Università degli studi di Padova

Anno accademico 2014 - 2015

Dipartimento di Ingegneria dell'Informazione

Tesi di laurea magistrale in Ingegneria dell'Automazione

Tecniche invarianti alla posa per riconoscimento facciale da dati RGB-D

Relatore: Prof. Emanuele Menegatti
Correlatore: Ing. Matteo Munaro

Laureanda: Giorgia Pitteri

Alla mia famiglia
che mi ha sempre sostenuto.

A Nicola
la mia forza, il mio sorriso.

A tutte le persone
che mi sono state vicino in questi cinque anni.

Non c'è piacere nel successo,
se non lo dividi con qualcuno
I Fantastici 4

Abstract

Questo lavoro di tesi propone un algoritmo invariante alla posa per il riconoscimento facciale da dati RGB-D acquisiti con una *Microsoft Kinect* di seconda generazione. L'approccio seguito consiste in un processing del dataset per la correzione delle pose dei soggetti i quali vengono ruotati frontalmente rispetto il frame della camera. Sfruttando i *Random Forest Classifiers* vengono rilevati i *keypoint* all'interno delle facce 2D e mappati nel 3D. Attorno a questi punti sono estratti dei descrittori ottenuti dalla concatenazione di descrittori locali sia 2D che 3D, i quali vengono successivamente classificati tramite Support Vector Machines (SVM) allenate in due differenti modi. Alcuni test sono stati eseguiti per testare l'algoritmo e confrontarlo con lo stato dell'arte calcolando come indici principali di riconoscimento il *rank-1* e il *false positive rate*.

Indice

Capitolo 1

Introduzione

Un'applicazione importante per la video sorveglianza e l'interazione uomo-robot è quella di riconoscere le persone dalla loro faccia.
Una definizione generale del problema di *face recognition* può essere formulata nel seguente modo:
Data una sequenza video o una immagine, identificare o verificare una o più persone nella scena basandosi su un database di facce.
La soluzione al problema comprende la segmentazione delle facce (*face detection*) dalle immagini con background diversi, l'estrazione delle *feature* dalle regioni della faccia ed infine il riconoscimento o la verifica della faccia. A volte è necessario pre-processare l'immagine prima dell'estrazione delle *feature* e, una volta ottenute queste ultime, ridurne la dimensione attraverso la *Principal Component Analysis*. Il flowchart di un generico sistema per il riconoscimento facciale è mostrato in Figura 1.1.
Nei problemi di identificazione, come quello affrontato in questa tesi, l'input al sistema è la faccia di un soggetto. Il sistema deve dire se tale soggetto è conosciuto e in tal caso quale è la sua identità confrontandolo con quelli presenti in un dataset.
Nei problemi di verifica invece una persona dichiara la sua identità e il sistema deve confermare o respingere tale affermazione.
In letteratura molti lavori hanno risolto il problema di *face recognition* nel dominio spaziale $2D$, ovvero ricevendo come input immagini RGB. Questi metodi che usano solamente l'informazione $2D$ per risolvere il problema del riconoscimento di facce soffrono di alcuni problemi di invarianza rispetto alla qualità dell'illuminazione, tipo di posa ed espressioni facciali.
Infatti, molti algoritmi si basano sull'informazione RGB per il riconoscimento di facce e, come ben noto, il colore che viene percepito non dipende solamente dalla natura della superficie ma anche dalla particolare condizione di luce. La cromaticità è un fattore essenziale nel riconoscimento facciale; l'intensità

del colore di un pixel e alcune relazioni tra pixel dipendono largamente dalle condizioni luminose. Due facce di uno stesso soggetto, ma riprese con condizioni di luce differente, possono mostrare più differenze rispetto a due facce di due soggetti diversi.
Un ulteriore criticità degli algoritmi presenti in letteratura è la varianza alla posa; essi infatti ottengono buone prestazioni esclusivamente su dataset contenenti soggetti ripresi solo frontalmente.
Nonostante sia stata ancora poco studiata, l'informazione $3D$ è chiaramente utile per superare i problemi precedentemente esposti essendo invariante a cambiamenti di illuminazione e posa. Inoltre, con l'avvento di telecamere commerciali RGB-D, come ad esempio la *Microsoft Kinect*, l'informazione $3D$ può essere utilizzata in sempre più applicazioni.

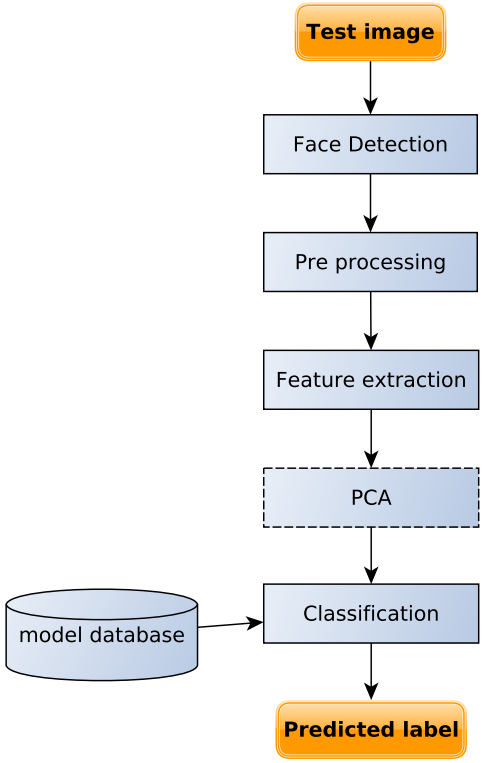


Figura 1.1: Schema generico di un sistema di *face recognition*.

1.1 Stato dell'arte

Tra le varie tecniche di *face recognition* nel dominio spaziale $2D$ presenti in letteratura si può trovare un algoritmo molto popolare [1], basato sulla *Principal Component Analysis* (PCA), chiamato *Eigenfaces*, il quale risolve il problema in modo olistico: l'immagine di una faccia viene considerata come un punto in uno spazio di alta dimensione e una rappresentazione in uno spazio di bassa dimensione viene trovata rendendo la classificazione più semplice. Il sottospazio di dimensione minore è ottenuto tramite la *Principal Component Analysis* la quale identifica gli assi con la massima varianza.
Tuttavia, tale analisi non tiene in considerazione nessuna classe e in questo modo molta informazione discriminativa non viene considerata. Basti pensare ad una situazione in cui la varianza nei dati sia generata da una sorgente esterna, come ad esempio potrebbe essere la luce. Le componenti identificate dalla PCA non contengono questa informazione e i campioni proiettati vengono mescolati assieme rendendo la classificazione quasi impossibile.
L'algoritmo *Fisherfaces* [2] risolve questo problema basandosi sulla *Linear Discriminant Analysis* (LDA), la quale permette una riduzione dello spazio in base a determinate classi. Per determinare la combinazione delle *feature* che meglio separa le classi, la *Linear Discriminant Analysis* massimizza il rapporto tra la varianza inter classe e quella intra classe [2] [3].
Altri metodi sfruttano l'estrazione di *feature* locali per evitare l'elevata dimensione dei dati di input, utilizzando in questo modo solo particolari regioni del volto per produrre descrittori i quali dovrebbero risultare più robusti a parziali occlusioni, illuminazione e a piccole dimensioni di campioni.
Algoritmi usati generalmente per l'estrazione delle *feature* locali sono *Gabor Wavelets* [9] [10], *Discret Cosinus Transform* e *Local Binary Pattern*.
Tuttavia, i metodi che usano solamente l'informazione $2D$ per risolvere il problema del riconoscimento di facce continuano a non essere invarianti a cambiamenti di illuminazione, espressioni facciali o cambiamenti di posa.
L'informazione $3D$, essendo invariante a tali cambiamenti, può essere la chiave per sorpassare i problemi appena esposti.
Per questo motivo negli ultimi anni le ricerche si sono focalizzate sull'inserimento di dati 3D e sull'informazione proveniente da essi nel problema generale di *object recognition*. Un primo traguardo è stato raggiunto da *Fanelli et al.* con lo sviluppo di un algoritmo per la stima della posa del soggetto da dati di profondità [4].

1.2 Obiettivo della tesi

Lo scopo di questa tesi è quello di sviluppare nuove tecniche di *face recognition* basandosi su dati RGB-D ottenuti con una *Microsoft Kinect* di 2° generazione.
Lo scenario applicativo può ad esempio essere un robot mobile il cui compito è assistere una persona anziana. Il robot deve infatti essere in grado di riconoscere la persona da assistere e i suoi parenti e generare un allarme nel caso di riconoscimento di una persona non nota.
La tecnica implementata utilizza entrambi i tipi di informazione a disposizione, 2D e 3D, per tentare di estrarre robusti descrittori sia dalle immagini RGB che da quelle RGB-D.
In particolare, per ottenere un algoritmo robusto alle variazioni di posa, ogni soggetto è stato ruotato frontalmente alla camera grazie alla stima della posa originaria determinata tramite l'algoritmo *open source* sviluppato da *Fanelli et al.* [4]. In queste condizioni, è stato seguito un approccio basato sull'utilizzo di *Random Forest Classifiers* per il rilevamento di punti interessanti all'interno della faccia da dati RGB [7], i quali sono stati successivamente mappati sui dati RGB-D. Tali punti hanno permesso l'estrazione di descrittori locali attraverso il calcolo e la concatenazione di tre tipi di descrittori : i descrittori 2D SURF forniti dalla libreria **OpenCV** che utilizzano l'informazione RGB, i descrittori 3D *Fast Point Feature Histograms* forniti dalla libreria **PCL** ed infine le distanze euclidee tridimensionali tra i punti salienti del volto. La classificazione finale è stata effettutata tramite *Support Vector Machines* (SVM) allenate in due differenti modi, *one against one* e *one against all*, come verrà spiegato in Sezione 5.
Per confrontare il metodo implementato con quelli presenti in letteratura, è stato acquisito un nuovo dataset RGB-D che contiene le immagini RGB-D di 27 persone in differenti pose, a 1 o 2 metri di distanza dal sensore e con diverse condizioni di luce. Il dataset di test contiene anche persone sconosciute non presenti nel dataset di training, come potrebbe essere in caso di controllo degli accessi o per altre applicazioni di interazione uomo-robot.
Per la valutazione dell'algoritmo implementato sono stati scelti due importanti indici di riconoscimento: il *rank-1*, ovvero la percentuale di riconoscimenti corretti, e il *false positive rate*, ovvero il numero di volte in cui una persona sconosciuta viene classificata come una persona nota. È stato ritenuto interessante anche osservare il numero di volte in cui una persona nota viene classificata come sconosciuta (*false rejected*) e il numero totale di match corretti sia di persone note che sconosciute.
La tesi è strutturata come segue:

Capitolo 2 : presenta l'algoritmo per il riconoscimento facciale basato su descrittori *Local Binary Pattern* e la sua implementazione tramite l'utilizzo della libreria **OpenCV**;

Capitolo 3 : presenta le tecniche sviluppate per il *processing* del dataset per ottenere un algoritmo invariante alla posa;

Capitolo 4 : espone l'estrazione dei descrittori per il *matching* tra le facce, in particolare l'algoritmo per il rilevamento dei *keypoint* e i diversi tipi di descrittori calcolati attorno ad essi;

Capitolo 5 : presenta i tipi di classificatori utilizzati;

Capitolo 6 : riporta i test effettutati, spiegando i dataset utilizzati, le metriche di valutazione adottate ed infine le prestazioni ottenute.

Capitolo 7 : conclude il lavoro esposto analizzando i risultati ottenuti, evidenziando gli aspetti sia positivi che negativi ed infine accennando possibili direzioni per eventuali lavori futuri.

Capitolo 2

Descrittori basati su Local Binary Patterns Histograms

Per confrontare l'algoritmo sviluppato e descritto in questa tesi con lo stato dell'arte, è stata inizialmente studiata e testata una delle tecniche più consolidate in letteratura: il riconoscimento facciale attraverso descrittori *Local Binary Pattern* (LBP) appartenente alla famiglia di algoritmi di *Local Feature Analisys* (LFA), la quale raggruppa tutti quegli algoritmi che prevedono l'estrazione di feature locali.
Grazie al proprio potere discriminativo, alla sua semplicità computazionale, oltre che alla robustezza rispetto alle variazioni di illuminazione, l'operatore LBP è diventato un approccio molto usato nel *face recognition* 2D a partire da immagini in scala di grigi. Il vantaggio di avere delle feature locali è sicuramente quello della robustezza alla variabilità di illuminazione e la possibilità di stimare facilmente le rotazioni.
L'algoritmo di Local Binary Pattern si basa su un operatore non parametrico che sintetizza la struttura locale di una immagine. Ad un particolare pixel p_c dell'immagine con coordinate (x_c, y_c), l'operatore LBP è definito come una sequenza binaria ordinata di confronti di intensità di colore fra il pixel p_c e i pixel appartenenti al vicinato considerato. In particolare, se l'intensità del pixel centrale p_c è maggiore o uguale dell'intensità del pixel adiacente allora viene assegnato il valore 1 altrimenti 0. Quindi, per un vicinato di 8 pixel, ad esempio, si avranno 2^8 possibili combinazioni. Il primo operatore LBP presente in letteratura utilizza un vicinato 3×3 fisso, come descritto in Figura 2.1.

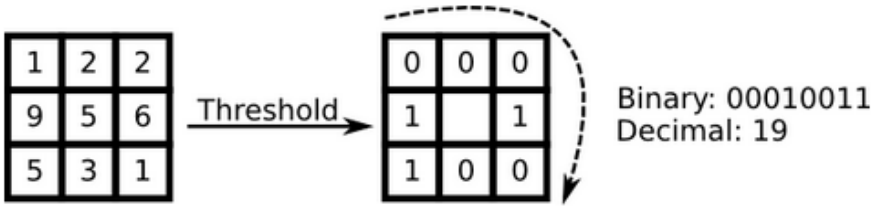


Figura 2.1: *Local Binary Pattern Code* calcolato considerando un vicinato di dimensione fissa 3×3.

La forma decimale della risultante parola da 8-bit (*LBP-Code*) letta in senso orario partendo dal pixel in alto a sinistra può essere così espressa

$$LBP(x_c, y_c) = \sum_{p=0}^{7} 2^p s(i_p - i_c) \tag{2.1}$$

dove i_c corrisponde al valore di grigio del pixel p_c, i_p al valore di grigio dei pixel vicini e la funzione segno $s(x)$ è così definita:

$$s(x) = \begin{cases} 1 & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases} \tag{2.2}$$

Tale forma decimale rappresenta il valore finale associato al pixel p_c. Si noti che ogni singolo bit nello *LBP-code* ha la stessa importanza degli altri e che due successivi valori hanno un significato totalmente differente. L'-operatore LBP non è influenzato da trasformazioni monotoniche su scala di grigi e mantiene l'ordine di intensità dei pixel vicini. Grazie alla sua proprietà discriminativa della *texture* e alla sua bassa complessità computazionale, l'algoritmo LBP è molto diffuso nella "*pattern recognition research*".
Tuttavia questo operatore LBPH ha un inconveniente legato alla dimensione dell'intorno; essa infatti ha un valore prefissato pari a 3 mentre potrebbe essere preferibile utilizzare intorni di dimensione diversa. Per tale motivo l'operatore LBP è stato esteso (*Extended Local Binary Patterns*) per gestire intorni di dimensione variabile. L'idea è quella di allineare un numero arbitrario di pixel vicini su un cerchio di raggio variabile, il quale è in grado di ottenere i vicinati rappresentati in Figura 2.2.

