi esterni degli occhi (la distanza viene usata come
             riferimento)
100      CvPoint rif[2];
101      CvPoint rif_n[2];
102      CvPoint rif_m[2];
103
104      // Ricerca dell'area degli occhi
105      CvRect* eyes_rect = detectEyes(face);
106      if(eyes_rect)
107      {
108         // Estrazione dell'area
109         eyes_img = getSubImage(face, eyes_rect);
110
111         // Ricerca dei punti caratteristici e calcolo dei
                parametri geometrici
112         IplImage* eyes_corner =
               cvCreateImage(cvSize(eyes_img->width,
               eyes_img->height), 8, 1);
113         EyesFeatures* e_ft = new EyesFeatures();
114         eyesDistance = e_ft->getEyesFeatures(eyes_img,
               eyes_corner, rif);
115         cvShowImage("Eyes", eyes_corner);
116
117         if(rc)
118         {
119            // Creazione dell'identikit per la componente occhi
                  nel caso di catalogo ridotto
120            ListNode* element_rc = identikit_rc->eyes->getFirst();
```

```
121             while(element_rc != NULL)
122             {
123                 int id_cat_el = element_rc->getID();
124                 EyesFeatures* e_ft_element_rc =
                        &(cat->eyesCat[id_cat_el]);
125                 double lh = calcEyesLikelihood(e_ft_element_rc,
                        e_ft);
126                 if(lh > 0)
127                     eyes->insert(id_cat_el, lh);
128                 element_rc = element_rc->getNext();
129             }
130         }
131         else
132         {
133             // Creazione dell'identikit per la componente occhi
                    nel caso di catalogo completo
134             int n_eyes_cat = cat->countEyes;
135             for(int i=0; i<n_eyes_cat; i++)
136             {
137                 EyesFeatures* e_ft_element_cat = &(cat->eyesCat[i]);
138                 double lh = calcEyesLikelihood(e_ft_element_cat,
                        e_ft);
139                 if(lh > 0)
140                     eyes->insert(i, lh);
141             }
142         }
143         if(eyes->length() > 0)
144         {
145             // Visualizzazione dell'elemento di catalogo piu'
                    somigliante
146             char str[100];
147             sprintf(str, "cat/eyes%i.bmp",
                        eyes->getElementAt(eyes->length()-1)+100);
148             const char* filename_best = str;
149             IplImage* img_best = cvLoadImage(filename_best,
                        CV_LOAD_IMAGE_COLOR);
150             cvNamedWindow("Catalog eyes", 1);
151             cvShowImage("Catalog eyes", img_best);
152         }
153
154         // Restrizione dell'area per la ricerca del naso
155         nose_init_space = cvRect(eyes_rect->x, eyes_rect->y,
                    eyes_rect->width, face->height-eyes_rect->y);
156         nose_init_img = getSubImage(face, &nose_init_space);
157     }
158     else
159     {
160         // Restrizione dell'area per la ricerca del naso se gli
                occhi non sono stati individuati.
161         nose_init_space = cvRect(0, 0, face->width, face->height);
162         nose_init_img = face;
163     }
164
165     // Ricerca dell'area del naso
166     CvRect* nose_rect = detectNose(nose_init_img);
167     double a = 0.1;
168     int nr_x = nose_rect->x + nose_init_space.x -
```

```
            (int)((nose_rect->width)*a);
169         int nr_y = nose_rect->y + nose_init_space.y;
170         int nr_w = nose_rect->width + 2*(int)((nose_rect->width)*a);
171         int nr_h = nose_rect->height;
172         *nose_rect = cvRect(nr_x, nr_y, nr_w, nr_h);
173
174         if(nose_rect)
175         {
176            int x_n_offset = eyes_rect->x - nose_rect->x;
177            int y_n_offset = eyes_rect->y - nose_rect->y;
178            rif_n[0] = cvPoint(x_n_offset + rif[0].x, y_n_offset +
                  rif[0].y);
179            rif_n[1] = cvPoint(x_n_offset + rif[1].x, y_n_offset +
                  rif[1].y);
180
181            // Estrazione dell'area
182            nose_img = getSubImage(face, nose_rect);
183
184            // Ricerca dei punti caratteristici e calcolo dei
                  parametri geometrici
185            IplImage* nose_with_points =
                  cvCreateImage(cvSize(nose_img->width,
                  nose_img->height), 8, 1);
186            NoseFeatures* n_ft = new NoseFeatures();
187            noseHeight = n_ft->getNoseFeatures(nose_img,
                  nose_with_points, rif_n);
188            cvShowImage("Nose", nose_with_points);
189
190            if(rc)
191            {
192               // Creazione dell'identikit per la componente naso nel
                     caso di catalogo ridotto
193               ListNode* element_rc = identikit_rc->nose->getFirst();
194               while(element_rc != NULL)
195               {
196                  int id_cat_el = element_rc->getID();
197                  NoseFeatures* n_ft_element_rc =
                        &(cat->nosesCat[id_cat_el]);
198                  double lh = calcNoseLikelihood(n_ft_element_rc,
                        n_ft);
199                  if(lh > 0)
200                     nose->insert(id_cat_el, lh);
201                  element_rc = element_rc->getNext();
202               }
203            }
204            else
205            {
206               // Creazione dell'identikit per la componente naso nel
                     caso di catalogo completo
207               int n_noses_cat = cat->countNoses;
208               for(int i=0; i<n_noses_cat; i++)
209               {
210                  NoseFeatures* n_ft_element_cat =
                        &(cat->nosesCat[i]);
211                  double lh = calcNoseLikelihood(n_ft_element_cat,
                        n_ft);
212                  if(lh > 0)
```

```
213                 nose->insert(i, lh);
214             }
215         }
216         if(nose->length() > 0)
217         {
218             // Visualizzazione dell'elemento di catalogo piu'
                    somigliante
219             char str[100];
220             sprintf(str, "cat/nose%i.bmp",
                    nose->getElementAt(nose->length()-1)+100);
221             const char* filename_best = str;
222             IplImage* img_best = cvLoadImage(filename_best,
                    CV_LOAD_IMAGE_COLOR);
223             cvNamedWindow("Catalog nose", 1);
224             cvShowImage("Catalog nose", img_best);
225         }
226
227         // Restrizione dell'area per la ricerca della bocca
228         mouth_init_space = cvRect(0, nose_rect->y +
                cvRound(nose_rect->height*0.8), face->width,
                face->height-nose_rect->y -
                cvRound(nose_rect->height*0.5));
229         mouth_init_img = getSubImage(face, &mouth_init_space);
230     }
231     else
232     {
233         // Restrizione dell'area per la ricerca della bocca se il
                naso non e' stato individuato.
