

FACE RECOGNITION

NACU ROBERTO LUCIAN

Realizzazione di un sistema per il riconoscimento
facciale

SOMMARIO

Obiettivo del lavoro è la realizzazione di un *Face Recognition System*. Più particolarmente, andremo ad implementare sul microcomputer Raspberry Pi uno degli algoritmi principali che ha permesso alle tecnologie di riconoscimento facciale di raggiungere un livello molto alto in termini di velocità e precisione; *Principal Component Analysis*, noto anche con il nome di ‘*Eigenfaces-based Algorithm*’.

INDICE

INTRODUZIONE.....	7
Capitolo 1. Sistemi di rilevazione e riconoscimento facciale	8
1.1 Struttura di un sistema di riconoscimento facciale	8
1.2 Face detection e le problematiche che la riguardano	8
Capitolo 2. Gli approcci scelti.....	10
2.1 Il metodo Viola-Jones	10
2.1.1 Ma come funziona esattamente il tutto?.....	10
2.2 Il metodo dell'analisi delle componenti principali.....	12
2.3 Il metodo EigenFaces	13
2.3.1 Il volto visto come un vettore.....	14
2.3.2 Lo spazio delle immagini	14
2.3.3 Autovettori nell'algebra lineare	16
2.4 Alcune premesse	17
Capitolo 3. Passiamo alla progettazione	18
3.1 Analisi e pianificazione.....	18
3.2 Introduzione ad OpenCV	20
3.3 Realizzazione del progetto	21
3.3.1 Rilevazione di un volto in un'immagine	21
3.3.2 Face tracking in un video	23
3.3.3 Primo sviluppo dell'algoritmo di riconoscimento.....	24
3.3.4 Creazione del database	27
3.3.5 Andiamo avanti con l'algoritmo di riconoscimento facciale ..	28
3.3.5.1 Analisi delle eigenfaces del nostro training set.....	30
3.3.5.2 Analisi della mean face del nostro training set	31
Capitolo 4. Risultati del progetto	32
4.1 Efficienza del sistema	32
4.2 Statistiche e sviluppi futuri	33
BIBLIOGRAFIA	34

INTRODUZIONE

Sistemi biometrici

Un sistema biometrico è un dispositivo automatico per la verifica di identità o l'identificazione di una persona sulla base di caratteristiche biologiche. Queste caratteristiche possono essere di varia natura e sono generalmente suddivise in fisiologiche (impronte digitali, volti, retina, iride, DNA) e comportamentali (voce, calligrafia, stile di battitura).

Face Recognition

Come tutte le soluzioni biometriche, le tecnologie di riconoscimento facciale misurano e estraggono le caratteristiche uniche di un individuo con il solo scopo di identificarlo o autenticarlo. Sfruttando spesso una fotocamera digitale, il software di riconoscimento facciale può rilevare volti nelle immagini, quantificare le loro caratteristiche e successivamente confrontarle con i modelli memorizzati in un database.

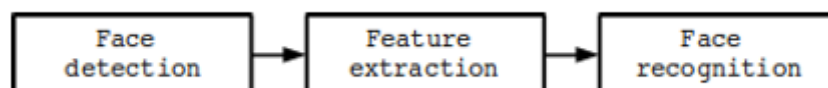
Il riconoscimento facciale, oltre a far parte del campo biometrico, è un'area molto attiva nella *Computer Vision* poichè è stata studiata per quasi 25 anni e sta producendo applicazioni utili in più ambiti partendo da quelli riguardanti la sicurezza fino ad arrivare a quelli più generici come la robotica o il semplice intrattenimento.

Capitolo 1.

Sistemi di rilevazione e riconoscimento facciale

1.1 Struttura di un sistema di riconoscimento facciale

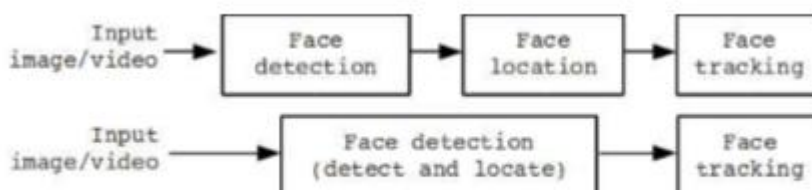
L'*input* di un sistema di riconoscimento facciale è sempre un'immagine o un video. L'*output* sarà invece l'identificazione o l'autenticazione del soggetto o dei soggetti presenti nell'immagine o nel video. Esso è un processo che può essere scomposto in 3 parti come nella seguente figura:



Il primo step si occupa di estrarre le facce dai diversi scenari. In questo modo, il sistema identifica una certa regione dell'immagine come una faccia. Il secondo ha il compito di ottenere caratteristiche facciali rilevanti. Nel terzo punto, invece, il sistema riconosce la faccia. Quest'ultima parte include un algoritmo di classificazione e una funzione di comparazione al fine di ottenere l'identità del soggetto da un *database*.

1.2 Face detection e le problematiche che la riguardano

Il concetto di *face detection*, così come quello di *face recognition*, include più sotto-processi che possono essere scomposti come nella seguente figura:



1. Sistemi di rilevazione e riconoscimento facciale

Prima di tutto, viene eseguita una riduzione della dimensione dell'immagine in modo tale da ottenere un tempo di risposta ammissibile dal sistema. Successivamente, alcuni algoritmi analizzano l'immagine così come è, mentre altri si occupano di estrarre regioni facciali rilevanti. Una volta fatto ciò, vengono estratte le caratteristiche facciali che verranno poi pesate, valutate e confrontate per decidere se è presente una faccia e dove si trova.

La rilevazione di un volto, tuttavia, deve avere a che fare con diverse problematiche. Questo è dovuto principalmente al fatto che, come ad esempio nei sistemi di video sorveglianza, non si può avere un controllo totale dell'ambiente. Proprio per questo motivo, ogni *face detection system* va incontro a problemi che sono determinati dai seguenti fattori:

- *Variazione della posa*: lo scenario ideale per la rilevazione facciale sarebbe uno nel quale vengono coinvolte solo immagini frontali. Come detto prima, però, ciò è quasi impossibile in condizioni incontrollate. La *performance* degli algoritmi di *face detection* cala drasticamente quando ci sono grandi variazioni di posa.
- *Occlusione delle caratteristiche*: la presenza di elementi quali barba, occhiali o cappelli introduce un'alta variabilità che riduce l'efficienza del sistema.
- *Espressioni facciali*.
- *Condizioni della cattura dell'immagine*: fotocamere differenti e condizioni ambientali possono compromettere la qualità dell'immagine, andando ad influenzare la comparsa di una faccia.

Ora che abbiamo un quadro generale del significato dei termini *face detection* e *face recognition* e le problematiche alle quali andremo incontro cercando di sviluppare un sistema di riconoscimento facciale, possiamo procedere con la spiegazione dell'approccio utilizzato.

Capitolo 2.

Gli approcci scelti

2.1 Il metodo Viola-Jones

Il rilevamento del volto è stato effettuato attraverso l'uso di un algoritmo di *face detection* automatico. Tra i numerosi metodi presenti in letteratura, si è scelto di utilizzare il metodo proposto da Viola-Jones ^[1], per le sue caratteristiche di accuratezza ed efficienza computazionale. Una possibile implementazione di tale algoritmo è possibile trovarla all'interno delle librerie Intel OpenCV ^[2], nella sezione dedicata agli esempi d'uso di queste ultime. L'obiettivo è quello di ottenere il cosiddetto “classificatore a cascata per volti frontali” da poter utilizzare per localizzare volti all'interno di immagini. L'algoritmo è in grado di rilevare volti in posa approssimativamente frontale, ovunque essi siano presenti nell'immagine e l'*output* è costituito da una serie di regioni rettangolari, ognuna centrata su un volto e i cui limiti racchiudono il volto stesso.