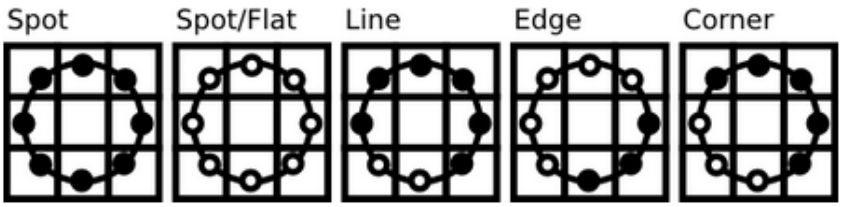


Figura 2.2: Possibili tipi di vicinati per l'operatore *Local Binary Pattern* esteso.

Dato il pixel centrale p_c, la posizione del vicino (x_p, y_p), $p \in P$ può essere calcolata nel seguente modo:

$$x_p = x_c + R\cos(\frac{2\pi p}{P}) \tag{2.3}$$

$$y_p = y_c - R\sin(\frac{2\pi p}{P}) \tag{2.4}$$

dove R è il raggio del cerchio e P il numero dei punti campione.
Se le coordinate di un punto sul cerchio non corrispondono alle coordinate dell'immagine il punto viene interpolato, ad esempio nel seguente modo:

$$f(x,y) \approx \begin{bmatrix} 1-x & x \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix} 1-y \\ y \end{bmatrix}. \tag{2.5}$$

2.1 Implementazione

Per applicare tale operatore al problema di *face recognition* l'idea consiste nel dividere l'immagine in m regioni locali ed estrarre un istogramma da ciascuna di esse. Il vettore delle *feature* da estrarre consiste nella concatenazione di questi istogrammi locali.

In questa tesi l'algoritmo di *face recognition* basato su descrittori LBPH è stato implementato in linguaggio di programmazione `C++` utilizzando la libreria *open source* **OpenCV**. La funzione `createLBPHFaceRecognizer()` permette la creazione del modello LBPH il quale utilizza come descrittori i *Local Binary Patterns* estesi. I valori di default del LBP esteso sono:

$$
\begin{aligned}
radius &= 1 \\
neighbors &= 8 \\
grid\_x &= 0 \\
grid\_y &= 0
\end{aligned}
$$

ma possono essere modificati semplicemente passando dei parametri al modello.
La funzione `train` permette di allenare il modello LBPHFaceRecognizer con un dataset di immagini e label di training, mentre la funzione `predict` predice la label di una immagine di test. Tale funzione permette di ottenere come output anche la confidenza sulla stima predetta.
Per la gestione di quattro classi rappresentanti le persone da riconoscere e di una classe *unknown* per i soggetti non noti, la scelta progettuale è stata: allenare il modello LBPHFaceRecognizer con le immagini e le label delle persone note e gestire la classe *unknown* in fase di classificazione. Ovvero, la confidenza sulla stima restituita dal metodo `predict` verrà confrontata con una soglia; se risulterà inferiore ad essa allora la classe predetta sarà quella *unknown*. Un estratto del codice è riportato in 2.1.

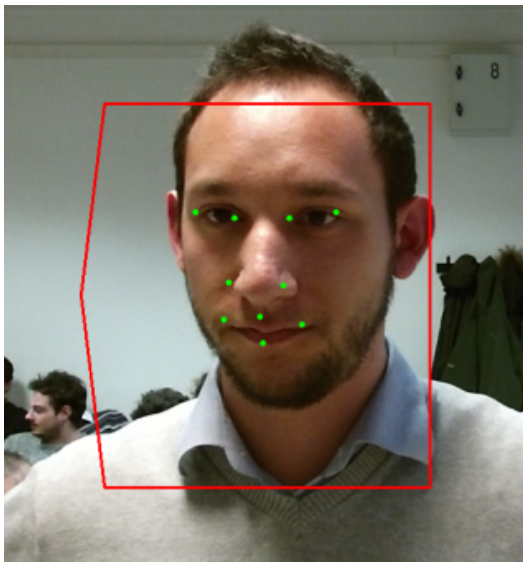
Listing 2.1: *Face recognition* basato su descrittori *LBPH*.

```
//Loading training images and labels
std::vector<cv::Mat> images;
std::vector<int> labels;
std::string yaml = "../training.yaml";
loadTrainingImages(yaml,images,labels);
if(images.size() <= 1) {
        CV_Error(CV_StsError, error_message);
}

//Loading testing image and label
cv::Mat testSample = cv::imread("../000_00_img.png");
int testLabel = 0;

//Creating and training LBPH model
Ptr<FaceRecognizer> model=createLBPHFaceRecognizer();
model->train(images, labels);

//Classification
int predictedLabel = model->predict(testSample);
```

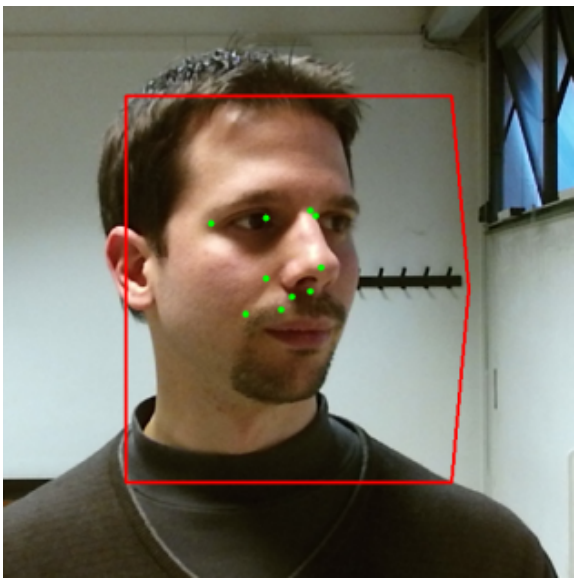
Capitolo 3

Utilizzo dell'informazione 3D per l'invarianza alla posa

Uno degli aspetti più critici nel *face recognition* basato su *feature* locali è l'accuratezza nella localizzazione dei punti di interesse, detti *keypoint*. La maggior parte degli algoritmi 2D presenti in letteratura ottiene migliori risultati quando il dataset contiene solo pose frontali, in quanto i punti identificativi del volto (occhi, naso e bocca) sono più visibili. In Figura 3.1b si nota come, a causa della rotazione del volto del soggetto ripreso, il rilevamento dei *keypoint* non è preciso quanto nel caso frontale (Figura 3.1a). Per questo motivo, negli ultimi anni, è stato introdotto l'uso dell'informazione 3D per rendere i sistemi di *face regonition* più robusti soprattutto a cambiamenti di posa. Inoltre, l'avvento di telecamere commerciali RGB-D, come la *Microsoft Kinect*, ha permesso un utilizzo sempre più ampio di tale informazione in vari campi.
L'algoritmo implementato in questa tesi, in particolare, utilizza l'informazione 3D per processare i campioni sia di training che di testing al fine di ruotarli in posizione frontale permettendo agli algoritmi 2D di *feature extraction* già presenti in letteratura di lavorare nelle condizioni ottimali.
In dettaglio, sono stati sviluppati dei metodi per le seguenti operazioni di processing: stima e correzione della posa del soggetto 3D, proiezione di esso su un piano 2D e simmetrizzazione del risultato ottenuto.



(a) Rilevamento di *keypoint* su soggetto con posa frontale.



(b) Rilevamento di *keypoint* su soggetto con posa non frontale.

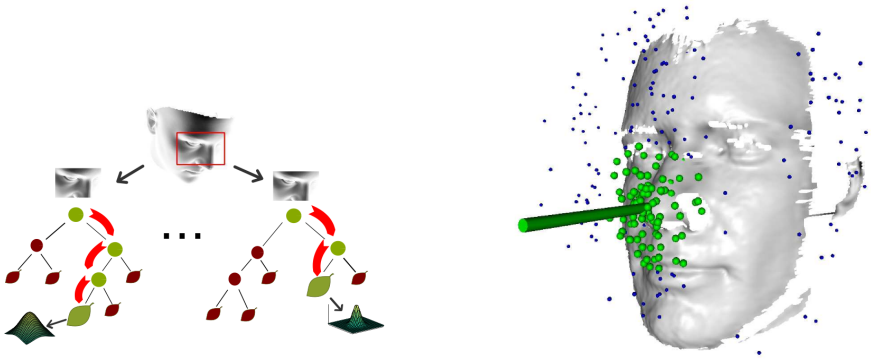
Figura 3.1: Robustezza degli algoritmi di *feature detection* in presenza di pose non frontali.

3.1 Stima della posa

La stima dell'orientazione e più in particolare della posa del volto è una componente determinante per ottenere un sistema di *face recognition* robusto.
L'idea generale è di implementare algoritmi che svolgano questo compito in modo automatico, preciso, veloce e robusto. In letteratura si trovano numerosi algoritmi che operano su immagini 2D e si possono distinguere principalmente due grandi categorie: i metodi che si basano sull'individuazione di *patch* del volto identificative per la particolare posa e quelli che invece cercano dei punti salienti della faccia come occhi, naso e bocca.
Tuttavia tali metodi 2D , come precedentemente detto per i metodi generali di *face recognition* 2D, sono molto sensibili a variazioni di illuminazione, occlusioni e assenza di definizione del volto. Per tale motivo la ricerca si è indirizzata verso l'utilizzo dell'informazione 3D, grazie anche alla diffusione di sensori di profondità economici.
Gli algoritmi 3D, presenti in letteratura, sfruttano le conformità della geometria del volto e spesso anche le *texture* per stimarne l'orientazione. Dato che tali metodi devono gestire grandi quantità di dati, per garantire l'esecuzione in *real-time*, vengono implementati per funzionare in GPU. Sfortunatamente le GPU richiedono molta potenza e spesso quindi non sono installate in scenari dove si desidera mantenere un alto livello di portabilità, ad esempio nei robot mobili.

Fanelli et al. in [4] propongono un algoritmo efficiente per la stima della posa del volto che utilizza l'informazione di profondità. Tale algoritmo riesce ad operare in *real-time* anche senza il supporto della GPU, è robusto a occlusioni parziali e a rotazioni del volto molto ampie. Il principio di funzionamento, descritto in [5], si basa sul concetto di *Discriminative Random Regression Forests* (DRRF), il quale unisce un processo di classificazione e uno di regressione. In particolare ad un set di alberi di decisione (*decision trees*), allenati opportunamente, vengono passate delle *patch* estratte in modo random dall'immagine. Ogni albero fornisce dei parametri della posa sotto forma di una distribuzione normale di probabilità con determinata media e varianza, come illustrato nello schema di Figura 3.2a. Infine attraverso la determinazione di un insieme di soglie sulla varianza vengono selezionate solo le distribuzioni
significative.
Sfruttando tali informazioni, attraverso una sequenza di *mean-shift*, viene identificata la posizione del naso e dopo una rimozione degli *outlier* ne viene stimata l'orientazione sommando le distribuzioni rimanenti. Il risultato è osservabile in Figura 3.2b.

La forza di questo algoritmo si basa sul fatto che, una volta allenati gli alberi di decisione, è estremamente efficiente, non necessita di alcuna inizializzazione e lavora senza particolare sforzo frame per frame.
Inizialmente l'algoritmo era stato implementato per operare su *range image* molto definite, poi riadattato in [6] per operare anche su *point cloud*, acquisite da sensori commerciali come la *Kinect* quindi con un basso rapporto segnale-rumore.



(a) Schema di principio della classificazione ad alberi.

(b) Risultato della stima finale della posa della faccia. Il cilindro verde rappresenta la direzione frontale, calcolata come media dei punti verdi. I punti blu sono stati considerati come outlier.

Figura 3.2: Principio di funzionamento dell'algoritmo proposto in [4], [5], [6].

3.2 Correzione della posa

Al fine di rendere il nostro algoritmo robusto alle variazioni di posa, è stato utilizzato il codice *open source* sviluppato da *Fanelli et al.* [4] per stimare la posa dei soggetti presenti nel dataset. Tale codice è stato incapsulato all'interno di una classe ad hoc chiamata `PoseEditor`, la quale data una *point cloud* in ingresso restituisce la stessa ruotata e centrata con posa facciale frontale. Il *flow chart* che ne descrive il funzionamento, rappresentato in Figura 3.3, prevede la configurazione e l'inizializzazione dell'oggetto `PoseEditor`, la stima della posa e la correzione di essa.
I parametri di configurazione richiesti dall'oggetto `PoseEditor` sono le strutture ad albero per la stima della posa. Nel caso in esame, è stato scelto di utilizzare le configurazioni di default fornite assieme al codice *open source*.
In fase di inizializzazione viene costruita una matrice in cui ogni elemento

contiene le coordinate spaziali del corrispondente punto scena. Per semplicità di esposizione, tale matrice verrà chiamata *matrice 3D*. In [4] tali coordinate vengono calcolate da una immagine RGB e dall'immagine corrispondente di profondità a differenza di quanto scelto in questa tesi dove vengono estratte direttamente dalle *point cloud* presenti nel dataset fornito. L'operazione di costruzione di tale matrice risulta agevolata dal fatto che le *cloud* sono organizzate permettendo una corrispondenza 1:1 tra gli elementi della matrice e i punti della *cloud*. La Figura 3.4 aiuta a comprendere meglio questa operazione.