234         mouth_init_space = cvRect(0, 0, face->width,
                face->height);
235         mouth_init_img = face;
236     }
237
238     // Ricerca dell'area della bocca
239     CvRect* mouth_rect = detectMouth(mouth_init_img);
240     int x_border = (int)(mouth_rect->width*0.3);
241     int y_border = (int)(mouth_rect->height*0.2);
242     *mouth_rect = cvRect(mouth_rect->x + mouth_init_space.x -
            x_border, mouth_rect->y + mouth_init_space.y - y_border,
            mouth_rect->width + (2*x_border), mouth_rect->height +
            (2*y_border));
243
244     if(mouth_rect)
245     {
246         int x_m_offset = eyes_rect->x - mouth_rect->x;
247         int y_m_offset = eyes_rect->y - mouth_rect->y;
248         rif_m[0] = cvPoint(x_m_offset + rif[0].x, y_m_offset +
                rif[0].y);
249         rif_m[1] = cvPoint(x_m_offset + rif[1].x, y_m_offset +
                rif[1].y);
250
251         // Estrazione dell'area
252         mouth_img = getSubImage(face, mouth_rect);
253
254         // Ricerca dei punti caratteristici e calcolo dei
                parametri geometrici
255         IplImage* mouth_with_points =
```

```
                cvCreateImage(cvSize(mouth_img->width,
                mouth_img->height), 8, 1);
256             MouthFeatures* m_ft = new MouthFeatures();
257             double m_param[2];
258             m_ft->getMouthFeatures(mouth_img, mouth_with_points,
                    rif_m, m_param);
259             mouthPosition = m_param[0];
260             mouthWidth = m_param[1];
261             cvShowImage("Mouth", mouth_with_points);
262
263             if(rc)
264             {
265                 // Creazione dell'identikit per la componente bocca
                        nel caso di catalogo ridotto
266                 ListNode* element_rc = identikit_rc->mouth->getFirst();
267                 while(element_rc != NULL)
268                 {
269                     int id_cat_el = element_rc->getID();
270                     MouthFeatures* m_ft_element_rc =
                            &(cat->mouthsCat[id_cat_el]);
271                     double lh = calcMouthLikelihood(m_ft_element_rc,
                            m_ft);
272                     if(lh > 0)
273                         mouth->insert(id_cat_el, lh);
274                     element_rc = element_rc->getNext();
275                 }
276             }
277             else
278             {
279                 // Creazione dell'identikit per la componente bocca
                        nel caso di catalogo completo
280                 int n_mouths_cat = cat->countMouths;
281                 for(int i=0; i<n_mouths_cat; i++)
282                 {
283                     MouthFeatures* m_ft_element_cat =
                            &(cat->mouthsCat[i]);
284                     double lh = calcMouthLikelihood(m_ft_element_cat,
                            m_ft);
285                     if(lh > 0)
286                         mouth->insert(i, lh);
287                 }
288             }
289             if(mouth->length() > 0)
290             {
291                 // Visualizzazione dell'elemento di catalogo piu'
                        somigliante
292                 char str[100];
293                 sprintf(str, "cat/mouth%i.bmp",
                        mouth->getElementAt(mouth->length()-1)+100);
294                 const char* filename_best = str;
295                 IplImage* img_best = cvLoadImage(filename_best,
                        CV_LOAD_IMAGE_COLOR);
296                 cvNamedWindow("Catalog mouth", 1);
297                 cvShowImage("Catalog mouth", img_best);
298             }
299         }
300     }
```

```
301     cvReleaseImage(&original_img);
302     cvReleaseImage(&image);
303     cvReleaseImage(&face);
304     cvReleaseImage(&eyes_img);
305     cvReleaseImage(&nose_img);
306     cvReleaseImage(&mouth_img);
307     return 0;
308 }
```

## Confronto tra identikit

Il confronto tra identikit implementa l'algoritmo definito nel capitolo 5 che parte dai valori di verosimiglianza dei due identikit a confronto per calcolarne un grado di somiglianza a livello di elemento di catalogo. A partire da questi valori viene calcolato un grado di somiglianza a livello di componente che porta a un grado di somiglianza globale che permette di decretare se i due identikit si possono considerare relativi allo stesso soggetto o meno.

Il metodo (codice 6.38) esegue un ciclo che considera le componenti una alla volta e recupera le liste relative alla componente nei due diversi identikit che non hanno lo stesso ruolo. Infatti l'identikit *new_id* è relativo al soggetto da autenticare ed è costruito a partire dall'identikit dell'utente dichiarato *saved_id*. Pertanto, ogni elemento di catalogo contenuto nelle liste di *new_id* sicuramente sarà presente anche nelle liste di *saved_id*.

Vengono infatti selezionati gli elementi di catalogo della lista relativa a quest'ultimo identikit e per ciascuno di essi viene verificata la presenza all'interno della lista relativa all'identikit del soggetto da identificare. Se l'elemento è presente in entrambi gli identikit, viene valutata la somiglianza a livello di catalogo, considerando il minimo tra i due valori di verosimiglianza associati all'elemento di catalogo in questione. Tra tutti i gradi di somiglianza viene memorizzato il massimo nel vettore *max_lh_comp*, nella posizione relativa alla componente, determinando così il grado di somiglianza a livello di componente come definito in precedenza.

Una volta determinati i gradi di somiglianza per tutte le componenti, si può calcolare la funzione che dà un valore al grado di somiglianza globale. Come accennato in precedenza non vengono considerati i parametri globali e il peso dato a ciascuna componente nella somma è lo stesso, ovvero $\frac{1}{3}$ dato che le componenti sono tre.

Il valore dato in uscita è la risposta positiva o negativa (*true* o *false*) alla richesta di autenticazione e viene valutato in base al confronto con la soglia di autenticazione prefissata, anch'essa valutata in fase di sperimentazione.

Codice 6.38: **identikit.cpp - identify**

```
310 /* CONFRONTO TRA DUE IDENTIKIT PER VERIFICARE SE CORRISPONDONO
       ALLA STESSO SOGGETTO.