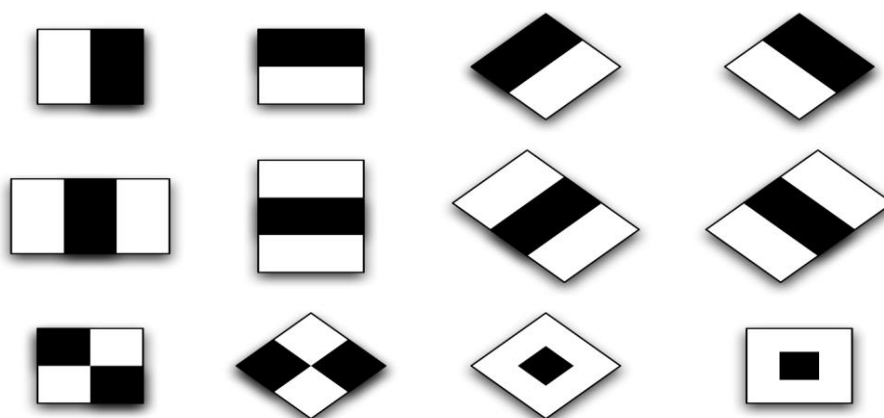
2.1.1 Ma come funziona esattamente il tutto?

Viola e Jones, tramite le *Haar wavelets* (onde oscillanti di lunghezza finita), hanno creato quelle che sono oggi note come *Haar-like features*. Una *Haar-like feature* considera rettangoli in una specifica locazione della finestra di individuazione (*detection window*), calcola l'intensità di ogni rettangolo come somma delle intensità dei pixel che lo compongono ed effettua la differenza di valori tra regioni adiacenti. Per ogni regione dell'immagine, vengono scalati ed applicati diverse volte gli operatori. In questo modo viene effettuata una catalogazione tra le sottosezioni di un'immagine.

Più particolarmente, ogni operatore permette di valutare, mediante una differenza normalizzata di pixel, quando una regione di immagine è importante per essere prima memorizzata e poi analizzata in fase di riconoscimento. Per ognuna delle sue sottoregioni, un operatore effettua la somma dei relativi pixel dell'immagine analizzata. Ottenute queste somme parziali, ne effettua la somma algebrica considerandone alcune positive ed altre negative. Normalizzando quest'ultimo risultato, lo si valuta e, se

2. Gli approcci scelti

questo fosse alto, significa che un operatore, in quella particolare zona dell'immagine originale, ha una grande importanza e rappresenta una feature.



Haar wavelet

Questi classificatori permettono di distinguere, per esempio, la regione degli occhi da quella delle guance, in quanto quest'ultima è generalmente molto più chiara della prima. Sebbene questo *framework* sia nato per il riconoscimento di visi, basandosi su una fase di apprendimento dell'oggetto da ricercare, può essere addestrato a riconoscere qualsiasi cosa.

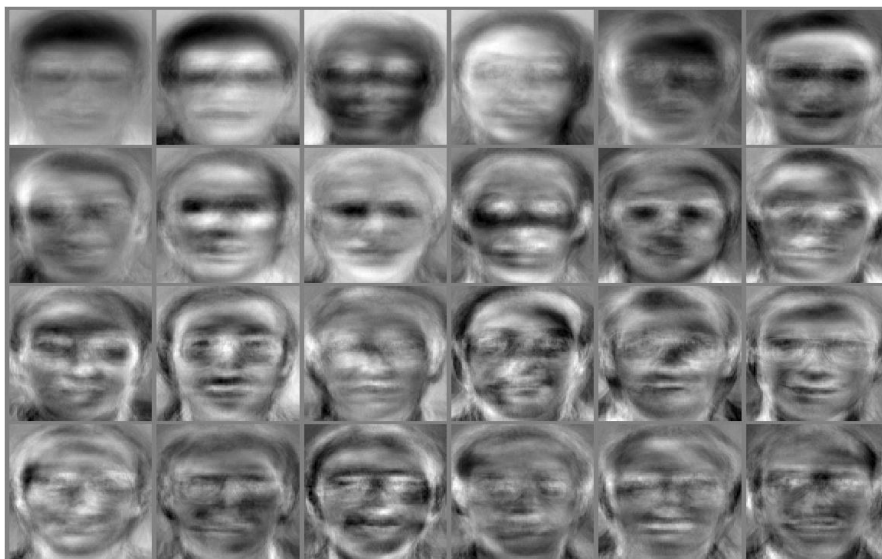
Riassumendo quanto detto finora, sfruttando questo approccio, si costruisce un modello molto robusto dell'oggetto a partire da un gruppo di immagini divise in due insiemi: positive e negative. Nelle prime compare l'oggetto da riconoscere, mentre nelle seconde no. Queste informazioni sono note a priori ed utilizzate dall'algoritmo di apprendimento.

Quindi, generalizzando tutto il procedimento prima sull'intera immagine e poi sull'insieme di immagini positive, viene costruita una rappresentazione XML dell'oggetto da cercare. Questo file altro non è che un descrittore delle sole *features* importanti ed è ciò che andremo ad utilizzare per effettuare la rilevazione di volti.

2. Gli approcci scelti

2.2 Il metodo dell'analisi delle componenti principali

PCA (*Principal Component Analysis*) costituisce una delle prime strategie sviluppate nell'ambito del riconoscimento dei volti. I primi ad introdurla furono, infatti, M. Kirby e L. Sirovich nel 1988. Tale metodo fa utilizzo di un *training set* costituito da un'insieme di immagini delle stesse dimensioni e "normalizzate" in maniera da evidenziare caratteristiche del volto, quali occhi e bocca. Questa strategia prevede successivamente una riduzione della dimensionalità dei dati a disposizione, tramite una proiezione in un sottospazio, nel quale vengono messe in risalto le caratteristiche salienti di un volto. Tale riduzione dimensionale, infatti, permette di escludere l'informazione che non viene considerata rilevante e precisamente decompone la struttura di un volto in una combinazione di componenti ortogonali, scorrelati tra loro, dette *eigenface*. Ogni immagine di volto può successivamente essere rappresentata come una somma pesata (vettore delle feature o *eigenvectors*) di queste *eigenfaces*, raccolte in un vettore monodimensionale. Il confronto di un'immagine di volto con le altre presenti nel *training set* viene effettuato semplicemente valutando la distanza tra questi vettori di caratteristiche locali.



Esempi di eigenfaces

2. Gli approcci scelti

Si può quindi dire che PCA è un utile metodo statistico che permette di trovare *patterns* all'interno di insiemi di dati di grandi dimensioni, classificando tali dati in base al loro grado di similarità.

2.3 Il metodo EigenFaces

Dal modello PCA è nato poi il metodo *Eigenfaces*, conosciuto anche come “metodo delle autofacce”, sviluppato nel 1991 da Matthew Turk e da Alex Pentland^[3]. In questa ‘nuova’ tecnica le osservazioni sono viste come set di immagini di facce prese sotto le stesse condizioni di luce, normalizzate per allineare occhi e bocca, e campionate alla stessa risoluzione.

Andando quindi ad effettuare il processo matematico delle PCA su un ampio insieme di immagini raffiguranti diversi volti umani, generiamo un insieme di *eigenfaces*.

Andiamo più nello specifico...

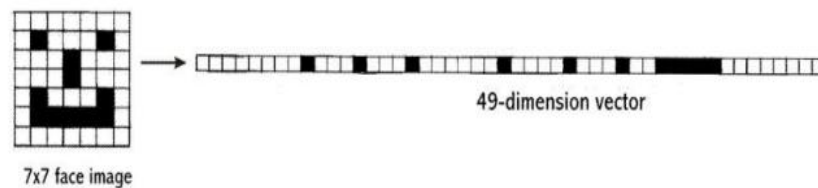
Come detto prima le *eigenfaces* sono gli autovettori usati come base del nuovo spazio e nelle cui coordinate sono descritte tutte le facce. Esse possono essere considerate come un insieme di “*standardized face ingredients*”, derivate dall'analisi statistica di molte immagini di volti. Ogni volto umano può essere considerato come una combinazione di questi volti standard. Sorprendentemente, non servono molte *eigenfaces* combinate insieme per raggiungere una buona approssimazione della maggior parte dei volti (sono sufficienti tra le 100 e le 150 *eigenfaces*). Inoltre, poichè il volto di una persona non viene registrato da una fotografia digitale, ma piuttosto come un semplice elenco di valori (un valore per ogni *eigenface* nel *database* utilizzato), molto meno spazio viene dato per il volto di ogni persona. Le *eigenfaces* che vengono create appariranno come zone chiare e scure disposte in un pattern specifico.