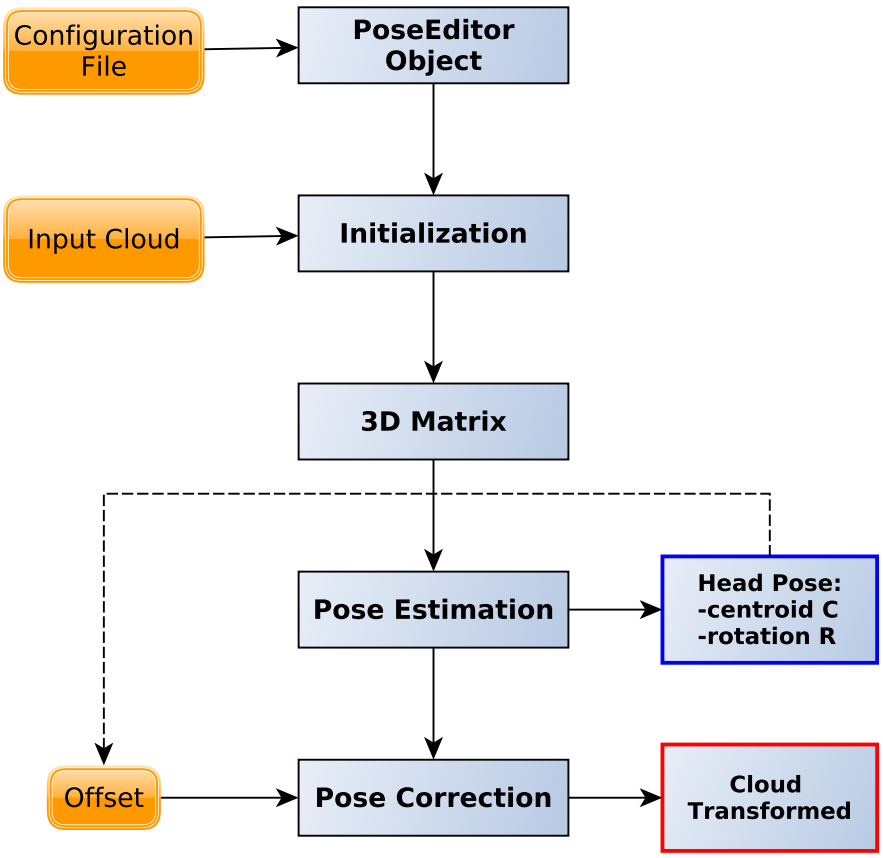


Figura 3.3: Flow chart dell'algoritmo di stima e correzione della posa.

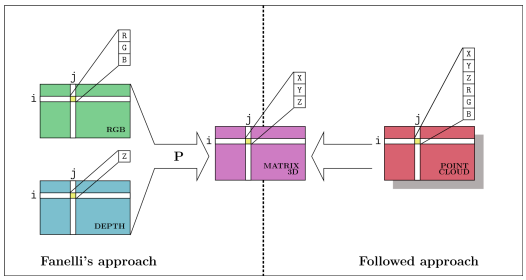


Figura 3.4: Calcolo della matrice 3D: approccio di *Fanelli et al.* [4] e approccio implementato.

La matrice 3D rappresenta l'input effettivo dell'algoritmo [4] il quale permette di ottenere il centroide della faccia $C = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$ e la rotazione rispetto all'origine del sistema di riferimento della camera $R = \begin{bmatrix} \theta_r & \theta_p & \theta_y \end{bmatrix}$, espressa come angoli di Eulero *roll-pitch-yaw* in radianti. Queste stime permettono di definire la seguente matrice di trasformazione T:

$$T = \begin{bmatrix} R^T & t \\ 0 & 1 \end{bmatrix}, \qquad t = C - (R^T C) \tag{3.1}$$

dove per R si intende, con abuso di notazione, la matrice di rotazione individuata dagli angoli $\theta_r, \theta_p, \theta_y$ e per t il vettore di traslazione per riposizionare la *cloud* approssimativamente nella posizione originale.
Tale trasformazione, applicata ad ogni punto 3D della *cloud* riscritto in coordinate omogenee, permette di roto-traslare la stessa per ottenere il soggetto con posa frontale.
In Figura 3.5 sono mostrati i passaggi intermedi per ruotare la *cloud* frontalmente alla camera. A sinistra è mostrato il confronto tra la *cloud* originale (blu) e la *cloud* ruotata tramite la matrice R^T (verde). Si nota che, a causa della rotazione attorno all'origine del frame della camera, la *cloud* verde risulta notevolmente spostata dalla posizione originale. Per tale motivo si è resa necessaria anche la traslazione finale t per riposizionare il tutto nella posizione di partenza. A destra è riportato il risultato finale.

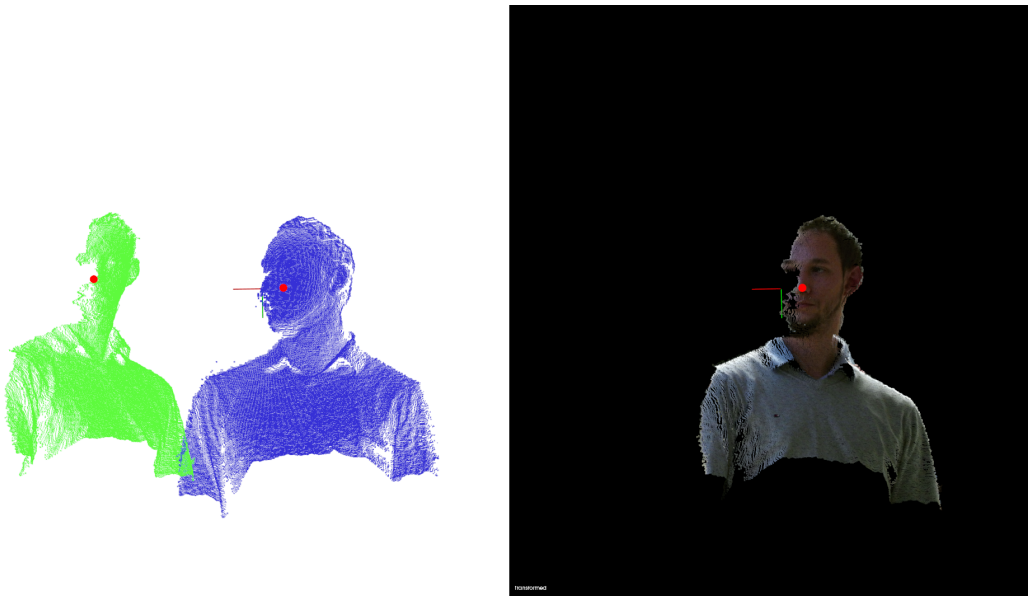


Figura 3.5: Screenshot della visualizzazione della roto-traslazione della *point cloud* di un campione del dataset.

Algorithm 1 Pose correction method

input: original cloud, final_position
output: transformed cloud
create the object pose_editor of class `PoseEdit`
set the input cloud
initialize pose_editor
estimate the pose of the subject
get the position of the centroid of the head from the pose estimated
get the rotation R of the head from the pose estimated
compute the final translation t to move the subject in the target position (t = target position-R*centroid)
transform the cloud with the tf (R, t) using the `PCL` function **transformPointCloud**
return transformed cloud

3.3 Proiezione di *point cloud*

A partire dalle *point cloud* trasformate, lo step successivo consiste nel costruire un nuovo dataset di immagini RGB per poter applicare l'algoritmo di *feature detection* [7] descritto in Sezione 4.1.
A tal fine sono state sfruttate le leggi della geometria proiettiva per passare da un punto tridimensionale ad un punto nello spazio 2D. La matrice di proiezione P, caratteristica del sensore RGB della *Kinect* utilizzata, è stata fornita come parametro di progetto:

$$P = \begin{bmatrix} f_x & 0 & \sigma_x & 0 \\ 0 & f_y & \sigma_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1052.67 & 0 & 962.41 & 0 \\ 0 & 1052.02 & 536.22 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.2}$$

Tuttavia, data la corrispondenza 2 : 1 tra immagini RGB e *point cloud*, per proiettare quest'ultima su una immagine di uguali dimensioni è stata utilizzata la matrice P ritarata nel seguente modo:

$$\bar{P} = \begin{bmatrix} \frac{f_x}{2} & 0 & \frac{\sigma_x}{2} & 0 \\ 0 & \frac{f_y}{2} & \frac{\sigma_y}{2} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} \frac{1052.67}{2} & 0 & \frac{962.42}{2} & 0 \\ 0 & \frac{1052.02}{2} & \frac{536.22}{2} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{3.3}$$

Considerando un generico punto 3D, scritto in coordinate omogenee, $\tilde{Q} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$, il punto 2D in coordinate omogenee, $\tilde{q} = \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \lambda \end{bmatrix}$, si ottiene nel seguente modo:

$$\tilde{q} = P\tilde{Q} \qquad \Longrightarrow \qquad q = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{\tilde{u}}{\lambda} \\ \frac{\tilde{v}}{\lambda} \end{bmatrix} \tag{3.4}$$

3.4 Mirroring

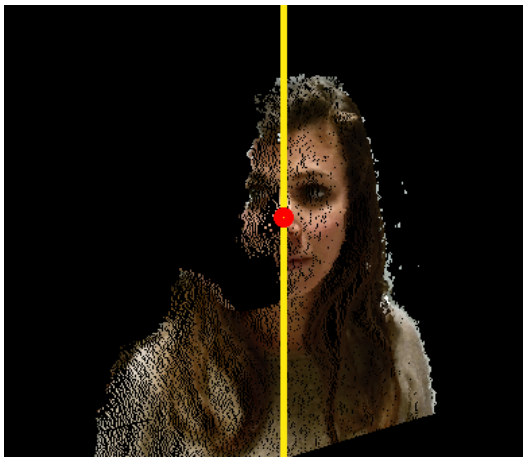
Una immagine RGB, ottenuta come proiezione di una *point cloud*, può presentare alcuni artefatti dovuti appunto alla proiezione oppure alla scarsa densità della *cloud*.
Infatti, le *cloud* di soggetti con pose non frontali non contengono punti delle porzioni di corpo occluse al sensore. Ciò comporta che, trasformando le *cloud* per ottenere la posa frontale, tali parti non possono essere visualizzate. Di conseguenza anche la proiezione soffrirà di tale problema, presentando aree prive di *texture*. Un esempio esemplificativo è riportato in Figura 3.6 nel quale si può notare come il volto del soggetto sia presente solo per metà.

(a) Immagine RGB originale di un soggetto con volto ruotato di circa 45°.

(b) Immagine RGB ottenuta proiettando la corrispondente *point cloud* ruotata frontalmente.

Figura 3.6: Problema della mancanza di punti dopo la rotazione e la proiezione su piano bidimensionale della *point cloud*.

Tale problema può essere affrontato ricostruendo la parte di volto mancante attraverso un processo di simmetrizzazione (*mirroring*), il quale può essere effettuato su dati 3D o su dati 2D. La scelta progettuale seguita in questa tesi è stata quella di applicare il *mirroring* alle immagini RGB ottenute dopo la proiezione, in quanto fornisce risultati migliori oltre ad essere computazionalmente più efficiente.



(a) Individuazione del piano di simmetria.

(b) Immagine RGB simmetrizzata.

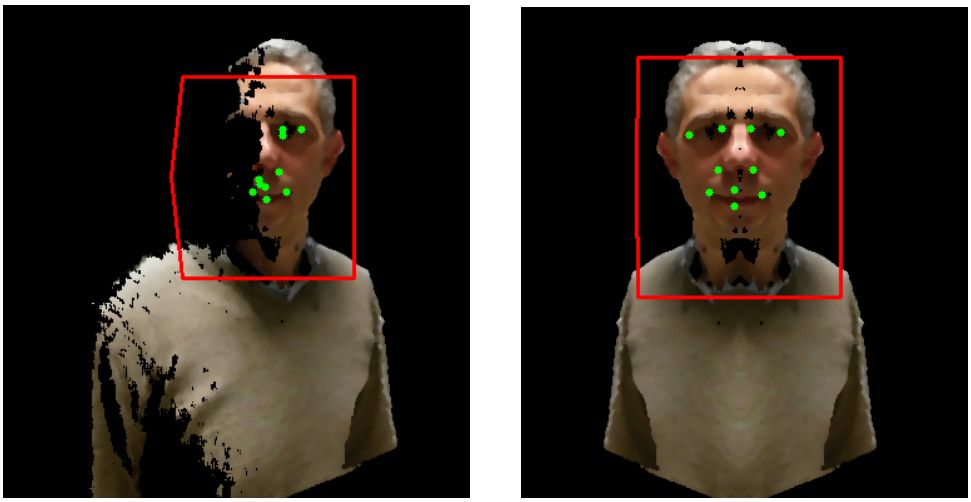
Figura 3.7: *Mirroring*.

Per eseguire il *mirroring* si è partiti dalla conoscenza della posizione 3D del centroide C della faccia e dall'allineamento della *cloud* con il frame della camera garantito dall'algoritmo [4].
In primo luogo sono state determinate le coordinate nell'immagine RGB del centroide attraverso la proiezione:

$$c = \begin{bmatrix} u_c \\ v_c \end{bmatrix} = PC \tag{3.5}$$

dove P è la matrice di proiezione.
Presupponendo che il soggetto sia perfettamente verticale, è stato individuato l'asse di simmetria come l'asse verticale passante per il centroide c.
Successivamente, grazie al segno dell'angolo di *pitch* restituito dalla stima

della posa, è stato possibile distinguere la parte da simmetrizzare.
In Figura 3.8a sono evidenziati i passaggi appena descritti e in Figura 3.8b è illustrata l'immagine finale.
In particolare, come si nota in ques'ultima, il volto ricostruito per simmetria non identifica il soggetto originario. Ovvero, se si dovessero calcolare descrittori globali su tale immagine i risultati sarebbero molto lontani da quelli ottenuti dal dato originale. Tuttavia, nel caso di estrazione di descrittori locali è lecito supporre che i descrittori di una metà del volto siano simili a quelli dell'altra metà.
L'efficacia di questa tecnica e l'effettivo miglioramento dei risultati supposti verranno
analizzati in Sezione 6.
Per il momento risulta interessante notare come, attraverso il *mirroring*, il rilevamento dei *keypoint* tramite [7] risulti essere più preciso, come evidenziato in Figura 3.8. Tale infatti era la motivazione alla base della correzione della posa.



(a) Rilevamento dei *kepoint* su volto non simmetrizzato.

(b) Rilevamento dei *kepoint* su volto simmetrizzato.

Figura 3.8: Miglioramenti apportati dal *mirroring* per quanto concerne il rilevamento dei punti salienti del volto.

Come si nota in Figura 3.8, le immagini RGB ottenute sono state filtrate prima di procedere all'estrazione delle feature. Questo è dovuto al problema dei buchi neri causato dalla proiezione su un piano bidimensionale di *point cloud* non dense, come precedentemente accennato. Come verrà descritto in

seguito, un tipo di descrittore che verrà calcolato si basa sull'informazione RGB. Si deduce quindi che la presenza di pixel neri comprometterebbe il risultato e creerebbe descrittori non in grado di identificare in modo univoco la determinata zona del volto.