311  */
312 bool identify(Identikit* saved_id, Identikit* new_id)
313 {
314     // Calcolo verosimiglianza delle 3 componenti: occhi, naso,
           bocca
315     double max_lh_comp[3] = {0,0,0};
316     for(int i=0; i<3; i++)
```

```
317    {
318       List* s_id_list;
319       List* n_id_list;
320       if(i==0)
321       {
322          s_id_list = saved_id->eyes;
323          n_id_list = new_id->eyes;
324       }
325       else if(i==1)
326       {
327          s_id_list = saved_id->nose;
328          n_id_list = new_id->nose;
329       }
330       else
331       {
332          s_id_list = saved_id->mouth;
333          n_id_list = new_id->mouth;
334       }
335       int n_elements = s_id_list->length();
336       ListNode* element_s = s_id_list->getFirst();
337       for(int j=0; j<n_elements; j++)
338       {
339          int id_s = element_s->getID();
340          ListNode* element_n = n_id_list->getFirst();
341          while(element_n != NULL)
342          {
343             if(element_n->getID() == id_s)
344             {
345                // Se viene trovato un elemento comune, viene
                      calcolato il grado di somiglianza a livello di
                      elemento di catalogo
346                double lh_n = element_n->getLikelihood();
347                double lh_s = element_s->getLikelihood();
348                double lh;
349                if(lh_n < lh_s)
350                   lh = lh_n;
351                else
352                   lh = lh_s;
353                if(lh > max_lh_comp[i])
354                   max_lh_comp[i] = lh;
355                break;
356             }
357             else
358             {
359                element_n = element_n->getNext();
360             }
361          }
362          element_s = element_s->getNext();
363       }
364    }
365
366    // Calcolo della funzione di valutazione complessiva
367    double tot_lh = (max_lh_comp[0] + max_lh_comp[1] +
          max_lh_comp[2])/3;
368    if(tot_lh < AUTHENTICATION_TH)
369       return false;
370    else
```

```
371         return true;
372 }
```

## Gestione degli utenti

Nel programma principale (sezione 6.1), oltre alla gestione dell'identikit, viene richiesta anche la gestione degli utenti del sistema in maniera tale che sia possibile salvare i dati in fase di registrazione e recuperarli in fase di autenticazione.

Per questi motivi viene creata la classe *User* (codice 6.39) che permette di gestire i dati relativi a un utente e di salvare e caricare tali informazioni attraverso i file *users.txt* e *countUsers.txt*.

I metodi costruttori permettono di inizializzare un utente senza parametri, nel caso in cui i parametri siano caricati successivamente da file, oppure fornendo tutti i dati necessari, nel caso in cui l'utente debba essere creato e registrato.

Gli altri due metodi necessari sono *saveUser* e *loadUser*, che permettono rispettivamente di registrare l'utente nella base di dati e caricare tutti i dati dell'utente identificato da nome e cognome passati come input.

La procedura *saveUser* per prima cosa incrementa il contatore inserito nel file *countUsers.txt*. Successivamente memorizza i parametri dell'utente cioè il codice numerico, il nome, il cognome e l'identikit nel file *users.txt* secondo una sintassi prefissata in cui ogni riga corrisponde ai dati di un utente diverso. I dati, per ogni riga, sono così organizzati:

- I primi tre parametri sono il codice numerico, il nome e il cognome.
- Il parametro successivo è un valore che indica il numero di elementi di catalogo nell'identikit dell'utente, per la componente occhi.
- Seguono tante coppie di valori quante indicate dal numero appena inserito. Queste indicano l'identificatore dell'elemento di catalogo e la verosimiglianza associata per tutti gli elementi della lista relativa agli occhi nell'identikit.
- Il parametro successivo indica il numero di elementi di catalogo nell'identikit dell'utente, per la componente naso.
- Seguono tante coppie di valori quante indicate dal numero appena inserito, le quali indicano l'identificatore dell'elemento di catalogo e la verosimiglianza associata per tutti gli elementi della lista relativa al naso nell'identikit.
- Il parametro successivo indica il numero di elementi di catalogo nell'identikit dell'utente, per la componente bocca.
- Seguono tante coppie di valori quante indicate dal numero appena inserito, le quali indicano l'identificatore dell'elemento di catalogo e la verosimiglianza associata per tutti gli elementi della lista relativa alla bocca nell'identikit.
- Infine vengono inseriti i quattro parametri globali dell'identikit, secondo il seguente ordine: *eyesDistance, noseHeight, mouthPosition, mouthWidth*.

Il metodo *loadUser* per prima cosa recupera il numero totale di utenti. Dopodichè, riga per riga, analizza i dati degli utenti registrati nel file *users.txt* fino a trovare l'utente corripondente ai dati dichiarati in ingresso. Solo quando lo trova, assegna ai parametri dell'utente i dati rilevati nel file, inclusa la ricostruzione completa dell'identikit.

Si osserva che una migliore organizzazione dei dati porterebbe a gestire queste procedure in maniera meno complessa e più efficiente, in quanto la mancata indicizzazione all'interno del file, porta alla necessità di scorrere l'intera lista di utenti.

Codice 6.39: **identikit.cpp - class User**

```
374 /* CLASSE USER
375  * Ha come attributi un codice identificativo, nome e cognome
       dell'utente e l'identikit associato.
376  * I metodi permettono di costruire un utente, salvarlo nella base
       di dati e caricarlo dalla base di dati.
377  */
378 class User
379 {
380    public:
381       int id;
382       char* name;
383       char* surname;
384       Identikit* identikit;
385       User();
386       User(char* n, char* s, Identikit* ident);
387       void saveUser();
388       bool loadUser(char* nm, char* snm);
389 };
390
391 /* INIZIALIZZAZIONE DI UN UTENTE DA CARICARE IN UN SECONDO MOMENTO.
392  */
393 User::User()
394 {
395    name = "";
396    surname = "";
397    identikit = new Identikit();
398 }
399
400 /* INIZIALIZZAZIONE DI UN UTENTE A PARTIRE DAGLI ATTRIBUTI IN
       INPUT.
401  */
402 User::User(char* n, char* s, Identikit* ident)
403 {
404    name = n;
405    surname = s;
406    identikit = ident;
407 }
408
409 /* SALVATAGGIO DELL'UTENTE NELLA BASE DI DATI.