Un fattore molto importante in questo tipo di approccio è che le immagini che costituiscono il *training set* devono essere prese sotto le stesse condizioni di illuminazione. Esse devono anche essere tutte ricampionate per avere la stessa risoluzione dei pixel. Ogni immagine è trattata come un vettore, semplicemente concatenando le righe di pixel dell'immagine originale, risultando in una sola riga con elementi $r \times c$.

2. Gli approcci scelti

2.3.1 Il volto visto come un vettore

L'immagine di un volto sostanzialmente può essere trasformata in un vettore. Se chiamiamo w la larghezza dell'immagine e h la sua altezza, entrambe in pixel, il numero delle componenti del vettore che si vuole ottenere è dato dal prodotto $w * h$, dove ogni pixel dell'immagine iniziale corrisponde ad una componente del vettore. La costruzione di tale vettore quindi può essere effettuata tramite una semplice concatenazione delle righe della matrice dell'immagine iniziale:



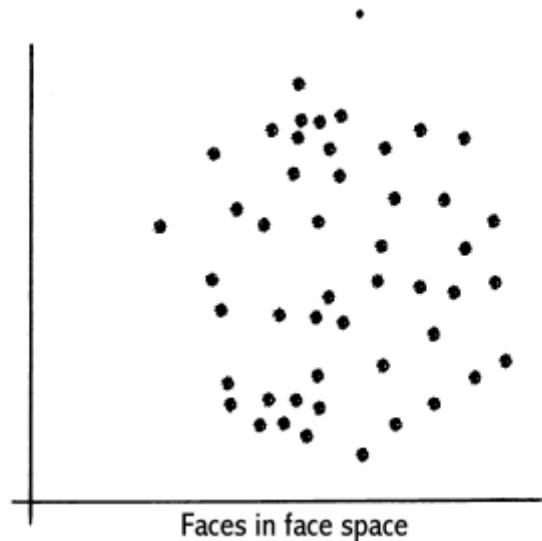
Trasformazione di un'immagine in un vettore

2.3.2 Lo spazio delle immagini

Il vettore descritto appartiene ad uno spazio vettoriale. Tale spazio è chiamato “spazio delle immagini” ed è lo spazio di tutte le immagini la cui dimensione è di $w * h$ pixels.

Tutti i volti presenti in ogni immagine si “assomigliano” fra loro; infatti ognuno è caratterizzato dalla presenza di due occhi, una bocca, un naso ecc, ovvero tutte caratteristiche localizzate nella stessa area. Ciò significa che tutti i punti rappresentanti i volti non si spargono in maniera omogenea nello spazio, bensì tendono a localizzarsi in un ristretto *cluster* nello spazio immagine, esattamente come mostrato dalla seguente figura:

2. Gli approcci scelti



Spazio dei volti e spazio delle immagini

Ovviamente non tutti i pixels che compongono l'immagine di un volto possono essere considerati "rilevanti" ed in particolare ognuno di questi dipende strettamente dal suo immediato vicino.

Tale considerazione ci permette di capire che la dimensione dello spazio dei volti sarà sicuramente diversa rispetto allo spazio delle immagini iniziali, ed in particolare la dimensione di tale spazio sarà sicuramente di una dimensione minore.

Come detto in precedenza l'obiettivo dell'approccio al riconoscimento facciale tramite PCA è proprio quello di cercare di ridurre la dimensionalità dei dati iniziali, proiettandoli in un sottospazio. Quest'ultimo è definito da una nuova base ed è in grado di descrivere al meglio il "modello" dell'insieme dei dati iniziali, insieme che in questo caso corrisponde a quello dei volti presenti nel *training set* iniziale. I vettori che compongono la base di questo spazio sono detti componenti principali, devono essere correlati fra loro e massimizzare la varianza stimata per le variabili originali.

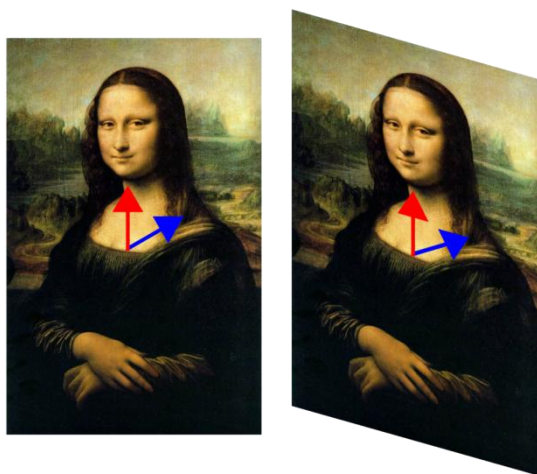
2. Gli approcci scelti

2.3.3 Autovettori nell'algebra lineare

Ogni punto dello spazio può essere descritto tramite un vettore. Esempi di spazi vettoriali sono il piano cartesiano e lo spazio euclideo. In uno spazio vettoriale è possibile effettuare trasformazioni lineari sui vettori che lo costituiscono. Esempi di trasformazioni lineari sono le rotazione e le riflessioni.

Nell'algebra lineare, un'autovettore di una trasformazione lineare è un vettore diverso da 0 la quale direzione non cambia quando gli viene applicata una trasformazione lineare.

Per capire più a fondo questo concetto andiamo ad osservare la seguente figura:



Trasformazione lineare della Gioconda

In questa trasformazione lineare l'immagine è modificata ma l'asse centrale verticale rimane fisso. Il vettore blu ha cambiato lievemente direzione, mentre quello rosso no. Quindi il vettore rosso è un autovettore della trasformazione e quello blu no.

2. Gli approcci scelti

2.4 Alcune premesse

Le *performance* a tempo di esecuzione di un sistema che usa *eigenfaces* sono molto buone. Su un moderno calcolatore si tratta di un calcolo di pochi secondi ^[4]. Il processo finale di riconoscimento riguarda il calcolo della distanza fra l'*eigenvector* del candidato e quelli dei soggetti del *training set*. Pentland ^[3] afferma che un confronto su database più contenuti (poche centinaia di individui) può raggiungere una velocità pari ai *frame rate* di una videocamera. La precisione di questo metodo alla distorsione facciale, alla posa e alle condizioni di illuminazione è discreta. Sebbene Sirovich e Kirby scoprirono che il loro sistema riconosce candidati che hanno una posa diversa rispetto ai rispettivi campioni, è inevitabile che la qualità del riconoscimento può degradare a seconda del tipo di posa, ma in generale le condizioni di illuminazione restano uno dei problemi principali che affligge questo metodo ^[4].