Il filtro utilizzato è il *Median Filter*, fornito dalla libreria **OpenCV** e appartenente alla famiglia dei filtri spaziali.
Un filtro spaziale, detto anche maschera o kernel, è una regione rettangolare caratterizzata da un'operazione predefinita. L'operazione viene eseguita sui pixel dell'immagine corrispondenti alla suddetta regione, e le modifiche indotte dal filtro generano la cosiddetta immagine filtrata.
Il filtro opera sovrapponendosi all'immagine partendo generalmente dall'angolo in alto a sinistra. Quando il punto centrale (*anchor point*) coincide con il pixel da modificare, quest'ultimo verrà trasformato nel nuovo pixel dell'immagine di output. Il kernel si sposta da sinistra verso destra, e procede in questo modo fino a termine riga per poi ripartire dalla riga successiva modificando i valori di input nel modo indicato. Il procedimento si ripete in questo modo per tutti i punti dell'immagine.
In particolare, il *Median Filter* è un filtro non lineare che sostituisce ad ogni pixel dell'immagine il valore mediano dei suoi vicini, incluso il pixel stesso (Figura 3.9). Ha quindi il vantaggio di non introdurre nuovi valori di grigio nell'immagine elaborata.
Il valore è calcolato ordinando tutti i pixel dell'intorno prescelto, e sostituendo al pixel in esame il valore centrale dell'insieme ordinato (se l'intorno contiene un numero pari di pixel si prende la media dei due valori centrali). In genere, i filtri non lineari permettono di operare selettivamente attenuando il rumore e preservando le strutture principali dell'immagine.

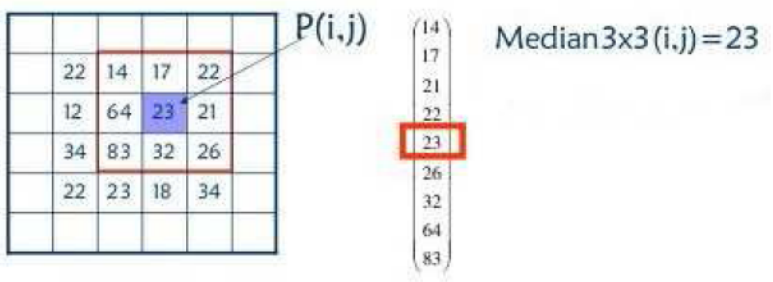


Figura 3.9: *Median filter* con kernel = 3.

Un modo alternativo per filtrare le immagini potrebbe essere un algoritmo di dilatazione. Tale algoritmo assegna come valore RGB a p la media degli stessi valori dei pixel non neri appartenenti all'intorno di p.

Come sopra spiegato, la simmetrizzazione del volto è stata effettuata sui dati RGB. Il passaggio successivo è stato simmetrizzare anche le *point cloud*. A tal fine, tutte le elaborazioni effettutate sulle immagini RGB sono state applicate anche alle immagini di profondità (*depth image*) ottenute dai dati RGB-D iniziali. Unendo l'immagine RGB simmetrizzata con la corrispondente immmagine di profondità si ottiene un dato tridimensionale anch'esso simmetrizzato.

Capitolo 4

Estrazione dei descrittori

Il problema di *face recognition*, o più in generale di *object recognition*, si basa sulla localizzazione di punti di interesse attorno ai quali vengono poi estratti i descrittori.
Per punti di interesse, o *keypoint*, si intende punti salienti il più possibili identificativi di quella porzione di immagine nel caso bidimensionale o di *point cloud* in quello tridimensionale.
Idealmente i *keypoint* devono essere il più possibile:

- **sparsi:** in quanto i punti significativi di un oggetto devono essere pochi e isolati;
- **ripetibili:** un particolare *keypoint* rilevato in una scena dovrebbe essere trovato in ogni altra scena simile e nella stessa posizione;
- **distinguibili:** devono identificare nel miglior modo la porzione di scena circostante per il calcolo di descrittori il più possibile univoci.

In letteratura, per il rilevamento di tali punti, esistono diversi *keypoint detector* implementati per applicazioni sia su immagini che su *point cloud*. Gli algoritmi si differenziano per velocità computazionale, robustezza e impiego di risorse e vengono adottati in base al particolare scopo (generalmente *registration* o *recognition*).
Una volta determinati i punti di interesse, vengono calcolati i descrittori, ovvero strutture
che codificano l'informazione in un intorno di ciascuno di essi.
I descrittori devono permettere il confronto tra *feature* e quindi devono possedere le seguenti proprietà:

- invarianza a trasformazioni rigide della scena;

- invarianza a *oversampling* e *undersampling*;
- robustezza alla presenza di rumore.

4.1 Rilevamento dei keypoint

I punti salienti del volto, importanti per il riconoscimento facciale, sono occhi, naso e bocca. Per localizzarli è stato utilizzato l'algoritmo *open source* sviluppato da *Dantone et al.* in [7]. Tale metodo si trova in letteratura tra i metodi di *feature extraction* 2D robusti soprattutto a variazioni di illuminazione.
L'algoritmo utilizza il concetto generale di *Random Forest Classifiers* per la localizzazione *real-time* di dieci *feature point* sulla faccia da immagini 2D anche di bassa qualità, come mostrato in Figura 4.1.

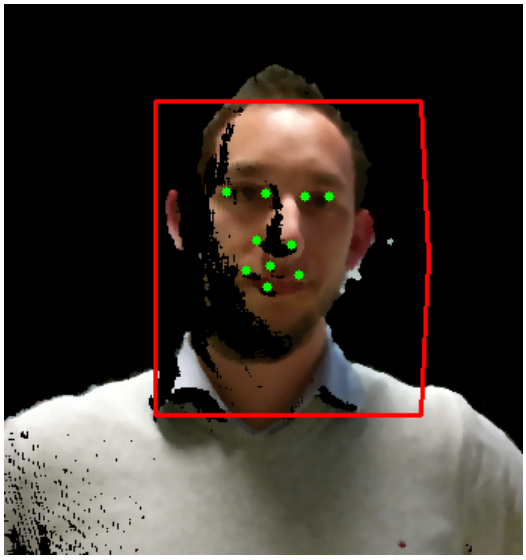


Figura 4.1: *Keypoint* 2D rilevati attraverso l'algoritmo di *Dantone et al.*

In [7] il concetto di *Regression Forest* viene esteso a quello di *Conditional Regression Forest* per il rilevamento dei *keypoint*. La differenza consiste nel fatto che le *Regression Forest* imparano le relazioni tra patch delle immagini delle facce e la posizione dei punti di interesse dall'intero set di facce a disposizione mentre le *Conditional Regression Forest* apprendono tali relazioni in modo condizionato da proprietà globali del volto.
Dato che molte variazioni di forma e illuminazione del volto dipendono da

proprietà globali di esso, in fase di training è possibile quindi allenare gli alberi in modo condizionato a tali proprietà. Successivamente, durante la fase di testing, un set di alberi pre-allenati viene selezionato in base alla stima delle proprietà globali.
Più in dettaglio, nel lavoro in questione viene considerata come proprietà globale la posa del volto. Come evidenziato in Figura 4.2, una *Conditional Regression Forest* è costituita da più foreste che sono allenate su un sottoinsieme del training set di facce scelto in base alla posa del volto (colorati in rosso, giallo e verde). In fase di testing di una immagine (facce in basso a sinistra in Figura 4.2) viene inizialmente stimata la posa del volto e i corrispondenti alberi delle varie foreste vengono selezionati per stimare le posizioni dei punti di interesse.

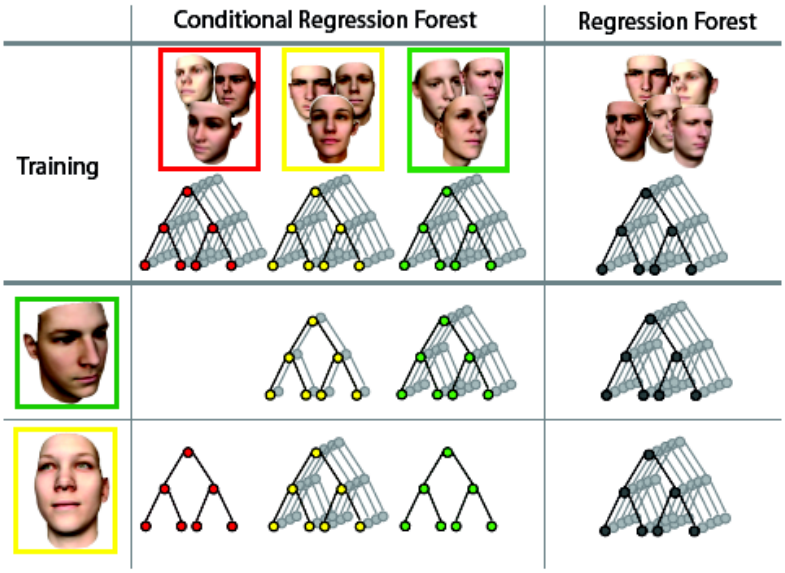
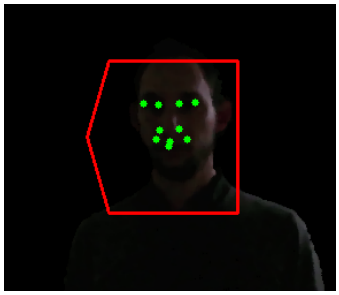
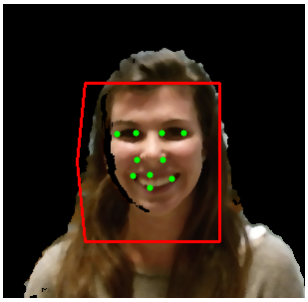


Figura 4.2: Fasi di training e testing basate su *Conditional Regression Forest* e *Regression Forest* [7].

(a) Soggetto a distanza di 2 metri dal sensore.



(b) Soggetto ripreso al buio.



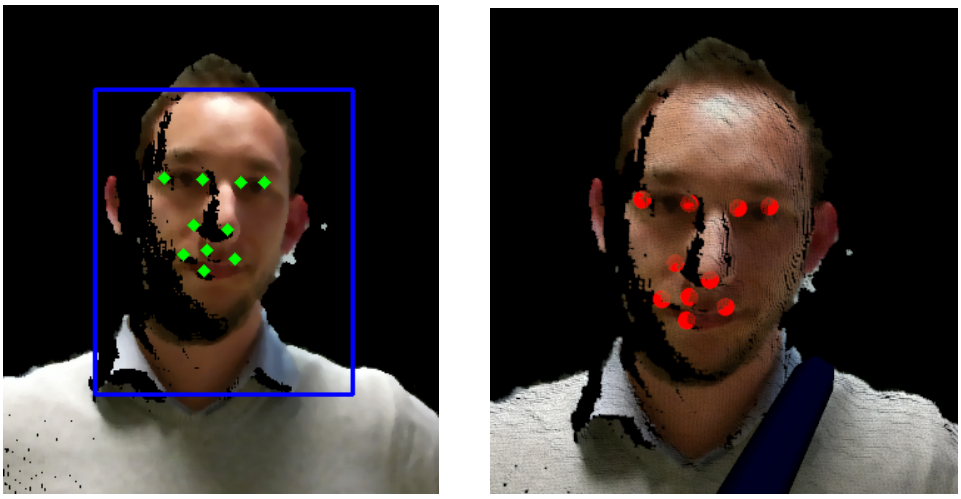
(c) Soggetto con espressione facciale non standard (sorriso).

Figura 4.3: Localizzazione di *keypoint* in diverse condizioni.

Un importante aspetto da sottolineare è che l'algoritmo risulta essere robusto, oltre che a variazioni di illuminazione, anche a variazioni di distanza del soggetto dalla camera (Figura 4.3a) e variazioni di espressione (Figura 4.3c). Infine si ottengono discreti risultati addirittura in condizioni di illuminazione molto scarsa (Figura 4.3b).

Come precedentemente accennato, in questa tesi si vuole sfruttare l'informazione 3D anche per l'estrazione delle *feature*. Per far ciò bisogna determinare la posizione dei *keypoint* non solo sulle immagini RGB ma anche sulle *point cloud*. Nonostante esistono metodi già implementati a tal fine, grazie alla particolare struttura del dataset a disposizione, ogni *keypoint* 2D è stato mappato direttamente nello spazio 3D. Infatti *point cloud* e immagini RGB sono in corrispondenza $1:1$ e ciò permette di accedere all'elemento 3D unicamente con le coordinate 2D.
Resta però da gestire il problema dei punti NaN (*Not a Number*) presenti nella *cloud*. Potrebbe succedere che un keypoint 2D venga mappato proprio su tale punto NaN. Per risolvere questo problema è stato sviluppato un particolare sistema di filtraggio che utilizza un kernel adattabile. Ad ogni punto p NaN presente viene assegnata come coordinata z la media delle stesse coor-

dinate dei punti non NaN appartenente all'intorno di p definito da un kernel iniziale. Se esiste almeno un punto valido in tale intorno la procedura termina altrimenti il kernel viene esteso con un incremento fisso e la procedura ripetuta.
Lo pseudocodice del metodo è riportato in 2, mentre la Figura 4.4 mostra i keypoint sia sull'immagine 2D che quelli mappati sul dato RGB-D.



(a) *Keypoint* 2D. (b) *Keypoint* 3D

Figura 4.4: Filtraggio di *point cloud* con kernel adattabile.

Algorithm 2

```
initialize local variables
dimension of kernel H = H_0
for each keypoint 2D do
  map the 2D keypoint in 3D space
  if 3D keypoint is NaN then
    repeat
      define the neighboorod of 3D keypoint as K with dimension H
      increment H
    until at least one point ∈ K is valid
    compute the average of z-coordinates of valid points ∈ K
  end if
  set the point's new 3D coordinates
end for
```

4.2 Descrittori

Una volta determinate le posizioni dei *keypoint* il passaggio successivo consiste nel codificare l'informazione dei pixel appartenenti al vicinato del punto calcolando un descrittore.
In questa tesi si vogliono inglobare le informazioni 2D e 3D e, per questo motivo, sono stati calcolati tre tipi di descrittori: i *SURF Descriptor* sulle immagini RGB, i *Fast Point Feature Histograms* e le distanze euclidee sulle *point cloud*.

4.2.1 Speed Up Robust Feature (SURF)

I descrittori *Speed Up Robust Feature* (SURF) risultano essere invarianti rispetto a rotazioni dell'immagine, cambiamenti di scala e di illuminazione e a cambiamenti di prospettiva
limitati.
L'algoritmo SURF consente di estrarre feature distintive (cioè ben discriminabili l'una rispetto all'altra) all'interno di un'immagine, in modo invariante rispetto a rotazioni, traslazioni, cambiamenti di scala, occlusioni parziali, deformazioni e cambiamenti di illuminazione.
La prima fase di costruzione dei descrittori prevede, per ogni punto d'interesse, la
individuazione dell'orientazione principale. Questa attività viene svolta calcolando la risposta ai filtri di Haar nelle direzioni x e y (rappresentati in

Figura 4.5), all'interno di in un intorno circolare del punto di dimensione $6s$ dove s è la scala caratteristica del punto.

Figura 4.5: Filtri di *Haar*.

Come mostra Figura 4.6, il risultato di questo processo può essere rappresentato con una serie di punti nello spazio bidimensionale e a partire da questi si determina l'orientazione dominante della *feature*.