410  */
411 void User::saveUser()
412 {
413    // Acquisizione del numero di utenti
414    FILE * countFile = NULL;
415    int count = 1;
```

```
416     countFile = fopen("db/countusers.txt", "r");
417     if (countFile != NULL)
418     {
419        fscanf(countFile, "%i", &count);
420        count++;
421        fclose(countFile);
422     }
423     this->id = count;
424
425     // Aggiornamento del contatore
426     countFile = fopen("db/countusers.txt", "w");
427     if(countFile != NULL)
428     {
429        fprintf(countFile, "%i", count);
430        fclose(countFile);
431     }
432
433     FILE * userFile = NULL;
434     userFile = fopen("db/users.txt", "a");
435     if (userFile != NULL)
436     {
437        // Dati dell'utente e lista relativa alla componente bocca
438        int n_eyes = this->identikit->eyes->length();
439        fprintf(userFile, "%d %s %s %d ", this->id, this->name,
              this->surname, n_eyes);
440        ListNode* element = this->identikit->eyes->getFirst();
441        for(int i=0; i<n_eyes; i++)
442        {
443           fprintf(userFile, "%d %f ", element->getID(),
                 element->getLikelihood());
444           element = element->getNext();
445        }
446
447        // Lista relativa alla componente naso
448        int n_noses = this->identikit->nose->length();
449        fprintf(userFile, "%d ", n_noses);
450        element = this->identikit->nose->getFirst();
451        for(int i=0; i<n_noses; i++)
452        {
453           fprintf(userFile, "%d %f ", element->getID(),
                 element->getLikelihood());
454           element = element->getNext();
455        }
456
457        // Lista relativa alla componente bocca
458        int n_mouths = this->identikit->mouth->length();
459        fprintf(userFile, "%d ", n_mouths);
460        element = this->identikit->mouth->getFirst();
461        for(int i=0; i<n_mouths; i++)
462        {
463           fprintf(userFile, "%d %f ", element->getID(),
                 element->getLikelihood());
464           element = element->getNext();
465        }
466
467        // Altri parametri
468        fprintf(userFile, "%f %f %f %f\n",
```

```
            this->identikit->eyesDistance,
            this->identikit->noseHeight,
            this->identikit->mouthPosition,
            this->identikit->mouthWidth);
469     }
470     fclose(userFile);
471 }
472
473 /* CARICAMENTO DI UN UTENTE DALLA BASE DI DATI A PARTIRE DA NOME E
        COGNOME.
474  */
475 bool User::loadUser(char* nm_dec, char* snm_dec)
476 {
477     // Acquisizione del numero di utenti
478     FILE * countFile = NULL;
479     int count = 0;
480     countFile = fopen("db/countusers.txt", "r");
481     if (countFile != NULL)
482     {
483         fscanf(countFile, "%i", &count);
484         fclose(countFile);
485     }
486
487     FILE * userFile = NULL;
488     userFile = fopen("db/users.txt", "r");
489     if (userFile != NULL)
490     {
491         for(int j=0; j<count; j++)
492         {
493             int id_f;
494             char nm_f[20];
495             char snm_f[20];
496             int n_e_f;
497             int res = fscanf(userFile, "%d %s %s %d ", &id_f, nm_f,
                    snm_f, &n_e_f);
498             if(res <= 0)
499                 break;
500             bool found = false;
501             if((strcmp(nm_f, nm_dec) == 0) && (strcmp(snm_f, snm_dec)
                    == 0))
502             {
503                 found = true;
504                 id = id_f;
505                 name = nm_f;
506                 surname = snm_f;
507             }
508
509             // Occhi
510             for(int i=0; i<n_e_f; i++)
511             {
512                 int eye_id;
513                 double lh;
514                 fscanf(userFile, "%d %lf ", &eye_id, &lh);
515                 if(found)
516                     identikit->eyes->insert(eye_id, lh);
517             }
518
```

```
519             // Naso
520             int n_n_f;
521             fscanf(userFile, "%d ", &n_n_f);
522             for(int i=0; i<n_n_f; i++)
523             {
524                 int nose_id;
525                 double lh;
526                 fscanf(userFile, "%d %lf ", &nose_id, &lh);
527                 if(found)
528                     identikit->nose->insert( nose_id, lh );
529             }
530 
531             // Bocca
532             int n_m_f;
533             fscanf(userFile, "%d ", &n_m_f);
534             for(int i=0; i<n_m_f; i++)
535             {
536                 int mouth_id;
537                 double lh;
538                 fscanf(userFile, "%d %lf ", &mouth_id, &lh);
539                 if(found)
540                     identikit->mouth->insert(mouth_id, lh);
541             }
542 
543             // Altri parametri
544             double ed;
545             double nh;
546             double mp;
547             double mw;
548             fscanf(userFile, "%lf %lf %lf %lf", &ed, &nh, &mp, &mw);
549             if(found)
550             {
551                 identikit->eyesDistance = ed;
552                 identikit->noseHeight = nh;
553                 identikit->mouthPosition = mp;
554                 identikit->mouthWidth = mw;
555                 return true;
556             }
557         }
558     }
559     fclose(userFile);
560     return false;
561 }
```

# Capitolo 7

# Analisi sperimentale

Con l'obiettivo di validare il sistema proposto, sono stati effettutati dei test attraverso l'esecuzione del programma. Questa fase di analisi mira a validare la parte del programma relativa al modello proposto, ovvero il modello geometrico, la costruzione dell'identikit e il confronto.

Per fare questo viene eseguita manualmente la fase di estrazione dei punti sull'immagine del volto, a partire dai quali viene definito il modello geometrico. Non viene effettuata la fase di *feature extraction* automatica poichè, introducendo una quantità considerevole di errore, andrebbe a compromettere i test sulle fasi successive del riconoscimento, sulle quali si vuole focalizzare l'analisi.

Per quanto riguarda le altre parti, il software mantiene la struttura presentata nel capitolo precedente. I test vengono effettuati modificando i seguenti parametri:

- il valore di soglia $TH$ per la decisione finale sul grado di somiglianza a livello globale;
- il valore di soglia $th_1$, distanza massima nella scelta degli elementi di catalogo per la componente *occhi*;
- il valore di soglia $th_2$, distanza massima nella scelta degli elementi di catalogo per la componente *naso*;
- il valore di soglia $th_3$, distanza massima nella scelta degli elementi di catalogo per la componente *bocca*;
- il numero massimo di elementi di catalogo per le liste dell'identikit relative a ciascuna componente ($MAX_l$).

L'obiettivo è quello di stabilire la combinazione di parametri che garantisca le migliori prestazioni, valutando *FAR* e *FRR* per i casi in cui i due indici di errore si equivalgono.

## 7.1 Test e risultati

I test sono stati effettuati su un una base di venti soggetti, per ciascuno dei quali sono state acquisite cinque fotografie del volto. Una di queste è usata per la registrazione dell'utente, le quattro rimanenti per effettuare le prove di autenticazione.

## Caratteristiche delle immagini

Le foto scattate presentano il volto nella sua visione frontale e in scala di grigi, coerentemente con le assunzioni fatte in fase di progettazione. È stato chiesto ai soggetti di mantenere un'espressione rilassata, con la bocca chiusa e gli occhi aperti, in maniera tale che le variazioni dovute all'espressione non penalizzassero in partenza il risultato.

Inoltre è stato inquadrato il volto in primo piano, in maniera tale da avere una grandezza del volto che renda facile l'individuazione dei punti caratteristici e ne garantisca la precisione. Questi sono stati posizionati manualmente con la stessa procedura adottata per le immagini degli elementi di catalogo, assegnando ai punti un colore specifico in maniera tale che si possano distinguere tra loro i pixel selezionati (vedi figura 7.1).