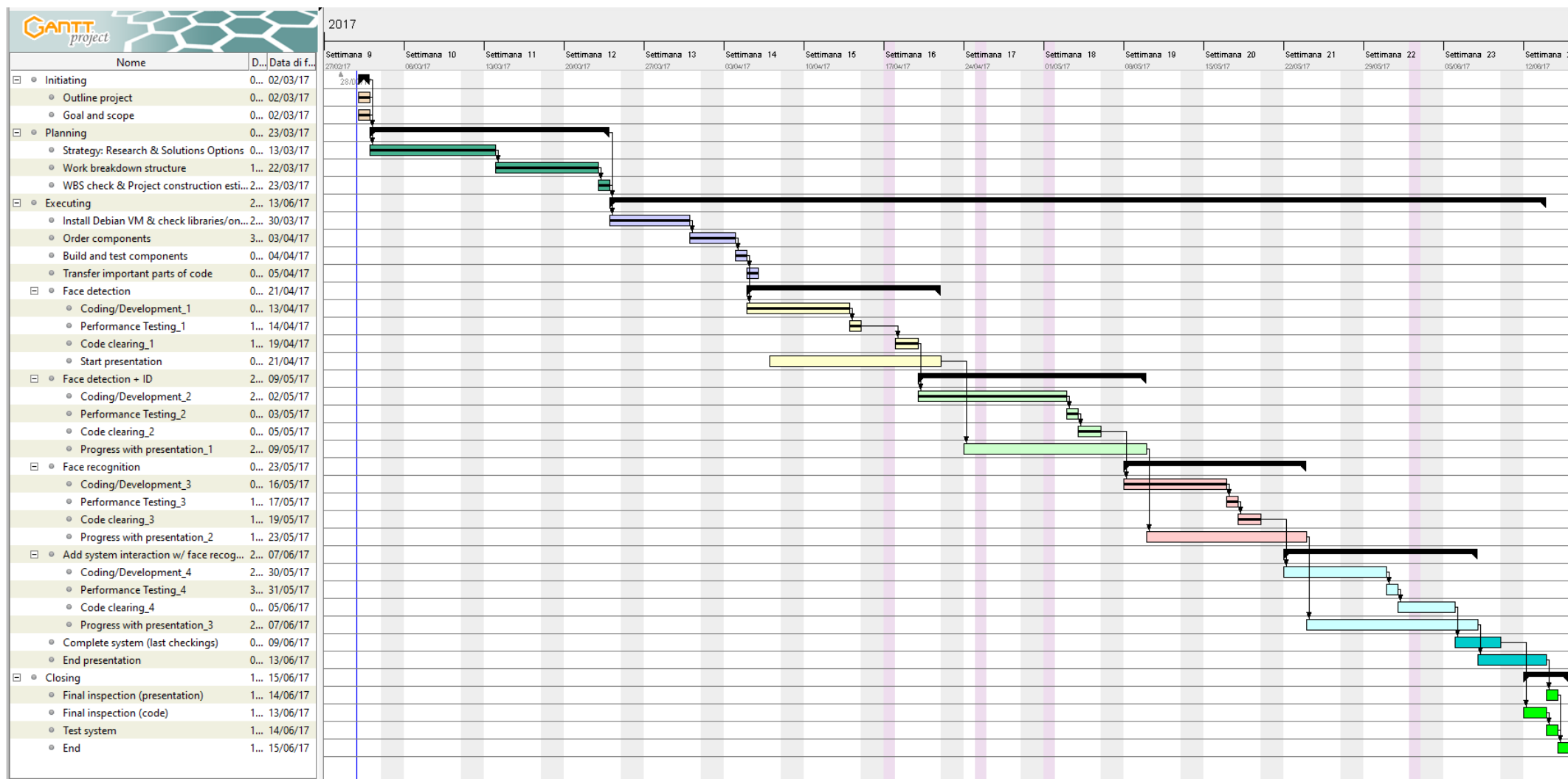
Ora che abbiamo le basi teoriche per comprendere tutto ciò che avverrà all'interno del nostro sistema di riconoscimento facciale, procediamo alla fase di realizzazione vera e propria

Capitolo 3.

Passiamo alla progettazione

3.1 Analisi e pianificazione

Prima di effettivamente cominciare a realizzare il progetto e implementare l'algoritmo, è stato necessario organizzare il tutto in modo tale da gestire il tempo a disposizione. Ciò è stato fatto andando a creare un *diagramma di Gantt* e suddividendo il lavoro in sezioni più piccole e facilmente gestibili come vediamo nella seguente figura:



Gantt Chart

3. Passiamo alla progettazione

Inizialmente, la quantità di codice e il numero di *task* non erano rilevanti. Tuttavia, col passare del tempo, i contenuti del progetto sono andati ad aumentare e ciò ha reso necessario l'utilizzo di un *sistema di versioning*, in particolare *GitHub*, ma anche di un *tool* per il *Project Management*, quale *MeisterTask*, con lo scopo di avere sott'occhio i *task* necessari al completamento del tutto.

Per realizzare tutto il progetto si è scelto di usare la libreria *OpenCV* e il linguaggio di programmazione *Python*.

3.2 Introduzione ad OpenCV

OpenCV è una libreria *open source* destinata alla Visione Artificiale sviluppata inizialmente da *Intel*. E' scritta in C e C++ ed è disponibile su diverse piattaforme. Grazie ad essa siamo stati in grado di effettuare la manipolazione delle immagini e il *Pattern Recognition* andando a identificare i volti nelle immagini.

Le funzioni messe a disposizione da questa libreria, tuttavia, sono circa 500: elaborazione immagini, *tracking* e *object detection*, calibrazione dei dispositivi, estrazione delle *feature* da un'immagine, riconoscimento di volti e così via.

All'interno della *SDK* di OpenCV sono stati trovati inoltre degli utili esempi da poter eseguire sul proprio pc.

3. Passiamo alla progettazione

3.3 Realizzazione del progetto

Dopo aver settato correttamente il nostro Raspberry Pi e la camera, ci siamo trovati pronti per effettivamente sviluppare il nostro sistema di riconoscimento facciale dal punto di vista *software*. Anche dopo un'analisi intensa e pur avendo trovato il metodo più veloce per arrivare al completamento del progetto, si sono voluti fare lo stesso piccoli passi alla volta in modo tale da comprendere a pieno ciò che si stava sviluppando. Proprio per questo motivo, il primo step è stato quello di creare uno *script* in grado di effettuare la rilevazione facciale.

3.3.1 Rilevazione di un volto in un'immagine

Al fine di rilevare i volti in un'immagine, abbiamo usato la classe *CascadeClassifier* messa a disposizione dalla potentissima libreria descritta in precedenza.

Essa permette di rilevare oggetti, tra cui ovviamente anche i volti, all'interno di un'immagine o di un video. Più particolarmente, abbiamo usato le funzioni *load* e *detectMultiScale* per portare a termine l'operazione di *face detection*.

La prima è servita principalmente a caricare il file classificatore. Essa accetta due tipi di file; XML e YAML. Date le conoscenze acquisite durante l'anno scolastico, si è scelto di usare il primo tipo di file. Come spiegato precedentemente nella sezione 2.1 e 2.1.1, siamo andati ad usare dei classificatori pre-caricati trovabili nella *repository* GitHub di OpenCV. Questi file contengono le caratteristiche principali di un oggetto. Andando più nello specifico, abbiamo usato il file '*haarcascade_frontalface_alt.xml*' in modo tale da rilevare i volti, in posa approssimativamente frontale, presenti in un'immagine.

Sfruttando poi la funzione *detectMultiScale*, abbiamo ottenuto la posizione di ogni volto presente nell'immagine. Come già accennato, questi classificatori ritornano le coordinate delle regioni rettangolari che racchiudono un volto.

3. Passiamo alla progettazione

Le due funzione sono strutturate nel seguente modo:

```
# Caricamento classificatore
haar_faces = cv2.CascadeClassifier('haarcascade_frontalface_alt.xml')

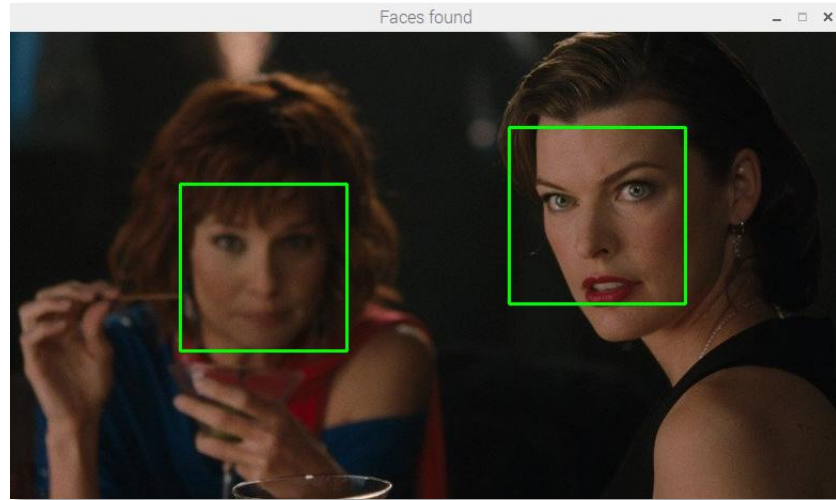
# Rilevamento volti
faces = haar_faces.detectMultiScale (
    image,
    scaleFactor= 1.2,
    minNeighbors= 4,
    minSize= (30, 30)
)
```