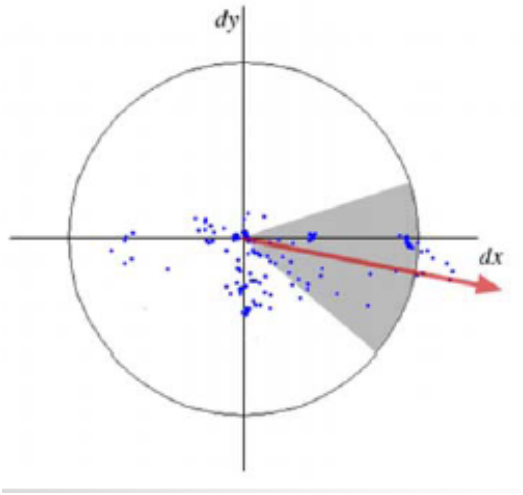


Figura 4.6: Calcolo dell'orientazione dominante.

Dopo aver determinato l'orientazione si procede con il calcolo del descrittore. Inizialmente bisogna definire una regione quadrata centrata nel punto di interesse e orientata come indicato dall'orientazione assegnata al punto.
La regione viene poi suddivisa in 4×4 sottoregioni quadrate di eguale dimensione, in modo da preservare le informazioni spaziali. Per ciascuna sottoregione, vengono calcolate le risposte alla funzione wavelet di Haar; per ragioni di semplicità, viene indicata con d_x la risposta alla funzione wavelet lungo l'asse orizzontale e con d_y la risposta alla funzione wavelet lungo l'asse verticale, dove naturalmente il concetto di 'verticale' e 'orizzontale' è da intendersi in relazione alla specifica orientazione della finestra (Figura 4.7).

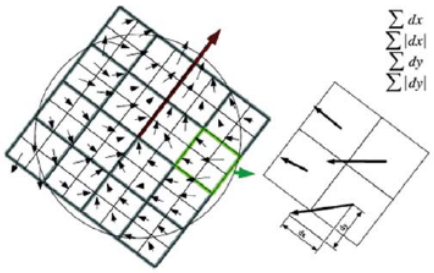


Figura 4.7: Suddivisione della finestra quadrata in sottoregioni e calcolo del descrittore a 64 componenti.

Per aumentare la robustezza verso deformazioni geometrice ed errori di localizzazione, la risposta alla wavelet viene pesata attraverso una Gaussiana centrata sulla zona d'interesse. A questo punto, sommando da un lato le risposte, indicate con d_x e d_y, ottenute da ciascuna sottofinestra e calcolando dall'altro lato la somma dei rispettivi valori assoluti, si ottiene un vettore $\mathbf{v}$ della singola sottofinestra caratterizzato da 4 componenti:

$$\mathbf{v} = (\sum d_x, \sum d_y, \sum \mid d_x \mid, \sum \mid d_y \mid) \tag{4.1}$$

Concatenando tutti i vettori delle 16 diverse regioni in cui è stata divisa la finestra, si ottiene un descrittore complessivo del punto d'interesse, descrittore composto appunto da 64 elementi.

4.2.2 Fast Point Feature Histograms (FPFH)

I descrittori 3D FPFH sono un'estensione dei descrittori *Point Feature Histograms* (PFH) con un'aumentata velocità di computazione e un minor impiego di risorse.
Entrambi si basano sulle proprietà geometriche dei punti appartenenti ad un intorno del *keypoint* su cui tale descrittore viene calcolato. Sono inoltre invarianti a variazioni di posa e sufficientemente robusti a *oversampling*, *undersampling* e alla presenza di rumore. Tuttavia la qualità del descrittore dipende fortemente dalla stima delle normali alla superficie.
Il descrittore è caratterizzato da un istogramma di valori che rappresentano le relazioni (e.g. distanza) tra coppie di punti appartenenti all'intorno considerato. Più nel dettaglio, per ogni *keypoint* della *point cloud* viene calcolato un istogramma nel seguente modo:

- per ogni punto p vengono selezionati i k vicini ovvero tutti i punti all'interno di una sfera (intorno) di raggio r e centro in p;

- per ogni coppia di punti p_i e p_j con $j \neq i$ appartenenti al vicinato del punto p viene definito un nuovo frame uvn ($u = n_i$, $v = (p_j - p_i) \times u$, $w = u \times v$) basandosi sulla stima delle normali n_i e n_j (p_i è il punto con l'angolo più piccolo tra la normale associata e la linea che connette i due punti considerati) e vengono calcolate le variazioni degli angoli delle normali n_i e n_j nel seguente modo:

$$\alpha = v \cdot n_j \tag{4.2}$$
$$\phi = u \cdot (p_j - p_i))/ \parallel p_j - p_i \parallel \tag{4.3}$$
$$\theta = arctan(w \cdot n_j, u \cdot n_j) \tag{4.4}$$

  In questo primo passo sono considerate quindi solamente le relazioni tra il punto p e i suoi vicini e viene calcolato l'istogramma definito *Simple Point Feature Histogram*;

- per ogni punto p vengono selezionati un'ulteriore volta i k vicini e i valori *SPFH* precedentemente ottenuti vengono utilizzati per pesare l'istogramma finale (FPFH) del punto p:

$$FPFH(p) = SPF(p) + \frac{1}{k} \sum_{i=1}^{k} \frac{1}{\omega_k} \cdot SPF(p_k) \tag{4.5}$$

dove il peso ω_k rappresenta la distanza tra il punto p e il punto vicino p_k definita in un dato spazio. Per capire meglio l'importanza di questo schema di pesi, la Figura 4.8 illustra lo schema a regione di influenza per un vicinato di k punti centrato in p_q.

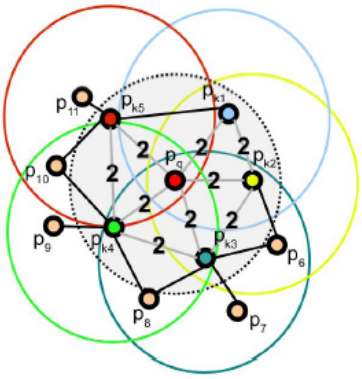


Figura 4.8: Calcolo dei *Fast Point Features Histograms*. Ogni punto p (rosso) è connesso solamente ai suoi diretti k vicini (posizionati all'interno del cerchio grigio). Ognuno di questi punti a sua volta è connesso ai suoi vicini e gli istogrammi ottenuti sono pesati assieme all'istogramma del punto p per formare l'istogramma FPF. Le connessioni segnate con il numero 2 contribuiscono due volte [8].

4.2.3 Distanze euclidee

In applicazione biometriche vengono spesso utilizzate le distanze tra occhi, naso e bocca di una persona in quanto costituiscono informazione invarianti a cambiamenti di posa, luce ed espressione.
Una delle scelte progettuali effettuate è stata quella di utilizzare le distanze euclidee tridimensionali calcolate tra i *keypoint* rilevati sui dati RGB-D come descrittori del volto del soggetto. Note le coordinate 3D dei 10 punti salienti del volto, per ogni coppia di punti (p_i, p_j) viene quindi calcolata la distanza euclidea

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{4.6}$$

Eventualmente, si potrebbe pensare di utilizzare la metrica di *Mahalanobis*, sempre più applicata in questi contesti, piuttosto che quella euclidea.
Per ogni *keypoint* viene così estratto un descrittore organizzato come un vettore di dimensione 10, ottenendo un descrittore del dato RGB-D strutturato a matrice 10×10. Caratteristica peculiare di tale matrice è, per costruzione, la diagonale nulla in quanto gli elementi sulla diagonale corrispondono alla distanza dal punto al punto stesso.

Questi tre tipi di descrittori sopra descritti risultano essere robusti in particolari situazioni e meno in altre. L'idea alla base di questa tesi, ovvero l'utilizzo

dell'informazione 2D e 3D, trova fondamento anche in questo modulo di estrazione di *feature*. A tal fine si vuole associare ad ogni esempio del dataset un unico descrittore generale dato dalla concatenazione dei descrittori parziali e organizzato quindi come una matrice di dimensione $(64 + 33 + 10) \times 10$. Tale operazione può essere estesa ad un numero qualsiasi di descrittori parziali purché calcolati sullo stesso numero di *keypoint*. Tuttavia bisogna sempre porre particolare attenzione a non aggiungere informazione ridondante o, nel caso peggiore, informazione contrastante con il rischio di peggiorare i risultati.

Capitolo 5

Classificatori

Il modulo successivo all'estrazione delle *feature*, in un sistema di *face recognition*, è costituito dal training di classificatori per costruire un database di modelli su cui classificare l'esempio di testing.
In letteratura si trovano diversi tipi di classificatori, alcuni dei quali, come ad esempio il *Nearest Neighbor*, risultano molto semplici da realizzare, ma, al contempo, si rivelano spesso fortemente sensibili alle eventuali variazioni presenti nelle immagini e il loro esito non è quindi sempre soddisfacente; altri classificatori, al contrario, sono pensati per essere più robusti e dunque più resistenti nei confronti delle possibili variazioni, ma, d'altra parte, la loro realizzazione risulta meno immediata, come è il caso del *Support Vector Machine*. Inoltre esistono anche metodi più recenti ed innovativi, come ad esempio il *Deep Learning*, ma richiedono un carico computazionale e un numero di esempi di training maggiore. In questa tesi è stato preferito quindi utilizzare metodi già consolidati negli anni e che lavorano discretamente su dataset di training non troppo ampi, come nel caso in questione. Per questo motivo la classificazione è stata effettuata tramite *Nearest Neighbor* e *Support Vector Machine*.

5.1 Nearest neighbor

Il metodo *Nearest Neighbour (NN)* richiede innanzitutto il calcolo delle distanze tra i descrittori che rappresentano il volto da riconoscere e tutti quelli delle facce contenute nel training set. Dopodichè, la classificazione vera e propria è eseguita associando l'immagine di input alla classe che contiene il volto che rende minima la distanza tra i descrittori. Esistono diversi modi per misurare il divario che separa le varie rappresentazioni dei volti, il più comune

dei quali consiste notoriamente nel calcolo della distanza euclidea. Nel corso del tempo, tuttavia, sono state proposte numerose alternative come, ad esempio, l'utilizzo della metrica di Mahalanobis. Inoltre, è possibile estendere il concetto di fondo della tecnica *Nearest Neighbour*, eseguendo la classificazione non soltanto sulla base del nodo (volto) più vicino, ma valutando nel complesso l'apporto di k nodi limitrofi, attraverso l'uso di un'opportuna regola di voto; questa evoluzione dell'impostazione originale del metodo, prende il nome di *K-nearest Neighbour*. La classificazione tramite *Nearest Neighbor* è stata implementata in ambiente `Matlab`; lo pseudocodice dell'algoritmo è riportato in Algorithm 3.

Algorithm 3 Classificazione NN

input: test example descriptors
N = nr training classes
distances = new vector(N)
for $i = 0$ **to** $i = N - 1$ **do**
  choose m random training examples of class i
  for $j = 1$ **to** m **do**
    compute distance between the descriptors of j-th training example and the descriptors of test example
    compare the distance with the previous computed distances
    if distance < i-th element of distances vector **then**
      store in the i-th element of distances vector the value of the last computed distance
    end if
  end for
end for
predicted_label = $\underset{1 \leq i \leq N}{\text{argmin}}(distances(\text{i}))$
if distances(predicted_label)>unknown_threshold **then**
  predicted_label = unknown
end if

5.2 Support Vector Machine

Il classificatore lineare *SVM* appartiene alla famiglia dei classificatori binari, i quali cercano una relazione che leghi il training set $S = \{(x_1, y_1) \dots (x_l, y_l)\} \in (\mathbb{X} \times \mathbb{Y})$ dove $\mathbb{X} \subseteq \mathbb{R}^n$ è il vettore che raccoglie i descrittori da usare per l'addestramento e $\mathbb{Y} = \{+1, -1\}$ lo spazio delle label associate.
Il classificatore *SVM* lineare partiziona, massimizzando il margine, lo spazio

delle *feature* (descrittori) calcolando un iperpiano $(\mathbf{w}, b)$ di separazione tra le due classi, come schematizzato in Figura 5.1, dove per margine si intende la minima distanza fra l'iperpiano di separazione e uno dei volti del training set.

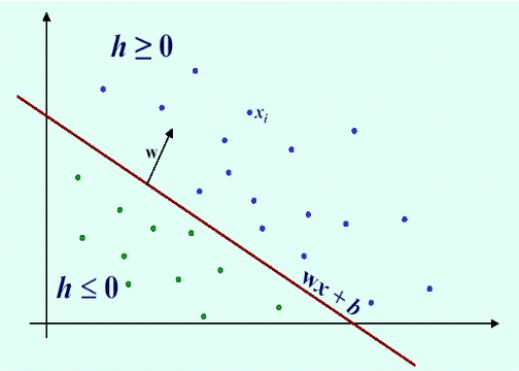


Figura 5.1: Classificatore binario SVM lineare. Individuazione dell'iperpiano che massimizza il margine tra i descrittori delle classi.

Alla fase di training, in cui viene allenato il classificatore, segue la fase di classificazione vera e propria (testing), dove si determina la stima sulla classe di appartenenza assieme ad una confidenza (distanza dall'iperpiano) su tale predizione.

5.2.1 SVM multiclasse

In questa tesi, il problema di classificazione è un problema a classe multipla in quanto l'algoritmo di *face recognition* deve stabilire, per ogni esempio di testing, se si tratta di una delle 4 persone note specificandone in particolare l'identità, oppure se si tratta di una persona sconosciuta. Questo richiede una estensione delle originarie SVM binarie alle SVM multiclasse per la gestione di 4 classi, associate alle persone note, e una classe *unknown* rappresentante tutti i soggetti sconosciuti.
Innanzitutto si è scelto di non addestrare la classe *unknown* ma di gestirla in fase di classificazione, con un controllo sulla confidenza ottenuta nella predizione. Questa scelta deriva dal fatto che i campioni di training delle classi note sono in numero molto inferiore rispetto a quelli dei soggetti sconosciuti. Infatti allenando la classe *unknown* in fase di addestramento dei classificatori si rischierebbe di ottenere un iperpiano non in grado di separare efficacemente gli spazi dei descrittori considerati.

Per implementare il classifcatore SVM multiclasse ci sono due modi possibili che vengono chiamati: *one against one* e *one against all*.

SVM *one against one*

Nel caso di N classi, si costruiscono $\frac{N(N-1)}{2}$ classificatori ognuno addestrato sui dati relativi a due sole classi. Il generico classificatore viene allenato considerando gli elementi delle due classi come antagonisti, ovvero prendendo gli elementi di una delle due classi come positivi e quelli dell'altra come negativi. Questa procedura viene iterata per ogni coppia di classi.
In questo modo si ottengono $\frac{N(N-1)}{2}$ funzioni di decisione (iperpiani):

$$sign(w^{mnT}x + b^{mn}), \quad \text{per ogni } m, n \in 1, ..., N \tag{5.1}$$

In fase di testing si ottengono quindi $\frac{N(N-1)}{2}$ stime della classe di appartenenza del campione in base al classificatore usato e le corrispondenti confidenze sulla bontà delle stime. Per decidere la classe di appartenenza di un generico campione di testing si adotta una strategia di *voting*. In particolare, ad ogni classe vengono associati tre parametri:

1. **vincite:** numero di volte che la classe viene stimata come classe di appartenenza dell'esempio;

2. **punteggi:** vettore contenente il modulo delle confidenze per ogni vincita;

3. **confidenza massima:** massima confidenza ottenuta nelle vincite.