Per rilevare i punti, il sistema deve eseguire un'analisi pixel per pixel in tutta l'immagine. Per non rendere troppo pesante il carico di lavoro, le foto sono state opportunamente ritagliate e ridotte in immagini di dimensioni $700 \times 525$. La codifica è quella di un'immagine *bitmap* a 24 bit, ovvero a tre canali di profondità 8 bit.

## I test

Sono stati effettuati sessanta test, ovvero sono state sperimentate sessanta combinazioni diverse dei parametri. I risultati si riferiscono a un'operazione di "tuning" mirata e non riferita a tutte le possibili combinazioni. Questo approccio è stato utilizzato poichè non è stato possibile effettuare un gran numero di test attraverso la stessa esecuzione del programma.

L'idea iniziale di far variare i parametri attraverso la ripetizione dei test in cicli nidificati che facevano variare i parametri in questione, non è stata realizzabile in quanto dopo un certo numero di cicli il processo si fermava per un errore. Non avendo trovato una causa precisa all'errore, che avviene solo dopo un numero determinato di iterazioni, la supposizione è quella di un problema di gestione della memoria della libreria *OpenCV* che non viene del tutto liberata al momento dell'elimenazione dell'immagine.

Un test completo per una configurazione di parametri è costituito, come prima cosa, dalla registrazione dei venti utenti. Successivamente viene presa in considerazione una delle foto rimanenti alla volta e vengono effettuati venti test in cui viene dichiarata ogni volta un'identità diversa. In questo modo vengono effettuati $20 \times 4 \times 20 = 1600$ autenticazioni, tra le quali $20 \times 4 \times 1 = 80$ sono casi positivi che si riferiscono allo stesso soggetto, mentre i rimanenti 1520 sono casi negativi in cui l'identità dichiarata non corrisponde all'immagine del soggetto.

I cosiddetti "casi positivi", sono quelli in cui l'avvenuta autenticazione corrisponde a una risposta corretta del sistema, mentre l'autenticazione mancata è un falso negativo, che va ad aumentare la percentuale di errore *FRR*.

Per i "casi negativi", invece, l'avvenuta autenticazione è un falso positivo e va ad incrementare l'indice *FAR*, mentre la mancata autenticazione è la riposta corretta al sistema.

Per effettuare un test completo, è necessario avviare quattro blocchi di test consecutivi, per il motivo sopraccitato. Per ciascuno di essi vengono calcolati e salvati su file: la configurazione dei parametri, la percentuale di falsi positivi nel blocco e

quella di falsi negativi. Per ottenere gli indici di errore relativi al test completo, viene eseguita la media tra i rispettivi valori dei quattro blocchi.

## I risultati

Come già precisato in precedenza, il tuning dei parametri è stato eseguito cercando di puntare sempre a una configurazione che garantisse di bilanciare la percentuale di falsi positivi e quella di falsi negativi.

In primo luogo si è cercato di arrivare a un valore per le soglie $th_1$, $th_2$ e $th_3$, che garantisca che tutti gli utenti abbiano almeno un elemento di catalogo per ogni componente del volto. Successivamente, le valutazioni sui test da eseguire hanno tenuto conto delle seguenti considerazioni:

- l'aumento della soglia $TH$ aumenta i falsi negativi e diminuisce i falsi positivi.
- l'aumento della soglia $th_1$ aumenta i falsi positivi e diminuisce i falsi negativi.
- l'aumento della soglia $th_2$ aumenta i falsi positivi e diminuisce i falsi negativi.
- l'aumento della soglia $th_3$ aumenta i falsi positivi e diminuisce i falsi negativi.
- l'aumento della soglia $MAX_l$ aumenta i falsi positivi e diminuisce i falsi negativi.

I dati ottenuti sono contenuti nelle tabelle 7.1 e 7.2, che seguono.

| Test | $TH$ | $th_1$ | $th_2$ | $th_3$ | $MAX_l$ | **FAR** | **FRR** |
|---|---|---|---|---|---|---|---|
| 1 | 0,005 | 0,005 | 0,005 | 0,005 | 1 | 0,0007 | 1,0000 |
| 2 | 0,005 | 0,010 | 0,010 | 0,010 | 1 | 0,0066 | 0,9750 |
| 3 | 0,005 | 0,050 | 0,050 | 0,050 | 1 | 0,2546 | 0,2750 |