Andiamo ora ad analizzare i parametri necessari alla funzione principale:

- *image*: questo parametro indica l'immagine data in input.
- *scaleFactor*: specifica quanto la grandezza dell'immagine è ridotta a ogni suo ridimensionamento. Lo *scaleFactor* è usato per creare una piramide scalare. In poche parole, il nostro modello ha una grandezza definita durante il suo allenamento, che è visibile nel file XML. Questo significa che questa grandezza del volto è rilevata nell'immagine se presente. Tuttavia, andando a ridimensionare l'immagine di input, si può ridimensionare un volto più largo in uno più piccolo, rendendolo rilevabile dall'algoritmo.
- *minNeighbors*: indica quanti vicini ogni rettangolo candidato deve avere per essere memorizzato.
- *minSize*: grandezza minima possibile del volto. Oggetti più piccoli di questo valore vengono ignorati.

Dopo aver impostato queste due funzioni, siamo andati ad eseguire lo script ottenendo così il risultato desiderato.

3. Passiamo alla progettazione



Volti trovati dando in input un immagine allo script appena descritto

Bisogna far tuttavia notare che i valori della funzione *detectMultiScale* devono essere modificati in base alla situazione nella quale ci si trova in quanto si potrebbero riscontrare dei falsi positivi.

Una volta fatto ciò, si è scelto di effettuare la rilevazione facciale in un video andando ad implementare uno *script* che permettesse una sorta di *face tracking*.

3.3.2 Face tracking in un video

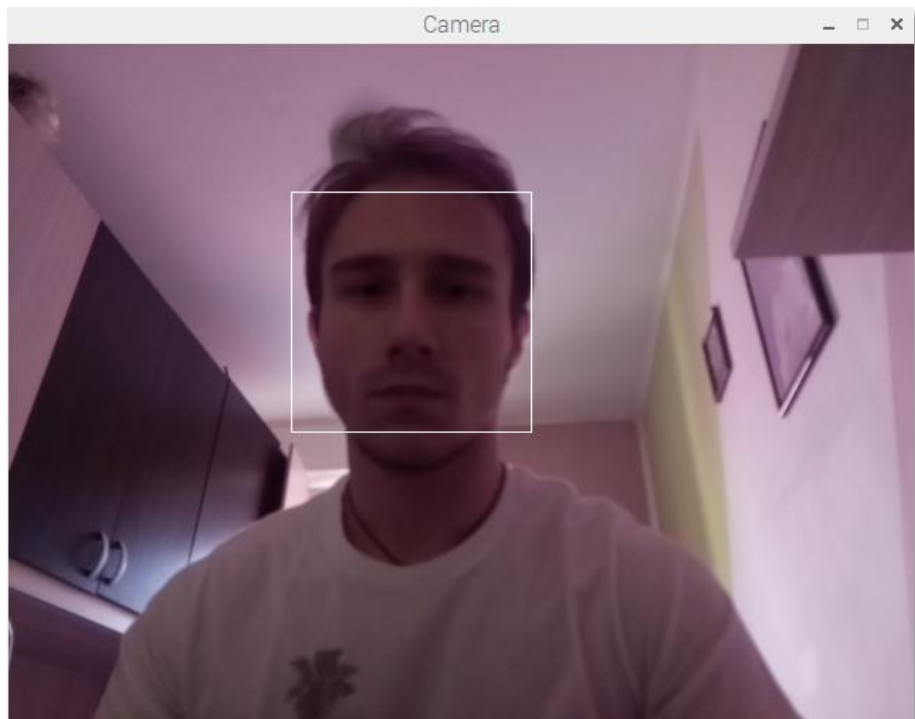
La differenza principale dallo *script* precedente è che, oltre ad aver dovuto ovviamente cambiare la modalità di input passando dall'avere un'immagine singola ad uno stream continuo e poter quindi accedere ad ogni frame per effettuare l'*image processing*, si è dovuto anche cambiare il tipo di immagine data in input alla funzione *detectMultiScale*. Più precisamente, siamo andati a convertire l'immagine originale in una scala di grigi in modo tale da rendere la rilevazione dei volti più veloce.

Infatti, per quanto riguarda gli algoritmi che hanno a che fare con l'*image processing*, essi lavorano più velocemente quando utilizzano immagini

3. Passiamo alla progettazione

senza colore. Per effettuare ciò abbiamo nuovamente sfruttato le funzionalità messe a disposizione dalla libreria OpenCV.

Il risultato dello *script* sviluppato è il seguente:



Esempio di face tracking in un video

Una volta ottenute le basi necessarie per comprendere totalmente il concetto di *face detection*, siamo andati a sviluppare la parte del sistema che si occupa del riconoscimento facciale vero e proprio.

3.3.3 Primo sviluppo dell'algoritmo di riconoscimento

Come prima cosa abbiamo dovuto ovviamente utilizzare un *training set*. E' stato scelto 'The ORL Database of Faces' appartenente ai laboratori AT&T di Cambridge. Questo set contiene 10 immagini differenti per 40 soggetti diversi. Le immagini sono state prese in tempi diversi, cambiando i livelli di luminosità e l'espressione facciale. I file sono in formato *PGM*, hanno tutti

3. Passiamo alla progettazione

256 livelli di grigio per pixel e si presentano tutti con la stessa dimensione: 92x112.

Come detto nella sezione 2.3, il fatto che tutte le immagini abbiano la stessa grandezza è una caratteristica importante e necessaria affinché l'algoritmo funzioni correttamente.

In questo primo script riguardante il riconoscimento facciale, ci siamo concentrati solamente sullo sviluppo dell'algoritmo, evitando di integrare una sorta di interfaccia video. L'obiettivo, oltre ad implementare correttamente l'algoritmo, è stato quello di ricavare le statistiche riguardanti l'identificazione di un individuo dopo aver confrontato il suo volto con il nostro *training set*. Per rendere il tutto più emozionante, è stato creato anche un secondo *script* che ci aiutasse a normalizzare delle foto personali scattate sul momento, per poi aggiungerle al nostro *dataset*. Questo per far sì che, utilizzando l'algoritmo, ci venisse ritornata la nostra identità.

Per sviluppare il tutto abbiamo utilizzato la classe *FaceRecognizer*. Quest'ultima ci ha permesso di gestire pienamente il nostro modello per il riconoscimento facciale. Dopo aver inizializzato l'oggetto, siamo andati ad allenare il nostro algoritmo.

L'allenamento richiede due parametri fondamentali; un array contenente tutte le immagini che si vogliono utilizzare e un altro con le *label* degli utenti.

Una volta conclusosi il *training*, abbiamo salvato il tutto in un file XML. Quest'ultimo è stato poi caricato e utilizzato per effettuare la *prediction* della nostra identità. Ogni volta che si vuole identificare un soggetto, viene eseguito il seguente codice (riportato nella sua versione essenziale).