Una *vincita* viene assegnata in base al segno della confidenza ottenuta mentre il suo valore assoluto individua il punteggio.
Ad esempio, se si confronta la *classe1* con la *classe3* e si ottiene una confidenza pari a -0.8 allora si assegna una vincita alla *classe3* con punteggio 0.8.
La classe di appartenza del campione di testing x è scelta come quella che ottiene il massimo numero di vincite, dunque è ottenuta come

$$classe(x) = \underset{1 \leq j \leq N}{\text{argmax}}(f_j(x)) \tag{5.2}$$

dove $f_j(x)$ rappresenta il numero di vincite della classe j.
Nel caso di classi con lo stesso numero di voti (cioè due classi i, j tali che $f_j(x) = f_i(x)$) viene scelta quella con il parametro confidenza massima più elevato.

Tuttavia, tale classe scelta è la classe candidata di appartenenza e non quella effettiva in quanto resta da gestire la classe *unknown*. A tal fine si deve verificare che la classe candidata abbia una confidenza massima sufficientemente elevata. È stata fissata una soglia sul valore minimo di tale parametro al di sotto della quale la classe candidata viene scartata e l'esempio viene riconosciuto come appartenente alla classe *unknown* mentre al di sopra della quale la classe candidata viene assunta come effettiva
In Algorithm 4 e 5 sono riportati gli pseudocodici degli algoritmi implementati in ambiente `Matlab` per il training e la classificazione con SVM *one against one*.

Algorithm 4 SVM *one against one* - fase di training

N = nr classes
m = dimension of descriptors
w = new vector $(\frac{N(N-1)}{2})$
b = new vector (N)
k = 1
for $i = 1$ **to** $N - 1$ **do**
  for $j = i + 1$ **to** $N - 1$ **do**
    N_{pos} = nr examples of i-th class
    N_{neg} = nr examples of j-th class
    store descriptors of i-th class in the first N_{pos} rows
    store descriptors of j-th class in the other rows
    labels = new vector(N_{pos}+N_{neg})
    store the value +1 in the first N_{pos} rows and the value -1 in the other rows
    compute the model with the `matlab` function **svm learning()**
    compute the parameters of the hyperplane with the `matlab` function **compute hyperplane()**
    store the parameters in the vectors w and b
    increment of 1 the counter k
  end for
end for
return w, b

Algorithm 5 SVM *one against one* - fase di testing

input = descriptors of test example
output = predicted label of test example
classes = new vector (N)
sum = new vector N //sum of confidences
compute the confidence with all class with the `matlab` function **class evaluation()**
for $i = 1$ **to** confidences.length **do**
  switch (i)
  case 1**:**
    //comparison between classes 1-2
    if confidences(i) ≥ 0 **then**
      classes(1)++
      sum(1) = sum + confidences(k)
    else
      classes(2)++
      sum(2) = sum + confidences(k)
    end if
      $\vdots$ //comparison between classes 1-3, 1-4, 2-3, 2-4
  case 6**:**
    //comparison between classes 3-4
    if confidences(i) ≥ 0 **then**
      classes(3)++
      sum(3) = sum + confidences(k)
    else
      classes(4)++
      sum(4) = sum + confidences(k)
    end if
  end switch
end for
predicted_label = $\underset{1 \leq j \leq N}{\text{argmax}}(classes(\text{j}))$
if max value is not unique **then**
  check on the max value sum(i)
end if
//check for the unknown class
if sum(predicted_label) < unknown_threshold **then**
  predicted_label = unknown
end if
return predicted_label

SVM *one against all*

In questo approccio si costruiscono N classificatori in cui N è il numero di classi. Il $j-esimo$ classificatore binario separa i vettori della classe j da quelli di tutte le altre classi. Viene quindi allenato prendendo come esempi positivi quelli appartenenti alla classe $j-esima$ e come esempi negativi quelli appartenenti a tutte le altre classi.
A differenza del caso precedente si ottengono quindi N iperpiani (w_j, b_j) come funzioni di decisione.
Per decidere la classe di appartenza di un generico campione di testing x si individua la funzioni di decisione a cui corrisponde il maggior valore dell'argomento, ovvero

$$classe(x) = \underset{i \leq j \leq N}{\operatorname{argmax}}(w_j^T + b_j) \tag{5.3}$$

Tale valore è la confidenza sulla stima della classe di appartenenza ottenuta. Per determinare la classe effettiva di appartenenza, gestendo il caso *unknown*, tale confidenza va confrontata con la soglia definita come nel caso precedente. In Algorithm 6 è riportato lo pseudocodice riguardante la costruzione della matrice di training contenente gli esempi considerati positivi e quelli negativi (diversa dal caso precedente) e in Algorithm 7 lo pseudocodice della classificazione vera e propria.

Algorithm 6 SVM *one against all* - fase di training

N = nr classess
m = dimension of descriptors
for $i = 1$ **to** $N - 1$ **do**
  N_{pos} = nr examples of i-th class
  N_{neg} = nr training examples of all classes - N_{pos}
  store descriptors of i-th class in the first N_{pos} rows
  store descriptors of all other classes in the other rows
end for

Algorithm 7 SVM *one against all* - fase di testing

input = descriptors of test example
output = predicted label of test example
compute the confidence with all class with the `matlab` function **class evaluation()**
predicted_label = $\underset{1 \leq j \leq N}{\text{argmax}}(confidences(\text{j}))$
//check for the unknown class
if sum(predicted_label) < unknown_threshold **then**
  predicted_label = unknown
end if
return predicted_label

Capitolo 6

Esperimenti

Questa sezione è dedicata alla valutazione delle prestazioni del sistema descritto in questo elaborato. Più in particolare, la Sezione 6.1 presenta il dataset acquisito per la valutazione dell'algoritmo implementato, la Sezione 6.2 riporta gli indici di riconoscimento adottati per valutare l'algoritmo e confrontare le varie tecniche, la Sezione 6.3 mostra i test effettuati sul dataset acquisito e i confronti tra le varie tecniche utilizzate ed infine la Sezione 6.4 evidenzia le prestazioni dell'algoritmo.

6.1 Dataset

Per testare le tecniche implementate descritte precedentemente e per confrontare l'algoritmo di *face recognition* con altri metodi presenti in letteratura è stato acquisito un nuovo dataset contenenti dati RGB-D.
In particolare il dataset di training, utilizzato per creare i classificatori, contiene 27 soggetti di cui 4 sono quelli da riconoscere (insieme dei soggetti noti). Ognuno di questi soggetti è stato ripreso 13 volte in diverse condizioni come riportato in Tabella 6.1.
Per quanto concerne le condizioni di ripresa si deve porre attenzione a due particolari: ogni soggetto è ritratto con un background variabile che può contenere altre persone ed inoltre il dataset è stato acquisito in due ambienti diversi in momenti diversi. Quest'ultimo fatto, in particolare, si ripercuote sulla precisione delle distanze dalla camera dei soggetti e di illuminazione.

Il dataset di testing, acquisito per testare la bontà del sistema di *face recognition*, contiene 19 soggetti distinti compresi i 4 considerati noti da riconoscere. I restanti non sono presenti nel dataset di training e dovranno essere riconosciuti come soggetti non noti (in applicazioni di sorveglianza ad esempio

a tali soggetti non deve essere consentito l'accesso).
Per verificare l'invarianza dell'algoritmo a cambiamenti di posa, espressione ed illuminazione, i soggetti di testing sono stati ripresi in diverse condizioni riassunte in Tabella 6.2.

Esempio	Posa	Illuminazione	Espressione	Distanza
1	frontale	buona	normale	1 metro
2	45° sinistra	buona	normale	1 metro
3	45° destra	buona	normale	1 metro
4	45° alto	buona	normale	1 metro
5	45° basso	buona	normale	1 metro
6	frontale	buona	sorriso	1 metro
7	frontale	scarsa	normale	1 metro
8	45° sinistra	scarsa	normale	1 metro
9	45° destra	scarsa	normale	1 metro
10	45° alto	scarsa	normale	1 metro
11	45° basso	scarsa	normale	1 metro
12	frontale	scarsa	sorriso	1 metro
13	frontale	buona	normale	2 metri

Tabella 6.1: Training dataset.

	Test1	Test2	Test3	Test4	Test5
Esempi per soggetto	2	1	2	1	1
Posa	frontale	frontale	frontale	mista	frontale
Illuminazione	buona/media	buona	buona	buona	buio
Distanza	1m	2m	1m	1m	1m
Espressione	normale	normale	sorriso/seria	normale	normale

Tabella 6.2: Testing dataset (per posa mista si intendono rotazioni del volto in alto,basso, a destra e a sinistra).

Resta da sottolineare che i valori di distanza riportati in Tabella 6.2 si riferiscono alla condizione teorica ma sperimentalmente si è osservato che la distanza reale è mediamente maggiore. Questo fatto si ripercuoterà sui risultati ottenuti, come si vedrà in seguito.

6.2 Criteri di valutazione

Per permettere una valutazione della bontà del sistema di *face recognition* sviluppato sono stati introdotti i seguenti indici di riconoscimento:

Rank-1: rappresenta il numero di riconoscimenti corretti sul totale di esempi classificati

$$rank-1 = \frac{1}{N+1} \cdot \sum_{i=1}^{N+1} \left(\frac{correct\_match(i)}{nr\_examples\_class(i)} \right) \tag{6.1}$$

dove $N \leq 4$ è il numero di classi considerate;
FP-rate: rappresenta la percentuale di volte in cui una persona sconosciuta viene erroneamente riconosciuta come una nota (*false positive*)

$$FP-rate = \frac{nr\_false\_positive}{nr\_examples\_unknown} \cdot 100 \tag{6.2}$$

FR-rate: rappresenta la percentuale di volte in cui una persona nota viene erroneamente riconosciuta come una sconosciuta (*false rejected*)

$$FR-rate = \frac{nr\_false\_rejected}{\sum_{i=1}^{N} nr\_examples\_class(i)} \cdot 100 \tag{6.3}$$

Total-matches: rappresenta la percentuale di match corretti sul totale senza considerare la suddivisione in classi

$$Total-matches = \frac{nr\_matches}{\sum_{i=1}^{N+1} nr\_examples\_class(i)} \cdot 100 \tag{6.4}$$

6.3 Test eseguiti

In questa sezione viene validato l'algoritmo sviluppato confrontandolo con il sistema descritto nel Capitolo 2 presente nello stato dell'arte. Inoltre, come descritto in questa tesi, l'algoritmo per il riconoscimento facciale può avere delle varianti:

- i descrittori estratti possono essere solo 2D, solo 3D oppure una concatenazione di entrambi i tipi;

- ai dati processati può venire applicata o meno la tecnica di simmetrizzazione del volto;
- si possono usare diversi tipi di classificatori.

I test eseguiti, riportati in questa Sezione, confrontano queste varianti del sistema implementato. Risulta importante sottolineare, però, che i seguenti grafici si riferiscono ai risultati ottenuti utilizzando i dati di testing su cui vengono rilevati i *keypoint* attorno a cui sono stati calcolati i descrittori per poi classificarli. Il numero di questi dati (riportato in Tabella 6.3) può essere diverso dal numero di dati totali presenti nel dataset di testing poichè l'algoritmo per il rilevamento dei *keypoint* [7] in alcuni casi fallisce.

	test1	*test2*	*test3*	*test4*	*test5*
Tot. Esempi caricati	38/38	18/19	38/38	16/19	12/19
Conosciuti	8/8	4/4	8/8	3/4	3/4
Sconosciuti	30/30	14/15	30/30	13/15	9/15

Tabella 6.3: Numero di esempi effettivamente caricati in fase di testing.

Il primo test serve a validare l'algoritmo proposto per dimostrare l'effettiva efficacia dell'informazione 3D confrontando i risultati con quelli ottenuti con l'algoritmo 2D basato sui descrittori LBPH descritto nel Capitolo 2. Per questo primo confronto i descrittori estratti dai dati di training e di testing sono stati ottenuti dalla concatenazione dei descrittori SURF e FPFH. Utilizzando il classificatore SVM *one against one* si ottiene un valore di rank-1 pari a circa 0.8 per soggetti con posa frontale a 1 e 2 metri dalla camera e con espressioni diverse e pari a 1 per soggetti ripresi in condizione di luce scarsa (considerando che su due soggetti del dataset ripreso l'algoritmo fallisce nel rilevare i *keypoint* e quindi non possono essere estratti i descrittori da classificare). Per quanto concerne invece soggetti con posa non frontale si ottengono risultati discreti (rank-1 pari a 0.6), i quali dimostrano come l'utilizzo dell'informazione 3D per correggere la posa permette un miglioramento rispetto i metodi 2D anche se tale tecnica necessita di ulteriori affinamenti per per ottenere un rank-1 pari alle altre condizioni di ripresa. Con l'utilizzo il classificatore SVM *one against all*, le performance del sistema in termini di rank-1 sono leggermente inferiori ma il numero di falsi positivi si riduce praticamente a zero per ogni condizione di ripresa dei soggetti.