| 4 | 0,005 | 0,100 | 0,100 | 0,050 | 1 | 0,6849 | 0,0000 |
| 5 | 0,010 | 0,100 | 0,100 | 0,050 | 1 | 0,6776 | 0,0000 |
| 6 | 0,050 | 0,100 | 0,100 | 0,050 | 1 | 0,5730 | 0,0500 |
| 7 | 0,100 | 0,100 | 0,100 | 0,050 | 1 | 0,4395 | 0,0875 |
| 8 | 0,200 | 0,100 | 0,100 | 0,050 | 1 | 0,1961 | 0,2750 |
| 9 | 0,300 | 0,100 | 0,100 | 0,050 | 1 | 0,0513 | 0,4500 |
| 10 | 0,300 | 0,100 | 0,100 | 0,050 | 5 | 0,1493 | 0,2125 |
| 11 | 0,300 | 0,100 | 0,100 | 0,050 | 10 | 0,2053 | 0,1375 |
| 12 | 0,300 | 0,100 | 0,100 | 0,050 | 15 | 0,2349 | 0,1375 |
| 13 | 0,300 | 0,070 | 0,100 | 0,050 | 10 | 0,1283 | 0,3125 |
| 14 | 0,250 | 0,100 | 0,100 | 0,050 | 10 | 0,3020 | 0,1000 |
| 15 | 0,350 | 0,070 | 0,100 | 0,050 | 10 | 0,0743 | 0,4125 |
| 16 | 0,400 | 0,070 | 0,100 | 0,050 | 10 | 0,0428 | 0,5125 |
| 17 | 0,300 | 0,080 | 0,100 | 0,050 | 10 | 0,1546 | 0,2500 |
| 18 | 0,250 | 0,090 | 0,100 | 0,050 | 10 | 0,2717 | 0,1125 |
| 19 | 0,250 | 0,100 | 0,100 | 0,050 | 10 | 0,3020 | 0,1000 |
| 20 | 0,250 | 0,080 | 0,100 | 0,050 | 10 | 0,2382 | 0,1375 |

Tabella 7.1: Prima parte dei risultati dei test

| Test | $TH$ | $th_1$ | $th_2$ | $th_3$ | $MAX_l$ | **FAR** | **FRR** |
|---|---|---|---|---|---|---|---|
| 21 | 0,350 | 0,120 | 0,100 | 0,050 | 10 | 0,1803 | 0,1875 |
| 22 | 0,300 | 0,100 | 0,120 | 0,050 | 10 | 0,2520 | 0,1000 |
| 23 | 0,350 | 0,100 | 0,100 | 0,050 | 10 | 0,1382 | 0,2875 |
| 24 | 0,320 | 0,100 | 0,100 | 0,050 | 10 | 0,1816 | 0,2000 |
| 25 | 0,320 | 0,100 | 0,100 | 0,050 | 8 | 0,1612 | 0,2375 |
| 26 | 0,350 | 0,100 | 0,100 | 0,050 | 8 | 0,1217 | 0,3125 |
| 27 | 0,300 | 0,100 | 0,100 | 0,050 | 8 | 0,1842 | 0,1875 |
| 28 | 0,250 | 0,100 | 0,100 | 0,050 | 8 | 0,2822 | 0,1250 |
| 29 | 0,300 | 0,200 | 0,200 | 0,100 | 10 | 0,8000 | 0,0000 |
| 30 | 0,400 | 0,200 | 0,200 | 0,100 | 10 | 0,5868 | 0,0125 |
| 31 | 0,500 | 0,200 | 0,200 | 0,100 | 10 | 0,3112 | 0,0625 |
| 32 | 0,600 | 0,200 | 0,200 | 0,100 | 10 | 0,1217 | 0,2750 |
| 33 | 0,550 | 0,200 | 0,200 | 0,100 | 10 | 0,1980 | 0,1750 |
| 34 | 0,550 | 0,200 | 0,200 | 0,100 | 15 | 0,2414 | 0,1375 |
| 35 | 0,300 | 0,080 | 0,080 | 0,040 | 10 | 0,0954 | 0,3875 |
| 36 | 0,200 | 0,080 | 0,080 | 0,040 | 10 | 0,2717 | 0,1625 |
| 37 | 0,250 | 0,080 | 0,080 | 0,040 | 10 | 0,1691 | 0,2625 |
| 38 | 0,250 | 0,080 | 0,080 | 0,040 | 5 | 0,1237 | 0,3250 |
| 39 | 0,200 | 0,080 | 0,080 | 0,040 | 5 | 0,2217 | 0,2125 |
| 40 | 0,300 | 0,100 | 0,100 | 0,040 | 10 | 0,1908 | 0,1750 |
| 41 | 0,300 | 0,100 | 0,100 | 0,030 | 10 | 0,1737 | 0,2250 |
| 42 | 0,250 | 0,100 | 0,100 | 0,030 | 10 | 0,2678 | 0,1375 |
| 43 | 0,300 | 0,100 | 0,090 | 0,050 | 10 | 0,1842 | 0,1875 |
| 44 | 0,350 | 0,100 | 0,090 | 0,050 | 10 | 0,1184 | 0,3250 |
| 45 | 0,320 | 0,100 | 0,090 | 0,050 | 10 | 0,1553 | 0,2500 |
| 46 | 0,310 | 0,100 | 0,090 | 0,050 | 10 | 0,1730 | 0,2375 |
| 47 | 0,290 | 0,100 | 0,090 | 0,050 | 10 | 0,1980 | 0,1750 |
| 48 | 0,290 | 0,100 | 0,090 | 0,050 | 9 | 0,1901 | 0,2000 |
| 49 | 0,280 | 0,100 | 0,090 | 0,050 | 9 | 0,2072 | 0,1625 |
| 50 | 0,300 | 0,100 | 0,090 | 0,050 | 11 | 0,1868 | 0,1750 |
| 51 | 0,300 | 0,100 | 0,090 | 0,050 | 12 | 0,1928 | 0,1750 |
| 52 | 0,320 | 0,100 | 0,090 | 0,050 | 12 | 0,1645 | 0,2500 |
| 53 | 0,310 | 0,100 | 0,090 | 0,050 | 12 | 0,1822 | 0,2375 |
| 54 | 0,300 | 0,090 | 0,080 | 0,040 | 12 | 0,1217 | 0,3625 |
| 55 | 0,280 | 0,090 | 0,080 | 0,040 | 12 | 0,1520 | 0,2750 |
| 56 | 0,270 | 0,090 | 0,080 | 0,040 | 12 | 0,1711 | 0,2625 |
| 57 | 0,260 | 0,090 | 0,080 | 0,040 | 12 | 0,1959 | 0,1781 |
| 58 | 0,290 | 0,100 | 0,100 | 0,050 | 10 | 0,1989 | 0,1583 |
| 59 | 0,310 | 0,100 | 0,100 | 0,050 | 10 | 0,1868 | 0,1688 |
| 60 | 0,315 | 0,100 | 0,100 | 0,050 | 10 | 0,1842 | 0,1875 |

Tabella 7.2: Seconda parte dei risultati dei test

Dai risultati ottenuti possono essere estratti i casi più significativi, ovvero quelli in cui i due indici di errori sono bilanciati. Questi sono stati estratti in tabella 7.3 e

evidenziano che si arriva ad ottenere un indice di errore *EER* intorno al valore **0,18** per le configurazioni migliori.