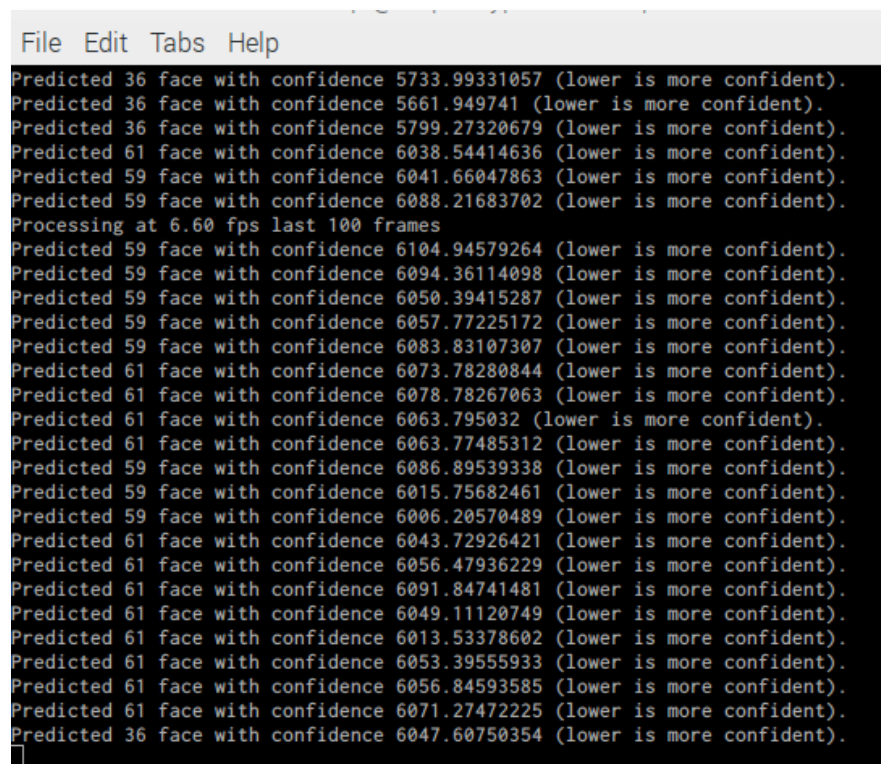
```
# Initialize model
model = cv2.face.createEigenFaceRecognizer()
# Training
model.train(param1, param2)
# Save trained model
model.save('training.xml')
# Load trained model
model.load('training.xml')
# Run prediction
label, confidence = model.predict(image)
```

3. Passiamo alla progettazione

Come possiamo vedere, la funzione principale per il riconoscimento facciale prende in input un'immagine e restituisce due valori. Il primo, la label, è l'id dell'utente che viene identificato. Il secondo, invece, è il livello di confidenza con il quale l'algoritmo ha riconosciuto l'utente. Quest'ultimo è un numero che, più è vicino allo 0, più indica che l'utente è stato autenticato con certezza.

Per esempio, nel caso in cui il livello di confidenza fosse 2000, ciò significherebbe che l'utente è stato identificato con totale sicurezza in quanto, date le condizioni amatoriali del progetto e la risoluzione non ottimale della camera, è quasi impossibile che il livello di confidenza scenda sotto questo valore.

Dopo aver sviluppato il tutto e aver unito lo script con quello descritto nella sezione 3.2.2 il risultato è il seguente:

A screenshot of a terminal window with a menu bar at the top containing 'File', 'Edit', 'Tabs', and 'Help'. The terminal displays a series of text lines representing face recognition results. Each line starts with 'Predicted' followed by a number (36 or 61), the text 'face with confidence', a long decimal number, and a note in parentheses: '(lower is more confident)'. The numbers 36 and 61 likely represent user IDs. The decimal numbers are large, with some having a decimal point and others not. The text is displayed in a monospaced font on a dark background.

```
File Edit Tabs Help
Predicted 36 face with confidence 5733.99331057 (lower is more confident).
Predicted 36 face with confidence 5661.949741 (lower is more confident).
Predicted 36 face with confidence 5799.27320679 (lower is more confident).
Predicted 61 face with confidence 6038.54414636 (lower is more confident).
Predicted 59 face with confidence 6041.66047863 (lower is more confident).
Predicted 59 face with confidence 6088.21683702 (lower is more confident).
Processing at 6.60 fps last 100 frames
Predicted 59 face with confidence 6104.94579264 (lower is more confident).
Predicted 59 face with confidence 6094.36114098 (lower is more confident).
Predicted 59 face with confidence 6050.39415287 (lower is more confident).
Predicted 59 face with confidence 6057.77225172 (lower is more confident).
Predicted 59 face with confidence 6083.83107307 (lower is more confident).
Predicted 61 face with confidence 6073.78280844 (lower is more confident).
Predicted 61 face with confidence 6078.78267063 (lower is more confident).
Predicted 61 face with confidence 6063.795032 (lower is more confident).
Predicted 61 face with confidence 6063.77485312 (lower is more confident).
Predicted 59 face with confidence 6086.89539338 (lower is more confident).
Predicted 59 face with confidence 6015.75682461 (lower is more confident).
Predicted 59 face with confidence 6006.20570489 (lower is more confident).
Predicted 61 face with confidence 6043.72926421 (lower is more confident).
Predicted 61 face with confidence 6056.47936229 (lower is more confident).
Predicted 61 face with confidence 6091.84741481 (lower is more confident).
Predicted 61 face with confidence 6049.11120749 (lower is more confident).
Predicted 61 face with confidence 6013.53378602 (lower is more confident).
Predicted 61 face with confidence 6053.39555933 (lower is more confident).
Predicted 61 face with confidence 6056.84593585 (lower is more confident).
Predicted 61 face with confidence 6071.27472225 (lower is more confident).
Predicted 36 face with confidence 6047.60750354 (lower is more confident).
```

Test d'identificazione in un video (senza interfaccia video per la camera)

3. Passiamo alla progettazione

3.3.4 Creazione del database

Fino a questo momento si è sempre lavorato utilizzando immagini salvate solamente sul *filesystem*. Tuttavia, date le conoscenze sviluppate durante l'anno scolastico, si è deciso di implementare un *database*.

Si è scelto di farlo utilizzando *SQLite* in quanto le sue performance sono ottime ed in parecchie occasioni risultano addirittura superiori a sistemi come MySQL. Le eccezionali performace di *SQLite* sono dovute alla semplicità e leggerezza: l'intera libreria occupa appena 500 KB e, cosa più importante, il database viene direttamente caricato nell'applicazione, non coinvolgendo quindi alcun tipo di connessione di rete che causerebbe dell'*overhead*.

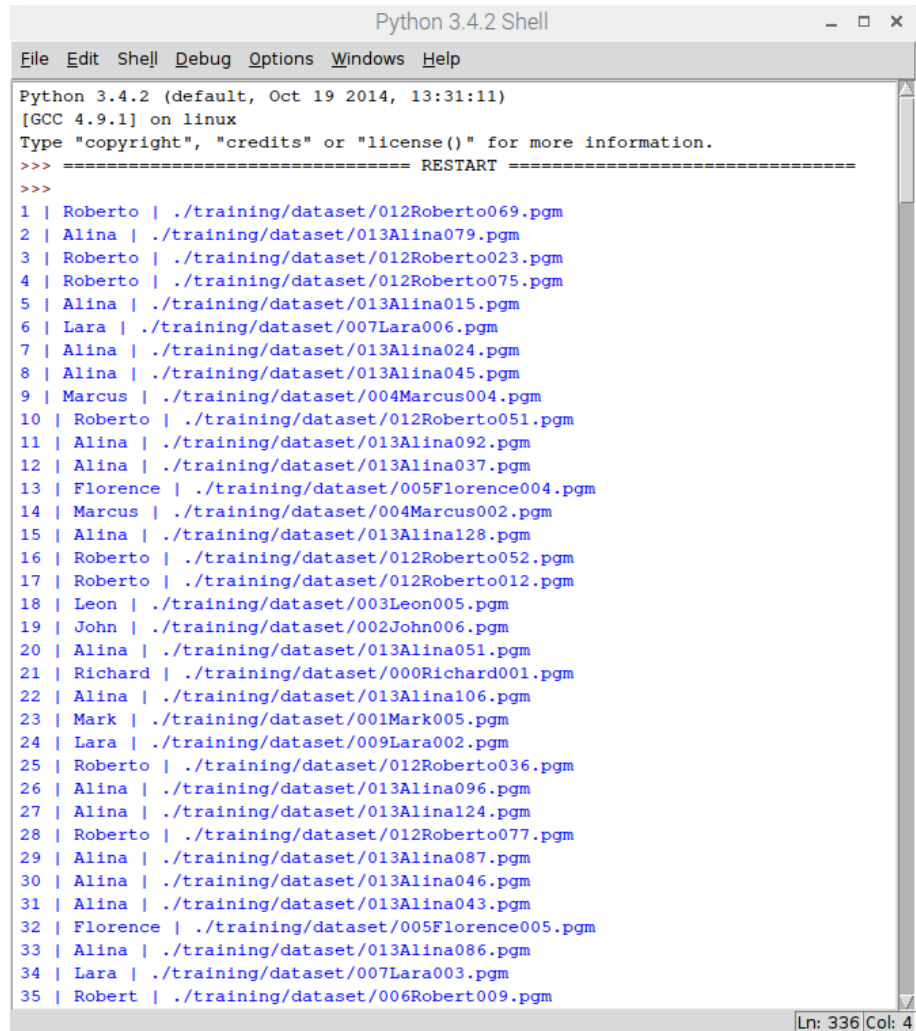
La soluzione migliore è stata quella di salvare nel database solamente i *path* alle immagini mentre le immagini vere e proprio sarebbero state lasciate sul *filesystem*.