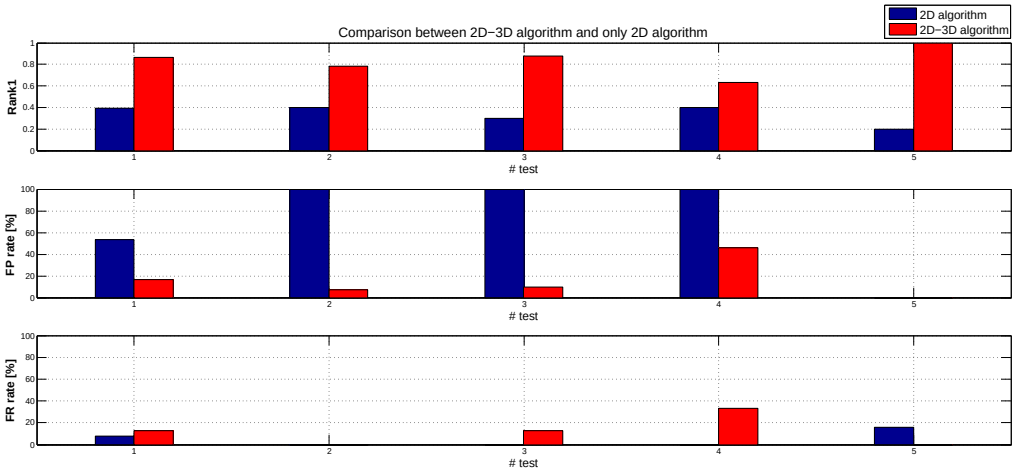
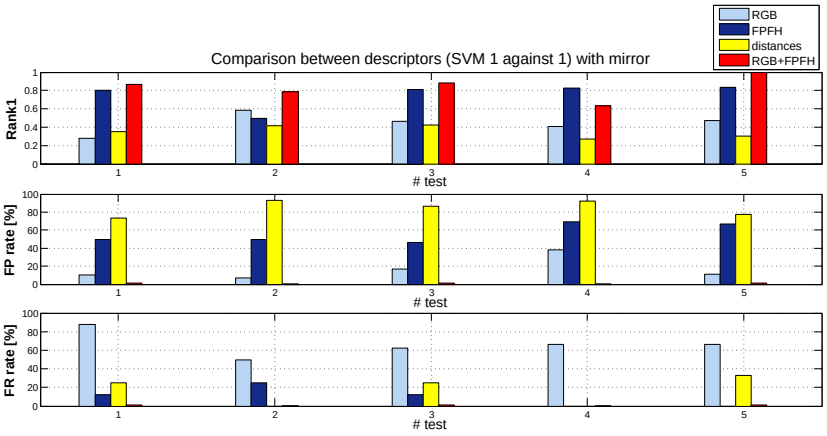
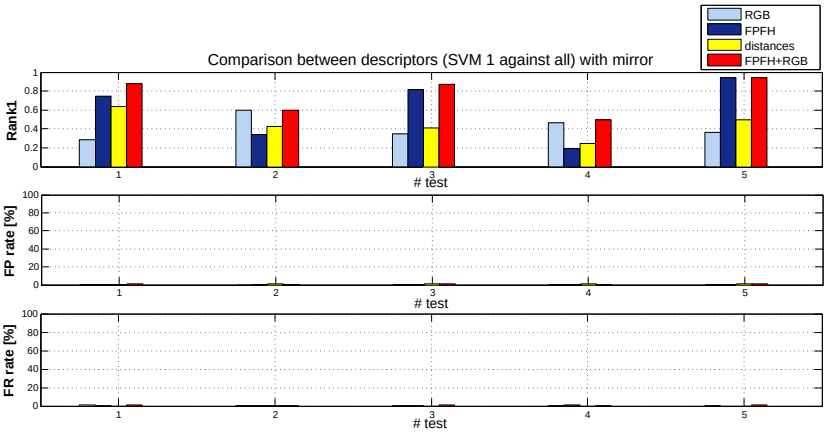


Figura 6.1: Confronto tra le prestazioni dell'algoritmo implementato e l'algoritmo 2D basato su descrittori LBPH.

Il secondo test confronta i tipi di descrittori estratti per il *matching* tra i soggetti di training e di testing ottenute in seguito alla simmetrizzazione del volto. Tale esperimento ha l'obiettivo di analizzare l'effettiva efficacia dell'informazione 3D anche per quanto riguarda l'estrazione delle *feature*. Come si trova in letteratura, ci si aspetta di ottenere performance migliori utilizzando descrittori bidimensionali rispetto ai soli descrittori tridimensionali. Tuttavia, come si può notare dai diagrammi a barre di Figura 6.2, i descrittori FPFH sembrano essere più robusti rispetto ai soli descrittori SURF, nonostante comportino un aumento del numero di falsi positivi. La motivazione di tale risultato può risiedere nel fatto che i descrittori 2D codificano l'informazione RGB, la quale può essere eccessivamente modificata durante il processing del dataset per la correzione della posa (presenza di zone nere anche dopo il filtraggio).
Descrittori più robusti si ottengono concatenando i descrittori SURF e FPFH portando così ad un utilizzo di entrambi i tipi di informazione 2D e 3D (diagrammi rossi). Il rank1 ottenuto con la concatenazione di questi due descrittori locali, in presenza di soggetti con posa non frontale, cala leggermente ma al tempo stesso anche il numero di falsi positivi diminuisce fino ad azzerarsi. Infine si nota come i risultati siano molto scarsi utilizzando unicamente le distanze euclidee come descrittori del volto (per questo motivo non è stata considerata la concatenazione dei descrittori SURF, FPFH con le distanze). Tali distanze sono calcolate per ogni coppia di *keypoint* e potrebbero essere non robuste a fronte di espressioni facciali differenti. Basti pensare che, come noto nel campo delle applicazioni biometriche, le distanze occhio-occhio e occhio-naso sono invarianti ma la distanza tra gli occhi e gli angoli della bocca cambiano in base alle espressioni. Tale motivo potrebbe essere la causa della diminuzione delle prestazioni dell'algoritmo.
In futuro si potrebbe pensare di determinare distanze geodesiche calcolate sulla superfice 3D per ottenere descrittori maggiormente identificativi per un problema di *face recognition* migliorando le performance qualitative del sistema. I classificatori usati in questo test sono il classificatore SVM allenato *one against one* (Figura 6.2a) e il classificatore SVM allenato *one against all* (Figura 6.2b).



(a) Confronto tra descrittori classificati con SVM *one against one*.



(b) Confronto tra descrittori classificati con SVM *one against all*.

Figura 6.2: Confronto tra i descrittori estratti per il *feature matching*.

Il terzo test è atto a valutare l'efficacia della tecnica di *mirroring* durante l'elaborazione del dataset di facce.
Come precedentemente detto, tale tecnica permette di localizzare in modo più preciso i *keypoint* e calcolare descrittori più identificativi del volto. I miglioramenti attesi dare risultati migliori sia in fase di training (addestramento di classificatori più robusti) sia in fase di testing di soggetti originariamente con posa non frontale. Tuttavia, come si evince dal grafico in Figura 3.6 si ottiene un lieve miglioramento ma affiancato da un aumento di falsi positivi, cosa non gradita specialmente in applicazioni di controllo degli accessi.
Da ciò si deduce che la tecnica di simmetrizzazione dei volti è una idea valida (posizione dei *keypoint* migliori) ma deve essere perfezionata per essere più accurata. Infatti l'implementazione del *mirroring* si basa su una ipotesi fondamentale: i soggetti devono essere allineati verticalmente rispetto il frame della camera. Tuttavia tale ipotesi spesso non è soddisfatta in quanto, a causa di una non corretta stima originaria della posa del soggetto, esso non viene portato ad essere in posizione esattamente verticale. Questo fa in modo che l'asse di simmetria effettivo si discosti dall'asse verticale comportando un risultato non ottimale.

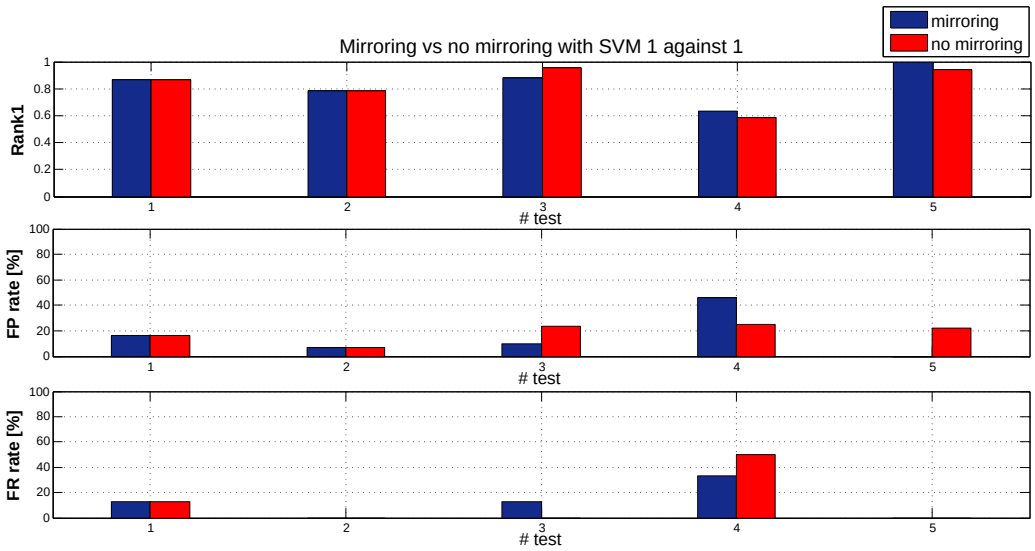


Figura 6.3: Confronto *mirroring* e *no mirroring* (SVM *one against one*).

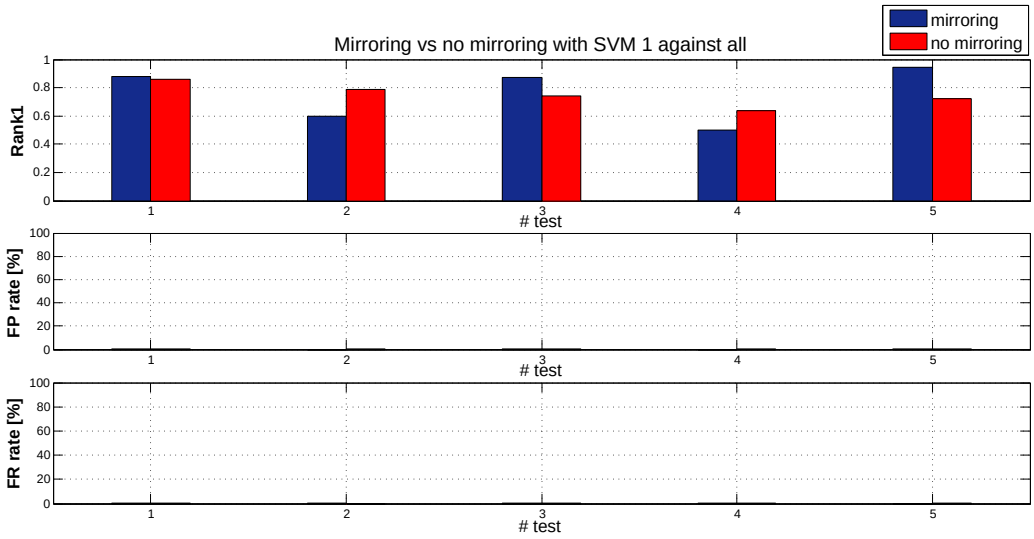


Figura 6.4: Confronto *mirroring* e *no mirroring* (SVM *one against all*).

L'ultimo test confronta i diversi classificatori usati e spiegati nel Capitolo 5 per classificare i descrittori ottenuti dalla concatenazione dei descrittori locali SURF e FPFH. Come si può notare dalla Figura 6.5, il classificatore *Nearest Neighbor* ottiene valori di rank-1 molto bassi. Questo risultato era prevedibile in quanto tale classificatore non gestisce in maniera robusta descrittori di grandi dimensioni, soprattutto ottenuti dalla concatenazione di diversi tipi di descrittori locali.
Per quanto riguarda la differenza tra i due classificatori SVM multiclasse si nota come, in caso di pose frontali a 1 metro dal sensore, con espressioni diverse e condizioni luminose sia buone sia scarse, i due classificatori ottengano valori di rank-1 simili ma quello allenato *one against all* non ottiene nessun falso positivo a differenza di quello *one against one*. Tuttavia quest'ultimo ottiene risultati migliori in presenza di soggetti a 2 metri dalla camera (rank-1 > 20 % rispetto *one against all*) e con pose non frontali (rank-1 > 10 %).

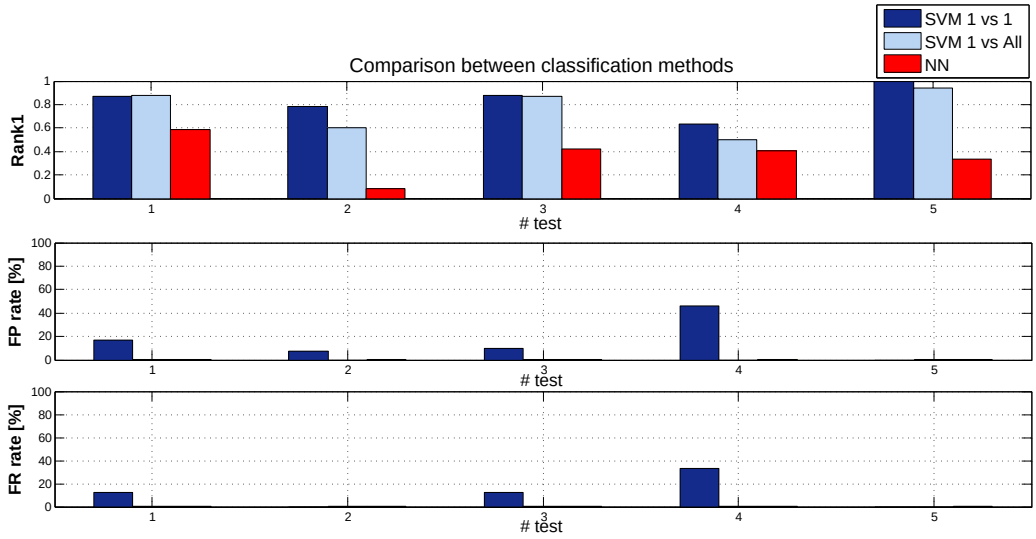


Figura 6.5: Confronto tra i classificatori utilizzati.

6.4 Prestazioni

I primi moduli del sistema di *face recognition* descritto in questa tesi, fino all'estrazione dei descrittori, sono stati sviluppati in linguaggio di programmazione `C++` sfruttando le risorse fornite dalle librerie *open source*: **OpenCV** e **PCL**. La fase di training e la fase di classificazione vera e propria sono state invece sviluppate in ambiente `Matlab`.

6.4.1 Librerie utilizzate

OpenCV (Open Source Compure Vision) [17] è una libreria di funzioni orientata allo sviluppo di applicazioni *real time* nell'ambito della computer vision. È cross-platform e viene rilasciata con licenza BSD, gratuita sia per un uso accademico che commerciale, per molte piattaforme tra cui Linux, Windows, MacOS, Android e iOS. La libreria comprende più di 2500 algoritmi e offre interfacce in `C++`, `C`, e `Python`. Questa libreria può essere utilizzata in diverse applicazioni tra cui: *Human-Computer Interaction* (HCI), *object identification*, *segmentation* e *recognition*, *face recognition*, *gesture recognition*, *motion tracking*, *ego motion*, *motion understanding*, *structure from motion* (SFM), *stereo* e *multi-camera calibration* e *depth computation*, *mobile robotics*.