| Test | $TH$ | $th_1$ | $th_2$ | $th_3$ | $MAX_l$ | **FAR** | **FRR** |
|---|---|---|---|---|---|---|---|
| 59 | 0,310 | 0,100 | 0,100 | 0,050 | 10 | 0,1868 | 0,1688 |
| 50 | 0,300 | 0,100 | 0,090 | 0,050 | 11 | 0,1868 | 0,1750 |
| 40 | 0,300 | 0,100 | 0,100 | 0,040 | 10 | 0,1908 | 0,1750 |
| 21 | 0,350 | 0,120 | 0,100 | 0,050 | 10 | 0,1803 | 0,1875 |
| 51 | 0,300 | 0,100 | 0,090 | 0,050 | 12 | 0,1928 | 0,1750 |
| 43 | 0,300 | 0,100 | 0,090 | 0,050 | 10 | 0,1842 | 0,1875 |
| 27 | 0,300 | 0,100 | 0,100 | 0,050 | 8 | 0,1842 | 0,1875 |
| 60 | 0,315 | 0,100 | 0,100 | 0,050 | 10 | 0,1842 | 0,1875 |

Tabella 7.3: Risultati significativi

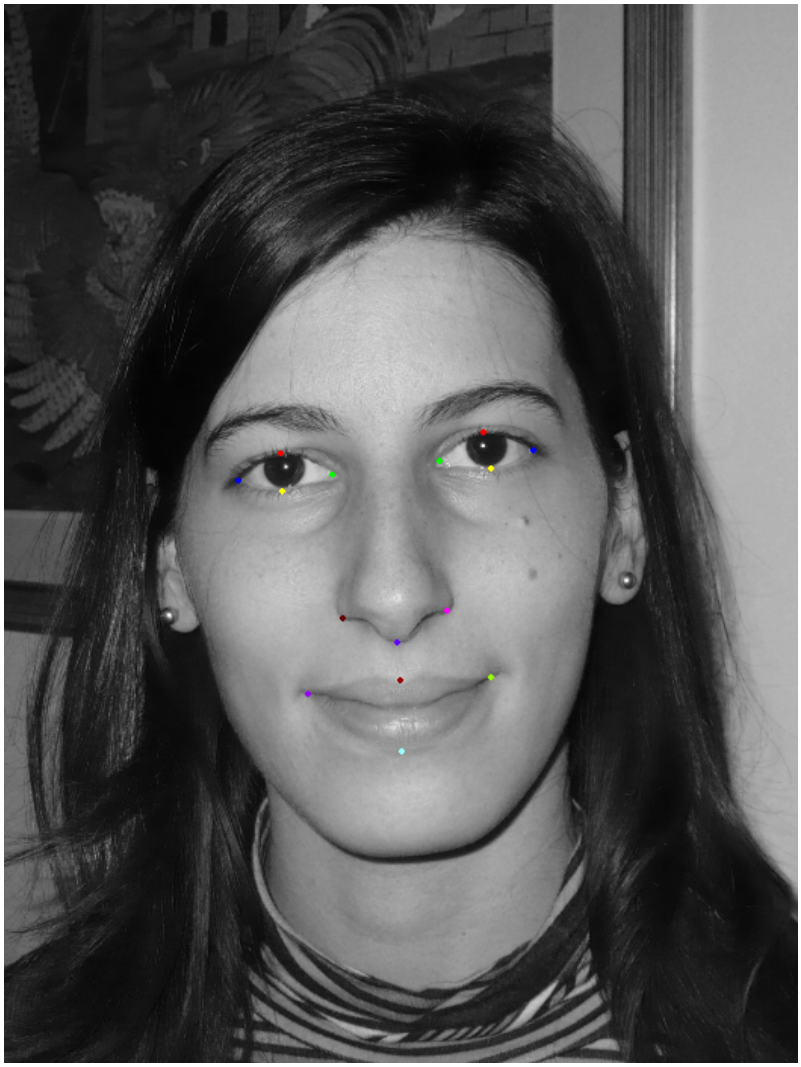

Figura 7.1: Immagine per i test, con i punti individuati manualmente.

## 7.2 Conclusioni

L'obiettivo della tesi è quello di proporre un nuovo approccio al problema di riconoscimento del volto attraverso il passaggio per un modello di descrizione basato sull'idea di identikit.

I risultati ottenuti non si possono ritenere all'altezza delle tecniche che sono già consolidate nel campo del riconoscimento del volto. Pertanto, per validare il sistema, sarebbe necessario considerare ogni sua parte per capire quali sono i punti in cui può essere migliorata, tenendo presente la struttura base dell'idea, ovvero la descrizione tramite elementi di un identikit per ciascun soggetto preso in esame. Inoltre dovrebbe essere aumentata anche la base dei soggetti del test, fattore da non sottovalutare in quanto aumenta la probabilità di avere soggetti che il sistema automatico può confondere l'uno con l'altro.

Un aspetto sicuramente positivo, nell'ottica del possibile miglioramento, è la modularità del sistema, caratteristica per la quale la modifica di una delle parti del sistema non va ad alterarne la struttura globale. Segue un'analisi che mira a descrivere i possibili miglioramenti, mostrando che i risultati ottenuti si possono considerare parzialmente significativi delle reali potenzialità del sistema che non sono state pienamente sfruttate.

### Catalogo e modello geometrico

Gli elementi del catalogo sono inseriti dal software *Faces* senza un criterio particolare, supponendo che costituiscano una buona base di partenza ovvero che possano rappresentare abbastanza accuratamante tutti i soggetti possibili per il sistema. Ogni elemento viene poi considerato come un punto in uno spazio $k$-dimensionale, dove $k$ è il numero di parametri rilevati per la componente del volto a cui l'elemento si riferisce.

La distribuzione dei punti relativi a tutti gli elementi di catalogo per una certa componente, non è stata presa in esame. È possibile, però, effettuare un'operazione di affinamento del catalogo, eliminando qualche elemento in maniera tale da garantire una distribuzione equa degli elementi nello spazio.

Quello che accade, infatti, è che la descrizione dei parametri geometrici è una semplificazione rispetto alla classificazione qualitativa fornita dal software. Di conseguenza due elementi che si distinguono per una caratteristica particolare, possono ritrovarsi molto vicini come punti nello spazio dei parametri geometrici. Non essendo controllato, questo aspetto può influire negativamente nella costruzione dell'identikit.

D'altro canto, si può ipotizzare che può essere sensato avere una densità maggiore in alcune zone dello spazio corrispondenti a caratteristiche molto comuni per la componente, nelle quali vengono proiettati più soggetti. Una presenza maggiore di elementi, potrebbe aiutare a distinguere meglio soggetti diversi.

Questi spunti rimangono da verificare e possono contribuire al miglioramento delle prestazioni globali del sistema.

## Costruzione dell'identikit

In fase di costruzione dell'identikit, due parametri per ciascuna componente sono stati oggetto dei test: la distanza massima e il numero massimo di elementi che possono essere inseriti in lista. Quest'ultimo, però, è stato considerato uguale per tutte le tre componenti considerate. Uno studio più accurato di questo parametro può essere eseguito cercando di scegliere il valore più opportuno per ciascuna componente.

Inoltre, sempre in questa fase, non sono stati assegnati pesi diversi ai parametri nel calcolo della distanza per ciascuna componente. Nella presentazione del modello, è stata proposta la distanza di Mahalanobis, che imposta i coefficienti moltiplicativi in base alla varianza del parametro e può considerarsi una prima soluzione per il confronto di parametri disomogenei tra loro. Lo studio dei coefficienti a questo livello, però, può essere ulteriormente ampliato per far sì che venga scelto ad hoc il peso di ciascun parametro in maniera tale da ottimizzare i risultati.

## Confronto finale

Anche nel confronto finale, non sono stati assegnati pesi diversi alle componenti del volto. Per ciascuna di esse, infatti, il grado di somiglianza viene inserito nella funzione di valutazione finale con lo stesso coefficiente. Anche la modifica di questi parametri può portare a migliorare i risultati del sistema, dando più peso alle componenti che sono più efficaci per il riconoscimento.

Infine non sono stati inseriti i parametri globali poichè il test ha come obiettivo la validazione del modello a identikit. Come stabilito nella descrizione teorica, questi possono essere valutati assieme ai gradi di somiglianza per componente, per arrivare a una descrizione ancora più completa della somiglianza tra i due volti a confronto.