Si è deciso di lavorare in questo modo per non appesantire troppo il *database* ma anche ovviamente per facilitare le operazioni che avremmo effettuato sfruttando le immagini.

La struttura del database non è per nulla complessa in quanto non necessitiamo di nulla se non una tabella per contenere i nomi delle persone e i *path* alle loro immagini. Abbiamo quindi creato una tabella *users* andando ad eseguire la seguente *query*:

```
'CREATE TABLE IF NOT EXISTS users (  
  
    id INTEGER PRIMARY KEY AUTOINCREMENT,  
    name VARCHAR(60) NOT NULL,  
    filepath TEXT NOT NULL  
  
) '
```

3. Passiamo alla progettazione



```
Python 3.4.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.4.2 (default, Oct 19 2014, 13:31:11)
[GCC 4.9.1] on linux
Type "copyright", "credits" or "license()" for more information.
>>> ===== RESTART =====
>>>
1 | Roberto | ./training/dataset/012Roberto069.pgm
2 | Alina | ./training/dataset/013Alina079.pgm
3 | Roberto | ./training/dataset/012Roberto023.pgm
4 | Roberto | ./training/dataset/012Roberto075.pgm
5 | Alina | ./training/dataset/013Alina015.pgm
6 | Lara | ./training/dataset/007Lara006.pgm
7 | Alina | ./training/dataset/013Alina024.pgm
8 | Alina | ./training/dataset/013Alina045.pgm
9 | Marcus | ./training/dataset/004Marcus004.pgm
10 | Roberto | ./training/dataset/012Roberto051.pgm
11 | Alina | ./training/dataset/013Alina092.pgm
12 | Alina | ./training/dataset/013Alina037.pgm
13 | Florence | ./training/dataset/005Florence004.pgm
14 | Marcus | ./training/dataset/004Marcus002.pgm
15 | Alina | ./training/dataset/013Alina128.pgm
16 | Roberto | ./training/dataset/012Roberto052.pgm
17 | Roberto | ./training/dataset/012Roberto012.pgm
18 | Leon | ./training/dataset/003Leon005.pgm
19 | John | ./training/dataset/002John006.pgm
20 | Alina | ./training/dataset/013Alina051.pgm
21 | Richard | ./training/dataset/000Richard001.pgm
22 | Alina | ./training/dataset/013Alina106.pgm
23 | Mark | ./training/dataset/001Mark005.pgm
24 | Lara | ./training/dataset/009Lara002.pgm
25 | Roberto | ./training/dataset/012Roberto036.pgm
26 | Alina | ./training/dataset/013Alina096.pgm
27 | Alina | ./training/dataset/013Alina124.pgm
28 | Roberto | ./training/dataset/012Roberto077.pgm
29 | Alina | ./training/dataset/013Alina087.pgm
30 | Alina | ./training/dataset/013Alina046.pgm
31 | Alina | ./training/dataset/013Alina043.pgm
32 | Florence | ./training/dataset/005Florence005.pgm
33 | Alina | ./training/dataset/013Alina086.pgm
34 | Lara | ./training/dataset/007Lara003.pgm
35 | Robert | ./training/dataset/006Robert009.pgm
Ln: 336 Col: 4
```

Esempio di dati presenti nel database

Successivamente si è deciso di continuare a sviluppare lo script principale riguardante il riconoscimento facciale.

3.3.5 Andiamo avanti con l'algoritmo di riconoscimento facciale

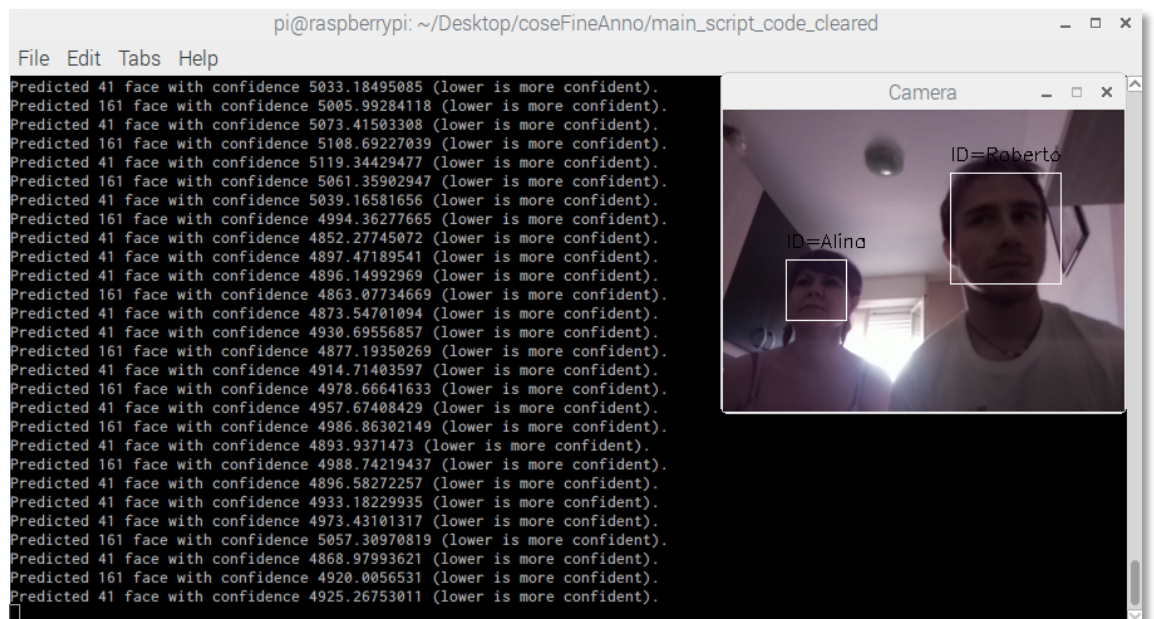
A questo punto è stata introdotta un'interfaccia video per l'utente e si è sviluppato lo script in modo tale che avvenisse un riconoscimento facciale continuo. Durante lo sviluppo di questa fase, ci si è ritrovati davanti ad un grande problema; il Raspberry Pi non era abbastanza potente da mantenere un numero stabile di *fps* nel video e ciò rendeva l'interfaccia video inutile.

3. Passiamo alla progettazione

Per ovviare a ciò si è deciso quindi di unire la classe *VideoCapture* di OpenCV con la classe *picam* della camera pi e utilizzare anche i *thread* in modo tale che l'acquisizione del video e l'*image processing* necessario per far funzionare l'algoritmo venissero eseguiti in modo parallelo. Ci sono stati dei miglioramenti nella quantità di *fps* ma ovviamente le capacità computazionali del Raspberry Pi sono limitate e non si è potuto quindi arrivare ad avere un video totalmente fluido.

Come fatto in precedenza per il *tracking* dei volti nei video, siamo andati a lavorare con ogni singolo frame dello *stream* al fine di identificare un individuo. Ciò è stato possibile sfruttando la *label* che ci viene ritornata dall'algoritmo.