PCL (Point Cloud Library) [18] è un progetto *open source* che contiene numerosi algoritmi allo stato dell'arte per l'elaborazione di *point cloud*, i quali includono *filtering*, *feature estimation*, *surface reconstruction*, *registration*, *model fitting* e *segmentation*. Questi algoritmi possono essere usati, per esempio, per filtrare valori anomali da dati rumorosi, effettuare lo stiching di point cloud 3D, segmentare parti rilevanti da una scena, individuare *keypoint* e calcolare descrittori per riconoscere oggetti nel mondo in base alla loro forma geometrica, creare superfici da *point cloud* e visualizzarle. **PCL** viene rilasciato secondo i termini della licenza BSD, gratuito sia per un utilizzo nell'ambito della ricerca che commerciale. È cross-platform ed attualmente è disponibile per sistemi Linux, Windows, MacOS e Android nella versione 1.7.
La libreria si compone di diversi moduli. In particolare i moduli utilizzati per lo sviluppo del sistema di *face recognition* sono:

- **features** contiene le strutture di dati e i meccanismi per stimare *feature* 3D a partire dalle *point cloud*. Le *feature* 3D descrivono pattern geometrici basati sulle informazioni disponibili attorno al punto. Lo spazio di dati selezionato attorno al punto considerato è solitamente conosciuto sotto il nome di k-neighborhood.

- **keypoints** contiene l'implementazione di diversi algoritmi per l'individuazione di *keypoint* nelle *point cloud*. Questi *keypoint* sono punti in una *point cloud* che sono stabili, distintivi e possono essere identificati usando dei criteri ben definiti. In genere, il numero di punti di interesse in una *point cloud* è molto inferiore rispetto al numero di punti totali, e quando usato in combinazione con descrittori di *feature* locali per ogni *keypoint*, i *keypoint* e i descrittori possono essere usati per formare una rappresentazione compatta ma distintiva dei dati originali.

- **registration** contiene molti algoritmi per la registrazione di *point cloud* organizzate e non organizzate. La registrazione tra due *point cloud* si occupa di trovare la trasformazione che minimizza la distanza (errore di allineamento) tra punti corrispondenti.

- **kdtree** fornisce la struttura dati kd-tree, che usa l'implementazione FLANN per effettuare velocemente ricerche nearest neighbor. Un Kd-tree (albero a k dimensioni) è una struttura di dati per partizionare lo spazio che memorizza un set di punti a k-dimensioni in una struttura ad albero che permette efficienti *range search* e *nearest neighbor search*. Può essere utilizzata per trovare corrispondenze tra gruppi di punti, descrittori di *features* o per definire un neighborhood locale attorno a un punto o ad un insieme di punti.

- **io** contiene classi e funzioni per la lettura e la scrittura di file di dati *point cloud* e per catturare *point cloud* da una varietà di sensori RGB-D compatibili, quali ad esempio la *Microsoft Kinect*.

- **visualization** questa libreria è stata creata con lo scopo di riuscire a prototipare velocemente e visualizzare i risultati degli algoritmi operanti su *point cloud*. Questo modulo permette di visualizzare *point cloud* e disegnare forme tridimensionali.

6.4.2 Tempi

I test effettutati sono stati svolti su un Hp Pavilion dv6 con specifiche principali: **OS**: Linux Ubuntu v12.04 32-bit, **RAM**: 4096MB, **Scheda video**: Radeon HD4650.
La Tabella 6.4 riporta i tempi di ogni singola operazione significativa. È importante sottolineare che, per velocizzare il calcolo dei descrittori FPFH (il quale inizialmente richiedeva 5 $[s]$ per ogni singolo esempio di test), è stata filtrata la *cloud* con un *Voxel Filter* con *Leaf Size* pari a 0.005.

Questi tempi acquisiti risultano fondamentali per capire se l'algoritmo implementato possa funzionare live. Sommando i tempi delle singole operazioni che vengono svolte sui dati RGB-D, si deduce che, per processare un singolo frame, la pipeline sviluppata impiega in media 1.5 secondi.

Operazione	**Tempi di elaborazione training [s]**
Stima posa	0.3
Correzione posa	0.12
Proiezione	0.2
Mirroring	0.001
Face e keypoints detection	0.3
Descrittori 2D	0.0011
Mappatura keypoints in 3D	0.04
Descrittori 3D	0.5

Tabella 6.4: Prestazioni dell'algoritmo di *face recognition*.

Capitolo 7

Conclusioni

L'obiettivo del progetto descritto in questa tesi era lo sviluppo di un algoritmo invariante alla posa per il riconoscimento facciale da dati RGB-D ottenuti con una *Microsoft Kinect* di 2° generazione.
Tale algoritmo è stato sviluppato utilizzando i due tipi di informazione a disposizione, 2D e 3D, che hanno permesso di affrontare i ricorrenti problemi di cui soffrono i metodi presenti in letteratura basati unicamente su dati bidimensionali: variazione di illuminazione, differenti espressioni facciali, pose non frontali. In particolare l'informazione 3D è risultata utile per affrontare il problema di soggetti con posa non frontale. L'algoritmo di stima e correzione della posa ha permesso di ruotare ogni soggetto in posizione frontale rispetto la camera. In questo modo ci si è posti nelle condizioni di lavoro migliori per l'algoritmo di *feature detection* basato su *Random Forest Classifiers*. Grazie a tale algoritmo sono stati localizzati dieci *keypoint* su occhi, naso e bocca sulle immagini RGB e, grazie alla corrispondenza tra dati RGB e dati RGB-D, questi punti sono stati facilmenti mappati sui dati 3D.
Avendo a disposizione *keypoint* sia 2D che 3D sono stati calcolati descrittori sia bidimensionali (SURF) che tridimensionali (FPFH e distanze euclidee). Infine, concatenando i descrittori parziali SURF e FPFH è stato ottenuto un unico descrittore per inglobare entrambi i tipi di informazione. Tali descrittori sono stati allenati e classificati tramite SVM (Support Vector Machines).
Per la valutazione dell'algoritmo implementato sono stati calcolati due importanti indici di riconoscimento: la percentuale di riconoscimenti corretti (*rank-1*) e la percentuale di volte in cui una persona sconosciuta viene classificata come una persona nota (*false positive rate*). È stato ritenuto interessante anche osservare il numero di volte in cui una persona nota viene classificata come sconosciuta. Come si evince dai grafici riportati in Sezione 6.3 l'algoritmo ottiene buoni risultati in condizioni di distanza dal sensore di 1 e 2 metri, di luce buona e scarsa e di posa frontale (rank-1 pari a 0.8/0.9)

e, grazie al metodo per la correzione della posa, le prestazioni iniziano a migliorare anche in presenza di dataset con pose miste (rank-1 pari a 0.6). In particolare, processare il dataset per ottenere solo pose frontali ha permesso un rilevamento di punti di interesse del volto più preciso e per un numero maggiore di soggetti effettivamente classificati.

L'algoritmo proposto ha però alcune limitazioni. I risultati non ottimali per quanto concerne i test su soggetti ripresi non frontalmente sono da imputare ad una non esatta stima della posa. Questo motivo è essenzialmente dovuto al fatto che tale stima risulta robusta in condizioni di distanza di 1 metro dal sensore. Il dataset acquisito, a differenza delle condizioni ideali descritte in Sezione 6.1, presenta soggetti anche a 1.5 metri implicando imprecisioni sul risultato.
Inoltre, il processing del dataset comporta l'inserimento di alcuni pixel di colore nero in seguito alla proiezione di *point cloud* non dense. I descrittori che codificano l'informazione RGB, estratti su tali dati proiettati, diventano meno robusti e meno identificativi del soggetto portando così ad una classificazione meno efficiente.

7.1 Sviluppi futuri

Per superare le limitazioni precedentemente esposte alcune soluzioni sono state pensate e potrebbero essere prese come idee di partenza per possibili lavori futuri:

- Sviluppare descrittori 3D più robusti: in [11] viene proposto il descrittore $LBPHmesh$, ovvero un'estensione del descrittore 2D LBPH al caso tridimensionale per caratterizzare una superficie 3D. L'algoritmo prevede la costruzione di una mesh di una *cloud*, ovvero una suddivisione di essa in tante sfaccettature. Per costruire questo tipo di descrittore vengono create delle sequenze di sfaccettature nel seguente modo: presa una sfaccettatura centrale, chiamata f_c, le sfaccettature attorno al contorno di f_c vengono suddivise in due categorie:

  - F_{out}: sfaccettature che hanno in comune un lato con f_c;
  - F_{gap}: sfaccettature che hanno in comune un vertice con f_c.

  L'algoritmo inizia trovando le sfaccettature F_{out} attorno a f_c e successivamente estrae le sfaccettature F_{gap} presenti tra ogni coppia di F_{out} consecutive. Al termine si ottiene un anello di sfaccettature in sequenza attorno a f_c come riportato in Figura 7.1.

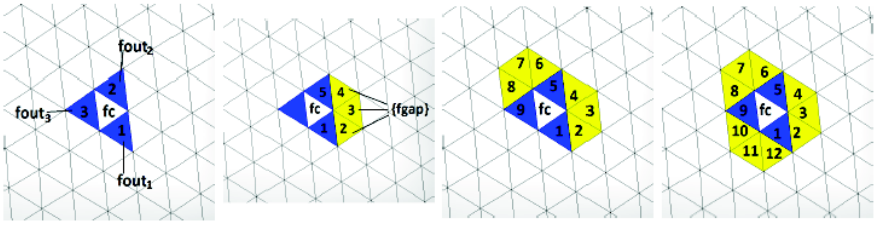


Figura 7.1

Su queste strutture ordinate di sfaccettature viene calcolato il descrittore noto come *LBPH mesh*. Presa una funzione scalare $h(f)$ basata sulla geometria (ad esempio la curvatura della superficie) o sul colore, l'operatore *LBPH mesh* è definito nel seguente modo:

$$LBPHmesh_m^r = \sum_{k=0}^{m-1} s(h(f_k^r) - h(f_c)) \cdot \alpha(k) \tag{7.1}$$

dove r è il numero di anelli, m il numero di sfaccettature distribuite su tali anelli e la funzione $\alpha(k)$ è stata inserita per ottenere possibili varianti. La funzione $s(x)$ invece è definita come:

$$s(x) = \begin{cases} 1 & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases}$$

- Rendere più robusto l'algoritmo di stima della posa. Infatti tale algoritmo funziona correttamente per persone distanti 1 metro dal sensore. Gli alberi di decisione infatti sono stati allenati per funzionare in queste condizioni [4]. Il dataset acquisito, tuttavia, presenta soggetti non sempre in posizione esattamente a 1 m dalla camera e ciò influenza la bontà della stima che si ottiene. Per risolvere questo problema si potrebbe pensare di allenare nuovamente gli alberi dell'algoritmo [4] sul dataset utilizzato.

- Migliorare la tecnica di simmetrizzazione del volto, cercando di gestire il caso di soggetti non perfettamente allineati verticalmente. Si potrebbe pensare di determinare l'asse di simmetria in base all'orientazione del volto. Ciò richiederebbe di fittare con una ellissi la faccia del soggetto e prendere l'asse maggiore come asse di simmetria.

Capitolo 8

Bibliografia

[1] M. Turk and A. Pentland, "Eigenfaces for recognition", in *Journal of Cognitive Neuroscience*, Mar. 1991, pp. 71–86.

[2] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, "Eigenfaces vs. Fisherfaces: Recognition using class specific linear projectio", in *IEEE Trans. Pattern Analysis and Machine Intelligence*, Jul. 1997, pp. 711–720.

[3] D. L. Swets and J. Weng, "Using discriminant eigenfeatures for image retrieval", in *IEEE Trans. Pattern Analysis and Machine Intelligence*,1996, pp. 831–836.

[4] G. Fanelli, J. Gall, and L. Van Gool, "Real time head pose estimation with random regression forests", in *Computer Vision and Pattern Recognition (CVPR)*, June 2011, pp. 617-624.

[5] G. Fanelli, M. Dantone, J. Gall, A. Fossati, and L. Van Gool, "Random Forests for Real Time 3D Face Analysi", *Int. J. Comput. Vision*, 2013, vol. 101, no. 3, pp. 437-458, February 2013.

[6] G. Fanelli, T. Weise, J. Gall, and L. Van Gool, "Real time head pose estimation from consumer depth cameras", in *33rd Annual Symposium of the German Association for Pattern Recognition (DAGM'11)*, September 2011.

[7] M. Dantone, J. Gall and G. Fanelli and L. Van Gool, "Real-time Facial Feature Detection using Conditional Regression Forests", in *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[8] R. B. Rusu, N. Blodow, and M. Beetz, "Fast Point Feature Histograms (FPFH) for 3D Registration", in *Robotics and Automation. IEEE International Conference*, May 2009.

[9] A.-A. Bhuiyan and C. H. Liu, "On face recognition using gabor filters", in *Proceedings of World Academy of Science, Engineering and Technology*, 2007, volume 22, pages 51–56.

[10] B. Kepenekci, "Face recognition using Gabor Wavelet Transformation", *PhD thesis, Te Middle East Technical University*, September 2001.

[11] N. Werghi, C. Tortorici, S. Berretti and A. Del Bimbo, "Representing 3D Texture on Mesh Manifolds for Retrieval and Recognition Applications ", in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[12] K. I. Chang, K. Bowyer and P. J. Flynn, "Face Recognition Using 2D and 3D Facial Data", in *Workshop in Multimodal User Authentication*, December 2003, pp. 25-32.

[13] Z. Cao, Q. Yin, X. Tang and J. Sun, "Face Recognition with Learning-based Descriptor", in *Computer Vision and Pattern Recognition (CVPR)*, 2012.

[14] D. R. Kisku, P. Gupta, J. K. Sing, "Face Recognition using SIFT Descriptor under Multiple Paradigms of Graph Similarity Constraints", in *International Journal of Multimedia and Ubiquitous Engineering Vol. 5, No. 4*, October 2010.

[15] D. Yi, Z. Lei and S. Z. Li, "Towards Pose Robust Face Recognition", in *Computer Vision and Pattern Recognition (CVPR)*, 2013.

[16] B. Chu, S.Romdhani and L.Chen, "3D-aided face recognition robust to expression and pose variations", in *Computer Vision and Pattern Recognition (CVPR)*, 2013.

[17] http://docs.opencv.org/

[18] http://pointclouds.org/

[19] R. Jafri and H. R. Arabnia, "A Survey of Face Recognition Techniques ", in *Journal of Information Processing Systems*, June 2009.

[20] A. F. Abate, M. Nappi, D. Riccio and G. Sabatino, "2D and 3D face recognition: A survey", in *Pattern Recognition Letters*, January 2007.

[21] L. Li,C. Xu b, W. Tang, C. Zhong, "3D face recognition by constructing deformation invariant image", in *Pattern Recognition Letters*, April 2008.

[22] N. Alyüz, B. Gökberk and L. Akarun, "Regional Registration for Expression Resistant 3-D Face Recognition", in *IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY*, September 2010.

[23] Y. Zhang, Z. Zhou, "Cost-Sensitive Face Recognition", in *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, October 2010.

[24] U. Park, Y. Tong and A. K. Jain, "Age-Invariant Face Recognition", in *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, May 2010.

[25] X. Zhao, X. Chai, Z. Niu, C. Heng and S. Shan, "Context modeling for facial landmark detection based on Non-Adjacent Rectangle (NAR) Haar-like feature", in *Image and Vision Computing*, 2012.