Essi non sono stati inseriti nell modello geometrico per ciascuna componente in quanto non possono essere valutati rispetto a un elemento di catalogo che considera una sola componente. Sono però di grande importanza per la definizione del volto e, essendo misure di distanze più grandi rispetto a quelle interne alle componenti, si possono considerate misure nel complesso meno soggette ad errore. Possono dunque giocare un ruolo importante nel miglioramento delle prestazioni del sistema.

## Considerazioni sulla struttura dell'identikit

Un aspetto che va oltre il problema specifico considerato in questo lavoro, è quello della struttura dell'identikit che porta a classificare un soggetto in base alla somiglianza rispetto a dei modelli prefissati, ovvero gli elementi del catalogo.

Data una componente di un volto, è possibile derivare l'insieme di elementi giudicati come somiglianti. A partire da questi, è possibile ricavare a sua volta quali utenti registrati nel sistema assomigliano al volto in questione, secondo la componente specifica.

In un problema di verifica come quello considerato, questo dettaglio è irrilevante in quanto il confronto avviene tra due identikit determinati dai dati in input. In un problema di identificazione, invece, può risultare una caratteristica importante che permette di ridurre la quantità di utenti con i quali è necessario eseguire il confronto. L'utilità diventa rilevante soprattutto nei casi in cui il numero di utenti è molto elevato.

# Bibliografia


[1] *Faces 4.0 Edu Plus*, software developed by IQ Biometrix, Inc. with the collaboration of CogniScience, Inc. (http://www.facesid.com/)

[2] *OpenCV (Open Source Computer Vision Library)*, library of programming functions for real time computer vision. (http://opencv.willowgarage.com/wiki/)

[3] W. Zhao, R. Chellappa, P.J. Phillips, A. Rosenfeld, "Face Recognition: A Literature Survey", *ACM Computing Surveys, Vol. 35, No. 4, December 2003, pp. 399–458.*

[4] A.F. Abate, M. Nappi, D. Riccio, G. Sabatino, "2D and 3D face recognition: A survey", *Dipartimento di Matematica e Informatica, Universita' di Salerno, 2007.*

[5] R. Jafri, H.R. Arabnia, "A Survey of Face Recognition Techniques", *Journal of Information Processing Systems, Vol.5, No.2, June 2009, pp. 41-68.*

[6] X. Tan, S. Chen, Z.H. Zhou, F. Zhang, "Face Recognition from a Single Image per Person: A Survey", *Nanjing University, China, 2006.*

[7] M.H. Yang, D. J. Kriegman, N. Ahuja, "Detecting Faces in Images: A Survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 24, No. 1, January 2002.*

[8] C. Zhang, Z. Zhang, "A Survey of Recent Advances in Face Detection", *Microsoft Research, Microsoft Corporation, June 2010.*

[9] P. Viola, M. Jones, "Robust real-time object detection", *International Journal of Computer Vision, Vol. 57, No. 2, 2002, pp. 137–154.*

[10] L. Ding, A.M. Martinez, "Features versus Context: An Approach for Precise and Detailed Detection and Delineation of Faces and Facial Features", *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, No. 11, November 2010.*

[11] P.I. Wilson, J. Fernandez, "Facial Feature Detection Using Haar Classifiers", *CCSC: South Central Conference - JCSC 21, 4 (April 2006) pp. 127-133.*

[12] A. Majumder, L. Behera, V.K. Subramanian, "Automatic and Robust Detection of Facial Features in Frontal Face Images", *UKSim - 13th International Conference on Modelling and Simulation, 2011.*

[13] J. Canny, "A Computational Approach to Edge Detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, pp. 679–698, November 1986.*

[14] H. Moon, R. Chellappa, A. Rosenfeld, "Optimal Edge-Based Shape Detection", *IEEE Transactions on Image Processing, Vol. 11, No. 11, November 2002.*

[15] C. Harris, M.J. Stephens, "A combined corner and edge detector", *Alvey Vision Conference, pages 147–152, 1988.*

[16] C. Schmid, R. Mohr, C. Bauckhage, "Evaluation of interest point detectors", *International Journal of Computer Vision, 37(2):151–172, June 2000.*

[17] R. Brunelli, T. Poggio, "Face Recognition: Features versus Templates", *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 15, No. 10, pp. 1042-1052, October 1993.*

[18] M. Turk, A. Pentland, "Eigenfaces for Recognition", *Journal of Cognitive Neuroscience, Vol. 3, No. 1, pp. 71-86, 1991.*

[19] M. Welling, "Fisher Linear Discriminant Analysis", *Department of Computer Science, University of Toronto.*

[20] H. Gu, G. Su, C. Du, "Feature Points Extraction from Faces", *Research Institute of Image and Graphics, Department of Electronic Engineering, Tsinghua University, Beijing, China.*

[21] H.Q. Li, S.Y. Wang, F.H. Qi, "Automatic face recognition by support vector machines", *Combinatorial Image Analysis, Proceedings, Vol. 3322, Lecture Notes In Computer Science, pp. 716-725, 2004.*

[22] F.S. Samaria, "Face recognition using Hidden Markov Models", *Trinity College, University of Cambridge, Cambridge, UK, Ph. D. Thesis 1994.*

[23] A.V. Nefian, M.H. Hayes, "Face Recognition using an embedded HMM", *IEEE International Conference Audio Video Biometric based Person Authentication, pp. 19-24, 1999.*

[24] B. Li, H. Yin, "Face Recognition Using RBF Neural Networks and Wavelet Transform", *Advances in Neural Networks – ISNN 2005, Vol. 3497, Lecture Notes in Computer Science: Springer Berlin / Heidelberg, pp. 105-111, 2005.*

[25] P. Melin, C. Felix, O. Castillo, "Face recognition using modular neural networks and the fuzzy Sugeno integral for response integration", *International Journal of Intelligent Systems, Vol. 20, pp.275-291, 2005.*

[26] I.J. Cox, J. Ghosn, and P.N. Yianilos, "Featurebased face recognition using mixture-distance", *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pp.209-216, 1996.*

[27] L. Wiskott, J.M. Fellous, N. Krüger, C. Von Der Malsburg, "Face Recognition by Elastic Bunch Graph Matching", *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, pp. 775-779, 1997.*

[28] X. Wang, X. Tang, "Face Photo-Sketch Synthesis and Recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 31, 2009*

[29] B. Klare, A.K. Jain, "Sketch to Photo Matching: A Feature-based Approach", *Department of Brain and Cognitive Engineering, Korea University, Seoul, Korea, 2009.*

[30] B. Klare, Z. Li, A.K. Jain, "Matching Forensic Sketches to Mug Shot Photos", *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 33, No. 3, pp. 639-646, March 2011.*

[31] C. Torres, P. Perrot, H. Talbot, "Face Recognition Based On Facial Composite".