Più particolarmente, abbiamo fatto in modo che ci fossero 3 modalità d'identificazione; utente identificato completamente, utente identificato parzialmente e utente non identificato. Nel primo caso sopra la testa verrà scritto il suo nome preceduto dal simbolo '=', nel secondo caso l'uguale verrà sostituito da un '+-' che indica la quasi completa identificazione dell'utente mentre nel terzo ed ultimo caso non verrà scritto nulla di particolare. In tutti e tre i casi siamo andati a disegnare sul video in output un rettangolo intorno ai volti.



Testing dello script finale con due soggetti identificati correttamente

3. Passiamo alla progettazione

Pur avendo concluso il progetto, si è voluto andare ad analizzare due funzioni messe a disposizione dal modello precedentemente creato. Esse aggiungono informazioni necessarie alla comprensione di come l'algoritmo effettui il riconoscimento facciale.

3.3.5.1 Analisi delle *eigenfaces* del nostro training set

La prima che andremo ad analizzare è la seguente:

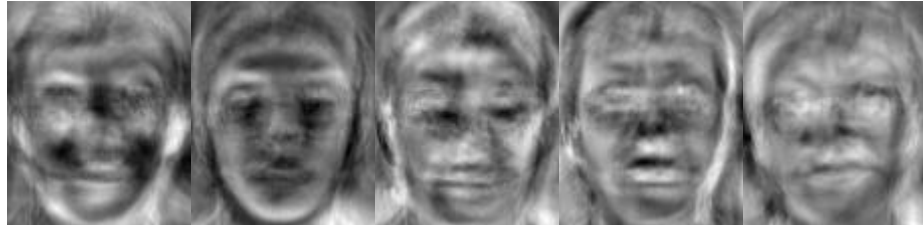
```
model.getEigenVectors()
```

Come abbiamo detto all'inizio, l'intero algoritmo che abbiamo usato si basa sull'uso delle *eigenfaces*. Tramite questa funzione possiamo ricavare le immagini che verranno usate dall'algoritmo per identificare una persona.

Più particolarmente, otterremo gli *eigenvectors* corrispondenti alle matrici di covarianza che formano le *eigenfaces*. Se l'immagine data in input all'algoritmo è simile a una di queste immagini, esso riuscirà ad identificare correttamente la persona.



3. Passiamo alla progettazione



Estrazione delle 15 migliori eigenfaces usando il nostro training set

3.3.5.2 Analisi della mean face del nostro training set

La seconda funzione, invece, ci permette di ottenere la media di tutte le nostre *eigenface*:

```
model.getMean()
```



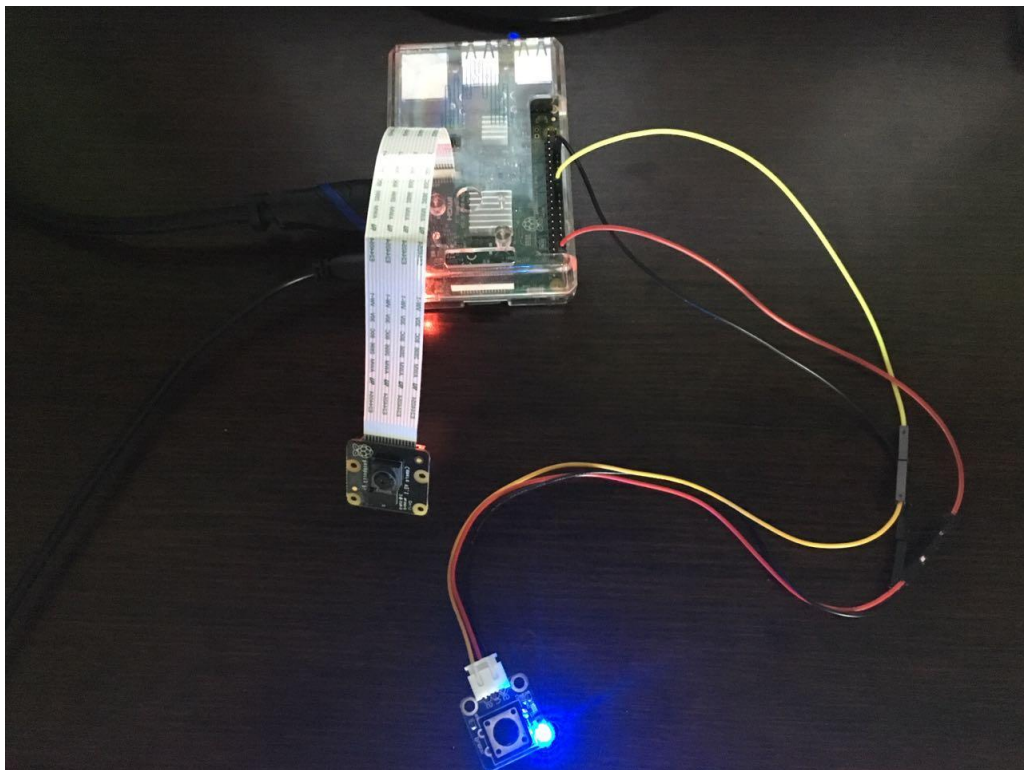
Mean face

Quest'immagine rappresenta il centro di gravità di tutte le eigenfaces e aiuta l'algoritmo a investigare la distribuzione dei volti nello spazio delle immagini.

Capitolo 4.

Risultati del progetto

A questo punto possiamo dire che il progetto è stato portato a termine nonostante le difficoltà incontrate nella realizzazione del sistema e la scarsa disponibilità, a parer mio, di documentazione riguardanti la libreria OpenCV. C'è da dire, infatti, che alcune parti di questa potentissima libreria non sono neanche documentate. Nonostante ciò, tutti gli obiettivi fissati all'inizio sono stati raggiunti andando a realizzare un sistema di riconoscimento facciale.



Vista del progetto (esteticamente non completo)

4.1 Efficienza del sistema

Come detto in precedenza nella sezione 1.2, ci sono ovviamente alcune problematiche che compromettono l'efficienza del sistema:

4. Risultati del progetto

- Risoluzione camera: come si può capire, la risoluzione della camera di soli 8 MegaPixel riduce l'efficienza del sistema.
- Capacità computazionale del Raspberry Pi: il fatto di dover prendere ogni immagine da un video e andare ad effettuare l'*image processing* su ogni *frame*, ha il suo impatto sull'efficienza del sistema. Infatti, la qualità delle immagini ovviamente è più scarsa rispetto allo scatto di foto diretto.
- Mancata professionalità nello scatto di foto: la mancata o l'eccessiva illuminazione durante lo scatto di foto da usare per l'allenamento del modello riducono drasticamente l'efficienza del sistema.

4.2 Statistiche e sviluppi futuri

Nonostante tutte le problematiche, il sistema realizzato ha soddisfatto le aspettative preposte andando ad effettuare un riconoscimento facciale corretto l'85% delle volte.

Cosa si può migliorare?

- Usare delle luci flash programmabili andando a posizionarle vicino alla camera in modo tale che, in mancanza di illuminazione, si possano accendere e aiutino a mantenere una certa efficienza dell'algoritmo.
- Implementare un engine *TTS/STT* (*Text To Speech/Speech To Text*) che permetta l'interazione con il.
- Ottimizzare ulteriormente le immagini date in input all'algoritmo andando ad effettuare un'allineamento del volto in base all'asse di simmetria degli occhi.
- Ampliare il *database* andando ad utilizzare un server web in modo tale che esso possa essere modificato anche in mancanza del file che contiene tutti i dati.
- Creare un sito web, principalmente per l'admin del sistema, che permetta la gestione degli utenti registrati.

BIBLIOGRAFIA

- [1] P. Viola - M. Jones, "Rapid object detection using boosted cascade of simple features" 2001.
- [2] Intel Open Source Computer Vision Library.
<http://www.intel.com/technology/computing/opencv/>.
- [3] A. Pentland - M. Turk, "Eigenfaces for Recognition," *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71-86, 1991.
- [4] Sani Romdhani, "Face Recognition using PCA Msc Thesis," University of Glasgow.